Efficient Learning Algorithms Yield Circuit Lower Bounds

Lance Fortnow and Adam Klivans

U. Chicago Comp. Sci., 1100 E. 58th St., Chicago, IL 60637,
UT-Austin Comp. Sci., 1 University Station C0500, Austin, TX 78712,
fortnow@cs.uchicago.edu,
klivans@cs.utexas.edu

Abstract. We describe a new approach for understanding the difficulty of designing efficient learning algorithms. We prove that the existence of an efficient learning algorithm for a circuit class C in Angluin's model of exact learning from membership and equivalence queries or in Valiant's PAC model yields a lower bound against C. More specifically, we prove that any subexponential time, determinstic exact learning algorithm for C (from membership and equivalence queries) implies the existence of a function f in $\mathsf{EXP}^{\mathsf{NP}}$ such that $f \notin C$. If C is PAC learnable with membership queries under the uniform distribution or Exact learnable in randomized polynomial time, we prove that there exists a function $f \in \mathsf{BPEXP}$ (the exponential time analog of BPP) such that $f \notin C$.

For C equal to polynomial-size, depth-two threshold circuits (i.e., neural networks with a polynomial number of hidden nodes), our result shows that efficient learning algorithms for this class would solve one of the most challenging open problems in computational complexity theory: proving the existence of a function in $\mathsf{EXP}^{\mathsf{NP}}$ or BPEXP that cannot be computed by circuits from C. We are not aware of any representation-independent hardness results for learning polynomial-size neural networks.

Our approach uses the framework of the breakthrough result due to Kabanets and Impagliazzo showing that derandomizing BPP yields non-trivial circuit lower bounds.

1 Introduction

Discovering the limits of efficient learnability remains an important challenge in computational learning theory. Traditionally, computational learning theorists have reduced problems from computational complexity theory or cryptography to learning problems in order to better understand the difficulty of various classification tasks.

There are two lines of research in learning theory in this direction. First, several researchers have shown that *properly* PAC learning well-known concept classes (i.e. learning with the requirement that the output hypothesis be of the same form as the concept being learned) such as DNF formulas, finite automata, or intersections of halfspaces is NP-hard with respect to randomized reductions [1; 2; 3].

Secondly, in a seminal paper, Kearns and Valiant [4] initiated a line of research that applied results from cryptography to learning. They proved that learning polynomial-size circuits, regardless of the representation of the hypothesis of the learner, would imply the existence of algorithms for breaking cryptographic primitives. Various researchers have extended this work to smaller subclasses of circuits [5; 6].

For some classes of circuits, however, we do not know how to create the cryptographic primitives needed to apply the Kearns-Valiant approach. It is a difficult open question, for example, as to whether polynomial-size depth-2 threshold circuits (polynomial-size neural networks) can compute cryptographic primitives. As such, we are unaware of any representation-independent hardness results for learning this class of circuits.

1.1 Reducing Circuit Lower Bounds to Learning Concept Classes

We give a new approach for showing the difficulty of proving that certain circuits classes admit efficient learning algorithms. We show that if a class of circuits C is efficiently learnable in either Angluin's exact model or Valiant's PAC model of learning, then we can prove a circuit lower bound against C. Hence, the existence of efficient learning algorithms (for many choices of C) would settle some important and well-studied open questions in circuit complexity.

Our first theorem states that a deterministic subexponential time exact learning algorithm for a concept class C implies a lower bound for Boolean circuits for a somewhat large complexity class:

Theorem 1. *Let C be a family of non-uniform, polynomial-size circuits. Assume that C is exactly learnable from membership and equivalence queries in time $2^{n^{o(1)}}$. Then there exists a function $f \in \mathsf{EXP}^{\mathsf{NP}}$ such that $f \notin C$.*

If we take C to be the class of polynomial-size neural networks we obtain the following corollary:

Corollary 1. *If there exists an algorithm for exactly learning neural networks (with a polynomial number of hidden nodes) in time $2^{n^{o(1)}}$ then there exists a function $f \in \mathsf{EXP}^{\mathsf{NP}}$ such that f is not computable by any polynomial-size, depth-2 threshold circuit.*

Finding a function in a uniform class such as $\mathsf{EXP}^{\mathsf{NP}}$ that cannot be computed by polynomial-size, depth-two threshold circuits has been a challenging open problem for over two decades in computational complexity. Additionally, we do not know of any representation independent hardness results for learning polynomial-size neural networks.

If we assume that our exact learning algorithm runs in polynomial-time rather than subexponential time we can show that even randomized exact learning algorithms imply circuit lower bounds against BPEXP, the exponential time version of BPP:

Theorem 2. *Let C be a family of non-uniform, polynomial-size circuits. Assume there exists a randomized polynomial-time algorithm for exactly learning C from membership and equivalence queries. Then there exists a function $f \in \mathsf{BPEXP}$ such that $f \notin C$.*

We are unaware of any lower bounds for circuit classes such as polynomial-size depth-2 threshold circuits against BPEXP.

Theorems 1 and 2 also directly applies to arithmetic circuits. If, in addition, we restrict the output hypothesis of the learner to be an arithmetic circuit or formula (of possibly larger size and depth) we can replace the NP oracle in Theorem 1 with an RP oracle and obtain a finer separation of uniform and non-uniform circuit classes.

Theorem 3. *Let C be a family of non-uniform, polynomial-size arithmetic formulas. Assume that C is exactly learnable from membership and equivalence queries in polynomial-time and the hypothesis of the learner is an arithmetic formula. Then there exists a function $f \in \mathsf{EXP}^{\mathsf{RP}}$ such that $f \notin C$.*

If we allow both C and the hypothesis to be arithmetic circuits then there exists an $f \in \mathsf{ZPEXP}^{\mathsf{RP}}$ such that $f \notin C$.

We note here that proving lower bounds against polynomial-size arithmetic formulas and even depth-3 arithmetic circuits remains one of the most difficult challenges in algebraic complexity. Furthermore, as with polynomial-size neural networks, we are not aware of any representation independent hardness results for learning restricted models of arithmetic circuits.

These results also apply to the PAC model. Due to the inherent role of randomness in the definition of PAC learning, our lower bounds apply to the complexity class BPEXP:

Theorem 4. *Let C be a family of non-uniform, polynomial-size circuits. Assume that C is PAC learnable in polynomial time. Then there exists a function $f \in \mathsf{BPEXP}$ such that $f \notin C$.*

The smallest uniform complexity class known to contain languages with super-polynomial Boolean circuit complexity is $\mathsf{MA}_{\mathsf{EXP}}$, the exponential-time analog of Merlin-Arthur proofs (see Setion 2.1 for a discussion). BPEXP is easily seen to be contained in $\mathsf{MA}_{\mathsf{EXP}}$.

1.2 Our Approach.

The proof of Theorem 1 follows the outline of the work of Kabanets and Impagliazzo [7] on derandomizing algebraic circuits: we assume that $\mathsf{EXP}^{\mathsf{NP}}$ is computable by some non-uniform circuit class C (otherwise there is nothing to prove). This implies, via a sequence of well-known reductions in complexity theory, that the Permanent is complete for $\mathsf{EXP}^{\mathsf{NP}}$ (the analogous statements with $\mathsf{EXP}^{\mathsf{NP}}$ replaced by BPEXP are not known to be true). At this point we need to use the supposed exact learning algorithm to construct an algorithm for computing the Permanent which runs in subexponential time and has access to an NP oracle. This leads to an immediate contradiction via time hierarchy theorems.

In the work of Kabanets and Impagliazzo, the assumption of a deterministic algorithm for polynomial-identity testing is used to develop a non-deterministic algorithm for computing the Permanent. In this work, we need to use the exact learning algorithm to construct the circuit $c \in C$ that computes the permanent. The main difficulty is that the exact learning algorithm needs access to a membership and equivalence query oracle, which we do not have. Using an idea from Impagliazzo and Wigderson's result on derandomizing BPP, we can simulate membership queries by inductively constructing circuits for the permanent on shorter input lengths. Simulating equivalence queries is slightly trickier and requires access to an NP oracle to find counterexamples.

To reduce the dependence on the NP oracle we can use randomness, but only in cases regarding arithmetic circuits and formulas, where output hypotheses can be suitably interpreted as low-degree polynomials. Our results on PAC learning and randomized Exact learners require a slightly different approach, as we are not aware of collapse consequences for the class BPEXP even if it is contained in $\mathsf{P/poly}$. We appeal to work on derandomization due to Impagliazzo and Wigderson [8] that makes use of the random-self-reduciblity of the Permanent.

1.3 Outline.

In Section 2 we define various learning models and state all the theorems from complexity theory necessary for proving our main result. In Section 3 we give a proof of our main result for exact learning in the Boolean case. Our results regarding learnability in the PAC model are in Section 4. In Section 5 we discuss applications to exact learning in the algebraic setting.

2 Preliminaries

Valiant's PAC model [9] and Angluin's model of Exact Learning from Membership and Equivalence queries [10] are two of the most well-studied learning models in computational learning theory. Recall that in Valiant's PAC model we fix a concept class C and a distribution D and a learner receives pairs of the form $(x, c(x))$ where x is chosen from D and c is some fixed concept in c. The learner's goal is to output, with probability $1-\delta$, a hypothesis h such that h is a $1-\epsilon$ accurate hypothesis with respect to c under D. We say that the learner is efficient if it requires at most t examples, runs in time at most t and outputs a hypothesis that can be evaluated in time t where $t = \mathsf{poly}(n, 1/\epsilon, 1/\delta, |c|)$ ($|c|$ denotes the size of the unknown concept). If the learner is allowed *membership queries* then it may query the unknown concept c at any point x of his choosing.

In Angluin's model of exact learning, the learner is trying to learn an unknown concept $c : \{0,1\}^n \rightarrow \{0,1\}$ and is allowed to make queries of the following form:

1. (*Membership Query*) What is the value of $c(x)$?
2. (*Equivalence Query*) Is h (the learner's current hypothesis) equal to c?

If the equivalence query is answered affirmatively, the learner outputs h and halts. Otherwise, the learner receives a counterexample, namely a point z such that $h(z) \neq c(z)$. We say that an algorithm A exactly learns a concept class C in time t if for every $c \in C$, A always halts and the time taken by A (including calls to the membership and equivalence query oracles) is bounded by t.

2.1 Uniform versus Non-uniform models of computation

We frequently mention standard complexity classes $\mathsf{EXP}, \mathsf{RP}, \mathsf{BPP}, \mathsf{NP}, \mathsf{MA}, \mathsf{ZPP}, \mathsf{EE}, \mathsf{ZPEXP}$ as well as relativized versions of the classes (e.g., $\mathsf{EXP}^{\mathsf{NP}}$). We refer the reader to the Complexity Zoo (`http://www.complexityzoo.com`) for further details on these classes .

We say a language L has polynomial-size circuits (P/poly) if there is a polynomial p and a sequence of logical (AND-OR-NOT) circuits $C_0, C_1, \ldots$ such that for all n,

1. The size of C_n is bounded by $p(n)$.
2. For all strings $x = x_1 \ldots x_n$, x is in L iff $C(x_1, \ldots, x_n) = 1$ where we use 1 for true and 0 for false.

Importantly, circuits describe nonuniform computation: the circuits for one input length may have no relation to the circuits for other lengths.

An algebraic circuit can only have addition, subtraction and multiplication gates, in particular no bit operations. All languages computed by algebraic circuits can be computed by a Boolean circuit (with a polynomial increase in size), but the converse is not known. A formula is a circuit described by a tree. It is well known that an arithmetic circuit is equivalent to a multivariate polynomial of degree at most exponential in the size of the circuit. Arithmetic formulas compute polynomials whose degree is at most a polynomial in the size of the formula.

The smallest complexity class known not to contain polynomial-size circuits is $\mathsf{MA_{EXP}}$ ([11], see also [12]), Merlin-Arthur games with an exponential-time verifier. Kabanets and Impagliazzo [7] use derandomization to show $\mathsf{NEXP^{RP}}$ does not have polynomial-size algebraic circuits. The relationship of $\mathsf{NEXP^{RP}}$ and $\mathsf{MA_{EXP}}$ is unknown.

It is a difficult open problem to improve upon $\mathsf{MA_{EXP}}$ as the smallest uniform class containing circuits of superpolynomial size even if we restrict ourselves to polynomial-size formulas, depth-2 threshold circuits (neural nets), or constant-depth logical circuits with Mod_m gates for any m not a prime power.

2.2 Hierarchy theorems

Theorem 5. $\mathsf{EXP^{NP}}$ *is not contained in* $\mathsf{SUBEXP^{NP}}$*, where* $\mathsf{SUBEXP} = \mathsf{DTIME}(2^{n^{o(1)}})$.

Proof: The seminal paper in computational complexity by Hartmanis and Stearns [13] shows that for any time-constructible functions $t_1(n)$ and $t_2(n)$ with $t_1^2(n) = o(t_2(n))$

$$\mathsf{DTIME}(t_1(n)) \subsetneq \mathsf{DTIME}(t_2(n))$$

and their proof relativizes. Theorem 5 follows by taking $t_1(n) = 2^n$, $t_2(n) = 2^{3n}$ and relativizing to SAT. $\square$

Let EE denote the class of languages computable in doubly-exponential time.

Theorem 6. EE *contains languages with super-polynomial circuit complexity.*

Proof: We follow a proof idea of Kannan [14]. We know by counting arguments there is some function that is not computed by circuits of size $2^{\sqrt{n}}$. In double exponential time we can, by brute force searching, find and evaluate the lexicographically least such function. $\square$

2.3 Properties of the Permanent

The Permanent of an $n \times n$ matrix A is defined by

$$\mathrm{P}erm(A) = \sum_{\sigma \in S_n} a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{n\sigma(n)}$$

Valiant [15] showed that the Permanent is complete for the class #P, i.e., complete for functions counting the number of solutions of NP problems. The Permanent remains #P-complete if we compute the Permanent over a sufficiently large finite field. We will frequently abuse notation and write "Permanent is complete for class C," even though the Permanent (for large fields) is not a Boolean function. This problem is explicitly resolved in Kabanets and Impagliazzo [7].

Toda [16] shows that the polynomial-time hierarchy reduces to #P and thus the Permanent.

We will use the following two well known facts about the Permanent:

Fact 1 *(Downward Self Reducibility of the Permanent) Computing the permanent of an $n \times n$ matrix is polynomial-time (Turing) reducible to computing n instances of the Permanent over $n-1 \times n-1$ matrices.*

Fact 1 follows easily from the cofactor expansion of the Permanent.

The Permanent, when defined over a finite field, is also random-self reducible [17], i.e., there is an efficient randomized procedure that will take an $n \times n$ matrix A and produce $n \times n$ matrices $A_1, \ldots, A_{n+1}$ and a polynomial-time function that takes the Permanents of the A_i's and compute the Permanent of A. Each A_i is uniformly random over the space of all $n \times n$ matrices, though A_i and A_j are not independent variables.
We state this as follows:

Theorem 7. *(Random-self-reducibility of the Permanent) [18; 17] Assume that we have a circuit c that computes the Permanent on all but a $1/n^2$ fraction of inputs (n is the length of the instance) with respect to any field $F, |F| \geq n^2$. Then there exists a randomized, polynomial-time algorithm A that uses c as an oracle such that for every input x, A computes Permanent on x correctly with probabilty at least $1 - 1/n$.*

Finally, we make use of a lemma due to Kabanets and Impagliazzo:

Lemma 1. *[7] Given an arithmetic circuit C, the problem of determining if C computes the Permanent on all inputs is in* coRP.

2.4 Collapse Theorems

A long line of research in complexity theory is devoted to understanding the consequences of large uniform complexity classes being contained in small, non-uniform ones. Perhaps the most well known collapse is due to Karp and Lipton, stating that if $\mathsf{NP} \subseteq \mathsf{P/poly}$ then the polynomial-time hierarchy collapses ($\mathsf{PH} = \Sigma_2$). We will need the following two "collapse" theorems:

Theorem 8. *[19] If* $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{EXP}/\mathsf{poly}$ *then* $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{EXP}$.

Theorem 9. *[20] If* $\mathsf{EXP} \subseteq \mathsf{P}/\mathsf{poly}$ *then* $\mathsf{EXP} = \mathsf{MA}$.

Since $\mathsf{MA} \subseteq \mathsf{PH}$ and $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$ [16] we conclude that $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{P}/\mathsf{poly}$ implies $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{P}^{\#\mathsf{P}}$. Applying Valiant's result on the complexity of the Permanent [15], we also have that $\mathsf{EXP}^{\mathsf{NP}} \subseteq \mathsf{P}/\mathsf{poly}$ implies Permanent is complete for $\mathsf{EXP}^{\mathsf{NP}}$.

3 Lower Bounds from Exact Learning Algorithms

In this section we prove our main result: algorithms for exactly learning circuits classes yield lower bounds against those same circuit classes.

Theorem 10. *Let C be a non-uniform class of polynomial-size circuits. Assume that C is exactly learnable from membership and equivalence in time $t = 2^{n^{o(1)}}$. Then $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq C$.*

Proof: First assume that $\mathsf{EXP}^{\mathsf{NP}} \subseteq C$ (since otherwise there is nothing to prove) and notice that since C is a class of polynomial-size circuits, Theorem 8 implies that $\mathsf{EXP}^{\mathsf{NP}} = \mathsf{EXP} = \mathsf{P}^{\#\mathsf{P}}$. As such, the Permanent function is now complete for $\mathsf{EXP}^{\mathsf{NP}}$ and is computable by some $c \in C$. We wish to give a $\mathsf{poly}(n,t)$ time algorithm for computing the permanent that is allowed to make calls to an NP oracle. Because $\mathsf{EXP}^{\mathsf{NP}}$ can be reduced to Permanent in polynomial-time, such an algorithm would violate a relativized version the time hierarchy theorem (Theorem 5) and complete the proof. Consider the following algorithm for computing the permanent:

Algorithm for Computing Permanent on input x:

For $i = 2$ to $n = |x|$:

1. Run the exact learning algorithm to find c_i, the circuit that computes permanent on inputs of length i.
2. Simulate required membership queries and equivalence queries using c_{i-1} and the NP oracle.

Output $c_n(x)$.

Computing c_1, the "base case," is trivial. The main difficulty is step 2, simulating the required membership and equivalence query oracles. If we can simulate membership and equivalence query oracles using an NP oracle, then the above steps for computing the permanent can be carried out in time $\mathsf{poly}(n,t)$ with an NP oracle, as desired.

Simulating Membership Queries:

Assume that our learning algorithm makes a membership query and requires the value of the permanent on input y of length i. By induction assume we have constructed a circuit c_{i-1} for computing the permanent on inputs of length $i-1$. Then, by applying Fact 1, the permanent on input y can be computed via i calls to the circuit c'. As we have assumed that c' is exactly correct for all inputs of length $i-1$, we are certain that we will obtain the correct value for the permanent on y (this is the same technique used by Impagliazzo and Wigderson [8] for making certain derandomization reductions more "uniform").

Simulating Equivalence Queries:

Assume that we wish to determine if our current hypothesis h computes permanent correctly on all inputs of length i. We make the following query to the NP oracle: "Does there exist an input z such that $h(z)$ does not equal the value obtained by using the downward self-reducible property of the permanent and circuit c_{i-1} (by Fact 1, we can compute the true value of the permanent via i calls to circuit c_{i-1})?" I.e. does there exist an input z where the self-reduction fails? If there is no such input z then we are guaranteed that our hypothesis h is exactly correct on all inputs of length n. Otherwise, we can use the NP oracle to reconstruct the input z where h is incorrect. We first make a query of the form "Does there exist an input z where the self-reduction fails that begins with a 0?" If the answer is "no" then we know there must exist a counterexample that begins with a 1. Our next query is "Does there exist an input z beginning with 10 such that..." and so on for the remaining bits until we have obtained a counterexample.

Since we can construct a circuit for exactly computing the permanent on inputs of length n using an NP oracle and a circuit for the computing the permanent on inputs of length $n-1$, we can find the circuit c that computes the permanent as required in step 1 of the above algorithm. Since c is from a polynomial-size circuit class we can evaluate $c(x)$ in polynomial time. This results in a time t, NP-oracle algorithm for computing the permanent and completes the proof. $\square$

We can make a similar statement regarding learnability in the mistake bounded model. Recall that in the mistake bounded model, the learner is required to make at most $poly(n, s)$ mistakes on any sequence of examples, where n is the length of the longest example and s is the size of the unknown concept to be learned. Assuming that the permanent function is computed by some circuit c, we can use the NP oracle as above to determine if our current hypothesis is exactly correct. If not, we can generate a counterexample in exactly the same manner as in the proof of Theorem 1. We have the following theorem:

Theorem 11. *Let C be a non-uniform class of polynomial-size circuits. Assume that C is efficiently learnable in the mistake bounded model. Then $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq C$.*

Note again that Theorem 1 does not place any restrictions on the output hypothesis of the learner, or of the concept to be class to be learned, other than the requirement that evaluating $c(x)$ can be carried out in time $t = 2^{n^{o(1)}}$. Additionally, it is important to point out that the smallest known complexity class containing functions of superpolynomial-size circuit complexity is $\mathsf{MA}_{\mathsf{EXP}}$, the exponential-time analogue of MA [11]. Proving $\mathsf{MA}_{\mathsf{EXP}} \subseteq \mathsf{EXP}^{\mathsf{NP}}$ would be a tremendous derandomization.

4 Lower Bounds from Randomized Learning Algorithms

In this section we show that efficient PAC algorithms for learning a circuit class C or polynomial-time randomized Exact learning algorithms imply the existence of a function $f \in \mathsf{BPEXP}$ such that f is not in C. Our result holds even if the PAC algorithm is allowed to make membership queries to the unknown concept. As with the complexity class $\mathsf{EXP}^{\mathsf{NP}}$, it is a difficult open problem as to whether BPEXP can be computed by circuit classes such as polynomial-size, depth-2 threshold circuits, or polynomial-size arithmetic formulas. The smallest complexity class known to strictly contain these circuit classes is $\mathsf{MA}_{\mathsf{EXP}}$. It is widely believed that BPEXP is strictly less powerful than $\mathsf{MA}_{\mathsf{EXP}}$.

We require the following lemma, which states that if the Permanent has polynomial-size circuits from some class C and C is PAC learnable, then the Permanent is computable in BPP. This lemma is implicit in the work of Impagliazzo and Wigderson [8] (although it was used there to obtain new results in derandomization). We provide a proof in the language of PAC learning:

Lemma 2. *([8], restated) Assume that the Permanent is computed by a non-uniform class of polynomial-size circuits C. If C is PAC learnable with respect to the uniform distribution (with membership queries) then the Permanent is in* BPP.

Proof: To compute Permanent on input x of length n, assume by induction we have a randomized circuit c_{n-1} such that for every x, c_{n-1} computes the permanent correctly on x with probabilty at least $2/3$. Since C is PAC learnable, consider its associated learning algorithm A that learns any $c \in C$ in time $\mathsf{poly}(n, 1/\epsilon, 1/\delta, |c|)$. Set $\epsilon = 1/n^2$ and $\delta = 1/3n$. Let t equal the number of labeled examples and membership queries required by A for this setting of parameters.

For any input x of length $n-1$, we can amplify the probability that c_{n-1} computes x correctly to $1 - 1/3n^2t$ by taking a majoriy vote of multiple invocations of c_{n-1} on x (note that $t = n^{O(1)}$).

Now randomly choose the t points $z_1, \ldots, z_t$ (some of which may be membership queries) required by the learning algorithm. To find the labels of these points (recall the label of z_i is Permanent(z_i)), we apply Fact 1 and query c_{n-1} at the appropriate tn points. Applying a union bound we see that the probability some point is mislabeled is less than $1/3n$. Hence with probability at least $1-\delta-1/3n$ we obtain a hypothesis h that computes Permanent correctly on all but a $1/n^2$ fraction of inputs. Applying Theorem 7 (the random self reducibility of the permanent), we obtain a randomized circuit c_n such that c_n computes Permanent correctly on each input with probability at least $1-1/n$.

Applying the union bound over all n iterations of this process, we see that the probability we fail to construct c_n properly is at most $n\delta + 1/3$. Since $\delta < 1/3n$, with probability at least $2/3$ we have obtained a randomized circuit for computing the permanent that is correct on every input with probability at least $1-1/n$. The lemma follows. $\square$

We can now state our main theorem showing PAC learning algorithms imply lower bounds against BPEXP. Since we do not know if $\mathsf{BPEXP} \subseteq \mathsf{P/poly}$ implies $\mathsf{BPEXP} = \mathsf{EXP}$, we must use the fact that doubly exponential time contains languages with superpolynomial circuit complexity:

Theorem 12. *Let C be a non-uniform class of polynomial-size circuits. Assume that C is PAC learnable (with membership queries) with respect to the uniform distribution in polynomial-time. Then* $\mathsf{BPEXP} \not\subseteq C$.

Proof: First assume that $\mathsf{EXP} \subseteq C$ as otherwise we have nothing to prove. Then applying Theorem 9 we have $\mathsf{EXP} = \mathsf{PSPACE} = \mathsf{P}^{\#\mathsf{P}}$. Thus Permanent is complete for EXP, and any EXP complete language L can be reduced to Permanent in polynomial-time. Applying Lemma 2 we have that Permanent is in BPP and thus $\mathsf{EXP} = \mathsf{BPP}$. This implies that $\mathsf{EE} \subseteq \mathsf{BPEXP}$. From Theorem 6, we know that EE contains a language not computable by C. Hence $\mathsf{BPEXP} \not\subseteq C$. $\square$

To extend these results to randomized Exact learning algorithms we require the following lemma:

Lemma 3. *Assume that the Permanent is computed by a non-uniform class of polynomial-size circuits C. If C is Exactly learnable from membership and equivalence queries in randomized polynomial-time then the Permanent is in* BPP.

Proof: Assume by induction we have a circuit c_{n-1} that computes Permanent correctly on at least a $1-1/n^2$ fraction of inputs. We say a circuit h on n inputs is *good* if it computes Permanent on at least a $1-1/n^3$ fraction of inputs of length n. In order to output, with high probability, a circuit c_n that is correct on at least a $1-1/n^2$ fraction of inputs, we run the exact learning algorithm and test whether the current hypothesis h is good. We can test, with high probability, if h is good by choosing sufficiently many random inputs and seeing if the answers obtained by c_{n-1} (applying both the downward self-reducible and random-self-reducible properties of the Permanent) agree with h. If h agrees with c_{n-1} on these random points then with high probabiltiy h is good so we set $h = c_n$. Otherwise, with high probability, we will have found a point where h differs Permanent, i.e. a counterexample, and we continue running the learning algorithm. We conclude that with high probability, in polynomial-time, we will have a circuit c_n computing Permanent on a $1-1/n^2$ fraction of inputs of length n. Applying Theorem 7 completes the proof. $\square$

Applying the same proof for Theorem 12 but using Lemma 3 we obtain

Theorem 13. *Let C be a non-uniform class of polynomial-size circuits. Assume that C is Exactly learnable in randomized polynomial-time. Then* $\mathsf{BPEXP} \not\subseteq C$.

Buhrman et al. [11] proved that $\mathsf{MA_{EXP}}$ contains languages with superpolynomial size circuits; this is still the smallest class known to contain languages of superpolynomial circuit complexity. Theorem 12 shows that PAC learnability of a circuit class such as depth-2 threshold circuits, even under the uniform distribution with membership queries, would implies a new lower bound. To contrast this with the work of Kabanets and Impagliazzo [7], they showed that under the assumption that there exists a non-deterministic subexponential time algorithm for polynomial identity-testing, NEXP contains languages with superpolynomial *arithmetic* circuit complexity.

5 Improved Lower Bounds from Learning Arithmetic Circuits

Several researchers have given deterministic, exact learning algorithms for various classes of algebraic models of computation including read-once arithmetic formulas, algebraic branching programs, and arithmetic circuits. Our main theorem applies to these models of computation as well. In fact, if we restrict the output hypothesis to be a polynomial-size arithmetic circuit or formula (or any hypothesis equal to a multivariate polynomial of degree bounded by $2^{n^{O(1)}}$), then we obtain a finer set of separations. We note that many exact learning algorithms for algebraic concepts do indeed output a hypothesis equal to polynomials of bounded degree (for example Bshouty et al. [21] or Klivans and Shpilka [22]). We require the following lemma:

Lemma 4. *Assume that polynomial-size arithmetic circuits are exactly learnable in polynomial-time and the output hypothesis is an arithmetic circuit. Then if Permanent is computed by polynomial-size arithmetic circuits, Permanent is in* $\mathsf{ZPP^{RP}}$.

Proof: We iteratively construct circuits $c_1, \ldots, c_n$ such that c_i computes the permanent on inputs of length i. At stage i, given c_{i-1}, membership queries are simulated as in the proof of Theorem 1. In order to find a counterexample, however, we cannot use an NP oracle. Instead, we use the fact that our output hypothesis is a polynomial-size arithmetic circuit. Lemma 1 shows that the problem of determining whether an arithmetic circuit computes permanent exactly is computable in polynomial-time given access to an RP oracle. If we discover that our hypothesis is correct we stop. Otherwise, we know that our hypothesis is not equal to the permanent.

At this point we need to compute a counterexample, namely a point z such that our current candidate for $h(z)$ does not equal Permanent of z where h is our current candidate for c_i. Since h is a polynomial-size arithmetic circuit it is equal to a polynomial of degree at most $2^{O(n^a)}$ for some fixed constant a (see Section 2.1). Thus a random z will be a counterexample if z is chosen from a field F of size $2^{O(n^{2a})}$. Thus, our algorithm chooses a random $z \in F$ and checks if $h(z)$ does not equal Permanent on z (the label for z can be computed by applying Lemma 1 and using c_{n-1}). With high probability we will obtain such a z. Due to the correctness of the learning algorithm, we will be assured of a correct hypothesis after at most $n^{O(1)}$ counterexamples. At each stage i the probability of failure can be amplified to be less than $1/3n$ so that the overall probability of failure (cumulative over all n stages) will be less than $1/3$. $\square$

We can now show that learning arithmetic circuits (by arithmetic circuits) yields a lower bound against $\mathsf{ZPEXP}^{\mathsf{RP}}$. Since we know of no collapse theorems for complexity classes such as $\mathsf{ZPEXP}^{\mathsf{RP}}$ (or even $\mathsf{EXP}^{\mathsf{RP}}$) we need to use a different argument than in the proof of Theorem 1:

Theorem 14. *Let C be a non-uniform class of polynomial-size arithmetic circuits. Assume that C is exactly learnable from membership and equivalence queries in time $\mathsf{poly}(n)$ and that the output hypothesis is an arithmetic circuit. Then $\mathsf{ZPEXP}^{\mathsf{RP}} \not\subseteq C$.*

Proof: We may assume that 1) the Permanent is computable by circuits from C and 2) $\mathsf{EXP} \subseteq C$, as otherwise there is nothing to prove. Notice that if $\mathsf{EXP} \subseteq C$ then $\mathsf{EXP} = \mathsf{P}^{\#\mathsf{P}}$ by Theorem 9 and Permanent is complete for EXP via a polynomial-time reduction. By Lemma 4, Permanent (and thus EXP) is in $\mathsf{ZPP}^{\mathsf{RP}}$. This implies that $\mathsf{EE} \subseteq \mathsf{ZPEXP}^{\mathsf{RP}}$, but by Theorem 6, EE contains functions with superpolynomial circuit complexity. Hence $\mathsf{ZPEXP}^{\mathsf{RP}}$ must also. $\square$

It is still an open problem is to whether polynomial-size arithmetic formulas are exactly learnable in polynmial-time; much progress has been made on restricted versions of this problem (for example [23; 24]). For the case of exactly learning arithmetic formulas (recall that no superpolynomial-lower bounds are known for this class) where the learner outputs a polynomial-size formula as its hypothesis, we can improve on Lemma 4:

Lemma 5. *Assume that polynomial-size arithmetic circuits are exactly learnable in polynomial-time and the output hypothesis is an arithmetic formula. Then if Permanent is computed by polynomial-size arithmetic formulas, Permanent is in P^{RP}.*

Proof: The proof is similar to the proof of Lemma 4, except that we can *deterministically* construct counterexamples using an oracle for RP. This is because the hypothesis is a formula rather than a circuit, and, as discussed in Section 2.1, its degree as a polynomial is polynomial is $O(n^a)$ for some constant a. We can then choose a field F of size $O(n^{2a})$ and substitute all values of F for x_1. For each substitution to x_1, query the RP oracle to determine if this restricted polynomial is non-zero. For some value $a = x_1 \in F$, the restricted polynomial must be non-zero. We can repeat this process to find an appropriate setting for $x_2, \ldots, x_n$. Since the size of F is at most $n^{O(1)}$, we will have found a polynomial-time algorithm for computing the permanent using an oracle for RP. □

Following the same outline for the proof of Theorem 14 but using Lemma 5 instead of Lemma 4 we obtain the following theorem:

Theorem 15. *Let C be a non-uniform class of polynomial-size arithmetic formulas. Assume that C is exactly learnable from membership and equivalence in polynomial time and that the output hypothesis is a an arithmetic formula. Then* $\mathsf{EXP}^{\mathsf{RP}} \not\subseteq C$.

Kabanets and Impagliazzo [7] have proved that there exists a function $f \in \mathsf{NEXP}^{\mathsf{RP}}$ that has superpolynomial arithmetic circuit complexity. Note that $\mathsf{NEXP}^{\mathsf{RP}}$ is not known to be contained in either $\mathsf{EXP}^{\mathsf{RP}}$ or $\mathsf{ZPEXP}^{\mathsf{RP}}$.

Conclusions and Open Problems. One interpretation of our results is that we have given added motivation for trying to develop learning algorithms for very restricted concept classes, as they would settle important and difficult questions in computational complexity theory. Techniques from circuit lower bounds have figured prominently in the development of powerful learning algorithms in the past (e.g., Linial et al. [25]), yet we are unaware of applications from learning theory to circuit lower bounds. An interesting open problem is to show that randomized subexponential time Exact (and PAC) learning algorithms yield new circuit lower bounds.

References

1. Pitt, L., Valiant, L.: Computational limitations on learning from examples. Journal of the ACM **35** (1988) 965–984
2. Gold, E.A.: Complexity of automaton identification from given data. Information and Control **37** (1978) 302–320
3. Alekhnovich, Braverman, Feldman, Klivans, Pitassi: Learnability and automatizability. In: FOCS: IEEE Symposium on Foundations of Computer Science (FOCS). (2004)
4. Kearns, M., Valiant, L.: Cryptographic limitations on learning Boolean formulae and finite automata. Journal of the ACM **41** (1994) 67–95
5. Kharitonov, M.: Cryptographic hardness of distribution-specific learning. In: Proceedings of the Twenty-Fifth Annual Symposium on Theory of Computing. (1993) 372–381

6. Jackson, J., Klivans, A., Servedio, R.: Learnability beyond AC^0. In: Proceedings of the 34th ACM Symposium on Theory of Computing. (2002)
7. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. In: Proceedings of the 35th ACM Symposium on the Theory of Computing, New York, ACM (2003) 355–364
8. Impagliazzo, R., Wigderson, A.: Randomness vs. time: Derandomization under a uniform assumption. Journal of Computer and System Sciences **63** (2001) 672–688
9. Valiant, L.: A theory of the learnable. Communications of the ACM **27** (1984) 1134–1142
10. Angluin, D.: Queries and concept learning. Machine Learning **2** (1988) 319–342
11. Buhrman, H., Fortnow, L., Thierauf, T.: Nonrelativizing separations. In: Proceedings of the 13th IEEE Conference on Computational Complexity. IEEE, New York (1998) 8–12
12. Miltersen, Vinodchandran, Watanabe: Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In: COCOON: Annual International Conference on Computing and Combinatorics. (1999)
13. Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Transactions of the American Mathematical Society **117** (1965) 285–306
14. Kannan, R.: Circuit-size lower bounds and non-reducibility to sparse sets. Information and Control **55** (1982) 40–56
15. Valiant, L.: The complexity of computing the permanent. Theoretical Computer Science **8** (1979) 189–201
16. Toda, S.: PP is as hard as the polynomial-time hierarchy. SIAM Journal on Computing **20** (1991) 865–877
17. Lipton, R.: New directions in testing. In Feigenbaum, J., Merritt, M., eds.: Distributed Computing and Cryptography. Volume 2 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence (1991) 191 – 202
18. Beaver, D., Feigenbaum, J.: Hiding instances in multioracle queries. In: Proceedings of the 7th Symposium on Theoretical Aspects of Computer Science. Volume 415 of Lecture Notes in Computer Science. Springer, Berlin (1990) 37–48
19. Buhrman, H., Homer, S.: Superpolynomial circuits, almost sparse oracles and the exponential hierarchy. In: Proceedings of the 12th Conference on the Foundations of Software Technology and Theoretical Computer Science. Volume 652 of Lecture Notes in Computer Science. Springer, Berlin, Germany (1992) 116–127
20. Babai, L., Fortnow, L., Nisan, N., A.Wigderson: BPP has subexponential time simulations unless EXPTIME has publishable proofs. Computational Complexity **3** (1993) 307–318
21. Beimel, A., Bergadano, F., Bshouty, N., Kushilevitz, E., Varricchio, S.: On the applications of multiplicity automata in learning. In: Proceedings of the Thirty-Seventh Annual Symposium on Foundations of Computer Science. (1996) 349–358
22. Klivans, Shpilka: Learning arithmetic circuits via partial derivatives. In: COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers. (2003)
23. Bshouty, Hancock, Hellerstein: Learning arithmetic read-once formulas. SICOMP: SIAM Journal on Computing **24** (1995)
24. Bshouty, Bshouty: On interpolating arithmetic read-once formulas with exponentiation. JCSS: Journal of Computer and System Sciences **56** (1998)
25. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, fourier transform, and learnability. Journal of the ACM **40** (1993) 607–620