Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pages 273–282, 1994.

On the Learnability of Discrete Distributions

(EXTENDED ABSTRACT)

Michael Kearns
AT&T Bell Laboratories

Yishay Mansour
Tel-Aviv University

Dana Ron
Hebrew University

Ronitt Rubinfeld
Cornell University

Robert E. Schapire
AT&T Bell Laboratories

Linda Sellie
University of Chicago

1 Introduction and History

We introduce and investigate a new model of learning *probability distributions* from independent draws. Our model is inspired by the popular Probably Approximately Correct (PAC) model for learning boolean functions from labeled examples [24], in the sense that we emphasize efficient and approximate learning, and we study the learnability of restricted classes of target distributions. The distribution classes we examine are often defined by some simple computational mechanism for transforming a truly random string of input bits (which is not visible to the learning algorithm) into the stochastic observation (output) seen by the learning algorithm. In this paper, we concentrate on discrete distributions over $\{0,1\}^n$.

The problem of inferring an approximation to an unknown probability distribution on the basis of independent draws has a long and complex history in the pattern recognition and statistics literature. For instance, the problem of estimating the parameters of a Gaussian density in high-dimensional space is one of the most studied statistical problems. Distribution learning problems have often been investigated in the context of *unsupervised learning*, in which a linear mixture of two or more distributions is generating the observations, and the final goal is not to model the distributions themselves, but to predict from which distribution each observation was drawn. Data clustering methods are a common tool here. There is also a large literature on *nonparametric density estimation*, in which no assumptions are made on the unknown target density. Nearest-neighbor approaches to the unsupervised learning problem often arise in the nonparametric setting. While we obviously cannot do justice to these areas here, the books of Duda and Hart [9] and Vapnik [25] provide excellent overviews and introductions to the pattern recognition work, as well as many pointers for further reading. See also Izenman's recent survey article [16].

Roughly speaking, our work departs from the traditional statistical and pattern recognition approaches in two ways. First, we place explicit emphasis on the computational complexity of distribution learning. It seems fair to say that while previous research has provided an excellent understanding of the information-theoretic issues involved in distribution learning — such as decay rates for the error of the maximum likelihood procedure, unbiasedness of estimators, and so on — little is known about how the computational difficulty of distribution learning scales with the computational effort required either to generate a draw from the target distribution, or to compute the weight it gives to a point. This scaling is the primary concern of this paper. Our second departure from the classical literature is a consequence of the first: in order to examine how this scaling behaves, we tend to study distribution classes chosen for their circuit complexity or computational complexity (in a sense to be made precise), and these classes often look quite different from the classically studied ones. Despite these departures, there remains overlap between our work and the classical research that we shall discuss where appropriate, and we of course invoke many valuable statistical tools in the course of our study of efficient distribution learning.

In our model, an unknown target distribution is chosen from a known restricted class of distributions over $\{0,1\}^n$, and the learning algorithm receives independent random draws from the target distribution. The algorithm also receives a confidence parameter δ and an approximation parameter ϵ. The goal is to output with probability at least $1-\delta$, and in polynomial time, a hypothesis distribution which has distance at most ϵ to the target distribution (where our distance measure is defined precisely later).

Our results highlight the importance of distinguishing between two rather different types of representations for a probability distribution D. The first representation, called an *evaluator* for D, takes as input any vector $\vec{y} \in \{0,1\}^n$, and outputs the real number $D[\vec{y}] \in [0,1]$, that is, the weight that $\vec{y}$ is given under D. The second and usually less demanding representation, called a *generator* for D, takes as input a string of truly random bits, and outputs a vector $\vec{y} \in \{0,1\}^n$ that is distributed according to D. It turns out that it can sometimes make a tremendous difference whether we insist that the hypothesis output by the learning algorithm be an evaluator or a generator.

For instance, one of our main positive results examines a natural class in which each distribution can be generated by a simple circuit of OR gates, but for which it is intractable to compute the probability that a given output is generated. In other words, each distribution in the class has a generator of polynomial size but not an evaluator of polynomial size; thus it appears to be unreasonable to demand that a learning algorithm's hypothesis be an evaluator. Nevertheless, we give an efficient algorithm for *perfectly* reconstructing the circuit *generating* the target distribution. This demonstrates the utility of the model of learning with a hypothesis that is a generator: despite the fact that evaluating probabilities

for these distributions is $\#P$-hard, there is still an efficient method for exactly reconstructing all high-order correlations between the bits of the distribution.

We then go on to give an efficient algorithm for learning distributions generated by simple circuits of parity gates. This algorithm outputs a hypothesis that can act as both an evaluator and a generator, and the algorithm relies on an interesting reduction of the distribution learning problem to a related PAC problem of learning a boolean function from labeled examples. In the part of our work that touches most closely on classically studied problems, we next give two different and incomparable algorithms for learning distributions that are linear mixtures of Hamming balls, a discrete version of mixtures of Gaussians.

We then turn our attention to hardness results for distribution learning. We show that under an assumption about the difficulty of learning parity functions with classification noise in the PAC model (a problem closely related to the long-standing coding theory problem of decoding a random linear code), the class of distributions defined by probabilistic finite automata is not efficiently learnable when the hypothesis must be an evaluator. Interestingly, if the hypothesis is allowed to be a generator, we are able to prove intractability only for the rather powerful class of distributions generated by polynomial-size circuits. The intractability results, especially those for learning with a hypothesis that is allowed to be a generator, seem to require methods substantially different from those used to obtain hardness in the PAC model.

We conclude with a discussion of a class of distributions that is artificial, but that has a rather curious property. The class is not efficiently learnable if the hypothesis must be an evaluator, but is efficiently learnable if the hypothesis is allowed to be a generator — but apparently only if the hypothesis is allowed to store the entire sample of observations drawn during the learning process. A function class with similar properties in the PAC model (that is, a class that is PAC learnable only if the hypothesis memorizes the training sample) provably does not exist [22]. Thus this construction is of some philosophical interest, since it is the first demonstration of a natural learning model in which the converse to Occam's Razor — namely, that efficient learning implies efficient compression — may fail. Note that this phenomenon is fundamentally computational in nature, since it is well-established in many learning models (including ours) that learning and an appropriately defined notion of compression are always equivalent in the absence of computational limitations.

2 Preliminaries

In this section, we describe our model of distribution learning. Our approach is directly influenced by the popular PAC model for learning boolean functions from labeled examples [24], in that we assume the unknown target distribution is chosen from a known class of distributions that are characterized by some simple computational device for generating independent observations or outputs. Although we focus on the learnability of discrete probability distributions over $\{0,1\}^n$, the definitions are easily extended to distributions and densities over other domains.

For any natural number $n \geq 1$, let $\mathcal{D}_n$ be a class of probability distributions over $\{0,1\}^n$. Throughout the paper, we regard n as a complexity parameter, and when considering the class $\mathcal{D}_n$ it is understood that our goal is to find a learning algorithm that works for any value of n, in time polynomial in n (and other parameters to be discussed shortly).

In order to evaluate the performance of a distribution learning algorithm, we need a measure of the distance between two probability distributions. For this we use the well-known Kullback-Leibler divergence. Let D and $\hat{D}$ be two probability distributions over $\{0,1\}^n$. Then

$$KL(D||\hat{D}) = \sum_{\vec{y} \in \{0,1\}^n} D[\vec{y}] \log \frac{D[\vec{y}]}{\hat{D}[\vec{y}]}$$

where $D[\vec{y}]$ denotes the probability assigned to $\vec{y}$ under D. Note that the Kullback-Leibler divergence is not actually a metric due to its asymmetry.

One can think of the Kullback-Leibler divergence in coding-theoretic terms. Suppose we use a code that is optimal for outputs drawn according to the distribution $\hat{D}$ in order to encode outputs drawn according to the distribution D. Then $KL(D||\hat{D})$ measures how many additional bits we use compared to an optimal code for D.

The Kullback-Leibler divergence is the most standard notion of the difference between distributions, and has been studied extensively in the information theory literature. One of its nicest properties is that it upper bounds other natural distance measures such as the L_1 distance:

$$L_1(D, \hat{D}) = \sum_{\vec{y} \in \{0,1\}^n} |D[\vec{y}] - \hat{D}[\vec{y}]|.$$

Thus it can be shown [8] that we always have

$$2 \ln 2 \sqrt{KL(D||\hat{D})} \geq L_1(D, \hat{D}).$$

It is also easily verified that if D is any distribution over $\{0,1\}^n$ and U is the uniform distribution, then $KL(D||U) \leq n$ (since we can always encode each output of U using n bits). Thus, the performance of the "random guessing" hypothesis has at worst Kullback-Leibler divergence n, and this will form our measuring stick for the performance of "weak learning" algorithms later in the paper. Another useful fact is that $KL(D||\hat{D}) \geq 0$ always, with equality only when $\hat{D} = D$.

Since we are interested in the computational complexity of distribution learning, we first need to define a notion of the complexity of a distribution. For our results it turns out to be crucial to distinguish between distributions that can be only *generated* efficiently, and distributions that can be both generated and *evaluated* efficiently. Similar distinctions have been made before in the context of average-case complexity [2, 13]. We now make these notions precise. We start by defining an efficient generator.

Definition 1 *Let $\mathcal{D}_n$ be a class of distributions over $\{0,1\}^n$. We say that $\mathcal{D}_n$ has* polynomial-size generators *if there are polynomials $p(\cdot)$ and $r(\cdot)$ such that for any $n \geq 1$, and for any distribution $D \in \mathcal{D}_n$, there is a circuit G_D, of size at most $p(n)$ and with $r(n)$ input bits and n output bits, whose* induced distribution *on $\{0,1\}^n$ is exactly D when the distribution of the $r(n)$ input bits is uniform. Thus, if $\vec{r} \in \{0,1\}^{r(n)}$ is a randomly chosen vector, then the random variable $G_D(\vec{r})$ is distributed according to D. We call G_D a* generator *for D.*

Next we define an efficient evaluator.

Definition 2 *Let $\mathcal{D}_n$ be a class of distributions over $\{0,1\}^n$. We say that $\mathcal{D}_n$ has* polynomial-size evaluators *if there is a polynomial $p(\cdot)$ such that for any $n \geq 1$, and for any distribution $D \in \mathcal{D}_n$, there is a circuit E_D, of size at most $p(n)$ and with n input bits, that on input $\vec{y} \in \{0,1\}^n$ outputs the*

binary representation of the probability assigned to $\vec{y}$ by D. Thus, if $\vec{y} \in \{0,1\}^n$, then $E_D(\vec{y})$ is the weight of $\vec{y}$ under D. We call E_D an evaluator *for D.*

All of the distribution classes studied in this paper have polynomial-size generators, but only some of them also have polynomial-size evaluators. Thus, we are interested both in algorithms that output hypotheses that are efficient generators only, and algorithms that output hypotheses that are efficient evaluators as well. To judge the performance of these hypotheses, we introduce the following notions.

Definition 3 *Let D be a distribution over $\{0,1\}^n$, and let G be a circuit taking $r(n)$ input bits and producing n output bits. Then we say that G is an ϵ-*good generator for *D if $KL(D||G) \leq \epsilon$, where $KL(D||G)$ denotes the Kullback-Leibler divergence of D and the induced distribution of G on $\{0,1\}^n$ (when the distribution of the $r(n)$ input bits to G is uniform). If E is a circuit with n input bits, we say that E is an ϵ-*good evaluator for *D if $KL(D||E) \leq \epsilon$, where $KL(D||E)$ denotes the Kullback-Leibler divergence of D and the distribution on $\{0,1\}^n$ defined by the mapping $E : \{0,1\}^n \rightarrow [0,1]$.*

We are now ready to define our learning protocol. Let $\mathcal{D}_n$ be a distribution class. When learning a particular target distribution $D \in \mathcal{D}_n$, a learning algorithm is given access to the oracle $GEN(D)$ that runs in unit time and returns a vector $\vec{y} \in \{0,1\}^n$ that is distributed according to D. We will often refer to a draw from $GEN(D)$ as an *observation* from D.

Definition 4 *Let $\mathcal{D}_n$ be a class of distributions. We say that $\mathcal{D}_n$ is* efficiently learnable with a generator (evaluator, respectively) *if there is an algorithm that, when given inputs $\epsilon > 0$ and $0 < \delta \leq 1$ and access to $GEN(D)$ for any unknown target distribution $D \in \mathcal{D}_n$, runs in time polynomial in $1/\epsilon$, $1/\delta$ and n and outputs a circuit G (E, respectively) that with probability at least $1-\delta$ is an ϵ-good generator (evaluator, respectively) for D.*

We will make use of several variations of this definition. First of all, we will say $\mathcal{D}_n$ is (efficiently) *exactly learnable* (either with a generator or with an evaluator) if the resulting hypothesis achieves Kullback-Leibler divergence 0 to the target (with high probability). In the opposite vein, we also wish to allow a notion of *weak learning*, in which the learning algorithm, although unable to obtain arbitrarily small error, still achieves some fixed but nontrivial accuracy. Thus, for any *fixed* $\epsilon > 0$ (possibly a function of n and δ), we say that $\mathcal{D}_n$ is (efficiently) ϵ-learnable with a generator or an evaluator, respectively, if the learning algorithm finds an ϵ-good generator or evaluator, respectively. In cases where our algorithms do not run in polynomial time but in quasi-polynomial time, or in cases where we wish to emphasize the dependence of the running time on some particular parameters of the problem, we may replace "efficiently learnable" by an explicit bound on the running time.

3 Learning OR-Gate Distributions

In this section and the next, we examine two classes of distributions over $\{0,1\}^n$ in which each distribution can most easily be thought of as being generated by a boolean circuit with exactly n outputs. The distribution is the output distribution of the circuit that is induced by providing truly random inputs to the circuit.

For any $k = k(n)$, we say that a distribution D over $\{0,1\}^n$ is a *k-OR distribution* if there is a depth-one circuit of n OR gates, each of fan-in at most k, such that when truly random input bits are given to the circuit, the resulting induced distribution on the n output bits is exactly D. Note that if every gate in such a circuit has fan-in exceeding $\log(2n^2/\epsilon)$, then the output of the circuit is $\vec{1}$ with probability at least $1-\epsilon/2n$, and such a distribution is trivially learnable both with a generator and with an evaluator. (Consider the evaluator that assigns $\vec{1}$ probability $1-\epsilon/2n$ and probability $\epsilon/(2n(2^n-1))$ to any other vector; this evaluator is ϵ-good.) However, even for a fixed k there can be correlations of arbitrarily high order in a k-OR distribution, because there are no restrictions on the fan-out of the inputs to the circuit. Thus, in some sense the smaller values of k are the most interesting. Also, note that without loss of generality any k-OR distribution over $\{0,1\}^n$ has at most kn inputs (corresponding to the case where each output gate has a disjoint set of inputs and is therefore independent of all other outputs).

Let OR_n^k denote the class of all k-OR distributions over $\{0,1\}^n$. What should we expect of a learning algorithm for the class OR_n^k? We begin by giving evidence that it would be overly ambitious to ask for an algorithm that learns with an evaluator, since polynomial-size evaluators for OR_n^k probably do not even exist:

Theorem 1 *For any $k \geq 3$, there is a fixed sequence of fan-in k OR-gate circuits $C_1, \ldots, C_n, \ldots$ such that it is $\#P$-hard to determine for a given $\vec{y} \in \{0,1\}^n$ the probability that $\vec{y}$ is generated by C_n. In other words, OR_n^k does not have polynomial-size evaluators, unless $\#P \subseteq P/\mathrm{poly}$.*

Proof: We use the fact that exactly counting the number of satisfying assignments to a monotone 2-CNF formula is a $\#P$-complete problem [23]. The circuit C_n will have inputs $x_1, \ldots, x_n$ that will correspond to the variables of a monotone 2-CNF formula, and also inputs $z_{i,j}$ for each possible monotone clause $(x_i \vee x_j)$. The outputs will consist of the "control" outputs $w_{i,j}$, each of which is connected to only the input $z_{i,j}$, and the outputs $y_{i,j}$, each of which is connected to $z_{i,j}$, x_i and x_j. The fan-in of each output gate is at most 3.

Now given a monotone 2-CNF formula f, we create a setting for the outputs of C_n as follows: for each clause $(x_i \vee x_j)$ appearing in f, we set $w_{i,j}$ to 0, and the rest of the $w_{i,j}$ are set to 1. The $y_{i,j}$ are also all set to 1. Let us call the resulting setting of the outputs $\vec{v}$. Note that the effect of setting a $w_{i,j}$ is to force its only input $z_{i,j}$ to assume the same value. If this value is 1, then the condition $y_{i,j} = 1$ is already satisfied (and thus we have "deleted" the clause $(x_i \vee x_j)$), and if this value is 0, then $y_{i,j} = 1$ will be satisfied only if $x_i = 1$ or $x_j = 1$ (and thus we have included the clause). It is easy to verify that if $\ell = n(n-1)/2$ is the number of possible clauses, then the probability that $\vec{v}$ is generated by C_n is exactly $1/2^\ell$ times the probability that the formula f is satisfied by a random assignment of its inputs, which in turn yields the number of satisfying assignments. $\square$(Theorem 1)

Since the distributions in OR_n^k probably do not have polynomial-size evaluators, it is unlikely that this class is efficiently learnable with an evaluator. The main result of this section is that for small values of k, OR_n^k is in fact efficiently learnable with a generator, even when we insist on exact learning (that is, $\epsilon = 0$). This result provides motivation for the model of learning with a generator: despite the fact that evaluating probabilities is intractable for this class, we can still learn to perfectly generate the distribution, and in fact can exactly reconstruct *all* of the dependencies between the output bits (since the structure of the generating circuit reveals this information).

Theorem 2 *The class OR_n^k is exactly learnable with a generator in time $O(n^2(2k)^{2k+\log k+1}(\log^2 k+\log(n/\delta)))$, which is polynomial in n, k^k and $\log 1/\delta$.*

Proof: We start by giving an overview of our algorithm. The goal of the algorithm is to construct an *OR* circuit which is isomorphic (up to renaming of the input bits) to the unknown target circuit, and thus generates the same distribution. Let $o_1,\ldots,o_n$ denote the n *OR* gates forming the outputs of the target circuit. The algorithm works in n phases. At any given time, the algorithm has a partial hypothesis *OR* circuit, and in phase i, it "correctly reconstructs" the connections of the ith *OR* gate to its inputs. It does so under the inductive assumption that in the previous $i-1$ phases it correctly reconstructed the connections of $o_1,\ldots,o_{i-1}$. The term *correctly reconstructed* means that the connections are correct up to isomorphism, and is formally defined as follows.

Let S_j be the set of input bits that feed the jth *OR* gate o_j in the target circuit, and let S'_j be the corresponding set in the hypothesis *OR* circuit constructed by the algorithm. Then the gates $o_1,\ldots,o_{i-1}$ have been *correctly reconstructed* by the algorithm if there exists a one-to-one mapping $\mu:\bigcup_{j=1}^{i-1}S'_j\rightarrow\bigcup_{j=1}^{i-1}S_j$, such that for every $1\leq j\leq i-1$, $\mu(S'_j)=S_j$, where $\mu(S'_j)$ is the image of S'_j under μ. If we correctly reconstruct all n gates of the target circuit, then by definition we have a circuit that is isomorphic to the target circuit, and thus generates the same distribution.

In order to correctly reconstruct the ith gate o_i, the algorithm first determines the *number* of inputs feeding o_i, and then determines *which* inputs feed o_i. The first task is simple to perform. Let $k_i\leq k$ be the number of inputs feeding gate o_i in the target circuit (that is, $k_i=|S_i|$), and let y_i denote the output of gate o_i. Then $\mathbf{Pr}[y_i=0]=1/2^{k_i}$. This probability (and hence k_i) can be computed *exactly*, with high probability, by observing $O(2^k)$ output vectors generated by the target circuit.

The second task, that of determining *which* inputs feed gate o_i, is considerably more involved. We shall eventually show that it reduces to the problem of computing the sizes of *unions* of input bit sets feeding at most $2\log k+2$ given gates in the target circuit. The next lemma shows that the sizes of such unions can be computed exactly with high probability from a sample of random vectors generated by the target circuit.

Lemma 3 *There is a procedure that, given access to random outputs of the target circuit and any set of r OR gates $o_{i_1},\ldots,o_{i_r}$ of fan-in k of the target circuit, computes $|S_{i_1}\cup\cdots\cup S_{i_r}|$ exactly with probability at least $1-\delta'$ in time $O(2^{2rk}\log 1/\delta')$.*

Proof: If $y_{i_1},\ldots,y_{i_r}$ are the outputs of gates $o_{i_1},\ldots,o_{i_r}$, let $y=y_{i_1}\vee\cdots\vee y_{i_r}$. Note that the value of y can be easily computed for every random output vector, and

$$\mathbf{Pr}[y=1]=1-1/2^{|S_{i_1}\cup\cdots\cup S_{i_r}|}.$$

Thus the procedure will simply use an estimate of $\mathbf{Pr}[y=1]$ on a sufficiently large random sample in the obvious way. Since $|S_{i_1}\cup\cdots\cup S_{i_r}|\leq kr$, the lemma follows from a Chernoff bound analysis. □(Lemma 3)

Later, we will set $r=2\log k+2$ and fix δ' to be $\delta/(n^2k^{\log k+1})$, which implies $O((2k)^{2k}(\log^2 k+\log(n/\delta)))$ running time.

We now show how the problem of determining which inputs should feed a given gate can be reduced to computations of the sizes of small unions of the S_j. For simplicity in the following presentation, we assume without loss of generality that if for some i, $o_1,\ldots,o_{i-1}$ are reconstructed correctly, then for every $1\leq j\leq i-1$, $S'_j=S_j$.

We define a *basic block* as a set of inputs that are indistinguishable with respect to the part of the target circuit that the algorithm has correctly reconstructed so far, in that every input in the basic block feeds exactly the same set of gates. More formally, given the connections of the gates $o_1,\ldots,o_{i-1}$, let us associate with each input bit x_j the set $O_j^i=\{o_\ell:\ell\in[i-1],j\in S_\ell\}$, which consists of the gates in $o_1,\ldots,o_{i-1}$ that are fed by the input x_j. Then we say that x_s and x_t are in the same *basic block at phase i* if $O_s^i=O_t^i$. The number of basic blocks in each phase is bounded by the number of inputs, which is at most kn.

Suppose for the moment that given any basic block B in phase i, we have a way of determining exactly how many of the inputs in B feed the next gate o_i (that is, we can compute $|S_i\cap B|$). Then we can correctly reconstruct o_i in the following manner. For each basic block B, we connect *any* subset of the inputs in B having the correct size $|S_i\cap B|$ to o_i. It is clear from the definition of a basic block that the choice of *which* subset of B is connected to o_i is irrelevant. If, after testing all the basic blocks, the number r of inputs feeding o_i is less than k_i, then o_i is connected to k_i-r additional *new* inputs (that is, inputs which are not connected to any of the previously reconstructed gates). It can easily be verified that if $o_1,\ldots,o_{i-1}$ were reconstructed correctly, then after reconstructing o_i as described above, $o_1,\ldots,o_i$ are reconstructed correctly as well.

Hence the only remaining problem is how to compute $|S_i\cap B|$. Without loss of generality let the inputs in B feed exactly the gates $o_1,\ldots,o_l$, where $l\leq i-1$. Then we may write

$$B=S_1\cap\cdots\cap S_l\cap\bar{S}_{l+1}\cap\cdots\cap\bar{S}_{i-1}.$$

This expression for B involves the intersection of $i-1$ sets, which may be as large as $n-1$. The following lemma shows that there is a much shorter expression for B.

Lemma 4 *The basic block B can be expressed as an intersection of at most k sets in $\{S_1,\ldots,S_{i-1},\bar{S}_1,\ldots,\bar{S}_{i-1}\}$.*

Proof: Pick any gate fed by the inputs in B, say o_1. If $B=S_1$ then we are done. Otherwise, let $S=S_1$, and pick either a set S_j such that $S\cap S_j$ is a proper subset of S or a set $\bar{S}_j$ such that $S\cap\bar{S}_j$ is a proper subset of S, and let S become $S\cap S_j$ or $S\cap\bar{S}_j$, respectively. Continue adding such subsets to the intersection S until $S=B$. Since initially $|S|=k$, and after each new intersection the size of S becomes strictly smaller, the number of sets in the final intersection is at most k. □(Lemma 4)

Based on this lemma, we can assume without loss of generality that

$$B=\left(\bigcap_{j=1}^{t}S_j\right)\cap\left(\bigcap_{j=t+1}^{t+\bar{t}}\bar{S}_j\right)$$

where $t+\bar{t}\leq k$. Hence,

$$|S_i\cap B|=\left|S_i\cap\left(\bigcap_{j=1}^{t}S_j\right)\cap\left(\bigcap_{j=t+1}^{t+\bar{t}}\bar{S}_j\right)\right|.$$

In order to simplify the evaluation of this expression, we show in the next lemma that there is an equivalent expression for $|S_i\cap B|$ which includes intersections of only *uncomplemented* sets S_j.

Lemma 5 *$|S_i\cap B|$ can be expressed as a sum and difference of the sizes of at most 2^{k+1} intersections of sets in*

$$\{S_i,S_1,\ldots,S_{t+\bar{t}}\}.$$

Proof: (Sketch) The lemma is proved by induction on the number $\bar{t}$ of complemented sets in the expression for $|S_i \cap B|$ following Lemma 4. The induction step is based on the identity that for any sets C and D, $|C \cap \bar{D}| = |C| - |C \cap D|$. □(Lemma 5)

Hence the problem of computing $|S_i \cap B|$ reduces to computing the sizes of all possible intersections among (at most) $k+1$ sets of inputs. Naively, we would explicitly compute the sizes of all 2^{k+1} intersections. We can obtain an improved bound by using a new result of Kahn, Linial, and Samorodintsky [17] which shows that the sizes of all 2^{k+1} intersections are in fact uniquely determined by the sizes of all intersections of at most $2\log k + 2$ sets. More formally:

Lemma 6 *(Implicit in Kahn et al. [17]) Let $\{T_i\}_{i=1}^m$ and $\{T'_i\}_{i=1}^m$ be two families of sets, where $T_i, T'_i \subseteq [\ell]$. For any $R \subseteq [m]$ let a_R be the size of $\bigcap_{i \in R} T_i$, and let a'_R be the size of $\bigcap_{i \in R} T'_i$. If $a_R = a'_R$ for every subset R of size at most $\log \ell + 1$, then $a_R = a'_R$ for every R.*

How exactly can we use this lemma? Let us first note that in our case, the size of the domain over which the sets $\mathcal{S} = \{S_i, S_1, \ldots, S_{t+\bar{t}}\}$ are defined is bounded by $(t + \bar{t} + 1)k \leq (k+1)k$. Assume that we have a way of computing the sizes of all intersections of at most $2\log k + 2 \geq \log((k+1)k) + 1$ of the sets in $\mathcal{S}$. Since the sets $\{S_1, \ldots, S_{t+\bar{t}}\}$ are known, we need only find a set S'_i so that the size of any intersection of at most $2\log k + 2$ sets in $\mathcal{S}' = \{S'_i, S_1, \ldots, S_{t+\bar{t}}\}$ equals the size of the corresponding intersection in $\mathcal{S}$. Lemma 6 then tells us that the size of any intersection (of any number of sets) in $\mathcal{S}'$ equals the size of the respective intersection in $\mathcal{S}$. In order to find such a set S'_i, we search through all $O(k^{2k})$ possible S'_i (that is, all possible connections of the new gate o_i to the inputs feeding the already correctly reconstructed gates $o_1, \ldots, o_{t+\bar{t}}$) until we find a connection consistent with the sizes of the intersections computed.

Thus, we are finally left only with the problem of computing the sizes of all small intersections. The next combinatorial lemma shows that this problem further reduces to computing the sizes of the corresponding *unions*, which finally allows us to apply the procedure of Lemma 3.

Lemma 7 *Let $T_1, \ldots, T_r$ be sets over some domain X. Given the sizes of all unions of the T_i, the sizes of all intersections of the T_i can be computed exactly in time $O(2^{2^r})$.*

Proof: (Sketch) Follows from the inclusion-exclusion identity and a simple inductive argument. □(Lemma 7)

We are now ready to complete the proof of the main theorem. By combining Lemma 5, Lemma 7, and Lemma 3, we have proved the following: for every $1 \leq i \leq n$, and for every basic block B in phase i, with probability at least $1 - \delta/(n^2 k)$, and in time $O((2k)^{2k+\log k}(\log^2 k + \log(n/\delta)))$, our algorithm computes exactly the number of inputs in B which should be connected to o_i.

In each phase there are at most kn basic blocks, and there are n phases. Hence, with probability at least $1 - \delta$ all computations are done correctly and consequently all gates are reconstructed correctly. The total running time of the algorithm is

$$O(n^2(2k)^{2k+\log k+1}(\log^2 k + \log(n/\delta))).$$

□(Theorem 2)

4 Learning Parity Gate Distributions

We say that distribution D over $\{0,1\}^n$ is a *parity distribution* if there is a depth-one circuit of n polynomially-bounded fan-in parity gates, such that when truly random bits are given as inputs to the circuit, the resulting induced distribution on the n output bits is exactly D.

Let $PARITY_n$ denote the class of all parity distributions on n outputs. Unlike the class OR_n^k distribution, this class has polynomial-size evaluators (see the remarks following the proof of Theorem 8), and in fact we show that it can be learned *exactly* with both a generator and an evaluator. Perhaps the most interesting aspect of this result is that it demonstrates a case in which a distribution learning problem can be solved using an algorithm for a related PAC learning problem.

Theorem 8 *The class $PARITY_n$ is efficiently exactly learnable with a generator and evaluator.*

Proof: The learning algorithm uses as a subroutine an algorithm for learning parity functions in the PAC model [10, 15] by solving a system of linear equations over the field of integers modulo 2. In the current context, this subroutine receives random examples of the form $\langle \vec{x}, f_S(\vec{x}) \rangle$, where $\vec{x} \in \{0,1\}^n$ is chosen uniformly, and f_S computes the parity of the vector $\vec{x}$ on the subset $S \subseteq \{x_1, \ldots, x_n\}$. With high probability, the subroutine determines S exactly.

Let $y_1, \ldots, y_n$ denote the output bits of the unknown parity circuit. Our algorithm relies on the following easy lemma, whose proof we omit.

Lemma 9 *For any i, either y_i can be computed as the linear sum $u_1 y_1 + \cdots + u_{i-1} y_{i-1}$ mod 2 for some $u_1, \ldots, u_{i-1} \in \{0,1\}$, or the output bit y_i is independent of $y_1, \ldots, y_{i-1}$.*

This lemma immediately suggests the following simple learning algorithm. The first output bit of our hypothesis distribution will always be determined by a fair coin flip. Now inductively at the ith phase of the learning algorithm, we take many random output vectors $\vec{y}$ from the target parity distribution, and give the pairs $\langle \vec{y}[i-1], y_i \rangle$ (where $\vec{y}[i-1]$ is the first $i-1$ bits of $\vec{y}$) as *labeled* examples to the subroutine for PAC learning parity functions. Lemma 9 shows that either the subroutine succeeds in finding coefficients $u_1, \ldots, u_{i-1}$ giving a linear expression for y_i in terms of $y_1, \ldots, y_{i-1}$ (in which case the ith output bit of the hypothesis distribution will simply compute this linear function of the first $i-1$ output bits), or it is impossible to find any such functional relationship (in which case the ith output bit of the hypothesis distribution will be determined by a fair coin flip). Which case has occurred is easily determined since, if given randomly labeled examples, it can be shown that the subroutine will fail to produce any hypothesis with high probability. A simple inductive argument establishes the correctness of the algorithm. □(Theorem 8)

Note that the proof establishes the fact that $PARITY_n$ has polynomial-size evaluators. Given a vector $\vec{y}$, we can use the hypothesis of our algorithm to evaluate the probability $\vec{y}$ is generated as follows: let ℓ be the number of hypothesis output bits determined by coin flips. Then if $\vec{y}$ is consistent with the linear dependencies of our hypothesis, the probability of generation is $1/2^\ell$, otherwise it is 0.

5 Learning Mixtures of Hamming Balls

A *Hamming ball distribution* over $\{0,1\}^n$ is defined by a *center* vector $\vec{x} \in \{0,1\}^n$ and a *corruption probability* $p \in [0,1]$. The distribution $\langle \vec{x}, p \rangle$ generates a vector $\vec{y} \in \{0,1\}^n$ in the following simple way: for each bit $1 \leq i \leq n$, $y_i = x_i$ with probability $1-p$, and $y_i = \overline{x}_i$ with probability p. Note that $p = 1/2$ yields the uniform distribution for any center vector $\vec{x}$. It is easy to see that Hamming ball distributions

have both polynomial-size generators and polynomial-size evaluators.

Hamming ball distributions are a natural model for concepts in which there is a "canonical" example of the concept (represented by the center vector) that is the most probable or typical example, and in which the probability decreases as the number of attributes in common with this canonical example decreases. For instance, we might suppose that there is a canonical robin (with typical size, wing span, markings, and so on) and that the distribution of robins resembles a Hamming ball around this canonical robin.

As we shall see shortly, there is an extremely simple and efficient algorithm for exactly learning Hamming balls with a generator and evaluator. In this section, we are interested in the learnability of *linear mixtures* of Hamming ball distributions. Thus, for any natural numbers n and k, the distribution class HB_n^k is defined as follows. Each distribution $D \in HB_n^k$ is parameterized by k triples $D = (\langle \vec{x}_1, p_1, q_1 \rangle, \ldots, \langle \vec{x}_k, p_k, q_k \rangle)$, where $\vec{x}_i \in \{0,1\}^n$, $p_i \in [0,1]$, and $q_i \in [0,1]$. The q_i are additionally constrained by $\sum_{i=1}^k q_i = 1$. The q_i are the *mixture coefficients*, and the distribution D is generated by first choosing an index i according to the distribution defined by the mixture coefficients q_i, and then generating $\vec{y} \in \{0,1\}^n$ according to the Hamming ball distribution $\langle \vec{x}_i, p_i \rangle$.

Linear mixtures of Hamming balls are a natural model for concepts in which there may be several unrelated subcategories of the concept, each with its own canonical representative. For instance, we might suppose that the distribution of birds can be approximated by a mixture of Hamming balls around a canonical robin, a canonical chicken, a canonical flamingo, and so on, with the mixture coefficient for chickens being considerably larger than that for flamingos.

The class $HB_n^k[U]$ is the subclass of HB_n^k in which the mixture coefficients are uniform, so $q_1 = q_2 = \cdots = q_k = 1/k$. The class $HB_n^k[C]$ is the subclass with the restriction that for each distribution, there is a common corruption probability for all balls in the mixture, so $p_1 = p_2 = \cdots = p_k$. The class $HB_n^k[U,C]$ obeys both restrictions.

In this section, we give two rather different algorithms for the class $HB_n^k[C]$ whose performance is incomparable. The first is a "weak" learning algorithm that is mildly superpolynomial. The second is a "strong" algorithm that is actually an exact learning algorithm for the subclass $HB_n^k[U,C]$ and runs in time polynomial in n but exponential in k. For k a superlogarithmic function of n, the first algorithm is faster, otherwise the second is faster.

Hamming ball mixtures are the distributions we study that perhaps come closest to those classically studied in pattern recognition, and they provide a natural setting for consideration of the unsupervised learning problem mentioned briefly in Section 1. The goal in the unsupervised learning problem for Hamming ball mixtures would not be to simply model the distribution of birds, but for each draw from the target distribution, to predict the type of bird (that is, to correctly associate each draw with the Hamming ball that actually generated the draw). Thus, we must classify the observations from the target distribution despite the fact that no classifications are provided with these observations, even during the training phase (hence the name unsupervised learning). There obviously may be some large residual error that is inevitable in this classification task — even if we know the target mixture exactly, there are some observations that may be equally likely to have been generated by several different centers. The optimal classifier is obtained by simply associating each observation with the center that assigns the highest likelihood to the observation (taking the mixture coefficients into account). Although we omit the details, the reader can easily establish that while our first learning algorithm for Hamming ball mixtures has no obvious application to the unsupervised learning problem, our second algorithm can in fact be used to obtain near-optimal classification in polynomial time.

In presenting our algorithms, we assume that the common corruption probability p is known; in the full paper, we show how this assumption can be weakened using a standard binary search method.

Recall that in Section 2 we argued that Kullback-Leibler divergence n was the equivalent of random guessing, so the accuracy achieved by the algorithm of the following theorem is nontrivial, although far from perfect.

Theorem 10 *The class $HB_n^k[C]$ is $\tilde{O}(\sqrt{pn})$-learnable*[1] *with an evaluator and generator in time*

$$O\left(k^{\Theta\left(\frac{p}{(1-2p)^2}\log\frac{n}{\delta}\right)}\right).$$

Proof: (Sketch) We sketch the main ideas for the smaller class $HB_n^k[U,C]$, and indicate how we can remove the assumption of uniform mixture coefficients at the end of the proof. Thus let $\{\vec{x}_1, \ldots, \vec{x}_k\}$ be the target centers, let $q_1 = \cdots = q_k = 1/k$, and let $p < 1/2$ be the fixed common corruption probability. We begin by giving a simple but important lemma.

Lemma 11 *For any Hamming ball $\langle \vec{x}, p \rangle$, the vector $\vec{x}$ can be recovered* exactly *in polynomial time with probability at least $1-\delta$, using only*

$$O\left(\frac{p}{(1-2p)^2}\log\frac{n}{\delta}\right)$$

observations.

Proof: (Sketch) The algorithm takes the bitwise majority vote of the observations to compute its hypothesis center. A simple Chernoff bound analysis yields the stated bound on the number of observations. □(Lemma 11)

We can now explain the main ideas behind our algorithm and its analysis. The algorithm is divided into two stages: the *candidate centers* stage, and the *covering* stage. In the candidate centers stage, we take a sample of vectors of size $\Theta(k \log k \cdot (p/(1-2p)^2)\log(n/\delta))$ from the mixture. This sample size is sufficient to ensure that with high probability, each of the target centers was used $\Omega((p/(1-2p)^2)\log(n/\delta))$ times to generate a sample vector (here we are using the fact that the mixture coefficients are uniform; in the general analysis, we replace this by a sample sufficiently large to hit all the "heavy" centers many times). By Lemma 11, if we knew a large enough subset of sample vectors which were all generated by corruptions of the same target center, we could simply take the bitwise majority of these vectors to recover this target center exactly. Since we do not know such a subsample, we instead obtain a bitwise majority *candidate center* for *every* subset of size $\Theta((p/(1-2p)^2)\log(n/\delta))$ in the sample. Lemma 11 guarantees that for those subsets that were actually generated by corruptions of a single target center, the bitwise majority will recover that center. The number of sample subsets we examine is thus

$$\begin{aligned} \ell &= \binom{\Theta(k \log k \cdot (p/(1-2p)^2)\log(n/\delta))}{\Theta((p/(1-2p)^2)\log(n/\delta))} \\ &= k^{\Theta\left(\frac{p}{(1-2p)^2}\log\frac{n}{\delta}\right)} \end{aligned}$$

[1] The $\tilde{O}(\cdot)$ notation hides logarithmic factors in the same way that $O(\cdot)$ notation hides constant factors.

The dependence on n in the bound on ℓ is mildly super-polynomial, and it is this quantity that dominates our final running time. Thus, the candidate centers stage results in a large set of vectors $\{\vec{x}'_1, \ldots, \vec{x}'_\ell\}$ that with high probability contains all the target centers. Our goal now is to construct a set covering problem in order to choose a polynomial-size subset of the ℓ candidate centers that form a "good" hypothesis mixture. This covering stage will run in time polynomial in ℓ.

We begin the covering stage by drawing an additional sample of m vectors $S = \{\vec{y}_1, \ldots, \vec{y}_m\}$, where m will be determined by the analysis. We say that a candidate center $\vec{x}'_i$ *d-covers* the sample vector $\vec{y}_j$ if $\rho(\vec{x}'_i, \vec{y}_j) \leq pn + d$, where $\rho(\vec{x}'_i, \vec{y}_j)$ denotes the Hamming distance between the vectors. Thus, a center covers a sample vector if the Hamming distance between them exceeds the expected value pn by at most d (where the expected value is taken under the assumption that the center actually did generate the vector). A collection C of candidate centers will be called a *d-cover* of S if each $s \in S$ is d-covered by some center $c \in C$.

The following lemma, whose proof is straightforward and omitted, provides a value for d ensuring that the target centers form a d-cover.

Lemma 12 *For any m and any δ, with probability $1 - \delta$ over the generation of the observations $S = \{\vec{y}_1, \ldots, \vec{y}_m\}$, the target centers $\{\vec{x}_1, \ldots, \vec{x}_k\}$ form an $O(\sqrt{pn \log(m/\delta)})$-cover of S.*

By identifying each candidate center with the subset of S that it $O(\sqrt{pn \log(m/\delta)})$-covers, by Lemma 12 we have constructed an instance of set cover in which the optimal cover has cardinality at most k. By applying the greedy algorithm, we obtain a subcollection of at most $k \log m$ candidate centers that covers S [7]. Let us assume without loss of generality that this subcollection is simply $\{\vec{x}'_1, \ldots, \vec{x}'_{k \log m}\} = C'$. Our hypothesis distribution is this subcollection, with corruption probability p and uniform mixture coefficients, that is, $q_i = 1/(k \log m)$.

To analyze our performance, we will take the standard approach of comparing the *log-loss* of our hypothesis on S to the log-loss of the target distribution on S [14]. We define the log-loss by $loss(D, S) = \sum_{\vec{y} \in S} -\log D[\vec{y}]$ where $D[\vec{y}]$ denotes the probability $\vec{y}$ is generated by the distribution D. Eventually we shall use the fact that for a sufficiently large sample, the difference between the log-loss of our hypothesis and the log-loss of the target gives an upper bound on the Kullback-Leibler divergence [14].

Note that since our hypothesis centers $O(\sqrt{pn \log(m/\delta)})$-cover the sample S, and each hypothesis center is given mixture coefficient $1/(k \log m)$, our hypothesis assigns probability at least

$$\frac{1}{k \log m} p^{pn + O\left(\sqrt{pn \log(m/\delta)}\right)} (1-p)^{n - \left(pn + O\left(\sqrt{pn \log(m/\delta)}\right)\right)}$$

to every vector in S. The following lemma translates this lower bound on the probability our hypothesis assigns to each vector into an upper bound on the log-loss incurred by our hypothesis on each vector.

Lemma 13

$$-\log\left(\frac{1}{k \log m} p^{pn+d}(1-p)^{n-(pn+d)}\right) = n\mathcal{H}(p) + d\left(\log\frac{1}{p} - \log\frac{1}{1-p}\right) + \log k + \log\log m.$$

(Here, $\mathcal{H}(p) = -p\log(p) - (1-p)\log(1-p)$ is the standard binary entropy function.)

Furthermore, it can be shown that the expected log-loss of the target distribution is lower bounded by $n\mathcal{H}(p)$ [8]. Thus, provided m is sufficiently large for the uniform convergence of the expected log-losses to the true log-losses [14], we are ensured that the expected log-loss of our hypothesis exceeds that of the target by at most

$$O\left(\sqrt{pn \log(m/\delta)}\right)\left(\log\frac{1}{p} - \log\frac{1}{1-p}\right) + \log k + \log\log m$$

and this is by definition an upper bound on the Kullback-Leibler divergence. It can be shown (details omitted) that the choice $m = \Omega(\log(1/p) k n^3)$ suffices, giving a final divergence bound that is $\tilde{O}(\sqrt{pn})$ as desired.

To dispose of the assumption of uniform mixture coefficients requires two steps that we merely sketch here. First, as we have already mentioned, in the candidate centers phase we will sample only enough to obtain a sufficiently large number of observations from the "heavy" centers. This will mean that in the covering phase, we will not be ensured that there is a complete covering of the second set of observations S in our candidate centers set, but there will be a partial covering. We can then use the greedy heuristic for the *partial cover problem* [19] and conduct a similar analysis. □(Theorem 10)

In contrast to the covering approach taken in the algorithm of Theorem 10, the algorithm of the following theorem uses an equation-solving technique.

Theorem 14 *For corruption probability $p < 1/2$, the class $HB_n^k[C]$ is learnable with an evaluator and a generator in time polynomial in n, $1/\epsilon$, $1/\delta$, 2^k and $(1-2p)^{-k}$.*

Proof: (Sketch) Let the target distribution $D \in HB_n^k[C]$ be $(\langle \vec{x}_1, p, q_1 \rangle, \ldots, \langle \vec{x}_k, p, q_k \rangle)$. Let X be the random variable representing the randomly chosen center vector (that is, $X = \vec{x}_i$ with probability q_i). Note that we do not have direct access to the random variable X.

Our algorithm for learning such a distribution makes use of a subroutine PROB which estimates the probability that a chosen set of bits of X are set to particular values. That is, PROB takes as input lists $i_1, \ldots, i_\ell \in [n]$ and $b_1, \ldots, b_\ell \in \{0,1\}$, and returns (with high probability) an estimate (to any given accuracy) of the probability that $X_{i_j} = b_j$ for $j = 1, \ldots, \ell$. Assuming for now that such a subroutine exists, we show how to learn D. Later, we sketch an implementation of the subroutine PROB.

To learn the distribution D, it suffices to learn the distribution of the random center X since the noise process is known. To do this, we use PROB to construct a binary tree T which represents an approximation of X's distribution (and that can be used for either generation or evaluation of the distribution D).

Each (internal) node of the tree T is labeled with an index $i \in [n]$ and a probability r. Each node has a 0-child and a 1-child. The leaves are labeled with an assignment $\vec{a} \in \{0,1\}^n$. We interpret such a tree as a representation of the distribution induced by the following process for choosing a vector $\vec{y}$: beginning at the root node labeled (i, r), we flip a biased coin with probability r of heads. If heads, we set $y_i = 1$ and we traverse to the 1-child of the current node; if tails, we set $y_i = 0$ and we move on to the 0-child. This process is repeated until a leaf node is reached with label $\vec{a}$. At this point, all the bits of $\vec{y}$ that have not already been assigned are set to the value given by $\vec{a}$.

A tree T representing approximately the distribution on centers X can be constructed using PROB as follows. Initially, the tree is empty. We begin by obtaining from PROB for each $i \in [n]$ an estimate of the probability that $X_i = 1$. If all of these estimates are very close to 0 or 1, then the probability must be high that X is equal to some vector $\vec{a}$; we therefore make the root a leaf labeled by $\vec{a}$. Clearly, T in this case is a good approximation of X.

On the other hand, if for some i, the estimated probability r that $X_i = 1$ is not close to 0 or 1, then we make the root a node labeled (i, r), and we recursively compute the subtrees subtended by the children of this node; these subtrees represent the distribution of the center X conditioned on X_i set to 0 or 1.

More specifically, we follow essentially the same procedure to compute the rest of the tree T. Suppose we are currently attempting to label a node in T that is reached by following a sequence of nodes labeled $i_1, \ldots, i_\ell$ where i_{j+1} is the b_j-child of i_j $(j = 1, \ldots, \ell - 1)$ and the current node is the b_ℓ-child of i_ℓ. For each $i \in [n]$, we use PROB to estimate the conditional probability that $X_i = 1$ given that $X_{i_j} = b_j$ for $j = 1, \ldots, \ell$. If, for all i, these estimates are close to 0 or 1, then this node is made into a leaf with the appropriate label. Otherwise, if the estimated conditional probability r for some index i is sufficiently far from 0 and 1, then the node is labeled (i, r), and the process continues recursively with the current node's children.

Assuming the reliability of subroutine PROB, we show in the full paper that the resulting tree T has at most k leaves. Briefly, this is shown by arguing that the number of centers $\vec{x}_i$ compatible with a node of the tree (so that the labels on the path to the node agree with the corresponding bits of $\vec{x}_i$) is strictly greater than the number of centers compatible with either of the node's children. Using this fact, it can be shown that only polynomially many calls to PROB are needed, and moreover that each call involves a list of at most k indices (that is, $\ell \leq k$ on each call to PROB). That T represents a good approximation of the distribution of X follows by a straightforward induction argument.

It remains then only to show how to construct the subroutine PROB. For ease of notation, assume without loss of generality that we are attempting to estimate the probability distribution on the first ℓ bits of X. We will show how this can be done in time polynomial in the usual parameters and $(1 - 2p)^{-\ell}$.

For a set $S \subseteq [\ell]$, let P_S be the probability that the chosen center vector X is such that $X_i = 1$ for $i \in S$ and $X_i = 0$ for $i \in [\ell] - S$. Our goal is to estimate one of the P_S's. Similarly, let Q_S be the probability that the observed vector Y is such that $Y_i = 1$ for $i \in S$ and $Y_i = 0$ for $i \in [\ell] - S$. Note that the Q_S's can be easily estimated from a random sample using Chernoff bounds.

Each Q_S can be written as a linear combination of the P_S's. Specifically,

$$Q_S = \sum_{T \subseteq [\ell]} (1-p)^{n-|S \triangle T|} p^{|S \triangle T|} P_T$$

(where $S \triangle T$ is the symmetric difference of S and T). That is, in matrix form, $Q = A_p P$ for some matrix A_p that depends only on the noise rate p. Since A_p is known, we thus can estimate the vector P by first estimating Q from a random sample by a vector $\hat{Q}$, and then computing $\hat{P} = A_p^{-1} \hat{Q}$.

To see that $\hat{P}$ is a good estimate of P, it can be shown that $||\hat{P} - P|| \leq |\lambda|^{-1} \cdot ||\hat{Q} - Q||$, where λ is the smallest eigenvalue of A_p. Moreover, it can shown that $\lambda = (1 - 2p)^\ell$ (details omitted).

This completes the sketch of PROB, and of the proof of Theorem 14. □(Theorem 14)

6 Hardness Results

In this section we give hardness results indicating the limits of efficient learnability in our model. Note that just as in the PAC model, we should distinguish between *representation dependent* hardness results, in which the intractability is the result of demanding that the learning algorithm output a hypothesis of certain syntactic form, and *representation independent* hardness results, in which a learning problem is shown hard regardless of the form of the hypothesis [20] and thus is inherently hard.

While we seek only results of the second type, we must still specify whether it is learning with an evaluator or learning with a generator that is hard, or both. We prove below that it is hard to learn certain probabilistic finite automata with an evaluator, under an assumption on the intractability of PAC learning parity functions with noise.

For learning with a generator, it is only for the powerful class of all polynomial-size circuit generators that we can prove hardness; the proof relies on the strong properties of pseudo-random functions [12].

6.1 Hardness of Learning Probabilistic Finite Automata with an Evaluator

We define a class of distributions PFA_n over $\{0,1\}^n$ generated by probabilistic finite automata. A distribution in PFA_n is defined by a finite automaton in which the number of states is bounded by a fixed polynomial in n, each state has a single outgoing transition labeled 0 and a single outgoing transition labeled 1, and each transition is also labeled by a probability such that for each state the sum of the two probabilities from that state is 1. There are no labels on the states. This automaton is used to generate n-bit strings by the following process: the automaton starts at its designated start state, and takes n steps. At each step, an outgoing transition is chosen at random according to its associated probability, and the $\{0,1\}$ label of the chosen transition is the next output bit. The resulting distribution has both polynomial-size generators and evaluators.

Abe and Warmuth [1] showed that it is hard in a representation dependent sense to learn a probabilistic automaton defined over a large alphabet. Here, we give evidence for the representation independent intractability of learning PFA_n with an evaluator (even when the alphabet has cardinality two). We argue this by demonstrating that the problem of learning parity functions in the presence of classification noise with respect to the uniform distribution can be embedded in the PFA_n learning problem. Thus we prove our theorem under the following conjecture, for which some evidence has been provided in recent papers [18, 3].

Conjecture 15 *(Noisy Parity Assumption) There is a constant $0 < \eta < \frac{1}{2}$ such that there is no efficient algorithm for learning parity functions under the uniform distribution in the PAC model with classification noise rate η.*

Theorem 16 *Under the Noisy Parity Assumption, the class PFA_n is not efficiently learnable with an evaluator.*

Proof: (Sketch) We show that for any parity function f_S on $\{0,1\}^{n-1}$, where $S \subseteq \{x_1, \ldots, x_{n-1}\}$ and $f_S(\vec{x}) = 1$ if and only if the parity of $\vec{x}$ on the set S is 1, there is a distribution D_S in PFA_n that is uniform on the first $n-1$ bits, and whose nth bit is f_S applied to the first $n-1$ bits with probability $1 - \eta$, and is the complement of this value with probability

η. Thus, the distribution D_S essentially generates random noisy labeled examples of f_S. This is easily accomplished by a probabilistic finite automaton with two parallel "tracks", the 0-track and the 1-track, of n levels each. If at any time during the generation of a string we are in the b-track, $b \in \{0,1\}$, this means that the parity of the string generated so far restricted to the variable set S is b. Let $s_{b,i}$ denote the ith state in the b-track. If the variable $x_i \notin S$ (so x_i is irrelevant to f_S), then both the 0 and 1 transitions from $s_{b,i}$ go to $s_{b,i+1}$ (there is no switching of tracks). If $x_i \in S$, then the 0-transition of $s_{b,i}$ goes to $s_{b,i+1}$, but the 1-transition goes to $s_{\neg b,i+1}$ (we switch tracks because the parity of S so far has changed). All these transitions are given probability $1/2$, so the bits are uniformly generated. Finally, from $s_{b,n-1}$ we make a b-transition with probability $1-\eta$ and a $\neg b$-transition with probability η. It is easily verified that this construction implements the promised noisy distribution for random labeled examples of f_S.

It can be shown that if $\hat{D}$ is a hypothesis evaluator satisfying $KL(D_S||\hat{D}) \leq \epsilon(1-\mathcal{H}(\eta))$, then for a random $\vec{x} \in \{0,1\}^{n-1}$ we can determine $f_S(\vec{x})$ with probability $1-\epsilon$ by checking which of $\vec{x}0$ and $\vec{x}1$ has larger probability under $\hat{D}$ and answering accordingly. This contradicts the Noisy Parity Assumption. □(Theorem 16)

6.2 Hardness of Learning Polynomial-Size Circuit Distributions with a Generator

While Theorem 16 demonstrates that we should not seek algorithms for learning probabilistic automata with an evaluator, it leaves open the possibility of learning with a generator. Which classes are hard to learn even with a generator?

Let $POLY_n$ denote the class of distributions over $\{0,1\}^n$ generated by circuits of size at most some fixed polynomial in n. In the following theorem, we show that $POLY_n$ is not efficiently learnable with a generator. The construction uses the strong properties of pseudo-random functions [12].

Theorem 17 *If there exists a one-way function, $POLY_n$ is not efficiently learnable with an evaluator or with a generator.*

Proof: (Sketch) We use the construction of small circuits indistinguishable from truly random functions due to Goldreich, Goldwasser and Micali [12]. Briefly, for every n there exists a class of functions $f_1, \ldots, f_{2^n} : \{0,1\}^n \rightarrow \{0,1\}^n$, each computable by a circuit of size polynomial in n, and with the following remarkable property: let k be chosen randomly, and let A be any polynomial-time algorithm provided with an oracle for the function f_k. After making a polynomial number of dynamically chosen queries $\vec{x}_1, \ldots, \vec{x}_{p(n)} \in \{0,1\}^n$ and receiving the responses $f_k(\vec{x}_1), \ldots, f_k(\vec{x}_{p(n)})$, algorithm A chooses any *exam* vector $\vec{x}$ satisfying $\vec{x} \neq \vec{x}_i$ for all $1 \leq i \leq p(n)$. A then receives $f_k(\vec{x})$ and a random $\vec{r} \in \{0,1\}^n$, but in a random order. Then the advantage that A has in distinguishing $f_k(\vec{x})$ from $\vec{r}$ vanishes faster than any inverse polynomial in n.

The hard subclass of distributions in $POLY_{2n}$ is defined as follows: for each of the functions f_k over $\{0,1\}^n$, let D_k denote the distribution over $\{0,1\}^{2n}$ that is uniform on the first n bits, but whose last n bits are always f_k applied to the first n bits. The fact that the D_k can be generated by polynomial-size circuits follows immediately from the small circuits for the f_k (in fact, the D_k have polynomial-size evaluators as well).

Now suppose for contradiction that A is a polynomial-time algorithm for learning $POLY$ with a generator. Then given an oracle for f_k, we can simulate A by generating many $2n$-bit vectors of the form $\langle \vec{x}_i, f_k(\vec{x}_i)\rangle$ by choosing $\vec{x} \in \{0,1\}^n$ randomly; these $2n$-bit vectors will be distributed exactly according to D_k. Let $\hat{D}$ denote the generator output by A following this simulation. Assume that the KL divergence is at most ℓ, that is,

$$KL(D_k||\hat{D}) = \sum_{\vec{x} \in \{0,1\}^n} \frac{1}{2^n} \log \frac{1}{\hat{D}[\vec{x}, f_k(\vec{x})]} - n \leq \ell.$$

The probability that $\hat{D}$ actually generates a correct pair $\langle \vec{x}, f_k(\vec{x})\rangle$ is simply $\sum_x \hat{D}[\vec{x}, f_k(\vec{x})]$. We claim that for at least a fraction $1/n$ of the $\vec{x}$, $\hat{D}[\vec{x}, f_k(\vec{x})] \geq 1/2^{2+\ell+n}$; otherwise the KL divergence would be more than ℓ. Therefore the probability that $\hat{D}$ generates a correct pair is at least $1/(4n2^\ell)$. Since only $p(n)$ of the $2^n/n$ correct pairs were drawn for the simulation of A, it follows that the probability that $\hat{D}$ outputs a new correct pair $\langle \vec{x}, f_k(\vec{x})\rangle$ is at least $1/5n2^\ell$. Therefore, for $\ell = O(\log n)$ this probability is an inverse polynomial, which is a contradiction. □(Theorem 17)

7 Distribution Learning and Compression

It is true in many probabilistic learning models that "compression implies learning": if there is an efficient algorithm that can always find a "short explanation" for a random sample, then that algorithm is a learning algorithm provided it is given a sufficiently large sample. This powerful principle goes by the name *Occam's Razor*, and it can be verified for many learning models, including our distribution learning model [5, 6, 21, 14].

In the distribution-free PAC model, the *converse* to Occam's Razor can be shown to hold as well [11, 22]. Specifically, if any class of polynomial-size circuits over $\{0,1\}^n$ is efficiently learnable in the distribution-free PAC model, then it is efficiently learnable by an algorithm whose hypothesis is a boolean circuit whose size depends polynomially on n but only *logarithmically* on $1/\epsilon$. (Such statements are interesting only in the computationally bounded setting; without computational constraints, they hold trivially.) This should be contrasted with the fact that for many distributions, it is possible to prove an $\Omega(1/\epsilon)$ lower bound on the number of examples any learning algorithm must see when learning under those specific distributions [6]. In other words, in the distribution-free PAC model it is *impossible* to construct a class of functions that is efficiently learnable *only* by an algorithm whose hypothesis stores a complete table of all the examples seen during training — there must always exist an efficient algorithm whose hypothesis manages to "forget" most of the sample.

Intriguingly, in our model, it seems entirely possible that there might be classes of distributions that are efficiently learnable *only* by "memorizing" algorithms — that is, algorithms whose hypothesis distribution has small log-loss, but whose size is not significantly smaller than the sample itself. It is interesting to note as an aside that many of the standard statistical algorithms (such as the nearest-neighbor and kernel-based algorithms surveyed by Izenman [16]) also involve the memorization of the entire sample.

We now make a concrete proposal for a counterexample to the converse of Occam's Razor for learning with a generator. We call the distribution class HC_n, standing for *Hidden Coin*, because each distribution can be thought of as generating a biased coin flip "hidden" in a number, with the property that no polynomial-time algorithm can determine the outcome of the coin flip, but the numbers are sufficient to generate further biased flips. The construction is simple,

and based on quadratic residues. For any n, each distribution in the class HC_n will be defined by a tuple $\langle p, q, r, z\rangle$. Here p and q are $n/4$-bit primes (let $N = p \cdot q$), $r \in [0, 1]$, and $z \in Z_N^*$ is any element such that $z \neq x^2 \ mod \ N$ for all $x \in Z_N^*$ (that is, z is a quadratic non-residue). The tuple $\langle p, q, r, z\rangle$ generates the following distribution: first a random $x \in Z_N^*$ is chosen. Then with probability r, we set $y = x^2 \ mod \ N$ (a residue), and with probability $1 - r$, we set $y = zx^2 \ mod \ N$ (a non-residue). The generated output is $(y, N) \in \{0, 1\}^n$. It is easy to verify that HC_n has both polynomial-size generators and evaluators.

Theorem 18 *The class HC_n is efficiently learnable with a generator, and under the Quadratic Residue Assumption [4] is not efficiently learnable with an evaluator.*

Proof: (Sketch) The hardness of learning with an evaluator is straightforward and omitted. The algorithm for learning with a generator simply takes a large sample $S = \langle (y_1, N), \ldots, (y_m, N)\rangle$ from the distribution. Note that if $\hat{r}$ is the fraction of the y_i appearing in S that are residues, then if $m = \Omega(1/\epsilon^2)$ we have $|r - \hat{r}| \leq \epsilon$ with high probability (although of course our polynomial-time algorithm has no obvious means of determining $\hat{r}$). Our algorithm simply outputs the entire sample S as its hypothesis representation. The distribution D_S defined by S is understood to be generated by first choosing $x \in Z_N^*$ randomly, then randomly selecting a y_i appearing in S, and letting the generated output be $(y_i x^2 \ mod \ N, N)$ (note that N is available from S). It is easy to see that D_S outputs a random residue with probability exactly $\hat{r}$, and thus has divergence at most ϵ to the target. □(Theorem 18)

The challenge is to find an efficient algorithm whose hypothesis is considerably more succinct than the one provided above, but we do not believe that such an algorithm exists. The following conjecture, if correct, would establish the failure of a strong converse to Occam's Razor for learning with a generator: unlike the PAC model, where hypothesis size always has an $O(\log(1/\epsilon))$ dependence on ϵ, we conjecture that for some positive α, an $\Omega(1/\epsilon^\alpha)$ hypothesis size dependence is required for the efficient learning of HC_n with a generator.

Conjecture 19 *Under the Quadratic Residue Assumption, for some $\alpha > 0$ there is no efficient algorithm for learning the class HC_n with a generator whose hypothesis size has an $O(1/\epsilon^\alpha)$ dependence on α.*

Acknowledgments

We would like to thank Nati Linial for helpful discussions on the inclusion-exclusion problem. Part of this research was conducted while Dana Ron, Ronitt Rubinfeld and Linda Sellie were visiting AT&T Bell Laboratories. Yishay Mansour was supported in part by the Israel Science Foundation administered by the Israel Academy of Science and Humanities, and by a grant from the Israeli Ministry of Science and Technology. Dana Ron would like to thank the Eshkol fellowship for its support. Ronitt Rubinfeld was supported by ONR Young Investigator Award N00014-93-1-0590 and United States-Israel Binational Science Foundation Grant 92-00226.

References

[1] Naoki Abe and Manfred K. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9(2–3):205–260, 1992.

[2] Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193–219, 1992.

[3] Avrim Blum, Merrick Furst, Michael Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *Pre-Proceedings of CRYPTO '93*, pages 24.1–24.10, 1993.

[4] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.

[5] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, April 1987.

[6] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[7] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[8] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[9] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[10] Paul Fischer and Hans Ulrich Simon. On learning ring-sum-expansions. *SIAM Journal on Computing*, 21(1):181–192, February 1992.

[11] Yoav Freund. An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 391–398, July 1992.

[12] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the Association for Computing Machinery*, 33(4):792–807, October 1986.

[13] Yuri Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42(3):346–398, 1991.

[14] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

[15] David Helmbold, Robert Sloan, and Manfred K. Warmuth. Learning integer lattices. *SIAM Journal on Computing*, 21(2):240–266, 1992.

[16] Alan Julian Izenman. Recent developments in nonparametric density estimation. *Journal of the American Statistical Association*, 86(413):205–224, March 1991.

[17] Jeff Kahn, Nathan Linial, and Alex Samorodintsky. Inclusion-exclusion: exact and approximate. Manuscript, 1993.

[18] Michael Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 392–401, 1993.

[19] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, August 1993.

[20] Michael Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 433–444, May 1989. To appear, *Journal of the Association for Computing Machinery*.

[21] Michael J. Kearns and Robert E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *31st Annual Symposium on Foundations of Computer Science*, pages 382–391, October 1990. To appear, *Journal of Computer and System Sciences*.

[22] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[23] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[24] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[25] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.