Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 129 (2013)

Constructing Hard Functions from Learning Algorithms

Adam Klivans
klivans@cs.utexas.edu

Pravesh Kothari
kothari@cs.utexas.edu

Igor C. Oliveira[*]
oliveira@cs.columbia.edu

September 6, 2013

Abstract

Fortnow and Klivans proved the following relationship between efficient learning algorithms and circuit lower bounds: if a class $\mathcal{C} \subseteq \mathsf{P/poly}$ of Boolean circuits is exactly learnable with membership and equivalence queries in polynomial-time, then $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathcal{C}$ (the class $\mathsf{EXP}^{\mathsf{NP}}$ was subsequently improved to EXP by Hitchcock and Harkins). In this paper, we improve on these results and show

- If $\mathcal{C}$ is exactly learnable with membership and equivalence queries in polynomial-time, then $\mathsf{DTIME}(n^{\omega(1)}) \not\subseteq \mathcal{C}$. We obtain even stronger consequences if the class $\mathcal{C}$ is learnable in the mistake-bounded model, in which case we prove an average-case hardness result against $\mathcal{C}$.
- If $\mathcal{C}$ is learnable in polynomial time in the PAC model then $\mathsf{PSPACE} \not\subseteq \mathcal{C}$, unless $\mathsf{PSPACE} \subseteq \mathsf{BPP}$. Removing this extra assumption from the statement of the theorem would provide an unconditional proof that $\mathsf{PSPACE} \not\subseteq \mathsf{BPP}$.
- If $\mathcal{C}$ is efficiently learnable in the Correlational Statistical Query (CSQ) model, we show that there exists an explicit function f that is average-case hard for circuits in $\mathcal{C}$. This result provides stronger average-case hardness guarantees than those obtained by SQ-dimension arguments (Blum et al. 1993). We also obtain a non-constructive extension of this result to the stronger Statistical Query (SQ) model.

Similar results hold in the case where the learning algorithm runs in subexponential time.

Our proofs regarding exact and mistake-bounded learning are simple and self-contained, yield explicit hard functions, and show how to use mistake-bounded learners to "diagonalize" over families of polynomial-size circuits. Our consequences for PAC learning lead to new proofs of Karp-Lipton-style collapse results, and the lower bounds from SQ learning make use of recent work relating combinatorial discrepancy to the existence of hard-on-average functions.

[*]Supported in part by NSF grants CCF-0915929 and CCF-1115703.

ISSN 1433-8092

1 Introduction

Understanding the computational complexity of learning circuit classes continues to be an important area of study in theoretical computer science. For example, recent work of Gentry and Halevi [GH11] makes use of results on the complexity of learning depth-3 arithmetic circuits [KS09] to construct improved homomorphic encryption schemes. More generally, the relationship between the complexity of learning circuits and cryptography has been extensively studied over the last twenty years (e.g., [Val84] and [KV94a]).

Less is known regarding the relationship between learning circuit classes and proving circuit lower bounds. Historically, circuit lower bounds for a class $\mathcal{C}$ typically precede the design of a learning algorithm for $\mathcal{C}$. Some intuition for this fact is that circuit lower bounds usually reveal important structural properties of these circuit classes, allowing them to be learned in some non-trivial way.

Fortnow and Klivans [FK09] were the first to codify this intuition and prove formally that efficient learning algorithms (in a variety of models) for a circuit class $\mathcal{C}$ yield circuit lower bounds against $\mathcal{C}$. Their result reduces the task of proving circuit lower bounds to the task of designing efficient learning algorithms. They showed, for example, that a polynomial-time PAC learning algorithm for $\mathcal{C}$ separates BPEXP from $\mathcal{C}$. Additionally they proved that a subexponential time *exact* learning algorithm separates $\mathsf{EXP}^{\mathsf{NP}}$ from $\mathcal{C}$ (this was subsequently improved to EXP by Hitchcock and Harkins [HH11] using techniques from resource bounded measure). Their proof uses a variety of classic complexity-theoretic results such as Toda's theorem, the complexity of the Permanent, collapse theorems ($\mathsf{EXP} \subseteq \mathsf{P/poly} \Rightarrow \mathsf{EXP} = \mathsf{MA}$ [BFNW93]), and hierarchy theorems.

In this paper we prove that it is possible to derive stronger circuit lower bounds from learning algorithms. Our results significantly improve and expand on the above initial work by Fortnow and Klivans. In many cases our proofs are simple, self-contained, and do not use machinery from computational complexity. We obtain these consequences in a variety of well-known learning models: PAC, online (mistake-bounded), exact, and statistical query learning. We begin by outlining our main results and contrasting them with previous work.

1.1 Our Results: Improved Separations for Online and Exact Learning

Our first set of results deals with learning algorithms in the online (mistake-bounded) model of learning and Angluin's model of exact learning with membership and equivalence queries. Recall that in the mistake-bounded model of learning, a function c from some class $\mathcal{C}$ is fixed, and a learner is sequentially presented with an arbitrary sequence of examples. After receiving example x_i, the learner must output its prediction for $c(x_i)$. We say that a learner succeeds with mistake bound m if for any (possibly infinite) sequence of examples, the learner makes at most m mistakes. The time-complexity $T(n, s)$ of the learner is the amount of time taken when presented with an example of length n and when c has size at most s. We prove the following theorem relating mistake-bounded learning to circuit lower bounds:

Theorem 1. *Let $\mathcal{C}^s$ be a non-uniform class of circuits where each $c \in \mathcal{C}^s$ has size at most s (according to some fixed representation). If $\mathcal{C}^s$ is learnable in the mistake-bounded model in time $T = T(n, s)$ and mistake bound $M = M(n, s) < 2^n$, then there exists an explicit function f computable in $\mathsf{DTIME}(M \cdot T)$ such that $f \notin \mathcal{C}^s$.*

Our proof actually shows that f is $\Omega(1/M)$-far from every $c \in \mathcal{C}$. For the class of polynomial-size circuits, the above theorem yields new circuit lower bounds as long as the learning algorithm has non-trivial run-time and mistake bound. If the learning algorithm is efficient (polynomial run-time and mistake bound) we obtain the following corollary:

Corollary 2. *Let $\mathcal{C}$ be any class of polynomial-size circuits (e.g., $\mathsf{AC}^0, \mathsf{TC}^0, \mathsf{P/poly}$). If $\mathcal{C}$ is efficiently learnable in the mistake-bounded model then $\mathsf{DTIME}(n^{\omega(1)}) \not\subseteq \mathcal{C}$.*

With more work, we prove analogous results for Angluin's model of exact learning with membership and equivalence queries. In this model the learner is required to exactly learn the target function c, and is allowed to query the value $c(x)$ on any input x (membership query). The learning algorithm can also check if a proposed hypothesis h is equivalent to c (equivalence query). If this is not the case, it is presented with a counterexample w for which $h(w) \neq c(w)$. It is not hard to see that learnability in the mistake-bounded model implies learnability in Angluin's model.

Previous work due to Fortnow and Klivans [FK09] and also Hitchcock and Harkins [HH11] proved, under a learning assumption, the existence of a hard function for polynomial-size circuits in $\mathsf{EXP}^{\mathsf{NP}}$ and EXP, respectively. In contrast, our proof yields an explicit function that is computable in any superpolynomial time class. Since we are able to explicitly construct hard functions in lower (uniform) deterministic time complexity classes (recall that our learning algorithms are assumed to be deterministic), we can prove that efficient learning algorithms imply a full derandomization of BPP:

Corollary 3. *Let $\mathcal{C}$ be the class of linear-size circuits. If $\mathcal{C}$ is efficiently exactly learnable (or learnable in the mistake-bounded model) then* $\mathsf{P} = \mathsf{BPP}$.

Our results for mistake-bounded and exact learning use the learning algorithms themselves in non-standard ways to construct hard functions. For example, in a typical learning scenario, the learning algorithm receives examples all of which are labelled according to some fixed function $c \in \mathcal{C}$. In our setting, we will run our mistake-bounded or exact learning algorithms in stages, using *different functions* to provide labels for the examples in each stage. More specifically, we will continually label new examples according to the *negation* of the learning algorithm's current hypothesis. At first glance, it would seem that no guarantees can be made about a learning algorithm that is not given examples labelled according to a fixed function. Still, we are able to use the learning algorithm to "fool" all potential functions that it might have to learn. At a high level, we consider this a sort of diagonalization over all elements of $\mathcal{C}$ (we give more details on this procedure in Section 3).

In contrast, the work of Fortnow and Klivans is considerably more complicated, requiring non-trivial collapse arguments, hierarchy theorems, Toda's theorem, and various well-known properties of the Permanent function in order to obtain their conclusion. Hitchcock and Harkins used betting games and ideas from resource bounded measure to obtain their improvement. As can be seen in Section 3.2, our proof is simple, self-contained, and yields a much finer separation. We note that the same proof was discovered independently by Impagliazzo and Kabanets [Kab13].

1.2 Our Results: Hard Functions in PSPACE

The previous set of results showed that *deterministic* learning algorithms in the exact or mistake-bounded model imply hard functions computable in subexponential-time uniform complexity classes. We also investigate the possibility of constructing hard functions in PSPACE, given the existence of non-trivial *randomized* learning algorithms. We prove that unless PSPACE lies in randomized sub-exponential time, non-trivial learning algorithms in the PAC model imply the existence of hard functions in PSPACE. Actually, this is true even if the PAC learning algorithm is allowed membership queries and only works under the uniform distribution:

Theorem 4. *Let $\mathcal{C}$ be any circuit class and suppose that there exists a randomized algorithm that* PAC *learns $\mathcal{C}$ under the uniform distribution using membership queries in time $O(T(n, \mathsf{size}(c)))$, where $c \in \mathcal{C}$ is the unknown concept. Then, for any function $s : \mathbb{N} \to \mathbb{N}$, at least one of the following conditions hold:*

- *(i) There are languages in* PSPACE *not computed by circuits from $\mathcal{C}$ of size s; or*
- *(ii)* $\mathsf{PSPACE} \subseteq \mathsf{BPTIME}(T(n, s))$.

In contrast, Fortnow and Klivans proved that for any circuit class $\mathcal{C} \subseteq \mathsf{P/poly}$, if $\mathcal{C}$ is PAC learnable under the uniform distribution by a polynomial-time algorithm with membership queries then $\mathsf{BPEXP} \nsubseteq \mathcal{C}$. Theorem 4 extends their work in the following directions: (*i*) we obtain interesting consequences for PSPACE instead of BPEXP; (*ii*) it is possible to derive new results for PSPACE even in the case that the learning algorithm does not run in polynomial time; (*iii*) $\mathcal{C}$ does not need to be contained in $\mathsf{P/poly}$, which means that this result can (under the learning assumptions) be used to obtain super-polynomial lower bounds. In Section 5, we explain how Fortnow and Klivans's original result can be derived from Theorem 4.

Note that the second condition in the conclusion of this theorem does not depend on the original circuit class that appears in the hypothesis. While this seems odd at first, we give a simple proof that removing this "or" condition from the conclusion of the theorem would give us an unconditional proof that $\mathsf{PSPACE} \nsubseteq \mathsf{BPP}$. In other words, proving strong theorems of the form "learning implies circuit lower bounds" yields important uniform separations.

Theorem 4 also explains the difficulty of designing non-trivial PAC learning algorithms for the class of polynomial-size depth-two threshold functions, also known as TC^0_2. This is one of the smallest circuit classes for which there are no known non-trivial circuit lower bounds. In particular, it could be the case that any language in BPEXP is in TC^0_2. Our result shows that the existence of a non-trivial PAC learning algorithm for this class provides strong evidence that PSPACE does not admit such circuits. Previous results required stronger assumptions. For instance, the original theorem proven by Fortnow and Klivans [FK09] assumes efficient PAC learnability, and the cryptographic hardness results of Klivans and Sherstov [KS09] do not hold with respect to the uniform distribution or when learner is allowed membership queries.

The main idea of the proof is to rework the Fortnow and Klivans approach but use a PSPACE-complete problem described by Trevisan and Vadhan [TV07] that is downward self-reducible and self-correctible. In contrast, Fortnow and Klivans used the Permanent function (and its well-known self-reducibility properties) but had to first go through a "collapse" argument to arrive in a scenario where the Permanent is complete for PSPACE. The proof of Theorem 4 is presented in Section 5.

We also observe that Karp-Lipton style collapse results follow easily from a relativized version of Theorem 4 and Occam's Razor (Blumer et al. [BEHW87]), for any complexity class with complete problems that are both downward self-reducible and self-correctible. To the best of our knowledge, this is the first proof of these theorems that relies on Occam's Razor, an important technique in computational learning theory.

1.3 Our Results: Average-Case Hard Functions from Statistical Query Learning

Our results above show that nontrivial learning algorithms in the exact, mistake-bounded, or PAC model yield functions that are hard to compute in the worst-case. We show that even weak learning algorithms that use only *Correlational Statistical Queries* (CSQs) yield not just circuit lower bounds but explicit functions that are hard to compute *on average*. Informally, a CSQ learner is allowed to make queries of the form $\mathbb{E}[q \cdot c]$ where c is the target function and q is some fixed (polynomial-time computable) predicate. The learner then receives an estimate of the true value of this query to within τ, a "tolerance" parameter. CSQ learning has been an active area of study recently in computational learning theory. It is known [Fel08] that the class of functions that are Evolvable (in the sense of Valiant[Val09]) are exactly equal to the functions that are learnable by CSQs (we define Correlational Statistical Queries formally in Section 2.4). We give the following consequence for CSQ learning:

Theorem 5. *Let $\mathcal{C}^s$ be a class of Boolean concepts of size s that is learnable in the CSQ model in time and query complexity $T(n, 1/\epsilon, 1/\tau, s)$ where ϵ is the accuracy parameter and $\tau \leq \mathsf{min}(\epsilon, 1-2\epsilon)$*

is the tolerance parameter. Then there exists a function $f \in \mathsf{DTIME}(\mathsf{poly}(T(n, 1/\epsilon, 1/\tau, s))$ such that for every $c \in \mathcal{C}^s$, f disagrees with c on at least a $\tau/4$ fraction of inputs.

We note that a weak average-case hardness for an explicit function (parity) can be obtained by a simple argument based on SQ-dimension [BFJ$^+$94]. For example, it follows from the definition of SQ-dimension that if $\mathcal{C}$ has polynomial SQ-dimension any $c \in \mathcal{C}$ differs from parity on a non-zero but negligible fraction of inputs (this is discussed in more detail in Section 7.2). Since τ is at least an inverse polynomial, Theorem 5 yields stronger average-case hardness result against $\mathcal{C}$.

The proof of Theorem 5 uses a diagonalization trick similar to the one used for obtaining lower bounds from online learning algorithms in order to construct a family of functions $\mathbb{G}$ such that for every $c \in C$ there is some $g \in \mathbb{G}$ that weakly *approximates* c. We can then apply recent work due to Chattopadhyay et al. [CKK12] relating explicit low-discrepancy colorings to hard on average functions to find an explicit function f that has low correlation with every function in $\mathbb{G}$. This will be the desired hard on average function.

For a subtle technical reason, we need additional assumptions to obtain results for the full SQ model of learning (see Section 7.3). We leave getting rid of these assumptions as an interesting open problem and discuss the difficulty in Section 8.

2 Preliminaries and Notation

A Boolean function (concept) maps $\{-1,1\}^n \to \{-1,1\}$. A family of Boolean functions $f = \{f_n\}_{n\in\mathbb{N}}$, where $f_n : \{-1,1\}^n \to \{-1,1\}$, naturally corresponds to the language $L_f = \{x \in \{-1,1\}^n \mid f(x) = -1\}$. We use $\mathcal{U}$ (or $\mathcal{U}_n$) to denote the uniform distribution on $\{-1,1\}^n$.

We will use $\mathcal{C} = \cup_{n\in\mathbb{N}}\, \mathcal{C}_n$ to denote a representation class of Boolean functions, such as DNFs, Boolean circuits, depth-two threshold circuits, etc. The size of $c \in \mathcal{C}$ in its representation will be denoted by $\mathsf{size}(c)$. For concreteness, $\mathsf{size}(c)$ can be assumed to be the number of bits required to write down the representation of c. We require the representation be such that the value at any input of any function c can be computed in deterministic time polynomial in n and the size of the representation. We will use T for denoting time bounds, and s for denoting sizes of representations, both of which we assume to be constructive and non-decreasing without explicit notice.

We now set up some notation to talk about languages and representation classes.

Definition 1 (Languages and Representation Classes)**.** *For any language $L \subseteq \{-1,1\}^*$, we denote the restriction of L to strings of length n by L_n. For any size function $s : \mathbb{N} \to \mathbb{N}$ and representation class $\mathcal{C}$,*

$$\mathcal{C}^s = \{L \subseteq \{-1,1\}^* \mid \forall n\ \exists c \in \mathcal{C}_n \text{ with } \mathsf{size}(c) \le s \text{ such that } x \in L \Leftrightarrow c(x) = -1\}.$$

Let $\mathsf{P/poly}[\mathcal{C}] = \cup_{c>0}\mathcal{C}^{n^c}$. When $\mathcal{C}$ is the class of circuits of AND*,* OR *and* NOT *gates, we denote $\mathcal{C}^s$ by $\mathsf{SIZE}(s)$ and $\mathsf{P/poly}[\mathcal{C}]$ by just $\mathsf{P/poly}$.*

As such each one of our results will hold for sufficiently large n and we will skip noting this explicitly in the interest of clarity. If we need to stress that we are dealing with functions in $\mathcal{C}$ of n dimensions, we will make this explicit by writing $\mathcal{C}^s_n$ for the class $\mathcal{C}^s$.

To denote that an algorithm has oracle access to a function family f, we write $\mathcal{A}^f$. Equivalently, if we see the oracle as a language L, we write $\mathcal{A}^L$.

We now define the various learning models we will deal with in this paper. We do not require any of our learning algorithms to be *proper*, that is, the hypothesis output by the algorithms need not be from the representation classes they learn.

2.1 Online Mistake Bound Learning

In the mistake-bounded model of learning, a concept c from some class $\mathcal{C}$ is fixed, and a learner is presented with an arbitrary sequence of examples. After receiving each example x_i, the learner must output its prediction for $c(x_i)$. The learner succeeds with mistake bound M if for any sequence of examples, the learner makes at most M mistakes. Formally:

Definition 2 (Mistake Bound Learning)**.** *Let $\mathcal{C}$ be any class of Boolean functions over an arbitrary domain X. A mistake bound learning algorithm $\mathcal{A}$ for $\mathcal{C}$ proceeds in rounds. Let $c \in \mathcal{C}$ be the target function. In round $i \geq 1$, $\mathcal{A}$:*

1. *is presented with an example point $x_i \in X$, and outputs a label $\mathcal{A}(x_i)$.*
2. *is provided (by the target function oracle) with the correct label $c(x_i)$.*
3. *runs an update procedure.*

$\mathcal{A}$ learns $\mathcal{C}^s$ with mistake bound $M(n, s)$ and time $T(n, s)$, if for any $c \in \mathcal{C}^s$ and any (possibly infinite) sequence of examples from X, $\mathcal{A}$ makes at most $M(n, s)$ mistakes while outputting the labels, and the update procedure runs in time $T(n, s)$.

2.2 Angluin's Model of Exact Learning

Angluin's model of exact learning [Ang88] provides the learner with more powerful access to the target function than the Online Mistake Bound Learning Model. It can be easily shown that any mistake bound algorithm can be translated into an exact learner in Angluin's model while preserving efficiency.

Let $c \in \mathcal{C}^s$ be a target function. In this model, the learning algorithm is allowed to ask two kinds of queries about c to the target function oracle:

- Membership Queries: the learner presents a point $x \in \{-1, 1\}^n$ and the target function oracle replies with $c(x)$.
- Equivalence Queries: the learner presents a Boolean function $\tilde{h} : \{-1, 1\}^n \to \{-1, 1\}$ to the oracle (represented as a circuit). If $\tilde{h} = c$, the oracle responds with "yes". Otherwise, the oracle responds with "not equivalent", and provides a counter example $x \in \{-1, 1\}^n$ such that $\tilde{h}(x) \neq c(x)$.

We can now define an exact learning algorithm for a class of Boolean functions $\mathcal{C}^s$.

Definition 3 (Exact Learning in Angluin's Model)**.** *A deterministic algorithm $\mathcal{A}$* exact learns *a representation class of Boolean functions $\mathcal{C}^s$ in time $T(n, s)$ and queries $Q(n, s)$ if for any target function $c \in \mathcal{C}^s$, $\mathcal{A}$ makes at most $Q(n, s)$ membership and equivalence queries to the oracle for c and outputs a hypothesis $h : \{-1, 1\}^n \to \{-1, 1\}$ such that $h(x) = c(x)$ for all $x \in \{-1, 1\}^n$ in time $T = T(n, s)$. Further, we assume that any equivalence query, $\tilde{h}$, is computable in time $O(T)$ on any input.*

2.3 PAC Learning

In the most common PAC learning framework, there is an unknown concept $c \in \mathcal{C}_n$ to be learned, and the learning algorithm receives random examples labelled consistently with c according to some fixed but unknown distribution $\mathcal{D}$ over $\{-1, 1\}^n$. Here we concentrate on the stronger model in which the learner can ask membership queries (present any point x and obtain the value of target function $c(x)$) about the unknown concept, and only needs to learn under the uniform distribution. In other words, we prove circuit lower bounds from a weaker assumption, namely, the existence of learning algorithms in a more powerful model.

Definition 4. *Let $\mathcal{C}$ be any class of Boolean functions. An algorithm $\mathcal{A}$* PAC-*learns $\mathcal{C}$ if for every $c \in \mathcal{C}$ and for any $\epsilon, \delta > 0$, given membership query access to c, algorithm $\mathcal{A}$ outputs with probability at least $1 - \delta$ over its internal randomness, a hypothesis h such that $\Pr_{x \sim \mathcal{U}_n}[c(x) \neq h(x)] \leq \epsilon$.*

We measure the running time of $\mathcal{A}$ as a function $T = T(n, 1/\epsilon, 1/\delta, \mathsf{size}(c))$, and require that h itself can be evaluated in time at most T. We say that $\mathcal{A}$ is efficient if T is bounded above by a fixed polynomial in its parameters.

2.4 Statistical Query Learning

Statistical query learning, defined by Kearns et. al. [Kea98] is a natural variant of PAC-learning when the underlying data is noisy. We start with the definition of Statistical Queries (SQs).

Definition 5 (Statistical Query Oracles and SQs). *Let $\mathcal{C}$ be a concept class on $\{-1,1\}^n$. For any $c \in \mathcal{C}$, a statistical query oracle for c of tolerance $\tau > 0$, $\mathsf{STAT}(c, \tau)$, takes input any representation of a bounded function $\psi : \{-1,1\}^n \times \{-1,1\} \to [-1,1]$ and returns $v \in [-1,1]$ such that $|\mathbb{E}_{x \sim \mathcal{U}}[\psi(x, c(x))] - v| \leq \tau$. A query function ψ is said to be target independent if for every $x \in \{-1,1\}^n$ and $y \in \{-1,1\}$, $\psi(x,y) = \psi(x,-y)$, that is ψ doesn't depend on the target function c.*

Bshouty and Feldman ([BF02]) noted that any Statistical Query can be simulated by 2 target independent queries and 2 correlational queries. We include their simple proof for completeness.

Proposition 6 (Bshouty and Feldman [BF02]). *Any statistical query can be decomposed into two statistical queries that are independent of the target and two correlational queries.*

Proof. Let ψ be a statistical query, and let c be the target function. The result follows immediately from the following equation:

$$\begin{aligned}\mathbb{E}[\psi(x, c(x))] &= \mathbb{E}\left[\psi(x,-1)\frac{1-c(x)}{2} + \psi(x,1)\frac{1+c(x)}{2}\right] \\ &= \frac{1}{2}\mathbb{E}[\psi(x,1)c(x)] - \frac{1}{2}\mathbb{E}[\psi(x,-1)c(x)] + \frac{1}{2}\mathbb{E}[\psi(x,1)] + \frac{1}{2}\mathbb{E}[\psi(x,-1)].\end{aligned}$$

□

The Correlational Statistical Query (CSQ) Oracle is a less powerful version of the SQ oracle which answers only correlational queries.

Definition 6 (Correlational Statistical Query Oracle). *Let $\mathcal{C}$ be a concept class on $\{-1,1\}^n$ and $\mathcal{D}$ be any distribution on $\{-1,1\}^n$. For any $c \in \mathcal{C}$, a correlational statistical query oracle for c of tolerance $\tau > 0$, $\mathsf{CSTAT}(c, \tau)$, takes input any representation of a bounded function $\psi : \{-1,1\}^n \to [-1,1]$ and returns $v \in [-1,1]$ such that $|\langle c, \psi \rangle_{\mathcal{D}} - v| \leq \tau$.*

We now define learning from SQs and CSQs. We note that CSQ learning algorithms on are equivalent to Valiant's [Val09] model of evolvability [Fel08].

Definition 7 (Correlational Statistical Query Learning). *Let $\mathcal{C}$ be a representation class of Boolean functions on $\{-1,1\}^n$. A (Correlational) Statistical Query learning algorithm $\mathcal{A}$ learns $\mathcal{C}^s$ on the uniform distribution in time $T = T(n, \frac{1}{\epsilon}, \frac{1}{\tau}, s)$ and queries $Q = Q(n, \frac{1}{\epsilon}, \frac{1}{\tau}, s)$ if, for any $c \in \mathcal{C}^s$ and any $\epsilon \geq \tau > 0$, $\mathcal{A}$ makes Q queries to $\mathsf{STAT}(c, \tau)$ ($\mathsf{CSTAT}(c, \tau)$) and uses at most T time units to return a hypothesis h such that*

$$\Pr_{x \sim \mathcal{D}}[h(x) \neq c(x)] \leq \epsilon.$$

$\mathcal{A}$ is said to be efficient if both T and Q depend polynomially on n, $\frac{1}{\epsilon}$, $\frac{1}{\tau}$ and s.

Using Proposition 6, we will assume that any SQ algorithm $\mathcal{A}$ learning $\mathcal{C}^s$ actually makes only target independent and correlational queries. Further, we will assume that each target independent query is a Boolean function specified by a circuit of size $\mathsf{poly}(s)$.

3 Mistake-Bounded and Exact Learning Algorithms Encode Hard Functions

In this section we give a simple and direct method for constructing a hard function given a (deterministic) mistake-bounded or exact learning algorithm. Specifically, we will show that if $\mathcal{C}^s$ (here $\mathcal{C}^s$ equals all concepts in $\mathcal{C}$ of size at most s) is learnable in the Online Mistake Bound Model [Lit88] (or the Exact Learning Model [Ang88]) with mistake bound (or number of queries for Exact Learning) less than 2^n, then there is a function computable in a uniform time class that is not computable by any function in $\mathcal{C}^s$. As a corollary, we will see that polynomial-time (deterministic) mistake-bounded or exact learning algorithms for even linear-sized circuit classes implies that $\mathsf{P} = \mathsf{BPP}$. Previous work relating learning algorithms to circuit lower bounds obtained only subexponential-time simulations of BPP.

Our proof shows how to use a mistake-bounded or exact learning algorithm to "fool" every circuit from some large class. We do this by iteratively presenting the learning algorithm labeled examples from *different* functions at each iteration (typically a learning algorithm will only succeed if it is presented with labeled examples from a function fixed in advance from within the class). Recall that our goal here is *not* to obtain an accurate hypothesis, but to construct a hard function using the learning algorithm as a (black box) subroutine. The running time of the algorithm for evaluating the hard function on any input is dependent on the time and mistake bound (or queries for exact learning) of the learning algorithm.

3.1 Lower Bounds from Mistake Bounds

In this section, we present our "diagonalization" trick and show that the existence of a Mistake-Bounded learning algorithm for a class $\mathcal{C}$ yields an explicit hard function for $\mathcal{C}$:

Theorem 7 (Mistake Bound Learning yields Lower Bounds)**.** *Let $\mathcal{C}$ be any class of Boolean functions. Suppose there exists an algorithm $\mathcal{A}$ that learns any $c \in \mathcal{C}$ with mistake bound $M = M(n, s)$ and time $T = T(n, s)$, where $s = s(n)$. Then, for any size s such that $M < 2^n$, there exists a function $f \in \mathsf{DTIME}(M \cdot T)$ such that for any $c \in \mathcal{C}^s$, we have*

$$\Pr_{x \sim \mathcal{U}_n}[f(x) \neq c(x)] > \frac{1}{M+1} - \frac{1}{2^n}.$$

Proof. We must define function f and prove that it cannot be computed by any circuit of size s (pseudocode for the hard function f is given in Algorithm 1). To do this, we describe f's behavior on input x.

Let $\{-1, 1\}^n$ be partitioned into consecutive blocks in lexicographic order $E_1, \ldots, E_k$, each of size $t = M + 1$ (the last block may have size smaller than t). A function ℓ is initialized to equal -1. On input x, f determines $j \in [k]$ such that $x \in E_j$. It then simulates learner $\mathcal{A}$ for $t' = |E_j|$ iterations by presenting it examples from E_j in lexicographic order. Let $\{x_1, x_2, \ldots, x_t\}$ be the examples in E_j. On the i^{th} iteration, f simulates $\mathcal{A}$ and presents it with example x_i. The learner responds with its prediction, $\mathcal{A}(x_i)$. The function f sets $\ell(x_i) = -\mathcal{A}(x_i)$ and informs the learner that it has made a mistake. The function f then simulates the update procedure of $\mathcal{A}$ by using the "true" label of x_i, namely $\ell(x_i)$. At the end of $|E_j|$ iterations, f halts and outputs $f(x) = \ell(x_i)$. Since $x \in E_j$, f halts in at most t iterations. Clearly f can be computed on any input in time $O(M \cdot T)$.

Algorithm 1 Hard function f that uses mistake-bounded learner $\mathcal{A}$ as a subroutine

```
    Input: $x \in \{-1,1\}^n$.
    Output: A value in $\{-1,1\}$.
1:  Set $t = M+1$, where $M = M(n, s(n))$, and let $\{-1,1\}^n$ be partitioned into
    sequential blocks $E_1, E_2, \ldots, E_k$ of size $t$, where $k = \lceil 2^n/t \rceil$ (the last block may
    contain less than $t$ points). Initialize function $\ell$ on $E_j$ to the constant $-1$.
2:  Find $j$ such that $x \in E_j$. Let $t' = |E_j|$ (note that $t' = M+1$ for any $j < k$).
3:  Obtain all the points in $E_j$ and order them lexicographically as $\{x_1, x_2, \cdots x_{t'}\}$.
4:  Start simulating the learner $\mathcal{A}$ on the sequence of points $\{x_1, x_2, \cdots x_{t'}\}$.
5:  for $i = 1$ to $t'$ do
6:      Simulate $\mathcal{A}$ by presenting $x_i$ and obtain prediction: $\mathcal{A}(x_i)$.
7:      Tell the learner $\mathcal{A}$ that it made a mistake and report true label of $x_i$ as $\ell(x_i) =$
        $-\mathcal{A}(x_i)$.
8:      Simulate the update procedure of $\mathcal{A}$.
9:  end for
10: return $\ell(x)$.
```

Assume for a moment that for any $c \in \mathcal{C}^s$, functions f and c differ in at least one point in E_j whenever $|E_j| = M+1$. Then if $|E_j| = M+1$ for each $1 \leq j \leq k$, we have that $\Pr_{x\sim\mathcal{U}_n}[h(x) \neq c(x)] = \frac{1}{M+1}$. If, on the other hand, $|E_k| < M+1$, then $\Pr_{x\sim\mathcal{U}_n}[h(x) \neq c(x)] \geq \frac{1}{M+1} \cdot (1 - \frac{|E_k|}{2^n}) > \frac{1}{M+1} - \frac{1}{2^n}$.

Let $j < k$ so $|E_j| = M+1$. To see why f and c differ on E_j, observe that if there exists a $c \in \mathcal{C}^s$ consistent with ℓ on all the examples in E_j, then the sequence of examples $\{x_1, \ldots, x_t\}$ and labels given to $\mathcal{A}$ by f are consistent with c. But we have forced the learner to make exactly $M+1$ mistakes on this sequence. This is a contradiction to the mistake bound of $\mathcal{A}$. Thus, the labeling given by ℓ for $\{x_1, \ldots, x_t\}$ cannot be consistent with any $c \in \mathcal{C}^s$. □

3.2 Exact Learning Yields Circuit Lower Bounds

In this section we show that the existence of an algorithm that learns a class $\mathcal{C}$ in Angluin's model of exact learning using less than 2^n membership and equivalence queries implies lower bounds against $\mathcal{C}$. Learnability in mistake-bounded model implies learning in the exact model, thus the results presented here are stronger than those stated in the previous section. On the other hand, we do not obtain an average case hard function as we could from a mistake-bounded algorithm. In the proof, we make use of a similar "diagonalization" trick, but there are a few more complications involved in simulating the equivalence and membership queries.

Theorem 8 (Exact Learning yields Lower Bounds)**.** *Let $\mathcal{C}$ be a class of Boolean functions. Suppose there exists an exact learning algorithm $\mathcal{A}$ that exact learns any target function $c \in \mathcal{C}$ in time $T = T(n,s)$ and $< 2^n$ equivalence and membership queries. Then there exists a function $f \in \mathsf{DTIME}(T^2)$ such that $f \notin \mathcal{C}^s$.*

Proof. As in the previous section, we describe a procedure to compute f using blackbox access to the exact learning algorithm $\mathcal{A}$. We will show that $f \notin \mathcal{C}^s$ and $f \in \mathsf{DTIME}(T^2)$. Let x be the input to f. Then f simulates the learner $\mathcal{A}$ and must give responses to the membership queries and equivalence queries that $\mathcal{A}$ makes. The function f keeps track of the membership queries made by $\mathcal{A}$ and counterexamples (in response to equivalence queries made by $\mathcal{A}$) in the set S. If $\mathcal{A}$ makes a membership query and asks for the label of w, and $w \notin S$, f replies with -1, adds w to the set S, and defines $\ell(w) = -1$. Otherwise f responds with $\ell(w)$. If $\mathcal{A}$ makes an equivalence query for hypothesis $\tilde{h} : \{-1,1\}^n \to \{-1,1\}$, f replies with "not equivalent", returns counterexample y, the lexicographically first string not in S (recall that $Q < 2^n$), and adds y to S. In addition, f sets $\ell(y) = -\tilde{h}(y)$.

If during f's simulation, $\mathcal{A}$ halts and outputs a hypothesis h, then f chooses a string y, the lexicographically smallest string not in S, adds y to S, and sets $\ell(y) = -h(y)$. This guarantees that ℓ differs from h on at least one point in S. Finally, we describe what f should output on input x. If $x \in S$, output $\ell(x)$. Otherwise, output -1.

We will need the following simple claim:

Claim 1. *Suppose an exact learner $\mathcal{A}$ for $\mathcal{C}^s$, running in time $T = T(n,s)$ that makes at most $Q = Q(n,s) < 2^n$ queries is provided answers to all its membership and equivalence queries that are consistent with some $c \in \mathcal{C}^s$. Let S be the union of the set of all membership queries made by $\mathcal{A}$ and the set of all counterexamples presented to $\mathcal{A}$. Then, if any $c' \in \mathcal{C}^s$ satisfies $c'(x) = c(x)$ for all $x \in S$ then, $c(x) = c'(x)$ for every $x \in \{-1,1\}^n$.*

Proof of Claim. Since $\mathcal{A}$ is an exact learner and all the membership and equivalence queries made by it are answered with replies consistent with c, $\mathcal{A}$ must halt in at most T steps after making at most Q queries with a hypothesis h such that $h(x) = c(x)$ for every $x \in \{-1,1\}^n$. On the other hand, since $c'(x) = c(x)$ for every $x \in S$, the answers for the membership and equivalence queries received by $\mathcal{A}$ are consistent with c' also, and thus, $h(x) = c'(x)$ for every $x \in \{-1,1\}^n$. □

We will now argue that $f \notin \mathcal{C}^s$. We need the following notation: let $S_{\mathcal{A}}$ be the value of S and $\ell_{\mathcal{A}}$ be the value of ℓ when f stops simulating $\mathcal{A}$. Similarly, let S_f be the value of S and ℓ_f the value of ℓ when f halts (recall that S_f and ℓ_f differ from $S_{\mathcal{A}}$ and $\ell_{\mathcal{A}}$ only if $\mathcal{A}$ returns a hypothesis h before f stops simulating it, in which case $S_{\mathcal{A}} \subset S_f$ and $\ell_{\mathcal{A}}$ and ℓ_f agree on all points in $S_{\mathcal{A}}$).

Suppose that there exists $c \in \mathcal{C}^s$ such that $f(x) = c(x)$ for every $x \in S_{\mathcal{A}}$. In other words, the replies to the queries made by the algorithm $\mathcal{A}$ are consistent with c. In this case, $\mathcal{A}$ must halt and return a hypothesis h in at most T steps. Moreover, since $\mathcal{A}$ is an exact learner, $h = c$. By Claim 1, c is the unique function in $\mathcal{C}^s$ that is consistent with f on $S_{\mathcal{A}}$. Thus, if f is computed by some function in $\mathcal{C}^s$, then $f = c$. But notice that, in this case, the procedure for computing f guarantees that there exists a $y \in S_f \setminus S_{\mathcal{A}}$ such that $f(y) \neq h(y)$. This implies that $f \neq c$. Thus, there is no function in $\mathcal{C}^s$ that computes f.

On the other hand if for every $c \in \mathcal{C}^s$, there is some value $x \in S_{\mathcal{A}}$ such that $f(x) \neq c(x)$, then we immediately conclude that f is not computed by any $c \in \mathcal{C}^s$. In either case, we have proved that $f \notin \mathcal{C}^s$.

The function f can simulate $\mathcal{A}$ in time $O(T)$. Since $\mathcal{A}$ makes at most T equivalence queries, each of which is computable at any point in deterministic time $O(T)$ (Section 2.2), $\mathcal{A}$ spends at most $O(T^2)$ time answering equivalence queries. All other computations of f involve searching for strings outside S which takes at most $O(S) = O(T)$ time. Thus we have that f runs in time at most $O(T^2)$.

□

As a simple corollary we obtain that efficient exact learnability of $\mathcal{C}$ yields $\mathsf{DTIME}(n^{\omega(1)}) \not\subseteq \mathsf{P/poly}[\mathcal{C}]$. We now apply the theorem above to the special case of $\mathsf{SIZE}(n)$ to compare our results with [FK09] and [HH11].

Corollary 9. *Suppose $\mathsf{SIZE}(n)$ is learnable -*

- *by a Mistake-Bounded Algorithm in time and mistake bound polynomial in n or*
- *by an Exact Learning Algorithm in time polynomial in n.*

Then, $\mathsf{DTIME}(n^{\omega(1)}) \not\subseteq \mathsf{P/poly}$.

Proof. By a simple padding argument, $\mathsf{P/poly}$ is efficiently learnable in the respective models. Applying Theorems 7 and 8 yields the result. □

For a comparison, note that [FK09] proves that if $\mathsf{P/poly}$ is efficiently exactly learnable in Angluin's model, then $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{P/poly}$ and [HH11] improve this result to obtain the conclusion that $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$.

4 Derandomization Consequences from Exact Learners

The improvements in our lower bounds allow us to obtain a complete derandomization of BPP from efficient learnability of $\mathsf{P/poly}$.

We will require the following celebrated result of Impagliazzo and Wigderson:

Theorem 10 (Impagliazzo and Wigderson [IW96])**.** *If there exists* $L \in \mathsf{DTIME}(2^{O(n)})$ *and* $\delta > 0$ *such that* $L \notin \mathsf{SIZE}(2^{\delta n})$*, then* $\mathsf{P} = \mathsf{BPP}$.

Previous work obtained only subexponential deterministic simulations of BPP given the existence of efficient learning algorithms.

Corollary 11. *Suppose* $\mathsf{SIZE}(n)$ *is efficiently learnable in Angluin's model of exact learning with membership and equivalence queries. Then* $\mathsf{P} = \mathsf{BPP}$.

We only state the above corollary starting from exact learning algorithms, as mistake-bounded learnability implies exact-learnability.

Proof. We again use a padding argument here, although we have to be a bit more explicit with our parameters. Suppose $\mathsf{SIZE}(n)$ is exactly learnable in time $O(n^k)$. By padding $\mathsf{SIZE}(s)$ is learnable in time $O(s^k)$. Let $s = 2^{\delta n}$, where $\delta = \frac{1}{2k}$. The result now follows easily from Theorem 8 and Theorem 10. □

We note that using similar tools from derandomization, we can show that the existence of sub-exponential time mistake-bounded learning algorithms for polynomial size circuits implies subexponential-time derandomization of BPP.

5 Lower Bounds from PAC Learning: Hard Functions in PSPACE

In this section we shift gears and obtain hard functions in PSPACE from PAC learning algorithms. Previous work [FK09] showed the existence of hard functions in BPEXP. Indeed, here we prove that unless randomness can speed-up arbitrary space-bounded computations, any non-trivial PAC learning algorithm for a circuit class $\mathcal{C}$ yields a hard function in PSPACE against $\mathcal{C}$. We begin with a few important definitions.

Definition 8 (Downward Self-Reducibility)**.** *We say that a language L is* downward-self-reducible *if there is a deterministic polynomial time algorithm A such that for all $x \in \{-1,1\}^n$, $A^{L_{n-1}}(x) = L(x)$. In other words, A efficiently computes $L(x)$ on any input x of size n when given oracle access to a procedure that computes L on inputs of size $n-1$.*

Definition 9 (Self-Correctibility)**.** *We say that a language L is* $\alpha(n)$-self-correctible *if there is a probabilistic polynomial time algorithm A such that, for any Boolean function $c : \{-1,1\}^n \to \{-1,1\}$ that disagrees with L_n on at most an $\alpha(n)$-fraction of the inputs of size n, we have* $\Pr[A^c(x) = L(x)] \geq 2/3$ *for any* $x \in \{-1,1\}^n$.

Using an appropriate arithmetization of quantified Boolean formulas, Trevisan and Vadhan [TV07] proved that there exists a PSPACE-complete language that is both downward-self-reducible and self-correctible. Actually, by employing better self-correction techniques introduced by Gemmel and Sudan [GS92] and a standard composition with the Hadamard error-correcting code, it follows from their construction that [Vad12]:

Proposition 12. *There exists a* PSPACE*-complete language* L_{PSPACE} *that is both downward-self-reducible and* α*-self-correctible, where* $\alpha = 1/100$.

Finally, for any language $\mathcal{O}$, we denote by $\mathsf{BPTIME}(T(n))^{\mathcal{O}}$ the class of languages that can be computed by probabilistic algorithms that have oracle access to $\mathcal{O}$ and run in time $O(T(n))$.

Theorem 13 (PAC Learning yields Lower Bounds)**.** *Let* $\mathcal{C}$ *be any concept class and suppose that there exists an algorithm that* PAC *learns any* $c \in \mathcal{C}^s$ *under the uniform distribution using membership queries when given access to an oracle* $\mathcal{O}$[1] *in time* $T(n, 1/\epsilon, \log 1/\delta, s)$*. Let* $L^\star$ *be a language that is both downward-self-reducible and* $\alpha(n)$*-self-correctible. Then, at least one of the following conditions hold:*

(*i*) $L^\star \notin \mathcal{C}^s$*; or*

(*ii*) $L^\star \in \mathsf{BPTIME}(\mathsf{poly}(T(n, 1/\alpha(n), \log n, s)))^{\mathcal{O}}$.

The proof of this result follows the same high-level approach employed by Fortnow and Klivans [FK09], which we sketch next. Suppose for simplicity that we have an efficient PAC learning algorithm for $\mathcal{C}$ that does not depend on any oracle $\mathcal{O}$, and that $L^\star \in \mathsf{P/poly}[\mathcal{C}]$ (otherwise there is nothing to prove). Note that in order to prove Theorem 13, it is enough to show that $L^\star \in \mathsf{BPP}$. This can be obtained by combining the learning algorithm for $\mathcal{C}$ with the downward-self-reducibility and self-correctibility of $L^\star$. Roughly speaking, we "learn" how to compute $L^\star$ on all inputs of size at most n starting from inputs of constant size, which can be easily computed by a truth-table. Assuming that we know how to compute $L^\star_k$ with high probability on every input, we can compute $L^\star_{k+1}$ with high probability as follows. Simulate the learning algorithm with unknown concept $c = L^\star_{k+1}$. Answer membership queries using downward-self-reducibility and the procedure for $L^\star_k$ obtained by induction. The learner outputs a hypothesis h for $c = L^\star_{k+1}$ that is close to $L^\star_{k+1}$. Now use the self-correctibility of $L^\star$ together with c to obtain an algorithm that computes $L^\star_{k+1}$ on every input with high probability. Observe that each stage can be computed efficiently from our assumptions. After n stages, we obtain a randomized algorithm that computes $L^\star$ on any input of size n, which completes the proof that $L^\star \in \mathsf{BPP}$. For completeness, we present the full proof of Theorem 13 in Appendix A.
The next corollary is immediate by taking $\mathcal{O}$ to be the empty language in the statement of Theorem 13.

Corollary 14. *Let* $\mathcal{C}$ *be any concept class and suppose that there exists an algorithm that* PAC *learns any* $c \in \mathcal{C}^s$ *under the uniform distribution using membership queries in time* $T(n, 1/\epsilon, \log 1/\delta, s)$*. Also, let* L_{PSPACE} *be the* PSPACE*-complete language given by Proposition 12. Then, at least one of the following conditions hold:*

(*i*) $L_{\mathsf{PSPACE}} \notin \mathcal{C}^s$*; or*

(*ii*) $L_{\mathsf{PSPACE}} \in \mathsf{BPTIME}(\mathsf{poly}(T(n, O(1), \log n, s)))$.

For instance, for efficient PAC learning algorithms we have the following:

Corollary 15. *Let* $\mathcal{C}$ *be any concept class and suppose that there exists a polynomial-time algorithm that* PAC *learns* $\mathcal{C}$ *under the uniform distribution using membership queries. Then, at least one of the following conditions hold:*

(*i*) $\mathsf{PSPACE} \nsubseteq \mathsf{P/poly}[\mathcal{C}]$*; or*

[1] We stress that the learner can ask both membership queries about the unknown concept and queries to oracle $\mathcal{O}$.

(ii) $\mathsf{PSPACE} \subseteq \mathsf{BPP}$.

Corollary 15 implies the original result of Fortnow and Klivans: if $\mathsf{PSPACE} \subseteq \mathsf{BPP}$, a simple padding argument gives $\mathsf{EXPSPACE} \subseteq \mathsf{BPEXP}$, and it is not hard to prove by diagonalization that $\mathsf{EXPSPACE}$ requires circuits of size $\Omega(2^n/n)$. Thus, under efficient PAC learnability of $\mathcal{C}$, it follows that either $\mathsf{PSPACE} \nsubseteq \mathsf{P/poly}[\mathcal{C}]$ or BPEXP requires circuits of size $\Omega(2^n/n)$. In particular, this implies that $\mathsf{BPEXP} \nsubseteq \mathsf{P/poly}[\mathcal{C}]$.

Note that the second condition in the conclusion above does not depend on the class $\mathcal{C}$ that appears in the hypothesis. We observe next that removing this "or" condition from the conclusion of Corollary 15 would give us an unconditional proof that $\mathsf{PSPACE} \neq \mathsf{BPP}$. To see this, suppose the following result is valid:

If $\mathcal{C}$ is PAC-learnable in polynomial-time then $\mathsf{PSPACE} \nsubseteq \mathsf{P/poly}[\mathcal{C}]$ $(\star)$

Let $\mathcal{C}$ be the class of Boolean circuits, i.e., $\mathsf{P/poly}[\mathcal{C}] = \mathsf{P/poly}$. Let $\mathsf{P/poly\text{-}PAC\text{-}learnable}$ denote that $\mathcal{C}$ is PAC learnable in polynomial time. We prove that both $\mathsf{P/poly\text{-}PAC\text{-}learnable}$ and its negation $\neg\mathsf{P/poly\text{-}PAC\text{-}learnable}$ imply $\mathsf{BPP} \neq \mathsf{PSPACE}$. First, if $\mathsf{P/poly\text{-}PAC\text{-}learnable}$ then it follows from $(\star)$ that $\mathsf{PSPACE} \nsubseteq \mathsf{P/poly}$. Since $\mathsf{BPP} \subseteq \mathsf{P/poly}$ (Adleman [Adl78]), this implies $\mathsf{BPP} \neq \mathsf{PSPACE}$. On the other hand, suppose we have $\neg\mathsf{P/poly\text{-}PAC\text{-}learnable}$. To show that that $\mathsf{BPP} \neq \mathsf{PSPACE}$, it is sufficient to prove that if $\mathsf{BPP} = \mathsf{PSPACE}$ then $\mathcal{C}$ is efficiently PAC-learnable. Using $\mathsf{PSPACE} \subseteq \mathsf{BPP}$, we can find a efficiently find a hypothesis consistent with the labeled examples with high probability and a well known result (Occam's Razor, see Proposition 16 below) now implies PAC-learning.

6 A new way to prove Karp-Lipton Collapse Theorems

In this section we show that Proposition 16 (Occam's Razor) together with Theorem 13 (PAC learning yields circuit lower bound) can be used to prove Karp-Lipton style collapse theorems (Karp and Lipton [KL80]). Recall that these theorems state that if some circuit lower bound does not hold then there is a unexpected collapse involving uniform complexity classes. These results are usually stated with respect to $\mathsf{P/poly}$. The most famous Karp-Lipton Theorem says that if $\mathsf{NP} \subseteq \mathsf{P/poly}$ then $\mathsf{PH} = \Sigma_2^p$, i.e., the polynomial time hierarchy collapses to its second level. Similar theorems are known for different complexity classes. To prove more refined results, we use $\mathsf{SIZE}(l(n))$ to denote the class of languages with circuits of size $O(l(n))$. For concreteness, we give a proof for PSPACE. However, it is clear that the same argument works for any complexity class containing complete problems that are both downward-self-reducible and self-correctible, such as $\#\mathsf{P}$. To the best of our knowledge, these are the first proofs of these theorems that rely on Occam's Razor.

We start by stating the Occam's Razor technique [BEHW87].

Proposition 16 (Occam's Razor Principle). *Let $\mathcal{C}$ be any representation class and $s : \mathbb{N} \to \mathbb{N}$ be an arbitrary constructive function. Suppose there exists an algorithm B that, given any set of $m \geq \frac{1}{\epsilon}\left(s + \log \frac{1}{\delta}\right)$ uniformly distributed random examples labelled according to some unknown concept $c \in \mathcal{C}_n^s$, outputs a hypothesis $h \in \mathcal{C}^s$ that is consistent with this set of examples. Then B is a* PAC *learning algorithm for $\mathcal{C}$. In other words, the hypothesis h outputted by B is ϵ-close to c with probability at least $1 - \delta$.*

Proposition 17. *Let $\mathcal{C}$ be an arbitrary concept class and $s' : \mathbb{N} \to \mathbb{N}$ be any constructive function with $s'(n) \geq n$. If $L_{\mathsf{PSPACE}} \in \mathcal{C}^{s'(n)}$ then $L_{\mathsf{PSPACE}} \in \mathsf{BPTIME}(\mathsf{poly}(s'(n)))^{\mathsf{NP}}$.*

Proof. For any concept class $\mathcal{C}$, there exists an algorithm $\mathcal{B}$ that uses an NP oracle and is able to learn any concept $c \in \mathcal{C}$ in time $T(n, 1/\epsilon, \log 1/\delta, \mathsf{size}(c)) = \mathsf{poly}(n, 1/\epsilon, \log 1/\delta, \mathsf{size}(c))$. This algorithm simply draws $m = \frac{1}{\epsilon}\left(\mathsf{size}(c) + \log \frac{1}{\delta}\right)$ random examples labelled according to c and uses its NP oracle together with a standard search to decision reduction to find a hypothesis

$h \in \mathcal{C}^{\mathsf{size}(c)}$ that is consistent with all examples. By Occam's Razor (Proposition 16), $\mathcal{B}$ is a PAC learning algorithm for $\mathcal{C}$.

Let $L^\star = L_{\mathsf{PSPACE}}$, $\mathcal{O} = \mathsf{NP}$, and $s = s'(n)$ in the statement of Theorem 13. It follows that either $L_{\mathsf{PSPACE}} \notin \mathcal{C}^{s'(n)}$ or $L_{\mathsf{PSPACE}} \in \mathsf{BPTIME}(\mathsf{poly}(T(n, O(1), \log n, s')))^{\mathsf{NP}} = \mathsf{BPTIME}(\mathsf{poly}(s'(n)))^{\mathsf{NP}}$. The result then follows from the assumption that $L_{\mathsf{PSPACE}} \in \mathcal{C}^{s'(n)}$. $\square$

Corollary 18. *If* $L_{\mathsf{PSPACE}} \in \mathsf{SIZE}(l(n))$ *then* $L_{\mathsf{PSPACE}} \in \mathsf{BPTIME}(\mathsf{poly}(l(n)))^{\mathsf{NP}}$.

Corollary 19. *If* $\mathsf{PSPACE} \subseteq \mathsf{P/poly}$ *then* $\mathsf{PSPACE} \subseteq \mathsf{BPP}^{\mathsf{NP}}$.

We remark that this is not a new result. It is known that if $\mathsf{PSPACE} \subseteq \mathsf{P/poly}$ then $\mathsf{PSPACE} \subseteq \mathsf{MA}$, and $\mathsf{MA} \subseteq \mathsf{ZPP}^{\mathsf{NP}} \subseteq \mathsf{BPP}^{\mathsf{NP}}$ (Goldreich and Zuckerman [GZ11]).

Following the terminology of Trevisan and Vadhan [TV07], one may interpret our results as a new way to prove "super Karp-Lipton" theorems for PSPACE. For instance, if there exists a polynomial-time learning algorithm for TC^0_2, it follows that $\mathsf{PSPACE} \subseteq \mathsf{TC}^0_2$ implies $\mathsf{PSPACE} = \mathsf{BPP}$.

7 SQ Learning Yields Circuit Lower Bounds

In this section we show that efficient SQ learning algorithms for a class $\mathcal{C}$ of circuits yield circuit lower bounds. We will first show that efficient CSQ algorithms yield explicit average case hard functions and then go on to obtain a non-constructive lower bound from an SQ learning algorithm.

7.1 Preliminaries

Definition 10 (Inner Product). *For any functions f, g mapping $\{-1,1\}^n$ into $\{-1,1\}$, we denote the inner product of f and g with respect to $\mathcal{U}_n$ by $\langle f, g\rangle$. The inner product with respect to the uniform distribution on $X \subseteq \{-1,1\}^n$, $\mathcal{U}_X$, is denoted by $\langle f, g\rangle_X$.*

Definition 11 (Hamming Distance). *For any two Boolean functions f, g mapping $\{-1,1\}^n$ into $\{-1,1\}$, the hamming distance between f and g denoted by $dist(f,g) = \frac{1}{2^n}|\{x \in \{-1,1\}^n \mid f(x) \neq g(x)\}|$. Observe that $dist(f,g) = \frac{1}{2}(1 - |\langle f, g\rangle|)$. Hamming distance is a metric on the space of Boolean functions on $\{-1,1\}^n$.*

We will now define discrepancy of a bounded function class [Cha00, Mat99, PA95]. The definition we present here (and used in [CKK12]) is a natural generalization of the standard definition of discrepancy to classes of bounded functions.

Definition 12 (Discrepancy of a Class of Bounded Functions). *Let $\mathcal{C}$ be a class of bounded functions mapping a finite set X into $[-1,1]$ and let $\chi : X \to \{-1,1\}$ be a coloring of X. The discrepancy of χ with respect to a function $c \in \mathcal{C}$ is defined as $\chi(c) = \sum_{x:c(x)\geq 0} \chi(x) \cdot c(x)$. The discrepancy of χ with respect to the class $\mathcal{C}$ on X is defined as $disc[X, \mathcal{C}](\chi) = \max_{c\in\mathcal{C}} |\chi(c)|$.*

A uniformly random coloring is, not surprisingly, a low discrepancy coloring. The proof is a direct application of the Chernoff-Hoeffding Bounds. Further, this procedure to construct a low-discrepancy coloring can be derandomized [Siv02].

Lemma 20 (Deterministic Construction of Low Discrepancy Coloring [Siv02]). *Let $\mathcal{C}$ be a class of bounded functions on X with $|\mathcal{C}| = m$. There exists a deterministic algorithm running in time $\mathsf{poly}(m, |X|)$ that produces a coloring with discrepancy at most $\sqrt{4|X|\log 4m}$ for $\mathcal{C}$.*

There is a simple connection between a low discrepancy coloring χ for $\mathcal{C}$ on X and average case hardness of χ for $\mathcal{C}$ observed in [CKK12].

Proposition 21 (Low Discrepancy $\Rightarrow$ Average Case Hard Function). *Let $\mathcal{C}$ be a class of bounded functions mapping X into $[-1,1]$. Let $-C = \{-c : c \in C\}$ denote the class of all negated functions from $\mathcal{C}$. If $\chi : X \to \{-1,1\}$ is a coloring of X with discrepancy at most $\epsilon|X|$ with respect to $\mathcal{C} \cup -\mathcal{C}$ then $|\langle \chi, c\rangle_{\mathcal{U}_X}| \le 2\epsilon$ for each $c \in \mathcal{C}$ on X.*

Proof. Let $c \in \mathcal{C}$. Since χ has discrepancy at most $\epsilon|X|$ with respect to c and $-c$, we have: $|\sum_{\substack{x\in X\\ c(x)\ge 0}} \chi(x)\cdot c(x)| \le \epsilon|X|$ and $|\sum_{\substack{x\in X\\ c(x)\le 0}} \chi(x)\cdot c(x)| \le \epsilon|X|$. Thus, $|\langle \chi, c\rangle_{\mathcal{U}_X}| = |\mathbb{E}_{x\sim\mathcal{U}_X}[\chi(x)\cdot c(x)]|$
$\le \frac{1}{|X|}\left(|\sum_{\substack{x\in X\\ c(x)\ge 0}} \chi(x)\cdot c(x)| + |\sum_{\substack{x\in X\\ c(x)\le 0}} \chi(x)\cdot c(x)|\right) \le \frac{1}{|X|}(\epsilon|X| + \epsilon|X|) = 2\epsilon.$

□

7.2 CSQ Learning Yields Lower Bounds

To show that CSQ learning algorithms yield circuit lower bounds, we use a learning algorithm $\mathcal{A}$ for $\mathcal{C}$, to construct a small set of functions $\mathbb{G}$ such that each function in $\mathcal{C}$ is non-trivially correlated with some function in $\mathbb{G}$. This construction is well known and has been employed in other contexts [Fel08]:

Lemma 22 (Small Weakly Correlating set from CSQ Algorithm). *Let $\mathcal{C}$ be a representation class of Boolean functions on $\{-1,1\}^n$. Suppose for some ϵ, τ such that $\frac{1}{2} > \epsilon \ge \tau > 0$ and $\tau \le 1-2\epsilon$, $\mathcal{C}^s$ is learnable on the uniform distribution in the CSQ model by an algorithm $\mathcal{A}$ running in time $T = T(n, \frac{1}{\epsilon}, \frac{1}{\tau}, s)$ while making at most $Q = Q(n, \frac{1}{\epsilon}, \frac{1}{\tau}, s)$ correlational queries of tolerance τ. Then, there exists a set $\mathbb{G}$ of at most $Q+1$ functions mapping $\{-1,1\}^n$ into $[-1,1]$ such that, for every $c \in \mathcal{C}^s$, there exist a $g \in \mathbb{G}$ such that $|\langle g, c\rangle| \ge \tau$. Moreover, such a set $\mathbb{G}$ can be recovered by an algorithm running in deterministic time T.*

Proof. We will simulate the CSQ oracle for $\mathcal{A}$ and simulate the learning algorithm to construct the set of functions $\mathbb{G}$.

Simulate the CSQ algorithm $\mathcal{A}$ for $\mathcal{C}^s$. Each time the algorithm makes a correlational query to the CSQ oracle, return 0. Stop the simulation if $\mathcal{A}$ runs for T steps or makes Q queries. Let $g_1, g_2, \cdots, g_k$ be the queries made by the algorithm we stop simulating $\mathcal{A}$. Then, $k \le Q$. If $\mathcal{A}$ returns a hypothesis h. Let $\mathbb{G} = \{g_i \mid 1 \le i \le k\} \cup \{h\}$. If $\mathcal{A}$ doesn't return any hypothesis, there must not be any function in $\mathcal{C}^s$ consistent with our answers for the CSQs which immediately yields that for every $c \in \mathcal{C}^s$, there exists a g_i for $i \in [k]$ such that $|\langle c, g_i\rangle| > \tau$. Now suppose $\mathcal{A}$ returns an hypothesis h.

We now verify that $\mathbb{G}$ satisfies the required conditions stated in the theorem statement. Let $c \in \mathcal{C}^s$. One of the following two conditions has to be true:

1. $|\langle c, g_i\rangle| \le \tau$ for each $1 \le i \le k$.
   In this case, observe that the answers returned to the algorithm while simulating the CSQ oracle are consistent with the target function c within the tolerance bound of τ. Thus, $\Pr_{x\sim\mathcal{U}}[h(x) \ne c(x)] \le \epsilon$ or $|\langle c, h\rangle| \ge 1-2\epsilon \ge \tau$.
2. There exists a j, $1 \le j \le k$ such that $|\langle g_j, c\rangle| \ge \tau$. In this case we are immediately done since $g_j \in \mathbb{G}$.

□

We now show that CSQ learning algorithms yield circuit lower bounds.

Theorem 23 (CSQ Learning Yields Circuit Lower Bounds). *Let $\mathcal{C}$ be a representation class of Boolean functions on $\{-1,1\}^n$. Let ϵ, τ be any parameters satisfying $\frac{1}{2} > \epsilon$ and $\tau \le \min\{\epsilon, 1-2\epsilon\}$. Suppose there exists an algorithm $\mathcal{A}$ that runs in time $T = T(n, \frac{1}{\epsilon}, \frac{1}{\tau}, s)$ that learns $\mathcal{C}^s$ on the uniform distribution in the CSQ model to accuracy $1-\epsilon$ by at most*

$Q = Q(n, \frac{1}{\epsilon}, \frac{1}{\tau}, s) \leq \mathsf{poly}(\tau) \cdot 2^{2^n}$ queries, each of tolerance τ. Then, there exists a Boolean function (family) $f \in \mathsf{DTIME}(T + \mathsf{poly}(Q, \frac{1}{\tau}))$ such that for every $c \in \mathcal{C}^s$, $\Pr_{x\sim\mathcal{U}}[f(x) \neq c(x)] \geq \frac{\tau}{4}$.

The intuitive idea is that we can use Lemma 22 to construct the set $\mathbb{G}$ of size $\leq Q + 1$. Running deterministic discrepancy minimization algorithm (Lemma 20) on $\mathbb{G} \cup -\mathbb{G}$ yields a function f that has low correlation with every function in $\mathbb{G}$ (using Proposition 21). The fact that every function in $\mathcal{C}^s$ is non-trivially correlated with some function in $\mathbb{G}$ is then invoked to argue that f should be far from $\mathcal{C}^s$.

Remark 1. *The theorem holds for any $\epsilon < 1/2$ and thus even a weak learning algorithm for $\mathcal{C}^s$ that uses only CSQs yields lower bounds against $\mathcal{C}^s$.*

Algorithm 2 Hard Function f that uses CSQ learner $\mathcal{A}$ as a subroutine

```
    Input: $x \in \{-1,1\}^n$ Output: A value in $\{-1,1\}$.
1:  Use learner $\mathcal{A}$ to obtain the set $\mathbb{G}$ of size at most $Q + 1$ of weakly correlating
    functions for $\mathcal{C}^s$.
2:  Let $\{-1,1\}^n$ be partitioned into consecutive blocks in lexicographic order
    $E_1, E_2, \cdots, E_k$ each of size $t$ (the last block may be of smaller size).
3:  Determine $j$ such that $x \in E_j$. Recover all the points in $E_j$.
4:  Run deterministic discrepancy minimization on the class $\mathbb{G} \cup -\mathbb{G}$ and domain $E_j$ to
    obtain a function $f_j : E_j \to \{-1,1\}$.
5:  Return $f_j(x)$.
```

Proof. We need to describe a procedure to compute f using blackbox access to the CSQ learning algorithm $\mathcal{A}$. We will show that f is far from $\mathcal{C}^s$ and $f \in \mathsf{DTIME}(T + \mathsf{poly}(Q))$. Let x be the input to f. First, construct $\mathbb{G}$, the set of weakly correlating functions by simulating the learning algorithm for $\mathcal{C}^s$ using Lemma 22. Notice that since the CSQ algorithm doesn't use any randomness of its own, the procedure produces a fixed set $\mathbb{G}$ in any run of the algorithm. Let $\{-1,1\}^n$ into consecutive blocks $E_1, E_2, \cdots E_k$ each of size $t = \lceil \frac{64 \log 2|\mathbb{G}|}{\tau^2} \rceil \leq \lceil \frac{64 \log 4Q}{\tau^2} \rceil$ (the last block E_k may be smaller). f first finds out j such that $x \in E_j$. It then runs the deterministic discrepancy minimization algorithm (Lemma 20) on the class $\mathbb{G} \cup -\mathbb{G}$ and domain E_j. Suppose $f_j : E_j \to \{-1,1\}^n$ is the function returned by the algorithm. f outputs $f_j(x)$. By using Proposition 21, we observe that for every $g \in \mathbb{G}$ and for every $j < k$, $|\langle f_j, g\rangle_{E_j}| \leq 2\sqrt{\frac{4 \log 4Q}{t}} \leq \tau/4$.

Fix any $j \in [k]$. Notice that for any $x \in E_j$, the algorithm runs the discrepancy minimization on the same class $\mathbb{G} \cup -\mathbb{G}$ and on the same domain E_j, thus constructing the same function f_j each time. Thus, for each $x \in \{-1,1\}^n$, $f(x) = f_j(x)$ whenever $x \in E_j$.

Thus, $|\langle f_j, g\rangle_{E_j}| \leq \tau/4$. Now for each g,

$$|\langle f, g\rangle| = |\sum_{j=1}^{k} \frac{|E_j|}{2^n} \langle f_j, g\rangle_{E_j}| \leq \frac{\tau}{2} \cdot (\sum_{j=1}^{k-1} \frac{|E_j|}{2^n}) + 1 \cdot \frac{t}{2^n} < \frac{\tau}{2} + \frac{1}{2^n} \cdot \lceil \frac{64 \log 4Q}{\tau^2} \rceil \leq \tau/2,$$

(since $Q \leq 2^{2^n} \cdot \mathsf{poly}(\tau)$).

To show the average case hardness of f for $\mathcal{C}^s$, fix any $c \in \mathcal{C}^s$. Let $g_c \in \mathbb{G}$ such that $|\langle c, g_c\rangle| \geq \tau$, but $|\langle f, g_c\rangle| \leq \frac{\tau}{2}$. Since by changing a single coordinate in the 2^n-dimensional vector representing function c we can only change the value of $\langle c, g_c\rangle$ by $\pm 2/2^n$, it must be the case that

c and f differ in at least a $\tau/4$ fraction of the inputs. In other words, $\Pr_{x\sim\mathcal{U}}[f(x) \neq c(x)] \geq \frac{\tau}{4}$, as desired. Finally, observe that the value of f at any $x \in \{-1,1\}^n$ can be evaluated in deterministic time $\mathsf{poly}(|\mathbb{G}|, \frac{1}{\tau}) = \mathsf{poly}(|Q|, \frac{1}{\tau})$. This completes the proof. □

Note that since the class of all parity functions requires $2^{\Omega(n)}$ SQs to learn [Kea98], we immediately obtain that if $\mathcal{C}$ is efficiently SQ learnable then $\mathcal{C}$ cannot compute some parity. Thus any efficient SQ learnability of a class $\mathcal{C}$ immediately yields a worst case lower bound. Such an argument can actually be extended to obtain even a weak average case lower bound. This is based on the characterization of CSQ learning by the SQ Dimension studied in [BFJ+94]. The SQ dimension of a class $\mathcal{C}$ on the uniform distribution is the largest number d such that there exist functions $c_1, c_2, \cdots c_d \in \mathcal{C}$, such that for every $i \neq j$, $|\langle c_i, c_j\rangle| \leq \frac{1}{d^3}$. Kearns et. al. characterized the query complexity of the best SQ algorithm that learns $\mathcal{C}$ on $\mathcal{U}$ to be within a polynomial factor of the SQ dimension of $\mathcal{C}$ on $\mathcal{U}$. The following proposition shows that efficient CSQ learnability of $\mathcal{C}$ by CSQs of tolerance τ implies that there is a parity which is $1 - 1/n^{\omega(1)}$-hard for $\mathsf{P/poly}[\mathcal{C}]$.

Compare this to Theorem 23 which shows that there is a function computable in super polynomial time and is $\tau/4$-hard for $\mathsf{P/poly}[\mathcal{C}]$ from the same assumption.

Lemma 24. *Suppose $\mathcal{C}^s$, a representation class of Boolean functions of size at most s, is learnable to an accuracy of $1/3$ by CSQs of tolerance τ (lower bounded by an inverse polynomial in n) in time $T(n,s)$ upper bounded by some polynomial in n, s where c is the target function. Then, there exists a parity χ_S, $|S| = O(\frac{\log s}{\log n})$, such that $\Pr_{x\sim\mathcal{U}}[\chi_S(x) \neq c(x)] \geq \frac{1}{n^{|S|}}$ for every $c \in \mathcal{C}^s$. Consequently for every $k = \omega(1)$, there exists a parity that cannot be computed on at least $\frac{1}{n^k}$ fraction of the inputs by any function in $\mathsf{P/poly}[\mathcal{C}]$*

Proof. Suppose $\mathcal{C}^s$ is learnable by a CSQ algorithm to accuracy of $1/3$ in time $T(n,s)$. Then the SQ dimension of $\mathcal{C}^s$ on the uniform distribution is bounded above by $T(n,s)$.

For some k, that we will fix later, consider the set of all parities over subsets of at most k variables out of n variables.

Suppose for each $T \subseteq [n]$ such that $|T| \leq k$, there exists a $c_T \in \mathsf{P/poly}[\mathcal{C}]$ such that $|\langle c_T, \chi_T\rangle| > 1 - \frac{1}{n^k}$. Consider $|\langle c_T, c_R\rangle| = 2(1 - dist(c_T, c_R))$ for some T, R such that $|T|, |R| \leq k$. Then, by triangle inequality (for Hamming distance), $dist(\chi_T, \chi_R) \leq dist(\chi_T, c_T) + dist(c_T, c_R) + dist(c_R, \chi_R)$. Using orthogonality of distinct parities, $dist(c_T, c_R) \geq 1 - 2 \cdot (\frac{1}{2} - \frac{1}{2n^k}) = \frac{1}{n^k}$ yielding, $|\langle c_T, c_R\rangle| < 1 - (1 - \frac{1}{n^k}) = \frac{1}{n^k}$. This yields that the set $\{c_T \mid |T| \leq k\}$ forms a set of n^k functions that satisfy $|\langle c_T, c_R\rangle| \leq \frac{1}{n^k}$ for every $T \neq R$ of size k, or that the SQ dimension of $\mathsf{P/poly}[\mathcal{C}]$ is at least n^k.

Now, choose a large enough constant k, such that $n^k > T(n,s)$ to obtain a contradiction yielding that some parity on k variables cannot be correlated with any $c \in \mathcal{C}^s$ by more than $1 - \frac{1}{n^k}$. □

Remark 2. *Observe that from the proof above, one can only obtain the conclusion that some parity differs from every $c \in \mathsf{P/poly}[\mathcal{C}]$ on a negligible fraction (inverse super-polynomial) of inputs.*

7.3 SQ Learning Yields Circuit Lower Bounds

In this section we prove that it is possible to obtain strong average-case hardness results from the existence of an algorithm that learns using statistical queries. Formally, we will show that if we can learn a class $\mathcal{C}$ in the statistical query model, then either there is an explicit function f that is average-case hard for $\mathcal{C}$, or $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P}$.

We have seen in the proof of Theorem 23 that a learning algorithm that uses only correlational queries yields an explicit average case hard function. Since the target independent queries do not depend on the target function, a deterministic algorithm to compute such queries will immediately give us the same conclusion starting from SQ algorithms. However, answering a target independent query involves estimating the expectation of a function specified by some Boolean circuit, and no efficient deterministic algorithm is known for this task. We explain why our proof technique cannot accommodate randomized learning algorithms in Section 8.

Here, we show that we can indeed prove that SQ algorithms yield lower bound, but our proof here will not be constructive as in the case of CSQ algorithms. Each target independent query asks the oracle an estimate for the expectation of a function, given by some circuit. Thus a $\#\mathsf{P}$ oracle is enough to answer such queries exactly. (Recall that $\#\mathsf{P}$ is the class of counting problems associated with NP-relations.)

The main result of this section follows from Proposition 6 and a slight modification of the argument used in the proof of Theorem 23.

Theorem 25 (SQ Learning Yields Circuit Lower Bounds)**.** *Let $\mathcal{C}$ be a representation class of Boolean functions on $\{-1,1\}^n$. Let ϵ, τ satisfy $\epsilon < 1/2$ and $\tau \le \min\{\epsilon, 1-2\epsilon\}$. Suppose there exists an algorithm $\mathcal{A}$ that runs in time $T = T(n, \frac{1}{\epsilon}, \frac{1}{\tau}, s)$ that learns $\mathcal{C}^s$ over the uniform distribution to an accuracy of $1-\epsilon$ using at most $Q = Q(n, \frac{1}{\epsilon}, \frac{1}{\tau}, s)$ SQs, each of tolerance τ. Then, at least one of the following conditions hold:*

- *There exists a function (family) $f \in \mathsf{DTIME}(\mathsf{poly}(T, Q, 1/\tau))$ such that for every $c \in \mathcal{C}^s$, we have*
$$\Pr[c(x) \neq f(x)] \geq \tau/4,$$
- *or $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P}$.*

Proof. If $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P}$, we are done. Assume $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{P}$. Thus, we have an efficient deterministic procedure to answer target independent queries. We now follow the proof of Theorem 23, relying on Proposition 6. Observe that if we show that the conclusion of Lemma 22 follows even from an SQ (instead of CSQ) algorithm, we are done, as the rest of the proof of Theorem 23 does not use $\mathcal{A}$. To show the conclusion of Lemma 22 starting from an SQ algorithm, we describe how to obtain a set $\mathbb{G}$ of almost orthogonal functions. As in the original proof, to every correlational query asked by $\mathcal{A}$, f replies with 0. We use the efficient algorithm granted by our assumption to answer target independent queries (each one is specified by some $\mathsf{poly}(s)$ size circuit) within τ. Let $g_1, g_2, \cdots, g_k$ be the correlational queries made by algorithm $\mathcal{A}$ before it returns a hypothesis h, and define as before $\mathbb{G} = \{g_i \mid 1 \le i \le k\} \cup \{h\}$. The same argument used in the proof of Lemma 22 applies to $\mathbb{G}$, and the result follows. □

Remark 3. *By using more powerful results it is possible to replace $\#\mathsf{P}$ by smaller complexity classes. In other words, the same argument works for any complexity class that allows us to answer the target independent queries. We omit the details.*

It is known that if $\mathsf{P} = \mathsf{NP}$ then EXP requires circuits of exponential size[2]. Therefore, we have unconditionally that either $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P}$ or EXP requires large circuits. Comparing this result with our theorem, we observe that under efficient learnability Theorem 25 implies that either $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P}$ or P contains functions that are average-case hard for $\mathcal{C}$.

8 Directions for Further Work

In this paper we showed that the existence of deterministic mistake-bounded or exact learning algorithms yield lower bounds as long as the mistake-bound (or queries, respectively) is less

[2]If $\mathsf{P} = \mathsf{NP}$ then PH collapses to P. Using a padding argument, it follows that the exponential time hierarchy collapses to EXP, which implies that this class contains functions of exponential circuit complexity by Kannan's theorem.

than the trivial bound of 2^n. Further, our proofs for these classes work even when the learning algorithms are allowed access to arbitrary oracles. Thus, obtaining a new learning algorithm for a circuit class can be a method to prove new lower bounds against it. An analogous approach was used by Williams [Wil10, Wil11] to obtain new circuit lower bounds from improved satisfiability algorithms.

Note, however, that our techniques do not yield an explicit lower bound starting from *randomized* learning algorithms. In order to construct a hard function, we must simulate a learning algorithm as a subroutine. If the behavior of the learning algorithm (on a fixed input) is not deterministic (due to say the learning algorithm's internal randomness) then our simulation is not fixed and so may give different output values starting from the same input. Thus, such learning algorithms will not yield a function.

This is also the underlying difficulty in simulating a general SQ (as opposed to CSQ) algorithm, as the SQ algorithm may ask for an estimate to $\mathbb{E}_x[g(x)]$ for an arbitrary (polynomial-time computable) predicate. In the CSQ model, we can appeal to Lemma 15 and deterministically obtain a fixed set of approximating functions.

We show that PAC learning algorithms yield a conditional lower bound (depending on whether PSPACE computations can be sped up by the use of randomness or not). On the other hand, Fortnow and Klivans [FK09] show that efficient PAC-learnability of a class $\mathcal{C}$ yields that BPEXP does not have polynomial size circuits from $\mathcal{C}$. Thus, another open question is to extend this line of work and obtain the same conclusion involving $\mathsf{BPSUBEXP}$ instead of BPEXP. As explained above, our "diagonalization" trick to prove lower bounds breaks down in this case, as these algorithms use internal randomness.

Acknowledgements

Igor would like to thank Rocco Servedio for encouraging him to work on the relation between learning and circuit lower bounds.

References

[Adl78] L. Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science*, pages 75–83. IEEE, 1978.

[Ang88] D. Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[BEHW87] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Inf. Process. Lett.*, 24(6):377–380, 1987.

[BF02] N.H. Bshouty and V. Feldman. On using extended statistical queries to avoid membership queries. *The Journal of Machine Learning Research*, 2:359–395, 2002.

[BFJ+94] Avrim Blum, Merrick L. Furst, Jeffrey C. Jackson, Michael J. Kearns, Yishay Mansour, and Steven Rudich. Weakly learning dnf and characterizing statistical query learning using fourier analysis. In *STOC*, pages 253–262, 1994.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[Cha00] Bernard Chazelle. *The discrepancy method: randomness and complexity.* Cambridge University Press, New York, NY, USA, 2000.

[CKK12] E. Chattopadhyay, A. Klivans, and P. Kothari. An explicit VC-theorem for low-degree polynomials. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *APPROX-RANDOM*, volume 7408 of *Lecture Notes in Computer Science*, pages 495–504. Springer, 2012.

[Fel08] Vitaly Feldman. Evolvability from learning algorithms. In *STOC*, pages 619–628, 2008.

[FK09] Lance Fortnow and Adam R. Klivans. Efficient learning algorithms yield circuit lower bounds. *J. Comput. Syst. Sci.*, 75(1):27–36, 2009.

[GH11] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In Rafail Ostrovsky, editor, *FOCS*, pages 107–109. IEEE, 2011.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Inf. Process. Lett.*, 43(4):169–174, 1992.

[GZ11] Oded Goldreich and David Zuckerman. Another proof that BPP $\subseteq$ PH (and more). In *Studies in Complexity and Cryptography*, pages 40–53. 2011.

[HH11] R. Harkins and J. Hitchcock. Exact learning algorithms, betting games, and circuit lower bounds. *Automata, Languages and Programming*, pages 416–423, 2011.

[IW96] Russell Impagliazzo and Avi Wigderson. P = BPP unless E has sub-exponential circuits: Derandomizing the XOR lemma (preliminary version). In *Proceedings of the 29th STOC*, pages 220–229. ACM Press, 1996.

[Kab13] Valentine Kabanets. Private communication, 2013.

[Kea98] Michael J. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998. Prelim. ver. in *Proc. of STOC'93*.

[KL80] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *STOC*, pages 302–309, 1980.

[KS09] Adam R. Klivans and Alexander A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. *J. Comput. Syst. Sci.*, 75(1):2–12, 2009.

[KV94a] M. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *JACM: Journal of the ACM*, 41, 1994.

[KV94b] Michael J. Kearns and Umesh V. Vazirani. *An introduction to computational learning theory.* MIT Press, Cambridge, MA, USA, 1994.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. In *Machine Learning*, pages 285–318, 1988.

[Mat99] Jiri Matousek. *Geometric Discrepancy: An Illustrated Guide (Algorithms and Combinatorics).* Springer, 1 edition, 1999.

[PA95] Janos Pach and Pankaj Agrawal. *Combinatorial Geometry.* Wiley-Interscience, October 1995.

[Siv02] D. Sivakumar. Algorithmic derandomization via complexity theory. In *IEEE Conference on Computational Complexity*, 2002.

[TV07] Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[Vad12] Salil P. Vadhan. Personal communication, 2012.

[Val84] Leslie G. Valiant. A theory of the learnable. In *STOC*, pages 436–445, 1984.

[Val09] Leslie G. Valiant. Evolvability. *J. ACM*, 56(1), 2009.

[Wil10] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 231–240. ACM, 2010.

[Wil11] R. Williams. Non-uniform acc circuit lower bounds. In *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*, pages 115–125. IEEE, 2011.

A Proof that PAC Learning Yields Circuit Lower Bounds

Here we provide the complete proof of Theorem 13. We state it again for convenience.

Theorem (Theorem 13)**.** *Let $\mathcal{C}$ be any concept class and suppose that there exists an algorithm that* PAC *learns $\mathcal{C}$ under the uniform distribution using membership queries when given access to an oracle $\mathcal{O}$ in time $T(n, 1/\epsilon, \log 1/\delta, \mathsf{size}(c))$. Let $L^\star$ be a language that is both downward-self-reducible and $\alpha(n)$-self-correctible. Then, for any constructive function $s : \mathbb{N} \to \mathbb{N}$, at least one of the following conditions hold:*

(i) $L^\star \notin \mathcal{C}^s$*; or*

(ii) $L^\star \in \mathsf{BPTIME}(\mathsf{poly}(T(n, 1/\alpha(n), \log n, s)))^{\mathcal{O}}$.

Proof. If $L^\star \notin \mathcal{C}^s$ then there is nothing to prove. Assume therefore that $L^\star \in \mathcal{C}^s$. In other words, there exists a constant n_0 such that, for every $n > n_0$, there is a concept $c_n \in \mathcal{C}_n^s$ such that $L_n^\star = c_n$. Let $d(n)$ be a non-decreasing polynomial that upper bounds the number of downward queries necessary to compute $L^\star$ on any input of size n given access to a routine that computes $L^\star$ on inputs of size $n-1$. Since $L^\star$ is self-correctible, there exists an efficient reduction such that, if we can compute $L_n^\star$ correctly on at least a $(1-\alpha(n))$-fraction of the inputs, then we can compute it correctly on every input of size n with probability at least $2/3$. Finally, let $\mathsf{Learner}$ be an algorithm that, when given access to oracle $\mathcal{O}$, is able to learn $\mathcal{C}$ in time $T(n, 1/\epsilon, \log 1/\delta, \mathsf{size}(c))$, where $\mathsf{size}(c)$ is an upper bound on the size of the unknown concept.

We need to prove that $L^\star \in \mathsf{BPTIME}(\mathsf{poly}(T(n, 1/\alpha(n), \log n, s)))^{\mathcal{O}}$. Given an input x of size n, we use $\mathsf{Learner}$ and the special properties of language $L^\star$ to "learn" how to compute $L^\star$ on every instance of size at most n. More specifically, for any integer k, given a procedure A_k that decides $L^\star$ on any instance of size k (with high probability), we use $\mathsf{Learner}$ together with the downward-self-reducibility of $L^\star$ and the fact that this language is self-correctible to obtain a procedure A_{k+1} that decides $L^\star$ on any instance of size $k+1$ (with high probability).

The execution of the algorithm for $L^\star$ proceed as follows. First, it starts with a procedure A_{n_0} that can be implemented by a lookup-table algorithm (recall that n_0 is a constant). Now we explain in more details how to go from, say, A_{n_0} to A_{n_0+1}. We simulate $\mathsf{Learner}$ pretending that the unknown concept is $c_{n_0+1} = L^\star_{n_0+1}$. If at some point the learning algorithm queries the value of $c_{n_0+1}(w)$ on some input w of size n_0+1, we use A_{n_0} together with the downward-self-reducibility property of $L^\star$ to provide an appropriate answer. If $\mathsf{Learner}$ queries its oracle $\mathcal{O}$, we provide the answer using our own oracle $\mathcal{O}$. After finishing its computation, the learning algorithm outputs a deterministic hypothesis h_{n_0+1} that is ϵ-close to c_{n_0+1} with high probability. We use the fact that $L^\star$ is self-correctible to obtain from h_{n_0+1} a procedure $\widetilde{A}_{n_0+1}$ that is correct on every input of size n_0+1 with probability at least $2/3$. Finally, using standard amplification techniques, it is possible to get from $\widetilde{A}_{n_0+1}$ a procedure A_{n_0+1} that is correct on every input with high probability. By repeating this process at most n stages, we obtain a procedure A_n and output $A_n(x)$. Let $\mathcal{A}$ be the algorithm that runs as described. The formal description of $\mathcal{A}$ is presented next.

First we argue that $\mathcal{A}$ computes $L^\star(x)$ correctly with high probability, then we upper bound its running time.

Claim 2. *For any input x, $\mathcal{A}$ outputs $L^\star(x)$ with probability at least* $2/3$.

Proof. Note that, for each stage k, $\mathcal{A}$ fails to obtain a good routine A_k only if:

- At least one the at most $t_n \cdot d(n)$ downward queries answered by A_{k-1} is incorrect. It follows by a union bound that this happens with probability at most $t_n \cdot d(n) \cdot \gamma = 1/20n$.
- Algorithm $\mathsf{Learner}$ does not output a good hypothesis. This also happens with probability at most $\delta = 1/20n$.

Algorithm 3 Description of Algorithm $\mathcal{A}$ that computes a hard function using PAC learner

```
Input: A string $x$ of size $n$ (and oracle access to $\mathcal{O}$).
Output: The value $L^\star(x)$ (with high probability).
1: Start with a "lookup-table" routine $A_{n_0}$ that computes $L$ correctly on all inputs of size $n_0$.
2: for $k = n_0 + 1$ to $n$ do
3:     Run Learner with parameters $k$, $\epsilon = \alpha(n)$, $\delta = 1/20n$, and $\mathsf{size}(c) = s(k)$ (here we use the fact that $s(\cdot)$ is constructible). Whenever Learner asks for the value $c_k(w)$ of some example $w$ of size $k$, use routine $A_{k-1}$ and the downward self-reducibility of $L^\star$ to compute a guess for $c_k(x) = L(x)$. Since $\mathcal{A}$ has oracle access to $\mathcal{O}$, any query to this oracle made by Learner can also be answered efficiently. When the learning algorithm finishes its computation, it outputs with probability at least $1 - \delta$ a deterministic hypothesis $h_k$ that is $\epsilon$-close to $c_k$ (note that $h_k$ does not have access to $\mathcal{O}$).
4:     Since $\epsilon = \alpha(n)$ and $L^\star$ is $\alpha(n)$-self-correctible, $\mathcal{A}$ uses $h_k$ and the self-correctibility of $L^\star$ to get a routine $\tilde{A}_k$ such that for any input $w$ of size $k$, $\tilde{A}_k(w) = L^\star(w)$ with probability at least 2/3. By running $\tilde{A}_k$ at most $O(\log 1/\gamma)$ times and taking a majority vote, it follows from standard Chernoff bounds that we obtain a routine $A_k$ that is incorrect on any input with probability at most $\gamma$. We set $\gamma = 1/(20 t_n d(n) n)$, where $t_n = T(n, 1/\alpha(n), \log 20n, s)$.
5: end for
6: return $A_n(x)$.
```

Overall, for each stage k, we fail to obtain a good algorithm A_k with probability no more than $1/10n$. Since there are at most n stages, A_n fails to compute $L^\star(x)$ with probability at most $1/10 + \gamma \leq 1/3$. □

Claim 3. *Given oracle access to $\mathcal{O}$, algorithm A runs in randomized time* $\mathsf{poly}(T(n, 1/\alpha(n), \log n, s)))$.

Proof. First we upper bound the running time of each procedure A_k. Observe that A_k uses $\tilde{A}_k$, which is obtained from h_k. Recall that the running time of h_k is bounded by the running time of Learner, which is at most $t(k, 1/\alpha(n), \log 20n, s(k)) \leq T(n, 1/\alpha(n), \log 20n, s) = t_n$, since both $s(\cdot)$ and $t(\cdot)$ are non-decreasing[3]. Further, to obtain $\tilde{A}_k$ we use the self-correctibility of $L^\star$, which is implemented by a polynomial-time reduction. In other words, $\tilde{A}_k$ runs in time $O(t_n^a)$ for some constant a. Finally, the amplification step that is used when we go from $\tilde{A}_k$ to A_k only needs to run $\tilde{A}_k$ for $O(\log 1/\gamma) = O(\log(20 t_n \cdot d(n) \cdot n))$ times, which implies that the overall time complexity of A_k, for any $1 \leq k \leq n$, is upper bounded by $O(t_n^a \cdot \log(20 t_n \cdot d(n) \cdot n)) = O(t_n^b)$ for some constant b (recall that $d(\cdot)$ is a polynomial).

Now we upper bound the overall running time of algorithm $\mathcal{A}$. Each stage k consists of simulating algorithm Learner for at most $t(k, 1/\alpha(n), \log 20n, s(k)) \leq t_n$ steps. In the worst-case, each step may involve a membership query to the unknown concept, which translates to at most $d(n)$ downward queries to A_{k-1}. Since A_{k-1} runs in time $O(t_n^b)$, the overall time complexity of each stage is at most $O(t_n \cdot d(n) \cdot t_n^b) = O(t_n^c)$ for some constant c. There are no more

[3]We have implicitly used the standard fact that any PAC learning algorithm can be efficiently converted into an algorithm that has a logarithmic dependence on $1/\delta$. A proof of this result can be found on Kearns and Vazirani [KV94b] textbook.

than n stages. It follows that, given oracle access to $\mathcal{O}$, algorithm $\mathcal{A}$ runs in randomized time $\mathsf{poly}(T(n, 1/\alpha(n), \log n, s)))$. $\square$

$\square$