Cryptographic Hardness of Distribution-specific Learning

Michael Kharitonov *

Abstract

We investigate cryptographic lower bounds on the learnability of Boolean formulas and constant depth circuits on the uniform distribution and other specific distributions. We first show that weakly learning Boolean formulas and constant depth threshold circuits with membership queries on the uniform distribution in polynomial time is as hard as factoring Blum integers (or inverting RSA, or deciding quadratic residuosity). We formalize the notion of a trivially learnable distribution and extend these hardness results to all non-trivial distributions. Moreover, we show that under appropriate assumptions on the hardness of factoring, the learnability of Boolean formulas and constant depth threshold circuits on any distribution is characterized by the distribution's Renyi entropy. Furthermore, we show that a sub-exponential lower bound for factoring implies a $\Omega(2^{\log^{\beta d} n})$ lower bound (for some constant β) for learning Boolean circuits of depth d on the uniform distribution (with membership queries), which matches the upper bound of Linial, Mansour, and Nisan [19]. From this we conclude that, assuming such a lower bound for factoring, there is no $O(n^{poly \log n})$ algorithm to learn all of AC^0 on the uniform distribution. We observe that, under cryptographic assumptions, all our bounds can be used to establish tradeoffs between the running time and the number of samples necessary to learn.

1 Introduction

Cryptographic tools and assumptions have been used heavily to show non-learnability results in the unrestricted hypothesis representation model of machine learning. Valiant pointed out in [25] that a pseudo-random function generator of Goldreich, Goldwasser, and Micali [11] can be used to show that, if one-way functions exist, arbitrary Boolean circuits are not predictable (the formal presentation of this argument can be found in [22]). A modified construction of a pseudo-random function generator was used by A. Blum [6] to separate the PAC learning model from the absolute mistake-bound model of learning. Kearns and Valiant [15] used specific public key encryption schemes to prove the unpredictability of NC^1 circuits (which are equivalent to Boolean formulas); using prediction preserving reductions of Pitt and Warmuth [22] they proved the unpredictability of deterministic finite acceptors and several other important classes. Angluin and Kharitonov [4] used chosen ciphertext secure public key encryption and secure digital signatures to prove similar results for prediction with membership queries. While these recent results were important in proving hardness of distribution-free learning, their common drawback is that distributions of examples used for this purpose were cryptographically oriented and unnatural. The possibility of using "malicious" distributions to show non-learnability highlighted the fact that the distribution independence requirement makes PAC learning of relatively simple concepts very difficult.

At the same time, significant progress has been achieved in constructing learning algorithms that work well on specific distributions. This is especially true for the uniform distribution. Linial, Mansour, and Nisan [19] used Fourier analysis to construct a $O(2^{\log^{\alpha d} n})$ algorithm (for some constant α) to learn Boolean circuits of depth d on the uniform distribution. Several improvements and extensions of this result followed, in some cases reducing both sample and time complexity and extending it to product distributions, though the running time of resulting algorithms remained super-polynomial. Very recently Mansour [21] used Fourier analysis to obtain a $O(n^{\log \log n})$ algorithm for learning DNF Boolean formulas with membership queries on the uniform distribution.

These successful efforts to construct learning algorithms that perform relatively well on the uniform distribution motivated our investigation of the cryptographic limitations on learning on specific distributions. Until recently, the only hardness result for the uniform (or any other "natural") distribution was the original observation of Valiant [25], which applied to unrestricted Boolean circuits. Recently, it was shown in [16] that, under the assumption that the subset sum problem of certain dimensions is hard, circuits of logarithmic depth are not learnable on the uniform distribution.

In this paper, much stronger results are proved. First, very general hardness results are proved for the Boolean

*Department of Computer Science, Stanford University, Stanford, CA 94305. Supported by a Fannie and John Hertz Fellowship and by NSF Presidential Young Investigator Grant CCR-8858097 with matching funds from AT&T and DEC. Present address: D. E. Shaw & Co., 120 West 45 St, New York, NY 10036

25th ACM STOC '93-5/93/CA,USA
© 1993 ACM 0-89791-591-7/93/0005/0372...$1.50

formulas and constant depth threshold circuits. We start by showing that, assuming factoring Blum integers is hard, Boolean formulas and constant depth threshold circuits are not predictable on the uniform distribution. A notion of a trivially learnable distribution is formalized. Then it is shown that, essentially, predicting Boolean formulas or constant depth threshold circuits on *any* non-trivial distribution is as hard as factoring. The result holds even when membership queries are allowed. The cryptographic assumption used, namely that factoring Blum integers is hard, is probably the most widely accepted cryptographic assumption. We observe that our results can also be shown assuming the security of RSA encryption or the hardness of deciding quadratic residuosity modulo a Blum integer.

Second, under a stronger, but very realistic assumption on the hardness of factoring, tight lower bounds are obtained for the learnability of constant depth Boolean circuits on the uniform distribution. This is the first hardness result for prediction of constant depth Boolean circuits, an important and natural class whose learnability was studied in several recent papers [19, 10]. Our lower bounds show that any significant improvement in either time or sample complexity of the algorithm in [19] is highly unlikely.

Throughout this paper we will use the Boolean circuit model of computation. All the circuits will be assumed to be over the standard Boolean basis $\wedge, \vee, \neg$ and of size polynomial in the length of their inputs (the number of Boolean inputs). Unless specified otherwise, all circuit families are assumed to be non-unform. We will discuss polynomial size circuits of unrestricted fan-in and circuits of fan-in two, and of various bounded depth. Let n denote the number of Boolean inputs to a circuit. Following the standard notation we will denote by NC^i the class of fan-in two circuits whose size is polynomial in n and whose depth is $O(\log^i n)$. We will denote by AC^i the class circuits with unrestricted fan-in whose size is polynomial in n and whose depth is $O(\log^i n)$. Clearly $NC^i \subseteq AC^i \subseteq NC^{i+1}$. Recall that the class NC^1 has the same expressive power as the class of polynomial size Boolean formulas.

We will use the parameter n as the *security parameter*. Unless specified otherwise, the lengths of inputs and the running times of efficient algorithms are bounded by polynomials in n, and the assumptions (and thus the theorems derived from them) are assumed to hold for all sufficiently large values of n. To be precise, all the statements involving the parameter n in any way should be quantified with "for all n sufficiently large". Without any loss of generality, we assume n (and some other parameters) to be a power of 2 whenever necessary to avoid dealing with fractional logarithms.

2 The Learning Model

In this section we review the model of concept representation and efficient prediction with membership queries on specific distributions. The definitions are mostly from [22, 4, 16].

2.1 Representations of concepts

For simplicity we will use a binary alphabet. Let X denote $\{0,1\}^*$; binary strings will represent both examples and concept names. If x is a string, $|x|$ denotes its length. For any natural number n, $X^{[n]} = \{x \in X : |x| \leq n\}$.

A *representation of concepts* $\mathcal{C}$ is any subset of $X \times X$. We interpret an element $\langle u, x\rangle$ of $X \times X$ as consisting of a *concept name* (or *concept representation*) u and an *example* x. The example x is a member of the concept u if and only if $\langle u, x\rangle \in \mathcal{C}$.

Define the *concept represented by* u as

$$\kappa_{\mathcal{C}}(u) = \{x : \langle u, x\rangle \in \mathcal{C}\}.$$

The *set of concepts represented by* $\mathcal{C}$ is

$$\{\kappa_{\mathcal{C}}(u) : u \in X\}.$$

If a set A of binary strings is a concept represented in $\mathcal{C}$, then we define $size_{\mathcal{C}}(A)$ to be the length of the shortest string u such that $\kappa_{\mathcal{C}}(u) = A$.

For example, the class of Boolean formulas is represented as follows. We fix a straightforward binary encoding of general Boolean formulas over the variables $X_1, X_2, \ldots$ and the basis AND ($\wedge$), OR ($\vee$), and NOT ($\neg$). Then $\langle u, x\rangle$ is an element of $\mathcal{C}_{BF}$ if and only if u represents a positive integer n and a Boolean formula ϕ over the variables $X_1, \ldots, X_n$ such that $|x| = n$ and the assignment $X_i = x_i$ for $i = 1, \ldots, n$ satisfies the formula ϕ.

Boolean circuits are represented in a similar way. Any Boolean circuit $\aleph$ on n inputs $X_1, \ldots, X_n$ is represented by $\{\langle u, x\rangle\}$ where u represents the positive integer n and the encoding of $\aleph$, $|x| = n$ and the assignment $X_i = x_i$ for $i = 1, \ldots, n$ causes the output of $\aleph$ to be 1. The class $\mathcal{C}_{\mathcal{P}/poly}$ of polynomial size Boolean circuits is specified analogously to Boolean formulas: $\langle u, x\rangle$ is an element of $\mathcal{C}_{\mathcal{P}/poly}$ if and only if u represents a positive integer n and a Boolean circuit C over the inputs $X_1, \ldots, X_n$, and $|x| = n$ and the assignment $X_i = x_i$ for $i = 1, \ldots, n$ causes the output of C to be 1.

When we say that a binary string u *encodes* or *represents* an object (such as a Boolean formula or a circuit of some type), we assume that a certain fixed and reasonable encoding scheme is used. With respect to any such scheme we can assume that all binary strings represent concepts by assigning an empty set to all binary

strings that do not form a valid encoding of an object. Unless stated otherwise, all numbers are represented in binary.

2.2 Prediction with queries

A *prediction with membership queries algorithm*, or *pwm-algorithm*, is a possibly randomized algorithm A that takes as input a bound s on the size of the target concept representation, a bound n on the length of examples, and an accuracy bound ϵ. It may make three different kinds of oracle calls, the responses to which are determined by the unknown target concept c and the specified distribution D on $X^{[n]}$, as follows.

1. A membership query takes a string $x \in X$ as input and returns 1 if $x \in c$ and 0 otherwise.
2. A request for a random classified example takes no input and returns a pair $\langle x, b \rangle$ where x is a string chosen independently according to D and $b = 1$ if $x \in c$ and $b = 0$ otherwise, and
3. A request for an element to predict takes no input and returns a string x chosen independently according to D.

A may make any number of membership queries or requests for random classified examples. However, A must eventually make one and only one request for an element to predict, and then eventually halt with an output of 1 or 0 without making any further oracle calls. The output is interpreted as A's guess of how the target concept classifies the element returned by the request for an element to predict. *A runs in polynomial time* if its running time (counting one step per oracle call) is bounded by a polynomial in s, n, and $1/\epsilon$.

We say that *A successfully predicts* a representation of concepts $\mathcal{C}$ on a distribution D if and only if for all positive integers s and n, for all positive rationals ϵ, for all concept names $u \in X^{[s]}$, when A is run with inputs s, n, and ϵ, and oracles determined by $c = \kappa_{\mathcal{C}}(u)$ and D, A asks membership queries that are in X and the probability is at most ϵ that the output of A is not equal to the correct classification of x by $\kappa_{\mathcal{C}}(u)$, where x is the string returned by the (unique) request for an element to predict. Recall that if $\epsilon = \frac{1}{2} - \frac{1}{n^c}$ for some constant $c > 0$ we say that *A weakly predicts $\mathcal{C}$* ([15]).

A representation of concepts $\mathcal{C}$ is *polynomially predictable with membership queries on a distribution D* if and only if there is a *pwm*-algorithm A that runs in polynomial time and successfully predicts $\mathcal{C}$ on D.

In particular we will focus on the case where D is the uniform distribution on $X^{[n]}$. Recall that a weak learning algorithm that is successful on a specific distribution does not necessarily imply the existence of a strong learning algorithm that is successful on the same distribution. (Unlike in the distribution independent model, where weak and strong learning were shown to be equivalent [24].) Thus it is even more desirable to show the hardness of weak prediction.

3 The $x^2 \bmod N$ Generator

A *pseudo-random generator* is a polynomial time computable function $f : \{0,1\}^n \to \{0,1\}^{l(n)}$, where $l(n) > n$, such that on a random input it produces an output that no polynomial time statistical test T can distinguish with non-negligible probability of success from a truly random sequence of the same length. More formally, we require that for any polynomial time computable function $T : \{0,1\}^{l(n)} \to \{0,1\}$ (i.e. the expected running time of a (randomized) algorithm computing T is polynomial in the size of its inputs) and for all c, the *success probability*

$$\begin{aligned} \delta_T(n) &= \left| Pr_{x \in \{0,1\}^n}[T(f(x)) = 1] \right. \\ &- \left. Pr_{y \in \{0,1\}^{l(n)}}[T(y) = 1] \right| < \frac{1}{n^c} \end{aligned}$$

where $x \in \{0,1\}^n$ and $y \in \{0,1\}^{l(n)}$ are chosen uniformly at random. We call $l(n)$ the *stretch factor* of the pseudo-random generator.

We describe the pseudo-random generator of Blum, Blum, and Shub [7] and show that, with pre-processing, any bit of the output sequence of this generator can be computed in NC^1 (or TC^0).

3.1 Background

Let us outline some of the results presented in [7]. For the background in number theory as it relates to cryptography we refer the reader to [2, 17].

Let n denote the *security parameter*, which means that the length of inputs and (expected) running times of efficient computations are bounded by polynomials in n. Let P and Q range over all primes such that $P \equiv Q \equiv 3 \pmod 4$ and let

$$\mathbf{N} = \{N \,|\, N = P \cdot Q, P \neq Q, |P| = |Q|\}$$

be the set of *parameter values*. For a given $N \in \mathbf{N}$ (such a number is called a *Blum integer*), let $X_N = \{x^2 \bmod N | x \in Z_N^*\}$ be the set of quadratic residues mod N. Let the disjoint union $X = \bigcup_{N \in \mathbf{N}} X_N$ be the *seed domain*. Let $X^n = \{(N, x) | N \in \mathbf{N}, |N| = n, x \in X_N\}$ be the set of *seeds of length n*. Define the distribution μ_n on X^n to be $\mu_n(N, X) = u_n(N) \cdot v_N(x)$, where u_n is the uniform distribution on $\{N \in \mathbf{N} || N| = n\}$ and v_N is the uniform distribution on X_N. Then μ_n is samplable in time polynomial in n. Thus we can assume that the input to the pseudo-random generator

comes from μ_n, rather than the uniform distribution on $\{0,1\}^n$.

The $x^2 \bmod N$ generator works as follows. Let (N, x_0) be chosen according to μ_n. For all values of $i \geq 0$, define inductively $x_{i+1} = x_i^2 \bmod N$ and let $b_i = lsb(x_i)$ be the parity bit (i.e. the least significant bit) of x_i. When the particular seed (N, x) is not clear from the context, we will denote the i^{th} bit generated from this seed by $b_i(N, x)$. Let the output of the generator be the infinite periodic sequence $b_0 b_1 b_2 \ldots$. Let λ denote the Carmicael's function. Note that $\lambda(P \cdot Q) = \text{lcm}[(P-1),(Q-1)]$. An important property of the $x^2 \bmod N$ generator described in [7] is that it is possible to efficiently access an arbitrary bit of its output sequence. Given any integer $i \geq 0$ we can compute b_i by computing $x_i = x_0^{2^i \bmod \lambda(N)} \bmod N$. We can assume $0 \leq i < \lambda(\lambda(N))$ because in [7] it is shown that the period of the sequence $x_0, x_1, \ldots$ is a divisor of $\lambda(\lambda(N))$, and can always be made equal to $\lambda(\lambda(N))$ by a more discriminating choice of the seed.

Assuming that recognizing quadratic residues modulo a Blum integer is hard, Blum, Blum, and Shub proved in [7] the following theorem:

Theorem 1 (Blum, Blum, and Shub) *For any polynomial $l(n)$, the $x^2 \bmod N$ generator that outputs $b_0 b_1 \ldots b_{l(n)-1}$ is a secure pseudo-random generator.*

Subsequently, Alexi, Chor, Goldreich, and Schnorr [1] and Vazirani and Vazirani [26] strengthened the security of the generator by showing that the Theorem 1 holds assuming that factoring Blum integers is hard. This assumption, which is probably the most widely accepted cryptographic assumption of all, will be used for all our results. We observe that all of our results hold under the quadratic residuosity assumption or assuming the security of RSA encryption (the first due to results in [7], the second due to results in [1]).

3.2 Super-efficient Random Access

The following theorem shows that a random bit from the output sequence of the $x^2 \bmod N$ pseudo-random generator can be computed extremely fast in parallel, provided that a certain amount of pre-processing is done at the time of seed selection. This fact allows us to construct shallow circuits that compute functions which are hard to learn on the uniform distribution. Let $TC^{0,d}_{p(n)}$ denote the class of threshold circuits on n variables with depth bounded by a fixed constant d and size bounded by some fixed polynomial $p(n)$.

Theorem 2 *With pre-processing, an arbitrary output bit of the $x^2 \bmod N$ pseudo-random generator can be computed by NC^1 or $TC^{0,d}_{p(n)}$ circuits.*

Proof: Omitted.

Theorem 2 shows that with pre-processing, each pseudo-random bit can be computed in parallel in time $O(\log n)$, which is sufficient for our purposes. But the $x^2 \bmod N$ pseudo-random generator is even more efficient. It was shown in [1, 26] that up to $\log n$ least significant bits of each x_i can be output by the generator without any loss of security. Therefore, in parallel, the $x^2 \bmod N$ pseudo-random generator works in constant amortized time per bit.

4 Main Hardness Results

We can use the $x^2 \bmod N$ pseudo-random generator to show that Boolean formulas and constant depth threshold circuits are not polynomial time predictable on specific non-trivial distributions, assuming that factoring Blum integers is hard. We first show this for the uniform distribution and then extend the hardness results to all non-trivial distributions.

4.1 The Uniform Distribution

We want to define a representation class of Boolean formulas such that a successful learning algorithm for this class could be used to perform a statistical test on an output sequence of the $x^2 \bmod N$ pseudo-random generator. Since we showed in Section 3.2 that an arbitrary bit of this sequence can be computed by a NC^1 circuit, and thus by a Boolean formula, the natural candidate for a hard concept class is the (parameterized) class

$$\mathcal{I}_n = \{\{i \in \{0,1\}^n | 0 \leq i < \lambda(\lambda(N)), \; b_i(N,x) = 1\} | (N,x) \in X^n\}$$

represented by Boolean formulas. Each seed for the $x^2 \bmod N$ pseudo-random generator defines a concept in this class which consists of those positions in the output sequence (within the $\lambda(\lambda(N))$ bit long prefix) that contain the bit 1. However, even if we could insure (by a discriminating choice of a seed) that the period of the output sequence is exactly $\lambda(\lambda(N))$, we would not be able to show that $\mathcal{I}_n$ is hard to learn. The reason for this is that there is no proof that a string consisting of a polynomial number of *non-consecutive* bits from the output sequence of the $x^2 \bmod N$ pseudo-random generator is pseudo-random. We only know, assuming that factoring Blum integers is hard, that any polynomially long prefix (or contiguous substring) of this sequence is pseudo-random.

We overcome this obstacle by introducing a new argument with a resemblance to the diagonalization technique used to show undecidability. Previous hardness results in computational learning theory were shown

by constructing representation classes which are "uniformly" hard, in the sense that for each polynomial time learning algorithm all but a small fraction of concepts in such a class are hard [15, 4]. In this section, on the other hand, we construct a concept class $\mathcal{G}_n$, represented by Boolean formulas, such that for each constant $b > 0$ and for each learning algorithm with running time bounded by n^b, a small fraction of representations in $\mathcal{G}_n$ is hard, even though these representations are easy to learn by an algorithm with running time $O(n^{db})$ for some fixed constant d. The formal development follows.

Given a number i expressed in binary, recall that $i \bmod 2^k$ is equal to the number represented by the k least significant bits of i. Define the (parameterized) concept class

$$\mathcal{G}_n = \{\{i \in \{0,1\}^n | b_{i \bmod 2^k}(N,x) = 1\} \\ |(N,x) \in X^n, 1 \leq k \leq n\}$$

In other words, each seed $(N, x_0) \in X^n$ and each number $1 \leq k \leq n$ together define a concept $g_{(N,x_0),k}$. A string $i \in \{0,1\}^n$ is in the concept $g_{(N,x_0),k}$ if and only if $b_{i \bmod 2^k}(N, x_0) = 1$.

Lemma 1 *There exist a (fixed) polynomial p such that the concept class $\mathcal{G}_n$ can be represented by a class of Boolean formulas of size bounded by $p(n)$.*

Proof: Omitted.

Lemma 2 *There exist a constant d and a (fixed) polynomial p such that the concept class $\mathcal{G}_n$ can be represented by a class of $TC^{0,d}_{p(n)}$ circuits.*

Proof: Omitted.

Now we must show that, assuming factoring is hard, no polynomial time learning algorithm can successfully weakly predict $\mathcal{G}_n$. We do this by showing how to convert such a learning algorithm into a successful statistical test for the $x^2 \bmod N$ pseudo-random generator, and thus into an algorithm for factoring.

Lemma 3 *Assuming that factoring Blum integers is hard, the concept class $\mathcal{G}_n$ (represented by Boolean formulas or $TC^{0,d}_{p(n)}$ circuits or any other representation of polynomial size) is not weakly predictable with membership queries on the uniform distribution in polynomial time.*

Proof: Assume to the contrary that there exists a polynomial time learning algorithm A that weakly predicts (with membership queries) $\mathcal{G}_n$ on the uniform distribution. Since all the target representations are of size bounded by n^s for some constant s, there exists a constant $b > 0$ such that the running time of A is bounded by n^a. (When A is run with inputs n^s (bounding the size of the target concept) and n (bounding the length of examples) and with labeled examples and prediction challenge chosen according to the uniform distribution.) Assume that A has success probability at least $\frac{1}{2} + \frac{1}{n^c}$ for some constant $c > 0$.

We now use A to construct a statistical test T for the $x^2 \bmod N$ pseudo-random generator with stretch n^{a+c}. Given a string z of length n^{a+c}, the test T assumes that it is the output of the $x^2 \bmod N$ pseudo-random generator on some unknown seed (N, x_0). Let $k = (a+c)\log n$. Then T simulates the learning algorithm A with inputs n^s and n on the target concept $g_{(N,x_0),k}$ from $\mathcal{G}_n$. The oracle queries of A are answered as follows.

1. When A makes a membership query with string i, if $|i| = n$, then T returns $z_{i \bmod 2^k}$ (i.e. the $(i \bmod 2^k)$th bit of z). If $|i| \neq n$, T returns 0. (This just means that i must be padded with leading zeroes by A.)

2. When A requests a random classified example, T chooses an example i uniformly at random from $\{0,1\}^n$ and returns $\langle i, z_{i \bmod 2^k} \rangle$ back to A.

3. When A requests an element to predict, T chooses j uniformly at random from $\{0,1\}^n$ and returns it to A as prediction challenge.

If within n^a steps A makes a prediction b on a challenge j, T outputs 1 if $b = z_{j \bmod 2^k}$ and 0 otherwise. If within n^a steps A fails to make a prediction (or request a challenge), T outputs 1 with probability 1/2 and 0 with probability 1/2.

If the string z is indeed the output of the $x^2 \bmod N$ pseudo-random generator on some seed (N, x_0), then the example and membership queries of A are evaluated according to the concept $g_{(N,x_0),k}$. Hence from correctness of A it follows that with probability at least $\frac{1}{2} + \frac{1}{n^c}$ the algorithm outputs correct prediction within n^a steps and the test T outputs 1.

If, on the other hand, z is a random n^{a+c} bit long string, we have to consider two possibilities. Since the running time of A is bounded by n^a, the number of labeled samples it obtains (via example or membership queries) is also bounded by n^a. Therefore, since the prediction challenge j is chosen uniformly at random from $\{0,1\}^n$, the probability that for some previously seen sample i we have $j \bmod 2^k = i \bmod 2^k$ is at most $\frac{1}{n^c}$. In this case we assume that A always outputs a correct prediction. In the case where for all previously seen samples i we have $j \bmod 2^k \neq i \bmod 2^k$, the truly random bit $z_{j \bmod 2^k}$ is never seen by the learning algorithm A, hence the probability that A predicts it correctly is exactly $\frac{1}{2}$. Therefore, when z is truly random, T outputs 1 with probability at most $\frac{1}{n^c} + (1 - \frac{1}{n^c}) \cdot \frac{1}{2} = \frac{1}{2} + \frac{1}{2n^c}$.

Therefore the success probability of T is

$$\delta \geq \frac{1}{2} + \frac{1}{n^c} - \frac{1}{2} - \frac{1}{2n^c} = \frac{1}{2n^c}$$

Clearly T runs in time $O(n^{a+c})$, hence T is a successful statistical test for the $x^2 \bmod N$ pseudo-random generator. By results reviewed in Section 3, such a test can be used to construct a (randomized) polynomial time algorithm for factoring Blum integers, which contradicts our assumption. ∎

From Lemmas 1 and 3 we immediately conclude that learning Boolean formulas on the uniform distribution is as hard as factoring Blum integers.

Theorem 3 *Assuming that factoring Blum integers is hard, there exists no prediction with membership queries algorithm that weakly predicts Boolean formulas on the uniform distribution.*

From Lemmas 2 and 3 we immediately conclude that there exist a constant d and a polynomial p such that learning $TC^{0,d}_{p(n)}$ circuits on the uniform distribution is as hard as factoring Blum integers.

Theorem 4 *Assuming that factoring Blum integers is hard, there exist a constant d and a polynomial p such that there exists no prediction with membership queries algorithm that weakly predicts $TC^{0,d}_{p(n)}$ circuits on the uniform distribution.*

As was mentioned at the end of Section 3.1, Theorems 3 and 4 also hold under the quadratic residuosity assumption or assuming the security of RSA encryption.

4.2 Other Distributions

After showing hardness of prediction on the uniform distribution, it is natural to consider other specific distributions. Clearly, some distributions are trivial to learn on by simply taking a sufficient number of samples and predicting via table lookup. We must first find a useful characterization of such "trivial" distributions. The definitions in this section are from [20]:

Definition 1 *Let $\mathcal{D}$ be a probability distribution on $\{0,1\}^n$ and let $X \in_{\mathcal{D}} \{0,1\}^n$ and $Y \in_{\mathcal{D}} \{0,1\}^n$ be independent random variables. Define the* Renyi entropy *of $\mathcal{D}$ as*

$$\mathrm{Re}(\mathcal{D}) = -\log\left(\Pr_{X,Y}[X = Y]\right)$$

Define the minimum entropy *of $\mathcal{D}$ as*

$$\mathrm{Me}(\mathcal{D}) = \min\left\{-\log\left(\Pr_X[X = x]\right) | x \in \{0,1\}^n\right\}$$

The Renyi entropy of a probability distribution allows us to compute the probability of prediction challenge being equal to a labeled example seen during the learning stage. The minimum entropy allows us to bound the same probability for membership queries. It is easy to see that for any distribution $\mathcal{D}$

$$\frac{\mathrm{Re}(\mathcal{D})}{2} \leq \mathrm{Me}(\mathcal{D}) \leq \mathrm{Re}(\mathcal{D})$$

The following theorem shows that all concept classes are trivially weakly learnable on distributions with small Renyi entropy.

Theorem 5 *Let $\mathcal{C}$ be any concept class over the domain $\{0,1\}^n$ and let $\mathcal{D}$ be any distribution on $\{0,1\}^n$. Then there exists a trivial algorithm to predict $\mathcal{C}$ on $\mathcal{D}$ with success probability $\frac{1}{2} + \frac{1}{2^{\mathrm{Re}(\mathcal{D})+1}}$.*

Proof: The algorithm simply requests a single labeled example and then the prediction challenge. By definition of Renyi entropy, the probability that the prediction challenge is equal to the labeled example is $\frac{1}{2^{\mathrm{Re}(\mathcal{D})}}$, and in this case the algorithm always predicts correctly. If the prediction challenge is not equal to the labeled example, the learning algorithm outputs 1 with probability $\frac{1}{2}$ and 0 also with probability $\frac{1}{2}$, thus predicting correctly with probability $\frac{1}{2}$. Therefore, the total success probability is $\frac{1}{2} + \frac{1}{2^{\mathrm{Re}(\mathcal{D})+1}}$. ∎

Note that the complexity and the success probability of the algorithm in the proof of Theorem 5 does not depend on the size or any other property of the target representation, just on the Renyi entropy of the sample distribution. From Theorem 5 it follows that *all* concept classes are efficiently weakly predictable on *all* distributions with Renyi entropy $O(\log n)$. We will call such distributions *trivial*. Assuming that factoring is hard, we will show that Boolean formulas and $TC^{0,d}_{p(n)}$ circuits are not weakly predictable on all non-trivial distributions; by Theorem 5 this is the strongest possible statement of this type.

If we want to focus on strong rather than weak learning, we must place additional restrictions on the distribution $\mathcal{D}$ for it to be trivially learnable. Namely, for $\mathcal{D}$ to be trivially learnable with success probability $1 - \epsilon$, the learner must be able to compile (through labeled samples or membership queries) a table of labeled examples whose probability under $\mathcal{D}$ is at least $1 - 2\epsilon$. Since our main goal is to prove hardness results, we concentrate on weak learning. All our results for non-trivial distributions can be modified in a straight-forward way to hold in the strong learning model.

Before we proceed, we need to define universal hash functions (introduced in [8]).

Definition 2 *Let $h : \{0,1\}^n \times \{0,1\}^{l(n)} \rightarrow \{0,1\}^{m(n)}$ be a function; let Y be a random variable distributed uniformly on $\{0,1\}^{l(n)}$. We say that $h(x,y)$ is a (pairwise independent) universal hash function if, for all $x, x' \in \{0,1\}^n$ such that $x \neq x'$, and for all $a, a' \in \{0,1\}^{m(n)}$,*

$$\Pr_Y [(h(x,Y) = a) \wedge (h(x',Y) = a')] = \frac{1}{2^{2m(n)}}$$

For a fixed $y \in \{0,1\}^{l(n)}$, we view $h(x,y)$ as a function $h_y(x)$ of x which "hashes" n bits to $m(n)$ bits. Intuitively, a universal hash function $h_y(x)$ has the property that, for a randomly chosen Y, every two distinct elements x and x' are mapped randomly and independently of each other by h_Y. In all our applications we have $n \geq m(n) \geq 1$. Several constructions of universal hash functions are described in [8, 20] and elsewhere. For our purposes, it is only important to know that for all $1 \leq m(n) \leq n$ there exist universal hash functions that can be computed by NC^1 or $TC^{0,d}_{p(n)}$ circuits.

We now use universal hash functions to concentrate the Renyi entropy of a probability distribution. The following lemma is analogous to the Smoothing Entropy Theorem of [14] (also see [20]).

Lemma 4 *Let $\mathcal{D}$ be a probability distribution on $\{0,1\}^n$ such that the Renyi entropy of $\mathcal{D}$ is at least $m(n)$. Let $h(x,y)$ be a universal hash function such that $|x| = n$, $|y| = l(n)$, $|h(x,y)| = m(n)$. Let $Y \in_U \{0,1\}^{l(n)}$. Then*

$$E_Y [\mathrm{Re}\,(h_Y(\mathcal{D}))] > m(n) - 1$$

Proof: Omitted.

We will use Lemma 4 in two ways. First, we can say that for any distribution $\mathcal{D}$ on $\{0,1\}^n$ such that $\mathrm{Re}(\mathcal{D}) \geq m(n)$, there exists $y' \in \{0,1\}^{l(n)}$ such that $h_{y'}$ preserves Renyi entropy, i.e. $\mathrm{Re}\,(h_{y'}(\mathcal{D})) > m(n) - 1$. Second, we can claim that if y is chosen uniformly at random from $\{0,1\}^{l(n)}$, then with high probability h_y preserves enough Renyi entropy.

We can now use the Renyi entropy preserving hash functions to specify which output bits of the $x^2 \bmod N$ pseudo-random generator determine the concepts of a class. For each number $1 \leq k \leq n$, let

$$h^k : \{0,1\}^n \times \{0,1\}^{l(n,k)} \rightarrow \{0,1\}^k$$

be a universal hash function. Define the (parameterized) concept class

$$\mathcal{G}'_n = \Big\{ \Big\{ i \in \{0,1\}^n | b_{h^k_y(i)}(N,x) = 1 \Big\} \\ |(N,x) \in X^n, 1 \leq k \leq n, y \in \{0,1\}^{l(n,k)} \Big\}$$

In other words, each seed $(N, x_0) \in X^n$, each number $1 \leq k \leq n$, and each string $y \in \{0,1\}^{l(n,k)}$ together define a concept $g_{(N,x_0),k,y}$. A string $i \in \{0,1\}^n$ is in the concept $g_{(N,x_0),k,y}$ if and only if $b_{h^k_y(i)}(N,x_0) = 1$.

Lemma 5 *There exist a (fixed) polynomial p such that the concept class $\mathcal{G}'_n$ can be represented by a class of Boolean formulas of size bounded by $p(n)$.*

Proof: Omitted.

Lemma 6 *There exist a constant d and a (fixed) polynomial p such that the concept class $\mathcal{G}'_n$ can be represented by a class of $TC^{0,d}_{p(n)}$ circuits.*

Proof: Omitted.

Let $\mathcal{D}$ be any non-trivial probability distribution on $\{0,1\}^n$. We want to show that no polynomial time learning algorithm can successfully weakly predict $\mathcal{G}'_n$ on $\mathcal{D}$. Recall that the only property of the uniform distribution used in the proof of Lemma 3 was that the probability that k least significant bits of the prediction challenge y are equal to k least significant bits of some previously seen labeled sample was low. By Lemma 4, we can achieve the same effect with any non-trivial distribution by hashing the whole sample into $k+1$ bits, instead of simply taking k least significant bits. To handle arbitrary distributions, we must assume that factoring is hard for non-uniform adversaries.

Lemma 7 *Assuming that factoring Blum integers is hard for polynomial size circuits, the concept class $\mathcal{G}'_n$ (represented by Boolean formulas or $TC^{0,d}_{p(n)}$ circuits or any other representation of polynomial size) is not weakly predictable with membership queries on any non-trivial distribution in polynomial time.*

Proof: Omitted (similar to the proof of Lemma 3).

If we restrict our attention to example distributions $\mathcal{D}$ that are uniform polynomial time samplable (i.e. there exist a randomized uniform Turing machine that generates $\mathcal{D}$ for all values of n), the test T no longer needs the oracle for sampling from $\mathcal{D}$. Then we can make T (and the resulting factoring algorithm) to be uniform by setting $|z| = n^{4(a+c)}$, $k = 4(a+c)\log n$, and choosing y' uniformly at random from $\{0,1\}^{l(n)}$.

From Lemmas 5 and 7 we immediately conclude that learning Boolean formulas on any non-trivial distribution is as hard as factoring Blum integers.

Theorem 6 *Assuming that factoring Blum integers is hard, there exists no prediction with membership queries algorithm that weakly predicts Boolean formulas on any non-trivial distribution.*

From Lemmas 6 and 7 we immediately conclude that there exist a constant d and a polynomial p such that learning $TC^{0,d}_{p(n)}$ circuits on any non-trivial distribution is as hard as factoring Blum integers.

Theorem 7 *Assuming that factoring Blum integers is hard, there exist a constant d and a polynomial p such that there exists no prediction with membership queries algorithm that weakly predicts $TC^{0,d}_{p(n)}$ circuits on any non-trivial distribution.*

Once again, Theorems 6 and 7 also hold under the quadratic residuosity assumption or assuming the security of RSA encryption.

5 Characterizations of Learnability

The results of the previous sections were shown under the assumption that no randomized polynomial time algorithm for factoring exists. Factoring is one of the most studied computational problems. The current state of the art algorithms for factoring can factor a n bit number N in expected time $O(e^{(1+o(1))\sqrt{n \log n}})$, under certain unproven assumptions on the distribution of primes. Thus it is realistic to assume an explicit super-polynomial lower bound on the hardness of factoring. In this section we prove even stronger hardness results for learning under such an assumption.

5.1 A Stronger Assumption

When we said in earlier sections that we assume "factoring Blum integers is hard" we meant that no polynomial time randomized algorithm (or, in some cases, polynomial size non-uniform circuit) can factor a random Blum integer. This assumption allowed us to prove impossibility of polynomial time learning. If we want to prove stronger hardness results for learning, we have to make a stronger assumption on the hardness of factoring.

The discussion of general (not necessarily polynomial time) assumptions and adversaries can be facilitated by the use of *achievement ratio*, suggested by Leonid Levin. Most of the definitions and proofs of facts stated in this section can be found in [20]. In the following discussion an *adversary* can be any type of algorithm. A *primitive* can be a hard problem, such as factoring, or a cryptographic primitive, such as a pseudo-random generator.

Definition 3 *The* achievement ratio *of an adversary A for a primitive f is defined as $\frac{\delta(n)}{t(n)}$, where $t(n)$ is the time bound of A and $\delta(n)$ is the success probability of A for f, and n is the security parameter. An adversary A is* $R(n)$-breaking *for a primitive f if the achievement ratio of A for f satisfies $\frac{\delta(n)}{t(n)} \geq R(n)$ (for all n sufficiently large). The primitive f is* $(1 - R(n))$-secure *if there exists no $R(n)$-breaking adversary for f.*

Thus, 0-secure means totally insecure, while 1-secure means totally secure. Traditional security with respect to polynomial time adversaries means $(1 - \frac{1}{n^c})$-secure for all constants $c > 0$. Note, for example, that since for any function defined on n-bit strings there exists an inverting algorithm that runs in 2^n time, there can be no $(1 - \frac{1}{2^n})$-secure one-way function.

We will modify slightly the $x^2 \bmod N$ pseudo-random generator by using the hidden bit of Goldreich and Levin [12] instead of the parity bit to output a pseudo-random sequence. If x and y are two n-bit binary strings, define $x \odot y$ to be the *inner product* mod 2 of x and y, i.e. $x \odot y = \sum_{i=1}^{n} x_i y_i \bmod 2$. For a given value of the security parameter n, the seed space will be $Y^n = X^n \times \{0,1\}^n$ with seed distribution $\nu(n)(N, X, R) = \mu(n)(N, X) \cdot w_n(R)$, where μ_n is the original seed distribution and w_n is the uniform distribution on $\{0,1\}^n$. In other words, the new seed contains an additional random n bit string. Given a seed (N, x_0, r) chosen according to $\nu(n)$, define as earlier $x_{i+1} = x_i^2 \bmod N$, but let $b_i = x_i \odot r$ (instead of $b_i = lsb(x_i)$).

First consider a pseudo-random generator with stretch $n + 1$ that, given a seed (N, x_0, r), outputs b_0, x_1, N, r. Using the new result on the hardness of the inner product bit (due to Levin [18]), it easy to show that, assuming factoring Blum integers is $(1 - R(n))$-secure, this pseudo-random generator is $(1 - n^a R(n))$-secure for some constant $a > 0$. (We require at least polynomial security for factoring, i.e. $R(n) < \frac{1}{n^c}$ for all constants c.) Now, for any desired stretch factor $s(n) < R(n)$, consider the pseudo-random generator that outputs the first $s(n)$ bits $b_0, \ldots, b_{s(n)-1}$ of the modified $x^2 \bmod N$ generator. By the results of [11], it follows that this pseudo-random generator is $(1 - n^a s(n) R(n))$-secure, which means that no statistical test for this generator can have achievement ratio of at least $n^a s(n) R(n)$.

5.2 Arbitrary Distributions Revisited

We can use a stronger assumption on the hardness of factoring to dispense with the diagonalization argument and prove explicit super-polynomial hardness results. In particular, we can show that the Renyi entropy of a distribution $\mathcal{D}$, under the matching assumption on the hardness of factoring, determines a lower bound on the complexity of any weak learning algorithm that predicts Boolean formulas and $TC^{0,d}_{p(n)}$ circuits on $\mathcal{D}$. This lower bound is fairly close to the upper bound of Theorem 5.

Intuitively, a stronger assumption on the hardness of factoring allows us to claim that a longer, super-polynomial prefix of the output sequence of the $x^2 \bmod N$ generator is pseudo-random with respect to all tests with similarly bounded running time. Because we ig-

nore polynomial factors and in general can be much more "sloppy", the diagonalization argument is no longer needed. Once again, we concentrate on the non-uniform case. The formal development follows.

Theorem 8 *Let $\mathcal{D}$ be a distribution on $\{0,1\}^n$ with Renyi entropy $\log n < \mathrm{Re}(\mathcal{D}) < \frac{\sqrt{n}}{2}$. Assume that factoring Blum integers is $(1-n^{-c}2^{-2\mathrm{Re}(\mathcal{D})})$-secure for all constants $c > 0$. Then for any $1 < g(n) < \mathrm{Re}(\mathcal{D})$ there exist no weak prediction algorithm for Boolean formulas (or $TC^{0,d}_{p(n)}$ circuits) on the distribution $\mathcal{D}$ with running time bounded by $2^{\mathrm{Re}(\mathcal{D})-g(n)}$ and success probability $\frac{1}{2}+\frac{1}{2^{g(n)}}$.*

Proof: Omitted.

We observe that the proof can be modified to allow membership queries; the assumption on the hardness of factoring must be increased to accommodate the change. Also, tighter bounds can be achieved with a more accurate analysis.

5.3 Tight Bounds for AC^0 functions

Perhaps the most interesting consequence of a stronger assumption on the hardness of factoring is that it allows us to show surprisingly strong hardness results for the learnability of constant depth Boolean circuits on the uniform distribution. Let $AC^{0,d}_{p(n)}$ denote the class of Boolean circuits (over the standard basis) on n variables with depth bounded by a fixed constant d and size bounded by some fixed polynomial $p(n)$. Linial, Mansour, and Nisan [19] constructed an algorithm that, for any value of d, learns $AC^{0,d}_{p(n)}$ on the uniform distribution in time $O(n^{\log^{\alpha\cdot d} n})$ for some constant α. In this section we prove a matching lower bound. Specifically, assuming that factoring is $(1-\frac{1}{2^{n^\epsilon}})$-hard for some $\epsilon > 0$, we show that the sample/time complexity of weakly learning $AC^{0,d}_{p(n)}$ circuits (for all sufficiently large constants d) is $\Omega(n^{\log^{\beta\cdot d} n})$ for some constant $\beta > 0$ (β depends on ϵ, but not directly on d).

The main idea behind the argument is that the strong assumption on the hardness of factoring allows us to use a shorter seed for the x^2 mod N pseudo-random generator while preserving reasonable security. The smaller size of the seed allows us to implement the generator with a $AC^{0,d}_{p(n)}$ circuit, where d depends on the desired stretch and security of the generator. The formal development follows.

Assume that factoring n bit long Blum integers is $(1-2^{-n^\epsilon})$-hard for some $\epsilon > 0$. Fix any sufficiently large constant γ and let $m = \log^{\frac{\gamma}{\epsilon}} n$. Then factoring m bit long Blum integers is $(1-2^{-\log^\gamma n})$-hard. Thus we can use an m bit long Blum integer and an m bit long quadratic residue as a seed for the generalized x^2 mod N pseudo-random generator described in Section 5.1. Define the (parameterized) concept class

$$\mathcal{G}''_n = \{\{i \in \{0,1\}^n | b_{i \bmod 2^k}(N,x,r) = 1\} \\ |(N,x,r) \in Y^m, 1 \le k \le m\}$$

Except for the size of the seed and the use of inner product to compute the output bits, $\mathcal{G}''_n$ is identical to $\mathcal{G}_n$.

Lemma 8 *There exist a (fixed) polynomial p and a constant d such that the concept class $\mathcal{G}''_n$ can be represented by a class of $AC^{0,d}_{p(n)}$ circuits.*

Proof: Omitted.

Recall that γ was an arbitrary constant. We can thus achieve our goal by proving that successful weak prediction of $\mathcal{G}''_n$ requires more than $2^{\log^{\frac{\gamma}{4}} n}$ time (and number of samples).

Lemma 9 *Assuming that factoring Blum integers is $(1-\frac{1}{2^{n^\epsilon}})$-hard, the concept class $\mathcal{G}''_n$ is not weakly predictable with membership queries on the uniform distribution in time $2^{\log^{\frac{\gamma}{4}} n}$.*

Proof: Omitted.

From Lemmas 8 and 9 we immediately conclude that, assuming factoring Blum integers is $(1-\frac{1}{2^{n^\epsilon}})$-hard, learning $AC^{0,d}_{p(n)}$ circuits on the uniform distribution requires $\Omega(n^{\log^{\beta d} n})$ time (number of samples).

Theorem 9 *Assuming that factoring Blum integers is $(1-\frac{1}{2^{n^\epsilon}})$-hard, there exist a constant β such that no prediction with membership queries algorithm weakly predicts $AC^{0,d}_{p(n)}$ circuits on the uniform distribution in time $O(n^{\log^{\beta d} n})$ with fewer than $\Omega(n^{\log^{\beta d} n})$ labeled samples.*

Thus, in particular, there can be no quasi-polynomial time algorithm with a polynomial number of samples that weakly predicts $AC^{0,d}_{p(n)}$ circuits.

Let $AC^0 = \bigcup_{d,p} AC^{0,d}_{p(n)}$. Then, by applying a diagonalization-type argument, it is easy to see that (under the same assumption on the hardness of factoring) there can be no $O(n^{poly \log n})$ algorithm to weakly predict AC^0.

6 Additional Remarks

In this paper we showed that predicting Boolean formulas and $TC^{0,d}_{p(n)}$ circuits on any non-trivial distribution is as hard as factoring Blum integers, even if membership queries are allowed. This fact can be combined with

some of the prediction with membership queries preserving reductions of [4] to show other representation classes to be similarly hard. Only the reductions that preserve example distributions can be used. For example, it follows from a reduction in [4] that two-way deterministic finite acceptors (2DFAs) are not predictable with membership queries on any non-trivial distribution. This contrasts sharply with one-way DFAs, which are exactly learnable with membership and equivalence queries [3], and thus PAC learnable with membership queries on any distribution.

The technique used in Section 4.2 to generalize the hardness results from the uniform to arbitrary distributions is quite general. In particular, it can be used to extend the results of [16] to all non-trivial distributions. The results for $AC^{0,d}_{p(n)}$ circuits can presently be generalized to some, but not all non-trivial distributions.

We observe that the lower bounds we prove are bounds on both the running time and the number of samples needed for successful prediction. In a way, they can be interpreted as a tradeoff between the running time and the number of samples - a prediction algorithm may be able to succeed with a small number of samples by using an amount of time sufficient to break the underlying pseudo-random generator (e.g. by factoring large Blum integers).

Finally, the learnability of DNF and CNF formulas remains open, though the $O(n^{\log\log n})$ prediction with membership queries algorithm for the uniform distribution (see [21]) is "almost" polynomial time. We are currently attempting to use a combination of our diagonalization method and techniques used to prove circuit lower bounds, such as random restrictions and Hastad's switching lemma [13], to extend our results for $AC^{0,d}_{p(n)}$ circuits to DNF and CNF formulas.

Acknowledgements

I wish to thank Farid Alizadeh, Avrim Blum, Andrew Goldberg, David Haussler, Michael Kearns, Mike Luby, Rajeev Motwani, Moni Naor, Madhu Sudan, Umesh Vazirani, Manfred Warmuth, and Moti Yung for useful comments and conversations. Extra thanks to Mike Luby for writing an excellent monograph on Pseudo-randomness [20]. I am grateful to the International Computer Science Institute, Berkeley, where part of this work was done.

References

[1] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions:certain parts are as hard as the whole. *SIAM J. on Computing*, 17:194–209, 1988.

[2] D. Angluin. Lecture notes on the complexity of some problems in number theory. Technical report, Yale University, Report No. TR-243, 1982.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[4] D. Angluin and M. Kharitonov. When Won't Membership Queries Help? In *Proc. of the 23d STOC*, pages 444–454. ACM, 1991.

[5] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM J. on Computing*, 15:994–1003, 1986.

[6] A. Blum. Separating Distribution-Free and Mistake-Bounded Learning Models over the Boolean Domain In *Proc. of the 31st FOCS*, pages 211–218. IEEE, 1990.

[7] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15:364–383, 1986.

[8] J. L. Carter and M. N. Wegman. Universal Classes of Hash Functions. *JCSS*, 18:143–154, 1979.

[9] A. K. Chandra, L. J. Stockmeyer, U. Vishkin. Constant depth reducibility. *SIAM J. on Computing*, 13:423–432, 1984.

[10] M. Furst, J. Jackson, and S. Smith. Improved learning of AC^0 functions. In *Proc. of the 4th COLT*, pages 317–325. Morgan Kaufmann, 1991.

[11] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33:792–807, 1986.

[12] O. Goldreich and L. Levin. A hard–core predicate for all one–way functions. In *Proc. of the 21st STOC*, pages 25–32. ACM, 1989.

[13] J. Hastad. *Computational limitations for small depth circuits*. MIT Press, 1986. Ph.D. thesis.

[14] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random Number Generation from Any One-way Function. In *Proc. 21st STOC*, pages 12–24. ACM, 1989.

[15] M. Kearns and L. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proc. 21st STOC*, pages 433–444. ACM, 1989.

[16] M. Kharitonov. Cryptographic Lower Bounds for Learnability of Boolean Functions on the Uniform Distribution In *Proc. of the 5th COLT*. Morgan Kaufmann, 1992.

[17] W. LeVeque. Fundamentals of Number Theory. Addison–Wesley, 1977.

[18] L. Levin. Manuscript.

[19] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. In *Proc. of the 30th FOCS*, pages 574–579. IEEE, 1989.

[20] M. Luby. Pseudo-randomness and Applications. Princeton University Press, to appear.

[21] Y. Mansour. An $O(n^{\log\log n})$ Learning Algorithm for DNF under the uniform distribution. In *Proc. of the 5th COLT*. Morgan Kaufmann, 1992.

[22] L. Pitt and M. Warmuth. Prediction-preserving reducibility. *JCSS*, 41:430–467, 1990.

[23] J. Reif. On threshold circuits and polynomial computation. In *Proc. of the 2d IEEE Structures*, pages 118–123. IEEE, 1987.

[24] R. E. Schapire. The strength of weak learnability. In *Proc. of the 30th FOCS*, pages 28–33. IEEE, 1989.

[25] L. G. Valiant. A theory of the learnable. *C. ACM*, 27:1134–1142, 1984.

[26] U. V. Vazirani and V. V. Vazirani. Efficient and secure pseudo-random number generation. In *Proc. of the 25th FOCS*, pages 458–463. IEEE, 1984.