

Exploring Learnability Between Exact and PAC

Nader H. Bshouty*

Department of Computer Science
Technion

Jeffrey C. Jackson[†]

Mathematics and Computer Science Department
Duquesne University

Christino Tamon

Department of Mathematics and Computer Science
Clarkson University

*Supported by the fund for promotion of research at the Technion, Research no. 120-138.

[†]This material is based upon work supported by the National Science Foundation under Grants No. CCR-9877079 and CCR-0209064.

Corresponding Author:

Christino Tamon

Department of Mathematics and Computer Science

Clarkson University

Potsdam, NY 13699-5815

U.S.A.

Email: tino@clarkson.edu

Tel: 315-268-6521

Fax: 315-268-2371

Abstract

We study a model of *Probably Exactly Correct* (PEXact) learning that can be viewed either as the Exact model (learning from Equivalence Queries only) relaxed so that counterexamples to equivalence queries are distributionally drawn rather than adversarially chosen or as the Probably Approximately Correct (PAC) model strengthened to require a perfect hypothesis. We also introduce a model of Probably Almost Exactly Correct (PAExact) learning that requires a hypothesis with negligible error and thus lies between the PEXact and PAC models. Unlike the Exact and PEXact models, PAExact learning is applicable to classes of functions defined over infinite instance spaces. We obtain a number of separation results between these models. Of particular note are some positive results for efficient parallel learning in the PAExact model, which stand in stark contrast to earlier negative results for efficient parallel Exact learning.

1 Introduction

Consider the following circuit design problem: you are given a representation of some Boolean function f , and you have in mind a target class C of function representations (say DNF, or sum-of-products, representations). You would like to efficiently find a reasonably small (by some measure) $c \in C$ such that $c \equiv f$. Can learning algorithms be applied to this problem?

Obviously, algorithms that produce only approximators to the target f , such as PAC learning algorithms [16], cannot be used for this task. On the other hand, in the traditional model of Exact learning from an equivalence oracle [2], the learning algorithm is presented with adversarially chosen counterexamples to its intermediate hypotheses, which seems to be a “harder” model of learning than is required for our problem. For example, in the circuit design problem, it might be reasonable to expect that counterexamples are chosen randomly according to a simple induced probability distribution over the set of all possible counterexamples.

Thus, we consider a model (introduced by Bshouty) that lies between the PAC and Exact models. The *Probably Exactly Correct* (PEXact) learning model (called PEC in [6]), like the PAC model, allows some chance that the algorithm will fail to find a good representation of the target function. However, unlike the PAC model, PEXact learning has the added requirement that the hypothesis produced by the learning algorithm must be perfect. Alternatively, as already indicated, one may view this as a variant of Angluin’s Exact model [2] in which each counterexample to an equivalence query is chosen randomly rather than maliciously. The PEXact model, then, is intended to lie between these two earlier models, requiring somewhat more of a learning algorithm than the PAC model but somewhat less than the Exact model.

In addition to the potential for applications arising from results in this model, there are significant theoretical reasons for studying it. Strong lower bounds for exact learning in parallel were given by Bshouty [6], who proved that every class that requires $\omega(\log n)$ sequential equivalence queries is not efficiently exactly learnable in parallel. Such classes include monotone conjunctions, monotone DNF formulae, decision trees, and others. Most of these lower bounds rely on adversarial strategies that maliciously choose counterexamples for the equivalence queries. This begs the question of whether these classes are learnable in parallel if distributional counterexamples are available, which led to Bshouty’s original interest in the PEXact model [6].

In fact, other authors have also considered the PEXact model: for example, some results on learning a restricted class of finite automata have previously been obtained [15, 10]. Such results further motivate study of the question of how this model compares with existing well-studied models of learning, in particular the PAC model, Exact learning and Online learning [13].

In this paper we provide some initial comparisons between the PEXact model and these earlier models. First, we give some simple simulation arguments to formalize the intuition that Exact learnability implies PEXact learnability which in turn implies PAC learnability. We also show that if a class is learnable with respect to arbitrary distributions by a deterministic PEXact algorithm (we call this DPEXact learning) then that class is in fact Exact learnable by a deterministic algorithm (a model that we call DEXact).

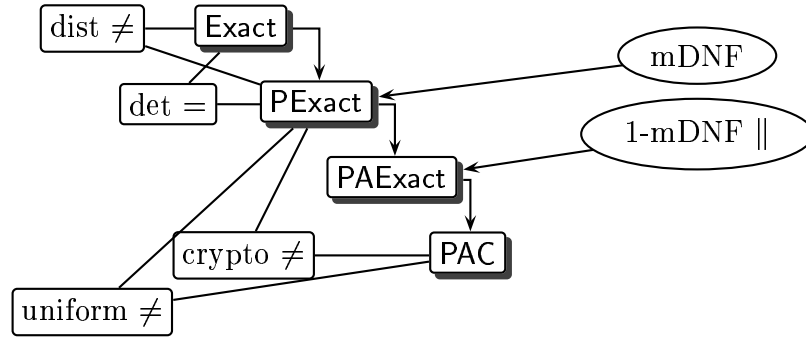


Figure 1: Summary of Results: (a) $\text{Exact} \subseteq \text{PEExact} \subseteq \text{PAExact} \subseteq \text{PAC}$. (b) $\text{DPEExact} = \text{DExact}$. (c) $\text{PAC} \neq \text{PEExact}$ if one-way functions exist. (d) $\text{PAC} \neq \text{PEExact}$ under the uniform distribution (without assumptions, but representation dependent). (e) $\text{Exact} \neq \text{PEExact}$ since monotone DNF is not properly Exact learnable but is properly PEXact learnable under some distribution. (f) monotone Conjunctions are PAExact-learnable (in parallel).

Next, we turn to some separation results. Making no assumptions, we are able to provide a weak (distribution- and representation-specific) separation between the PEXact and PAC models. Also, Blum’s well-known separation between Exact and PAC [4], which is based on the standard cryptographic assumption that one-way functions exist, is adapted in order to more strongly separate the PEXact and PAC models. Furthermore, we show that there is a (contrived) distribution such that monotone DNF is learnable in the PEXact model, which, due to Angluin’s hardness result for monotone DNF in Exact [3], separates PEXact from Exact in a distribution-specific sense.

Finally, we introduce a new model that lies between the PEXact and PAC models. This Probably Almost Exactly Correct (PAExact) model requires that the hypothesis produced by the learning algorithm have negligible (smaller than inverse polynomial) error, which is stronger than the PAC requirement and weaker than the PEXact requirement. In this model we are able to obtain some positive results for efficient learning in parallel, in marked contrast to Bshouty’s profoundly negative results for Exact learning. Figure 1 summarizes the key results of this paper.

It should also be noted that neither the Exact nor the PEXact model applies to learning infinite classes of functions defined over infinite instance spaces. As a simple concrete example, note that we cannot Exact learn an arbitrary interval over the reals from counterexamples alone, even if we are allowed constant probability of failure. The PAExact model, on the other hand, can potentially overcome this limitation by allowing for a negligible amount of error in the hypothesis. Thus, the PAExact model appears to be a particularly good analog of Exact learning for use in infinite instance space settings.

2 Preliminaries

We are interested in learnability between two well-studied models of learning: Valiant’s model of Probably Approximately Correct (PAC) learning [16] and Angluin’s model of Exact learning [2]. In all of the models we consider, some class of Boolean functions C over some instance space X is fixed in advance, and the key question is whether or not any algorithm exists that can “efficiently learn” an arbitrary $f \in C$ given access to an oracle for f . What varies in the models is the definition of “efficiently learn” and the form of the oracle. In all models, we will assume that X and C are actually unions of sets X_n and C_n parameterized by a natural number n . Generally, we assume that for each n , X_n is finite. For example, X will in this paper typically be $\cup_{n=0}^{\infty} \{0, 1\}^n$. Furthermore, we will assume that each C we study has an associated complexity measure that we call the *size* of a function in C . For example, if C_n is the class of all monotone Boolean functions over $\{0, 1\}^n$, then for any $f \in C$, $size(f)$ might be defined as the number of terms in the smallest monotone DNF expression representing f . This complexity measure is useful in giving a reasonable definition of efficient learnability: if we are trying to learn a representation of f as a monotone DNF, then our algorithm should be allowed sample size and run time related in some way to $size(f)$, since at least this much time is required just to write down the solution.

In the PAC model, the learning algorithm is provided with an *example oracle* $EX_{D,f}$. Each query to the example oracle provides the learning algorithm with an *example* $\langle x, f(x) \rangle$, where x is chosen randomly from X_n according to an unknown but fixed probability distribution D over X_n and $f(x)$ is called the *label* of the example. We say that a function class C is *PAC learnable* if there is an algorithm \mathcal{A} (possibly randomized) such that for any $\epsilon, \delta > 0$, any natural number n , any $f \in C_n$ (the *target function*), and any distribution D over X_n , with probability at least $1 - \delta$ over the random choice of examples x and any randomization in \mathcal{A} , algorithm $\mathcal{A}(EX_{D,f}, \epsilon, \delta)$ produces a hypothesis function h such that $\Pr_{x \sim D}[f(x) \neq h(x)] < \epsilon$, and does this with a number of example queries polynomial in n , $size(f)$, $1/\epsilon$, and $1/\delta$. If the run time of \mathcal{A} is also polynomially bounded in these parameters, then C is said to be *efficiently PAC learnable*. If \mathcal{D} is a family of distributions over X containing one D_n for each value of n , then we say that C is *PAC learnable with respect to* (or *under*) \mathcal{D} if C is PAC learnable given that the target distribution is a member of \mathcal{D} . If \mathcal{D} is the family of uniform distributions over X , then we simply say that C is *PAC learnable with respect to the uniform distribution*. If C is PAC learnable and the hypothesis h output by \mathcal{A} is an element of C , then C is said to be *properly PAC learnable*.

In the Exact model, the learning algorithm is provided with an *equivalence oracle* EQ_f . The algorithm queries the oracle by providing a hypothesis function h over the same space X_n for which f is defined. The oracle then either returns “yes”, indicating that f and h are equivalent over X_n , or returns a *counterexample* $x \in X_n$ such that $f(x) \neq h(x)$. We say that a function class C is *Exact learnable* if there is an algorithm \mathcal{A} (possibly randomized) such that for any n , any $\delta > 0$, and any $f \in C_n$, algorithm $\mathcal{A}(EQ_f, \delta)$ produces—with probability at least $1 - \delta$ over the random choices made by \mathcal{A} —a hypothesis function h such that $f \equiv h$, using a number of equivalence queries polynomial in n , $size(f)$, and $1/\delta$. Again, if the run time of \mathcal{A} is also polynomial in these parameters, then C is said to be *efficiently Exact learnable*. If all hypotheses used in queries by \mathcal{A} belong to C , then C is said to be *properly*

Exact learnable. If C is Exact learnable by a deterministic algorithm with a sample size polynomial in n and $size(f)$ alone (δ is not relevant for a deterministic Exact algorithm), then C is said to be *Deterministic Exact* (DEXact) learnable. Angluin’s original definition of exact learning [2] is essentially the model we call DEXact, but for purpose of the comparisons made in this paper we find it useful to allow for randomized learners as well.

Now we formally define the PEXact and PAEXact learning models (to simplify notation, the parameter n is often dropped from X and C from now on and should be clear from context). Let D be any fixed probability distribution over instance space X . The *PEXact equivalence oracle* $EQ_{D,f}$ for a target concept f takes an input hypothesis h from the learner and returns a random counterexample drawn according to the induced distribution of D on $X_{f\Delta h} \stackrel{\text{def}}{=} \{x \in X \mid f(x) \neq h(x)\}$. For any subset $S \subseteq X$, $D(S)$ represents the probability that a random draw x according to D is an element of S .

Definition 1 (*PEXact learnable*)

A concept class C is PEXact learnable under $\mathcal{D} = \{D_n\}_n$ if there is an algorithm \mathcal{A} (possibly randomized) and a polynomial $p(\cdot, \cdot, \cdot)$ such that for each $\delta > 0$ and for each $f \in C$, algorithm \mathcal{A} queries $EQ_{D,f}$ on at most $p(n, \delta^{-1}, size(f))$ equivalence queries and then outputs h so that $D(X_{f\Delta h}) = 0$ with probability at least $1 - \delta$ over the random choice of counterexamples and any randomness in \mathcal{A} .

Definition 2 (*PAEXact learnable*)

A concept class C is PAEXact learnable under $\mathcal{D} = \{D_n\}_n$ if there is an algorithm \mathcal{A} (possibly randomized), a function $q(\cdot, \cdot)$ that is superpolynomial in both of its parameters, and a polynomial $p(\cdot, \cdot, \cdot)$ such that for any $\delta > 0$ and for each $f \in C$, algorithm \mathcal{A} queries $EQ_{D,f}$ on at most $p(n, \delta^{-1}, size(f))$ equivalence queries and then outputs h so that $D(X_{f\Delta h}) \leq 1/q(n, size(f))$ with probability at least $1 - \delta$ over the random choice of counterexamples and any randomness in \mathcal{A} .

The class C is *efficiently* PEXact (PAEXact) learnable with respect to \mathcal{D} if there is a polynomial $p_2(\cdot, \cdot, \cdot)$ such that the running time of the PEXact (PAEXact) algorithm is bounded by $p_2(n, \delta^{-1}, size(f))$. We say that C is (*distribution-free*) PEXact (PAEXact) learnable if there is an algorithm that PEXact (PAEXact) learns C under any distribution family \mathcal{D} .

To illustrate the difference between the adversarial Exact model and the uniform-distribution PEXact model, consider the following simple example of learning monotone conjunctions (expressions of the form $v_{i_1} \wedge v_{i_2} \wedge \dots \wedge v_{i_k}$) over the instance space $\{0, 1\}^n$.

Example 1 Using the standard list-crossoff technique [16], the algorithm first queries the PEXact oracle with the always-0 hypothesis h_0 . This either returns “yes”, in which case the algorithm outputs h_0 and terminates, or it returns a counterexample x which the target labels 1. The algorithm’s next hypothesis, h_1 , is the conjunction of those variables that correspond to 1’s in the counterexample x . The PEXact oracle is then queried with this hypothesis. For each $i > 1$, the hypothesis h_i is formed by taking a conjunction of those variables in h_{i-1} that correspond to 1’s in the counterexample received from the oracle call using h_{i-1} as the hypothesis.

Notice that if variable v_i is irrelevant then each counterexample assigns v_i a 0 value with probability $1/2$. Therefore, for any $\delta > 0$, with probability at least $1 - \delta$ variable v_i will

not be included in the hypothesis produced by the algorithm after $\log(\delta^{-1})$ counterexamples have been seen. Let ℓ represent the number of irrelevant variables in the target conjunction. Then by the union bound, the hypothesis produced after $\log \ell/\delta \leq \log n/\delta$ counterexamples will exactly agree with the target with probability at least $1 - \delta$.

On the other hand, it is easy to see that an adversarial equivalence oracle can force any Exact learner to make ℓ queries. Specifically, on any query with a hypothesis that does not include all relevant variables, the oracle will respond with the vector 1^n of all 1's, which gives no information other than that the function is not the always-0 function. On any query with a hypothesis h that contains all relevant variables and at least one irrelevant variable v_i , a counterexample will be provided that has 1's corresponding to all variables in h except v_i . This allows the algorithm to eliminate one irrelevant variable, but provides no additional information.

In this work, parallel learnability of certain concept classes will also be considered. A *parallel algorithm* is an algorithm that is run simultaneously on multiple machines that are identical except that each processor has a unique identifier. The machines are assumed to have local stores as well as a shared global store. When the machines access oracles, it is assumed that the oracle can respond to all requests simultaneously and, if the oracle is randomized, with independent responses. The sample and time complexities of the parallel algorithm are the maximum sample and time complexities of any single machine. As a simple example, consider an algorithm \mathcal{A} such that whenever \mathcal{A} draws an example from its oracle it also draws $t - 1$ more; that is, \mathcal{A} draws examples in batches of size t . Then a parallel version of this algorithm on t processors can be constructed that will have sample complexity $1/t$ that of \mathcal{A} .

A class C is *efficiently learnable in parallel* in a learning model (PAC, PExact, etc.) if there is a parallel algorithm with polynomially (in all parameters appropriate to the model) many processors learning C in time and sample complexity polylogarithmic in the number of processors. That is, the time and sample complexity must be polynomial in $\log n$, $\log(\text{size}(f))$, and, as applicable to the model, $\log(1/\epsilon)$ and $\log(1/\delta)$. Each oracle call is counted as unit time even if a hypothesis provided to the oracle cannot be computed efficiently in parallel.

3 Containment Results

In this section we give our first results, formally showing containments between various classes. We also show that the deterministic, distribution-free versions of the PExact and Exact models are equivalent.

For a learning model A (e.g., PAC) we use the notation \mathbf{A} (e.g., PAC) to represent the set of all function classes that are learnable in model A . Thus, for two learning models A and B , we use $\mathbf{A} \subseteq \mathbf{B}$ to denote that every class that is learnable in model A is also learnable in model B .

Theorem 1 $\text{Exact} \subseteq \text{PExact} \subseteq \text{PAExact} \subseteq \text{PAC}$.

Proof The relation $\text{Exact} \subseteq \text{PEXact} \subseteq \text{PAExact}$ is immediate. The relation $\text{PAExact} \subseteq \text{PAC}$ is also not hard to see since a PAC simulation of the PAExact algorithm will use the PAC example oracle to provide a random counterexample drawn according to the underlying distribution. This type of rejection sampling will sample according to the induced distribution on the set of counterexamples. Note also that if the PAExact algorithm is efficient then the PAC algorithm will be as well, since failure to find a counterexample after polynomially many examples indicates that with high probability the algorithm has found an adequate hypothesis. \square

The following lemma will be particularly useful for our next result—relating the PEXact and Exact models—as well as later in the paper.

Lemma 2 *Let X_n be an instance set of finite cardinality N and let $x_1 < x_2 < \dots < x_N$ be a total ordering of X_n . Fix a target f over X_n . Let EQ_f be defined as follows: given any hypothesis $h \neq f$, $EQ_f(h)$ returns the first (according to the given total ordering) x_j such that $h(x_j) \neq f(x_j)$. Let h_1, h_2, \dots, h_m be any finite sequence of $m < N$ hypotheses over X_n such that for every h_k there exists a j for which $h_k(x_j) \neq f(x_j)$. Then for any $\delta > 0$, there is a distribution D over X_n that is a function solely of δ and the ordering on X_n with the following property: if the sequence of m hypotheses h_1, h_2, \dots, h_m is queried against both EQ_f and $EQ_{D,f}$ then, with probability at least $1 - \delta$ over the choices made by $EQ_{D,f}$, for all of the m hypotheses h_k , $EQ_{D,f}(h_k)$ produces the same counterexample as $EQ_f(h_k)$.*

Proof For $1 \leq i < N$, define $D(x_i) \stackrel{\text{def}}{=} (\delta/N)^{i-1}(1 - (\delta/N))$, and define $D(x_N) \stackrel{\text{def}}{=} (\delta/N)^{N-1}$. It is easily verified that for any $1 \leq \ell \leq N$, $\sum_{i=\ell}^N D(x_i) = (\delta/N)^{\ell-1}$, since the sum telescopes. Therefore, D is a probability distribution over X_n . Next, for any fixed $1 \leq k \leq m$, let x_j be the counterexample returned by $EQ_f(h_k)$. Then there is some set $S \subseteq \{j+1, \dots, N\}$ such that the probability that $EQ_{D,f}(h_k)$ fails to return the same counterexample x_j is

$$\frac{\sum_{i \in S} D(x_i)}{D(x_j) + \sum_{i \in S} D(x_i)}.$$

Furthermore, for any $1 \leq j \leq N$, this quantity is at most δ/N . To see this, first note that if $EQ_f(h_k) = x_N$ then $j = N$ and $S \subseteq \{j+1, \dots, N\} = \emptyset$. So in this case the numerator of the above quantity is 0. On the other hand, for any fixed $j < N$,

$$\begin{aligned} & \frac{\sum_{i \in S} D(x_i)}{D(x_j) + \sum_{i \in S} D(x_i)} \leq \frac{\delta}{N} \\ \Leftrightarrow & \sum_{i \in S} D(x_i) \leq \frac{\delta}{N} \left(D(x_j) + \sum_{i \in S} D(x_i) \right) \\ \Leftrightarrow & \left(1 - \frac{\delta}{N} \right) \sum_{i \in S} D(x_i) \leq \left(\frac{\delta}{N} \right)^j \left(1 - \frac{\delta}{N} \right) \end{aligned}$$

and this last inequality holds since $\sum_{i \in S} D(x_i) \leq \sum_{i=j+1}^N D(x_i) = (\delta/N)^j$. Therefore, by the union bound, the probability that there exists a k such that $EQ_{D,f}(h_k)$ fails to return

$EQ_f(h_k)$ is at most δ . □

We refer to distributions such as the one defined in Lemma 2 as *stair-step distributions*, since a vertical bar graph of the ordered weights would resemble a set of (irregularly spaced) stairs. Such distributions are of course highly unnatural, but must be allowed in any fully distribution-free model.

Now we show a close relationship between PExact and Exact learning. Define the Deterministic PExact model DPEXact as the PExact model with the requirement that the learner is a deterministic algorithm. That is, the confidence parameter δ is provided only to account for uncertainty inherent in accessing examples through the probability distribution D and not to cover any randomness in the algorithm itself. Similarly, define the Deterministic Exact model DExact as the Exact model with the requirement that the learner is deterministic. Then we have:

Theorem 3 *Let C be any class for which there exists a deterministic algorithm \mathcal{M} that (efficiently) DPEXact learns C . Then C is (efficiently) DExact learnable. That is, in the distribution-free setting,*

$$\text{DPEXact} = \text{DExact}.$$

Proof Consider an algorithm \mathcal{M}' that is identical to \mathcal{M} except that it asks queries in such a way that it prevents the oracle from providing a counterexample x more than once. Specifically, we can think of \mathcal{M}' as intercepting each oracle call made by \mathcal{M} with hypothesis h and evaluating h on each counterexample x_1, x_2, \dots, x_k that \mathcal{M}' has already seen, beginning with the first counterexample received and continuing in the order in which they were received. If h is correct on all of the k previous counterexamples, then \mathcal{M}' calls the oracle with h and returns the counterexample received to \mathcal{M} . Otherwise, the first counterexample for which h produces the wrong label is returned as a counterexample to \mathcal{M} .

We claim that if \mathcal{M} DPEXact learns C , then \mathcal{M}' DExact learns C . For assume otherwise. Then for any polynomial $p(\cdot, \cdot)$ there is some n , $f \in C_n$, and sequence $S = x_1, x_2, \dots, x_m$ of distinct counterexamples such that after $m = p(n, \text{size}(f))$ queries h_1, h_2, \dots, h_m , \mathcal{M}' has still not learned f . Fixing $\delta = 1/4$, we can construct a “stair step” distribution D as described in Lemma 2 such that with probability at least $3/4$, \mathcal{M} learning from $EQ_{D,f}$ sees exactly the same sequence of counterexamples that it sees if it is run as a subroutine of \mathcal{M}' and \mathcal{M}' sees the sequence S . Thus, with probability at least $3/4$, \mathcal{M} fails to learn f in polynomially many examples, which contradicts the assumption that \mathcal{M} DPEXact learns C . □

4 Separation Results

In this section we observe a couple of separation results between the PExact and PAC models. The first separation result shows representation-independent hardness for PExact-learning of a cryptographically-based concept class that is PAC-learnable. This is based on Blum’s work [4] on separating the PAC model from the Exact model.

Theorem 4 *There is a class C that is PAC learnable but not PExact learnable if one-way functions exist.*

Proof

Blum [4] defines a class C with the property that for each n , C_n contains 2^k functions which are defined by applying a fixed transformation to each of the k -bit strings, for $k = \lfloor \sqrt{n} \rfloor - 1$ (the size of each function f in C_n can therefore be defined such that $\text{size}(f) = O(\sqrt{n})$). Blum then proves that if an EQ_f oracle for any $f \in C$ presents its counterexamples in reverse lexicographic order then any DExact learning algorithm for C can be used to “break” a one-way function g employed in the definition of C . In fact, his proof immediately extends to our more general definition of Exact learning that allows the learner to be randomized rather than deterministic. Specifically, it follows that if C can be Exact learned by a randomized algorithm with non-negligible probability then there is a polynomial-time randomized algorithm that breaks g with non-negligible probability, and therefore one-way functions do not exist.

Now we will show that it is also the case that if C is PExact learnable with non-negligible probability then one-way functions do not exist. Assume that there exists a polynomial $p(\cdot, \cdot)$ and a PExact algorithm \mathcal{A} such that for any n , any distribution D over X_n , any $0 < \delta \leq 1$, and any $f \in C_n$, $\tau = p(n, 2/\delta)$ queries are sufficient for \mathcal{A} to learn f with probability at least $1 - \delta/2$, where the probability is over the randomness in \mathcal{A} and in the PExact oracle $EQ_{D,f}$. If we fix any particular $\delta = 1/n^{O(1)}$ and f , by Lemma 2 there exists a particular distribution D —independent of f —such that the probability is at most $\delta/2$ that $EQ_{D,f}$ deviates from EQ_f on any hypothesis query made by \mathcal{A} in the course of learning f , regardless of the sequence of hypotheses chosen by \mathcal{A} .¹ It follows that the probability that \mathcal{A} fails to learn f in τ queries if the queries are made to EQ_f rather than to $EQ_{D,f}$ is at most $(\delta/2)/(1 - \delta/2) \leq \delta$. Therefore, if C is PExact learnable with respect to D with non-negligible probability then C is Exact learnable (by the same algorithm that PExact learns C) with non-negligible probability, implying that one-way functions do not exist.

Finally, Blum also shows that C is PAC learnable [4]. So C yields a separation between the PExact and PAC models. \square

The second separation result shows—without the need for any unproven assumptions—that the PAC model under the uniform distribution is not identical to the PExact model under the uniform distribution when the output of the algorithm is required to belong to a particular representation class. This result builds on the AC^0 learning algorithm of Linial, Mansour, and Nisan [14] and a lower bound of Krause and Pudlák [11].

Theorem 5 *There is a subclass of AC^0 that is efficiently PAC learnable for any constant $\epsilon > 0$ under the uniform distribution producing a hypothesis that is a threshold of parity functions, but this class is not PExact learnable under the uniform distribution using the same representation.*

¹Technically, if the number of queries made by \mathcal{A} is more than $|X_n|$, then Lemma 2 does not apply. However, since $|X_n| = 2^n$ for this C , we can assume that the number of queries is much less than $|X_n|$.

Proof A result Krause and Pudlák shows that there is a function in AC_3^0 (polynomial-size circuits of AND/OR/NOT gates of depth 3) that cannot be represented as a threshold of parity gates containing fewer than exponentially many parities. On the other hand, Linial *et al.* proved that all functions in AC^0 can be *uniformly* approximated by a quasi-polynomial size circuit of threshold of parity gates. This yields a separation between PExact and PAC that is representation dependent. We formalize the details of this separation in the following.

Krause and Pudlák define an AC_3^0 function F on n variables that has size (number of AND/OR/NOT gates) at most $n/3$ and that cannot be represented exactly as a threshold of parity gates containing fewer than $2^{(n/12)^{1/3}}$ gates for sufficiently large n . On the other hand, Linial-Mansour-Nisan show that an $(\epsilon/2)$ -approximation h to any function f computable by an AC^0 circuit of depth d and size M can be PAC learned using a threshold of at most $\binom{\tilde{n}}{t} \leq \tilde{n}^t$ parities, for $t \geq (20 \log_2(4M/\epsilon))^d$, where \tilde{n} is the number of *relevant* variables of f . The number of examples used by the learning algorithm is polynomial in the number of parities and in the other learning parameters.

Now fix any constant $\epsilon > 0$ and consider a Krause-Pudlák function F on n variables of which only

$$\tilde{n} = \frac{3\epsilon}{4} 2^{\log_2^{1/4} n}$$

are relevant variables. Note that the Linial-Mansour-Nisan sign-threshold $(\epsilon/2)$ -approximator requires at most

$$\begin{aligned} \tilde{n}^t &\leq 2^{20^3 (\log_2 \tilde{n}) \log_2^3(4\tilde{n}/(3\epsilon))} \\ &\leq 2^{20^3 \log_2^4(4\tilde{n}/(3\epsilon))} \\ &= n^{20^3} \end{aligned}$$

parity functions. But by the Krause-Pudlák lower bound on F , any exact sign-threshold representation requires at least

$$2^{(\tilde{n}/12)^{1/3}} = \exp\left(\left[2^{\log_2^{1/4} n}\right]^{1/3}\right) = n^{\omega(1)}.$$

□

Next, we show that the PExact and Exact models are distinct with respect to *distribution-specific proper* learning. We do this by showing that there is a distribution under which monotone DNF is properly learnable in the PExact model. That monotone DNF is not Exact properly learnable was shown by Angluin [3].

Theorem 6 *There is a distribution for which monotone DNF is properly PExact learnable.*

Proof Let $|x|$ denote the Hamming weight of (number of 1's in) $x \in \{0, 1\}^n$. Consider the total ordering $<$ over the set $\{0, 1\}^n$ where

$$x < y \quad \text{iff} \quad |x| < |y|, \tag{1}$$

where ties, *i.e.*, whenever $|x| = |y|$, are broken arbitrarily. By Lemma 2, for any $\delta > 0$, there is a distribution D over $\{0, 1\}^n$ (that depends solely on δ and the ordering $<$) such that, with probability at least $1 - \delta$, the PExact oracle $EQ_{D,f}$ and the *ordered* EQ_f oracles return the same counterexamples. The *ordered* EQ_f oracle returns the least counterexample with respect to the ordering $<$.

Input: PExact oracle $EQ_{D,f}$ for monotone DNF f and specific distribution D constructed as a function of unknown $\delta > 0$

Output: Minimal monotone DNF h that is equivalent to target f , with probability at least $1 - \delta$

1. $h \leftarrow \mathbf{false}$
2. **while** $EQ_{D,f}(h)$ produces counterexample x
3. Create term $t_x = \bigwedge_i \{v_i : x_i = 1\}$
4. Add t_x to h
5. **end while**

Figure 2: An algorithm for learning monotone DNF from a PExact oracle.

Now consider the simple algorithm of Figure 2 for learning monotone DNF f from $EQ_{D,f}$ (each v_i represents one of the n variables of f). Notice first that every counterexample will be labeled **true**. If each counterexample is returned by the ordered oracle EQ_f , then it is not hard to verify that each counterexample causes a term from the target to be added to the hypothesis h . So the algorithm succeeds if each of the examples comes from the ordered oracle EQ_f , which occurs with probability at least $1 - \delta$. \square

5 Relaxing Exactness and Parallel Learnability

We obtain several positive results for parallel learnability in the PAExact model; for example, we show that monotone conjunctions are efficiently learnable in parallel in the PAExact model. In contrast, Bshouty [6] had shown that this class is not efficiently Exact learnable in parallel. In fact, all classes currently known to be Exact learnable are not efficiently learnable in parallel in the Exact model. The PAExact model therefore appears to be much more interesting than the Exact model for purposes of studying parallel learnability.

Theorem 7 *The class of monotone conjunctions is efficiently PAExact learnable in parallel.*

Proof Consider the algorithm of Figure 3. Let D be the target distribution and f the target monomial, and let D_0 be the induced distribution obtained by restricting D to positive examples of f . We will say that a hypothesis h_0 is an ϵ -*approximator* to f with respect to D if $\Pr_{x \sim D}[h_0(x) \neq f(x)] \leq \epsilon$. Then the well-known Occam bound for finite hypothesis classes [8] combined with the fact that there are only 2^n monotone conjunctions shows that for any $\epsilon > 0$ and m as defined in the figure, the probability is at most $\delta/(\ln n)$ that if m

Input: Oracle $EQ_{D,f}$, confidence δ , number of parallel processors $t = n \lceil (\ln n)(n \ln 2) + \ln((\ln n)/\delta) \rceil$.

Output: Monotone conjunction T_k such that with probability at least $1 - \delta$, $\Pr_{x \sim D}[T_k(x) \neq f(x)] < 1/n^{\ln \ln n}$.

1. $h \leftarrow \text{false}$
2. $m \leftarrow t/n$
3. Query $EQ_{D,f}(h)$ m times producing counterexamples $\{x^j\}_{j \in [1..m]}$
4. Create term $T_1 = \bigwedge_i \{v_i : \forall j, x_i^j = 1\}$
5. $k \leftarrow 1$
6. **while** $EQ_{D,f}(T_k)$ produces counterexample and $k < \ln n$
7. Query $EQ_{D,f}(T_k)$ m times producing counterexamples $\{x^j\}_{j \in [1..m]}$
8. Create term $T_{k+1} = \bigwedge_i \{v_i : v_i \in T_k \text{ and } \forall j, x_i^j = 1\}$
9. $k \leftarrow k + 1$
10. **end while**

Figure 3: An algorithm for PAExact parallel learning of a monotone conjunction.

examples are chosen according to D_0 there will exist a monotone conjunctive hypothesis h_0 consistent with the examples that is not a $(1/(\ln n))$ -approximator to f (with respect to D_0). Since f is sampled with respect to D_0 by the oracle call at line 3 of the algorithm, and since the hypothesis T_1 produced by the algorithm is consistent with the examples drawn, T_1 is a $(1/(\ln n))$ -approximator to f with respect to D_0 with probability at least $1 - \delta/(\ln n)$. Notice also that by construction T_1 will be consistent with all negative examples of f ; that is, $T_1 \implies f$. Therefore, T_1 is also a $(1/(\ln n))$ -approximator to f with respect to D with the same probability.

Next, notice that $\Pr_{x \sim D}[T_2(x) \neq f(x) \mid T_1(x) = f(x)] = 0$, since $T_1 \implies T_2 \implies f$. Therefore,

$$\Pr_{x \sim D}[T_2(x) \neq f(x)] = \Pr_{x \sim D}[T_2(x) \neq f(x) \mid T_1(x) \neq f(x)] \cdot \Pr_{x \sim D}[T_1(x) \neq f(x)]. \quad (2)$$

Let D_1 represent the induced distribution obtained by restricting D to those instances such that $T_1(x) \neq f(x)$. Then the first term in the product of (2) is just $\Pr_{x \sim D_1}[T_2(x) \neq f(x)]$. Furthermore, $EQ_{D,f}(T_1)$ effectively draws examples of f according to D_1 . Therefore, since T_2 is consistent with these examples by construction, a second application of the Occam argument gives that the first term of the product in (2) is also bounded above by $1/(\ln n)$ with probability at least $1 - \delta/(\ln n)$. This implies that $\Pr_{x \sim D}[T_2(x) \neq f(x)] < (1/\ln(n))^2$ with probability at least $1 - 2\delta/(\ln n)$ by the union bound. More generally, with probability at least $1 - \delta$, the error in T_k at the end of the at most $\ln n$ stages of the algorithm will be at most $(1/\ln n)^{\ln n}$.

We can create T_1 efficiently in parallel by loading each bit x_i^j of each of the m examples onto its own processor. The m processors loaded with bit i from each of the m examples can then compute the AND of these m bits in time $\log m$ using a standard binary tree computation for AND. Computing n parallel AND's of m bits each, the overall time to create

T_1 is $O(\log m)$ and the number of processors used is $mn = t$. This number of processors is also sufficient for the remaining stages of the algorithm, and the total time required is $O((\log m)(\log n))$. \square

A standard reduction argument (see, e.g., [12]) can also be used to show that for any constant k , monotone k -DNF (disjunction of conjunctions of at most k variables) can be learned. A Boolean variable V_i can be defined for each of the $O(n^k)$ conjunctions of at most k of the original variables. A disjunction over this new set of variables V_i is then equivalent to a monotone k -DNF over the original set of variables. Thus, a dual of the above algorithm could be used to learn monotone k -DNF if an oracle over the new set of variables was available. But we can easily simulate such an oracle: given a conjunctive hypothesis $h(V)$ over the new variables, we simply create the corresponding k -DNF hypothesis $h'(v)$ over the original variables such that for every possible assignment to the original variables v , $h'(v) = h(V)$. We then query the original oracle with the hypothesis h' . Thus monotone k -DNF is efficiently parallel PAExact learnable. A similar construction (adding auxiliary variables representing negations of the original literals) can be used to prove that both nonmonotone conjunctions and nonmonotone k -DNF are also efficiently PAExact learnable in parallel.

The above theorem also leads naturally to the following more general observation, which is obtained by simply abstracting the properties of monotone conjunctions used in the proof of the previous theorem:

Theorem 8 *Let function class C have the following properties:*

- *There exists a polynomial $p_1(\cdot)$ such that for all n , $|C_n| \leq 2^{p_1(n)}$.*
- *There is a polynomial $p_2(\cdot)$ and a parallel algorithm \mathcal{A} such that for any $f \in C$ and any $\delta > 0$, if $m = \lceil (\ln n)(p_1(n)(\ln 2) + \ln((\ln n)/\delta)) \rceil$ examples of f are each replicated over $p_2(n)$ processors then \mathcal{A} can produce a hypothesis h consistent with the m examples in time polylogarithmic in n , $\text{size}(f)$, and $1/\delta$.*
- *There is an initial hypothesis h_0 such that for any $f \in C$, if the processors are loaded with m counterexamples to h_0 , then with counterexamples to any hypothesis h_1 consistent with these examples, then with counterexamples to a hypothesis h_2 that is consistent with the second set of examples, and so on, then any sequence h_1, h_2, \dots of hypotheses produced by running \mathcal{A} after each set of counterexamples is loaded will be such that for all inputs x , $(h_i(x) = f(x)) \implies (h_{i+1}(x) = f(x))$ (notice that h_0 is excluded from this requirement).*

Then C is efficiently PAExact learnable in parallel.

Similarly, we can obtain a relationship between PAExact and PAC learning (in parallel or serially):

Theorem 9 *If a class C is efficiently (parallel) PAC learnable with the guarantee that for every target f and distribution D the hypothesis h produced is such that $h \implies f$, then C is efficiently (parallel) PAExact learnable. Furthermore, if C is properly PAC learnable and is closed under union then C is properly PAExact learnable.*

Proof Let \mathcal{A} be an algorithm that efficiently PAC learns C , always producing a hypothesis such that $h \implies f$. The idea is that if we replace calls to the PAC example oracle with calls to an oracle $EQ_{D,f}(h_i)$ for a specific series of logarithmically many hypotheses h_i , then we can drive the error of the PAC algorithm down from ϵ to $\epsilon^{\log n}$.

Specifically, we will first run \mathcal{A} using the oracle $EQ_{D,f}(h_0)$, where h_0 is the always-false hypothesis, and with probability at least $1 - \delta/(\ln n)$ efficiently (in parallel, if \mathcal{A} runs efficiently in parallel) produce a hypothesis h_1 that is an ϵ -approximator to f with respect to the induced distribution D^+ formed by restricting D to positive examples of f . Since the assumption is that \mathcal{A} produces (with high probability) h_1 such that h_1 implies f , h_1 can be assumed to agree perfectly with f over the negative examples of f . Therefore, h_1 is also, with probability at least $1 - \delta/(\ln n)$, an ϵ -approximator to f with respect to D .

Next, \mathcal{A} is run a second time using $EQ_{D,f}(h_1)$ as its example oracle. By reasoning as before, with probability at least $1 - 2\delta/(\ln n)$, \mathcal{A} efficiently produces a hypothesis h_2 such that $h'_2 \stackrel{\text{def}}{=} h_1 \vee h_2$ is an ϵ^2 -approximator to f with respect to D . Setting $\epsilon = 1/n$ and repeating this process $\ln n$ times gives the main result. If \mathcal{A} learns efficiently in parallel then each step can be performed efficiently in parallel and there are $\ln n$ steps, so the resulting algorithm is also an efficient parallel learning algorithm using the same number of processors as \mathcal{A} . Also notice that if C is closed under union and C is properly PAC learnable then all hypotheses made to the PAExact oracle will also belong to C .

□

As an example of an application of this theorem, consider the class of intervals on the real line mentioned in the Introduction. Since this class is PAC learnable by an algorithm that satisfies the requirements of the theorem (see, *e.g.*, [1]), this class is also PAExact learnable.

6 Further Work

While this paper has begun the exploration of learning models that lie between the PAC and Exact models, it also leaves a number of interesting questions open. Two key questions are:

- Are the PAC and PAExact models equivalent? Since the publication of the conference version of this work, Bshouty and Gavinsky [7] have answered this open question, using a boosting approach to show that (in a distribution-free sense) the models are equivalent.
- Are the Exact and distribution-free, randomized PExact models equivalent?

Acknowledgments

The two referees of the journal version of this paper provided numerous helpful questions and comments that resulted in significant improvements.

References

- [1] Martin Anthony, Norman Biggs. *Computational Learning Theory*. Cambridge University Press, 1992.
- [2] Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2:319-342, 1988.
- [3] Dana Angluin. Negative Results for Equivalence Queries. *Machine Learning*, 5:121-150, 1990.
- [4] Avrim Blum. Separating Distribution-Free and Mistake-Bound Learning Models over the Boolean Domain. *SIAM Journal on Computing*, 23(5):990-1000, 1994.
- [5] Nader H. Bshouty. Towards the Learnability of DNF Formulae. *Proceedings of the ACM Annual Symposium on Theory of Computing*, 131-140, 1996.
- [6] Nader H. Bshouty. Exact Learning of Formulas in Parallel. *Machine Learning*, 26:25-41, 1997.
- [7] Nader H. Bshouty and Dmitry Gavinsky. PAC = PAExact and other Equivalent Models in Learning. *Proceedings of the 43rd Annual Symposium on the Foundations of Computer Science*, 167-176, 2002.
- [8] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377-380, 1987.
- [9] Shai Ben-David, Eyal Kushilevitz, and Yishay Mansour. Online Learning versus Offline Learning. *Machine Learning*, 29:45-63, 1997.
- [10] Francois Denis. Learning Regular Languages from Simple Positive Examples. *Machine Learning*, 44(1/2):37-66, 2001.
- [11] Matthias Krause, Pavel Pudlak. On the Computational Power of Depth 2 Circuits with Threshold and Modulo Gates. *Theoretical Computer Science*, 174:137-156, 1997.
- [12] Michael J. Kearns, Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.
- [13] Nick Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning*, 2:285-318, 1988.
- [14] Nathan Linial, Yishay Mansour, Noam Nisan. Constant Depth Circuits, Fourier Transform, and Learnability. *Journal of the Association for Computing Machinery*, 40(3):607-620, 1993.
- [15] Rajesh Parekh and Vasant Honavar. Simple DFA are polynomially probably exactly learnable from simple examples. *Proceedings of the 16th International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, 298-306, 1999.

- [16] Leslie G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134-1142, 1984.