Electronic Colloquium on Computational Complexity, Report No. 122 (2018)

Pseudo-derandomizing learning and approximation

Igor C. Oliveira* University of Oxford

Rahul Santhanam† University of Oxford

July 3, 2018

Abstract

We continue the study of *pseudo-deterministic algorithms* initiated by Gat and Goldwasser [GG11]. A pseudo-deterministic algorithm is a probabilistic algorithm which produces a fixed output with high probability. We explore pseudo-determinism in the settings of *learning* and *approximation*. Our goal is to simulate known randomized algorithms in these settings by pseudo-deterministic algorithms in a generic fashion – a goal we succinctly term *pseudo-derandomization*.

Learning. In the setting of learning with membership queries, we first show that randomized learning algorithms can be derandomized (resp. pseudo-derandomized) under the standard hardness assumption that E (resp. BPE) requires large Boolean circuits. Thus, despite the fact that learning is an algorithmic task that requires interaction with an oracle, standard hardness assumptions suffice to (pseudo-)derandomize it. We also *unconditionally* pseudo-derandomize any quasi-polynomial time learning algorithm for polynomial size circuits on infinitely many input lengths in sub-exponential time.

Next, we establish a generic connection between learning and derandomization in the reverse direction, by showing that deterministic (resp. pseudo-deterministic) learning algorithms for a concept class $\mathfrak{C}$ imply hitting sets against $\mathfrak{C}$ that are computable deterministically (resp. pseudo-deterministically). In particular, this suggests a new approach to constructing hitting set generators against $\mathcal{AC}^0[p]$ circuits by giving a deterministic learning algorithm for $\mathcal{AC}^0[p]$.

Approximation. Turning to approximation, we *unconditionally* pseudo-derandomize any polytime randomized approximation scheme for integer-valued functions infinitely often in sub-exponential time over any samplable distribution on inputs. As a corollary, we get that the $(0,1)$-Permanent has a fully pseudo-deterministic approximation scheme running in sub-exponential time infinitely often over any samplable distribution on inputs.

Finally, we investigate the notion of *approximate canonization* of Boolean circuits. We use a connection between pseudodeterministic learning and approximate canonization to show that if BPE does not have sub-exponential size circuits infinitely often, then there is a pseudo-deterministic approximate canonizer for $\mathcal{AC}^0[p]$ computable in quasi-polynomial time.

*igor.carboni.oliveira@cs.ox.ac.uk. Supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2014)/ERC Grant Agreement no. 615075.

†rahul.santhanam@cs.ox.ac.uk. Supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2014)/ERC Grant Agrement no. 615075.

ISSN 1433-8092

Contents

1 Introduction

1.1 Context and Motivation

Randomness is a powerful algorithmic resource, used widely in tasks such as cryptography, distributed computing, learning, sampling and approximation. Although it often makes algorithmic tasks more efficient, randomness comes with issues. It introduces *uncertainty* – running a randomized algorithm multiple times, we cannot always expect to get the same answer. Moreover, randomized algorithms assume access to a source of independent and unbiased random bits, and this assumption is not always justified in the physical world.

Ideally, we would like to perform any efficient randomized task almost as efficiently without using randomness at all, or while using as little randomness as possible. This is the goal of *derandomization*, which has been widely studied in complexity theory. While generic derandomization is possible in many settings under widely believed circuit lower bound hypotheses, it also *implies* circuit lower bounds that are believed to be hard to establish (cf. [IKW02, KI04, Wil13]). Thus, while many specific randomized tasks can be derandomized, provable generic derandomization seems out of reach with our current state of knowledge.

A few years ago, Gat and Goldwasser [GG11] introduced the notion of *pseudo-deterministic algorithms*, motivated by applications in cryptography and distributed computing. A pseudo-deterministic algorithm is one that, on a given input, produces a fixed output with very high probability. Thus, a pseudo-deterministic algorithm is one that *looks* deterministic to an outside observer who is computationally bounded – even if such an observer were to run the algorithm multiple times, she is likely to always get the same answer.

Pseudo-deterministic algorithms have a very desirable feature possessed by deterministic algorithms, viz. little to no uncertainty in the output. Thus it is of interest to convert randomized algorithms to equivalent pseudo-deterministic ones – we term such a conversion "pseudo-derandomization".

There are several interesting examples of pseudo-derandomization known, including finding primitive roots [Gro15] and quadratic non-residues (cf. [GG11]) in prime fields, finding variable settings for polynomial identity testing [GG11], and finding perfect matchings in bipartite graphs in parallel [GG17]. These pseudo-derandomization results exploit specific properties of the known randomized algorithms for these problems. The authors introduced a generic pseudo-derandomization approach for search problems in [OS17b], and used it to give a sub-exponential pseudo-deterministic construction of primes infinitely often. This generic approach has been explored further by [Hol17] and [GGH17].

One limitation of the generic approach of [OS17b] is that it seems to work only for *search problems* whose underlying relation is decidable in P or in BPP. In particular, the approach requires the ability to test in P or in BPP whether a given sequence of random choices made by a randomized algorithm is "good" or not. There are several important settings of randomized tasks where such a test is not available. We consider two such settings in this paper: *learning* and *approximation*.

1.2 Pseudoderandomization and learning

Our learning model is that of learning with membership queries, where the accuracy of the output hypothesis is measured with respect to the uniform distribution. A pseudo-deterministic

learning algorithm in this model is a randomized algorithm that, when given access to a predetermined oracle, makes a fixed set of queries and outputs a fixed output hypothesis with high probability.[1] (The pseudo-deterministic learning model is formalized in the natural way in Section 2.2.)

This setting falls outside the "search problem" paradigm for a couple of different reasons: first, the algorithmic task is not self-contained but requires interaction with an oracle, and second, the test of whether an output hypothesis is good is not precise but approximate, and again requires interaction with the oracle.[2]

Pseudo-determinism is naturally a desirable property for learning algorithms. Indeed, consider a setting where Alice and Bob run the same learning program independently on the same data but wish to co-ordinate their predictions. Pseudo-determinism of the learning algorithm enables them to co-ordinate their predictions perfectly with high probability. In an alternative scenario, suppose Alice runs the learning algorithm to generate a hypothesis, and the hypothesis gets corrupted. Alice can recover the original hypothesis with high probability just by running the learning algorithm again.

The main question we ask is: can learning algorithms be derandomized or at least pseudo-derandomized in a generic fashion? Our first result is the observation that standard pseudo-random generators suffice to derandomize learning. This is somewhat surprising because standard pseudo-random generators are designed for self-contained algorithmic tasks, while learning requires interaction with an unknown oracle.

Recall that we consider randomized algorithms that learn under the uniform distribution and have membership-query access to the unknown function.

Theorem 1 (Conditional derandomization and pseudo-derandomization of learning)**.**
Let $\mathfrak{C} \subseteq \mathsf{P/poly}$ be an arbitrary circuit class, and suppose $\mathfrak{C}(s(n))$ can be learned to any constant accuracy by a randomized algorithm running in time $t(n) \geq n$.

- *If $\mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$ requires circuits of size $2^{\Omega(n)}$ on all large input lengths, then there exists a constant $c \geq 1$ such that $\mathfrak{C}$ can be deterministically learned to any constant accuracy in time at most $O(t(n)^c)$.*
- *If $\mathsf{BPE} = \mathsf{BPTIME}[2^{O(n)}]$ requires circuits of size $2^{\Omega(n)}$ on all large input lengths, then there exists a constant $c \geq 1$ such that $\mathfrak{C}(s)$ can be pseudo-deterministically learned to any constant accuracy in time at most $O(t(n)^c)$.*

The proof of conditional derandomization in Theorem 1 works as follows. Under the assumption that E requires exponential-size Boolean circuits almost everywhere, it is a standard consequence from [NW94, IW97, Uma03] that there is a pseudo-random generator G computable in time $\mathrm{poly}(t(n))$ with seed length $O(\log(t(n))$ secure against circuits of size $t(n)^3$. We simulate the randomized learning algorithm using each output of the generator G as random sequence in turn to obtain hypothesis circuits $D_1 \ldots D_{\mathrm{poly}(t(n))}$, and then output the majority of these circuits as our hypothesis. To argue that this works, we show that if the simulation fails to output a correct hypothesis, there is a distinguisher for the PRG G, contrary to our assumption. The key idea is

[1] We stress that we allow adaptive learning algorithms, and that the "canonical" set of inputs queried by the learner can depend on the target function. We do not require the order of the queries to be fixed.

[2] Also observe that the usual way of testing a learning hypothesis by drawing a set of random examples is not pseudo-deterministic.

that a distinguisher can be constructed in $t(n)^3$ size by replacing the oracle in the simulation of the learning algorithm by a circuit from the class $\mathfrak{C}$ for which the simulation fails. The proof of conditional pseudo-derandomization works similarly, but we need to use an additional idea from [OS17b]. Details are in Section 3.[3]

We get some interesting corollaries from Theorem 1. Under standard hardness assumptions, both Jackson's polynomial-time learning algorithm for DNFs with membership queries [Jac97] and the recent algorithm of [CIKK16] for $\mathcal{AC}^0[p]$ can be derandomized. Note that the randomized learner of [LMN93] for $\mathcal{AC}^0$ has already been derandomized unconditionally by Sitharam [Sit95].[4] Sitharam's deterministic learner exploits specific properties of $\mathcal{AC}^0$ circuits, while we are interested here in generic methods to derandomize and pseudo-derandomize learning algorithms.

Theorem 1 is conditional, but it can be used to establish an *unconditional* result for pseudo-derandomizing learning. This is in contrast to generic derandomization, which can only be done conditionally given our current knowledge of circuit lower bounds.

Theorem 2 (Unconditional pseudo-derandomization of learning)**.**
If $\mathsf{P/poly}$ *can be learned to any constant accuracy by a randomized algorithm running in quasi-polynomial time, then for each* $\gamma > 0$, $\mathsf{P/poly}$ *can be pseudo-deterministically learned to any constant accuracy in time* $O(2^{n^\gamma})$ *for infinitely many input lengths* n*.*

The proof of Theorem 2 proceeds in two steps. In the first step, we use a result of [OS17a] to get circuit lower bounds for BPE from a non-trivial randomized learning for $\mathsf{P/poly}$. In the second step, we apply a variant of Theorem 1 to derive an infinitely-often subexponential-time pseudo-deterministic learner using the circuit lower bounds for BPE.

The assumption in Theorem 2 is very strong; indeed, under standard cryptographic assumptions, $\mathsf{P/poly}$ does not have non-trivial learning algorithms (see e.g. [BR17]). However, the proof technique of Theorem 2 works in the more general setting of *self-learners*, where a self-learner is a learning algorithm for a circuit class $\mathfrak{C}$ that produces a hypothesis in $\mathfrak{C}$ and moreover can itself be implemented in $\mathfrak{C}$. Theorem 2 is just the cleanest instantiation of this proof technique, since any learner for $\mathsf{P/poly}$ is automatically a self-learner. (Self-learning is a phenomenon that might be of independent interest, and we refer to Section 3.3 for further discussion of this concept.) The more general version of Theorem 2 presented in Section 3.3 shows that the same result holds for any self-learnable class that contains $\mathcal{TC}^0$ and is closed under composition. (For the interested reader, we mention that threshold gates are necessary to perform hardness amplification, a technical ingredient in our proof.) This is still a strong assumption, and we leave obtaining a version of Theorem 2 under a weaker hypothesis as an interesting research direction.

Theorem 1 applies pseudo-random generators to the setting of learning. Our next result goes in the opposite direction, showing that derandomizing or pseudo-derandomizing learning algorithms has interesting consequences in the theory of pseudo-randomness. We say that a circuit is γ-*dense* if it accepts at least a γ-fraction of strings in $\{0,1\}^n$.

Theorem 3 (Hitting sets from deterministic and pseudo-deterministic learning)**.**
Let $\mathfrak{C} = \{\mathcal{C}_n\}$ *be an arbitrary circuit class, and assume that for every* $\varepsilon > 0$, $\mathfrak{C}$*-circuits of size* $s(n)$ *can be deterministically learned to accuracy* ε *in time* $T(n) \geq n$*.*

[3] For simplicity, we have restricted the statement of Theorem 1 to constant-accuracy learners. As explained in Section 3, from a randomized ε-accuracy learner one can get a deterministic $O(\varepsilon)$-accuracy learner by using sufficiently strong generators (see Lemma 7).

[4] See also [SS97] for related results in the context of learnability using a linear combination of parity functions.

Then, for every $\gamma > 0$, there exists a hitting set generator $G_n \colon \{0,1\}^{\log T(n)} \to \{0,1\}^n$ computable in time $O(T(n))$ against the class of γ-dense circuits in $\mathcal{C}_n(s(n))$. Similarly, if $\mathfrak{C}$ is pseudo-deterministically learnable, there exist pseudodeterministic hitting set generators against $\mathcal{C}_n(s(n))$ with the same parameters.

The proof of Theorem 3 is along the lines of the argument used to prove that a deterministic black-box PIT algorithm implies a hitting set. Suppose that there exists a deterministic learner. We run the deterministic learner with oracle the identically zero function, and output the set of queries it makes as our candidate hitting set. If the set of queries is not a hitting set, then there must be a somewhat dense function f computable in $\mathfrak{C}$ for which all the queries answer 0, just as they do for the identically zero function. But by the correctness of the learning algorithm, this would mean that f can be well-approximated by the identically zero function, which contradicts the assumption that it is somewhat dense. The consequence for pseudo-deterministic learners is shown by appropriately adapting this argument.

An interesting application of Theorem 3 is to the question of whether small hitting sets exist for $\mathcal{AC}^0[p]$ circuits. Despite much effort, no hitting sets even of sub-exponential size are known for such circuits (we refer to [FSUV13] for related results and discussion). Theorem 3 suggests an approach to this question via learning. Carmosino et al. [CIKK16] recently gave a quasi-polynomial time randomized learning algorithm for $\mathcal{AC}^0[p]$ – if this algorithm could be made deterministic, we would immediately get quasi-polynomial size hitting sets for $\mathcal{AC}^0[p]$ in quasi-polynomial time! In particular, that would imply that randomized poly-size $\mathcal{AC}^0[p]$ circuits with one-sided error can be simulated by deterministic quasi-poly size circuits. Even a pseudo-derandomization of the [CIKK16] algorithm would be interesting, as this would give somewhat efficient pseudo-deterministic hitting sets against $\mathcal{AC}^0[p]$, which is also unknown. We mention that the derandomization of *weak* learning algorithms can also be used to construct pseudorandomness, and refer to Section 4 for additional results.[5]

Theorem 3 also has consequences for non-uniform circuit lower bounds that can be derived from learning algorithms. It is known that *non-trivial* learning algorithms (i.e., those running in time $2^n/n^{\omega(1)}$) for a circuit class $\mathfrak{C}$ yield lower bounds against $\mathfrak{C}$ (cf. [KKO13, OS17a, FK09, HH13, Vol14]). However, different algorithms provide different types of lower bounds. For a deterministic learner, one obtains a function in E that is hard almost everywhere [KKO13], while for randomized learners, the hard function lives in BPE and is only hard infinitely often [OS17a]. Interestingly, it is possible to use Theorem 3 to get something stronger from non-trivial *pseudo-deterministic* (randomized) learning algorithms: they can be used to define a function in BPE that is hard almost-everywhere for $\mathfrak{C}$. We discuss this application in more detail in Section 4.3.

Table 1 appearing below summarizes and puts in perspective some of our learning results.

1.3 Pseudoderandomization and approximation

We next turn our attention to a different setting, the setting of *approximation*. We are interested in integer-valued functions, i.e., functions from strings to non-negative integers, that have efficient randomized approximation schemes. The question is whether the existence of a good randomized approximation scheme generically implies the existence of a somewhat efficient pseudo-deterministic

[5] Recall that a weak learner is only required to output a hypothesis that correctly computes the unknown function on slightly more than half of the input strings.

Circuit Class	Existing learning algorithms	PRGs/HSGs (seed length)	Self-learning → Pseudodet. learning	Pseudodet. learning → Pseudodet. HSGs
DNF(poly)	randomized, poly-time	$\approx (\log n)^2$	unknown	yes
$\mathcal{AC}^0_d$	deterministic, quasi-poly	$(\log n)^{d+O(1)}$	unknown	yes
$\mathcal{AC}^0[p]$	randomized, quasi-poly	$(1-o(1))n$	unknown	yes
$\mathcal{TC}^0$	unknown	n (trivial)	yes	yes
P/poly	unknown	n (trivial)	yes	yes

Table 1: An informal description of some learning algorithms (cf. [Jac97, LMN93, Sit95, CIKK16]) and PRGs/HSGs (see e.g. [DETT10, TX13, FSUV13]) for different circuit classes, including implications among these notions in the pseudodeterministic setting (Theorems 2 and 3 and extensions). These results further motivate the investigation of pseudodeterministic learning algorithms, self-learnability, and connections between learning and pseudorandomness.

approximation scheme. (Pseudo-deterministic approximation schemes are formalized in the natural way in Section 2.3.)

Note that this setting too does not conform to the "search problem" paradigm. Given a value w, it might be hard to test if the value is close to the correct value, since the correct value might be very hard to compute. Indeed, in our results, we make no assumptions about the complexity of exact computation of the integer-valued function.

Our main result here is a generic pseudo-derandomization of randomized approximation schemes; however, this pseudo-derandomization is only guaranteed to work on infinitely many input lengths with high probability over any poly-time samplable distribution of inputs.

Theorem 4 (Unconditional pseudo-derandomization of approximation)**.** *Let $f\colon \{0,1\}^* \to \mathbb{N}$ be any function with a polynomial-time randomized approximation scheme. Then for each polynomial-time samplable sequence $\mathfrak{D}$ of distributions and for each constant $\delta > 0$, f has a pseudo-deterministic approximation scheme for infinitely many n over $\mathfrak{D}$ running in time $O(2^{n^\delta})$.*

The main idea in the proof of Theorem 4 is to exploit the uniform hardness-randomness tradeoffs used in the generic pseudo-derandomization results of [OS17b], but adapted to this new setting. The crucial point is: how do we test efficiently that a value w is a good approximation to the correct value? We test this simply by running the randomized approximation scheme to produce a value w' and checking if w is close to w'. This is not a deterministic polynomial-time test or indeed a bounded-error probabilistic polynomial-time test; however, we can show that it is good enough for our purposes. The details are technical, and can be found in Section 5.

As a corollary of this result and [JSV04], we get unconditionally that the $(0,1)$-Permanent has a pseudo-deterministic approximation scheme running in sub-exponential time on infinitely many input lengths over any poly-time samplable distribution on inputs.

Finally, we consider a notion of *approximate canonization* of circuits. Canonization is a natural notion for an equivalence relation, where for each element of the set we compute a representative member of its equivalence class. Needless to say, canonization and canonical forms are fundamental notions with a variety of applications both in mathematics and computer science. We are interested in the natural equivalence relation between circuits: two circuits are equivalent if they compute the same function.

It is easy to see that efficient canonization is impossible for even weak circuit classes such as DNFs, under standard complexity assumptions (see Section 5.2). Therefore we *relax* the notion of canonization. We still require the output of the canonizer to be the same for any two equivalent circuits, but this output need not be a circuit equivalent to the original circuit, instead it is allowed to be *close* to the original circuit over the uniform distribution on inputs.[6]

Inspired by an observation in [BGI+12], we show that efficient deterministic (resp. pseudo-deterministic) learning implies efficient deterministic (resp. pseudo-deterministic) approximate canonization. (We refer to Section 2.3 for a precise definition of approximate canonization.) Using Theorem 1 and the learning algorithm in [CIKK16], we get quasi-polynomial time pseudo-deterministic approximate canonization for $\mathcal{AC}^0[p]$ circuits under a standard circuit lower bound assumption for BPE.

Theorem 5 (Approximate canonization for $\mathcal{AC}^0[p]$, Informal)**.**
Let $p \geq 2$ be a fixed prime. If BPE *requires circuits of size $2^{n^{\Omega(1)}}$ almost everywhere, then $\mathcal{AC}^0[p]$ circuits can be approximately canonized in pseudo-deterministic quasi-polynomial time.*

The proof of Theorem 5 and other related results appear in Section 5.2. We leave as an open problem obtaining an unconditional version of this theorem. Another interesting research direction is the investigation of connections between approximate canonization and other meta-computational problems. In this sense, we mention that [BBF16] provides evidence that expressive circuit classes do not admit approximate canonization. (In fact, even more relaxed notions of approximate canonization are conditionally ruled out by the results from [BBF16], and we refer to their work for further details.)

2 Preliminaries

While the exposition of our main ideas is mostly self-contained, we assume familiarity with basic notions from complexity theory (cf. [AB09]), learning theory (cf. [KV94]), and circuit complexity theory (cf. [Juk12]). In particular, we refer to these references for the definition of standard circuit classes.

Let $\mathcal{F}_n$ be the set of all boolean functions $f\colon \{0,1\}^n \to \{0,1\}$ on n input variables, and $\mathfrak{F} = \bigcup_{n\geq 1} \mathcal{F}_n$ be the set of all boolean functions. We use boldface letters such as $\boldsymbol{w}$ and $\boldsymbol{x}$ to denote random variables. We say that boolean functions f and g from $\mathcal{F}_n$ are ε-close if $\Pr_{\boldsymbol{x}\sim\mathcal{U}_n}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})] \leq \varepsilon$, where $\mathcal{U}_n$ denotes the uniform distribution over $\{0,1\}^n$. We often view a string in $\{0,1\}^*$ that represents a boolean circuit D as if it were the actual circuit D, or the function that it computes.

2.1 Randomness, pseudorandomness and pseudodeterminism

We will require the notion of polynomial-time samplability of a sequence of distributions. Let $\mathfrak{D} = \{\mathcal{D}_n\}$ be a sequence of distributions, where each $\mathcal{D}_n$ is supported on $\{0,1\}^n$. We say that $\mathfrak{D}$ is polynomial-time samplable if there is a probabilistic polynomial-time algorithm B such that for each $n \in \mathbb{N}$ and each $y \in \{0,1\}^*$, $\Pr_{\boldsymbol{B}}[\boldsymbol{B}(1^n) = y] = \Pr[y \in \mathcal{D}_n]$.

[6] A form of approximate obfuscation is also investigated in [BGI+12], but their definition requires a much stronger correctness guarantee.

We also require notions of pseudorandomness, introduced next.

Pseudorandom generators and hitting set generators. Let $\mathcal{D}_m$ be a probability distribution supported over $\{0,1\}^m$. We say that $\mathcal{D}_m$ is (η, s)-*pseudorandom* for a circuit class $\mathcal{C} \subseteq \mathcal{F}_m$ if for each size-s circuit $g \in \mathcal{C}(s)$,

$$\left| \Pr_{\boldsymbol{x} \sim \mathcal{U}_m} [g(\boldsymbol{x}) = 1] - \Pr_{\boldsymbol{y} \sim \mathcal{D}_m} [g(\boldsymbol{y}) = 1] \right| < \eta.$$

In other words, the circuit g is η-*fooled* by $\mathcal{D}_m$. This definition extends to an ensemble $\mathfrak{D} = \{\mathcal{D}_m\}_{m \geq 1}$ of distributions, by requiring the condition above to hold for every $m \geq 1$. Moreover, we say that a function $G_m \colon \{0,1\}^{\ell(m)} \to \{0,1\}^m$ is an η-*pseudorandom generator* for a class $\mathcal{C}$ if the induced distribution $G_m(\mathcal{U}_{\ell(m)})$ is (η, m)-pseudorandom for $\mathcal{C}$. Equivalently, the induced distribution η-fools every size-m $\mathcal{C}$-circuit over m-input variables. The function $\ell(m)$ computes the *seed length* of the generator G_m.

We also consider the weaker notion of hitting sets. We say that a set $\mathcal{H}_m \subseteq \{0,1\}^m$ is an (η, s)-*hitting set* for $\mathcal{C}$ if for each size-s circuit $g \in \mathcal{C}(s)$ such that $\Pr_{\boldsymbol{x} \sim \mathcal{U}_m}[g(\boldsymbol{x}) = 1] \geq \eta$, we have $g^{-1}(1) \cap \mathcal{H}_n \neq \emptyset$. This definition extends to ensembles of sets in the natural way. Similarly, a function $H_m \colon \{0,1\}^{\ell(m)} \to \{0,1\}^m$ is an η-*hitting set* for $\mathcal{C}$ if the induced set $H_m(\{0,1\}^{\ell(m)}) \subseteq \{0,1\}^m$ is an (η, m)-hitting set for $\mathcal{C}$. (Note that the support of a pseudorandom distribution is a hitting set with the same parameter η.)

We say that a pseudorandom generator or a hitting set generator is *quick* if it can be computed in time $2^{O(\ell(m))}$, where $\ell(m)$ is the corresponding seed length.

The following result will be useful.

Theorem 6 ([Uma03])**.** *Given a function $f \colon \{0,1\}^{\log \ell} \to \{0,1\}$ of circuit complexity at least s, it is possible to construct a pseudorandom generator $G \colon \{0,1\}^{O(\log \ell)} \to \{0,1\}^m$ that $(1/m)$-fools size m circuits, where $m = s^{\Omega(1)}$. Moreover, G can be computed in time $\ell^{O(1)}$ given the description of the truth table of f.*

Pseudodeterministic pseudorandommness. We will make use of pseudorandom distributions $\mathcal{D}_m$ and hitting sets $\mathcal{H}_m$ that are constructed *pseudodeterministically*. For our purposes, we define the relevant concepts as follows. Let $G_m \colon \{0,1\}^{t(m)} \times \{0,1\}^{\ell(m)} \to \{0,1\}^m$. We say that G_m is a μ-pseudodeterministic (η, m)-pseudorandom generator for $\mathcal{C}$ if there is an (η, m)-pseudorandom generator $G^\star_m \colon \{0,1\}^{\ell(m)} \to \{0,1\}^m$ for $\mathcal{C}$ such that

$$\Pr_{\boldsymbol{a} \sim \mathcal{U}_t} [G(\boldsymbol{a}, \cdot) \equiv G^\star_m(\cdot)] \;\geq\; 1 - \mu,$$

where the "$\equiv$" symbol represents identity among functions. A μ-pseudodeterministic hitting set generator $H_m \colon \{0,1\}^{t(m)} \times \{0,1\}^{\ell(m)} \to \{0,1\}^m$ is defined analogously. Analogously, we say that a pseudodeterministic pseudorandom or hitting set generator is *quick* if it can be computed in time $2^{O(\ell(m))}$, where $\ell(m)$ is the seed length.

Note that the pseudodeterministic parameter μ of a quick pseudorandom or hitting set generator can be boosted by standard techniques. Indeed, since quick generators can tolerate a running time overhead of $2^{O(\ell(n))}$, one can always design a new generator that uses a larger random string, samples independent copies of the initial pseudodeterministic generator, and behaves as the most common generator among the induced generators provided by these samples.

2.2 Learning

Learning algorithms. We consider randomized learning algorithms under the uniform distribution that can make membership queries to the unknown function. We formalize such algorithms next.

Fix a class of functions $\mathfrak{C} \subseteq \mathfrak{F}$, often referred to as the *concept class*. For convenience, we write $\mathfrak{C} = \{\mathcal{C}_n\}_{n\geq 1}$, where $\mathcal{C}_n \subseteq \mathcal{F}_n$. A randomized algorithm A (ε, δ)-*learns* a class $\mathfrak{C}$ if for every $n \geq 1$ and for each $f \in \mathcal{C}_n$, when given oracle access to f and access to inputs 1^n, $\varepsilon > 0$ (accuracy), and $\delta > 0$ (confidence), A outputs the description of a boolean circuit D such that

$$\Pr_{\boldsymbol{w}}[\boldsymbol{D} = A^f(1^n, \varepsilon, \delta, \boldsymbol{w}) \text{ is } \varepsilon\text{-close to } f\ ] \geq 1 - \delta.$$

Here $\boldsymbol{w} \in \{0,1\}^*$ is a uniformly random boolean string representing the randomness of A, and $\boldsymbol{D} = A^f(1^n, w, \varepsilon, \delta)$ is a random variable denoting the (representation of the) circuit output by A over these inputs and with oracle access to f. For convenience, we might omit some input parameters when discussing the computation of A.[7]

While many of our results hold in a more general setting, for simplicity we will focus on the learnability of classes of boolean circuits. Therefore, $\mathcal{C}_n$ will always denote a class of the form $\mathcal{C}(s(n))$, where $\mathcal{C} \in \{\mathcal{AC}^0, \mathcal{AC}^0[p], \mathcal{TC}^0, \text{etc.}\}$, and $s(n)$ is an upper bound on circuit size complexity. When there is no risk of confusion, we might write $\mathcal{C}_d$ to restrict the class to circuits of depth at most d. The worst-case running time of the learning algorithm A over the choice of $f \in \mathcal{C}(s(n))$ and of its internal random string w is measured by the function $t_A(n, s, 1/\varepsilon, 1/\delta)$.

Pseudodeterministic learning. A randomized algorithm A $(\varepsilon, \delta, \gamma)$-*pseudodeterministically learns* a class $\mathfrak{C} = \{\mathcal{C}_n\}$ if A (ε, δ)-learns this class, and moreover for every $n \geq 1$ and $f \in \mathcal{C}_n$ there is a fixed set of queries $Q_f \subseteq \{0,1\}^n$ and a fixed string D_f representing a boolean circuit such that

$$\Pr_{\boldsymbol{w}}[A^f(1^n, \boldsymbol{w}) \text{ queries } f \text{ exactly over } Q_f \ \text{ and } \ A^f(1^n, \boldsymbol{w}) = D_f] \geq 1 - \gamma.$$

In other words, with high probability the learner makes the same set of queries and outputs the same boolean circuit (representation) as its hypothesis. We say in this case that A is a γ-*pseudodeterministic* learning algorithm.

We would like to stress that it makes sense to consider variants of this notion where only the set of queries is pseudodeterministic (*query-pseudodet. learner*), or where only the output hypothesis is pseudodeterministic (*hypothesis-pseudodet. learner*). For instance, if A is a pseudodeterministic learner, running several independent copies of A and outputting the most common hypothesis will boost the initial hypothesis-pseudodeterminism parameter, but the resulting learner will be no longer query-pseudodeterministic.

The circuit complexity of learning algorithms. It is crucial in our investigations to consider a notion of complexity for learning algorithms that is more refined than running time. We measure instead the *circuit complexity* of learning algorithms. In other words, we specify a learning algorithm by a sequence $\{D_n\}_{n\geq 1}$ of *multi-output oracle* circuits D_n that have access to $\boldsymbol{w}$, ε, and δ, and whose *oracle queries* are answered according to the unknown function $f \in \mathcal{C}_n$. (In particular, the

[7] It is well-known that the confidence parameter δ can be made arbitrarily small (cf.[KV94]). It is also known how to boost the accuracy parameter ε if the concept class satisfies a certain closure property (see e.g. [BL93]).

main input string of the oracle circuit is the random string, and in many cases we will fix ε and δ in advance.) The output bits of D_n encode a circuit describing the output hypothesis.[8]

In the case of learning circuits that are less powerful than general circuits (such as $\mathcal{AC}^0$, $\mathcal{TC}^0$, etc.), we further restrict the output hypothesis of the learner. We say that a class $\mathfrak{C}$ is learnable by $\mathfrak{D}$-circuits if the sequence $\{D_n\}$ consists of circuits from $\mathfrak{D}$, and moreover the output string is an *effective encoding* of a $\mathfrak{D}$-circuit. The meaning of effective description is that it should be possible for $\mathfrak{D}$-circuits to interpret the output string as the description of a $\mathfrak{D}$-circuit, and to efficiently evaluate computations given this description. We will not be explicit about such encodings, and simply note that they exist for the typical circuit classes investigated in our work.[9]

For definiteness, we briefly discuss a notion of *uniformity* for such sequence of learning circuits. In *learning upper bounds*, we assume that the sequence can be generated from 1^n by a deterministic algorithm that runs in time polynomial in the size of the circuits. We will not discuss *circuit lower bounds for learning* in this paper, but in such a context it is also natural to consider *non-uniform* sequences of learning circuits (see e.g. [OS17a, Section 4]).

We say that a circuit class $\mathfrak{C}$ is *self-learnable* if it can be learned by a sequence of $\mathfrak{C}$-circuits, typically of quasi-polynomial size. This is an informal working definition, since for instance we do not specify the dependence on the parameters ε and δ. We will leave such details to the formal statement of our results, where we will often assume that these learning parameters are sufficiently small constants, and allow the class $\mathcal{C}_d(n^k)$ to be learned by $\mathcal{C}_{d'}(n^{(\log n)^{k'}})$-circuits (multi-output and with oracle gates).

We assume that functions related to algorithmic parameters such as time bounds, circuit size, learning accuracy, etc. are sufficiently constructive, in the sense that they do not affect the asymptotic complexity of our reductions whenever an algorithm needs to compute one of these functions.

2.3 Approximation

Approximation schemes. We define notions of approximation for computing integer-valued functions. An *integer-valued function* is a function from strings to non-negative integers, i.e., from $\{0,1\}^*$ to $\mathbb{N}$. We say that an integer-valued function f has a *polynomial-time randomized approximation scheme* (PRAS) if for each rational number $\epsilon > 0$ there is a probabilistic polynomial-time machine M, which given any string x as input, outputs an integer $M(x)$ (which might depend on the random choices of M) such that with probability $1 - 2^{-\Omega(|x|)}$ over the random choices of M, we have that $(1-\epsilon)f(x) \leq M(x) \leq (1+\epsilon)f(x)$. We say that f has an *fully polynomial-time randomized approximation scheme* (FPRAS) if there is a probabilistic machine M, which given a string x and a rational number ϵ (in some prespecified format) as input, runs in time $\mathrm{poly}(|x|, 1/\epsilon)$ and outputs a number $M(x)$ such that with probability $1 - 2^{-\Omega(|x|)}$ over the random choices of M, we have that $(1-\epsilon)f(x) \leq M(x) \leq (1+\epsilon)f(x)$.

An example of an integer-valued function is the permanent of a $(0,1)$-matrix, when the matrix is represented as a bitstring. By the celebrated result of Jerrum, Sinclair and Vigoda [JSV04], this integer-valued function has an FPRAS.

[8] In the case of *deterministic* learning circuits, we remark that each D_n has access to the constant input bits 0 and 1, and one can think of its "input string" as the first batch of answers provided by the oracle queries.

[9] For bounded-depth circuit classes, we tolerate a constant-factor depth blow-up during the evaluation if this is necessary from the choice of encoding.

We will be interested in converting randomized approximation schemes to *pseudo-deterministic* ones, where with high probability the algorithm outputs a fixed number that is a good approximation to the correct value. Given a time function $T\colon \mathbb{N} \to \mathbb{N}$, we say that an integer-valued function f has a *pseudo-deterministic approximation scheme* (PDAS) running in time T if for each rational number $\epsilon > 0$ there is a function $g\colon \{0,1\}^* \to \mathbb{N}$ and a probabilistic machine M, which given a string x as input, runs in time $T(|x|)$ and outputs $g(x)$ with probability $1 - 2^{-\Omega(|x|)}$, and moreover we have that $(1-\epsilon)f(x) \leq g(x) \leq (1+\epsilon)f(x)$. A PDAS running in polynomial time is called a PPDAS. We say that an integer-valued function f has a *fully pseudo-deterministic approximation scheme* (FPDAS) running in time T if there is a function $g\colon \{0,1\}^* \times \mathbb{Q}^+ \to \mathbb{N}$ and a probabilistic machine M, which given a string x and a rational number ϵ (in some prespecified format) as input, runs in time $T(|x|, 1/\epsilon)$ and outputs $g(x,\epsilon)$ with probability $1-2^{-\Omega(|x|)}$, and moreover we have that $(1-\epsilon)f(x) \leq g(x,\epsilon) \leq (1+\epsilon)f(x)$. An FPDAS running in polynomial time is called a PFPDAS.

Note that a *d*eterministic approximation scheme running in time T is a special case of a PDAS running in time T where the machine M uses no randomness, and similarly a *fully deterministic approximation scheme* running in time T is a special case of an FPDAS running in time T where the machine M uses no randomness.

We also need more relaxed notions of pseudo-deterministic approximation schemes which are not guaranteed to work for all inputs. An *infinitely-often pseudo-deterministic approximation scheme* (i.o.PDAS) is only guaranteed to be pseudo-deterministic and output a correct approximation for infinitely many input lengths (rather than all of them). The notion of *infinitely-often fully pseudo-deterministic approximation scheme* (i.o.FPDAS) is defined analogously.

Finally, given a samplable distribution $\mathfrak{D} = \{\mathcal{D}_n\}$, an i.o.PDAS over $\mathfrak{D}$ is only guaranteed to be pseudo-deterministic and output a correct approximation with probability $1 - 1/n^{\omega(1)}$ over inputs sampled according to $\mathcal{D}_n$, for infinitely many n. Again, the notion of i.o.FPDAS over $\mathfrak{D}$ is defined analogously.

Canonization and approximate canonization. Next we define notions of *canonization* and *approximate canonization* for circuit classes. Let $\mathfrak{C}$ be a circuit class and $s\colon \mathbb{N} \to \mathbb{N}$ be a size function. Given a time function $T\colon \mathbb{N} \to \mathbb{N}$, we say that $\mathfrak{C}(s(n))$ has deterministic (resp. pseudo-deterministic) canonization in time T if there is a deterministic (resp. 1/3-pseudo-deterministic) Turing machine M such that (i) M operates in time $T(n)$ when given as input any circuit C in $\mathfrak{C}(s(n))$, (ii) for any circuit C in $\mathfrak{C}(s(n))$, $M(C)$ is a Boolean circuit on n variables that is *equivalent* to C, i.e., computes the same Boolean function as C, and (iii) for any two equivalent circuits C and C' in $\mathfrak{C}(s(n))$, $M(C) = M(C')$. Note that a pseudo-deterministic canonization algorithm is allowed to output an arbitrary circuit with probability at most 1/3.

We relax the notion of canonization by requiring the output to only be *approximately* equivalent to the input. Given a parameter $\epsilon > 0$, we say that $\mathfrak{C}(s(n))$ has deterministic (resp. pseudo-deterministic) ϵ-approximate canonization in time T if there is a deterministic (resp. 1/3-pseudo-deterministic) Turing machine M such that (i) M operates in time $T(n)$ when given as input any circuit C in $\mathfrak{C}(s(n))$, (ii) for any circuit C in $\mathfrak{C}(s(n))$, $M(C)$ is a Boolean circuit on n variables that is an ϵ-approximation to C, i.e., disagrees with C on at most an ϵ-fraction of inputs of length n, and (iii) for any two equivalent circuits C and C' in $\mathfrak{C}(s(n))$, $M(C) = M(C')$.

Finally, we stress that in the definition of canonization and approximate canonization it is important that the size bound $s(n)$ is fixed before the formalization of the problem. Indeed, by using an alternative definition that simply postulates that on an arbitrary circuit C from the

class $\mathfrak{C}$ the machine M must output (say) in polynomial time on the description length of C an equivalent canonical circuit $M(C)$, one can easily use M to define a *natural property* useful against $\mathfrak{C}$ (in the sense of [RR97]).[10] However, as far as we know, there might be circuit classes that admit approximate canonization in the original sense introduced above, but do not admit natural properties. The first definition is therefore preferred.

3 Pseudo-derandomization for randomized learning algorithms

In this section we consider the derandomization and pseudoderandomization of learning algorithms via pseudorandom generators and pseudodeterministic pseudorandom generators, respectively. For simplicity, we will mostly focus on self-learnable circuit classes, but our results can be extended to more general settings, as explained later in this section.

3.1 Derandomizing from a pseudorandom generator

We start with a technical lemma showing that standard pseudorandom generators can be used to derandomize learning algorithms.

Lemma 7 (PRG-based derandomization of learning algorithms)**.** *Let $\mathfrak{C}$ be a circuit class closed under composition. Let $s, s' \colon \mathbb{N} \to \mathbb{N}$ be functions, where $s'(n) \geq n$. Further, let $\varepsilon, \delta > 0$ be real-valued parameters satisfying $\delta \leq \varepsilon \leq 1/100$ and possibly depending on n. Finally, assume that for each $n \geq 1$ the depth-d class $\mathcal{C}_d(s(n))$ can be (ε, δ)-learned by a (randomized) oracle $\mathcal{C}_{d'}(s'(n))$-circuit.*

There are constants $e = O(d \cdot d')$ and $k \geq 1$ for which the following holds. If there is a family of quick pseudorandom generators $G_m \colon \{0,1\}^{\ell(m)} \to \{0,1\}^m$ that ε-fool depth-e size-m $\mathfrak{C}$-circuits, for

$$m = O(s(n) \cdot s'(n) + s'(n)^k),$$

then $\mathcal{C}_d(s(n))$ can be deterministically learned to accuracy $\varepsilon' = 8\varepsilon$ in time at most $2^{O(\ell(m))} \cdot \mathsf{poly}(s'(n))$.[11]

Proof. Let $\{D_n\}_{n \geq 1}$ be the corresponding (uniform) sequence of learning circuits with fixed parameters ε and δ. We claim that the deterministic algorithm A described below learns every function $f \in \mathcal{C}_d(s(n))$ to accuracy ε' in time at most $2^{O(\ell(m))} \cdot \mathsf{poly}(s'(n))$.

Algorithm A. Input: 1^n and oracle access to an unknown function $f \in \mathcal{C}_d(s(n))$.

1. Computes a multi-set $S_m \stackrel{\text{def}}{=} \{G_m(a) \mid a \in \{0,1\}^{\ell(m)}\}$ of m-bit strings (with multiplicities), where G_m is the pseudorandom generator with parameters as in the statement of the lemma.

2. For each $w \in S_m$, simulates D_n with oracle access to f and with its random input set to w. Let $h_w \stackrel{\text{def}}{=} D_n^f(w)$ be the hypothesis output by the learning circuit under f and w.

3. Outputs the description of a circuit $\widetilde{C}_f$ that on an input $x \in \{0,1\}^n$ computes the majority

[10] Given a truth-table f, construct an exponential size $\mathfrak{C}$-circuit C for f. Since M is a canonizer and has to run in polynomial time on every circuit D equivalent to C, one can infer from its output on C the approximate $\mathfrak{C}$-circuit complexity of f.

[11] For unbounded-depth classes, the circuit depth parameters can be omitted from the statement.

function over the multi-set $\{h_w(x) \mid w \in S_m\}$.

Clearly, under our assumptions A is a deterministic algorithm that runs in time at most $2^{O(\ell(m))} \cdot \mathsf{poly}(s'(n))$. Suppose now that A fails to learn some function $f^\star \in \mathcal{C}_d[s(n)]$. In other words, the corresponding output hypothesis $\widetilde{C}_{f^\star}$ is not ε'-close to $f^\star$. We use this information to construct a *randomized* $\mathfrak{C}$-circuit B of size at most m and depth at most e that distinguishes the output of G_m from random with advantage at least ε. We then fix the randomness of B using a standard argument in order to obtain a deterministic distinguisher. This contradicts the pseudorandomness of G_m, completing the proof of the lemma.

In the description of B presented next, z is a candidate string (either produced from the generator, or uniformly random), and r is a fixed string sampled according to $\boldsymbol{r} \sim \mathcal{U}_n$, a random variable representing the randomness of the distinguisher.

Description of $\boldsymbol{B}$. Input: $z \in \{0,1\}^m$ and $r \in \{0,1\}^n$.

1. Let $C_{f^\star}$ be a $\mathcal{C}_d(s(n))$-circuit that computes $f^\star$. B uses a prefix of z as the randomness of D_n, and simulates the oracle computation $D_n^{f^\star}(z)$ with $C_{f^\star}$ replacing its oracle gates.
2. Suppose h_z is the output hypothesis. B outputs 1 if and only if $h_z(r) = C_{f^\star}(r)$.

Since $C_{f^\star}$ has depth $\leq d$ and size $\leq s(n)$, and D_n is an oracle circuit of depth $\leq d'$ and size $\leq s'(n)$, Step 1 can be implemented by a $\mathfrak{C}$-circuit of depth at most $d \cdot d'$ and of size at most $s(n) \cdot s'(n)$. By definition, the output hypothesis of D_n is restricted to circuits in $\mathcal{C}_{d'}(s'(n))$, and h_z is an effective description of a $\mathfrak{C}$-circuit. Consequently, the evaluation $h_z(r)$ in Step 2 can be computed by a $\mathfrak{C}$-circuit of depth $O(d')$ and size $\mathsf{poly}(s'(n))$. It follows that Step 2 can be implemented by a $\mathfrak{C}$-circuit of depth no more than $O(d' + d)$ and of size no more than $O(s(n) + \mathsf{poly}(s'(n)))$. Overall, we get that B is a (randomized) $\mathfrak{C}$-circuit of depth at most e and of size at most m, where these parameters are as in the statement of the lemma.

We argue in what follows that

$$\left| \Pr_{\boldsymbol{x} \sim \mathcal{U}_m, \boldsymbol{r} \sim \mathcal{U}_n} [B(\boldsymbol{x}, \boldsymbol{r}) = 1] \; - \; \Pr_{\boldsymbol{y} \sim \mathcal{U}_{\ell(m)}, \boldsymbol{r} \sim \mathcal{U}_n} [B(G_m(\boldsymbol{y}), \boldsymbol{r}) = 1] \right| \; > \; \varepsilon. \tag{1}$$

Observe that this implies in particular that for some fixed choice of $r \in \{0,1\}^n$, $B_r \stackrel{\text{def}}{=} B(\cdot, r)$ is a *deterministic* $\mathfrak{C}$-circuit of no larger complexity that distinguishes $\mathcal{U}_m$ and $G_m(\mathcal{U}_{\ell(m)})$ with advantage at least ε, which completes the proof.

Consider the leftmost probability in Equation 1. Since D_n learns every $f \in \mathcal{C}_d(s(n))$ to accuracy ε and with confidence parameter δ and $C_{f^\star} \equiv f^\star$, with probability at least $1 - \delta$ over $\boldsymbol{x}$, Step 2 of circuit B computes a hypothesis $h_{\boldsymbol{x}}$ that is ε-close to $f^\star$. For each fixed $x \in \{0,1\}^m$ that produces an ε-close h_x, in Step 2 circuit B accepts the input pair (x, r) with probability at least $1 - \varepsilon$ over the choice of $r \sim \boldsymbol{r}$. Consequently, using that $\delta \leq \varepsilon$, the leftmost probability in Equation 1 is at least $(1 - \delta)(1 - \varepsilon) \geq 1 - 2\varepsilon$.

It remains to upper bound the rightmost probability. Because A fails to learn $f^\star$ to accuracy ε', there is a set $T \subseteq \{0,1\}^n$ of measure at least ε' such that on every $x \in T$, $\widetilde{C}_{f^\star}(x) \neq f^\star(x)$. Consequently, for $x \in T$ at least half of the values $h_w(x)$ generated in Step 3 of A's description

do not agree with $f^\star(x)$. It follows that over the choice of $\boldsymbol{y}$ and $\boldsymbol{r}$, $B(G_m(\boldsymbol{y}), \boldsymbol{r})$ rejects with probability at least $\varepsilon'/2 = 4\varepsilon$. Consequently, the rightmost probability $\leq 1 - 4\varepsilon$.

It follows from these estimates that the distinguishing probability in Equation 1 is strictly larger than ε, from which the result follows. □

It is important in the preceding argument for the distribution employed in the derandomization to be pseudorandom against *non-uniform* $\mathfrak{C}$-circuits.[12] First, this allows us to disregard the complexity of uniformly generating D_n in the proof that B is an appropriate distinguisher. Most importantly, we have no control over the "bad" function $f^\star$ where the derandomization might fail, and consequently $C_{f^\star}$ appears as a non-uniform advice in the proof of Lemma 7. Finally, r is also fixed non-uniformly when derandomizing the distinguisher.

3.2 Pseudoderandomization of learning algorithms

Similarly to Lemma 7, we now show that self-learning classes admit pseudodeterministic learners under the existence of suitable pseudodeterministic pseudorandom generators.

Lemma 8 (Pseudoderandomization via pseudodeterministic PRGs)**.** *Let $\mathfrak{C}$ be a circuit class closed under composition. Let $s, s' \colon \mathbb{N} \to \mathbb{N}$ be functions, where $s'(n) \geq n$. Further, let $\varepsilon, \delta, \mu > 0$ be real-valued parameters satisfying $\delta \leq \varepsilon \leq 1/100$ and possibly depending on n. Finally, assume that for each $n \geq 1$ the depth-d class $\mathcal{C}_d(s(n))$ can be (ε, δ)-learned by a (randomized) oracle $\mathcal{C}_{d'}(s'(n))$-circuit.*

There are constants $e = O(d \cdot d')$ and $k \geq 1$ for which the following holds. If there is a family of quick μ-pseudodeterministic pseudorandom generators $G_m \colon \{0,1\}^{t(m)} \times \{0,1\}^{\ell(m)} \to \{0,1\}^m$ that ε-fool depth-e size-m $\mathfrak{C}$-circuits, for

$$m = O(s(n) \cdot s'(n) + s'(n)^k),$$

then $\mathcal{C}_d(s(n))$ can be $(8\varepsilon, \mu, \mu)$-pseudodeterministically learned in randomized time at most $2^{O(\ell(m))} \cdot \mathsf{poly}(s'(n))$.

Proof. We proceed as in the proof of Lemma 7, except that the corresponding derandomized algorithm A is replaced here by a pseudoderandomized algorithm A'. This procedure uses its random input $\boldsymbol{y} \in \{0,1\}^{t(m)}$ to define a candidate (deterministic) pseudorandom generator $G_m^{\boldsymbol{y}} \stackrel{\text{def}}{=} G_m(\boldsymbol{y}, \cdot) \colon \{0,1\}^{\ell(m)} \to \{0,1\}^m$. By assumption, it succeeds with probability at least $1 - \mu$, and whenever this happens, A' outputs a hypothesis $h_{\boldsymbol{y}}$ that is ε'-close to f, the unknown function, where $\varepsilon' = 8\varepsilon$ is as in Lemma 7. Consequently, A' is a (ε', μ)-learner for the class. Furthermore, with probability at least $1 - \mu$, A' constructs the same pseudorandom generator. Since the rest of its computation is deterministic, the corresponding learner will make a fixed set Q_f of queries, and generate a fixed output hypothesis h_f. This shows that A' is μ-pseudodeterministic. As the running time of A and A' are the same up to low order terms, it follows that $\mathcal{C}_d(s(n))$ can be $(8\varepsilon, \mu, \mu)$-pseudodeterministically learned in randomized time at most $2^{O(\ell(m))} \cdot \mathsf{poly}(s'(n))$. □

Remark. As we alluded to before, Lemmas 7 and 8 hold in more generality provided that we have sufficiently strong pseudorandom generators. In particular, it is sufficient to have a generator

[12] Distributions that are pseudorandom against uniform algorithms were crucially employed in the pseudodeterministic construction of primes from [OS17b].

that fools a circuit class closed under composition that is expressive enough to simulate circuits in the concept class, the learning circuit, and its hypothesis class. Consequently, existing learning algorithms can be derandomized under hardness assumptions.

Theorem 9 (Conditional learning derandomization)**.** *Let* $\mathfrak{C} \subseteq \mathsf{P/poly}$ *be an arbitrary circuit class, and suppose* $\mathfrak{C}(s(n))$ *can be learned to any constant accuracy by a randomized algorithm running in time* $t(n) \geq n$*. If* $\mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$ *requires circuits of size* $2^{\Omega(n)}$ *on all large input lengths, then there exists a constant* $c \geq 1$ *such that* $\mathfrak{C}(s)$ *can be deterministically learned to any constant accuracy in time at most* $O(t(n)^c)$*.*

Proof. Observe that a learning algorithm running in time $t(n)$ can be implemented by oracle circuits of size at most $\mathsf{poly}(t(n))$. The result is then a direct consequence of Lemma 7 and the hardness vs. randomness paradigm (Theorem 6). □

As a concrete example, Theorem 9 and Jackson's polynomial time learning algorithm for DNFs [Jac97] immediately imply the following result.

Corollary 10. *If* $\mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$ *requires circuits of size* $2^{\Omega(n)}$ *on all large input lengths, then polynomial size* DNF*s can be learned to constant accuracy in deterministic polynomial time.*

The same approach provides pseudoderandomization via Lemma 8 using that a hard truth-table can be *pseudodeterministically* constructed from a weaker lower bound assumption.

Theorem 11 (Conditional learning pseudoderandomization)**.** *Let* $\mathfrak{C} \subseteq \mathsf{P/poly}$ *be an arbitrary circuit class, and suppose* $\mathfrak{C}(s(n))$ *can be learned to any constant accuracy by a randomized algorithm running in time* $t(n) \geq n$*. If* $\mathsf{BPE} = \mathsf{BPTIME}[2^n]$ *requires circuits of size* $2^{\Omega(n)}$ *on all large input lengths, then there exists a constant* $c \geq 1$ *such that* $\mathfrak{C}(s)$ *can be pseudodeterministically learned to any constant accuracy in time at most* $O(t(n)^c)$*.*

Proof. Note that, under this lower bound assumption, there exists a randomized algorithm that on input 1^ℓ, runs in time at most $2^{O(\ell)}$ and outputs with high probability the description of a fixed function $f_\ell \colon \{0,1\}^\ell \to \{0,1\}$ that requires circuits of size $2^{\Omega(\ell)}$. In other words, exponentially hard boolean functions can be pseudo-deterministically constructed in time polynomial in the size of their truth tables. The result now follows from the learning assumption, Theorem 6, and Lemma 8. □

For instance, thanks to the quasi-polynomial time randomized learning algorithm for $\mathcal{AC}^0[p]$ from [CIKK16], we get the following conditional result.

Corollary 12. *If there is* $\gamma > 0$ *and a language in* $\mathsf{BPE} = \mathsf{BPTIME}[2^n]$ *that requires circuits of size* $\geq 2^{n^\gamma}$ *on all large input lengths, then* $\mathcal{AC}^0[p]$ *circuits can be learned to any constant accuracy in pseudodeterministic quasi-polynomial time.*

Proof. Simply observe that this lower bound is enough to get quasi-polynomial time (pseudo-deterministic) derandomizations using the hardness versus randomness paradigm (Theorem 6). The result follows as in the proof of Theorem 11 using the learning algorithm from [CIKK16]. □

A remark on PAC learning derandomization. In contrast to the PAC model framework [Val84], it is crucial in derandomization applications that we define the learning model using an oracle to the unknown function, even if the learner only queries the oracle at random inputs. More

precisely, one can consider the PAC model formulation where a random example oracle $\mathcal{EX}(f)$ (relative to some distribution) is provided to the learner, instead of membership-query access to f. However, in the former model the success probability of a learner is measured with respect to both its internal randomness and $\mathcal{EX}(f)$. A derandomized learner will still fail to learn to high accuracy if the sequence of examples is uninformative. The derandomization procedure only acts on the learning algorithm, and not on the example oracle.

3.3 Pseudodeterministic learners from randomized learners

Recall that $\mathcal{AC}^0$ circuits can be deterministically learned in quasi-polynomial time [SS97], and that $\mathcal{AC}^0[p]$ circuits are known to be learnable in randomized quasi-polynomial time [CIKK16]. In this section, we prove a general result showing that, for strong enough self-learnable circuit classes, any randomized learner running in quasi-polynomial time admits a non-trivial pseudoderandomization.

As opposed to the results discussed in Section 3, the next theorem is *unconditional* and does not assume the existence of pseudorandom generators. Theorem 2 is a particular case of this result.

Theorem 13 (Pseudodeterministic learners from randomized self-learners)**.** *Let $\mathfrak{C}$ be a circuit class that contains $\mathcal{TC}^0$ and is closed under compositions. Suppose that for every $\delta, \varepsilon > 0$, $\mathfrak{C}(\mathsf{poly})$ can be (ε, δ)-learned by (uniform) $\mathfrak{C}$-circuits of quasi-polynomial size. Then, for every $\gamma > 0$ and $c \geq 1$, $\mathfrak{C}(\mathsf{poly})$ can be μ-pseudodeterministically learned to accuracy $\leq n^{-c}$ on infinitely many input lengths by an algorithm running in time $O(2^{n^\gamma})$, where $\mu = 2^{-n}$.*

Proof. First, the assumption implies by a padding argument that for every $\varepsilon > 0$ and $k \geq 1$, there exists $k' \geq 1$ such that $\mathcal{C}(\exp((\log n)^k))$ can be learned to accuracy ε by a uniform family $\mathfrak{D}^{(k)} = \{D_n^{(k)}\}_{n \geq 1}$ of $\mathfrak{C}$-circuits of size at most $\exp((\log n)^{k'})$.[13] We recall the following result from [KKO13], which for convenience we state here as follows.

Lemma 14. *There is a* PSPACE*-complete language L computable in linear space and a constant $b \geq 1$ such that the following holds. If $\mathfrak{C}(s(n))$ is learnable to error and accuracy $\leq n^{-b}$ in time at most $t(n) \geq n$, then either*

(i) $L \notin \mathfrak{C}(s(n))$; or

(ii) $L \in \mathsf{BPTIME}[\mathsf{poly}(t(n))]$.

By amplifying the success probability, we can assume that each family $\mathfrak{D}^{(k)}$ learns with confidence parameter $\delta \leq n^{-b}$, and by the result of [BL93], we can assume without loss of generality that the accuracy parameter is also $\leq n^{-b}$. This implies via Lemma 14 that either there exists no constant $a \geq 1$ such that $L \in \mathfrak{C}(\exp((\log n)^a))$, or for some $a' \geq 1$, we have $L \in \mathsf{BPTIME}[\exp((\log n)^{a'})]$. In the former case, since L is computable in linear space, we get that $\mathsf{BPE} \not\subseteq \mathfrak{C}[\exp((\log n)^{O(1)})]$. On the other hand, in the latter scenario, as $\mathsf{DSPACE}[s'(n)]$ can diagonalize against circuits of size $s'(n)^{\Omega(1)}$ (cf. [OS17a, Corollary 39]), this fact together with a standard padding argument implies that $\mathsf{BPE} \not\subseteq \mathfrak{C}[\exp((\log n)^{O(1)})]$.

Let $f \in \mathsf{BPE}$ be a function that cannot be computed by quasi-polynomial size circuits from $\mathfrak{C}$ on infinitely many input lengths. We claim that the following result holds.

[13] See for instance the proof of Lemma 7 in [OS17a].

Claim 15. *Under our assumptions, there exists a function $f' \in \mathsf{BPTIME}[2^{O(n)}]$ such that for every constant $\beta \geq 1$, on infinitely many input lengths n, any $\mathfrak{C}$-circuit of size $\leq \exp((\log n)^{\beta})$ can compute f'_n with advantage at most $\exp((\log n)^{-\beta})$.*

Indeed, since $\mathcal{TC}^0 \subseteq \mathfrak{C}$, efficient worst-case to average-case reductions can be used to amplify the hardness of f (cf. [GGH+07, GR08, GNW11]). In a bit more detail, a reduction of this form is well-known to hold for functions $f \in \mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$. In order to amplify a function f in BPE, it is enough to observe that the entire truth-table of f can be computed in randomized time $2^{O(n)}$, except with negligible probability. Since the worst-case to average-case reduction acts on truth-tables, it defines with high probability a fixed function f' obtained from f.

Let $f' = \{f'_n\}_{n \geq 1}$ be given by Claim 15, and E be a randomized algorithm running in time $2^{O(n)}$ that prints the truth-table of f'_n with probability at least $1 - 2^{-n}$. We use E together with the Nisan-Wigderson generator [NW94] to pseudodeterministically compute a generator against $\mathfrak{C}$. (While their result is stated with respect to general boolean circuits, it is well-known and easy to check that their construction works for any circuit class containing $\mathcal{TC}^0$.)

Theorem 16 (Corollary of Theorem 1 from [NW94])**.** *Let $m \leq t(m) \leq 2^m$, and suppose there is $h \in \mathsf{DTIME}[2^{O(m)}]$ such that, on infinitely many input lengths, every $\mathfrak{C}$-circuit D_m of size $\leq t(m)$ satisfies $\Pr_{\boldsymbol{x}}[D_m(\boldsymbol{x}) \neq h_m(\boldsymbol{x})] \geq 1/m$. Then there exists a constant $\lambda > 0$ and a quick pseudorandom generator $G\colon \{0,1\}^m \to \{0,1\}^{t(m^\lambda)}$ that $t(m^\lambda)$-fools $\mathcal{C}(t(m^\lambda))$-circuits on infinitely many input lengths.*

Using Theorem 16, it is possible to prove the following result.

Claim 17. *For every constants $c \geq 1$, $k \geq 1$, and $\gamma > 0$, there exists a function $G\colon \{0,1\}^* \times \{0,1\}^{\ell(n)} \to \{0,1\}^n$ that is a quick μ-pseudodeterministic generator that η-fools $\mathfrak{C}$-circuits of size $\leq \exp((\log n)^k)$ on infinitely many input lengths, where $\mu = 2^{-n}$, $\eta = n^{-c}$, and $\ell(n) = n^\gamma$.*

Claim 17 is established by a standard application of the Nisan-Wigerson generator to the family f', adapted to the pseudo-deterministic setting in the natural way.

Finally, using that $\mathfrak{C}$ is closed under composition, the existence of such generators immediately imply the statement of the theorem via an application of Lemma 8. □

Ideally, we would like to obtain a pseudodeterministic learner of comparable running time. However, this does not seem to be possible with these techniques. Consider for instance the more extreme case of designing a sub-exponential time pseudodeterministic learner from a sub-exponential time randomized learner. The main difficulty is that the lower bounds obtained from such a learner are not strong enough to derandomize an algorithm that runs in sub-exponential time.

Our techniques also require a strong assumption on the circuit class, namely, that it is closed under composition and able to compute threshold functions. Since there is evidence that circuit classes containing $\mathcal{TC}^0$ cannot be learned [NR04], it would be extremely interesting to obtain an analogue of Theorem 13 under weaker assumptions. In particular, one might be able to apply such a result to pseudoderandomize existing algorithms, such as [CIKK16].

Two remarks on the self-learnability of weak classes. These results further motivate the study of self-learning circuit classes, a direction that some might find of independent philosophical interest. In other words,

When is a circuit class $\mathfrak{C}$ learnable by algorithms that are no more powerful than $\mathfrak{C}$-circuits?

For very weak classes, this is probably impossible, given the very weak resources available to the learning algorithm, and the fact that a self-learner is in particular a proper learner. However, when $\mathfrak{C}$ becomes stronger, as in the extreme case where $\mathfrak{C} = \mathsf{P/poly}$, if learning algorithms exist then they are automatically proper learners.

It is possible to show that $\mathsf{MAJ} \circ \mathcal{AC}^0$ circuits are self-learnable by a uniform family of sub-exponential size circuits. This follows for instance from the results of [GS10], since the learning algorithm is based on the estimation of fourier coefficients of bounded size, and the corresponding parity computations can be simulated by randomized oracle $\mathcal{AC}^0$ circuits that output a sub-exponential size hypothesis in $\mathsf{MAJ} \circ \mathcal{AC}^0$.[14] Therefore, self-learnability is a phenomenon that is present even in constant-depth classes.

On the other hand, we do not know if $\mathcal{AC}^0$ is self-learnable by quasi-polynomial size $\mathcal{AC}^0$ circuits.[15] A natural approach here is to try to implement the LMN algorithm [LMN93] using $\mathcal{AC}^0$ circuits, perhaps by replacing a threshold gate for an approximate majority, which is known to be computable in this class (see e.g. [Vio14]). However, as we briefly explain next, this and other similar approaches cannot work. A self-learning algorithm of quasi-polynomial complexity for the class $\mathcal{AC}^0$ is required to output a hypothesis that is itself a quasi-polynomial size $\mathcal{AC}^0$ circuit. However, the approach in [LMN93] is also able to learn functions that cannot be approximated by such circuits. This is an immediate consequence of [RST15, Theorem 3] using the connection between the influence of a boolean function and fourier concentration (cf. [O'D14]).

4 Pseudorandomness from pseudodeterministic learning

In this section we explore the construction of pseudorandom objects and the proof of non-uniform circuit lower bounds from the existence of pseudodeterministic learning algorithms. Recall that our learning algorithms make membership queries to the unknown function, and learn under the uniform distribution.

4.1 Deterministic learning algorithms

Our first result concerns deterministic learners. We say that a boolean function g is γ-dense if $\Pr_{\boldsymbol{x} \sim \mathcal{U}_n}[g(\boldsymbol{x}) = 1] \geq \gamma$.

Lemma 18 (Weak hitting set generators from deterministic learners)**.** *Let $\mathfrak{C}$ be any class of functions, and assume there exists a deterministic learning algorithm A for $\mathfrak{C}(s)$ with advantage $\alpha(n) > 0$. If $\gamma = \gamma(n)$ satisfies*

$$\gamma > 1 - 2\alpha, \tag{2}$$

there exists a function $H \colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ with the following properties. Let $Q_n \subseteq \{0,1\}^n$ be the set of queries that A makes to the oracle according to the execution corresponding to $A^{0^{(n)}}(1^n)$,

[14] For the interested reader, we stress that during this implementation the empirical estimate of each fourier coefficient is not computed by the circuit, since $\mathcal{AC}^0$ circuits cannot count. The bits obtained from products of the form $\chi_S(x) \cdot f(x)$ are hard-coded directly into the final hypothesis, which can make use of a single threshold gate to compute the sign function.

[15] Note that, if this were the case, our techniques from Section 3 would provide an alternative, conceptually simpler proof of a result of [Sit95] showing that such circuits can be learned in deterministic quasi-polynomial time.

where $0^{(n)}$ is the identically zero function on n-bit inputs. Moreover, let $q_n = |Q_n|$, and let $t(n)$ be a constructive upper bound on the number of steps used by A until the last such query is determined. Then,

- $H\colon \{1\}^n \times \{0,1\}^{\log q_n} \to \{0,1\}^n$ *is a hitting set generator for the family of γ-dense functions in $\mathcal{C}(s(n))$.*
- *On every $w \in \{0,1\}^{\log q_n}$, $H(1^n, w)$ can be computed in time $\leq t(n)$.*
- *Furthermore, $H(\{1\}^n \times \{0,1\}^{\log q_n}) = Q_n$, and $H(1^n, w)$ is the w-th query made by A according to the computation corresponding to $A^{0^{(n)}}(1^n)$.*

Proof. In order to establish the result, it is enough to argue that the procedure H described in the statement is a hitting set for the family of γ-dense functions in $\mathcal{C}(s(n))$. Recall that A is a deterministic algorithm. Fix $n \in \mathbb{N}$. Suppose there exists a γ-dense function $f \in \mathcal{C}(s(n))$ such that $Q_n \cap f^{-1}(1) = \emptyset$. We use the correctness of the learning algorithm A to derive a contradiction.

Given boolean functions $g_1, g_2\colon \{0,1\}^n \to \{0,1\}$, we let $\mathsf{dist}(g_1, g_2) \stackrel{\text{def}}{=} \Pr_{\boldsymbol{x}}[g_1(\boldsymbol{x}) \neq g_2(\boldsymbol{x})]$ be the relative hamming distance between these functions. Let $h_0 = A^{0^{(n)}}(1^n)$ be the hypothesis output by A on the identically zero function over n input bits, and let $h_f = A^f(1^n)$ be the correspondent hypothesis for f. Using that f is γ-dense and the advantage parameter of the learner, we have:

$$\begin{aligned} \mathsf{dist}(h_0, 0^{(n)}) &\leq 1/2 - \alpha(n), && (3)\\ \mathsf{dist}(h_f, f) &\leq 1/2 - \alpha(n), && (4)\\ \mathsf{dist}(f, 1^{(n)}) &\leq 1 - \gamma(n). && (5) \end{aligned}$$

Under our assumption that Q_n does not hit f, and using that A is deterministic, it follows that the answers provided by the functions $0^{(n)}$ and f to the queries made by A on input 1^n are identical. Consequently,

$$\mathsf{dist}(h_0, h_f) = 0, \qquad (6)$$

i.e., A outputs the same hypothesis on both functions.

Finally, it follows from inequalities 2, 3, 4, 5, and 6 that

$$\begin{aligned} 1 = \mathsf{dist}(0^{(n)}, 1^{(n)}) &\leq \mathsf{dist}(0^{(n)}, h_0) + \mathsf{dist}(h_0, h_f) + \mathsf{dist}(h_f, f) + \mathsf{dist}(f, 1^{(n)})\\ &\leq (1/2 - \alpha(n)) + 0 + (1/2 - \alpha(n)) + (1 - \gamma(n))\\ &= 1 + (1 - \gamma(n)) - 2\alpha(n)\\ &< 1, \end{aligned}$$

which is a contradiction. This completes the proof that H is indeed a hitting set generator with the desired parameters. □

If the learning advantage α is too small in Lemma 18, the hitting set is only guaranteed to hit very dense circuits. This can be easily fixed by the following straightforward construction.

Lemma 19 (Amplification of uniform family of hitting sets)**.** *Let $\mathfrak{C}$ be a class closer under disjunctions, $s(m)$ be a non-decreasing function, and $0 < \zeta'(m) < \beta(m) < 1$. Let $\{H_m\}_{m \geq 1}$ be a*

uniform family of functions $H_m\colon \{0,1\}^{\ell(m)} \to \{0,1\}^m$*, where each* H_m *is a hitting set for* $\beta(m)$*-dense circuits in* $\mathcal{C}(s(m))$*. Furthermore, assume* H_m *is computable in time* $t(m)$*. If* $k'(m)\colon \mathbb{N} \to \mathbb{N}$ *and* $\zeta'\colon \mathbb{N} \to [0,1]$ *are functions satisfying*

$$1 - (1 - \zeta'(m))^{k'(m)} \;\geq\; \beta(m \cdot k'(m)),$$

the following holds. There exists a uniform family $\{H'_m\}_{m\geq 1}$ *of functions* $H'_m\colon \{0,1\}^{\ell'(m)} \to \{0,1\}^m$ *computable in time* $t'(m)$ *such that each* H'_m *is a hitting set for* $\zeta'(m)$*-dense circuits in* $\mathcal{C}(s'(m))$*, where*

$$\ell'(m) = \ell(m \cdot k'(m)) + \log k'(m), \quad t'(m) = O(t(m \cdot k'(m))), \quad \text{and} \quad s'(m) = s(m \cdot k'(m))/k'(m).$$

Proof. We define H'_m as follows. It decomposes its input $x \in \{0,1\}^{\ell'(m)}$ as a string $x = x^1x^2$, where $|x^1| = \ell(m \cdot k'(m))$ and $|x^2| = \log k'(m)$. Let $y = H_{m\cdot k'(m)}(x^1) \in \{0,1\}^{m\cdot k'(m)}$, and decompose $y = y^1 \dots y^{k'(m)}$, where each $|y^j| = m$, $j \in [k'(m)]$. The generator H'_m view x^2 as an index $i \in [k'(m)]$, and outputs y^i. Clearly, the running time of H'_m is as desired.

Consider a $\zeta'(m)$-dense circuit $D' \in \mathcal{C}(s'(m))$ over m-bit inputs. Let $D(z^1, \dots, z^{k'(m)}) = \bigvee_{i=1}^{k'(m)} D'(z^i)$ be the circuit obtained by the disjunction of $k'(m)$ copies of D' over a disjoint set of input variables. Since $\mathfrak{C}$ is closed under disjunctions, D is a $\mathfrak{C}$-circuit of size at most $k'(m) \cdot s'(m) \leq s(m \cdot k'(m))$. Furthermore, since D' is $\zeta'(m)$-dense, the assumption on $k'(m)$ and the definition of D imply that D is $\beta(m \cdot k'(m))$-dense. Consequently, D is hit by some string $w \in \{0,1\}^{m\cdot k'(m)}$ in the support of $H_{m\cdot k'(m)}$. In particular, for $w = w^1 \dots w^{k'(m)}$, there exists $i \in [k'(m)]$ such that the i-th copy of D' in D is hit by $w^i \in \{0,1\}^m$. By construction, w^i is also in the support of H'_m, which shows that D' is hit by H'_m. In other words, for every $m \geq 1$, H'_m hits every $\zeta'(m)$-dense circuit in $\mathcal{C}(s'(m))$, which completes the proof. □

We can therefore get hitting set generators with strong parameters (i.e. hitting sparse circuits and with non-trivial seed length) even from very weak deterministic learning algorithms.

Theorem 20 (Hitting set generators from weak deterministic learning algorithms)**.** *Let* $\mathfrak{C}$ *be a circuit class closed under disjunctions. Suppose there is a deterministic algorithm that learns* $\mathcal{C}(s(n))$ *with advantage* $\alpha(n) > 0$ *in time* $t(n) \geq n$*. Then, for every function* $\zeta^\star(n) > 0$ *for which there is a function* $k^\star(n)$ *satisfying*

$$(1 - \zeta^\star(n))^{k^\star(n)} \;\leq\; \alpha(n \cdot k^\star(n)),$$

there is a uniform family $\{H^\star_n\}_{n\geq 1}$ *of functions* $H^\star_n\colon \{0,1\}^{\ell^\star(n)} \to \{0,1\}^n$ *for which the following holds. Each* $H^\star_n$ *is a quick hitting set generator for the class of* $\zeta^\star(n)$*-dense circuits in* $\mathcal{C}(s^\star(n))$*, where*

$$\ell^\star(n) = \log(t(n \cdot k^\star(n)) \cdot k^\star(n)) \quad \text{and} \quad s^\star(n) = s(n \cdot k^\star(n))/k^\star(n).$$

Proof. Every learning algorithm is a restricted learner for all values of γ. The result is immediate from Lemmas 18 and 19 using the parameter $\beta(n) \stackrel{\text{def}}{=} 1-\alpha(n) > 1-2\alpha(n)$, where we used $\alpha(n) > 0$. Note that the resulting hitting set generator is indeed a quick generator, due to the value of $\ell^\star$. □

We give a concrete application next. Recall that [CIKK16] recently discovered a quasi-polynomial time randomized learning algorithm for $\mathcal{AC}^0[p]$, and that constructing hitting sets and pseudorandom generators against this circuit class is a notorious open problem (see e.g. [FSUV13]). Consequently, a non-trivial derandomization of a much weaker formulation of [CIKK16] would have important consequences in pseudorandomness.

Corollary 21. *If there is a deterministic algorithm that learns $\mathcal{AC}^0[p]$ in sub-exponential time $2^{n^{o(1)}}$ and with an exponentially small advantage $\geq 2^{-n^{1-\varepsilon}}$, where $\varepsilon > 0$ is fixed, then there are hitting sets $H_n \colon \{0,1\}^{\ell(n)} \to \{0,1\}^n$ for $(1/2)$-dense $\mathcal{AC}^0[p]$-circuits with sub-polynomial seed length $\ell(n) = n^{o(1)}$.*

In many cases, learning algorithms are designed in two stages. The first stage explores a technique that provides a learner with small advantage over random guessing, while the second stage employs additional machinery to boost the original learner to a high-accuracy learning algorithm (cf. [Jac97], [CIKK16]). As a consequence of our results, derandomizing the first stage of such algorithms is enough to produce non-trivial hitting set generators.

4.2 Pseudodeterministic learning algorithms

In this short section, we adapt the preceding results to the pseudodeterministic case. We start with an analogue of Lemma 18, stated below as a theorem.

Theorem 22 (Pseudodet. hitting set generators from pseudodet. learners)**.** *Let $\mathfrak{C}$ be any class of functions, suppose $\delta \leq \mu < 1/2$ and $\varepsilon < 1/2$, and assume that there is a $(\varepsilon, \delta, \mu)$-pseudodeterministic learning algorithm A for $\mathfrak{C}(s)$ running in time $t = t(n, \varepsilon, \delta, \mu)$. If $\gamma > 2\varepsilon$, there exists a uniform family $\{H_n\}_{n \geq 1}$ of functions $H_n \colon \{0,1\}^t \times \{0,1\}^{\log t} \to \{0,1\}^n$ that is a quick μ-pseudodeterministic hitting set for γ-dense functions in $\mathcal{C}(s(n))$.*

Proof. The proof is adapted in a natural way. For $w \in \{0,1\}^t$ and $i \in [t]$ represented as a string on $\log t$ bits, $H_n(w, i)$ outputs the i-th query made by the randomized learner A when it computes with oracle access to $0^{(n)}$ and random input string w. Clearly, H_n can be computed in time $O(t)$.

Since A is μ-deterministic, there exists a canonical set of queries $Q_n \subseteq \{0,1\}^n$ and a canonical output hypothesis h_0 such that

$$\Pr_{\boldsymbol{w}}[A^{0^{(n)}}(1^n, \boldsymbol{w}) \text{ queries } 0^{(n)} \text{ exactly over } Q_n \ \text{ and } \ A^{0^{(n)}}(1^n, \boldsymbol{w}) = h_0] \ \geq \ 1 - \mu.$$

Moreover, since $\delta \leq \mu \leq 1/2$, we must have $\mathsf{dist}(h_0, 0^{(n)}) \leq \varepsilon$. This also implies that there is a fixed function $H^\star \colon \{0,1\}^{\log t} \to \{0,1\}^n$ with support Q_n such that $\Pr_{\boldsymbol{w}}[H(\boldsymbol{w}, \cdot) \equiv H^\star] \geq 1 - \mu$. We claim that $H^\star$ is a hitting set for γ-dense functions in $\mathcal{C}(s(n))$.

Suppose not, and that some function f from this class is not hit by $Q_n = H^\star(\{0,1\}^{\log t})$. Then, with probability at least $1 - \mu > 1/2$, the computation of A on f is identical to the computation of A on $0^{(n)}$. If this happens then A^f also outputs the hypothesis h_0 for f. Now if $\mathsf{dist}(h_0, f) \leq \varepsilon$, since f is γ-dense we get that

$$1 \ = \ \mathsf{dist}(0^{(n)}, 1^{(n)}) \ \leq \ \mathsf{dist}(0^{(n)}, h_0) + \mathsf{dist}(h_0, f) + \mathsf{dist}(f, 1^{(n)}) \ \leq \ \varepsilon + \varepsilon + (1 - \gamma) \ < \ 1,$$

where the last inequality used the assumption that $\gamma > 2\varepsilon$. Therefore, we must have $\mathsf{dist}(h_0, f) > \varepsilon$. As this happens with probability strictly larger than $1/2$ and $\delta < 1/2$, we obtain a contradiction to the correctness of A on f. □

We provide a concrete instantiation of Theorem 22, showing a setting where a pseudoderandomization would have interesting consequences. Recall that CNFs can be learned to any accuracy $\varepsilon > 0$ by a randomized polynomial time algorithm [Jac97]. On the other hand, it is a notable open problem to construct pseudorandom generators and hitting set generators with logarithmic seed length for the class of polynomial size CNFs (see [ST17] for a recent related result).

Corollary 23. *If there is a $(\varepsilon, \delta, \mu)$-pseudodeterministic polynomial-time learning algorithm for the class of polynomial size* CNF*s, where $\varepsilon, \delta, \mu \leq 1/10$, then for every $k \geq 1$ there is a μ-pseudodeterministic hitting set generator $H_n \colon \{0,1\}^{\mathsf{poly}(n)} \times \{0,1\}^{O(\log n)} \to \{0,1\}^n$ for the class of 3ε-dense* CNF*s of size at most n^k.*

It is easy to see that an analogue of Lemma 19 holds for pseudodeterministic hitting sets, and as a direct consequence, they can also be obtained from weak pseudodeterministic learning algorithms. We state for reference the following general result, which can be established in a way completely analogous to the proof of Theorem 20.

Theorem 24 (Pseudodet. hitting sets from weak pseudodet. learning)**.** *Let $\mathfrak{C}$ be a circuit class closed under disjunctions. Suppose there is a $\mu(n)$-pseudodeterministic algorithm that learns $\mathcal{C}(s(n))$ with advantage $\alpha(n) > 0$ in time $t(n) \geq n$, where $\mu(n) < 1/2$. Then, for every function $\zeta^\star(n) > 0$ for which there is a function $k^\star(n)$ satisfying*

$$(1 - \zeta^\star(n))^{k^\star(n)} \leq \alpha(n \cdot k^\star(n)),$$

there is a uniform family $\{H^\star_n\}_{n \geq 1}$ of functions $H^\star_n \colon \{0,1\}^* \times \{0,1\}^{\ell^\star(n)} \to \{0,1\}^n$ for which the following holds. Each $H^\star_n$ is a quick $\mu^\star(n)$-pseudodeterministic hitting set generator for the class of $\zeta^\star(n)$-dense circuits in $\mathcal{C}(s^\star(n))$, where

$$\mu^\star(n) = \mu(n \cdot k^\star(n)), \quad \ell^\star(n) = \log(t(n \cdot k^\star(n)) \cdot k^\star(n)), \quad \text{and} \quad s^\star(n) = s(n \cdot k^\star(n))/k^\star(n).$$

The following immediate consequence might be of complexity-theoretic interest.

Corollary 25. *For every constant $\mu < 1/2$, if there is a μ-pseudodeterministic algorithm that learns polynomial size circuits in time $2^{o(n)}$ and with a constant advantage $\alpha > 0$, then there is a quick μ-pseudodeterministic hitting set $H_n \colon \{0,1\}^{t(n)} \times \{0,1\}^{\ell(n)} \to \{0,1\}^n$ for $(1/2)$-dense polynomial size circuits, where $\ell^\star(n) = o(n)$ is the seed-length, and $t(n) = 2^{o(n)}$ is the randomness complexity.*

4.3 Consequences for circuit lower bounds from learning algorithms

It is well-known that hitting set generators can be used to define hard boolean functions computable in deterministic exponential time. We observe next that pseudodeterministic hitting sets provide hard functions computable in randomized exponential time. This will allow us to use our techniques to derive a new result showing that pseudodeterministic learning algorithms yield almost everywhere circuit lower bounds.

Lemma 26 (Lower bounds from pseudodeterministic hitting set)**.** *Let $\mathfrak{C}$ be an arbitrary class of functions. Let $\{H_m\}_{m \geq 1}$ be a quick $(1/3)$-pseudodeterministic hitting set for $(1/2)$-dense circuits in a class $\mathcal{C}(s(m))$, where $H_m \colon \{0,1\}^{t(m)} \times \{0,1\}^{\ell(m)} \to \{0,1\}^m$, $\ell(m) < m$, and $t(m) \leq 2^{O(\ell(m))}$. Then there is f in $\mathsf{BPE} = \mathsf{BPTIME}[2^{O(n)}]$ such that $f_n \notin \mathcal{C}(s(\ell^{-1}(n)))$ for all $n \geq 1$, where $f = \{f_n\}_{n \geq 1}$ and $f_n \colon \{0,1\}^{n+1} \to \{0,1\}$.*

Proof. Given $n \geq 1$, let m be the largest integer such that $n = \ell(m)$, where we assume for simplicity that it exists. We define a probabilistic algorithm B that computes f_n. On an input $x \in \{0,1\}^{n+1}$, it samples a random string $\boldsymbol{w} \sim \{0,1\}^{t(m)}$, and accepts x is and only if x is *not* the $(n+1)$-bit prefix of an m-bit string in $H_m(\boldsymbol{w}, \{0,1\}^{\ell(m)})$.

By construction and using that H_m is quick, A runs in time at most $2^{O(n)}$. Since H_m is (1/3)-pseudodeterministic, it is easy to see that A is a bounded probabilistic algorithm, i.e., it defines a function in $\mathsf{BPTIME}[\cdot]$. Moreover, each f_n is (1/2)-dense, since it is defined over $\{0,1\}^{n+1}$, and there are at most 2^n strings in $H_m(w, \{0,1\}^{\ell(m)})$, for an arbitrary $w \in \{0,1\}^{t(m)}$.

We claim that f_n requires $\mathfrak{C}$-circuits of size larger than $s(\ell^{-1}(n))$. Suppose otherwise, and let D_n be a circuit of such size and over $n+1$ input bits that computes f_n. Then D_n viewed as a circuit over m input variables still computes a (1/2)-dense function (that does not depend on the new variables) of circuit complexity $\leq s(m)$ that is not hit by H_m. This contradicts the hitting set property of the generator, and completes the proof of the lemma. □

Theorem 27 (Lower bounds from non-trivial pseudodeterministic learners)**.** *Let $\mathfrak{C}$ be an arbitrary circuit class. If there is a (1/3)-pseudodeterministic learner that learns $\mathfrak{C}(s(n))$ to accuracy $\varepsilon < 1/4$ and in time $t(n) = o(2^n)$, then there are functions in BPE that require $\mathfrak{C}$-circuits of size $\geq s(n)$ on every input length $n \geq 1$.*

Proof. The result follows from Theorem 22 (using $\gamma = 1/2$) and Lemma 26. □

In comparison to the result from [OS17a], which shows BPE lower bounds from arbitrary randomized learning algorithms, the proof of Theorem 27 is elementary and does not employ advanced machinery from complexity theory. Most importantly, the new result shows that one can get *almost everywhere* lower bounds from randomized learning algorithms operating pseudodeterministically, while [OS17a] only yields infinitely often separations.

The proof of Theorem 27 also applies to the simpler setting of deterministic learners, and in this case it shows lower bounds for a function in $\mathsf{E} = \mathsf{DTIME}[2^{O(n)}]$. This is similar to a result from [KKO13], but there are important differences in the two cases. In [KKO13], the learner is allowed equivalence queries in addition to membership queries. On the other hand, their result assumes that the hypothesis is correct on every input string, while the deterministic formulation of Theorem 27 only requires an approximate hypothesis.[16]

5 Pseudo-derandomizing approximation

In this section, we provide *unconditional* pseudo-derandomizations for approximate counting algorithms, and present connections between (pseudo)deterministic learning and approximate canonization.

5.1 Approximation of integer-valued functions

We need the following simple lemma.

Lemma 28 (Computing the median in logarithmic space)**.** *The median of N integers, each at most N bits long, can be computed in space $O(\log(N))$.*

Proof. The proof uses a simple guess-and-check procedure. Given N integers $X_1, X_2 \ldots X_N$, we guess each one in turn to be the median, and attempt to verify the guess that X_i is the median by counting the number of $X_j, j \neq i$ such that $X_j < X_i$ and the number of $X_j, j \neq i$ such that

[16] Note that one can also get a trade-off between the running time assumption and the learning advantage, by employing Theorem 24 instead of Theorem 22 in the proof of Theorem 27.

$X_j > X_i$. If both these counts are at most $\lfloor N/2 \rfloor$, we know that X_i is the median. If either of them is larger, we move on to the next guess X_{i+1}.

In more detail, we search for the median in (up to) N phases, where phase i corresponds to the guess that X_i is the median. We use an $O(\log(N))$ bit counter to keep track of the current phase i. In phase i, we loop through j from 1 to N, $j \neq i$, and check for each j in logarithmic space whether $X_j < X_i$, $X_j = X_i$ or $X_j > X_i$. We maintain two counters a and b that are initialized to zero at the beginning of each phase – a keeps a running count of the number of $j, j \neq i$ such that $X_j < X_i$ and b keeps a running count of the number of $j, j \neq i$ such that $X_j > X_i$. Thus, if $X_j < X_i$, we increment a by 1; if $X_j > X_i$, we increment b by 1. Note that the comparison of X_i with X_j can be done in space $O(\log(N))$ since X_i and X_j are each at most N bits long. Similarly, incrementing a or b can also be done in space $O(\log(N))$, as each of those counters can be represented in $O(\log(N))$ bits. When we have looped through all j, we check if a and b are both at most $\lfloor N/2 \rfloor$. Again, this check can be done in space $O(\log(N))$, as it involves comparing $O(\log(N))$-bit numbers. If both checks succeed, we copy X_i onto the output tape, otherwise, we increment i and move on to the next phase. As the median exists, some phase with index at most N succeeds. □

We apply Lemma 28 to pseudo-derandomize approximation algorithms in the case that PSPACE collapses to BPP.

Lemma 29 (Pseudo-derandomizing approximation when PSPACE = BPP)**.** *Suppose* PSPACE = BPP*. Then every integer-valued function with a PRAS (resp. FPRAS) has a PPDAS (resp. PFPDAS).*

Proof. We establish the result for an integer-valued function with a PRAS, and then indicate how the proof needs to be modified slightly for the case when the function has an FPRAS.

Let f be an integer-valued function with a PRAS. Fix $\epsilon > 0$. Since f has a PRAS, there is a probabilistic polynomial-time machine M such that for each input x, with probability $1 - 2^{-\Omega(|x|)}$, we have that $(1-\epsilon)f(x) \leq M(x) \leq (1+\epsilon)f(x)$. We can assume without loss of generality that M uses a random sequence of length n^k and halts in time n^k on any input of length n, where k is some constant. Given a string $R \in \{0,1\}^{n^k}$, let $M(x,R)$ denote the output of M on input x of length n and randomness R.

We define a language L as follows: an input $< x, 1^i >$ is in L if $i \leq |x|^k$ and moreover the i'th least significant bit of the median of the $2^{|x|^k}$ integers $M(x,R), R \in \{0,1\}^{|x|^k}$ is 1. We argue that $L \in$ PSPACE, by defining a polynomial-space bounded machine N to decide L. Given input $< x, 1^i >$, N first checks if $i \leq |x|^k$, rejecting if not. If this check succeeds, N simulates the algorithm of Lemma 28 to compute the median of the $2^{|x|^k}$ numbers $M(x,R), R \in \{0,1\}^{|x|^k}$. Each time the algorithm accesses a number X_j, N re-computes and stores $M(x,R_j)$, where R_j is the j'th string of length $|x|^k$, in lexicographic order. Clearly this re-computation and storage can be implemented in polynomial space, as M can be simulated in polynomial time when its random sequence is fixed. The median-finding algorithm runs in space logarithmic in the size of the list it processes, which is polynomial as a function of $|x|$. The simulation ends by computing the median value $g(x)$, which N stores. Finally, N checks if the i'th least significant bit of $g(x)$ is 1, accepting if and only if this check succeeds. It should be clear that N can be implemented in polynomial space overall.

By assumption, PSPACE = BPP, and hence there is a probabilistic polynomial-time machine N' which decides L, with error at most $2^{-|y|}$ on any input y. We use N' to define a polynomial-time pseudo-deterministic approximation scheme M' for f, as follows. On input x, M' simulates N' on

$< x, 1^i >$ for each $i, 1 \leq i \leq |x|^k$. Let $b_i = 0$ if N' rejects on $< x, i >$, and 1 otherwise. M' outputs $b_{|x|^k} \ldots b_1$.

M' can be implemented in polynomial time, since it simulates the polynomial-time machine N' polynomially many times. We need to argue that M' implements a pseudo-deterministic approximation scheme for f. First, using a simple union bound and the fact that the probabilistic machine N' deciding L has error at most $2^{-|y|}$, we have that with probability at least $1 - \text{poly}(|x|)2^{-|x|}$, we get that M' outputs the true median $g(x)$ of $M(x, R), R \in \{0,1\}^{|x|^k}$, and hence satisfies the pseudo-determinism condition. From the fact that M outputs a good approximation with very high probability, it follows that the median $g(x)$ of the values $M(x, R), R \in \{0,1\}^{|x|^k}$ is a good approximation to f, i.e., $(1-\epsilon)f(x) \leq g(x) \leq (1+\epsilon)f(x)$. Thus we conclude that M' implements a polynomial-time pseudo-deterministic approximation scheme for f.

Now we indicate how the argument needs to be changed if the integer-valued function has an FPRAS. In this case, we run the argument as before, but with ϵ given as an extra input to the machines M, M', N and N', and in the definition of the language L. All our previous reasoning continues to be valid, and we conclude that there is a fully polynomial-time pseudo-deterministic approximation scheme for f. □

Note that in the above argument, we do not need any assumption about the complexity of exact computation of the integer-valued function f. The argument works in the more general setting where all we know is that a good probabilistic approximation can be efficiently computed. Indeed, the argument can be shown to hold more generally when the function is real-valued, but we choose not to emphasize this as it would involve us in questions of how to represent real values.

We will require the following uniform hardness-randomness tradeoff, which is essentially due to Trevisan and Vadhan [TV07], in the proof of our main result in this section.

Theorem 30 (Uniform hardness-randomness tradeoff [TV07])**.** *For each constant $c \geq 1$, there is a family of functions $\{G_n^c\}_{n\geq 1}$, where $G_n^c \colon \{0,1\}^n \to \{0,1\}^{n^c}$ is computable in deterministic time 2^n, such that if there is a probabilistic polynomial-time machine A with Boolean output and a constant d for which $\left|\Pr_{\boldsymbol{A},\boldsymbol{x}\sim\mathcal{U}_{n^c}}[\boldsymbol{A}(1^n,\boldsymbol{x})] - \Pr_{\boldsymbol{A},\boldsymbol{z}\sim\mathcal{U}_n}[\boldsymbol{A}(1^n, G_n^c(\boldsymbol{z}))]\right| > 1/n^d$ for all but finitely many n, then $\mathsf{PSPACE} = \mathsf{BPP}$.*

Now we are ready to state, and establish our main lemma.

Lemma 31 (Pseudo-derandomization if no complexity collapse)**.** *Let f be an integer-valued function with a PRAS (resp. FPRAS). Then at least one of the following holds:*

1. $\mathsf{PSPACE} = \mathsf{BPP}$.
2. *For each polynomial-time samplable sequence $\mathfrak{D}$ of distributions and each $\delta > 0$, f has a i.o. deterministic (resp. fully deterministic) approximation scheme over $\mathfrak{D}$ running in time $2^{O(m^\delta)}$.*

Proof. We establish the result for an integer-valued function f with a PRAS, and sketch afterward how to modify the proof to work for f with an FPRAS.

Suppose f has a PRAS, and let $\epsilon > 0$ be arbitrary. We assume without loss of generality that $\epsilon < 1/2$; to see this, note that the machine implementing the approximation scheme for parameter $\epsilon = 1/4$ can also be used for all larger ϵ. Let M be a probabilistic polynomial-time machine

implementing the approximation scheme for f with error parameter $\epsilon' = \epsilon/8$. Let k be a constant such that M uses at most $|x|^k$ random bits and runs in time at most $|x|^k$ on any input x.

Let $\delta > 0$ be any constant. We define a deterministic algorithm B_δ running in time $2^{O(m^\delta)}$ on inputs of length m, and argue that if there exists a polynomial-time samplable sequence $\mathfrak{D}$ of distributions such that B_δ does not compute an ϵ-approximation to f on infinitely many input lengths over $\mathfrak{D}$, then $\mathsf{PSPACE} = \mathsf{BPP}$.

B_δ operates as follows on input x with $|x| = m$. It runs the generator $G^{k/\delta}_{m^\delta}$ given by Theorem 30 on each possible input $z \in \{0,1\}^{m^\delta}$ to obtain 2^{m^δ} outputs R_z, where each R_z is of length m^k. For each R_z, it simulates the probabilistic polynomial-time machine M with random sequence R_z to obtain an output w_z. It computes the median $g(x)$ of the 2^{m^δ} values w_z, and outputs this value.

First we bound the running time of B_δ. By Theorem 30, the generator $G^{k/\delta}_{m^\delta}$ can be computed in time 2^{m^δ} on any input of length m^δ. Thus, the computation of all strings R_z can be done in time 2^{2m^δ}. Simulating M with random sequence R_z can be done in polynomial time for each R_z, and hence cumulatively in time $2^{m^\delta}\,\mathrm{poly}(m)$. Computing the median of 2^{m^δ} values, each of size $\mathrm{poly}(m)$ can be done in time $\mathrm{poly}(m)2^{m^\delta}$. Thus, overall, the running time of B_δ is $2^{O(m^\delta)}$, as promised.

Next we argue that if there exists a polynomial-time samplable sequence $\mathfrak{D}$ of distributions such that B_δ does not compute an ϵ-approximation to f on infinitely many input lengths over $\mathfrak{D}$, then $\mathsf{PSPACE} = \mathsf{BPP}$. This is the main argument in the proof, and crucially uses information about the generator given by Theorem 30.

Suppose there exists a polynomial-time samplable sequence $\mathfrak{D}$ of distributions such that B_δ does not compute an ϵ-approximation to f on infinitely many input lengths over $\mathfrak{D}$. This means that there is a constant b such that for all large enough m, with probability at least $1/m^b$ over $x \sim \mathcal{D}_m$, B_δ outputs a value $g(x)$ such that $g(x) < (1-\epsilon)f(x)$ or $g(x) > (1+\epsilon)f(x)$. In contrast, by assumption on M, we have that with all but exponentially small probability over the randomness of M, $(1-\epsilon/8)f(x) \leq M(x) \leq (1+\epsilon/8)f(x)$. We will use this contrast to define a probabilistic algorithm distinguishing the output of $G^{k/\delta}_{m^\delta}$ from random with noticeable probability, and then use Theorem 30 to conclude that $\mathsf{PSPACE} = \mathsf{BPP}$.

We define a probabilistic algorithm A satisfying the condition specified in Theorem 30 with respect to the generator $\{G^{k/\delta}_{m^\delta}\}$. A operates as follows on input $(1^n, y)$. It first determines m such that $n = m^\delta$ (we assume here and elsewhere that real parameters are truncated to integers when necessary). It checks that y is of length m^k, rejecting if not. It then samples an input x of length m according to the polynomial-time samplable distribution $\mathcal{D}_m$. It simulates M on x with random sequence y to obtain a number w. Note that M uses $|x|^k = m^k$ random bits, and y is of length m^k, hence this simulation can be carried out successfully. A then simulates M *without* a specified random sequence, i.e., using its own internal randomness to obtain a number w'. It then checks if $(1-\epsilon/2)w \leq w' \leq (1+\epsilon/2)w$. If yes, it accepts, otherwise it rejects.

We show that $\left|\Pr_{\boldsymbol{A},\boldsymbol{y}\sim\mathcal{U}_{n^{k/\delta}}}[\boldsymbol{A}(1^n,\boldsymbol{y})] - \Pr_{\boldsymbol{A},\boldsymbol{z}\sim\mathcal{U}_n}[\boldsymbol{A}(1^n, G_n^{k/\delta}(\boldsymbol{z}))]\right| > 1/n^d$ for some constant d and all but finitely many n, which by Theorem 30 implies $\mathsf{PSPACE} = \mathsf{BPP}$. We first show that A rejects with noticeable probability on $(1^n, y)$ when y is chosen randomly from the range of the generator, and then show that A accepts with probability nearly 1 when y is chosen uniformly at random.

First, we estimate the rejection probability of A when run on $(1^n, y)$ for y chosen at random from the output distribution of the generator, and n large enough. Let N be large enough that for all input lengths $m \geq M = N^{1/\delta}$, B_δ fails to compute an ϵ-approximation to f over $\mathfrak{D}$. By the

definition of computing an ϵ-approximation to f over $\mathfrak{D}$, this means that there is a constant b such that for all input lengths $m \geq M$, with probability at least $1/m^b$ over x chosen according to $\mathcal{D}_m$, B_δ outputs a value $g(x)$ such that $g(x) < (1-\epsilon)f(x)$ or $g(x) > (1+\epsilon)f(x)$. Since the value output by B_δ is the median of the values obtained by running M using strings in the range of $G^{k/\delta}$ as random sequence, this implies that for at least half the strings R_z in the range (counting multiplicities), we have that $w_z < (1-\epsilon)f(x)$, or that for at least half the strings R_z in the range, we have that $w_z > (1+\epsilon)f(x)$. Note, on the other hand, that when M is run using a uniformly random sequence, the output w' is between $(1-\epsilon/8)f(x)$ and $(1+\epsilon/8)f(x)$ with all but exponentially small probability, by using the fact that M implements a randomized approximation scheme with parameter $\epsilon/8$. Thus, we have that with probability at least $1/3m^b$ over the randomness of A and the randomness of the seed to $G^{k/\delta}_{m^\delta}$, either the w obtained by A does not satisfy the condition $(1-\epsilon/2)w \leq w'$ (this happens in the case that $w > (1+\epsilon)f(x)$, using the fact that $\epsilon < 1/2$ and the non-negativity of f), or the w obtained by A does not satisfy the condition $w' \leq (1+\epsilon/2)w$ (this happens in the case that $w < (1-\epsilon)f(x)$, again using the bound on ϵ and the non-negativity of f).

In contrast, when A is run on $(1^n, y)$, where y is chosen uniformly at random, we have that with all but exponentially small probability, the values w and w' satisfy $(1-\epsilon/8)f(x) \leq w \leq (1+\epsilon/8)f(x)$ and $(1-\epsilon/8)f(x) \leq w' \leq (1+\epsilon/8)f(x)$. This implies that with all but exponentially small probability, w and w' are within a factor $(1+\epsilon/8)/(1-\epsilon/8)$ of each other, and in particular, using the bound on ϵ and the non-negativity of f, we have that $(1-\epsilon/2)w \leq w' \leq (1+\epsilon/2)w$.

Thus we get that A distinguishes the output of the generator from random with probability at least $1/4m^b$ for $m \geq M$, which is at least $1/4n^d$ for $d = b/\delta$. We conclude that $\mathsf{PSPACE} = \mathsf{BPP}$, applying Theorem 30.

We have shown that if there exist a $\delta > 0$, $\epsilon > 0$ and a polynomial-time samplable sequence $\mathfrak{D}$ of distributions such that B_δ does not compute an ϵ-approximation to f infinitely often over $\mathfrak{D}$, then $\mathsf{PSPACE} = \mathsf{BPP}$, which gives the first item of the lemma. On the other hand, if for each $\delta > 0$, $\epsilon > 0$ and polynomial-time samplable sequence $\mathfrak{D}$ of distributions, B_δ does compute an ϵ-approximation to f infinitely often over D, then we have that for each polynomial-time samplable sequence $\mathfrak{D}$ of distributions and each $\delta > 0$, f has an i.o. deterministic approximation scheme running in time $2^{O(m^\delta)}$ over $\mathfrak{D}$, which gives the second item of the lemma.

We now sketch how to obtain the analogous result for an integer-valued function f with an FPRAS. We run the argument as before, except that we don't fix ϵ in advance, but supply it as a parameter to M and to B_δ. The probabilistic algorithm A used as the distinguisher samples not just an x but also an ϵ, and when it does the simulation of M, it uses the parameter $\epsilon/8$. The rest of the argument works as before. □

Theorem 32 (Unconditional pseudo-derandomization of polynomial-time approximation)**.** *Let f be an integer valued function with a PRAS (resp. FPRAS). Then for each polynomial-time samplable sequence $\mathfrak{D}$ of distributions and for each constant $\delta > 0$, f has an i.o.PDAS (resp. i.o.FPDAS) over $\mathfrak{D}$ running in time $O(2^{n^\delta})$.*

Proof. Suppose f has a PRAS (resp. FPRAS). We use a simple win-win argument. We apply Lemma 31. If the first item in the lemma holds, we use Lemma 29 to conclude that f has a PPDAS (resp. PFPDAS). If the second item holds, then by Lemma 31, we have that for each polynomial-time samplable sequence $\mathfrak{D}$ of distributions and for each constant $\delta > 0$, f has an i.o. deterministic (resp. i.o. fully deterministic) approximation scheme running in time $O(2^{n^\delta})$ over $\mathfrak{D}$. In either case,

for each polynomial-time samplable sequence $\mathfrak{D}$ of distributions and for each constant $\delta > 0$, f has an i.o.PDAS (resp. i.o. FPDAS) running in time $O(2^{n^\delta})$ over $\mathfrak{D}$. □

Corollary 33 (Unconditional pseudo-derandomization of approximation for Permanent)**.** *For each polynomial-time samplable sequence $\mathfrak{D}$ of distributions and each constant $\delta > 0$, the $(0,1)$-Permanent has an i.o.FPDAS over $\mathfrak{D}$ running in time $O(2^{n^\delta})$.*

Corollary 33 follows from Theorem 32 using the FPRAS for $(0,1)$-Permanent due to [JSV04].

5.2 Canonization and approximate canonization

In this section, we discuss connections between learnability and approximate canonization. Our results are inspired by similar ideas from [BGI+12] in the context of deterministic exact learning algorithms.

Proposition 34 (Exact canonization is hard)**.** *Let $\mathfrak{C}$ be any circuit class that contains linear-sized CNFs or DNFs. If $\mathfrak{C}$(poly) has deterministic polynomial-time canonization, then $\mathsf{NP} = \mathsf{P}$. If $\mathfrak{C}$(poly) has pseudo-deterministic polynomial-time canonization, then $\mathsf{NP} \subseteq \mathsf{BPP}$.*

Proof. Let $\mathfrak{C}$ be a circuit class that contains linear-sized CNFs (the case where $\mathfrak{C}$ contains linear-sized DNFs can be treated dually). Assume $\mathfrak{C}$ has deterministic polynomial-time canonization, and let f be a polynomial-time computable canonizing function. We can solve CNF satisfiability for linear-sized CNFs in polynomial time as follows: given a linear-sized CNF ϕ, we compute $f(\phi)$ and $f(\phi_{false})$, where ϕ_{false} is some easily computable unsatisfiable linear-sized CNF on the same number of variables as ϕ. Note that since $\mathfrak{C}$ contains linear-sized CNFs, the canonizing function is applicable to CNFs. If $f(\phi) = f(\phi_{false})$ we output "unsatisfiable", else we output "satisfiable". Clearly this algorithm runs in polynomial time. If ϕ is satisfiable, then ϕ is not equivalent to ϕ_{false}, and hence the canonizing function f gives different outputs on input ϕ and input ϕ_{false}. Consequently, we correctly output "satisfiable". If ϕ is unsatisfiable, then ϕ is equivalent to ϕ_{false}, and hence the canonizing function f gives the same output on inputs ϕ and ϕ_{false}. Consequently, we correctly output "unsatisfiable". Since satisfiability of linear-sized CNFs is NP-complete, we get that $\mathsf{NP} = \mathsf{P}$.

The argument is similar if $\mathfrak{C}$ has pseudo-deterministic polynomial-time canonization. In this case again, we compare $f(\phi)$ and $f(\phi_{false})$ and accept iff they are different, where f is now a canonizing function computable in pseudo-deterministic polynomial time. The only difference from the previous case is that now the procedure is randomized, and is correct with high probability, since the canonical outputs $f(\phi)$ and $f(\phi_{false})$ are both computed correctly with high probability. □

Theorem 35 (Pseudo-deterministic approximate canonization from learning)**.** *Let $\mathfrak{C}$ be an arbitrary circuit class and $\epsilon > 0$ be a parameter. Let $t\colon \mathbb{N} \to \mathbb{N}$ be a time bound, and $s\colon \mathbb{N} \to \mathbb{N}$ be a polynomial-time computable size function. If there is a deterministic learning algorithm which learns $\mathfrak{C}[s(n)]$ to accuracy ϵ in time $t(n)$, then $\mathfrak{C}[s(n)]$ has deterministic ϵ-approximate canonization in time $t(n)s(n)\,\mathrm{poly}(n)$. If there is a $(1/3)$-pseudodeterministic learning algorithm which learns $\mathfrak{C}[s(n)]$ to accuracy ϵ in time $t(n)$, then $\mathfrak{C}[s(n)]$ has pseudo-deterministic ϵ-approximate canonization in time $t(n)s(n)\,\mathrm{poly}(n)$.*

Proof. We first show the result for deterministic learning algorithms, and the result for pseudo-deterministic learning algorithms follows similarly.

Suppose there is a deterministic learning algorithm A which learns $\mathfrak{C}[s(n)]$ to accuracy ϵ in time $t(n)$. We define a deterministic Turing machine M which computes an ϵ-approximate canonization function f for $\mathfrak{C}[s(n)]$ in time $t(n)s(n)\operatorname{poly}(n)$ as follows. Given input C from $\mathfrak{C}$, M checks if the size of C is at most $s(n)$, using the computability of the size function, where n is the number of variables of C. If not, M produces an arbitrary output. If the check succeeds, M simulates the learning algorithm $A(1^n)$. Each time A asks a membership query y, M answers the membership query by evaluating $C(y)$. Each such evaluation takes time at most $s(n)\operatorname{poly}(n)$, since C is of size at most $s(n)$. Eventually, A halts with a hypothesis circuit D. M outputs D. It should be clear that M operates in time $t(n)s(n)\operatorname{poly}(n)$ overall.

We argue that M computes an ϵ-approximate canonization function f for $\mathfrak{C}[s(n)]$. We first argue that the output of M is always the same for two different circuits of size $s(n)$, and next that the output is an ϵ-approximation to the input circuit. To see the first point, note that if two circuits C and C' of size at most $s(n)$ are equivalent, answers to membership queries made by the learning algorithm A are answered the same way in the simulation by M. Also, the same membership queries are made, since A is deterministic. Thus the output hypothesis circuit D is the same irrespective of whether M is run on C or on C'. To argue that the output is an ϵ-approximation to the input circuit C, we just use the fact that A is a correct deterministic learning for $\mathfrak{C}[s(n)]$ to accuracy ϵ. Since C belongs to the class $\mathfrak{C}[s(n)]$, the simulation of A by M produces an output hypothesis that is an ϵ-approximation to C.

The argument for pseudo-deterministic learning algorithms is analogous. The machine M computing a pseudodeterministic ϵ-approximate canonization is now probabilistic. It operates the same way as before, except that it simulates the pseudodeterministic learning algorithm A. The arguments that the same output is obtained with high probability on equivalent circuits C and C', and that a fixed hypothesis circuit D that is an ϵ-approximation to C is output with high probability, are essentially the same as before, using the assumption that A is a $(1/3)$-pseudodeterministic learning algorithm. □

Corollary 36 (Approximate canonization for $\mathcal{AC}^0$)**.** *For each $\epsilon > 0$ and $k \geq 1$, the class $\mathcal{AC}^0(n^k)$ has deterministic ϵ-approximate canonization in quasi-polynomial time.*

Proof. The result follows from Theorem 35 using Sitharam's deterministic quasi-polynomial time algorithm [Sit95] for polynomial-size $\mathcal{AC}^0$. □

Corollary 37. *Let $p \geq 2$ be a fixed prime. The class $\mathcal{AC}^0[p]$ admits approximate canonization under the following assumptions:*

1. *If E requires circuits of size $2^{n^{\Omega(1)}}$ almost everywhere, then for each $\epsilon > 0$ and $k \geq 1$, $\mathcal{AC}^0[p](n^k)$ has deterministic ϵ-approximate canonization in quasi-polynomial time.*

2. *If BPE requires circuits of size $2^{n^{\Omega(1)}}$ almost everywhere, then for each $\epsilon > 0$ and $k \geq 1$, $\mathcal{AC}^0[p](n^k)$ has pseudo-deterministic ϵ-approximate canonization in quasi-polynomial time.*

Proof. The second result follows from Theorem 35 and Corollary 12. The proof of the first result is similar. □

Acknowledgements

We would like to thank Chris Brzuska for bringing [BBF16] to our attention and for discussions related to approximate canonization. We are also grateful to Roei Tell for conversations about learning derandomization. Finally, we thank the anonymous reviewers for comments that improved the presentation and simplified one of our proofs.

References

[AB09] Sanjeev Arora and Boaz Barak. *Complexity Theory: A Modern Approach*. Cambridge University Press, 2009. 8

[BBF16] Zvika Brakerski, Christina Brzuska, and Nils Fleischhacker. On statistically secure obfuscation with approximate correctness. In *International Cryptology Conference (CRYPTO)*, pages 551–578, 2016. 8, 31

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6:1–6:48, 2012. 8, 29

[BL93] Dan Boneh and Richard J. Lipton. Amplification of weak learning under the uniform distribution. In *Conference on Computational Learning Theory (COLT)*, pages 347–351, 1993. 10, 17

[BR17] Andrej Bogdanov and Alon Rosen. Pseudorandom functions: Three decades later. In *Tutorials on the Foundations of Cryptography*, pages 79–158. 2017. 5

[CIKK16] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *Conference on Computational Complexity (CCC)*, pages 10:1–10:24, 2016. 5, 6, 7, 8, 16, 17, 18, 21, 22

[DETT10] Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. Improved pseudorandom generators for depth 2 circuits. In *International Conference on Randomization and Computation* (RANDOM), pages 504–517, 2010. 7

[FK09] Lance Fortnow and Adam R. Klivans. Efficient learning algorithms yield circuit lower bounds. *Journal of Computer and System Sciences*, 75(1):27–36, 2009. 6

[FSUV13] Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9:809–843, 2013. 6, 7, 21

[GG11] Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:136, 2011. 1, 3

[GG17] Shafi Goldwasser and Ofer Grossman. Bipartite perfect matching in pseudo-deterministic NC. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 87:1–87:13, 2017. 3

[GGH+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *Symposium on Theory of Computing (STOC)*, pages 440–449, 2007. 18

[GGH17] Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-deterministic proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:105, 2017. 3

[GNW11] Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR lemma. In *Studies in Complexity and Cryptography*, pages 273–301. 2011. 18

[GR08] Dan Gutfreund and Guy N. Rothblum. The complexity of local list decoding. In *International Workshop on Approximation, Randomization and Combinatorial Optimization (RANDOM-APPROX)*, pages 455–468, 2008. 18

[Gro15] Ofer Grossman. Finding primitive roots pseudo-deterministically. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:207, 2015. 3

[GS10] Parikshit Gopalan and Rocco A. Servedio. Learning and lower bounds for AC^0 with threshold gates. In *International Workshop on Approximation, Randomization, and Combinatorial Optimization (RANDOM-APPROX)*, pages 588–601, 2010. 19

[HH13] Ryan C. Harkins and John M. Hitchcock. Exact learning algorithms, betting games, and circuit lower bounds. *Transactions on Computation Theory (TOCT)*, 5(4):18:1–18:11, 2013. 6

[Hol17] Dhiraj Holden. A note on unconditional subexponential-time pseudo-deterministic algorithms for BPP search problems. ArXiv:1707.05808, 2017. 3

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002. 3

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Symposium on the Theory of Computing (STOC)*, pages 220–229, 1997. 4

[Jac97] Jeffrey C. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55(3):414–440, 1997. 5, 7, 16, 22

[JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, 2004. 7, 11, 29

[Juk12] Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*. Springer, 2012. 8

[KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004. 3

[KKO13] Adam Klivans, Pravesh Kothari, and Igor C. Oliveira. Constructing hard functions using learning algorithms. In *Conference on Computational Complexity (CCC)*, pages 86–97, 2013. 6, 17, 24

[KV94] Michael Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994. 8, 10

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *Journal of the ACM*, 40(3):607–620, 1993. 5, 7, 19

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, 51(2):231–262, 2004. 18

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. 4, 18

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. 19

[OS17a] Igor C. Oliveira and Rahul Santhanam. Conspiracies between learning algorithms, circuit lower bounds, and pseudorandomness. In *Computational Complexity Conference (CCC)*, pages 18:1–18:49, 2017. 5, 6, 11, 17, 24

[OS17b] Igor C. Oliveira and Rahul Santhanam. Pseudodeterministic constructions in subexponential time. In *Symposium on Theory of Computing (STOC)*, pages 665–677, 2017. 3, 5, 7, 15

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997. 13

[RST15] Benjamin Rossman, Rocco A. Servedio, and Li-Yang Tan. An average-case depth hierarchy theorem for boolean circuits. *CoRR*, abs/1504.03398, 2015. 19

[Sit95] Meera Sitharam. Pseudorandom generators and learning algorithms for AC^0. *Computational Complexity*, 5(3/4):248–266, 1995. 5, 7, 19, 30

[SS97] Meera Sitharam and Timothy Straney. Derandomized learning of boolean functions. In *International Conference on Algorithmic Learning Theory (ALT)*, pages 100–115, 1997. 5, 17

[ST17] Rocco A. Servedio and Li-Yang Tan. Deterministic search for CNF satisfying assignments in almost polynomial time. In *Symposium on Foundations of Computer Science (FOCS)*, pages 813–823, 2017. 22

[TV07] Luca Trevisan and Salil Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007. 26

[TX13] Luca Trevisan and Tongke Xue. A derandomized switching lemma and an improved derandomization of AC^0. In *Conference on Computational Complexity* (CCC), pages 242–247, 2013. 7

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003. 4, 9

[Val84] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, pages 1134–1142, 1984. 16

[Vio14] Emanuele Viola. Randomness buys depth for approximate counting. *Computational Complexity*, 23(3):479–508, 2014. 19

[Vol14] Ilya Volkovich. On learning, lower bounds and (un)keeping promises. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 1027–1038, 2014. 6

[Wil13] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013. 3

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il