Learning to create is as hard as learning to appreciate

David Xiao *

Abstract

We explore the relationship between a natural notion of unsupervised learning studied by Kearns et al. (STOC '94), which we call here "learning to create" (LTC), and the standard PAC model of Valiant (CACM '84), which is a form of supervised learning and can be thought of as a formalization of "learning to appreciate". Our main theorem states that "if learning to appreciate is hard, then so is learning to create". That is, we prove that if PAC learning with respect to efficiently samplable input distributions is hard, then solving the LTC problem is also hard. We also investigate ways in which our result are tight.

1 Introduction

The **P** vs. **NP** question is often cast in the intuitively appealing language of "creativity" and whether "creativity can be automated" (see *e.g.* the survey of Wigderson [28]). To explain this view, one often uses as an analogy a great artist, say Beethoven, who produced widely appreciated works of music. We model the process of deciding whether a piece of music is pleasing by an efficient circuit f_{music}. Then the process of creating pleasing music amounts to finding a satisfying assignment to f_{music}, while appreciating music only requires evaluating f_{music} on a given input.[1] Therefore, if $\mathbf{P} = \mathbf{NP}$, one can automate the task of composing pleasing music because there would be an efficient algorithm that found pleasing pieces of music (*i.e.* the satisfying assignments of f_{music}).

The above analogy is not the only way one can view creativity through a computational lens. In this paper we explore the question of automating creativity from a learning-theoretic point of view. We will explore two models of learning that correspond to "learning to appreciate" and "learning to create". Both the studied models are standard: the first is the PAC model of Valiant [27], while the second is a form of unsupervised learning, which we call "learning to create" (LTC), whose complexity-theoretic study was initiated by Kearns et al. [16].

PAC learning is a "label prediction" problem, and in particular is a form of *supervised* learning. In the PAC model, the learning algorithm is given examples labelled according to a hidden function f and is supposed to learn how to label new examples as f would. For example, we might try to learn how a particular person Alice appreciates music. In this case, we model Alice's taste by a function $f^{\mathsf{Alice}}_{\mathsf{music}}$ that takes input a piece of music and outputs whether or not Alice finds it pleasing. The learning task would be, given a set of examples of music each labelled $f^{\mathsf{Alice}}_{\mathsf{music}}$, to output a hypothesis that labels new pieces of music the same way as $f^{\mathsf{Alice}}_{\mathsf{music}}$ would. Of course, Alice's taste may be very different from another person Bob, so the same learning algorithm should successfully learn f^{p}_{music} for *all* persons p, given examples labelled according to f^{p}_{music}.

LTC is a "pattern reconstruction " problem, and in particular is a form of *unsupervised* learning. In the LTC model, the learning algorithm is given many unlabelled examples drawn from a hidden distribution D, and is supposed to construct a circuit that generates new examples that are

*LRI, Université Paris-Sud, dxiao@lri.fr

[1] Of course the analogy is not entirely accurate since we believe that $\mathbf{P} \neq \mathbf{NP}$ while, presumably, Beethoven was bound by the Extended Church-Turing Hypothesis and could not solve **NP**-hard problems. But let us ignore this detail and suppose that Beethoven's creativity was indeed the result of solving an **NP**-hard problem.

distributed close to D. One can think of D as say generating a piece of music as Beethoven would. Of course, the Beethoven's distribution of music $D_{\mathsf{Beethoven}}$ is very different from, say, $D_{\mathsf{John\ Lennon}}$, and the learning algorithm should learn using examples how to produce music according to one style or the other.

In this paper, when we refer to the PAC or LTC problem, we mean solving these problems for "complete" concept classes (unless we specifically say otherwise). For instance, we study whether it is possible to PAC learn all labellings computable by $\mathsf{SIZE}(n^2)$ circuits, and whether it is possible to solve LTC for the class of distributions samplable by $\mathsf{SIZE}(n^2)$ circuits.

Our first result says roughly that if PAC is hard, then so is LTC.

Theorem 1.1 (LTC is as hard as PAC learning, informal)**.** *If the PAC learning problem with respect to efficiently samplable input distributions cannot be solved by a polynomial-time algorithm, then the LTC problem cannot be solved by a polynomial-time algorithm.*

This theorem holds even for the more stringent requirement of agnostic learning. In addition to our analogy about learning to create vs. learning to appreciate, one can also interpret our result as saying that "unsupervised learning is as hard as supervised learning" in the context of these particular models.

One may ask whether Theorem 1.1 can be strengthened to say that *for every concept class* $\mathcal{F}$, if it is hard to PAC learn $\mathcal{F}$, then it is also hard to solve LTC for the class $\mathcal{F}$. [2] This is a stronger statement than Theorem 1.1, since, as we will see, the proof of Theorem 1.1 will take a class $\mathcal{F}$ that is hard in the PAC model and transform it into a (more complex) class $\mathcal{F}'$ that is hard in the LTC model. We show that this stronger statement is false by exhibiting concrete concept classes for which it does not hold.

Theorem 1.2 (PAC vs. LTC for specific concept classes, informal)**.** *Under standard cryptographic assumptions, there exist concept classes for which PAC learning (even with respect to the uniform input distribution) is hard while LTC is easy.*

Another weakness in Theorem 1.1 is that it considers only PAC learning with respect to efficiently samplable input distributions. In general, PAC learning allows the examples given to the learning algorithm to be generated from *any* distribution; Theorem 1.1 adds the restriction that the examples must be generated from a distribution that can be sampled by, say, a $\mathsf{SIZE}(n^2)$ circuit.

In our last result, we study whether Theorem 1.1 can be generalized to encompass PAC learning with respect to *unsamplable* input distributions. We show this is unlikely: we exhibit an oracle relative to which LTC is easy (and therefore PAC learning is easy for all efficiently samplable input distributions), but there exist functions that are hard to learn with respect to an unsamplable distribution.

Theorem 1.3 (Separating LTC and PAC learning unsamplable input distributions)**.** *There exists an oracle* $\mathcal{O}$ *relative to which solving LTC for* $\mathsf{SIZE}^{\mathcal{O}}(n^2)$ *is easy, while there is a function* $f \in \mathsf{SIZE}^{\mathcal{O}}(n^2)$ *that is an efficiently computable function that is hard to PAC learn on an unsamplable input distribution.*

1.1 Relation to previous work

Hardness of learning. It is widely believed that both the PAC and LTC problems are hard. Both problems can be proven hard if one assumes the existence of cryptography [27, 17] or the weaker assumption that zero knowledge is non-trivial [1] (see also Corollary 2.11). However, to the best of our knowledge, prior to this work there were no results establishing a relationship *between* the hardness of PAC learning and the LTC problem.

Previous work on complexity of LTC. The computational complexity of PAC learning has been studied extensively since its first appearance [27, 15, 22, 17, 4]. The complexity of LTC was first studied in [16] but overall is less well-understood. One question about LTC that has received some attention is its relation to a related notion of "learning to evaluate probabilities": given samples as in the LTC problem, construct a hypothesis $h : \{0,1\}^n \to [0,1]$ such that $h(x) = \Pr[D = x]$, *i.e.* the hypothesis evaluates the probability that x is drawn from D. It is known that under reasonable assumptions, there is a concrete concept class for which "learning to evaluate probabilities" is hard and LTC is easy [16], while there is also a concept class for which "learning to evaluate probabilities" is easy yet LTC is hard [20].

[2] Since LTC deals with classes of functions with multi-bit outputs while PAC deals with classes of functions with single-bit outputs, it must be clarified how we obtain both single- and multi-bit functions from a single class $\mathcal{F}$. We defer this discussion to Section 4.

Complexity of learning via complete problems. Much of the literature comparing different learning models has focused on *concrete* problems. In this kind of comparison, one exhibits a single concept class that is learnable in one model but not in another (under some reasonable complexity assumption). This has led to valuable insights into the complexity of various models, including the power of membership oracles [5], faulty vs. perfect membership oracles [25, 6], and the aforementioned difference between learning to evaluate probabilities and LTC [16, 20]. Furthermore, this kind of comparison is perhaps the most reasonable when comparing models where one model is obviously more complex than the other, and the goal is to show that this relationship is in some sense strict (*e.g.* [5], where clearly having a membership oracle makes the learning task easier).

On the other hand, this approach has the drawback that in some cases it can lead to conflicting evidence, as in the case of learning to evaluate probabilities and LTC, where looking at one concept class suggests that the learning in the first model is harder than in the second model, but looking at a different concept class suggests the opposite.

A different approach to understanding the complexity of the learning model is to examine a *complete problem* for the model. Namely, if we consider the problem of learning a "complete" concept class (*e.g.* $\mathsf{SIZE}(n^2)$ circuits) in learning model M, then the existence of any hard-to-learn concept class for M would imply that learning $\mathsf{SIZE}(n^2)$ is hard. This approach was explored in Applebaum et al. [1], Xiao [29, 30] to understand the complexity of PAC learning relative to the complexity of **NP**, auxiliary-input one-way functions [21], and zero knowledge. In this paper we apply this approach to the complete problems for PAC learning and for LTC.

PAC learning with respect to efficient distributions. We already noted that Theorem 1.1 only relates LTC to PAC learning or agnostic learning with respect to efficiently samplable input distributions. We believe that this is a reasonable restriction: according to the strong Church-Turing thesis, all physical phenomena can be explained by efficient (polynomial-time) computation. Therefore, one can suppose that from whatever source one obtains the examples to be learned, if one can suppose that they are i.i.d. samples from a distribution then one may as well suppose that this distribution is polynomial-time samplable. Furthermore, Theorem 1.3 says that no relativizing reduction can strengthen Theorem 1.1 to include PAC learning with respect to unsamplable distributions.

1.2 Our techniques

1.2.1 Proving Theorem 1.1

Cryptography using circuits. Our Theorem 1.1 is proven using a variation of standard cryptographic techniques. Standard cryptography is based on *uniform* cryptographic primitives: for example, one-way functions or pseudo-random functions that are computable using one Turing Machine for all input lengths. This is because in order to use these primitives, one needs an efficient way to compute them for any desired input size. On the other hand, it is often required that these primitives are hard-to-break even for non-uniform families of polynomial-size circuits, since it may happen that the adversary has some side-information about the cryptosystem that is best modelled as non-uniform advice.

Because we are looking at things from a learning-theoretic point of view, we consider analogues of one-way functions and pseudorandom functions that are computable by circuits (with non-uniform advice), but which are only required to be secure against uniform adversaries. Such primitives, which we call one-way circuits or pseudorandom circuits in this paper, were studied in the context of zero knowledge, first in Ostrovsky and Wigderson [21] and later in Vadhan [26], who used a slightly different definition. (They were called *auxiliary-input one-way functions* in these contexts.) The connection between one-way circuits and learning theory (also linking learning theory to zero knowledge) was explored in Applebaum et al. [1], Xiao [29, 30]. The main fact about one-way circuits we use to prove Theorem 1.1 is that if one-way circuits exist, then the LTC problem is hard (Corollary 2.11).

Circuit agnostic learning. Another ingredient in the proof of Theorem 1.1 is a variation of the standard PAC/agnostic learning problem that we call circuit agnostic learning, implicit in [1] and made explicit in [30] (see Definition 3.1). Whereas in the standard PAC/agnostic learning models, the learning algorithm only receives a collection of labelled examples, in the circuit agnostic learning model the learning algorithm is given a circuit that samples the distribution of labelled examples. Notice that this does not trivialize the problem: knowing a circuit that *generates* labelled examples does not necessarily reveal how to *label* examples. See Section 3.1 for more discussion.

We know that if one-way circuits exist, then PAC learning is hard [1], but the converse is unknown to hold, namely it is unknown whether the hardness of PAC learning implies that one-way circuits exists (and in fact, it was shown in [29] that this converse cannot be proven by relativizing

techniques). However, the converse *does* hold if we consider circuit agnostic learning: if PAC learning is hard, then the circuit agnostic learning problem is hard (see Lemma 3.2). This is the second main ingredient that we use in the proof of Theorem 1.1.

Proof idea behind Theorem 1.1. Combining the above two tools, the proof idea for Theorem 1.1 is to use a LTC algorithm to solve PAC learning with respect to efficiently samplable distributions as follows. First, obtain a set of labelled examples, and use the LTC algorithm on this set of labelled examples to obtain a circuit C that samples the distribution from which these labelled examples were drawn. Note that such a circuit exists because we are promised that the PAC learning instance is on an input distribution that is efficiently samplable. This effectively transforms the original PAC learning problem into an instance of the circuit agnostic learning problem. Next, combining our two theorems about one-way circuits (Corollary 2.11) and circuit agnostic learning (Lemma 3.2), it is possible to use the LTC algorithm to solve this circuit agnostic learning problem, which in turn solves the original PAC learning problem as well.

1.2.2 Understanding Theorem 1.3

Oracle separations. Theorems such as Theorem 1.3 are called *oracle separations*, and have long been used in theoretical computer science to "separate" various classes. Baker et al. [2] showed an oracle separation between **P** and **NP** and Impagliazzo and Rudich [14] showed an oracle separation between one-way permutations and the intuitively harder task of key exchange. More recently, such oracle separations were also used in learning theory to separate PAC learning from the hardness of zero knowledge [29].

The motivation behind such oracle separations is as follows. There are very few unconditional separations in theoretical computer science (and almost non-existent when going beyond weak complexity classes such as $\mathsf{AC0}$). On the other hand, many if not most complexity results are proved using relativizing techniques (for example black-box reductions and diagonalization). Therefore by proving an oracle separation between classes A and B, one shows that in order to prove A reduces to B, one would need to come up with a non-relativizing technique. This arguably attests to the difficulty of the task.[3] In this spirit, we interpret Theorem 1.3 to mean that strengthening Theorem 1.1 to encompass PAC learning even with respect to unsamplable distributions would require new non-relativizing techniques.

2 Preliminaries

2.1 Notation and basic lemmas

If X is a probability distribution, then let $\mathrm{supp}(X)$ denote the support of X, *i.e.* the values that X takes on with positive probability. For two distributions X_1, X_2, the total variation distance (or statistical distance) is defined as $\Delta(X_1, X_2) = \frac{1}{2}\sum_x |\Pr[X_1 = x] - \Pr[X_2 = x]|$, where the sum is taken over all x in the supports of X_1, X_2. We let U_n denote the uniform distribution $\{0,1\}^n$. For finite sets S, we will sometimes abuse notation and let S also stand for the uniform distribution over S. For a circuit $C : \{0,1\}^m \to \{0,1\}^n$, we say that C samples a distribution X if $C(U_m) = X$. We let $\mathsf{SIZE}(q(n))$ denote the set of (functions computable by) circuits of size $q(n)$, and we let $\mathsf{SIZE}^{\mathcal{O}}(q(n))$ denote the same where the circuits are allowed oracle gates $\mathcal{O}$ at unit cost.

We will use the following lemma of Borel-Cantelli, which says that if a countable sequence of events each have small probability of occuring, then the probability that an infinite number of them occurs is 0.

Theorem 2.1 (Borel-Cantelli lemma)**.** *Let $\{E_n\}_{n\in\mathbb{N}}$ be a sequence of events. Suppose $\sum_{n=1}^{\infty} \Pr[E_n]$ exists and is finite. Then $\Pr[\exists I \subseteq \mathbb{N}, |I| = \infty,\ \forall n \in I,\ E_n \text{ occurs}] = 0$.*

2.2 PAC learning (or learning to appreciate)

Define the learning error of a function $f : \{0,1\}^n \to \{0,1\}$ with respect to a distribution (X, Y) over $\{0,1\}^{n+1}$ to be $\mathsf{err}((X,Y), f) = \Pr_{X,Y}[f(X) \neq Y]$. For a class of functions $\mathcal{F}$, define $\mathsf{err}((X,Y), \mathcal{F}) = \min_{f\in\mathcal{F}} \mathsf{err}((X,Y), f)$.

Definition 2.2 (PAC Learning)**.** An algorithm A PAC learns the concept class $\mathcal{F}$ if the following holds for every $n \in \mathbb{N}, \varepsilon > 0$, $f \in \mathcal{F}$, and every distribution X over $\{0,1\}^n$. Given access to an example oracle that generates labelled examples according to $(X, f(X))$, A produces with success probability $\geq 1 - 2^{-n}$ an ε-good hypothesis h (represented as a circuit), namely $\mathsf{err}((X, f(X)), h) \leq \varepsilon$. Furthermore, A runs in time $\mathrm{poly}(n, 1/\varepsilon)$.

[3] Of course, such a result does *not* imply that the task is impossible. Indeed, many of the most surprising results bypass oracle separations or other notions of separation (for example [19, 24, 3]).

Definition 2.3. We say that A learns $\mathcal{F}$ *w.r.t. efficient distributions* if the PAC learning guarantee is only required to hold for all $X = C(U_m)$, where $m = \text{poly}(n)$ and C is a polynomial-size circuit.

Worst possible error is $1/2$*:* we will assume that A always outputs a hypothesis h such that $\mathsf{err}((X, f(X)), h) \leq 1/2$. One can assure that this occurs with probability $\geq 1 - 2^{-n}$ by checking whether the majority of labels output by h agrees with the majority on the examples drawn from the oracle, and if they disagree outputting $1 - h$ instead of h.

A word on padding: In this paper we use $\mathsf{SIZE}(n^2)$, the class of functions computable by size n^2 circuits, as a complete concept class. This class is complete for functions computable by polynomial-size circuits because of a padding argument. For example, in order to learn circuits of size n^c, it suffices to pad an example x of length n to $x0^{n^{c/2}-n}$ which has length $n' = n^{c/2}$, and then running the learner for $\mathsf{SIZE}(n^2)$. This same kind of padding works for the LTC setting defined below.

Agnostic learning: we will also work with an even more demanding notion of learning, called *agnostic learning*, defined in Kearns et al. [17], where the examples may not be labelled according to any fixed function. Here, the goal is to obtain a hypothesis that performs (almost) as well as the best hypothesis in a concept class.

Definition 2.4 (Agnostic Learning)**.** A procedure A agnostically learns the concept class $\mathcal{F}$ if the following holds for every $n \in \mathbb{N}, \varepsilon > 0$, and every distribution (X, Y) over $\{0,1\}^{n+1}$. Given access to an example oracle that generates labelled examples according to (X, Y), A produces with success probability $\geq 1 - 2^{-n}$ an ε-good hypothesis h (represented as a circuit), namely $\mathsf{err}((X, Y), h) \leq \mathsf{err}((X, Y), \mathcal{F}) + \varepsilon$. Furthermore, A runs in time $\text{poly}(n, 1/\varepsilon)$.

Definition 2.5. We say that A agnostically learns $\mathcal{F}$ *w.r.t. efficient distributions* if the agnostic learning guarantee holds for all $(X, Y) = C(U_m)$, where $m = \text{poly}(n)$ and C is a polynomial-size circuit.

2.3 Learning to create

Definition 2.6 (LTC)**.** A procedure A *solves LTC* for the concept class $\mathcal{F}$ if the following holds for every $n \in \mathbb{N}, \varepsilon > 0$, and $f \in \mathcal{F}$, where $f : \{0,1\}^m \to \{0,1\}^n$. Given access to an oracle that generates samples according to $f(U_m)$, A outputs with probability $1 - \varepsilon$ an ε-close hypothesis $h : \{0,1\}^{m'} \to \{0,1\}^n$ (represented as a circuit), namely $\Delta(f(U_m), h(U_{m'})) \leq \varepsilon$. [4] Furthermore, A runs in time $\text{poly}(n, 1/\varepsilon)$.

The accuracy of the hypotheses: our definition of PAC learning requires a strong notion of accuracy: for an example oracle (X, Y), we require the hypothesis h to satisfy $\mathsf{err}((X, Y), h) \leq 1/\text{poly}(n)$. In the PAC model we know one may apply Boosting [23, 7, 8] to show that "strong PAC learning" is equivalent to "weak PAC learning", where the hypothesis is only required to satisfy $\mathsf{err}((X, Y), h) \leq \frac{1}{2} - 1/\text{poly}(n)$. In contrast, there is no known equivalent boosting technique for the LTC problem, so we must acknowledge that our definition that $\Delta(f(U_m), h(U_{m'})) \leq 1/\text{poly}(n)$ is indeed a strong requirement, and this is necessary for our results.

2.4 Cryptography using circuits

We assume the reader is familiar with the standard notions of one-way functions and pseudorandom functions/permutations, and refer to [9] for further details.

Definition 2.7. Pseudorandom circuits (PRC) exist if for every efficient uniform algorithm D, there exists an infinite collection W of functions where for every $f \in W$, $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, f is computable by a circuit of size $s(n) = \text{poly}(n)$ and it holds that

$$\left| \Pr_{D, k \xleftarrow{R} U_n} [D^{f_k}(f, 1^s) = 1] - \Pr_{D, \phi} [D^{\phi}(f, 1^s) = 1] \right| \leq s^{-\omega(1)} \tag{2.1}$$

where f is passed to D as a circuit, $f_k = f(k, \cdot)$ and ϕ is a truly random function from $\{0,1\}^n \to \{0,1\}$.

f is a (uniform) pseudorandom permutation (PRP) if $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, f_k is a permutation for all k, f is computable by a uniform Turing Machine, and Equation 2.1 holds for all efficient D.

[4] Kearns et al. [16] defined closeness using KL divergence. We use statistical distance as this is sufficient in most applications and simplifies our presentation. All of our results also hold for KL divergence with appropriate (but qualitatively equivalent) scaling of parameters.

Definition 2.8. Distributional one-way circuits (DOWC) exist if for every efficient algorithm I, there exists a polynomial $p(s)$ and an infinite collection W of functions where for every $f \in W$, $f : \{0,1\}^n \to \{0,1\}^m$, f is computable by a circuit of size $s(n) = \text{poly}(n)$ and it holds that

$$\Delta((x, f(x)), (I(f, y), y \mid y = f(x))) > 1/p(s)$$

over random choice of $x \leftarrow_{\mathrm{R}} U_n$ and the random coins of I, and f is given to I as a circuit.

Our definition is based on [21] (although we require the inverter to *distributionally* invert the circuit, *i.e.* it must output a nearly-uniform preimage rather than an arbitrary preimage). One-wayness and pseudorandom functions are known to be equivalent for the uniform case [11, 10, 13], and the reductions establishing this extend immediately to the non-uniform case.

Theorem 2.9 ([11, 10, 13])**.** *DOWC exist if and only if PRC exist.*

2.5 Solving LTC implies inverting circuits

It was shown in Kearns et al. [16] that one can use an algorithm solving LTC in order to distinguish PRP from truly random functions. By looking at their proof, we observe that it also applies to PRC.

Theorem 2.10 (Kearns et al. [16])**.** *If LTC is efficiently solvable for the class* $\mathsf{SIZE}(n^2)$*, then PRC do not exist,* i.e. *there is a polynomial-time algorithm that distinguishes any efficient circuit from a truly random function.*

Corollary 2.11 (Follows from Theorem 2.10 and Theorem 2.9)**.** *If LTC is efficiently solvable for the class* $\mathsf{SIZE}(n^2)$*, then no family of circuits is distributionally one-way,* i.e. *there is a polynomial-time algorithm that distributionally inverts any polynomial-size circuit.*

3 Solving LTC implies solving agnostic learning w.r.t. efficient distributions

The idea of our proof of Theorem 1.1 is that we can use an LTC solver to learn the circuit that samples the distribution of labelled examples. This makes the PAC learning problem easier because we now have a circuit generating labelled examples (rather than just a set of labelled examples), and we show that the LTC solver can also be used to solve this relaxed PAC learning problem.

3.1 Circuit agnostic learning

To prove Theorem 1.1, we use a tool called "circuit agnostic learning". In standard notions of learning, the learning algorithm is given access only to examples drawn from the distribution (X, Y). One can also ask what happens when the learning algorithm gets access to a circuit that samples from the distribution (X, Y). For the setting of agnostic learning, we call this relaxed (and potentially easier) problem *circuit agnostic learning*:

Definition 3.1. A procedure A circuit-agnostic-learns a concept class $\mathcal{F}$ if on input circuit C of size s sampling a distribution (X, Y) over $\{0,1\}^{n+1}$, A outputs a hypothesis h such that

$$\mathsf{err}((X, Y), h) \leq \mathsf{err}((X, Y), \mathcal{F}) + \varepsilon$$

and A runs in time $\text{poly}(s, 1/\varepsilon)$.

At first glance it might seem that this model is trivially easy because the learning algorithm has access to C, which allows the learning algorithm to generate labelled examples by himself and may allow the learning algorithm to create a good hypothesis. However the problem remains non-trivial because C generates an example and its label *simultaneously*, while the problem the learning algorithm must solve is to compute the label on an example given as input.[5]

The following lemma implicitly was proved in [1] and explicitly appears in Xiao [30]

Lemma 3.2 ([1] (see also [30], Lemma 3.5.1))**.** *If there is an efficient algorithm that distributionally inverts all polynomial-size circuits, then there is an algorithm running in time* $\text{poly}(n, 1/\varepsilon)$ *that circuit-agnostically-learns* $\mathsf{SIZE}(n^2)$*.*

[5] To see a concrete example of a class for which circuit-agnostic learning is hard, consider the concept class in Section 4.2. For this concept class the standard PAC learning problem and the circuit agnostic learning problem are equivalent, since after $O(\log n)$ samples in the standard PAC model one can obtain the modulus N, which using Algorithm 4.8 allows one to construct a circuit sampling the input distribution $(U_n, f_N(U_n))$. This means that the Quadratic Residuosity assumption implies that the circuit agnostic learning problem for this concept class is hard.

3.2 Proof of Theorem 1.1

Theorem 3.3 (Theorem 1.1, formal)**.** *If there exists a polynomial-time algorithm A_{LTC} that solves LTC for the class $\mathsf{SIZE}(n^2)$, then there exists a polynomial-time algorithm A_{Agn} that solves agnostic learning with respect to the concept class $\mathsf{SIZE}(n^2)$ and with respect to all distributions samplable by $\mathsf{SIZE}(n^2)$ circuits.*

Proof of Theorem 3.3. By hypothesis there exists a polynomial-time algorithm A_{LTC} that solves LTC for the class $\mathsf{SIZE}(n^2)$. By Corollary 2.11 it follows that there is a polynomial-time algorithm I that distributionally inverts all circuits. By Lemma 3.2 there is a polynomial-time algorithm $A_{\mathsf{CircLearn}}$ that circuit-agnostically-learns $\mathsf{SIZE}(n^2)$.

Defining the agnostic learning algorithm: the algorithm A_{Agn} that learns agnostically w.r.t. efficiently samplable distributions does the following. By padding, we may assume that the input distribution X, Y is sampled by a circuit of size n^2. First, A_{Agn} obtains enough samples $(x_1, y_1), \ldots, (x_{t(n)}, y_{t(n)})$ from the example oracle to run A_{LTC} with error parameter $\varepsilon/2$. Let C be its output. Run $A_{\mathsf{CircLearn}}$ on C with error $\varepsilon/2$, and let h be the output of $A_{\mathsf{CircLearn}}$. Output h.

Analyzing A_{Agn}: since A_{LTC} solves LTC for the concept class $\mathsf{SIZE}(n^2)$, we get with probability $1 - 2^{-n}$ a hypothesis $C : \{0,1\}^m \to \{0,1\}^n$ such that $\Delta(C(U_m), (X, Y)) \leq \varepsilon/2$. Letting (X', Y') be the distribution samples by C, this implies that $\Delta((X', Y'), (X, Y)) \leq \varepsilon/2$. Since $A_{\mathsf{CircLearn}}$ solves $\mathsf{CircLearn}^{\mathsf{SIZE}(n^2)}$, it follows that with probability $1 - 2^{-n}$, the output hypothesis h satisfies $\mathsf{err}((X', Y'), h) \leq \varepsilon/2$. Together, it follows that $\mathsf{err}((X, Y), h) \leq \varepsilon$. ∎

Remark 3.4. All of the ingredients used in the proof of Theorem 3.3 relativize, and therefore the statement of Theorem 3.3 also relativizes. Namely, relative to any oracle $\mathcal{O}$, if solving LTC for the class $\mathsf{SIZE}^{\mathcal{O}}(n^2)$ is easy, then PAC learning $\mathsf{SIZE}^{\mathcal{O}}(n^2)$ with respect to input distributions samplable by $\mathsf{SIZE}^{\mathcal{O}}(\mathrm{poly}(n))$ circuits is easy.

4 LTC and PAC learning for concrete classes

In this section we show that, in contrast to Theorem 1.1, if one studies concrete concept classes that are not complete, then it is possible that PAC learning is harder than LTC.

Because PAC learning deals with single-bit output function while LTC deals with multi-bit output functions, in order to compare the two models for concrete concept classes we use two different ways to obtain both single- and multi-bit output functions from a single concept class:

1. Direct products: let $\mathcal{F}$ be a class of single-bit output functions. Then we compare PAC learning the class $\mathcal{F}$ to solving LTC for the class $\mathcal{F}^\ell$ where each function $f \in \mathcal{F}^\ell$ maps $\{0,1\}^n \to \{0,1\}^\ell$ and can be decomposed as $f(x) = (f_1(x), \ldots, f_\ell(x))$ where each $f_i \in \mathcal{F}$. Here, the number of copies $\ell(n)$ satisfies $\omega(\log n) \leq \ell(n) \leq \mathrm{poly}(n)$.

2. Generating labelled examples: let $\mathcal{F}$ be a class of single-bit output functions and let $\mathcal{D}$ be a class of distributions that are efficiently samplable. Then we compare PAC learning $\mathcal{F}$ with respect to input distributions in $\mathcal{D}$ to solving LTC for the class of distributions of the form $(X, f(X))$ where $X \in \mathcal{D}$ and $f \in \mathcal{F}$.

To motivate the above notions, the direct product notion is natural when thinking of $\mathcal{F}$ as being a syntactic complexity class, such as DNF formulas or $\mathsf{AC0}$ circuits. Thus, in the PAC model the function to be learned has complexity $\mathcal{F}$, and similarly in the LTC problem each bit of output in the output distribution has complexity $\mathcal{F}$.

The "generating labelled examples" notion is motivated by the proof of Theorem 1.1. In the proof of Theorem 1.1, the first step is to apply the LTC algorithm to produce a circuit that generates (approximately) the distribution of labelled examples. By considering this notion, we will see that the proof of Theorem 1.1 does not immediately generalize to hold for concrete concept classes.

Since we have no unconditional lower bounds for polynomial-time computation (which would be necessary to show that polynomial-time algorithms cannot solve PAC or LTC), all of the following results are conditional, *i.e.* they assume that some (standard) computational problem is hard.

4.1 Direct product

Proposition 4.1. *Assuming one-way functions exist, then there exists a concept class $\mathcal{F}$ that is hard to learn in the PAC model but such that solving LTC for the class $\mathcal{F}^\ell$ is easy.*

Proof. Since we assume one-way functions exist, therefore [11, 10, 18] implies that there exists a (uniform) pseudorandom permutation of the form $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ (we use the shorthand $f_k(x) = f(k,x)$) such that f_k is a permutation, and for all efficient distinguishers D,

$$|\Pr_{\phi,D}[D^\phi(1^n) = 1] - \Pr_{k,D}[D^{f_k}(1^n) = 1]| \leq \varepsilon(n)$$

where the distinguishing advantage $\varepsilon(n) = n^{-\omega(1)}$ is negligible. We make the following claim, which says that there exists an infinite collection of keys K such that f_k is hard-to-compute for all $k \in K$:

Lemma 4.2. *Let $\{f_k\}_{k\in\{0,1\}^*}$ be a collection of pseudorandom permutations with distinguishing advantage $\varepsilon(n)$. Then there exists an infinite set $K = \{k_n\}_{n\in\mathbb{N}}$ such that $\forall k \in K$, $n = |k|$, it holds for all efficient algorithms A that for large enough n,*

$$\Pr_A[A^{f_{k_n}} = h \text{ and } \mathsf{err}((U_n, f_{k_n}(U_n)), h) \leq 1/2 - 2\sqrt{\varepsilon(n)}] \leq n^2\sqrt{\varepsilon(n)}$$

We will prove this lemma shortly, first we use it to define $\mathcal{F}$ and prove Proposition 4.1.

Defining $\mathcal{F}$: let $K = \{k_n\}_{n\in\mathbb{N}}$ be the set of hard keys defined by Lemma 4.2. Let $f_{k_n}(x)_i$ denote the i'th bit of $f_{k_n}(x)$. We define

$$\mathcal{F} = \bigcup_{n\in\mathbb{N}} \{g_i : \{0,1\}^n \to \{0,1\}, g_i(x) = f_{k_n}(x)_i \mid i \in [n]\}$$

Claim 4.3. *PAC learning $\mathcal{F}$ is hard.*

This follows from Lemma 4.2: if there were an algorithm that PAC learns $\mathcal{F}$, then it could in particular be used to compute f_{k_n} for all n: given oracle access to f_{k_n}, one can simulate example oracles $(U_n, g_i(U_n))$ for all $i \in [n]$, and using the PAC learning algorithm for $\mathcal{F}$ with error $1/n^2$ one could obtain with high probability hypotheses h_i such that $\Pr[h_i(U_n) = g_i(U_n)] \geq 1 - 1/n^2$. Letting $h = (h_1, \ldots, h_n)$, we see that $\mathsf{err}((U_n, f_{k_n}(U_n)), h) \leq 1/n$, which contradicts Lemma 4.2 since $1/n \ll 1/2 - 2\sqrt{\varepsilon(n)}$.

Claim 4.4. *Solving LTC for the class $\mathcal{F}^\ell$ is easy.*

Fix any function $(g_{i_1}, \ldots, g_{i_\ell}) \in \mathcal{F}^\ell$. Since f_{k_n} is a permutation, $f_{k_n}(U_n)$ is uniform. Therefore, $g_{i_p}(U_n)$ and $g_{i_q}(U_n)$ are independent uniform bits if $i_p \neq i_q$, and they are always equal if $i_p = i_q$.

We now describe an algorithm that solves LTC for the class $\mathcal{F}^\ell$. Let D be the distribution to be learned, $D = (g_{i_1}(r), \ldots, g_{i_q}(r) \mid r \leftarrow_{\mathrm{R}} U_n)$.

1. Initialize a graph G on n vertices to be the complete graph, where the vertices are labelled $1, \ldots, n$.
2. Repeat the following $t = n \log \binom{n}{2}$ times. Sample $x \leftarrow_{\mathrm{R}} D$, and for every pair $u, v \in [n]$ such that $x_u \neq x_v$, remove the edge (u, v) from G.
3. The output hypothesis h does the following: for each connected component of G, sample a random bit. Output x where x_u equals the bit of the connected component containing u.

We claim that with all but probably 2^{-n}, the distribution sampled by h will be *exactly* the distribution generated by $(g_{i_1}, \ldots, g_{i_\ell})$. The only time that this hypothesis will be different is if there is some pair u, v such that $i_u \neq i_v$ and yet, for all examples x that the algorithm draws, it holds that $x_{i_u} = x_{i_v}$. Since for each sample this happens independently with probably $1/2$, and since there are $t = n \log \binom{n}{2}$ samples, by a union bound over all edges this happens with probably at most $\binom{n}{2} 2^{-n \log \binom{n}{2}} \leq 2^{-n}$. ∎

Proof of Lemma 4.2. For an algorithm A and a key k of length n, let

$$p_{A,k} = \Pr_A[A^{f_k} = h \text{ and } \mathsf{err}((U_n, f_k(U_n)), h) \leq 1/2 - 2\sqrt{\varepsilon(n)}]$$

where the probability is only over the random coins of A. It must hold that:

Claim 4.5. *For all sufficiently large n, $\Pr_{k\leftarrow_R\{0,1\}^n}[p_{A,k} > n^2\sqrt{\varepsilon(n)}] \leq 1/n^2$*

This claim holds because otherwise one could use A to break the pseudorandomness of f by the following distinguisher D. First D runs A to obtain h. Then, D queries its oracle on a new uniform $x \leftarrow_R U_n$; let b be the oracle's response. D accepts if $b = h(x)$ and rejects otherwise. It is easy to compute $\Pr_{\phi,D}[D^{\phi}(1^n) = 1] \leq \frac{1}{2} + p(n)2^{-n}$ where $p(n) = \text{poly}(n)$ is the maximum number of queries made by A. On the other hand, if Claim 4.5 does not hold, then $\Pr_{k,D}[D^{f_k}(1^n) = 1] \geq \frac{1}{2} + 2\varepsilon(n)$. We can assume *w.l.o.g.* that $\varepsilon \geq 2^{-n/2}$ (Lemma 4.2 only gets weaker if we increase ε), therefore this gives a distinguishing probability $2\varepsilon - p(n)2^{-n} \gg \varepsilon(n)$, contradicting the pseudorandomness of f_k.

Define $p_A = \Pr_{k_1,k_2,\ldots}[\text{For inifinitely many } n,\ p_{A,k_n} > n^2\sqrt{\varepsilon(n)}]$ where $k_n \leftarrow_R \{0,1\}^n$. Since the series $\sum_{n=1}^{\infty} 1/n^2 < \infty$, applying Theorem 2.1 and Claim 4.5 implies that $p_A = 0$. Since there is a countable number of algorithms A, this implies

$$\Pr_{k_1,k_2,\ldots}[\exists A,\ \text{For infinitely many } n,\ p_{A,k_n} > n^2\sqrt{\varepsilon(n)}] \leq \sum_A p_A = 0$$

Therefore, a random choice of K will satisfy the conclusion of Lemma 4.2 with probability 1. ■

4.2 Generating labelled examples

Our result for this model is based on the hardness of quadratic residuosity over Blum integers. We say that $N = pq$ is a Blum integer of length n if p, q are prime, $\lceil \log N \rceil = n$, $n - \lceil \log p \rceil \leq 2$, $n - \lceil \log q \rceil \leq 2$ and $p \equiv q \equiv 3 \pmod 4$. We say that x is a quadratic residue mod a if $\exists y \in \mathbb{Z}_N$ such that $x = y^2 \bmod a$ for $a \in \mathbb{N}$. The *Legendre symbol* $(\frac{x}{p})$ is equal to 0 if $x = 0 \bmod p$, it is equal to 1 if x is a quadratic residue mod p and -1 if x is a quadratic non-residue mod p. The *Jacobi symbol* is defined $(\frac{x}{N}) = (\frac{x}{p})(\frac{x}{q})$. It is possible to efficiently compute the Jacobi symbol using Euclid's algorithm. It is known that for a Blum integer N, $(\frac{-1}{N}) = 1$ but -1 is *not* a quadratic residue mod N.

Let $\text{QR}(N, x) = 1$ if $x = y^2 \mod N$ and 0 otherwise, and write $\text{QR}_N(x) = \text{QR}(N, x)$. The hardness of quadratic residuosity over Blum integers says that there is no polynomial time algorithm that evaluates $\text{QR}_N(x)$ given a Blum integer N and $x \in \mathbb{Z}_N$.[6] [7]

Proposition 4.6. *Assuming that Quadratic Residuosity is hard over Blum integers, there is a concept class $\mathcal{F}$ for which PAC learning with respect to the uniform distribution is hard while solving LTC for distributions $(U_n, f(U_n))$ where $f \in \mathcal{F}$ is easy.*

Proof. Define the functions $f_N : \mathbb{Z}_N \times [\lceil \log N \rceil] \times \{0,1\} \to \{0,1\}$ where

$$f_N(x, i, b) = \begin{cases} \text{QR}_N(x) & b = 0 \\ N_i & b = 1 \end{cases}$$

where N_i denotes the i'th bit of N. Let n the input length of f_N and let $\mathcal{F} = \{f_N \mid N \text{ is a Blum integer}\}$.

Claim 4.7. *PAC learning $\mathcal{F}$ is hard for the uniform distribution.*

Suppose we have a PAC learning algorithm A for $\mathcal{F}$ (it even suffices if A only works for uniformly distributed inputs). Given N, one can simulate an example oracle for $(U_n, f_N(U_n))$ as follows:

Algorithm 4.8 (Sampling from $(U_n, f_N(U_n))$:)**.**

1. Pick $x' \leftarrow_R \mathbb{Z}_N, i \leftarrow_R [\lceil \log N \rceil], b \leftarrow_R \{0,1\}$.
2. If $b = 1$ then output $((x', i, b), N_i)$, otherwise compute the Jacobi symbol $(\frac{x'}{N})$.
3. If $(\frac{x'}{N}) \neq 1$, then output $((x', i, b), 0)$.
4. If $(\frac{x'}{N}) = 1$, then sample $r \leftarrow_R \mathbb{Z}_N, a \leftarrow_R \{-1, 1\}$, and output $((ar^2, i, b), \frac{1+a}{2})$.

This simulated example oracle is identical to a true example oracle for f_N.

[6] Here, the fact that the factorization N is unknown to the algorithm is necessary to ensure hardness. Indeed, it is possible to compute $\text{QR}_N(x)$ using an efficient circuit that has the factorization p, q as advice. This is, for instance, why the class $\mathcal{F}$ defined in Proposition 4.6 is efficiently computable.

[7] This example can also be phrased in terms of the generic assumption that trapdoor permutations exist, but the presentation using Quadratic Residuosity is simpler.

Using a PAC learner to define an algorithm A' solving quadratic residuosity: on input (x, N), A' first checks if $(\frac{x}{N}) \neq 1$, and if so outputs 0. Otherwise, use Algorithm 4.8 to simulate an example oracle for $(U_n, f_N(U_n))$, then run A on this example oracle to obtain a hypothesis h. Finally, A' picks a random $r \leftarrow_R \mathbb{Z}_N, i \leftarrow_R [n]$ and outputs $h(xr^2, i, 0)$.

We claim that A' solves quadratic residuosity. Let S_1 be the set of quadratic residues in $\mathbb{Z}_N$, and let S_{-1} be the set of quadratic non-residues $y \in Z_N$ with Jacobi symbol $(\frac{y}{N}) = 1$. Observe that $\Pr_{y \leftarrow_R \mathbb{Z}_N}[y \in S_1] = \Pr_{y \leftarrow_R Z_N}[y \in S_{-1}] = 1/4 + o(1)$. Therefore, for all $a \in \{-1, 1\}$, it holds over uniform i that

$$\mathsf{err}(((S_a, i, 0), f_N(S_a, i, 0)), h) \leq (8 + o(1))\mathsf{err}((U_n, f_N(U_n)), h) \leq 9\varepsilon$$

for large enough n. Since for $a \in \{-1, 1\}$ and every $x \in S_a$, the variable $xr^2 \bmod N$ for random r is distributed uniformly in S_a, this implies that A' outputs $\mathrm{QR}_N(x)$ correctly with probability $1 - 9\varepsilon - 2^{-n}$.

Claim 4.9. *Solving LTC for $(U_n, f_N(U_n))$ for $f_N \in \mathcal{F}$ is easy.*

After seeing $O((\log N)^2) = \mathrm{poly}(n)$ samples, with all but negligible probability, N is revealed. Given N, one can sample from $(U_n, f_N(U_n))$ using Algorithm 4.8. ∎

5 PAC learning unsamplable distributions

Our construction of $\mathcal{O}$ will be randomized: we will select $\mathcal{R}$ from a distribution of oracles and show that with high probability that the oracle $\mathcal{O} = (\mathcal{R}, \mathbf{PSPACE})$ satisfies Theorem 1.3. The distribution will be as follows:

Definition 5.1. On input length n, let $\mathcal{R} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be chosen as follows: select $z \leftarrow_R \{0,1\}^n$. Pick the set S_z from the following distribution: for each $x \in \{0,1\}^n$, put x into S_z with probability $2^{-n/2}$. Then, for each $x \in S_z$, let $\mathcal{R}_z(x) = \mathcal{R}(z, x) = 1$ with probability $1/2$ and 0 otherwise. For all $z' \neq z$, let $\mathcal{R}_{z'}(x) = 0$ for all $x \in \{0,1\}^n$.

Intuitively, for each input length we first pick a "hard instance" z, then we pick a "hard set" S_z that is a sparse random subset of $\{0,1\}^n$ of size roughly $2^{n/2}$. We then define $\mathcal{R}_z$ to be a random function on S_z and 0 elsewhere, and also $\mathcal{R}_{z'}$ is identically zero for all $z' \neq z$. We remark that the definition of this oracle was also proposed by Impagliazzo [12], but as an alternative oracle for the main result of [29]. It was not studied with respect to the question of this paper, and the analyses we provide below are new.

Proof of Theorem 1.3. We will show that with overwhelming probability over choice of such $\mathcal{R}$, solving LTC relative to $\mathcal{O} = (\mathcal{R}, \mathbf{PSPACE})$ is easy, but PAC learning relative to $\mathcal{O} = (\mathcal{R}, \mathbf{PSPACE})$ with respect to unsamplable distributions is hard.

Lemma 5.2 (PAC learning unsamplable distribution is hard)**.** *With probability 1 over $\mathcal{R}$, for all efficient algorithms A with oracle access to $\mathcal{O} = (\mathcal{R}, \mathbf{PSPACE})$, and for all but finitely many n, let $z \in \{0,1\}^n$ be the hard instance on length n, then*

$$\Pr_A[A^{\mathcal{O}} \text{ given access to } (S_z, \mathcal{R}_z(S_z)) \text{ outputs } h \text{ s.t. } \mathsf{err}((S_z, \mathcal{R}_z(S_z)), h) \leq \tfrac{1}{2} - n^{-\log n}] \leq n^{-\log n}$$

The proof of this deferred to the full version. The intuition is that $\mathcal{R}_z(x)$ looks like a random bit: the only way an algorithm could predict $\mathcal{R}_z(x)$ is either if x was one of the examples it was given in the set of labelled examples, or if the learning algorithm finds the value z so that it can query the oracle at $\mathcal{R}_z(x)$. The first case is unlikely because $|S_z| \approx 2^{n/2}$, while the second is unlikely because z is chosen uniformly at random and therefore hard for the learning algorithm to find.

On the other hand, the following also holds:

Lemma 5.3 (Solving LTC easy)**.** *There is an efficient $A^{\mathcal{O}}$ such that with probability 1 over the choice of $\mathcal{R}$ where $\mathcal{O} = (\mathcal{R}, \mathbf{PSPACE})$, $A^{\mathcal{O}}$ solves LTC for the concept class $\mathsf{SIZE}^{\mathcal{O}}(n^2)$.*

We sketch the proof of this lemma shortly. Together, these two lemmas imply Theorem 1.3. Notice that we did not need to prove separately that S_z is unsamplable; this follows immediately since learning w.r.t. S_z is hard while learning w.r.t efficiently samplable distributions is easy: Remark 3.4 and Lemma 5.3 imply that there is an efficient algorithm that solves PAC learning for the concept class $\mathsf{SIZE}^{\mathcal{O}}(n^2)$ with respect to efficiently samplable distributions. ∎

Proof of Lemma 5.3. **The maximum likelihood approach:** We will use a maximum likelihood approach to solve LTC. The maximum likelihood algorithm says that, given a sample $T = (x_1, \ldots, x_t)$ that was obtained from one distribution out of a class of distributions $\mathcal{D}$, it suffices to pick the $D \in \mathcal{D}$ such that $\Pr[D^t = T]$ is maximized.

More formally, let $\mathrm{ML}_{\mathcal{D}}(x_1, \ldots, x_t) = \mathrm{argmax}_{D \in \mathcal{D}}\{\Pr[D^t = (x_1, \ldots, x_t)]\}$. The following holds:

Claim 5.4 (Folklore)**.** *Fix $\mathcal{D}$ of size $|\mathcal{D}| \leq 2^{\mathrm{poly}(n)}$ and such that for every $D \in \mathcal{D}$ and every $x \in \mathrm{supp}(D)$, $\Pr[D = x] \geq 2^{-\mathrm{poly}(n)}$. Then for $t = \mathrm{poly}(n, 1/\varepsilon)$ and for any $D \in \mathcal{D}$, it holds that:*

$$\Pr_{x_1,\ldots,x_t \leftarrow_R D}[\mathrm{ML}_{\mathcal{D}}(x_1, \ldots, x_t) = D' \wedge \Delta(D', D) > \varepsilon] \leq 2^{-n}$$

We defer a proof to the full version.

Let $\mathcal{D}^{\mathcal{O}}$ be the class of distributions sampled by circuits in $\mathsf{SIZE}^{\mathcal{O}}(n^2)$. To solve LTC for $\mathsf{SIZE}^{\mathcal{O}}(n^2)$, we would like to run the maximum likelihood approach over $\mathcal{D}^{\mathcal{O}}$. However, in order to calculate or even roughly approximate $\Pr[(D^{\mathcal{O}})^t = (x_1, \ldots, x_t)]$ for distributions $D^{\mathcal{O}}$ that might be sampled by circuits including $\mathcal{O}$ gates, one would need to know how $\mathcal{O} = (\mathcal{R}, \mathbf{PSPACE})$ behaves everywhere, and this requires querying $\mathcal{R}$ exponentially many times.

Our approach is to still use the maximum likelihood approach, but rather than applying the approach using $\mathcal{D}^{\mathcal{O}}$ as the class of distributions, we apply it to a related class $\mathcal{D}'_q = \{\mathcal{D}'_{q,n}\}_{n \in \mathbb{N}}$ for which having a **PSPACE** oracle is sufficient to calculate the maximum likelihood hypothesis.

Defining $\mathcal{D}'_q$ using truncated oracles: $\mathcal{D}'_{q,n}$ will be the following class of distributions. For $z \in \{0,1\}^n, S \subseteq \{0,1\}^n$, define the function $R_n^{z,S} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ to be $R_n^{z,S}(z, x) = 1$ if $x \in S$ and zero elsewhere. Note that $R_n^{z,S}$ can be concisely represented if $|S| = \mathrm{poly}(n)$. The class of $q(n)$-truncated oracles on length n is the following:

$$\mathcal{R}_{q,n} = \left\{ R = (R_1^{z_1,S_1}, \ldots, R_n^{z_n,S_n}) \mid \forall i \in [n],\ z_i \in \{0,1\}^i, S_i \subseteq \{0,1\}^i,\ |S_i| \leq q(n) \right\}$$

A distribution D is in $\mathcal{D}'_{q,n}$ if there exists an $R \in \mathcal{R}_{q,n^2}$ and oracle circuit of size n^2 that has (in addition to AND, OR, and NOT gates) **PSPACE** gates and R gates. Note that the circuit is allowed oracle gates only for a single $R \in \mathcal{R}_{q,n^2}$, or in other words it cannot have two different kind of oracle gates evaluating two different $R \neq R' \in \mathcal{R}_{\varepsilon,n^2}$. Also note that because we can explicitly represent S_i, each of these circuits is contained in $\mathsf{SIZE}^{\mathbf{PSPACE}}(n^3(1+q))$.

The following straightforward claim says that, with a **PSPACE** oracle, it is possible to efficiently evaluate the probability that $D \in \mathcal{D}'_{q,n}$ generates a particular sample:

Claim 5.5. *There exists an algorithm using a* **PSPACE** *oracle that runs in time* $\mathrm{poly}(n, 1/\varepsilon)$ *and computes* $\mathrm{ML}_{\mathcal{D}'_{q,n}}$.

This follows from the simple fact that calculating $\Pr[D = x]$ for a distribution D that is samplable by a $\mathsf{SIZE}^{\mathbf{PSPACE}}(\mathrm{poly}(n))$ circuit can be efficiently done with a **PSPACE** oracle. Since, as remarked above, $\mathcal{D}'_{q,n}$ can be sampled by $\mathsf{SIZE}^{\mathbf{PSPACE}}(n^3(1+q))$ circuits, this suffices to build the algorithm in Claim 5.5.

It therefore remains to prove that, with high probability over the choice of funtion $\mathcal{R}$, the class $\mathcal{D}'_{q,n}$ is a good approximation for the class $\mathcal{D}^{\mathcal{O}} = \{\mathcal{D}_n^{\mathcal{O}}\}_{n \in \mathbb{N}}$ sampled by circuits in $\mathsf{SIZE}^{\mathcal{O}}(n^2)$. We say that $\mathcal{D}_n^{\mathcal{O}}$ is ε-approximable by $\mathcal{D}'$ if for every $D \in \mathcal{D}_n^{\mathcal{O}}$, there exists $D' \in \mathcal{D}'$ such that $\Delta(D, D') \leq \varepsilon$.

Lemma 5.6. *For all n, ε, let $q = 16n^9/\varepsilon^3$, then $\Pr_{\mathcal{R}}[\mathcal{D}_n^{\mathcal{O}}$ is ε-approximable by $\mathcal{D}'_{q,n}] > 1 - 2^{-n}$.*

We defer the proof of this lemma to the full version. We briefly sketch the intuition here: let $C \in \mathsf{SIZE}^{\mathcal{O}}(n^2)$ be the circuit sampling D. Following an idea of [29], we prove that it is only necessary to know the queries that C makes to $\mathcal{R}$ that are "heavy", *i.e.* that occur with large probability. Then we can simply replace $\mathcal{R}$ gates by a truth table that includes values for all the heavy queries. This modified circuit is a circuit in $\mathcal{D}'_q$, and we show that this modification does not change the behavior of the output distribution by much.

Next use Lemma 5.6 to prove the theorem.

The learning algorithm A_{LTC}. We now combine our claims to obtain the following algorithm:

Algorithm 5.7.
Algorithm A_{LTC}: input size n, error parameter ε.

1. Let $t = \mathrm{poly}(n, 2/\varepsilon)$ be the appropriate polynomial to apply Claim 5.4 with error $\varepsilon/2$. A_{LTC} draws t examples $x_1, \ldots, x_t$ from D.
2. Using the algorithm of Claim 5.5, set $q = 16n^9(2t/\varepsilon)^3$ and compute $D' = \mathrm{ML}_{\mathcal{D}'_{q,n}}(x_1, \ldots, x_t)$. Output D'.

Proof of correctness: We prove that A_{LTC} indeed solves the LTC problem for $\mathsf{SIZE}^{\mathcal{O}}(n^2)$. By Theorem 2.1, it suffices to show that for all n, with probability $1-2^{-n}$ over the choice of $\mathcal{R}$, it holds for all $D \in \mathcal{D}^{\mathcal{O}}_n$ that

$$\Pr_{x_1,\ldots,x_t \leftarrow_{\mathrm{R}} D}[A_{\mathsf{LTC}}(x_1, \ldots, x_t) = D' \quad \wedge \quad \Delta(D', D) > \varepsilon] \leq \varepsilon \tag{5.1}$$

(This error can be reduced to 2^{-n} by repeating A_{LTC} and taking the best hypothesis it output.) For $q = 16n^9(2t/\varepsilon)^3$, Lemma 5.6 implies that with probability $1-2^{-n}$ over the choice of $\mathcal{R}$, $\mathcal{D}^{\mathcal{O}}_n$ is $(\varepsilon/2t)$-approximable by $\mathcal{D}'_{q,n}$. In this case, for every $D \in \mathcal{D}^{\mathcal{O}}_n$, there exists $D' \in \mathcal{D}'_{q,n}$ such that by the triangle inequality it holds that $\Delta(D^t, (D')^t) \leq \varepsilon/2$. Therefore

$$\begin{aligned}
\Pr_{x_1,\ldots,x_t \leftarrow_{\mathrm{R}} D}[A_{\mathsf{LTC}} = D' \wedge \Delta(D', D) > \varepsilon] &\leq \Pr_{x_1,\ldots,x_t \leftarrow_{\mathrm{R}} D'}[A_{\mathsf{LTC}} = D'' \wedge \Delta(D'', D) > \varepsilon] + \varepsilon/2 \\
&\leq \Pr_{x_1,\ldots,x_t \leftarrow_{\mathrm{R}} D'}[A_{\mathsf{LTC}} = D'' \wedge \Delta(D'', D') > \varepsilon - \varepsilon/(2t)] + \varepsilon/2 \\
&\leq 2^{-n} + \varepsilon/2 \leq \varepsilon
\end{aligned}$$

where penultimate inequality follows from Claim 5.4, since $D' \in \mathcal{D}'_{q,n}$ and A_{LTC} evaluates $\mathrm{ML}_{\mathcal{D}'_{q,n}}$ (the conditions of the hypothesis are satisfied because D is samplable by a polynomial-size oracle circuit). This proves Equation 5.1. Furthermore, observe that Claim 5.5 implies that A_{LTC} runs in polynomial time using a $(\mathcal{R}, \mathbf{PSPACE})$ oracle. ■

6 Acknowledgements

The author would like to thank the anonymous referees for their helpful comments.

References

[1] B. Applebaum, B. Barak, and D. Xiao. On basing lower-bounds for learning on worst-case assumptions. In *Proc. FOCS '08*, pages 211–220, 2008.

[2] T. Baker, J. Gill, and R. Solovay. Relativizations of the $\mathcal{P}$ =?$\mathcal{NP}$ question. *SIAM Journal on Computing*, 4(4):431–442, 1975. doi: 10.1137/0204037. URL http://link.aip.org/link/?SMJ/4/431/1.

[3] B. Barak. How to go beyond the black-box simulation barrier. In *Proc. 42nd FOCS*, pages 106–115. IEEE, 2001.

[4] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. In *CRYPTO '93*, pages 278–291, 1993. ISBN 3-540-57766-1.

[5] V. Feldman. On the power of membership queries in agnostic learning. *J. Mach. Learn. Res.*, 10:163–182, 2009. ISSN 1532-4435.

[6] V. Feldman and S. Shah. Separating models of learning with faulty teachers. *Theor. Comput. Sci.*, 410(19):1903–1912, 2009. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2009.01.017.

[7] Y. Freund. Boosting a weak learning algorithm by majority. In *Proc. COLT '90*, pages 202–216, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc. ISBN 1-55860-146-5.

[8] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting,. *Journal of Comp. and Sys. Sci.*, 55(1):119–139, 1997.

[9] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. Earlier version available on http://www.wisdom.weizmann.ac.il/~oded/frag.html .

[10] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986. ISSN 0004-5411. Preliminary version in FOCS' 84.

[11] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. of Com.*, 28(4):1364–1396, 1999. Preliminary versions appeared in STOC' 89 and STOC' 90.

[12] R. Impagliazzo. Private communication, 2009.
[13] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *Proc. 30th FOCS*, pages 230–235, 1989.
[14] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *STOC '89*, pages 44–61. ACM, 1989. ISBN 0-89791-307-8. doi: http://doi.acm.org/10.1145/73007.73012.
[15] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proc. STOC '89*, pages 433–444, New York, NY, USA, 1989. ACM. ISBN 0-89791-307-8. doi: http://doi.acm.org/10.1145/73007.73049.
[16] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 273–282, New York, NY, USA, 1994. ACM. ISBN 0-89791-663-8. doi: http://doi.acm.org/10.1145/195058.195155.
[17] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic learning. In *COLT '92*, pages 341–352, 1992. ISBN 0-89791-497-X. doi: http://doi.acm.org/10.1145/130385.130424.
[18] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. of Com.*, 17(2):373–386, 1988. Preliminary version in STOC' 86.
[19] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proc. 31st FOCS*, pages 2–10. IEEE, 1990.
[20] M. Naor. Evaluation may be easier than generation (extended abstract). In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 74–83, New York, NY, USA, 1996. ACM. ISBN 0-89791-785-5. doi: http://doi.acm.org/10.1145/237814.237833.
[21] R. Ostrovsky and A. Wigderson. One-way functions are essential for non-trivial zero-knowledge. Technical Report TR-93-073, International Computer Science Institute, Berkeley, CA, Nov. 1993. Preliminary version in Proc. 2nd Israeli Symp. on Theory of Computing and Systems, 1993, pp. 3–17.
[22] L. Pitt and M. K. Warmuth. Prediction-preserving reducibility. *J. Comput. Syst. Sci.*, 41(3): 430–467, 1990. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/0022-0000(90)90028-J.
[23] R. Schapire. The strength of weak learnability. *Proc. FOCS '89*, pages 28–33, 1989. doi: http://doi.ieeecomputersociety.org/10.1109/SFCS.1989.63451.
[24] A. Shamir. Ip = pspace. *J. ACM*, 39(4):869–877, 1992. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/146585.146609.
[25] H.-U. Simon. How many missing answers can be tolerated by query learners? In *STACS '02: Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, pages 384–395, London, UK, 2002. Springer-Verlag. ISBN 3-540-43283-3.
[26] S. P. Vadhan. An unconditional study of computational zero knowledge. *FOCS '04*, pages 176–185, 2004. ISSN 0272-5428. doi: http://doi.ieeecomputersociety.org/10.1109/FOCS.2004.13.
[27] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1968.1972.
[28] A. Wigderson. **P**, **NP**, and mathematics - a computational complexity perspective. In *Proceedings of the ICM '06*, volume 1, pages 665–712, Zurich, Switzerland, 2006. EMS Publishing House.
[29] D. Xiao. On basing $\mathbf{ZK} \neq \mathbf{BPP}$ on the hardness of PAC learning. In *In Proc. CCC '09*, pages 304–315, 2009.
[30] D. Xiao. *New Perspectives on the Complexity of Computational Learning, and Other Problems in Theoretical Computer Science*. PhD thesis, Princeton University, 2009.