Learning from satisfying assignments

Anindya De*
Institute for Advanced Study

Ilias Diakonikolas†
University of Edinburgh

Rocco A. Servedio‡
Columbia University

Abstract

This paper studies the problem of learning "low-complexity" probability distributions over the Boolean hypercube $\{-1,1\}^n$. As in the standard PAC learning model, a learning problem in our framework is defined by a class $\mathcal{C}$ of Boolean functions over $\{-1,1\}^n$, but in our model the learning algorithm is given uniform random satisfying assignments of an unknown $f \in \mathcal{C}$ and its goal is to output a high-accuracy approximation of the uniform distribution over $f^{-1}(1)$. This distribution learning problem may be viewed as a demanding variant of standard Boolean function learning, where the learning algorithm only receives positive examples and — more importantly — must output a hypothesis function which has small *multiplicative* error (i.e. small error relative to the size of $f^{-1}(1)$).

As our main results, we show that the two most widely studied classes of Boolean functions in computational learning theory — linear threshold functions and DNF formulas — have efficient distribution learning algorithms in our model. Our algorithm for linear threshold functions runs in time $\mathrm{poly}(n, 1/\epsilon)$ and our algorithm for polynomial-size DNF runs in time $\mathrm{quasipoly}(n, 1/\epsilon)$. We obtain both these results via a general approach that combines a broad range of technical ingredients, including the complexity-theoretic study of approximate counting and uniform generation; the Statistical Query model from learning theory; and hypothesis testing techniques from statistics. A key conceptual and technical ingredient of this approach is a new kind of algorithm which we devise called a "densifier" and which we believe may be useful in other contexts.

We also establish limitations on efficient learnability in our model by showing that the existence of certain types of cryptographic signature schemes imply that certain learning problems in our framework are computationally hard. Via this connection we show that assuming the existence of sufficiently strong unique signature schemes, there are no sub-exponential time learning algorithms in our framework for intersections of two halfspaces, for degree-2 polynomial threshold functions, or for monotone 2-CNF formulas. Thus our positive results for distribution learning come close to the limits of what can be achieved by efficient algorithms.

*anindya@math.ias.edu. Research done while the author was a graduate student at UC Berkeley, supported by NSF award CCF-0915929 and Umesh Vazirani's Templeton Foundation Grant 21674.

†ilias.d@ed.ac.uk. Most of this work was done while the author was at UC Berkeley supported by a Simons Postdoctoral Fellowship.

‡rocco@cs.columbia.edu. Supported by NSF grant CCF-1115703, CCF-0915929.

1 Introduction

The learnability of Boolean functions has been an important research topic in theoretical computer science since Valiant's work [Val84] 30 years ago. The existence of a strong connection between learning and complexity theory is a recurring theme in this line of research, especially for the model of learning Boolean functions under the uniform distribution on the hypercube. More recently – over the past decade or so – the learnability of *probability distributions* has emerged as another important topic in TCS. Much of this recent work has been on learning various types of continuous distributions, such as mixtures of Gaussians, over high-dimensional spaces (see e.g., [Das99, KSV08, MV10, BS10] and many other papers); by and large research in this distribution-learning vein does not have a "complexity-theoretic flavor." One notable exception was the early paper of Kearns et al. [KMR$^+$94] (and followup work of Naor [Nao96]). The [KMR$^+$94] paper defined a model of learning discrete distributions over $\{-1,1\}^m$, where the distribution is viewed as being generated by an m-output, n-input circuit that is fed uniform input strings from $\{-1,1\}^n$. The [KMR$^+$94] paper studied how the complexity of learning such a distribution scales with the complexity of the circuit generating the distribution. However, as shown by [KMR$^+$94], even very simple circuits (depth-1 circuits of bounded fanin OR gates) can generate distributions that are hard to learn in this model.

In this paper we revisit the problem of learning discrete distributions over $\{-1,1\}^n$ from a complexity-theoretic perspective which is different from [KMR$^+$94]. While the [KMR$^+$94] work studies distributions that are generated by a simple multi-output circuit as described above, here we consider distributions that can be described as the *uniform distribution over the satisfying assignments of a low-complexity Boolean function*. In other words, for a Boolean function $f : \{-1,1\}^n \to \{-1,1\}$, we consider the distribution $\mathcal{U}_{f^{-1}(1)}$ which is the uniform distribution over $f^{-1}(1)$, the satisfying assignments of f. Thus, in our framework, a learning problem is defined by a class $\mathcal{C}$ of Boolean functions over $\{-1,1\}^n$. The unknown "target distribution" is $\mathcal{U}_{f^{-1}(1)}$ for an unknown $f \in \mathcal{C}$, and the learning algorithm receives independent draws from $\mathcal{U}_{f^{-1}(1)}$ (i.e. independent uniform positive examples of f). The goal of the learning algorithm is to output a hypothesis distribution D over $\{-1,1\}^n$ (or more precisely, a sampler for D) so that the total variation distance $d_{\mathrm{TV}}(D, \mathcal{U}_{f^{-1}(1)}) := \frac{1}{2}\sum_{x\in\{-1,1\}^n} |D(x) - \mathcal{U}_{f^{-1}(1)}(x)|$ is at most ϵ with probability $1-\delta$. We refer to such an algorithm for a class $\mathcal{C}$ of Boolean functions as a *distribution learning algorithm for* $\mathcal{C}$.

1.1 Motivation and Related Work. Our learning framework has some similarities to the model of "uniform-distribution learning from positive examples only" (see e.g. [DGL05, Den98]) since in both settings the input to the algorithm is a sample of points drawn uniformly at random from $f^{-1}(1)$. However, there are several important differences. One obvious difference is that in uniform-distribution learning from positive examples the goal is to output a hypothesis function h, whereas here our goal is to output a hypothesis *distribution*. A more significant difference is that the success criterion for our framework is much more demanding than for standard uniform-distribution learning. In uniform-distribution learning of a Boolean function f over the hypercube $\{-1,1\}^n$, the hypothesis h must satisfy $\mathbf{Pr}[h(x) \neq f(x)] \leq \epsilon$, where the probability is *uniform over all* 2^n *points in* $\{-1,1\}^n$. Thus, for a given setting of the error parameter ϵ, in uniform-distribution learning the constant -1 function is an acceptable hypothesis for any function f that has $|f^{-1}(1)| \leq \epsilon 2^n$; this essentially means that the learning algorithm gets a free pass whenever the function is relatively biased towards outputting -1. In contrast, in our learning framework we measure error by the total variation distance between $\mathcal{U}_{f^{-1}(1)}$ and the hypothesis distribution D, so no such "easy way out" is possible when $|f^{-1}(1)|$ is small; indeed the hardest instances in our learning scenario are often those for which $f^{-1}(1)$ is a very small fraction of $\{-1,1\}^n$. This means that we require a hypothesis with small *multiplicative* error relative to $|f^{-1}(1)|/2^n$ rather than the additive-error criterion that is standard in uniform-distribution learning. We are not aware of prior theoretical work on learning Boolean functions in which such a "multiplicative-error" criterion has been employed (though the routinely used notions of error, precision and recall in machine learning are similar to the multiplicative-error criterion used here).

(We further observe that, as detailed below, we prove negative results in our learning model for classes such as monotone 2-CNF formulas and degree-2 polynomial threshold functions. Since efficient algorithms are known for learning these classes of functions, our negative results show that our distribution learning model is indeed significantly more challenging than the standard uniform-distribution Boolean function learning model for some natural and well-studied classes of functions.)

Concurrent (but independent) to our work, Anderson *et al.* [AGR13] considered the following problem: Given access to random samples drawn uniformly from an unknown simplex $\mathcal{X}$ (i.e. an intersection of $n+1$ halfspaces) over $\mathbb{R}^n$, estimate the simplex. More precisely, their task is to output $n+1$ halfspaces $H_1, \ldots, H_{n+1}$ such that if $\mathcal{X}' = H_1 \cap \ldots \cap H_{n+1}$, then $d_{TV}(\mathcal{U}_X, \mathcal{U}_{X'}) \leq \epsilon$. Anderson *et al.* give a $\mathrm{poly}(n/\epsilon)$-time algorithm for this problem. Combining this with an efficient algorithm for sampling from convex bodies [DFK91], we get a $\mathrm{poly}(n/\epsilon)$-time algorithm that outputs a sampler for the distribution $\mathcal{U}_{X'}$. This is the same as our learning model for the class of intersections of $n+1$ halfspaces, but with one crucial difference: the underlying measure is the Lebesgue measure on $\mathbb{R}^n$ as opposed to the uniform measure on $\{-1,1\}^n$ (as it is in our case). The distinction between these two measures is indeed a very significant one; as we show in this paper, an analogous result is impossible (under a cryptographic hardness assumption) even for an intersection of *two* halfspaces for the uniform measure on $\{-1,1\}^n$. Perhaps not too surprisingly, the techniques of Anderson *et al.* (algorithmic convex geometry and Independent Component Analysis) are rather disjoint from the techniques in this paper.

On the hardness side, Naor [Nao96] constructed a family $\mathcal{F}$ of explicit distributions such that under cryptographic assumptions, there is no efficient learning algorithm for $\mathcal{F}$. This family $\mathcal{F}$ is "low-complexity" in the sense that there is an efficient algorithm which given random samples from an unknown distribution $X \in \mathcal{F}$ and an input x, can compute $\mathbf{Pr}[X = x]$ to high accuracy. While our hardness results are for a different problem than that of Naor, the same intuition underlies both the results (i.e. using some modification of secure signature schemes). However, because our requirements are different, we need to use a different construction; in particular, our results are based on unique signature schemes whereas the construction in [Nao96] employs a modification of NIZK based signature schemes.

We close this subsection with two motivations for the study of our framework, one non-technical and one technical. Starting with the non-technical one, we briefly note that learning scenarios of the sort that we consider — in which (i) the learner is given access only to positive examples over some discrete space, (ii) positive examples are potentially a very sparse subset of all possible examples, and (iii) the learner's goal is to generate new positive examples — arise quite naturally in a range of real-world settings. As one example, a language learner (such as a baby or an adult learning a new language) is typically exposed only to correct utterances (positive examples), which comprise a tiny fraction of all possible vocalizations (sparsity), and successful learning essentially amounts to being able to produce new correct utterances (positive examples).

As a more technical motivation, we note that certain specific learning results in our framework *must* be achieved as a first step in order to beat the "curse of dimensionality" for some natural continuous high-dimensional density estimation problems. A "k-piece d-dimensional histogram" is a probability distribution p over the domain $[0,1]^d$ of the following sort: $[0,1]^d$ is partitioned into k axis-aligned hyper-rectangles $R_1, \ldots, R_k$, and the distribution p is piecewise constant over each rectangle R_i. What is the complexity (in terms of sample size and running time) of learning such a distribution from random examples – standard approaches give algorithms whose complexity is *exponential* in d, but is there a way to do better? Our learning model turns out to be highly relevant to this question; an easy reduction shows that a special case of the k-piece d-dimensional histogram learning problem, corresponds exactly to the problem of learning the uniform distribution over satisfying assignments of an unknown size-k decision tree over d Boolean variables. Our positive results for DNF, described later in this introduction, give a $d^{O(\log(k/\epsilon))}$ algorithm for this problem, and thus break the "curse of dimensionality" for this special case of the multidimensional histogram learning problem.

1.2 Our results. We give both positive and negative results for our learning model.

Positive results: A general technique and its instantiations. We begin by presenting a general technique for designing learning algorithms in our model. This technique combines approximate uniform generation and counting algorithms from complexity theory, Statistical Query (SQ) learning algorithms from computational learning theory, and hypothesis testing techniques from statistics. A key new ingredient which lets us combine these disparate tools is an algorithm called a "densifier" which we introduce and define in Section 3. Roughly speaking, the densifier lets us prune the entire space $\{-1,1\}^n$ to a set S which (essentially) contains all of $f^{-1}(1)$ and is not too much larger than $f^{-1}(1)$ (so $f^{-1}(1)$ is "dense" in S). By generating approximately uniform elements of S it is possible to run an SQ learning algorithm and obtain a high-accuracy hypothesis for f. This hypothesis can be used, in conjunction with an approximate uniform generation algorithm, to obtain an efficiently samplable distribution which is close to the uniform distribution over $f^{-1}(1)$. (The approximate counting algorithm is needed for technical reasons which we explain in Section 3.1.) In Section 3 we describe this technique in detail and prove a general result establishing its effectiveness.

In Sections 4 and 5 we give our two main positive results which are obtained by applying this general technique to specific classes of functions. The first of these is the class $\mathbf{LTF}$ of all linear threshold functions (LTFs) over $\{-1,1\}^n$. We prove:

Theorem 1. (Informal statement) *There is a $\mathrm{poly}(n, 1/\epsilon)$-time algorithm for learning $\mathcal{U}_{f^{-1}(1)}$ where f is any LTF over $\{-1,1\}^n$.*

Our main technical contribution here is to construct a densifier for LTFs; we do this by carefully combining known efficient online learning algorithms for LTFs (based on interior-point methods for linear programming) [MT94] with known algorithms for approximate uniform generation and counting of satisfying assignments of LTFs.

As mentioned before, our distribution learning algorithms essentially entail learning the underlying Boolean function with a *multiplicative error* guarantee. Indeed, as a by-product of our approach in the proof of Theorem 1, we also get the following statement which we feel is of independent interest:

Theorem 2. (Informal statement) *There is a $\mathrm{poly}(n, 1/\epsilon)$-time algorithm which given random samples from the distribution $\mathcal{U}_{f^{-1}(1)}$ (for an unknown LTF f), outputs a hypothesis h such that $\mathbf{Pr}_{z\in\mathcal{U}_n}[f(z) \neq h(z)] \leq \epsilon \cdot |f^{-1}(1)|/2^n$.*

Our second main positive result for a specific class, in Section 5, is for the well-studied class $\mathbf{DNF}_{n,s}$ of all size-s DNF formulas over n Boolean variables. Here our main technical contribution is to give a densifier which runs in time $n^{O(\log(s/\epsilon))}$ and outputs a DNF formula. A challenge here is that known SQ algorithms for learning DNF formulas require time exponential in $n^{1/3}$. To get around this, we show that our densifier's output DNF is an OR over $n^{O(\log(s/\epsilon))}$ "metavariables" (corresponding to all possible conjunctions that could be present in the DNF output by the densifier), and that it is possible to apply known *malicious noise tolerant* SQ algorithms for learning *sparse disjunctions* as the SQ-learning component of our general approach. Since efficient approximate uniform generation and approximate counting algorithms are known [JVV86, KL83] for DNF formulas, with the above densifier and SQ learner we can carry out our general technique, and we thereby obtain our second main positive result for a specific function class:

Theorem 3. (Informal statement) *There is a $n^{O(\log(s/\epsilon))}$-time algorithm for learning $\mathcal{U}_{f^{-1}(1)}$ where f is any s-term DNF formula over $\{-1,1\}^n$.*

Similar to Theorem 2, our approach gives a learning algorithm for DNFs (over the uniform distribution on $\{-1,1\}^n$) with a *multiplicative error guarantee.*

Theorem 4. (Informal statement) *There is a $n^{O(\log(s/\epsilon))}$-time algorithm which given random samples from $\mathcal{U}_{f^{-1}(1)}$ (for an unknown DNF f), outputs a hypothesis h such that $\mathbf{Pr}_{z\in\mathcal{U}_n}[f(z) \neq h(z)] \leq \epsilon \cdot |f^{-1}(1)|/2^n$.*

We emphasize that our positive results for LTFs and DNFs go well beyond the standard techniques used to learn these classes in other less demanding models. As evidence of this, observe that Theorem 2 and Theorem 4 give learning algorithms for LTFs and DNFs with *multiplicative* error ϵ whereas all previous approaches in the learning theory literature incur an additive error. We also believe that the use of approximate counting and approximate uniform generation algorithms is novel in this learning context (as well as the new notion of a "densifier" which we introduce in this work) and may be of use elsewhere.

Negative results based on cryptography. We establish strong negative results for our learning model via a connection to signature schemes from public-key cryptography. Intuitively, viewing $\mathcal{U}_{f^{-1}(1)}$ as the uniform distribution over signed messages, the ability to construct a high-accuracy hypothesis distribution D given samples from $\mathcal{U}_{f^{-1}(1)}$ implies the ability to generate new signed messages, which contradicts the definition of a secure signature scheme. However, there are significant gaps in this rough intuition, and getting around these gaps requires the use of more specialized machinery, namely *unique* signature schemes [MRV99, Lys02]. Building on this intuition, we establish the following negative results which show that our positive results (for LTFs and DNFs) lie quite close to the boundary of what can be efficiently learned in our model (see Section F.1 for a precise statement):

Theorem 5. (Informal statement) *Under known constructions of secure signature schemes, there is no subexponential-time algorithm for learning $\mathcal{U}_{f^{-1}(1)}$ where f is (i) an unknown monotone 2-CNF formula; (ii) an unknown intersection of two halfspaces; or (iii) an unknown degree-2 polynomial threshold function.*

Structure of this paper. After the preliminaries in Section 2, we present our general algorithmic technique in Section 3. In Sections 4 and 5 we apply this technique to obtain efficient learning algorithms for LTFs and DNFs respectively. Because of space constraints our hardness results are in Appendix F.

2 Preliminaries and Useful Tools

Notation and definitions. For $n \in \mathbb{Z}_+$, we will denote by $[n]$ the set $\{1, \ldots, n\}$. For a distribution D over a finite set $\mathcal{W}$ we denote by $D(x)$, $x \in \mathcal{W}$, the probability mass that D assigns to point x, so $D(x) \geq 0$ and $\sum_{x \in \mathcal{W}} D(x) = 1$. For $S \subseteq \mathcal{W}$, we write $D(S)$ to denote $\sum_{x \in S} D(x)$. For a random variable x, we write $x \sim D$ to indicate that x follows distribution D. Let D, D' be distributions over $\mathcal{W}$. The *total variation distance* between D and D' is $d_{\mathrm{TV}}(D, D') \stackrel{\text{def}}{=} \max_{S \subseteq \mathcal{W}} |D(S) - D'(S)|$.

We will denote by $\mathcal{C}_n$, or simply $\mathcal{C}$, a Boolean concept class, i.e., a class of functions mapping $\{-1, 1\}^n$ to $\{-1, 1\}$. We usually consider syntactically defined classes of functions such as the class of all n-variable linear threshold functions or the class of all n-variable s-term DNF formulas. We stress that throughout this paper a class $\mathcal{C}$ is viewed as a *representation class*. Thus we will say that an algorithm "takes as input a function $f \in \mathcal{C}$" to mean that the input of the algorithm is a *representation* of $f \in \mathcal{C}$.

We will use the notation $\mathcal{U}_n$ (or simply $\mathcal{U}$, when the dimension n is clear from the context) for the uniform distribution over $\{-1, 1\}^n$. Let $f : \{-1, 1\}^n \to \{-1, 1\}$. We will denote by $\mathcal{U}_{f^{-1}(1)}$ the uniform distribution over satisfying assignments of f. Let D be a distribution over $\{-1, 1\}^n$ with $0 < D(f^{-1}(1)) < 1$. We write $D_{f,+}$ to denote the conditional distribution D restricted to $f^{-1}(1)$; so for $x \in f^{-1}(1)$ we have $D_{f,+}(x) = D(x)/D(f^{-1}(1))$. Observe that, with this notation, we have that $\mathcal{U}_{f^{-1}(1)} \equiv \mathcal{U}_{f,+}$.

We use familiar notions of samplers, approximate counting, and approximate uniform generation; see Appendix A for precise definitions of these notions and of the learning model that we study.

Hypothesis testing. Our algorithms work by generating a collection of hypothesis distributions, one of which is close to the target distribution $\mathcal{U}_{f^{-1}(1)}$. Thus, we need a way to select a high-accuracy hypothesis distribution from a pool of candidate distributions which contains at least one high-accuracy hypothesis.

This problem has been well studied, see e.g. Chapter 7 of [DL01]. We use the following result which is an extension of Lemma C.1 of [DDS12a] (see see Appendix B for a discussion and proof):

Proposition 6. *Let D be a distribution over a finite set $\mathcal{W}$ and $\mathcal{D}_\epsilon = \{D_j\}_{j=1}^N$ be a collection of N distributions over $\mathcal{W}$ with the property that there exists $i \in [N]$ such that $d_{\mathrm{TV}}(D, D_i) \leq \epsilon$. There is an algorithm $\mathcal{T}^D$ which is given an accuracy parameter ϵ, a confidence parameter δ, and is provided with access to (i) samplers for D and D_k, for all $k \in [N]$, and (ii) a $(1+\beta)$–approximate evaluation oracle $\mathrm{EVAL}_{D_k}(\beta)$, for all $k \in [N]$, which, on input $w \in \mathcal{W}$, deterministically outputs a value $\widetilde{D}_k^\beta(w)$, such that $D_k(w)/(1+\beta) \leq \widetilde{D}_k^\beta(w) \leq (1+\beta)D_k(w)$, where $\beta > 0$ is any parameter satisfying $(1+\beta)^2 \leq 1 + \epsilon/8$. This algorithm has the following behavior: It makes $m = O\left((1/\epsilon^2) \cdot (\log N + \log(1/\delta))\right)$ draws from D and from each D_k, $k \in [N]$, and $O(m)$ calls to each oracle $\mathrm{EVAL}_{D_k}(\beta)$, $k \in [N]$, performs $O(mN^2)$ arithmetic operations, and with probability $1-\delta$ outputs an index $i^\star \in [N]$ that satisfies $d_{\mathrm{TV}}(D, D_{i^\star}) \leq 6\epsilon$.*

3 A general technique for learning in our model

In this section we present a general technique for designing learning algorithms in our model. Our main positive results follow this framework.

At the heart of our approach is a new type of algorithm which we call a *densifier* for a concept class $\mathcal{C}$. Roughly speaking, this is an algorithm which, given uniform random positive examples of an unknown $f \in \mathcal{C}$, constructs a set S which (essentially) contains all of $f^{-1}(1)$ and which is such that $f^{-1}(1)$ is "dense" in S. Our main result in this section, Theorem 8, states (roughly speaking) that the existence of (i) a computationally efficient densifier, (ii) an efficient approximate uniform generation algorithm, (iii) an efficient approximate counting algorithm, and (iv) an efficient *statistical query (SQ)* learning algorithm, together suffice to yield an efficient algorithm for our distribution learning problem.

Recall that the *statistical query* (SQ) learning model is a natural restriction of the PAC learning model in which a learning algorithm is allowed to obtain estimates of statistical properties of the examples but cannot directly access the examples themselves. Let D be a distribution over $\{-1,1\}^n$. In the SQ model [Kea98], the learning algorithm has access to a *statistical query oracle*, $\mathrm{STAT}(f, D)$, to which it can make queries of the form (χ, τ), where $\chi : \{-1,1\}^n \times \{-1,1\} \to [-1,1]$ is the *query function* and $\tau > 0$ is the *tolerance*. The oracle responds with a value v such that $|\mathbf{E}_{x\sim D}[\chi(x, f(x))] - v| \leq \tau$, where $f \in \mathcal{C}$ is the target concept. The goal of the algorithm is to output a hypothesis $h : \{-1,1\}^n \to \{-1,1\}$ such that $\mathbf{Pr}_{x\sim D}[h(x) \neq f(x)] \leq \epsilon$. (See Appendix A for a precise definition of Statistical Query learning.) We sometimes write an "(ϵ, δ)–SQ learning algorithm" to explicitly state the accuracy parameter ϵ and confidence parameter δ.

To state our main result, we introduce the notion of a *densifier* for a class $\mathcal{C}$ of Boolean functions. Intuitively, a densifier is an algorithm which is given access to samples from $\mathcal{U}_{f^{-1}(1)}$ (where f is an unknown element of $\mathcal{C}$) and outputs a subset $S \subseteq \{-1,1\}^n$ which is such that (i) S contains "almost all" of $f^{-1}(1)$, but (ii) S is "much smaller" than $\{-1,1\}^n$ – small enough that $f^{-1}(1) \cap S$ is (moderately) "dense" in S.

Definition 7. *Fix a function $\gamma(n, 1/\epsilon, 1/\delta)$ taking values in $(0,1]$ and a class $\mathcal{C}$ of n-variable Boolean functions. An algorithm $\mathcal{A}_{\mathrm{den}}^{(\mathcal{C},\mathcal{C}')}$ is said to be* a γ-densifier for function class $\mathcal{C}$ using class $\mathcal{C}'$ *if it has the following behavior: For every $\epsilon, \delta > 0$, every $1/2^n \leq \widehat{p} \leq 1$, and every $f \in \mathcal{C}$, given as input $\epsilon, \delta, \widehat{p}$ and a set of independent samples from $\mathcal{U}_{f^{-1}(1)}$, the following holds: Let $p \stackrel{def}{=} \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1]$. If $p \leq \widehat{p} < (1+\epsilon)p$, then with probability at least $1-\delta$, algorithm $\mathcal{A}_{\mathrm{den}}^{(\mathcal{C},\mathcal{C}')}$ outputs a function $g \in \mathcal{C}'$ such that:*

(a) $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq 1 - \epsilon$, *and* *(b)* $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x) = 1] \geq \gamma(n, 1/\epsilon, 1/\delta)$.

We will sometimes write an "$(\epsilon, \gamma, \delta)$–densifier" to explicitly state the parameters in the definition.

Our main conceptual approach is summarized in the following theorem:

Theorem 8 (General Algorithmic Approach)**.** *Let $\mathcal{C}, \mathcal{C}'$ be classes of n-variable Boolean functions. Suppose that*

- $\mathcal{A}_{\mathrm{den}}^{(\mathcal{C},\mathcal{C}')}$ *is an $(\epsilon, \gamma, \delta)$-densifier for $\mathcal{C}$ using $\mathcal{C}'$ running in time $T_{\mathrm{den}}(n, 1/\epsilon, 1/\delta)$.*
- $\mathcal{A}_{\mathrm{gen}}^{\mathcal{C}'}$ *is an (ϵ, δ)-approximate uniform generation algorithm for $\mathcal{C}'$ running in time $T_{\mathrm{gen}}(n, 1/\epsilon, 1/\delta)$.*
- $\mathcal{A}_{\mathrm{count}}^{\mathcal{C}'}$ *is an (ϵ, δ)-approximate counting algorithm for $\mathcal{C}'$ running in time $T_{\mathrm{count}}(n, 1/\epsilon, 1/\delta)$.*
- $\mathcal{A}_{\mathrm{SQ}}^{\mathcal{C}}$ *is an (ϵ, δ)-*SQ *learning algorithm for $\mathcal{C}$ such that: $\mathcal{A}_{\mathrm{SQ}}^{\mathcal{C}}$ runs in time $t_1(n, 1/\epsilon, 1/\delta)$, $t_2(n)$ is the maximum time needed to evaluate any query provided to $\mathrm{STAT}(f, D)$, and $\tau(n, 1/\epsilon)$ is the minimum value of the tolerance parameter ever provided to $\mathrm{STAT}(f, D)$ in the course of $\mathcal{A}_{\mathrm{SQ}}^{\mathcal{C}}$'s execution.*

Then there exists a distribution learning algorithm $\mathcal{A}^{\mathcal{C}}$ for $\mathcal{C}$. The running time of $\mathcal{A}^{\mathcal{C}}$ is polynomial in $T_{\mathrm{den}}(n, 1/\epsilon, 1/\delta)$, $1/\gamma$, $T_{\mathrm{gen}}(n, 1/\epsilon, 1/\delta)$, $T_{\mathrm{count}}(n, 1/\epsilon, 1/\delta)$, $t_1(n, 1/\epsilon, 1/\delta)$, $t_2(n)$ and $1/\tau(n, 1/\epsilon)$ provided that $T_{\mathrm{den}}(\cdot)$, $T_{\mathrm{gen}}(\cdot)$, $T_{\mathrm{count}}(\cdot)$, $t_1(\cdot)$, $t_2(\cdot)$ and $\tau(\cdot)$ are polynomial in their input parameters.

Sketch of the algorithm. The distribution learning algorithm $\mathcal{A}^{\mathcal{C}}$ for $\mathcal{C}$ works in three main conceptual steps. Let $f \in \mathcal{C}$ be the unknown target function and recall that our algorithm $\mathcal{A}^{\mathcal{C}}$ is given access to samples from $\mathcal{U}_{f^{-1}(1)}$.

(1) In the first step, $\mathcal{A}^{\mathcal{C}}$ runs the densifier $\mathcal{A}_{\mathrm{den}}^{(\mathcal{C},\mathcal{C}')}$ on a set of samples from $\mathcal{U}_{f^{-1}(1)}$. Let $g \in \mathcal{C}'$ be the output function of $\mathcal{A}_{\mathrm{den}}^{(\mathcal{C},\mathcal{C}')}$.

Note that by setting the input to the approximate uniform generation algorithm $\mathcal{A}_{\mathrm{gen}}^{\mathcal{C}'}$ to g, we obtain an approximate sampler C_g for $\mathcal{U}_{g^{-1}(1)}$. The output distribution D' of this sampler is (by definition of an approximate uniform generation algorithm, see Definition 18) supported on $g^{-1}(1)$ and is close to $D = \mathcal{U}_{g^{-1}(1)}$ in total variation distance.

(2) The second step is to run the SQ-algorithm $\mathcal{A}_{\mathrm{SQ}}^{\mathcal{C}}$ to learn the function $f \in C$ under the distribution D. Let h be the hypothesis constructed by $\mathcal{A}_{\mathrm{SQ}}^{\mathcal{C}}$.

(3) In the third and final step, the algorithm simply samples from C_g until it obtains an example x that has $h(x) = 1$, and outputs this x.

Remark 9. The reader may have noticed that the above sketch does not seem to use the approximate counting algorithm $\mathcal{A}_{\mathrm{count}}^{\mathcal{C}'}$; we will revisit this point below.

3.1 Intuition, motivation and discussion. To motivate the high-level idea behind our algorithm, consider a setting in which $f^{-1}(1)$ is only a tiny fraction (say $1/2^{\Theta(n)}$) of $\{-1, 1\}^n$. It is intuitively clear that we would like to use some kind of a learning algorithm in order to come up with a good approximation of $f^{-1}(1)$, but we need this approximation to be accurate at the "scale" of $f^{-1}(1)$ itself rather than at the scale of all of $\{-1, 1\}^n$, so we need some way to ensure that the learning algorithm's hypothesis is accurate at this small scale. By using a densifier to construct g such that $g^{-1}(1)$ is not too much larger than $f^{-1}(1)$, we can use the distribution $D = \mathcal{U}_{g^{-1}(1)}$ to run a learning algorithm and obtain a good approximation of $f^{-1}(1)$ at the desired scale. (Since $d_{\mathrm{TV}}(D, D')$ is small, this implies we also learn f with respect to D'.)

To motivate our use of an SQ learning algorithm rather than a standard PAC learning algorithm, observe that there seems to be no way to obtain correctly labeled examples distributed according to D. However, we show that it *is* possible to accurately simulate statistical queries under D having access only to random positive examples from $f^{-1}(1)$ and to unlabeled examples drawn from D (subject to additional technical

caveats discussed in the appendix). We discuss the issue of how it is possible to successfully use an SQ learner in our setting in more detail below.

Discussion and implementation issues. While the three main conceptual steps (1)-(3) of our algorithm may (hopefully) seem quite intuitive in light of the preceding motivation, a few issues immediately arise in thinking about how to implement these steps. The first one concerns running the SQ-algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ in Step 2 to learn f under distribution D (recall that $D = \mathcal{U}_{g^{-1}(1)}$ and is close to D'). Our algorithm $\mathcal{A}^{\mathcal{C}}$ needs to be able to efficiently simulate $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ given its available information. While it would be easy to do so given access to random labeled examples $(x, f(x))$, where $x \sim D$, such information is not available in our setting. To overcome this obstacle, we show (see Proposition 29) that for *any* samplable distribution D, we can efficiently simulate a statistical query algorithm under D using samples from $D_{f,+}$. This does not quite solve the problem, since we only have samples from $\mathcal{U}_{f^{-1}(1)}$. However, we show (see Claim 32) that for our setting, i.e., for $D = \mathcal{U}_{g^{-1}(1)}$, we can simulate a sample from $D_{f,+}$ by a simple rejection sampling procedure using samples from $\mathcal{U}_{f^{-1}(1)}$ and query access to g.

Some more issues remain to be handled. First, the simulation of the statistical query algorithm sketched in the previous paragraph only works under the assumption that we are given a sufficiently accurate approximation $\widetilde{b_f}$ of the probability $\mathbf{Pr}_{x\sim D}[f(x) = 1]$. (Intuitively, our approximation should be smaller than the smallest tolerance τ provided to the statistical query oracle by the algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$.) Second, by Definition 7, the densifier only succeeds under the assumption that it is given in its input an $(1+\epsilon)$-multiplicative approximation $\widehat{p}$ to $p = \mathbf{Pr}_{x\in\mathcal{U}_n}[f(x) = 1]$.

We handle these issues as follows: First, we show (see Claim 33) that, given an accurate estimate $\widehat{p}$ and a "dense" function $g \in \mathcal{C}'$, we can use the approximate counting algorithm $\mathcal{A}^{\mathcal{C}'}_{\mathrm{count}}$ to efficiently compute an accurate estimate $\widetilde{b_f}$. (This is one reason why Theorem 8 requires an approximate counting algorithm for $\mathcal{C}'$.) To deal with the fact that we do not a priori have an accurate estimate $\widehat{p}$, we run our sketched algorithm for all possible values of $\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1]$ in an appropriate multiplicative "grid" of size $N = O(n/\epsilon)$, covering all possible values from $1/2^n$ to 1. We thus obtain a set $\mathcal{D}$ of N candidate distributions one of which is guaranteed to be close to the true distribution $\mathcal{U}_{f^{-1}(1)}$ in variation distance. At this point, we would like to apply our hypothesis testing machinery (Proposition 6) to find such a distribution. However, in order to use Proposition 6, in addition to sample access to the candidate distributions (and the distribution being learned), we also require a *multiplicatively accurate* approximate evaluation oracle to evaluate the probability mass of any point under the candidate distributions. We show (see Lemma 43) that this is possible in our generic setting, using properties of the densifier and the approximate counting algorithm $\mathcal{A}^{\mathcal{C}'}_{\mathrm{count}}$ for $\mathcal{C}'$.

This concludes the overview of our approach; see Appendix C for a full proof of Theorem 8.

4 Linear Threshold Functions

In this section we apply our general framework from Section 3 to obtain a polynomial time distribution learning algorithm for n-variable linear threshold functions over $\{-1, 1\}^n$. More formally, we prove:

Theorem 10. *There is an algorithm $\mathcal{A}^{\mathbf{LTF}}$ which is a $\mathrm{poly}(n, 1/\epsilon, 1/\delta)$-time distribution learning algorithm for the class $\mathbf{LTF}_n$ of n-variable linear threshold functions over $\{-1, 1\}^n$.*

The above theorem will follow as an application of Theorem 8 for $\mathcal{C}' = \mathcal{C} = \mathbf{LTF}_n$. As detailed in Appendix D, the literature provides us with three of the four ingredients that our general approach requires for LTFs – approximate uniform generation, approximate counting, and Statistical Query learning – and our main technical contribution is giving the fourth necessary ingredient, a densifier. This is the main technical contribution of this section and is summarized in the following theorem:

Theorem 11. *Set* $\gamma(\epsilon,\delta,n) \stackrel{def}{=} \Theta\left(\delta/(n^2\log n)\right)$. *There is an* (ϵ,γ,δ)*–densifier* $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}$ *for* $\mathbf{LTF}_n$ *that, for any input parameters* $0<\epsilon,\delta$, $1/2^n \le \widehat{p} \le 1$, *outputs a function* $g \in \mathbf{LTF}_n$ *and runs in time* $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$.

Discussion and intuition. Before we prove Theorem 11, we provide some intuition. Let $f \in \mathbf{LTF}_n$ be the unknown LTF and suppose that we would like to design an (ϵ,γ,δ)–densifier $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}$ for f. That is, given sample access to $\mathcal{U}_{f^{-1}(1)}$, and a number $\widehat{p}$ satisfying $p \le \widehat{p} < (1+\epsilon)p$, where $p = \mathbf{Pr}_{x\in\mathcal{U}_n}[f(x)=1]$, we would like to efficiently compute (a weights–based representation for) an LTF $g:\{-1,1\}^n \to \{-1,1\}$ such that the following conditions are satisfied:

(a) $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x)=1] \ge 1-\epsilon$; and (b) $\mathbf{Pr}_{x\sim\mathcal{U}_n}[g(x)=1] \le (1/\gamma)\cdot\mathbf{Pr}_{x\sim\mathcal{U}_n}[f=1]$.

(While condition (b) above appears slightly different than property (b) in our Definition 7, because of property (a), the two statements are essentially equivalent up to a factor of $1/(1-\epsilon)$ in the value of γ.)

We start by noting that it is easy to handle the case that $\widehat{p}$ is large. In particular, observe that if $\widehat{p} \ge 2\gamma$ then $p = \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1] \ge \widehat{p}/(1+\epsilon) \ge \widehat{p}/2 \ge \gamma$, and we can just output $g \equiv 1$ since it satisfies both properties of the definition. For the following intuitive discussion we will assume that $\widehat{p} \le 2\gamma$.

Recall that our desired function g is an LTF, i.e., $g(x) = \mathrm{sign}(v\cdot x - t)$, for some $(v,t) \in \mathbb{R}^{n+1}$. Recall also that our densifier has sample access to $\mathcal{U}_{f^{-1}(1)}$, so it can obtain random positive examples of f, each of which gives a linear constraint over the v,t variables. Hence a natural first approach is to attempt to construct an appropriate linear program over these variables whose feasible solutions satisfy conditions (a) and (b) above. We begin by analyzing this approach; while it turns out to not quite work, it will gives us valuable intuition for our actual algorithm, which is presented further below.

Following this approach, condition (a) is relatively easy to satisfy. Indeed, consider any $\epsilon > 0$ and suppose we want to construct an LTF $g = \mathrm{sign}(v\cdot x - t)$ such that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x)=1] \ge 1-\epsilon$. This can be done as follows: draw a set S_+ of $N_+ = \Theta\left((1/\epsilon)\cdot(n^2+\log(1/\delta))\right)$ samples from $\mathcal{U}_{f^{-1}(1)}$ and consider a linear program $\mathcal{LP}_+$ with variables $(w,\theta) \in \mathbb{R}^{n+1}$ that enforces all these examples to be positive. That is, for each $x \in S_+$, we will have an inequality $w\cdot x \ge \theta$. It is clear that $\mathcal{LP}_+$ is feasible (any weights–based representation for f is a feasible solution) and that it can be solved in $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ time, since it is defined by $O(N_+)$ many linear constraints and the coefficients of the constraint matrix are in $\{\pm 1\}$. The following simple claim, proved in Appendix D, shows that with high probability any feasible solution of $\mathcal{LP}_+$ satisfies condition (a):

Claim 12. *With probability* $1-\delta$ *over* S_+, any $g \in \mathbf{LTF}_n$ *consistent with* S_+ *satisfies condition (a).*

The above claim implies that with high probability *any* feasible solution (w^*,θ^*) to $\mathcal{LP}_+$ has $g^*(x) = \mathrm{sign}(w^*\cdot x - \theta^*)$ satisfy condition (a), but an arbitrary feasible solution to $\mathcal{LP}_+$ is by no means guaranteed to satisfy condition (b). (Note for example that the constant 1 function is certainly feasible for $\mathcal{LP}_+$.) Hence, a natural idea is to include additional constraints in our linear program so that condition (b) is also satisfied.

Along these lines, consider the following procedure: Draw a set S_- of $N_- = \lfloor \delta/\widehat{p} \rfloor$ uniform unlabeled samples from $\{-1,1\}^n$ and label them negative. That is, for each sample $x \in S_-$, we add the constraint $w\cdot x < \theta$ to our linear program. Let $\mathcal{LP}$ be the linear program that contains all the constraints defined by $S_+ \cup S_-$. It is not hard to prove that with probability at least $1-2\delta$ over the sample S_-, we have that $S_- \subseteq f^{-1}(-1)$ and hence (any weight based representation of) f is a feasible solution to $\mathcal{LP}$. In fact, it is possible to show that *if* γ *is sufficiently small* — roughly, $\gamma \le \delta/\left(4(n^2+\log(1/\delta))\right)$ is what is required — then with high probability each solution to $\mathcal{LP}$ also satisfies condition (b). The catch, of course, is that the above procedure is not computationally efficient because N_- may be very large – if $\widehat{p}$ is very small, then it is infeasible even to write down the linear program $\mathcal{LP}$.

Algorithm Description. The above discussion motivates our actual densifier algorithm as follows: The problem with the above described naive approach is that it generates (the potentially very large set) S_- all

at once at the beginning of the algorithm. Note that having a large set S_- is not necessarily in and of itself a problem, since one could potentially use the ellipsoid method to solve $\mathcal{LP}$ if one could obtain an efficient separation oracle. Thus intuitively, if one had an online algorithm which would generate S_- *on the fly*, then one could potentially get a feasible solution to $\mathcal{LP}$ in polynomial time.

More concretely, our densifier $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}$ will invoke a computationally efficient *online* learning algorithm for LTFs. In particular, $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}$ will run the online learner $\mathcal{A}_{\mathrm{online}}^{\mathbf{LTF}}$ for a sequence of stages and in each stage it will provide as counterexamples to $\mathcal{A}_{\mathrm{online}}^{\mathbf{LTF}}$, random labeled examples from a carefully chosen distribution. These examples will be positive for the online learner's current hypothesis, but negative for f (with high probability). Since $\mathcal{A}_{\mathrm{online}}^{\mathbf{LTF}}$ makes a small number of mistakes in the worst-case, this process is guaranteed to terminate after a small number of stages (since in each stage we *force* the online learner to make a mistake).

In Appendix D.2 we formalize this intuitive discussion by giving a precise description of the algorithm $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}$ and proving the following theorem, which directly gives Theorem 11:

Theorem 13. *Algorithm $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$ runs in time $\mathrm{poly}\,(n, 1/\epsilon, \log(1/\delta))$. If $p \leq \widehat{p} < (1+\epsilon)p$ then with probability $1-\delta$ it outputs a vector (w, θ) such that $g(x) = \mathrm{sign}(w \cdot x - \theta)$ satisfies conditions (a) and (b).*

5 DNFs

In this section we apply Theorem 8 to give a quasipolynomial-time distribution learning algorithm for s-term DNF formulas. Let $\mathbf{DNF}_{n,s}$ denote the class of all s-term DNF formulas over n Boolean variables (which for convenience we think of as $0/1$ variables). Our main result of this section is:

Theorem 14. *There is an algorithm $\mathcal{A}^{\mathbf{DNF}_{n,s}}$ which is a distribution learning algorithm for the class $\mathbf{DNF}_{n,s}$. Given input parameters ϵ, δ the algorithm runs in time $\mathrm{poly}\left(n^{\log(s/\epsilon)}, \log(1/\delta)\right)$.*

Even in the standard uniform distribution learning model the fastest known running time for learning s-term DNF formulas to accuracy ϵ is $\mathrm{poly}(n^{\log(s/\epsilon)}, \log(1/\delta))$ [Ver90, Val12]. Thus it seems likely that obtaining a $\mathrm{poly}(n, s, 1/\epsilon)$-time algorithm would require a significant breakthrough in learning theory.

For our application of Theorem 8 for DNFs we shall have $\mathcal{C} = \mathbf{DNF}_{n,s}$ and $\mathcal{C}' = \mathbf{DNF}_{n,t}$ for some t which we shall specify later. As detailed in Appendix E, the literature straightforwardly provides us with two of the three ingredients that our general approach requires for DNF, namely approximate uniform generation and approximate counting. As we explain below, though, some work is required for the Statistical Query portion of our approach, and we give an entirely new algorithm for the densifier. In the rest of this section we sketch the SQ algorithm and densifier construction and show how these ingredients are combined to give Theorem 14; full details are provided in Appendix E.

Statistical Query learning. The fastest known algorithm in the literature for SQ learning s-term DNF formulas under arbitrary distributions runs in time $n^{O(n^{1/3}\log s)} \cdot \mathrm{poly}(1/\epsilon)$ [KS04], which is much more than our desired running time bound. However, we show that we are able to use known *malicious noise tolerant* SQ learning algorithms for learning *sparse disjunctions* over N Boolean variables rather than DNF formulas. In more detail, our densifier will provide us with a set of $N = n^{O(\log(s/\epsilon))}$ many conjunctions which is such that the target function f is very close to a disjunction (which we call f') over an unknown subset of at most s of these N conjunctions. Thus intuitively any learning algorithm for disjunctions, run over the "feature space" of conjunctions provided by the densifier, would succeed if the target function were f', but the target function is actually f (which is not necessarily exactly a disjunction over these N variables). Fortunately, known results on the malicious noise tolerance of specific SQ learning algorithms imply that it is in fact possible to use these SQ algorithms to learn f to high accuracy, as we explain below.

The precise SQ learning result that we will use is the following theorem, which is a direct consequence of, e.g., Theorems 5 and 6 of [Dec93] or alteratively of Theorems 5 and 6 of [AD98]:

Theorem 15. *(Malicious noise tolerant* SQ *algorithm for learning sparse disjunctions) Let $\mathcal{C}_{\mathbf{DISJ},k}$ be the class of all disjunctions of length at most k over N Boolean variables $x_1, \ldots, x_N$. There is a distribution-independent* SQ *learning algorithm $\mathcal{A}_{\mathrm{SQ}}^{\mathbf{DISJ}}$ for $\mathcal{C}_{\mathbf{DISJ},k}$ that has running time $t_1 = \mathrm{poly}(N, 1/\epsilon, \log(1/\delta))$, uses at most $t_2 = \mathrm{poly}(N)$ time to evaluate each query, and requires tolerance of its queries no smaller than $\tau = 1/\mathrm{poly}(k, 1/\epsilon)$. The algorithm outputs a hypothesis which is a disjunction over $x_1, \ldots, x_N$.*

Moreover, there is a fixed polynomial $\ell(\cdot)$ such that algorithm $\mathcal{A}_{\mathrm{SQ}}^{\mathbf{DISJ}}$ has the following property: Fix a distribution D over $\{0,1\}^N$. Let f be an N-variable Boolean function which is such that $\mathbf{Pr}_{x\sim D}[f'(x) \neq f(x)] \leq \kappa$, where $f' \in \mathcal{C}_{\mathbf{DISJ},k}$ is some k-variable disjunction and $\kappa \leq \ell(\epsilon/k) < \epsilon/2$. Then if $\mathcal{A}_{\mathrm{SQ}}^{\mathbf{DISJ}}$ is run with a $\mathrm{STAT}(f, D)$ oracle, with probability $1-\delta$ it outputs a hypothesis h such that $\mathbf{Pr}_{x\sim D}[h(x) \neq f'(x)] \leq \epsilon/2$, and hence $\mathbf{Pr}_{x\sim D}[h(x \neq f(x)] \leq \epsilon$.

A densifier for $\mathbf{DNF}_{n,s}$. Our main theorem giving a densifier for DNF formulas is the following:

Theorem 16. *Let $\gamma(n, s, 1/\epsilon, 1/\delta) = 1/(4n^{2\log(2s/\ell(\epsilon/s))}\log(s/\delta))$. Algorithm $\mathcal{A}_{\mathrm{den}}^{\mathbf{DNF}_{n,s}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$ outputs a collection $\mathcal{S}$ of conjunctions $C_1, \ldots, C_{|\mathcal{S}|}$ and has the following performance guarantee: If $p \stackrel{def}{=} \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1] \leq \widehat{p} < (1+\epsilon)p$, then with probability at least $1-\delta$, the function $g(x) \stackrel{def}{=} \vee_{i\in[|\mathcal{S}|]} C_i$ satisfies the following:*

1. $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq 1-\epsilon$;
2. $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x) = 1] \geq \gamma(n, s, 1/\epsilon, 1/\delta)$;
3. *There is a DNF $f' = C_{i_1} \vee \cdots \vee C_{i_{s'}}$, which is a disjunction of $s' \leq s$ of the conjunctions $C_1, \ldots, C_{|\mathcal{S}|}$, such that $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f'(x) \neq f(x)] \leq \ell(\epsilon/s)$, where $\ell(\cdot)$ is the polynomial from Theorem 15.*

The size of $\mathcal{S}$ and the running time of $\mathcal{A}_{\mathrm{den}}^{\mathbf{DNF}_{n,s}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$ is $\mathrm{poly}(n^{\log(s/\epsilon)}, \log(1/\delta))$.

With a slight abuse of terminology we may rephrase the above theorem as saying that $\mathcal{A}_{\mathrm{den}}^{\mathbf{DNF}_{n,s}}$ is a $(\epsilon, \gamma, \delta)$-densifier for function class $\mathcal{C} = \mathbf{DNF}_{n,s}$ using class $\mathcal{C}' = \mathbf{DNF}_{n,t}$ where $t = n^{O(\log(s/\epsilon))}$.

Proof sketch of Theorem 16: Let $f = T_1 \vee \cdots \vee T_s$ be the target s-term DNF formula, where $T_1, \ldots, T_s$ are the terms (conjunctions). The high-level idea of our densifier is quite simple: If T_i is a term which is "reasonably likely" to be satisfied by a uniform draw of x from $f^{-1}(1)$, then T_i is at least "mildly likely" to be satisfied by $r = 2\log n$ consecutive independent draws of x from $f^{-1}(1)$. Such a sequence of draws $x^1, \ldots, x^r$ will with high probability *uniquely identify* T_i. By repeating this process sufficiently many times, with high probability we will obtain a pool $C_1, \ldots, C_{|\mathcal{S}|}$ of conjunctions which contains all of the terms T_i that are reasonably likely to be satisfied by a uniform draw of x from $f^{-1}(1)$. Theorem 16 follows straightforwardly from this. We give detailed pseudocode for our densifier algorithm, and a full proof of Theorem 16, in Appendix E. □

We conclude this section by showing how Theorem 14 follows from the SQ algorithm and densifier described above.

Proof of Theorem 14. The proof is essentially just an application of Theorem 8. The only twist is the use of a SQ disjunction learning algorithm rather than a DNF learning algorithm, but the special properties of Algorithm $\mathcal{A}_{\mathrm{SQ}}^{\mathbf{DISJ}}$ let this go through without a problem.

In more detail, in Step 2(e) of Algorithm $\mathcal{A}'^{\mathcal{C}}$ (see Section C.2), in the execution of Algorithm $\mathcal{A}_{\mathrm{SQ-SIM}}$, the SQ algorithm that is simulated is the algorithm $\mathcal{A}_{\mathrm{SQ}}^{\mathbf{DISJ}}$ run over the feature space $\mathcal{S}$ of all conjunctions that are output by Algorithm $\mathcal{A}_{\mathrm{den}}^{\mathbf{DNF}_{n,s}}$ in Step 1 of Algorithm $\mathcal{A}'^{\mathcal{C}}$ (i.e., these conjunctions play the role of variables $x_1, \ldots, x_N$ for the SQ learning algorithm). Property (3) of Theorem 16 and Theorem 15 together imply that the algorithm $\mathcal{A}_{\mathrm{SQ}}^{\mathbf{DISJ}}$, run on a $\mathrm{STAT}(f, \mathcal{U}_{g^{-1}(1)})$ oracle with parameters ϵ, δ, would

with probability $1-\delta$ output a hypothesis h' satisfying $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[h'(x) \neq f(x)] \leq \epsilon$. Hence the h that is output by $\mathcal{A}_{\mathrm{SQ-SIM}}$ in Step 2(e) of Algorithm $\mathcal{A}'^{\mathcal{C}}$ fulfills the accuracy (with respect to f under $D = \mathcal{U}_{g^{-1}(1)}$) and confidence requirements, and the overall algorithm $A^{\mathcal{C}}$ succeeds as claimed in Theorem 8.

Finally, combining the running time bounds of $\mathcal{A}_{\mathrm{den}}^{\mathbf{DNF}_{n,s}}$ and $\mathcal{A}_{\mathrm{SQ}}^{\mathbf{DISJ}}$ with the time bounds of the other procedures described earlier, one can straightforwardly verify that the running time of the overall algorithm $\mathcal{A}^{\mathcal{C}}$ is $\mathrm{poly}(n^{\log(s/\epsilon)}, \log(1/\delta))$. □

Acknowledgements. We thank Alistair Sinclair, Luca Trevisan, Jonathan Katz, Tal Malkin, Krzysztof Pietrzak, Salil Vadhan, Kousha Etessami, Elchanan Mossel, Li-Yang Tan, Thomas Watson and Avi Wigderson for helpful conversations.

References

[AD98] J. Aslam and S. Decatur. Specification and simulation of statistical query algorithms for efficiency and noise tolerance. *Journal of Computer and System Sciences*, 56:191–208, 1998.

[AGR13] J. Anderson, N. Goyal, and L. Rademacher. Efficient Learning of Simplices. In *Conference on Learning Theory*, pages 1020–1045, 2013.

[BFKV97] A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial time algorithm for learning noisy linear threshold functions. *Algorithmica*, 22(1/2):35–52, 1997.

[BS10] Mikhail Belkin and Kaushik Sinha. Polynomial learning of distribution families. In *Proc. 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 103–112, 2010.

[Das99] S. Dasgupta. Learning mixtures of gaussians. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 634–644, 1999.

[DDS12a] C. Daskalakis, I. Diakonikolas, and R.A. Servedio. Learning k-modal distributions via testing. In *SODA*, pages 1371–1385, 2012.

[DDS12b] C. Daskalakis, I. Diakonikolas, and R.A. Servedio. Learning poisson binomial distributions. In *STOC*, pages 709–728, 2012.

[Dec93] S. Decatur. Statistical queries and faulty PAC oracles. In *Proceedings of the Sixth Workshop on Computational Learning Theory*, pages 262–268, 1993.

[Den98] F. Denis. PAC Learning from Positive Statistical Queries. pages 132–145, 1998.

[DFK91] M. Dyer, A. Frieze, and R. Kannan. A Random Polynomial Time Algorithm for Approximating the Volume of Convex Bodies. *J. ACM*, 38(1):1–17, 1991.

[DGL05] F. Denis, R. Gilleron, and F. Letouzey. Learning from positive and unlabeled examples. *Theoretical Computer Science*, 348:70–83, 2005.

[DL01] L. Devroye and G. Lugosi. *Combinatorial methods in density estimation*. Springer Series in Statistics, Springer, 2001.

[Dye03] M. Dyer. Approximate counting by dynamic programming. In *STOC*, pages 693–699, 2003.

[Gol04] Oded Goldreich. *Foundations of Cryptography-volume 2*. Cambridge University Press, Cambridge, 2004.

[HW10] S. Hohenberger and B. Waters. Constructing Verifiable Random Functions with Large Input Spaces. In *EUROCRYPT*, pages 656–672, 2010.

[JVV86] M. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[Kea98] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998.

[KL83] R.M. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. In *FOCS*, pages 56–64, 1983.

[KLM89] R. M. Karp, M. Luby, and N. Madras. Monte-Carlo Approximation Algorithms for Enumeration Problems. *Journal of Algorithms*, 10(3):429–448, 1989.

[KMR$^+$94] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. On the learnability of discrete distributions. In *Proceedings of the 26th Symposium on Theory of Computing*, pages 273–282, 1994.

[KS04] A. Klivans and R. Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. *Journal of Computer & System Sciences*, 68(2):303–318, 2004.

[KSV08] Ravindran Kannan, Hadi Salmasian, and Santosh Vempala. The spectral method for general mixture models. *SIAM J. Comput.*, 38(3):1141–1156, 2008.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *Advances in Cryptology — (CRYPTO 2002)*, volume 2442 of *Lecture Notes in Computer Science*, pages 597–612. Springer-Verlag, 2002.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable Random Functions. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 120–130, 1999.

[MT94] W. Maass and G. Turan. How fast can a threshold gate learn? In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems*, pages 381–414. MIT Press, 1994.

[Mur71] S. Muroga. *Threshold logic and its applications*. Wiley-Interscience, New York, 1971.

[MV10] Ankur Moitra and Gregory Valiant. Settling the polynomial learnability of mixtures of gaussians. In *Proc. 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 93–102, 2010.

[Nao96] M. Naor. Evaluation may be easier than generation. In *Proceedings of the 28th Symposium on Theory of Computing (STOC)*, pages 74–83, 1996.

[Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Val12] G. Valiant. Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas. In *FOCS*, 2012.

[Ver90] Karsten A. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time. In Mark A. Fulk, editor, *Conference on Learning Theory*, pages 314–326. Morgan Kaufmann, 1990.

[Wat12] Thomas Watson. The complexity of estimating Min-entropy. Technical Report 70, Electronic Colloquium in Computational Complexity, 2012.

A Preliminaries

For completeness we give precise definitions of approximate counting and approximate uniform generation algorithms for a class of Boolean functions as well as some related notions that we require.

Definition 17 (approximate counting)**.** *Let $\mathcal{C}$ be a class of n-variable Boolean functions. A randomized algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{count}}$ is an* efficient approximate counting algorithm for class $\mathcal{C}$*, if for any $\epsilon, \delta > 0$ and any $f \in \mathcal{C}$, on input ϵ, δ and $f \in \mathcal{C}$, it runs in time $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ and with probability $1-\delta$ outputs a value $\widehat{p}$ such that*

$$\frac{1}{(1+\epsilon)} \cdot \mathbf{Pr}_{x\sim\mathcal{U}}[f(x)=1] \leq \widehat{p} \leq (1+\epsilon) \cdot \mathbf{Pr}_{x\sim\mathcal{U}}[f(x)=1].$$

Definition 18 (approximate uniform generation)**.** *Let $\mathcal{C}$ be a class of n-variable Boolean functions. A randomized algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{gen}}$ is an* efficient approximate uniform generation algorithm for class $\mathcal{C}$*, if for any $\epsilon > 0$ and any $f \in \mathcal{C}$, there is a distribution $D = D_{f,\epsilon}$ supported on $f^{-1}(1)$ with*

$$\frac{1}{1+\epsilon} \cdot \frac{1}{|f^{-1}(1)|} \leq D(x) \leq (1+\epsilon) \cdot \frac{1}{|f^{-1}(1)|}$$

for each $x \in f^{-1}(1)$, such that for any $\delta > 0$, on input ϵ, δ and $f \in \mathcal{C}$, algorithm $A^{\mathcal{C}}_{\mathrm{gen}}(\epsilon, \delta, f)$ runs in time $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ and either outputs a point $x \in f^{-1}(1)$ that is distributed precisely according to $D = D_{f,\epsilon}$, or outputs $\perp$. Moreover the probability that it outputs $\perp$ is at most δ.

We will also need the notion of a *Statistical Query* learning algorithm for a class $\mathcal{C}$ of Boolean functions.

Definition 19. *Let $\mathcal{C}$ be a class of n-variable Boolean functions and D be a distribution over $\{-1,1\}^n$. An* SQ learning algorithm for $\mathcal{C}$ under D *is a randomized algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ that for every $\epsilon, \delta > 0$, every target concept $f \in \mathcal{C}$, on input ϵ, δ and with access to oracle $\mathrm{STAT}(f, D)$ and to independent samples drawn from D, outputs with probability $1-\delta$ a hypothesis $h : \{-1,1\}^n \to \{-1,1\}$ such that $\mathbf{Pr}_{x\sim D}[h(x) \neq f(x)] \leq \epsilon$. Let $t_1(n, 1/\epsilon, 1/\delta)$ be the running time of $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ (assuming each oracle query is answered in unit time), $t_2(n)$ be the maximum running time to evaluate any query provided to $\mathrm{STAT}(f, D)$ and $\tau(n, 1/\epsilon)$ be the minimum value of the tolerance parameter ever provided to $\mathrm{STAT}(f, D)$ in the course of $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$'s execution. We say that $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ is* efficient *(and that $\mathcal{C}$ is* efficiently SQ *learnable with respect to distribution D), if $t_1(n, 1/\epsilon, 1/\delta)$ is polynomial in n, $1/\epsilon$ and $1/\delta$, $t_2(n)$ is polynomial in n and $\tau(n, 1/\epsilon)$ is lower bounded by an inverse polynomial in n and $1/\epsilon$. We call an* SQ *learning algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ for $\mathcal{C}$* distribution independent *if $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ succeeds for any distribution D. If $\mathcal{C}$ has an efficient distribution independent* SQ *learning algorithm we say that $\mathcal{C}$ is* efficiently SQ learnable (distribution independently).

Before we formally define our learning model, we need the notion of a *sampler* for a distribution:

Definition 20. *Let D be a distribution over $\{-1,1\}^n$. A* sampler for D *is a circuit C with $m = \mathrm{poly}(n)$ input bits $z \in \{-1,1\}^m$ and n output bits $x \in \{-1,1\}^n$ which is such that when $z \sim \mathcal{U}_m$ then $x \sim D$. For $\epsilon > 0$, an* ϵ-sampler for D *is a sampler for some distribution D' which has $d_{\mathrm{TV}}(D', D) \leq \epsilon$.*

For clarity we sometimes write "C is a 0-sampler for D" to emphasize the fact that the outputs of $C(z)$ are distributed *exactly* according to distribution D.

We are now ready to formally define the notion of a distribution learning algorithm in our model.

Definition 21. *Let $\mathcal{C}$ be a class of n-variable Boolean functions. A randomized algorithm $\mathcal{A}^{\mathcal{C}}$ is a* distribution learning algorithm for class $\mathcal{C}$*, if for any $\epsilon, \delta > 0$ and any $f \in \mathcal{C}$, on input ϵ, δ and sample access to $\mathcal{U}_{f^{-1}(1)}$, with probability $1-\delta$ algorithm $A^{\mathcal{C}}$ outputs an ϵ-sampler C_f for $\mathcal{U}_{f^{-1}(1)}$.*

B Hypothesis testing: Proof of Proposition 6

Discussion. We note that there are certain significant differences between the current setting and the setting of [DDS12a, DDS12b] (as well as other related works that use versions of Proposition 6). In particular, in our setting, the set $\mathcal{W}$ is of size 2^n, which was not the case in [DDS12a, DDS12b]. Hence, we cannot assume the distributions D_i are given explicitly in the input. Thus Proposition 6 carefully specifies what kind of access to these distributions is required. Proposition 6 is an extension of similar results in the previous works; while the idea of the proof is essentially the same, the details are more involved.

Proof of Proposition 6. At a high level, the algorithm $\mathcal{T}^D$ performs a tournament by running a "competition" $\texttt{Choose-Hypothesis}^D$ for every pair of distinct distributions in the collection $\mathcal{D}_\epsilon$. It outputs a distribution $D^\star \in \mathcal{D}_\epsilon$ that was never a loser (i.e., won or achieved a draw in all its competitions). If no such distribution exists in $\mathcal{D}_\epsilon$ then the algorithm outputs "failure." We start by describing and analyzing the competition subroutine between a pair of distributions in the collection.

Lemma 22. *In the context of Proposition 6, there is an algorithm* $\texttt{Choose-Hypothesis}^D(D_i, D_j, \epsilon', \delta')$ *which is given access to*

(i) independent samples from D and D_k, for $k \in \{i, j\}$,

(ii) an evaluation oracle $\mathrm{EVAL}_{D_k}(\beta)$*, for $k \in \{i, j\}$,*

an accuracy parameter ϵ' and a confidence parameter δ', and has the following behavior: It uses $m' = O\left((1/{\epsilon'}^2)\log(1/\delta')\right)$ samples from each of D, D_i and D_j, it makes $O(m')$ calls to the oracles $\mathrm{EVAL}_{D_k}(\beta)$*, $k \in \{i, j\}$, performs $O(m')$ arithmetic operations, and if some D_k, $k \in \{i, j\}$, has $d_{\mathrm{TV}}(D_k, D) \le \epsilon'$ then with probability $1 - \delta'$ it outputs an index $k^\star \in \{i, j\}$ that satisfies $d_{\mathrm{TV}}(D, D_{k^\star}) \le 6\epsilon'$.*

Proof. To set up the competition between D_i and D_j, we consider the following subset of $\mathcal{W}$:

$$H_{ij} = H_{ij}(D_i, D_j) \stackrel{\text{def}}{=} \{w \in \mathcal{W} \mid D_i(w) \ge D_j(w)\}$$

and the corresponding probabilities $p_{i,j} \stackrel{\text{def}}{=} D_i(H_{ij})$ and $q_{i,j} \stackrel{\text{def}}{=} D_j(H_{ij})$. Clearly, it holds $p_{i,j} \ge q_{i,j}$ and by definition of the total variation distance we can write $d_{\mathrm{TV}}(D_i, D_j) = p_{i,j} - q_{i,j}$.

For the purposes of our algorithm, we would ideally want oracle access to the set H_{ij}. Unfortunately though, this is not possible since the evaluation oracles are only approximate. Hence, we will need to define a more robust version of the set H_{ij} which will turn out to have similar properties. In particular, we consider the set

$$H_{ij}^\beta \stackrel{\text{def}}{=} \{w \in \mathcal{W} \mid \widetilde{D}_i^\beta(w) \ge \widetilde{D}_j^\beta(w)\}$$

and the corresponding probabilities $p_{i,j}^\beta \stackrel{\text{def}}{=} D_i(H_{ij}^\beta)$ and $q_{i,j}^\beta \stackrel{\text{def}}{=} D_j(H_{ij}^\beta)$. We claim that the difference $\Delta \stackrel{\text{def}}{=} p_{i,j}^\beta - q_{i,j}^\beta$ is an accurate approximation to $d_{\mathrm{TV}}(D_i, D_j)$. In particular, we show:

Claim 23. *We have*

$$\Delta \le d_{\mathrm{TV}}(D_i, D_j) \le \Delta + \epsilon/4. \tag{1}$$

Before we proceed with the proof, we stress that (1) crucially uses our assumption that the evaluation oracles provide a *multiplicative* approximation to the exact probabilities.

Proof. To show (1) we proceed as follows: Let $A = H_{ij} \cap H_{ij}^{\beta}$, $B = H_{ij} \cap \overline{H_{ij}^{\beta}}$ and $C = \overline{H_{ij}} \cap H_{ij}^{\beta}$. Then we can write $d_{\mathrm{TV}}(D_i, D_j) = (D_i - D_j)(A) + (D_i - D_j)(B)$ and $\Delta = (D_i - D_j)(A) + (D_i - D_j)(C)$. We will show that

$$0 \leq (D_i - D_j)(B) \leq \epsilon/8 \tag{2}$$

and similarly

$$-\epsilon/8 \leq (D_i - D_j)(C) \leq 0 \tag{3}$$

from which the claim follows. We proceed to prove (2), the proof of (3) being very similar. Let $w \in B$. Then $D_i(w) \geq D_j(w)$ (since $w \in H_{ij}$) which gives $(D_i - D_j)(B) \geq 0$, establishing the LHS of (2). We now establish the RHS. For $w \in B$ we also have that $\widetilde{D}_i^{\beta}(w) < \widetilde{D}_j^{\beta}(w)$ (since $w \in \overline{H_{ij}^{\beta}}$). Now by the definition of the evaluation oracles, it follows that $\widetilde{D}_i^{\beta}(w) \geq \frac{D_i(w)}{(1+\beta)}$ and $\widetilde{D}_j^{\beta}(w) \leq (1+\beta)D_j(w)$. Combining these inequalities yields

$$D_i(w) \leq (1+\beta)^2 D_j(w) \leq (1+\epsilon/8)D_j(w)$$

where the second inequality follows by our choice of β. Therefore, $(D_i - D_j)(B) = \sum_{w \in B} (D_i(w) - D_j(w)) \leq (\epsilon/8) \cdot D_j(B) \leq \epsilon/8$ as desired. □

Note that the probabilities $p_{i,j}^{\beta}$ and $q_{i,j}^{\beta}$ are not available to us explicitly. Hence, `Choose-Hypothesis` requires a way to empirically estimate each of these probability values (up to a small additive accuracy). This task can be done efficiently because we have sample access to the distributions D_i, D_j and oracle access to the set H_{ij}^{β} thanks to the $\mathrm{EVAL}_{D_k}(\beta)$ oracles. The following claim provides the details:

Claim 24. *There exists a subroutine* `Estimate`$(D_i, H_{ij}^{\beta}, \gamma, \delta)$ *which is given access to*

(i) independent samples from D_i,

(ii) an evaluation oracle $\mathrm{EVAL}_{D_k}(\beta)$, for $k \in \{i, j\}$,

an accuracy parameter γ and a confidence parameter δ, and has the following behavior: It makes $m = O\left((1/\gamma^2)\log(1/\delta)\right)$ draws from D_i and $O(m)$ calls to the oracles $\mathrm{EVAL}_{D_k}(\beta)$, $k = i, j$, performs $O(m)$ arithmetic operations, and with probability $1 - \delta$ outputs a number $\widetilde{p}_{i,j}^{\beta}$ such that $|\widetilde{p}_{i,j}^{\beta} - p_{i,j}^{\beta}| \leq \gamma$.

Proof. The desired subroutine amounts to a straightforward random sampling procedure, which we include here for the sake of completeness. We will use the following elementary fact, a simple consequence of the additive Chernoff bound.

Fact 25. *Let X be a random variable taking values in the range $[-1, 1]$. Then $\mathbf{E}[X]$ can be estimated to within an additive $\pm\tau$, with confidence probability $1 - \delta$, using $m = \Omega((1/\tau^2)\log(1/\delta))$ independent samples from X. In particular, the empirical average $\widehat{X}_m = (1/m)\sum_{i=1}^{m} X_i$, where the X_i's are independent samples of X, satisfies $\mathbf{Pr}\left[|\widehat{X}_m - \mathbf{E}[X]| \leq \tau\right] \geq 1 - \delta$.*

We shall refer to this as "empirically estimating" the value of $\mathbf{E}[X]$.

Consider the indicator function $I_{H_{ij}^{\beta}}$ of the set H_{ij}^{β}, i.e., $I_{H_{ij}^{\beta}} : \mathcal{W} \to \{0, 1\}$ with $I_{H_{ij}^{\beta}}(x) = 1$ if and only if $x \in H_{ij}^{\beta}$. It is clear that $\mathbf{E}_{x \sim D_i}\left[I_{H_{ij}^{\beta}}(x)\right] = D_i(H_{ij}^{\beta}) = p_{i,j}^{\beta}$. The subroutine is described in the following pseudocode:

Subroutine $\texttt{Estimate}(D_i, H_{ij}^{\beta}, \gamma, \delta)$:

Input: Sample access to D_i and oracle access to $\mathrm{EVAL}_{D_k}(\beta)$, $k = i, j$.

Output: A number $\widetilde{p}_{ij}^{\beta}$ such that with probability $1 - \delta$ it holds $|\widetilde{p}_{ij}^{\beta} - D_i(H_{ij}^{\beta})| \leq \gamma$.

1. Draw $m = \Theta\left((1/\gamma^2)\log(1/\delta)\right)$ samples $\mathbf{s} = \{s_\ell\}_{\ell=1}^{m}$ from D_i.
2. For each sample s_ℓ, $\ell \in [m]$:
   (a) Use the oracles $\mathrm{EVAL}_{D_i}(\beta)$, $\mathrm{EVAL}_{D_j}(\beta)$, to approximately evaluate $D_i(s_\ell)$, $D_j(s_\ell)$.
   (b) If $\widetilde{D}_i^{\beta}(s_\ell) \geq \widetilde{D}_j^{\beta}(s_\ell)$ set $I_{H_{ij}^{\beta}}(s_\ell) = 1$, otherwise $I_{H_{ij}^{\beta}}(s_\ell) = 0$.
3. Set $\widetilde{p}_{ij}^{\beta} = \frac{1}{m}\sum_{\ell=1}^{m} I_{H_{ij}^{\beta}}(s_\ell)$.
4. Output $\widetilde{p}^{\beta}{}_{ij}$.

The computational efficiency of this simple random sampling procedure follows from the fact that we can efficiently decide membership in H_{ij}^{β}. To do this, for a given $x \in \mathcal{W}$, we make a query to each of the oracles $\mathrm{EVAL}_{D_i}(\beta)$, $\mathrm{EVAL}_{D_j}(\beta)$ to obtain the probabilities $\widetilde{D}_i^{\beta}(x)$, $\widetilde{D}_j^{\beta}(x)$. We have that $x \in H_{ij}^{\beta}$ (or equivalently $I_{H_{ij}^{\beta}}(x) = 1$) if and only if $\widetilde{D}_i^{\beta}(x) \geq \widetilde{D}_j^{\beta}(x)$. By Fact 25, applied for the random variable $I_{H_{ij}^{\beta}}(x)$, where $x \sim D_i$, after $m = \Omega((1/\gamma^2)\log(1/\delta))$ samples from D_i we obtain a $\pm\gamma$-additive estimate to $p_{i,j}^{\beta}$ with probability $1 - \delta$. For each sample, we make one query to each of the oracles, hence the total number of oracle queries is $O(m)$ as desired. The only non-trivial arithmetic operations are the $O(m)$ comparisons done in Step 2(b), and Claim 24 is proved. □

Now we are ready to prove Lemma 22. The algorithm $\texttt{Choose-Hypothesis}^{D}(D_i, D_j, \epsilon', \delta')$ performing the competition between D_i and D_j is the following:

Algorithm $\texttt{Choose-Hypothesis}^{D}(D_i, D_j, \epsilon', \delta')$:

Input: Sample access to D and D_k, $k = i, j$, oracle access to $\mathrm{EVAL}_{D_k}(\beta)$, $k = i, j$.

1. Set $\widetilde{p}_{i,j}^{\beta} = \texttt{Estimate}(D_i, H_{ij}^{\beta}, \epsilon'/8, \delta'/4)$ and $\widetilde{q}_{i,j}^{\beta} = \texttt{Estimate}(D_j, H_{ij}^{\beta}, \epsilon'/8, \delta'/4)$.
2. If $\widetilde{p}_{i,j}^{\beta} - \widetilde{q}_{i,j}^{\beta} \leq 9\epsilon'/2$, declare a draw and return either i or j. Otherwise:
3. Draw $m' = \Theta\left((1/\epsilon'^2)\log(1/\delta')\right)$ samples $\mathbf{s}' = \{s_\ell\}_{\ell=1}^{m'}$ from D.
4. For each sample s_ℓ, $\ell \in [m']$:
   (a) Use the oracles $\mathrm{EVAL}_{D_i}(\beta)$, $\mathrm{EVAL}_{D_j}(\beta)$ to evaluate $\widetilde{D}_i^{\beta}(s_\ell)$, $\widetilde{D}_j^{\beta}(s_\ell)$.
   (b) If $\widetilde{D}_i^{\beta}(s_\ell) \geq \widetilde{D}_j^{\beta}(s_\ell)$ set $I_{H_{ij}^{\beta}}(s_\ell) = 1$, otherwise $I_{H_{ij}^{\beta}}(s_\ell) = 0$.
5. Set $\tau = \frac{1}{m'}\sum_{\ell=1}^{m'} I_{H_{ij}^{\beta}}(s_\ell)$, i.e., τ is the fraction of samples that fall inside H_{ij}^{β}.
6. If $\tau > \widetilde{p}_{i,j}^{\beta} - \frac{13}{8}\epsilon'$, declare D_i as winner and return i; otherwise,
7. if $\tau < \widetilde{q}_{i,j}^{\beta} + \frac{13}{8}\epsilon'$, declare D_j as winner and return j; otherwise,
8. declare a draw and return either i or j.

It is not hard to check that the outcome of the competition does not depend on the ordering of the pair of distributions provided in the input; that is, on inputs (D_i, D_j) and (D_j, D_i) the competition outputs the same result for a fixed set of samples $\{s_1, \ldots, s_{m'}\}$ drawn from D.

The upper bounds on sample complexity, query complexity and number of arithmetic operations can be straightforwardly verified. Hence, it remains to show correctness. By Claim 24 and a union bound, with probability at least $1 - \delta'/2$, we will have that $|\widetilde{p}^{\beta}_{i,j} - p^{\beta}_{i,j}| \leq \epsilon'/8$ and $|\widetilde{q}^{\beta}_{i,j} - q^{\beta}_{i,j}| \leq \epsilon'/8$. In the following, we condition on this good event. The correctness of `Choose-Hypothesis` is then an immediate consequence of the following claim.

Claim 26. *Suppose that $d_{\mathrm{TV}}(D, D_i) \leq \epsilon'$. Then:*

(i) If $d_{\mathrm{TV}}(D, D_j) > 6\epsilon'$, then the probability that the competition between D_i and D_j does not declare D_i as the winner is at most $e^{-m'\epsilon'^2/8}$. (Intuitively, if D_j is very far from D then it is very likely that D_i will be declared winner.)

(ii) The probability that the competition between D_i and D_j declares D_j as the winner is at most $e^{-m'\epsilon'^2/8}$. (Intuitively, since D_i is close to D, a draw with some other D_j is possible, but it is very unlikely that D_j will be declared winner.)

Proof. Let $r^{\beta} = D(H^{\beta}_{ij})$. The definition of the variation distance implies that $|r^{\beta} - p^{\beta}_{i,j}| \leq d_{\mathrm{TV}}(D, D_i) \leq \epsilon'$. Therefore, we have that $|r^{\beta} - \widetilde{p}^{\beta}_{i,j}| \leq |r^{\beta} - p^{\beta}_{i,j}| + |\widetilde{p}^{\beta}_{i,j} - p^{\beta}_{i,j}| \leq 9\epsilon'/8$. Consider the indicator $(0/1)$ random variables $\{Z_\ell\}_{\ell=1}^{m'}$ defined as $Z_\ell = 1$ if and only if $s_\ell \in H^{\beta}_{ij}$. Clearly, $\tau = \frac{1}{m'}\sum_{\ell=1}^{m'} Z_\ell$ and $\mathbf{E}_{\mathbf{s}'}[\tau] = \mathbf{E}_{s_\ell \sim D}[Z_\ell] = r^{\beta}$. Since the Z_ℓ's are mutually independent, it follows from the Chernoff bound that $\mathbf{Pr}[\tau \leq r^{\beta} - \epsilon'/2] \leq e^{-m'\epsilon'^2/8}$. Using $|r^{\beta} - \widetilde{p}^{\beta}_{i,j}| \leq 9\epsilon'/8$. we get that $\mathbf{Pr}[\tau \leq \widetilde{p}^{\beta}_{i,j} - 13\epsilon'/8] \leq e^{-m'\epsilon'^2/8}$.

- For part (i): If $d_{\mathrm{TV}}(D, D_j) > 6\epsilon'$, from the triangle inequality we get that $p_{i,j} - q_{i,j} = d_{\mathrm{TV}}(D_i, D_j) > 5\epsilon'$ Claim 23 implies that $p^{\beta}_{i,j} - q^{\beta}_{i,j} > 19\epsilon'/4$ and our conditioning finally gives $\widetilde{p}^{\beta}_{i,j} - \widetilde{q}^{\beta}_{i,j} > 9\epsilon'/2$. Hence, the algorithm will go beyond Step 2, and with probability at least $1 - e^{-m'\epsilon'^2/8}$, it will stop at Step 6, declaring D_i as the winner of the competition between D_i and D_j.

- For part (ii): If $\widetilde{p}^{\beta}_{i,j} - \widetilde{q}^{\beta}_{i,j} \leq 9\epsilon'/2$ then the competition declares a draw, hence D_j is not the winner. Otherwise we have $\widetilde{p}^{\beta}_{i,j} - \widetilde{q}^{\beta}_{i,j} > 9\epsilon'/2$ and the argument of the previous paragraph implies that the competition between D_i and D_j will declare D_j as the winner with probability at most $e^{-m'\epsilon'^2/8}$.

This concludes the proof of Claim 26. □

This completes the proof of Lemma 22. □

We now proceed to describe the algorithm $\mathcal{T}^D$ and establish Proposition 6. The algorithm performs a tournament by running the competition $\texttt{Choose-Hypothesis}^D(D_i, D_j, \epsilon, \delta/(2N))$ for every pair of distinct distributions D_i, D_j in the collection $\mathcal{D}_\epsilon$. It outputs a distribution $D^\star \in \mathcal{D}_\epsilon$ that was never a loser (i.e., won or achieved a draw in all its competitions). If no such distribution exists in $\mathcal{D}_\epsilon$ then the algorithm outputs "failure." A detailed pseudocode follows:

Algorithm $\mathcal{T}^D(\{D_j\}_{j=1}^N, \epsilon, \delta)$:

Input: Sample access to D and D_k, $k \in [N]$, and oracle access to EVAL_{D_k}, $k \in [N]$.

1. Draw $m = \Theta\left((1/\epsilon^2)(\log N + \log(1/\delta))\right)$ samples from D and each D_k, $k \in [N]$.
2. For all $i, j \in [N]$, $i \neq j$, run $\texttt{Choose-Hypothesis}^D(D_i, D_j, \epsilon, \delta/(2N))$ using this sample.
3. Output an index $i^\star$ such that $D_{i^\star}$ was never declared a loser, if one exists.
4. Otherwise, output "failure".

We now proceed to analyze the algorithm. The bounds on the sample complexity, running time and query complexity to the evaluation oracles follow from the corresponding bounds for `Choose-Hypothesis`. Hence, it suffices to show correctness. We do this below.

By definition, there exists some $D_i \in \mathcal{D}_\epsilon$ such that $d_{\mathrm{TV}}(D, D_i) \leq \epsilon$. By Claim 26, the distribution D_i never loses a competition against any other $D_j \in \mathcal{D}_\epsilon$ (so the algorithm does not output "failure"). A union bound over all N distributions in $\mathcal{D}_\epsilon$ shows that with probability $1 - \delta/2$, the distribution D' never loses a competition.

We next argue that with probability at least $1 - \delta/2$, every distribution $D_j \in \mathcal{D}_\epsilon$ that never loses has small variation distance from D. Fix a distribution D_j such that $d_{\mathrm{TV}}(D_j, D) > 6\epsilon$; Claim 26(i) implies that D_j loses to D_i with probability $1 - 2e^{-m\epsilon^2/8} \geq 1 - \delta/(2N)$. A union bound yields that with probability $1 - \delta/2$, every distribution D_j that has $d_{\mathrm{TV}}(D_j, D) > 6\epsilon$ loses some competition.

Thus, with overall probability at least $1 - \delta$, the tournament does not output "failure" and outputs some distribution $D^\star$ such that $d_{\mathrm{TV}}(D, D^\star)$ is at most 6ϵ. The proof of Proposition 6 is now complete. □

Remark 27. As stated Proposition 6 assumes that algorithm $\mathcal{T}^D$ has access to samplers for all the distributions D_k, so each call to such a sampler is guaranteed to output an element distributed according to D_k. Let $D_k^\perp$ be a distribution over $\mathcal{W} \cup \{\perp\}$ which is such that (i) $D_k^\perp(\perp) \leq 1/2$, and (ii) the conditional distribution $(D_k^\perp)_{\mathcal{W}}$ of $D_k^\perp$ conditioned on not outputting $\perp$ is precisely D_k. It is easy to see that the proof of Proposition 6 extends to a setting in which $\mathcal{T}^D$ has access to samplers for $D_k^\perp$ rather than samplers for D_k; each time a sample from D_k is required the algorithm can simply invoke the sampler for $D_k^\perp$ repeatedly until an element other than $\perp$ is obtained. (The low-probability event that many repetitions are ever needed can be "folded into" the failure probability δ.)

C General Algorithmic Approach: Proof of Theorem 8

C.1 Simulating statistical query algorithms. Our algorithm $\mathcal{A}^{\mathcal{C}}$ will need to simulate a statistical query algorithm for $\mathcal{C}$, with respect to a specific distribution D. Note, however that $\mathcal{A}$ only has access to uniform positive examples of $f \in \mathcal{C}$, i.e., samples from $\mathcal{U}_{f^{-1}(1)}$. Hence we need to show that a statistical query algorithm can be efficiently simulated in such a setting. To do this it suffices to show that one can efficiently provide valid responses to queries to the statistical query oracle $\mathrm{STAT}(f, D)$, i.e., that one can simulate the oracle. Assuming this can be done, the simulation algorithm $\mathcal{A}_{\mathrm{SQ-SIM}}$ is very simple: Run the statistical query algorithm $\mathcal{A}_{\mathrm{SQ}}$, and whenever it makes a query to $\mathrm{STAT}(f, D)$, simulate it. To this end, in the following lemma we describe a procedure that simulates an SQ oracle. (Our approach here is similar to that of earlier simulation procedures that have been given in the literature, see e.g. Denis *et al.* [DGL05].)

Lemma 28. *Let $\mathcal{C}$ be a concept class over $\{-1,1\}^n$, $f \in \mathcal{C}$, and D be a samplable distribution over $\{-1,1\}^n$. There exists an algorithm $\texttt{Simulate-STAT}_f^D$ with the following properties: It is given access to independent samples from $D_{f,+}$, and takes as input a number $\widetilde{b_f} \in [0,1]$, a $t(n)$-time computable query function $\chi : \{-1,1\}^n \times \{-1,1\} \to [-1,1]$, a tolerance τ and a confidence δ. It has the following behavior: it uses $m = O\left((1/\tau^2)\log(1/\delta)\right)$ samples from D and $D_{f,+}$, runs in time $O\left(m \cdot t(n)\right)$, and if $|\widetilde{b_f} - \mathbf{Pr}_{x\sim D}[f(x) = 1]| \leq \tau'$, then with probability $1 - \delta$ it outputs a number v such that*

$$|\mathbf{E}_{x\sim D}\left[\chi\left(x, f(x)\right)\right] - v| \leq \tau + \tau'. \tag{4}$$

Proof. To prove the lemma, we start by rewriting the expectation in (4) as follows:

$$\mathbf{E}_{x\sim D}\left[\chi(x, f(x))\right] = \mathbf{E}_{x\sim D_{f,+}}\left[\chi(x,1)\right] \cdot \mathbf{Pr}_{x\sim D}[f(x) = 1] + \mathbf{E}_{x\sim D_{f,-}}\left[\chi(x,-1)\right] \cdot \mathbf{Pr}_{x\sim D}[f(x) = -1].$$

We also observe that

$$\mathbf{E}_{x\sim D}\left[\chi(x,-1)\right] = \mathbf{E}_{x\sim D_{f,+}}\left[\chi(x,-1)\right]\cdot \mathbf{Pr}_{x\sim D}[f(x)=1] + \mathbf{E}_{x\sim D_{f,-}}\left[\chi(x,-1)\right]\cdot \mathbf{Pr}_{x\sim D}[f(x)=-1].$$

Combining the above equalities we get

$$\mathbf{E}_{x\sim D}\left[\chi(x,f(x))\right] = \mathbf{E}_{x\sim D}\left[\chi(x,-1)\right] + \mathbf{E}_{x\sim D_{f,+}}\left[\chi(x,1)-\chi(x,-1)\right]\cdot \mathbf{Pr}_{x\sim D}[f(x)=1]. \tag{5}$$

Given the above identity, the algorithm $\texttt{Simulate-STAT}_f^D$ is very simple: We use random sampling from D to empirically estimate the expectations $\mathbf{E}_{x\sim D}\left[\chi(x,-1)\right]$ (recall that D is assumed to be a samplable distribution), and we use the independent samples from $D_{f,+}$ to empirically estimate $\mathbf{E}_{x\sim D_{f,+}}\left[\chi(x,1)-\chi(x,-1)\right]$. Both estimates are obtained to within an additive accuracy of $\pm\tau/2$ (with confidence probability $1-\delta/2$ each). We combine these estimates with our estimate $\widetilde{b_f}$ for $\mathbf{Pr}_{x\sim D}[f(x)=1]$ in the obvious way (see Step 2 of pseudocode below).

Subroutine $\texttt{Simulate-STAT}_f^D(D, D_{f,+}, \chi, \tau, \widetilde{b_f}, \delta)$:

Input: Independent samples from D and $D_{f,+}$, query access to $\chi : \{-1,1\}^n \to \{-1,1\}$, accuracy τ, $\widetilde{b_f} \in [0,1]$ and confidence δ.

Output: If $|\widetilde{b_f} - \mathbf{Pr}_{x\sim D}[f(x)=1]| \leq \tau'$, a number v that with probability $1-\delta$ satisfies $|\mathbf{E}_{x\sim D}[\chi(x,f(x))] - v| \leq \tau + \tau'$.

1. Empirically estimate the values $\mathbf{E}_{x\sim D}[\chi(x,-1)]$ and $\mathbf{E}_{x\sim D_{f,+}}[\chi(x,1)-\chi(x,-1)]$ to within an additive $\pm\tau/2$ with confidence probability $1-\delta/2$. Let $\widetilde{E}_1, \widetilde{E}_2$ be the corresponding estimates.
2. Output $v = \widetilde{E}_1 + \widetilde{E}_2 \cdot \widetilde{b_f}$.

By Fact 25, we can estimate each expectation using $m = \Theta\left((1/\tau^2)\log(1/\delta)\right)$ samples (from D, $D_{f,+}$ respectively). For each such sample the estimation algorithm needs to evaluate the function χ (once for the first expectation and twice for the second). Hence, the total number of queries to χ is $O(m)$, i.e., the subroutine $\texttt{Simulate-STAT}_f^D$ runs in time $O(m\cdot t(n))$ as desired.

By a union bound, with probability $1-\delta$ both estimates will be $\pm\tau/2$ accurate. The bound (4) follows from this latter fact and (5) by a straightforward application of the triangle inequality. This completes the proof of Lemma 28. □

Given the above lemma, we can state and prove our general result for simulating SQ algorithms:

Proposition 29. *Let $\mathcal{C}$ be a concept class and D be a samplable distribution over $\{-1,1\}^n$. Suppose there exists an* SQ*-learning algorithm $\mathcal{A}_{\mathrm{SQ}}$ for $\mathcal{C}$ under D with the following performance: $\mathcal{A}_{\mathrm{SQ}}$ runs in time $T_1 = t_1(n, 1/\epsilon, 1/\delta)$, each query provided to $\mathrm{STAT}(f,D)$ can be evaluated in time $T_2 = t_2(n)$, and the minimum value of the tolerance provided to $\mathrm{STAT}(f,D)$ in the course of its execution is $\tau = \tau(n, 1/\epsilon)$. Then, there exists an algorithm $\mathcal{A}_{\mathrm{SQ-SIM}}$ that is given access to*

(i) independent samples from $D_{f,+}$; and

(ii) a number $\widetilde{b_f} \in [0,1]$,

and efficiently simulates the behavior of $\mathcal{A}_{\mathrm{SQ}}$. In particular, $\mathcal{A}_{\mathrm{SQ-SIM}}$ has the following performance guarantee: on input an accuracy ϵ and a confidence δ, it uses $m = O\left((1/\tau^2)\cdot\log(T_1/\delta)\cdot T_1\right)$ samples from D and $D_{f,+}$, runs in time $T_{\mathrm{SQ-SIM}} = O(mT_2)$, and if $|\widetilde{b_f} - \mathbf{Pr}_{x\sim D}[f(x)=1]| \leq \tau/2$ then with probability $1-\delta$ it outputs a hypothesis $h : \{-1,1\}^n \to \{-1,1\}$ such that $\mathbf{Pr}_{x\sim D}[h(x)\neq f(x)] \leq \epsilon$.

Proof of Proposition 29. The simulation procedure is very simple. We run the algorithm $\mathcal{A}_{\mathrm{SQ}}$ by simulating its queries using algorithm $\texttt{Simulate-STAT}_f^D$. The algorithm is described in the following pseudocode:

Algorithm $\mathcal{A}_{\mathrm{SQ-SIM}}(D, D_{f,+}, \epsilon, \widetilde{b_f}, \delta)$:

Input: Independent samples from D and $D_{f,+}$, $\widetilde{b_f} \in [0,1]$, ϵ, $\delta > 0$.

Output: If $|\widetilde{b_f} - \mathbf{Pr}_{x\sim D}[f(x) = 1]| \leq \tau/2$, a hypothesis h that with probability $1-\delta$ satisfies $\mathbf{Pr}_{x\sim D}[h(x) \neq f(x)] \leq \epsilon$.

1. Let $\tau = \tau(n, 1/\epsilon)$ be the minimum accuracy ever used in a query to $\mathrm{STAT}(f, D)$ during the execution of $\mathcal{A}_{\mathrm{SQ}}(\epsilon, \delta/2)$.
2. Run the algorithm $\mathcal{A}_{\mathrm{SQ}}(\epsilon, \delta/2)$, by simulating each query to $\mathrm{STAT}(f, D)$ as follows: whenever $\mathcal{A}_{\mathrm{SQ}}$ makes a query (χ, τ) to $\mathrm{STAT}(f, D)$, the simulation algorithm runs $\texttt{Simulate-STAT}_f^D(D, D_{f,+}, \chi, \tau/2, \tau/2, \delta/(2T_1))$.
3. Output the hypothesis h obtained by the simulation.

Note that we run the algorithm $\mathcal{A}_{\mathrm{SQ}}$ with confidence probability $1-\delta/2$. Moreover, each query to the $\mathrm{STAT}(f, D)$ oracle is simulated with confidence $1-\delta/(2T_1)$. Since $\mathcal{A}_{\mathrm{SQ}}$ runs for at most T_1 time steps, it certainly performs at most T_1 queries in total. Hence, by a union bound over these events, with probability $1-\delta/2$ all answers to its queries will be accurate to within an additive $\pm\tau/2$. By the guarantee of algorithm $\mathcal{A}_{\mathrm{SQ}}$ and a union bound, with probability $1-\delta$, the algorithm $\mathcal{A}_{\mathrm{SQ-SIM}}$ will output a hypothesis $h : \{-1,1\}^n \to \{-1,1\}$ such that $\mathbf{Pr}_{x\sim D}[h(x) \neq f(x)] \leq \epsilon$. The sample complexity and running time follow from the bounds for $\texttt{Simulate-STAT}_f^D$. This completes the proof of Proposition 29. □

Proposition 29 tells us we can efficiently simulate a statistical query algorithm for a concept class $\mathcal{C}$ under a samplable distribution D if we have access to samples drawn from $D_{f,+}$ (and a very accurate estimate of $\mathbf{Pr}_{x\sim D}[f(x) = 1]$). In our setting, we have that $D = \mathcal{U}_{g^{-1}(1)}$ where $g \in \mathcal{C}'$ is the function that is output by $\mathcal{A}_{\mathrm{den}}^{(\mathcal{C},\mathcal{C}')}$. So, the two issues we must handle are (i) obtaining samples from D, and (ii) obtaining samples from $D_{f,+}$.

For (i), we note that, even though we do not have access to samples drawn *exactly* from D, it suffices for our purposes to use a τ'-sampler for D for a sufficiently small τ'. To see this we use the following fact:

Fact 30. *Let D, D' be distributions over $\{-1,1\}^n$ with $d_{\mathrm{TV}}(D, D') \leq \tau'$. Then for any bounded function $\phi : \{-1,1\}^n \to [-1,1]$ we have that $|\mathbf{E}_{x\sim D}[\phi(x)] - \mathbf{E}_{x\sim D'}[\phi(x)]| \leq 2\tau'$.*

Proof of Fact 30. By definition we have that $|\mathbf{E}_{x\sim D}[\phi(x)] - \mathbf{E}_{x\sim D'}[\phi(x)]|$ equals

$$\begin{aligned}
\left|\sum_{x\in\{-1,1\}^n} \big(D(x) - D'(x)\big)\,\phi(x)\right| &\leq \sum_{x\in\{-1,1\}^n} \big|\big(D(x) - D'(x)\big)\big|\,|\phi(x)| \\
&\leq \max_{x\in\{-1,1\}^n} |\phi(x)| \cdot \sum_{x\in\{-1,1\}^n} \big|D(x) - D'(x)\big| \\
&\leq 1 \cdot \|D - D'\|_1 = 2d_{\mathrm{TV}}(D, D') \leq 2\tau'.
\end{aligned}$$

□

The above fact implies that the statement of Proposition 29 continuous to hold with the same parameters if instead of a 0-sampler for D we have access to a τ'-sampler for D, for $\tau' = \tau/8$. The only difference is that in Step 1 of the subroutine $\texttt{Simulate-STAT}_f^D$ we empirically estimate the expectation $\mathbf{E}_{x\sim D'}[\chi(x,-1)]$ up to an additive $\pm\tau/4$. By Fact 30, this will be a $\pm(\tau/4 + 2\tau') = \pm\tau/2$ accurate estimate for the $\mathbf{E}_{x\sim D}[\chi(x,-1)]$. That is, we have:

Corollary 31. *The statement of Proposition 29 continues to hold with the same parameters if instead of a* 0*-sampler for* D *we have access to a* $\tau' = \tau/8$*-sampler for* D*.*

For (ii), even though we do not have access to the distribution $D = \mathcal{U}_{g^{-1}(1)}$ directly, we note below that we can efficiently sample from $D_{f,+}$ using samples from $\mathcal{U}_{f^{-1}(1)}$ together with evaluations of g (recall again that g is provided as the output of the densifier).

Claim 32. *Let* $g : \{-1,1\}^n \to \{-1,1\}$ *be a* $t_g(n)$ *time computable function such that* $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq \epsilon'$*. There is an efficient subroutine that is given* ϵ' *and a circuit to compute* g *as input, uses* $m = O((1/\epsilon')\log(1/\delta))$ *samples from* $\mathcal{U}_{f^{-1}(1)}$*, runs in time* $O(m \cdot t_g(n))$*, and with probability* $1-\delta$ *outputs a sample* x *such that* $x \sim D_{f,+}$*, where* $D = \mathcal{U}_{g^{-1}(1)}$*.*

Proof of Claim 32. To simulate a sample from $D_{f,+}$ we simply draw samples from $\mathcal{U}_{f^{-1}(1)}$ until we obtain a sample x with $g(x) = 1$. The following pseudocode makes this precise:

> Subroutine $\texttt{Simulate-sample}^{D_{f,+}}(\mathcal{U}_{f^{-1}(1)}, g, \epsilon', \delta)$:
>
> **Input:** Independent samples from $\mathcal{U}_{f^{-1}(1)}$, a circuit computing g, a value $\epsilon' > 0$ such that $\epsilon' \leq \mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1]$ and confidence parameter δ.
>
> **Output:** A point $x \in \{-1,1\}^n$ that with probability $1-\delta$ satisfies $x \sim D_{f,+}$.
>
> 1. Repeat the following at most $m = \Theta\left((1/\epsilon')\log(1/\delta)\right)$ times:
>   (a) Draw a sample $x \sim \mathcal{U}_{f^{-1}(1)}$.
>   (b) If the circuit for g evaluates to 1 on input x then output x.
> 2. If no point x with $g(x) = 1$ has been obtained, halt and output "failure."

Since $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq \epsilon'$, after repeating this process $m = \Omega\left((1/\epsilon')\log(1/\delta)\right)$ times, we will obtain a satisfying assignment to g with probability at least $1-\delta$. It is clear that such a sample x is distributed according to $D_{f,+}$. For each sample we need to evaluate g once, hence the running time follows. □

Getting a good estimate $\widetilde{b_f}$ of $\mathbf{Pr}_{x\sim D}[f(x) = 1]$. The simulations presented above require an additively accurate estimate $\widetilde{b_f}$ of $\mathbf{Pr}_{x\sim D}[f(x) = 1]$. We now show that in our context, such an estimate can be easily obtained if we have access to a good estimate $\widehat{p}$ of $p = \mathbf{Pr}_{x\in\mathcal{U}_n}[f(x) = 1]$, using the fact that we have an efficient approximate counting algorithm for $\mathcal{C}'$ and that $D \equiv \mathcal{U}_{g^{-1}(1)}$ where $g \in \mathcal{C}'$.

Claim 33. *Let* $g : \{-1,1\}^n \to \{-1,1\}$*,* $g \in \mathcal{C}'$ *be a* $t_g(n)$ *time computable function, satisfying* $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x) = 1] \geq \gamma'$ *and* $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq 1-\epsilon'$*. Let* $\mathcal{A}^{\mathcal{C}'}_{\text{count}}$ *be an* (ϵ,δ)*-approximate counting algorithm for* $\mathcal{C}'$ *running in time* $T_{\text{count}}(n, 1/\epsilon, 1/\delta)$*. There is a procedure* `Estimate-Bias` *with the following behavior:* `Estimate-Bias` *takes as input a value* $0 < \widehat{p} \leq 1$*, a parameter* $\tau' > 0$*, a confidence parameter* δ'*, and a representation of* $g \in \mathcal{C}'$*.* `Estimate-Bias` *runs in time* $O(t_g \cdot T_{\text{count}}(n, 2/\tau', 1/\delta'))$ *and satisfies the following: if* $p \stackrel{def}{=} \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1] < \widehat{p} \leq (1+\epsilon')p$*, then with probability* $1-\delta'$ `Estimate-Bias` *outputs a value* $\widetilde{b_f}$ *such that* $|\widetilde{b_f} - \mathbf{Pr}_{x\sim D}[f(x) = 1]| \leq \tau'$*.*

Proof of Claim 33. The procedure `Estimate-Bias` is very simple. It runs $\mathcal{A}^{\mathcal{C}'}_{\text{count}}$ on inputs $\epsilon^\star = \tau'/2, \delta'$, using the representation for $g \in \mathcal{C}'$. Let p_g be the value returned by the approximate counter; `Estimate-Bias` returns $\widehat{p}/p_g$.

The claimed running time bound is obvious. To see that the procedure is correct, first observe that by Definition 17, with probability $1-\delta'$ we have that

$$\frac{|g^{-1}(1)|}{2^n} \cdot \frac{1}{1+\epsilon^\star} \leq p_g \leq \frac{|g^{-1}(1)|}{2^n} \cdot (1+\epsilon^\star).$$

For the rest of the argument we assume that the above inequality indeed holds. Let A denote $|g^{-1}(1)|$, let B denote $|f^{-1}(1) \cap g^{-1}(1)|$, and let C denote $|f^{-1}(1) \setminus g^{-1}(1)|$, so the true value $\mathbf{Pr}_{x\sim D}[f(x)=1]$ equals $\frac{B}{A}$ and the above inequality can be rephrased as $\frac{A}{1+\epsilon^\star} \le p_g \cdot 2^n \le A \cdot (1+\epsilon^\star)$. By our assumption on $\widehat{p}$ we have that $B + C \le \widehat{p} \cdot 2^n \le (1+\epsilon')(B+C)$; since $\mathbf{Pr}_{x\sim \mathcal{U}_{f^{-1}(1)}}[g(x)=1] \ge 1-\epsilon'$ we have $\frac{C}{B+C} \le \epsilon'$ (i.e., $C \le \frac{\epsilon'}{1-\epsilon'} \cdot B$); and since $\mathbf{Pr}_{x\sim \mathcal{U}_{g^{-1}(1)}}[f(x)=1] \ge \gamma'$ we have $\frac{B}{A} \ge \gamma'$. Combining these inequalities we get

$$\frac{1}{1+\epsilon^\star} \cdot \frac{B}{A} \le \frac{1}{1+\epsilon^\star} \cdot \frac{B+C}{A} \le \frac{\widehat{p}}{p_g} \le \frac{B}{A} \cdot (1+\epsilon')(1+\epsilon^\star)\left(1+\frac{\epsilon'}{1-\epsilon'}\right) = \frac{B}{A} \cdot (1+\epsilon^\star)$$

Hence

$$\left|\frac{B}{A} - \frac{\widehat{p}}{p_g}\right| \le \frac{B}{A}\left(1+\epsilon^\star - \frac{1}{1+\epsilon^\star}\right) \le \frac{2\epsilon^\star}{1+\epsilon^\star} \le 2\epsilon^\star,$$

where we have used $B \le A$. Recalling that $\epsilon^\star = \tau'/2$, the lemma is proved. □

C.2 An algorithm that succeeds given the (approximate) bias of f. In this section, we present an algorithm $\mathcal{A}'^{\mathcal{C}}(\epsilon, \delta, \widehat{p})$ which, in addition to samples from $\mathcal{U}_{f^{-1}(1)}$, takes as input parameters $\epsilon, \delta, \widehat{p}$. The algorithm succeeds in outputting a hypothesis distribution D_f satisfying $d_{\mathrm{TV}}(D_f, \mathcal{U}_{f^{-1}(1)}) \le \epsilon$ if the input parameter $\widehat{p}$ is a multiplicatively accurate approximation to $\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1]$. The algorithm follows the three high-level steps previously outlined and uses the subroutines of the previous subsection to simulate the statistical query algorithm. Detailed pseudocode follows:

Algorithm $\mathcal{A}'^{\mathcal{C}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$:

Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, accuracy and confidence parameters ϵ, δ, and a value $1/2^n < \widehat{p} \le 1$.

Output: If $\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1] \le \widehat{p} < (1+\epsilon)\,\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1]$, with probability $1-\delta$ outputs an ϵ-sampler C_f for $\mathcal{U}_{f^{-1}(1)}$.

1. **[Run the densifier to obtain g]**
   Fix $\epsilon_1 \stackrel{\text{def}}{=} \epsilon/6$ and $\gamma \stackrel{\text{def}}{=} \gamma(n, 1/\epsilon_1, 3/\delta)$. Run the γ-densifier $\mathcal{A}_{\mathrm{den}}^{(\mathcal{C},\mathcal{C}')}(\epsilon_1, \delta/3, \widehat{p})$ using random samples from $\mathcal{U}_{f^{-1}(1)}$. Let $g \in \mathcal{C}'$ be its output.
2. **[Run the SQ-learner, using the approximate uniform generator for g, to obtain hypothesis h]**
   (a) Fix $\epsilon_2 \stackrel{\text{def}}{=} \epsilon\gamma/7$, $\tau_2 \stackrel{\text{def}}{=} \tau(n, 1/\epsilon_2)$ and $m \stackrel{\text{def}}{=} \Theta\left((1/\tau_2^2)\cdot\log(T_1/\delta)\cdot T_1\right)$, where $T_1 = t_1(n, 1/\epsilon_2, 12/\delta)$.
   (b) Run the generator $\mathcal{A}_{\mathrm{gen}}^{\mathcal{C}'}(g, \tau_2/8, \delta/(12m))$ m times and let $S_D \subseteq \{-1,1\}^n$ be the multiset of samples obtained.
   (c) Run $\texttt{Simulate-sample}^{D_{f,+}}(\mathcal{U}_{f^{-1}(1)}, g, \gamma, \delta/(12m))$ m times and let $S_{D_{f,+}} \subseteq \{-1,1\}^n$ be the multiset of samples obtained.
   (d) Run `Estimate-Bias` with parameters $\widehat{p}$, $\tau' = \tau_2/2$, $\delta' = \delta/12$, using the representation for $g \in \mathcal{C}'$, and let $\widetilde{b_f}$ be the value it returns.
   (e) Run $\mathcal{A}_{\mathrm{SQ-SIM}}(S_D, S_{D_{f,+}}, \epsilon_2, \widetilde{b_f}, \delta/12)$. Let $h : \{-1,1\}^n \to \{-1,1\}$ be the output hypothesis.

3. **[Output the sampler which does rejection sampling according to h on draws from the approximate uniform generator for g]**

   Output the sampler C_f which works as follows:

   For $i = 1$ to $t = \Theta\left((1/\gamma)\log(1/(\delta\epsilon))\right)$ do:
   (a) Set $\epsilon_3 \stackrel{\text{def}}{=} \epsilon\gamma/48000$.
   (b) Run the generator $\mathcal{A}_{\text{gen}}^{\mathcal{C}'}(g, \epsilon_3, \delta\epsilon/(12t))$ and let $x^{(i)}$ be its output.
   (c) If $h(x^{(i)}) = 1$, output $x^{(i)}$.
   If no $x^{(i)}$ with $h(x^{(i)}) = 1$ has been obtained, output the default element $\perp$.

   Let $\hat{D}$ denote the distribution over $\{-1,1\}^n \cup \{\perp\}$ for which C_f is a 0-sampler, and let $\hat{D}'$ denote the conditional distribution of $\hat{D}$ restricted to $\{-1,1\}^n$ (i.e., excluding $\perp$).

We note that by inspection of the code for C_f, we have that the distribution $\hat{D}'$ is identical to $(D_{g,\epsilon_3})_{h^{-1}(1)}$, where D_{g,ϵ_3} is the distribution corresponding to the output of the approximate uniform generator when called on function g and error parameter ϵ_3 (see Definition 18) and $(D_{g,\epsilon_3})_{h^{-1}(1)}$ is D_{g,ϵ_3} conditioned on $h^{-1}(1)$.

We have the following:

Theorem 34. *Let $p \stackrel{def}{=} \mathbf{Pr}_{x\in\mathcal{U}_n}[f(x) = 1]$. Algorithm $\mathcal{A}'^{\mathcal{C}}(\epsilon, \delta, \widehat{p})$ has the following behavior: If $p \le \widehat{p} < (1+\epsilon)p$, then with probability $1-\delta$ the following both hold:*

(i) the output C_f is a sampler for a distribution $\hat{D}$ such that $d_{\text{TV}}(\hat{D}, \mathcal{U}_{f^{-1}(1)}) \le \epsilon$; and

(ii) the functions h, g satisfy $|h^{-1}(1) \cap g^{-1}(1)|/|g^{-1}(1)| \ge \gamma/2$.

The running time of $\mathcal{A}'^{\mathcal{C}}$ is polynomial in $T_{\text{den}}(n, 1/\epsilon, 1/\delta)$, $T_{\text{gen}}(n, 1/\epsilon, 1/\delta)$, $T_{\text{count}}(n, 1/\epsilon, 1/\delta)$, $t_1(n, 1/\epsilon, 1/\delta)$, $t_2(n)$, $1/\tau(n, 1/\epsilon)$, and $1/\gamma(n, 1/\epsilon, 1/\delta)$.

Proof. We give an intuitive explanation of the pseudocode in tandem with a proof of correctness. We argue that Steps 1-3 of the algorithm implement the corresponding steps of our high-level description and that the algorithm succeeds with confidence probability $1-\delta$.

We assume throughout the argument that indeed $\widehat{p}$ lies in $[p, (1+\epsilon)p)$. Given this, by Definition 7 with probability $1-\delta/3$ the function g satisfies properties (a) and (b) of Definition 7, i.e., $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \ge 1-\epsilon_1$ and $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x) = 1] \ge \gamma$. We condition on this event (which we denote E_1) going forth.

We now argue that Step 2 simulates the SQ learning algorithm $\mathcal{A}_{\text{SQ}}^{\mathcal{C}}$ to learn the function $f \in \mathcal{C}$ under distribution $D \equiv \mathcal{U}_{g^{-1}(1)}$ to accuracy ϵ_2 with confidence $1-\delta/3$. Note that the goal of Step (b) is to obtain m samples from a distribution D'' (the distribution "$D_{g,\tau_2/8}$" of Definition 18) such that $d_{\text{TV}}(D'', D) \le \tau_2/8$. To achieve this, we call the approximate uniform generator for g a total of m times with failure probability $\delta/(12m)$ for each call (i.e., each call returns $\perp$ with probability at most $\delta/(12m)$). By a union bound, with failure probability at most $\delta/12$, all calls to the generator are successful and we obtain a set S_D of m independent samples from D''. Similarly, the goal of Step (c) is to obtain m samples from $D_{f,+}$ and to achieve it we call the subroutine $\texttt{Simulate-sample}^{D_{f,+}}$ a total of m times with failure probability $\delta/(12m)$ each. By Claim 32 and a union bound, with failure probability at most $\delta/12$, this step is successful, i.e., it gives a set $S_{D_{f,+}}$ of m independent samples from $D_{f,+}$. The goal of Step (d) is to obtain a value $\widetilde{b_f}$ satisfying $|\widetilde{b_f} - \mathbf{Pr}_{x\sim D}[f(x) = 1]| \le \tau_2/2$; by Claim 33, with failure probability at most $\delta/12$ the value $\widetilde{b_f}$ obtained in this step is as desired. Finally, Step (e) applies the simulation algorithm $\mathcal{A}_{\text{SQ}-\text{SIM}}$ using the samples S_D and $S_{D_{f,+}}$ and the estimate $\widetilde{b_f}$ of $\mathbf{Pr}_{x\sim D}[f(x) = 1]$ obtained in the previous steps. Conditioning

on Steps (b), (c) and (d) being successful Corollary 31 implies that Step (e) is successful with probability $1 - \delta/12$, i.e., it outputs a hypothesis h that satisfies $\mathbf{Pr}_{x\sim D}[f(x) \neq h(x)] \leq \epsilon_2$. A union bound over Steps (c), (d) and (e) completes the analysis of Step 2. For future reference, we let E_2 denote the event that the hypothesis h constructed in Step 2(e) has $\mathbf{Pr}_{x\sim D}[f(x) \neq h(x)] \leq \epsilon_2$ (so we have that E_2 holds with probability at least $1 - \delta/3$; we additionally condition on this event going forth). We observe that since (as we have just shown) $\mathbf{Pr}_{x\sim \mathcal{U}_{g^{-1}(1)}}[f(x) \neq h(x)] \leq \epsilon_2$ and $\mathbf{Pr}_{x\sim \mathcal{U}_{g^{-1}(1)}}[f(x) = 1] \geq \gamma$, we have $\mathbf{Pr}_{x\sim \mathcal{U}_{g^{-1}(1)}}[h(x) = 1] \geq \gamma - \epsilon_2 \geq \gamma/2$, which gives item (ii) of the theorem; so it remains to establish item (i) and the claimed running time bound.

To establish (i), we need to prove that the output distribution $\hat{D}$ of the sampler C_f is ϵ-close in total variation distance to $\mathcal{U}_{f^{-1}(1)}$. This sampler attempts to draws t samples from a distribution D' such that $d_{\mathrm{TV}}(D', D) \leq \epsilon_3$ (this is the distribution "D_{g,ϵ_3}" in the notation of Definition 18) and it outputs one of these samples that satisfies h (unless none of these samples satisfies h, in which case it outputs a default element $\perp$). The desired variation distance bound follows from the next lemma for our choice of parameters:

Lemma 35. *Let $\hat{D}$ be the output distribution of $\mathcal{A}'^{\mathcal{C}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$. If $\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1] \leq \widehat{p} \leq (1 + \epsilon)\,\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1]$, then conditioned on Events E_1 and E_2, we have*

$$\begin{aligned} d_{\mathrm{TV}}(\hat{D}, \mathcal{U}_{f^{-1}(1)}) &\leq \frac{\epsilon}{6} + \frac{\epsilon}{6} + \frac{4\epsilon_3}{\gamma} + \epsilon_1 + \frac{\epsilon_2}{2\gamma} + \frac{\epsilon_2}{\gamma - \epsilon_2} \\ &\leq \frac{\epsilon}{6} + \frac{\epsilon}{6} + \frac{\epsilon}{12000} + \frac{\epsilon}{6} + \frac{\epsilon}{14} + \frac{\epsilon}{6} < \epsilon. \end{aligned}$$

Proof. Consider the distribution $D' = D_{g,\epsilon_3}$ (see Definition 18) produced by the approximate uniform generator in Step 3 of the algorithm. Let $D'|_{h^{-1}(1)}$ denote distribution D' restricted to $h^{-1}(1)$. Let S denote the set $g^{-1}(1) \cap h^{-1}(1)$. The lemma is an immediate consequence of Claims 36, 38, 39 and 40 below using the triangle inequality (everything below is conditioned on E_1 and E_2). □

Claim 36. $d_{\mathrm{TV}}(\hat{D}, \hat{D}') \leq \epsilon/6$.

Proof. Recall that $\hat{D}'$ is simply $\hat{D}$ conditioned on not outputting $\perp$.

We first claim that with probability at least $1 - \delta\epsilon/12$ all t points drawn in Step 3 of the code for C_f are distributed according to the distribution $D' = D_{g,\epsilon_3}$ over $g^{-1}(1)$. Each of the t calls to the approximate uniform generator has failure probability $\delta\epsilon/(12t)$ (of outputting $\perp$ rather than a point distributed according to D') so by a union bound no calls fail with probability at least $1 - \delta\epsilon/12$, and thus with probability at least $1 - \delta\epsilon/12$ indeed all t samples are independently drawn from such a distribution D'.

Conditioned on this, we claim that a satisfying assignment for h is obtained within the t samples with probability at least $1 - \delta\epsilon/12$. This can be shown as follows:

Claim 37. *Let $h : \{-1, 1\}^n \to \{-1, 1\}$ be the hypothesis output by $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ-SIM}}$. We have*

$$\mathbf{Pr}_{x\sim D'}[h(x) = 1] \geq \gamma/4.$$

Proof. First recall that, by property (b) in the definition of the densifier (Definition 7), we have $\mathbf{Pr}_{x\sim D}[f(x) = 1] \geq \gamma$. Since $d_{\mathrm{TV}}(D', D) \leq \epsilon_3$, by definition we get $\mathbf{Pr}_{x\sim D'}[f(x) = 1] \geq \mathbf{Pr}_{x\sim D}[f(x) = 1] - \epsilon_3 \geq \gamma - \epsilon_3 \geq 3\gamma/4$. Now by the guarantee of Step 2 we have that $\mathbf{Pr}_{x\sim D}[f(x) \neq h(x)] \leq \epsilon_2$. Combined with the fact that $d_{\mathrm{TV}}(D', D) \leq \epsilon_3$, this implies that $\mathbf{Pr}_{x\sim D'}[f(x) \neq h(x)] \leq \epsilon_2 + \epsilon_3 \leq \gamma/2$. Therefore, we conclude that

$$\mathbf{Pr}_{x\sim D'}[h(x) = 1] \geq \mathbf{Pr}_{x\sim D'}[f(x) = 1] - \mathbf{Pr}_{x\sim D'}[f(x) \neq h(x)] \geq 3\gamma/4 - \gamma/2 \geq \gamma/4$$

as desired. □

Hence, for an appropriate constant in the big-Theta specifying t, with probability at least $1 - \delta\epsilon/12 > 1 - \delta/12$ some $x^{(i)}$ is a satisfying assignment of h. that with probability at least $1 - \delta\epsilon/12$ some $x^{(i)}$, $i \in [t]$, has $h(x) = 1$. Thus with overall failure probability at most $\delta\epsilon/6$ a draw from $\hat{D}'$ is not $\perp$, and consequently we have $d_{\mathrm{TV}}(\hat{D}, \hat{D}') \leq \delta\epsilon/6 \leq \epsilon/6$. □

Claim 38. $d_{\mathrm{TV}}(\hat{D}', D'|_{h^{-1}(1)}) \leq \epsilon/6$.

Proof. The probability that any of the t points $x^{(1)}, \ldots, x^{(t)}$ is not drawn from D' is at most $t \cdot \delta\epsilon/(12t) < \epsilon/12$. Assuming that this does not happen, the probability that no $x^{(i)}$ lies in $h^{-1}(1)$ is at most $(1 - \gamma/4)^t < \delta\epsilon/12 < \epsilon/12$ by Claim 37. Assuming this does not happen, the output of a draw from $\hat{D}$ is distributed identically according to $D'|_{h^{-1}(1)}$. Consequently we have that $d_{\mathrm{TV}}(\hat{D}, D'|_{h^{-1}(1)}) \leq \epsilon/6$ as claimed. □

Claim 39. $d_{\mathrm{TV}}(D'|_{h^{-1}(1)}, \mathcal{U}_S) \leq 4\epsilon_3/\gamma$.

Proof. The definition of an approximate uniform generator gives us that $d_{\mathrm{TV}}(D', \mathcal{U}_{g^{-1}(1)}) \leq \epsilon_3$, and Claim 37 gives that $\mathbf{Pr}_{x \sim D'}[h(x) = 1] \geq \gamma/4$. We now recall the fact that for any two distributions D_1, D_2 and any event E, writing $D_i|_E$ to denote distribution D_i conditioned on event E, we have

$$d_{\mathrm{TV}}(D_1|_E, D_2|_E) \leq \frac{d_{\mathrm{TV}}(D_1, D_2)}{D_1(E)}.$$

The claim follows since $\mathcal{U}_{g^{-1}(1)}|_{h^{-1}(1)}$ is equivalent to $\mathcal{U}_S$. □

Claim 40. $d_{\mathrm{TV}}(\mathcal{U}_S, \mathcal{U}_{f^{-1}(1)}) \leq \epsilon_1 + \frac{\epsilon_2}{2\gamma} + \frac{\epsilon_2}{\gamma - \epsilon_2}$.

Proof. The proof requires a careful combination of the properties of the function g constructed by the densifier and the guarantee of the SQ algorithm. Recall that $S = g^{-1}(1) \cap h^{-1}(1)$. We consider the set $S' = g^{-1}(1) \cap f^{-1}(1)$. By the triangle inequality, we can bound the desired variation distance as follows:

$$d_{\mathrm{TV}}(\mathcal{U}_S, \mathcal{U}_{f^{-1}(1)}) \leq d_{\mathrm{TV}}(\mathcal{U}_{f^{-1}(1)}, \mathcal{U}_{S'}) + d_{\mathrm{TV}}(\mathcal{U}_{S'}, \mathcal{U}_S). \tag{6}$$

We will bound from above each term of the RHS in turn. To proceed we need an expression for the total variation distance between the uniform distribution on two finite sets. The following fact is obtained by straightforward calculation:

Fact 41. *Let A, B be subsets of a finite set $\mathcal{W}$ and $\mathcal{U}_A, \mathcal{U}_B$ be the uniform distributions on A, B respectively. Then,*

$$d_{\mathrm{TV}}(\mathcal{U}_A, \mathcal{U}_B) = (1/2) \cdot \frac{|A \cap \overline{B}|}{|A|} + (1/2) \cdot \frac{|B \cap \overline{A}|}{|B|} + (1/2) \cdot |A \cap B| \cdot \left| \frac{1}{|A|} - \frac{1}{|B|} \right|. \tag{7}$$

To bound the first term of the RHS of (6) we apply the above fact for $A = f^{-1}(1)$ and $B = S'$. Note that in this case $B \subseteq A$, hence the second term of (7) is zero. Regarding the first term, note that

$$\frac{|A \cap \overline{B}|}{|A|} = \frac{|f^{-1}(1) \cap \overline{g^{-1}(1)}|}{|f^{-1}(1)|} \leq \epsilon_1,$$

where the inequality follows from Property (a) of the densifier definition. Similarly, for the third term we can write

$$|A \cap B| \cdot \left| \frac{1}{|A|} - \frac{1}{|B|} \right| = |B| \cdot \left| \frac{1}{|A|} - \frac{1}{|B|} \right| = 1 - \frac{|B|}{|A|} = 1 - \frac{|f^{-1}(1) \cap g^{-1}(1)|}{|f^{-1}(1)|} \leq \epsilon_1,$$

where the inequality also follows from Property (a) of the densifier definition. We therefore conclude that $d_{\mathrm{TV}}(\mathcal{U}_{f^{-1}(1)}, \mathcal{U}_{S'}) \leq \epsilon_1$.

We now proceed to bound the second term of the RHS of (6) by applying Fact 41 for $A = S'$ and $B = S$. It turns out that bounding the individual terms of (7) is trickier in this case. For the first term we have:

$$\frac{|A \cap \overline{B}|}{|A|} = \frac{|f^{-1}(1) \cap g^{-1}(1) \cap \overline{h^{-1}(1)}|}{|f^{-1}(1) \cap g^{-1}(1)|} = \frac{|f^{-1}(1) \cap g^{-1}(1) \cap \overline{h^{-1}(1)}|}{|g^{-1}(1)|} \cdot \frac{|g^{-1}(1)|}{|f^{-1}(1) \cap g^{-1}(1)|} \leq \frac{\epsilon_2}{\gamma},$$

where the last inequality follows from the guarantee of the SQ learning algorithm and Property (b) of the densifier definition. For the second term we have

$$\frac{|B \cap \overline{A}|}{|B|} = \frac{|\overline{f^{-1}(1)} \cap g^{-1}(1) \cap h^{-1}(1)|}{|g^{-1}(1) \cap h^{-1}(1)|}.$$

To analyze this term we recall that by the guarantee of the SQ algorithm it follows that the numerator satisfies $|\overline{f^{-1}(1)} \cap g^{-1}(1) \cap h^{-1}(1)| \leq \epsilon_2 \cdot |g^{-1}(1)|$. From the same guarantee we also get $|f^{-1}(1) \cap g^{-1}(1) \cap \overline{h^{-1}(1)}| \leq \epsilon_2 \cdot |g^{-1}(1)|$. Now, Property (b) of the densifier definition gives $|f^{-1}(1) \cap g^{-1}(1)| \geq \gamma \cdot |g^{-1}(1)|$. Combing these two inequalities implies that

$$|g^{-1}(1) \cap h^{-1}(1)| \geq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| \geq (\gamma - \epsilon_2) \cdot |g^{-1}(1)|.$$

In conclusion, the second term is upper bounded by $(1/2) \cdot \frac{\epsilon_2}{\gamma - \epsilon_2}$.

For the third term, we can write

$$|A \cap B| \cdot \left| \frac{1}{|A|} - \frac{1}{|B|} \right| = |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| \cdot \left| \frac{1}{|f^{-1}(1) \cap g^{-1}(1)|} - \frac{1}{|g^{-1}(1) \cap h^{-1}(1)|} \right|.$$

To analyze these term we relate the cardinalities of these sets. In particular, we can write

$$\begin{aligned} |f^{-1}(1) \cap g^{-1}(1)| &= |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + |f^{-1}(1) \cap g^{-1}(1) \cap \overline{h^{-1}(1)}| \\ &\leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + \epsilon_2 \cdot |g^{-1}(1)| \\ &\leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + \frac{\epsilon_2}{\gamma} \cdot |f^{-1}(1) \cap g^{-1}(1)| \end{aligned}$$

where the last inequlity is Property (b) of the densifier defintion. Therefore, we obtain

$$(1 - \frac{\epsilon_2}{\gamma}) \cdot |f^{-1}(1) \cap g^{-1}(1)| \leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| \leq |f^{-1}(1) \cap g^{-1}(1)|.$$

Similarly, we have

$$\begin{aligned} |g^{-1}(1) \cap h^{-1}(1)| &= |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + |\overline{f^{-1}(1)} \cap g^{-1}(1) \cap h^{-1}(1)| \\ &\leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + \epsilon_2 \cdot |g^{-1}(1)| \\ &\leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + \frac{\epsilon_2}{\gamma - \epsilon_2} \cdot |g^{-1}(1) \cap h^{-1}(1)| \end{aligned}$$

and therefore

$$(1 - \frac{\epsilon_2}{\gamma - \epsilon_2}) \cdot |g^{-1}(1) \cap h^{-1}(1)| \leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| \leq |g^{-1}(1) \cap h^{-1}(1)|.$$

The above imply that the third term is bounded by $(1/2) \cdot \frac{\epsilon_2}{\gamma - \epsilon_2}$. This completes the proof of the claim. □

With Lemma 35 established, to finish the proof of Theorem 34 it remains only to establish the claimed running time bound. This follows from a straightforward (but somewhat tedious) verification, using the running time bounds established in Lemma 28, Proposition 29, Corollary 31, Claim 32 and Claim 33. □

C.3 Getting from $\mathcal{A}'^{\mathcal{C}}$ to $\mathcal{A}^{\mathcal{C}}$: An approximate evaluation oracle. Recall that the algorithm $\mathcal{A}'^{\mathcal{C}}$ from the previous subsection is only guaranteed (with high probability) to output a sampler for a hypothesis distribution $\hat{D}$ that is statistically close to the target distribution $\mathcal{U}_{f^{-1}(1)}$ if it is given an input parameter $\widehat{p}$ satisfying $p \leq \widehat{p} < (1+\epsilon)p$, where $p \stackrel{\text{def}}{=} \mathbf{Pr}_{x \in \mathcal{U}_n}[f(x) = 1]$. Given this, a natural idea is to run $\mathcal{A}'^{\mathcal{C}}$ a total of $k = O(n/\epsilon)$ times, using "guesses" for $\widehat{p}$ that increase multiplicatively as powers of $1+\epsilon$, starting at $1/2^n$ (the smallest possible value) and going up to 1. This yields hypothesis distributions $\hat{D}_1, \ldots, \hat{D}_k$ where $\hat{D}_i$ is the distribution obtained by setting $\widehat{p}$ to $\widehat{p}_i \stackrel{\text{def}}{=} (1+\epsilon)^{i-1}/2^n$. With such distributions in hand, an obvious approach is to use the "hypothesis testing" machinery of Section 2 to identify a high-accuracy $\hat{D}_i$ from this collection. This is indeed the path we follow, but some care is needed to make the approach go through; we present the detailed argument below.

Recall that as described in Proposition 6, the hypothesis testing algorithm requires the following:

1. independent samples from the target distribution $\mathcal{U}_{f^{-1}(1)}$ (this is not a problem since such samples are available in our framework);
2. independent samples from $\hat{D}_i$ for each i (also not a problem since the i-th run of algorithm $\mathcal{A}'^{\mathcal{C}}$ outputs a sampler for distribution $\hat{D}_i$; and
3. a $(1+O(\epsilon))$-approximate evaluation oracle $\mathrm{EVAL}_{\hat{D}_i}$ for each distribution $\hat{D}_i$.

In this subsection we show how to construct item (3) above, the approximate evaluation oracle. In more detail, we first describe a randomized procedure `Check` which is applied to the output of each execution of $\mathcal{A}'^{\mathcal{C}}$ (across all k different settings of the input parameter $\widehat{p}_i$). We show that with high probability the "right" value $\widehat{p}_{i^*}$ (the one which satisfies $p \leq \widehat{p}_{i^*} < (1+\epsilon)p$) will pass the procedure `Check`. Then we show that for each value $\widehat{p}_{i^*}$ that passed the check a simple deterministic algorithm gives the desired approximate evaluation oracle for $\hat{D}_i$.

We proceed to describe the `Check` procedure and characterize its performance.

Algorithm $\texttt{Check}(g, h, \delta', \epsilon)$:

Input: functions g and h as described in Lemma 42, a confidence parameter δ', and an accuracy parameter ϵ

Output: If $|h^{-1}(1) \cap g^{-1}(1)|/|g^{-1}(1)| \geq \gamma/2$, with probability $1-\delta'$ outputs a pair (α, κ) such that $|\alpha - |h^{-1}(1) \cap g^{-1}(1)|/|g^{-1}(1)|| \leq \mu \cdot |h^{-1}(1) \cap g^{-1}(1)|/|g^{-1}(1)|$ and $\frac{|g^{-1}(1)|}{1+\tau} \leq \kappa \leq (1+\tau)|g^{-1}(1)|$, where $\mu = \tau = \epsilon/40000$.

1. Sample $m = O(\log(2/\delta')/(\gamma\mu^2))$ points $x^1, \ldots, x^m$ from $\mathcal{A}^{\mathcal{C}'}_{\mathrm{gen}}(g, \gamma/4, \delta'/(2m))$. If any $x^j = \perp$ halt and output "failure."
2. Let α be $(1/m)$ times the number of points x^j that have $h(x) = 1$.
3. Call $\mathcal{A}^{\mathcal{C}'}_{\mathrm{count}}(\tau, \delta'/2)$ on g and set κ to 2^n times the value it returns.

Lemma 42. *Fix $i \in [k]$. Consider a sequence of k runs of $\mathcal{A}'^{\mathcal{C}}$ where in the i-th run it is given $\widehat{p}_i \stackrel{\text{def}}{=} (1+\epsilon)^{i-1}/2^n$ as its input parameter. Let g_i be the function in $\mathcal{C}'$ constructed by $\mathcal{A}'^{\mathcal{C}}$ in Step 1 of its i-th run and h_i be the hypothesis function constructed by $\mathcal{A}'^{\mathcal{C}}$ in Step 2(e) of its i-th run. Suppose* `Check` *is given as input g_i, h_i, a confidence parameter δ', and an accuracy parameter ϵ'. Then it either outputs "no" or a pair $(\alpha_i, \kappa_i) \in [0,1] \times [0, 2^{n+1}]$, and satisfies the following performance guarantee: If $|h_i^{-1}(1) \cap g_i^{-1}(1)|/|g_i^{-1}(1)| \geq \gamma/2$ then with probability at least $1-\delta'$* `Check` *outputs a pair (α_i, κ_i) such that*

$$\left| \alpha_i - \frac{|h_i^{-1}(1) \cap g_i^{-1}(1)|}{|g_i^{-1}(1)|} \right| \leq \mu \cdot \frac{|h_i^{-1}(1) \cap g_i^{-1}(1)|}{|g_i^{-1}(1)|} \tag{8}$$

and

$$\frac{|g_i^{-1}(1)|}{1+\tau} \le \kappa_i \le (1+\tau)|g_i^{-1}(1)|, \tag{9}$$

where $\mu = \tau = \epsilon/40000$.

Proof. Suppose that i is such that $|h_i^{-1}(1) \cap g_i^{-1}(1)|/|g_i^{-1}(1)| \ge \gamma/2$. Recall from Definition 18 that each point x^j drawn from $\mathcal{A}_{\mathrm{gen}}^{\mathcal{C}'}(g_i, \gamma/4, \delta'/(2m))$ in Step 1 is with probability $1-\delta'/(2m)$ distributed according to $D_{g_i,\gamma/4}$; by a union bound we have that with probability at least $1-\delta'/2$ all m points are distributed this way (and thus none of them are $\perp$). We condition on this going forward. Definition 18 implies that $d_{\mathrm{TV}}(D_{g_i,\gamma/4}, \mathcal{U}_{g_i^{-1}(1)}) \le \gamma/4$; together with the assumption that $|h_i^{-1}(1) \cap g_i^{-1}(1)|/|g_i^{-1}(1)| \ge \gamma/2$, this implies that each x^j independently has proability at least $\gamma/4$ of having $h(x) = 1$. Consequently, by the choice of m in Step 1, a standard multiplicative Chernoff bound implies that

$$\left|\alpha_i - \frac{|h^{-1}(1) \cap g^{-1}(1)|}{|g^{-1}(1)|}\right| \le \mu \cdot \frac{|h^{-1}(1) \cap g^{-1}(1)|}{|g^{-1}(1)|}$$

with failure probability at most $\delta'/4$, giving (8).

Finally, Definition 17 gives that (9) holds with failure probability at most $\delta'/4$. This concludes the proof. □

Next we show how a high-accuracy estimate α_i of $|h_i^{-1}(1) \cap g_i^{-1}(1)|/|g_i^{-1}(1)|$ yields a deterministic approximate evaluation oracle for $\hat{D}'_i$.

Lemma 43. *Algorithm* `Simulate-Approx-Eval` *(which is deterministic) takes as input a value* $\alpha \in [0,1]$*, a string* $x \in \{-1,1\}^n$*, a parameter* κ*, (a circuit for)* $h : \{-1,1\}^n \to \{-1,1\}$*, and (a representation for)* $g : \{-1,1\}^n \to \{-1,1\}$*,* $g \in \mathcal{C}'$*, where* h, g *are obtained from a run of* $\mathcal{A}'^{\mathcal{C}}$*. Suppose that*

$$\left|\alpha - \frac{|h^{-1}(1) \cap g^{-1}(1)|}{|g^{-1}(1)|}\right| \le \mu \cdot \frac{|h^{-1}(1) \cap g^{-1}(1)|}{|g^{-1}(1)|} \quad \textit{and} \quad \frac{|g^{-1}(1)|}{1+\tau} \le \kappa \le (1+\tau)|g^{-1}(1)|$$

where $\mu = \tau = \epsilon/40000$*. Then* `Simulate-Approx-Eval` *outputs a value* ρ *such that*

$$\frac{\hat{D}'(x)}{1+\beta} \le \rho \le (1+\beta)\hat{D}'(x), \tag{10}$$

where $\beta = \epsilon/192$*,* $\hat{D}$ *is the output distribution constructed in Step 3 of the run of* $\mathcal{A}^{\mathcal{C}}$ *that produced* h, g*, and* $\hat{D}'$ *is* $\hat{D}$ *conditioned on* $\{-1,1\}^n$ *(excluding* $\perp$*).*

Proof. The `Simulate-Approx-Eval` procedure is very simple. Given an input $x \in \{-1,1\}^n$ it evaluates both g and h on x, and if either evaluates to -1 it returns the value 0. If both evaluate to 1 then it returns the value $1/(\kappa\alpha)$.

For the correctness proof, note first that it is easy to see from the definition of the sampler C_f (Step 3 of $\mathcal{A}'^{\mathcal{C}}$) and Definition 18 (recall that the approximate uniform generator $\mathcal{A}_{\mathrm{gen}}^{\mathcal{C}'}(g)$ only outputs strings that satisfy g) that if $x \in \{-1,1\}^n$, $x \notin h^{-1}(1) \cap g^{-1}(1)$ then $\hat{D}$ has zero probability of outputting x, so `Simulate-Approx-Eval` behaves appropriately in this case.

Now suppose that $h(x) = g(x) = 1$. We first show that the value $1/(\kappa\alpha)$ is multiplicatively close to $1/|h^{-1}(1) \cap g^{-1}(1)|$. Let us write A to denote $|g^{-1}(1)|$ and B to denote $|h^{-1}(1) \cap g^{-1}(1)|$. With this notation we have $\left|\alpha - \frac{B}{A}\right| \le \mu \cdot \frac{B}{A}$ and $\frac{A}{1+\tau} \le \kappa \le (1+\tau)A$. Consequently, we have

$$B(1-\mu-\tau) \le B \cdot \frac{1-\mu}{1+\tau} = \frac{B}{A}(1-\mu) \cdot \frac{A}{1+\tau} \le \kappa\alpha \le \frac{B}{A}(1+\mu) \cdot (1+\tau)A \le B(1+2\mu+2\tau),$$

and hence

$$\frac{1}{B} \cdot \frac{1}{1+2\mu+2\tau} \leq \frac{1}{\kappa\alpha} \leq \frac{1}{B} \cdot \frac{1}{1-\mu-\tau}. \tag{11}$$

Now consider any $x \in h^{-1}(1) \cap g^{-1}(1)$. By Definition 18 we have that

$$\frac{1}{1+\epsilon_3} \cdot \frac{1}{|g^{-1}(1)|} \leq D_{g,\epsilon_3}(x) \leq (1+\epsilon_3) \cdot \frac{1}{|g^{-1}(1)|}.$$

Since a draw from $\hat{D}'$ is obtained by taking a draw from D_{g,ϵ_3} and conditioning on it lying in $h^{-1}(1)$, it follows that we have

$$\frac{1}{1+\epsilon_3} \cdot \frac{1}{B} \leq \hat{D}'(x) \leq (1+\epsilon_3) \cdot \frac{1}{B}.$$

Combining this with (11) and recalling that $\mu = \tau = \frac{\epsilon}{40000}$ and $\epsilon_3 = \frac{\epsilon\gamma}{48000}$, we get (10) as desired. □

C.4 The final algorithm: Proof of Theorem 8. Finally we are ready to give the distribution learning algorithm $\mathcal{A}^{\mathcal{C}}$ for $\mathcal{C}$ and compete the proof of Theorem 8.

Algorithm $\mathcal{A}^{\mathcal{C}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta)$

Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, accuracy and confidence parameters ϵ, δ.
Output: With probability $1-\delta$ outputs an ϵ-sampler C_f for $\mathcal{U}_{f^{-1}(1)}$.

1. For $i = 1$ to $k = O(n/\epsilon)$:
   (a) Set $\widehat{p}_i \stackrel{\text{def}}{=} (1+\epsilon)^{i-1}/2^n$.
   (b) Run $\mathcal{A}'^{\mathcal{C}}(\mathcal{U}_{f^{-1}(1)}, \epsilon/12, \delta/3, \widehat{p}_i)$. Let $g_i \in \mathcal{C}'$ be the function constructed in Step 1, h_i be the hypothesis function constructed in Step 2(e), and $(C_f)_i$ be the sampler for distribution $\hat{D}_i$ constructed in Step 3.
   (c) Run `Check`$(g_i, h_i, \delta/3, \epsilon)$. If it returns a pair (α_i, κ_i) then add i to the set S (initially empty).
2. Run the hypothesis testing procedure $\mathcal{T}^{\mathcal{U}_{f^{-1}(1)}}$ over the set $\{\hat{D}'_i\}_{i \in S}$ of hypothesis distributions, using accuracy parameter $\epsilon/12$ and confidence parameter $\delta/3$. Here $\mathcal{T}^{\mathcal{U}_{f^{-1}(1)}}$ is given access to $\mathcal{U}_{f^{-1}(1)}$, uses the samplers $(C_f)_i$ to generate draws from distributions $\hat{D}'_i$ (see Remark 27), and uses the procedure `Simulate-Approx-Eval`$(\alpha_i, \kappa_i, h_i, g_i)$ for the $(1+\epsilon/192)$-approximate evaluation oracle $\mathrm{EVAL}_{\hat{D}'_i}$ for $\hat{D}'_i$. Let $i^\star \in S$ be the index of the distribution that it returns.
3. Output the sampler $(C_f)_{i^\star}$.

Proof of Theorem 8: Let $p \equiv \mathbf{Pr}_{x \in \mathcal{U}_n}[f(x) = 1]$ denote the true fraction of satisfying assignments for f in $\{-1,1\}^n$. Let i^* be the element of $[k]$ such that $p \leq \widehat{p}_{i^*} < (1+\epsilon/6)p$. By Theorem 34, with probability at least $1-\delta/3$ we have that both

(i) $(C_f)_{i^*}$ is a sampler for a distribution $\hat{D}_{i^*}$ such that $d_{\mathrm{TV}}(\hat{D}_{i^*}, \mathcal{U}_{f^{-1}(1)}) \leq \epsilon/6$; and

(ii) $|h_{i^*}^{-1}(1) \cap g_{i^*}^{-1}(1)|/|g_{i^*}^{-1}(1)| \geq \gamma/2$.

We condition on these two events holding. By Lemma 42, with probability at least $1-\delta/3$ the procedure `Check` outputs a value α_{i^*} such that

$$\left| \alpha_{i^*} - \frac{|h_{i^*}^{-1}(1) \cap g_{i^*}^{-1}(1)|}{|g_{i^*}^{-1}(1)|} \right| \leq \mu \cdot \frac{|h_{i^*}^{-1}(1) \cap g_{i^*}^{-1}(1)|}{|g_{i^*}^{-1}(1)|}$$

for $\mu = \epsilon/40000$. We condition on this event holding. Now Lemma 43 implies that $\texttt{Simulate-Approx-Eval}((C_f)_{i^*})$ meets the requirements of a $(1+\beta)$-approximate evaluation oracle for $\mathrm{EVAL}_{\hat{D}'_{i^*}}$ from Proposition 6, for $\beta = \frac{\epsilon}{192}$. Hence by Proposition 6 (or more precisely by Remark 27) with probability at least $1 - \delta/3$ the index $i^\star$ that $\mathcal{T}^{\mathcal{U}_{f^{-1}(1)}}$ returns is such that $\hat{D}'_{i^\star}$ is an $\epsilon/2$-sampler for $\mathcal{U}_{f^{-1}(1)}$ as desired.

As in the proof of Theorem 34, the claimed running time bound is a straightforward consequence of the various running time bounds established for all the procedures called by $\mathcal{A}^{\mathcal{C}}$. This concludes the proof of our general positive result, Theorem 8. □

D Details from Section 4: Linear Threshold Functions

D.1 Tools from the literature. We first record two efficient algorithms for approximate uniform generation and approximate counting for $\mathbf{LTF}_n$, due to Dyer [Dye03]:

Theorem 44. *There is an algorithm $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{gen}}$ that on input (a weights–based representation of) an arbitrary $h \in \mathbf{LTF}_n$ and a confidence parameter $\delta > 0$, runs in time $\mathrm{poly}(n, \log(1/\delta))$ and with probability $1-\delta$ outputs a point x such that $x \sim \mathcal{U}_{h^{-1}(1)}$.*

Theorem 45. *There is an algorithm $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{count}}$ that on input (a weights–based representation of) an arbitrary $h \in \mathbf{LTF}_n$, an accuracy parameter $\epsilon > 0$ and a confidence parameter $\delta > 0$, runs in time $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ and outputs $\widehat{p} \in [0,1]$ that with probability $1-\delta$ satisfies $\widehat{p} \in [1-\epsilon, 1+\epsilon] \cdot \mathbf{Pr}_{x\sim\mathcal{U}_n}[h(x)=1]$.*

We also need an efficient SQ learning algorithm for halfpaces. This is provided to us by a result of Blum et. al. [BFKV97]:

Theorem 46. *There is a distribution-independent SQ learning algorithm $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{SQ}}$ for $\mathbf{LTF}_n$ that has running time $t_1 = \mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$, uses at most $t_2 = \mathrm{poly}(n)$ time to evaluate each query, and requires tolerance of its queries no smaller than $\tau = 1/\mathrm{poly}(n, 1/\epsilon)$.*

Proof of Claim 12: Consider an LTF g and suppose that it does not satisfy condition (a), i.e., $\mathbf{Pr}_{x\sim\mathcal{U}_n}[g(x) = -1 | f(x) = 1] > \epsilon$. Since each sample $x \in S_+$ is uniformly distributed in $f^{-1}(1)$, the probability it does not "hit" the set $g^{-1}(-1) \cap f^{-1}(1)$ is at most $1-\epsilon$. The probability that *no* sample in S_+ hits $g^{-1}(-1) \cap f^{-1}(1)$ is thus at most $(1-\epsilon)^{N_+} \leq \delta/2^{n^2}$. Recalling that there exist at most 2^{n^2} distinct LTFs over $\{-1,1\}^n$ [Mur71], it follows by a union bound that the probability there exists an LTF that does not satisfy condition (a) is at most δ as desired. □

D.2 Proof of Theorem 13 We start by recalling the notion of *online learning* for a class $\mathcal{C}$ of Boolean functions. In the online model, learning proceeds in a sequence of stages. In each stage the learning algorithm is given an unlabeled example $x \in \{-1,1\}^n$ and is asked to predict the value $f(x)$, where $f \in \mathcal{C}$ is the unknown target concept. After the learning algorithm makes its prediction, it is given the correct value of $f(x)$. The goal of the learner is to identify f while minimizing the total number of mistakes. We say that an online algorithm learns class $\mathcal{C}$ with mistake bound M if it makes at most M mistakes on *any* sequence of examples consistent with some $f \in \mathcal{C}$. Our densifier makes essential use of a computationally efficient online learning algorithm for the class of linear threshold functions by Maass and Turan [MT94]:

Theorem 47. *There exists a $\mathrm{poly}(n)$ time deterministic online learning algorithm $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{online}}$ for the class $\mathbf{LTF}_n$ with mistake bound $M(n) \stackrel{def}{=} \Theta(n^2 \log n)$. In particular, at every stage of its execution, the current hypothesis maintained by $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{online}}$ is a (weights–based representation of an) LTF that is consistent with all labeled examples received so far.*

We note that $\mathcal{A}^{\mathbf{LTF}}_{\text{online}}$ works by reducing online learning of LTFs to a convex optimization problem; however, our densifier will use algorithm $\mathcal{A}^{\mathbf{LTF}}_{\text{online}}$ as a black box.

We now proceed with a more detailed description of our densifier followed by pseudocode and a proof of correctness. As previously mentioned, the basic idea is to execute the online learner to learn f while cleverly providing counterexamples to it in each stage of its execution. Our algorithm starts by sampling N_+ samples from $\mathcal{U}_{f^{-1}(1)}$ and making sure that these are classified correctly by the online learner. This step guarantees that our final solution will satisfy condition (a) of the densifier. Let $h \in \mathbf{LTF}_n$ be the current hypothesis at the end of this process. If h satisfies condition (b) (we can efficiently decide this by using our approximate counter for $\mathbf{LTF}_n$), we output h and terminate the algorithm. Otherwise, we use our approximate uniform generator to construct a uniform satisfying assignment $x \in \mathcal{U}_{h^{-1}(1)}$ and we label it negative, i.e., we give the labeled example $(x, -1)$ as a counterexample to the online learner. Since h does not satisfy condition (b), i.e., it has "many" satisfying assignments, it follows that with high probability (roughly, at least $1 - \gamma$) over the choice of $x \in \mathcal{U}_{h^{-1}(1)}$, the point x output by the generator will indeed be negative for f. We continue this process for a number of stages. If all counterexamples thus generated are indeed consistent with f (this happens with probability roughly $1 - \gamma \cdot M$, where $M = M(n) = \Theta(n^2 \log n)$ is an upper bound on the number of stages), after at most M stages we have either found a hypothesis h satisfying condition (b) or the online learner terminates. In the latter case, the current hypothesis of the online learner is identical to f, as follows from Theorem 47. (Note that the above argument puts an upper bound of $O(\delta/M)$ on the value of γ.) Detailed pseudocode follows:

Algorithm $\mathcal{A}^{\mathbf{LTF}}_{\text{den}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$:

Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, parameters $\epsilon, \delta > 0$, and a value $1/2^n \leq \widehat{p} \leq 1$.

Output: If $p \leq \widehat{p} \leq (1+\epsilon)p$, with probability $1-\delta$ outputs a function $g \in \mathbf{LTF}_n$ satisfying conditions (a) and (b).

1. Draw a set S_+ of $N_+ = \Theta\left((1/\epsilon) \cdot (n^2 + \log(1/\delta))\right)$ examples from $\mathcal{U}_{f^{-1}(1)}$.
2. Initialize $i = 0$ and set $M \stackrel{\text{def}}{=} \Theta(n^2 \log n)$.
   While $(i \leq M)$ do the following:
   (a) Execute the i-th stage of $A^{\mathbf{LTF}}_{\text{online}}$ and let $h^{(i)} \in \mathbf{LTF}_n$ be its current hypothesis.
   (b) If there exists $x \in S_+$ with $h^{(i)}(x) = -1$ do the following:
   - Give the labeled example $(x, 1)$ as a counterexample to $A^{\mathbf{LTF}}_{\text{online}}$.
   - Set $i = i + 1$ and go to Step 2.

   (c) Run $\mathcal{A}^{\mathbf{LTF}}_{\text{count}}(h^{(i)}, \epsilon, \delta/(4M))$ and let $\widehat{p_i}$ be its output.
   (d) Set $\gamma \stackrel{\text{def}}{=} \delta/(16M)$. If $\widehat{p_i} \leq \widehat{p}/\left(\gamma \cdot (1+\epsilon)^2\right)$ then output $h^{(i)}$;
   (e) otherwise, do the following:
   - Run $\mathcal{A}^{\mathbf{LTF}}_{\text{gen}}(h^{(i)}, \delta/(4M))$ and let $x^{(i)}$ be its output.
   - Give the point $(x^{(i)}, -1)$ as a counterexample to $A^{\mathbf{LTF}}_{\text{online}}$.
   - Set $i = i + 1$ and go to Step 2.
3. Output the current hypothesis $h^{(i)}$ of $A^{\mathbf{LTF}}_{\text{online}}$.

Proof. First note that by Claim 12, with probability at least $1 - \delta/4$ over S_+ any LTF consistent with S_+ will satisfy condition (a). We will condition on this event and also on the event that each call to the approximate counting algorithm and to the approximate uniform generator is successful. Since Step 2 involves at most M iterations, by a union bound, with probability at least $1 - \delta/4$ all calls to $\mathcal{A}^{\mathbf{LTF}}_{\text{count}}$ will be successful, i.e., for all i we will have that $p_i/(1+\epsilon) \leq \widehat{p_i} \leq (1+\epsilon) \cdot p_i$, where $p_i = \mathbf{Pr}_{x \in \mathcal{U}_n}[h^{(i)}(x) = 1]$. Similarly,

with failure probability at most $\delta/4$, all points $x^{(i)}$ constructed by $\mathcal{A}_{\text{gen}}^{\mathbf{LTF}}$ will be uniformly random over $(h^{(i)})^{-1}(1)$. Hence, with failure probability at most $3\delta/4$ all three conditions will be satisfied.

Conditioning on the above events, if the algorithm outputs a hypothesis $h^{(i)}$ in Step 2(d), this hypothesis will certainly satisfy condition (b), since $p_i \leq (1+\epsilon)\widehat{p}_i \leq \widehat{p}/\big(\gamma \cdot (1+\epsilon)\big) \leq p/\gamma$. In this case, the algorithm succeeds with probability at least $1 - 3\delta/4$. It remains to show that if the algorithm returns a hypothesis in Step 3, it will be successful with probability at least $1 - \delta$. To see this, observe that if no execution of Step 2(e) generates a point $x^{(i)}$ with $f(x^{(i)}) = 1$, all the counterexamples given to $A_{\text{online}}^{\mathbf{LTF}}$ are consistent with f. Therefore, by Theorem 47, the hypothesis of Step 3 will be identical to f, which trivially satisfies both conditions.

We claim that with overall probability at least $1 - \delta/4$ all executions of Step 2(e) generate points $x^{(i)}$ with $f(x^{(i)}) = -1$. Indeed, fix an execution of Step 2(e). Since $\widehat{p}_i > \widehat{p}/\left((1+\epsilon)^2 \cdot \gamma\right)$, it follows that $p \leq (4\gamma)p_i$. Hence, with probability at least $1 - 4\gamma$ a uniform point $x^{(i)} \sim \mathcal{U}_{(h^i)^{-1}(1)}$ is a negative example for f, i.e., $x^{(i)} \in f^{-1}(-1)$. By a union bound over all stages, our claim holds except with failure probability $4\gamma \cdot M = \delta/4$, as desired. This completes the proof of correctness.

It remains to analyze the running time. Note that Step 2 is repeated at most $M = O(n^2 \log n)$ times. Each iteration involves (i) one round of the online learner $\mathcal{A}_{\text{online}}^{\mathbf{LTF}}$ (this takes $\text{poly}(n)$ time by Theorem 47), (ii) one call of $\mathcal{A}_{\text{count}}^{\mathbf{LTF}}$ (this takes $\text{poly}(n, 1/\epsilon, \log(1/\delta))$ time by Theorem 45), and (iii) one call to $\mathcal{A}_{\text{gen}}^{\mathbf{LTF}}$ (this takes $\text{poly}(n, 1/\epsilon, \log(1/\delta))$ time by Theorem 44). This completes the proof of Theorem 13. □

Discussion. As mentioned before, the algorithm $\mathcal{A}_{\text{den}}^{\mathbf{LTF}}$ is the most important technical contribution of this section and hence it is instructive to understand, at a high level, the ingredients which are combined to construct a densifier. Let $\mathcal{C}$ be a class of Boolean functions and $\mathcal{C}_n$ consist of functions in $\mathcal{C}$ over n variables. While we do not formalize it here, one can modify the proof of Theorem 13, *mutatis mutandis*, to show that $\mathcal{C}_n$ has a $(\epsilon, \gamma, \delta)$-densifier $\mathcal{A}_{\text{den}}^{\mathcal{C}}$ with running time $T(n)$ (where ϵ, γ and δ are $1/\text{poly}(n)$ and $T(n)$ is $\text{poly}(n)$) provided that the following conditions hold:

(i) $\mathcal{C}_n$ has an (ϵ, δ)-approximate counting algorithm and an (ϵ, δ)-approximate uniform generation algorithm both of which run in time $\text{poly}(n, 1/\epsilon, 1/\delta)$;

(ii) There is an online learning algorithm $\mathcal{A}_{OL}^{\mathcal{C}}$ for $\mathcal{C}$ with a $\text{poly}(n)$ running time and $\text{poly}(n)$ mistake bound.

It will be interesting to see if there are other interesting classes of functions for which this framework gives an "automatic" construction of densifiers.

E Proof of Theorem 14: DNF

E.1 Tools from the literature for DNF. Karp, Luby and Madras [KLM89] have given approximate uniform generation and approximate counting algorithms for DNF formulas. (We note that [JVV86] give an efficient algorithm that with high probability outputs an *exactly* uniform satisfying assignment for DNFs.)

Theorem 48. *(Approximate uniform generation for DNFs, [KLM89]) There is an approximate uniform generation algorithm $\mathcal{A}_{\text{gen}}^{\mathbf{DNF}_{n,t}}$ for the class $\mathbf{DNF}_{n,t}$ that runs in time $\text{poly}(n, t, 1/\epsilon, \log(1/\delta))$.*

Theorem 49. *(Approximate counting for DNFs, [KLM89]) There is an approximate counting algorithm $\mathcal{A}_{\text{gen}}^{\mathbf{DNF}_{n,t}}$ for the class $\mathbf{DNF}_{n,t}$ that runs in time $\text{poly}(n, t, 1/\epsilon, \log(1/\delta))$.*

Algorithm $\mathcal{A}_{\text{den}}^{\mathbf{DNF}_{n,s}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$:

Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, parameters $\epsilon, \delta > 0$, and a value $1/2^n < \widehat{p} \leq 1$.

Output: If $p \leq \widehat{p} \leq (1+\epsilon)p$, with probability $1-\delta$ outputs a set $\mathcal{S}$ of conjunctions $C_1, \ldots, C_{|\mathcal{S}|}$ as described in Theorem 16

1. Initialize set $\mathcal{S}$ to $\emptyset$. Let $\ell(\cdot)$ be the polynomial from Theorem 15.
2. For $i = 1$ to $M = 2n^{2\log(2s/\ell(\epsilon/s))}\log(s/\delta)$, repeat the following:
   (a) Draw $r = 2\log n$ satisfying assignments $x^1, \ldots, x^r$ from $\mathcal{U}_{f^{-1}(1)}$.
   (b) Let C_i be the AND of all literals that take the same value in all r strings $x^1, \ldots, x^r$ (note C_i may be the empty conjunction). We say C_i is a *candidate term.*
   (c) If the candidate term C_i satisfies $\mathbf{Pr}_{x\sim\mathcal{U}_n}[C_i(x) = 1] \leq \widehat{p}$ then add C_i to the set $\mathcal{S}$.
3. Output $\mathcal{S}$.

E.2 Proof of Theorem 16 The following crucial claim makes the intuition described in the proof sketch of Theorem 16 precise:

Claim 50. *Suppose T_j is a term in f such that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[T_j(x) = 1] \geq \ell(\epsilon/s)/(2s)$. Then with probability at least $1-\delta/s$, term T_j is a candidate term at some iteration of Step 2 of Algorithm $\mathcal{A}_{\text{den}}^{\mathbf{DNF}_{n,s}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$.*

Proof. Fix a given iteration i of the loop in Step 2. With probability at least

$$(\ell(\epsilon/s)/(2s))^{2\log n} = (1/n)^{2\log(2s/\ell(\epsilon/s))},$$

all $2\log n$ points $x^1, \ldots, x^{2\log n}$ satisfy T_j; let us call this event E, and condition on E taking place. We claim that conditioned on E, the points $x^1, \ldots, x^{2\log n}$ are independent uniform samples drawn from $T_j^{-1}(1)$. (To see this, observe that each x^i is an independent sample chosen uniformly at random from $f^{-1}(1) \cap T_j^{-1}$; but $f^{-1}(1) \cap T_j^{-1}(1)$ is identical to $T_j^{-1}(1)$.) Given that $x^1, \ldots, x^{2\log n}$ are independent uniform samples drawn from $T_j^{-1}(1)$, the probability that any literal which is *not* present in T_j is contained in C_i (i.e., is satisfied by all $2\log n$ points) is at most $2n/n^2 \leq 1/2$. So with overall probability at least $\frac{1}{2n^{2\log(2s/\ell(\epsilon/s))}}$, the term T_j is a candidate term at iteration i. Consequently T_j is a candidate term at some iteration with probability at least $1-\delta/s$, by the choice of $M = 2n^{2\log(2s/\ell(\epsilon/s))}\log(s/\delta)$. □

Now we are ready to prove Theorem 16:

Proof of Theorem 16. The claimed running time bound of $\mathcal{A}_{\text{den}}^{\mathbf{DNF}_{n,s}}$ is easily verified, so it remains only to establish (1)-(3). Fix $\widehat{p}$ such that $p \leq \widehat{p} < (1+\epsilon)p$ where $p = \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1]$.

Consider any fixed term T_j of f such that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[T_j(x) = 1] \geq \ell(\epsilon/s)/(2s)$. By Claim 50 we have that with probability at least $1-\delta/s$, term T_j is a candidate term at some iteration of Step 2 of the algorithm. We claim that in step (c) of this iteration the term T_j will in fact be added to $\mathcal{S}$. This is because by assumption we have

$$\mathbf{Pr}_{x\sim\mathcal{U}_n}[T_j(x) = 1] \leq \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1] = p \leq \widehat{p}.$$

So by a union bound, with probability at least $1-\delta$ every term T_j in f such that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[T_j(x) = 1] \geq \ell(\epsilon/s)/(2s)$ is added to $\mathcal{S}$.

Let L be the set of those terms T_j in f that have $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[T_j(x) = 1] \geq \ell(\epsilon/s)/(2s)$. Let f' be the DNF obtained by taking the OR of all terms in L. By a union bound over the (at most s) terms that are in

f but not in f', we have $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[f'(x)=1] \geq 1-\ell(\epsilon/s)/2$. Since g (as defined in Theorem 16 has $g(x)=1$ whenever $f'(x)=1$, it follows that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x)=1] \geq 1-\ell(\epsilon/s)/2 \geq 1-\epsilon$, giving item (1) of the theorem.

For item (2), since $f(x)=1$ whenever $f'(x)=1$, we have $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x)=1] \geq \mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f'(x)=1]$. Every x such that $f'(x)=1$ also has $g(x)=1$ so to lower bound $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f'(x)=1]$ it is enough to upper bound the number of points in $g^{-1}(1)$ and lower bound the number of points in $f'^{-1}(1)$. Since each C_i that is added to $\mathcal{S}$ is satisfied by at most $\widehat{p}2^n \leq (1+\epsilon)p2^n$ points, we have that $|g^{-1}(1)| \leq (1+\epsilon)pM2^n$. Since at least $1-\epsilon$ of the points that satisfy f also satisfy f', we have that $|f'^{-1}(1)| \geq p(1-\epsilon)2^n$. Thus we have $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f'(x)=1] \geq p(1-\epsilon)/((1+\epsilon)pM) = \frac{1-\epsilon}{1+\epsilon}\cdot\frac{1}{M} > \frac{1}{2M}$, giving (2).

Finally, for (3) we have that $f(x) \neq f'(x)$ only on those inputs that have $f(x)=1$ but $f'(x)=0$ (because some term outside of L is satisfied by x and no term in L is satisfied by x). Even if all such inputs x lie in $g^{-1}(1)$ (the worst case), there can be at most $(\ell(\epsilon/s)/2)p2^n$ such inputs, and we know that $|g^{-1}(1)| \geq |f^{-1}(1)| \geq p(1-\epsilon)2^n$. So we have $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x)\neq f'(x)] \leq \frac{\ell(\epsilon/s)/2}{1-\epsilon} \leq \ell(\epsilon/s)$, and we have (3) as desired. □

F Cryptographic hardness of distribution learning

In this section, we will prove hardness results for distribution learning of specific classes $\mathcal{C}$ of Boolean functions. As is standard in computational learning theory, our hardness results are based on cryptographic hardness assumptions. The hardness assumptions we use are well studied assumptions in cryptography such as the strong RSA assumption and the Decisional Diffie-Hellman problem.

F.1 Hardness results based on signature schemes. In this subsection we prove a general theorem, Theorem 56, which relates the hardness of distribution learning to the existence of certain secure signature schemes in cryptography. Roughly speaking, Theorem 56 says that if secure signature schemes exist, then the distribution learning problem is computationally hard for any class $\mathcal{C}$ which is Levin-reducible from CIRCUIT-SAT. We will use this general result to establish hardness of distribution learning for several natural classes of functions, including 3-CNF formulas, intersections of two halfspaces, and degree-2 polynomial threshold functions (PTFs).

We begin by recalling the definition of public key signature schemes. For an extensive treatment of signature schemes, see [Gol04]. For simplicity, and since it suffices for our purposes, we only consider schemes with deterministic verification algorithms.

Definition 51. *A signature scheme is a triple (G,S,V) of polynomial-time algorithms with the following properties :*

- **(Key generation algorithm)** *G is a randomized algorithm which on input 1^n produces a pair (pk,sk) (note that the sizes of both pk and sk are polynomial in n).*
- **(Signing algorithm)** *S is a randomized algorithm which takes as input a message m from the message space $\mathcal{M}$, a secret key sk and randomness $r\in\{0,1\}^n$, and outputs a signature $\sigma = S(m,sk,r)$.*
- **(Verification algorithm)** *V is a deterministic algorithm such that $V(m,pk,\sigma)=1$ for every $\sigma = S(m,sk,r)$.*

For the rest of this paper, we will only consider the so-called *unique signature schemes* (which are defined below). While it is possible to prove our theorems using somewhat more general signature schemes, we choose unique signature schemes to simplify the exposition.

Definition 52. *A signature scheme (G, S, V) is said to be a unique signature scheme if for every choice of (pk, sk) and $m \in \mathcal{M}$, there is a unique σ_m such that $V(m, pk, \sigma_m) = 1$. In this case, the signing algorithm S is deterministic.*

From now on, whenever we refer to signature schemes, we will mean unique signature schemes. We next define the standard notion of existential unforgeability under Random Message Attack (RMA).

Definition 53. *A signature scheme (G, S, V) is said to be* (t, ϵ)-RMA secure *if the following holds: Let $(pk, sk) \leftarrow G(1^n)$. Let $(m_1, \ldots, m_t)$ be chosen uniformly at random from $\mathcal{M}$. Let $\sigma_i \leftarrow S(m_i, sk)$. Then, for any probabilistic algorithm A running in time t,*

$$\Pr_{(pk,sk),(m_1,\ldots,m_t),(\sigma_1,\ldots,\sigma_t)}[A(pk, m_1, \ldots, m_t, \sigma_1, \ldots, \sigma_t) = (m', \sigma')] \leq \epsilon$$

where $V(m', pk, \sigma') = 1$ and $m' \neq m_i$ for all $i = 1, \ldots, t$.

Next we need to formally define the notion of hardness of distribution learning for a class $\mathcal{C}$:

Definition 54. *Let $\mathcal{C}$ be a class of n-variable Boolean functions. $\mathcal{C}$ is said to be* $(t(n), \epsilon, \delta)$-hard for distribution learning *if there is no algorithm A running in time $t(n)$ which is an (ϵ, δ) distribution learning algorithm for $\mathcal{C}$.*

Finally, we will also need the definition of an invertible Levin reduction:

Definition 55. *A binary relation R is said to reduce to another binary relation R' by a* time-t invertible Levin reduction *if there are three algorithms α, β and γ, each running in time $t(n)$ on instances of length n, with the following property:*

- *For every $(x, y) \in R$, it holds that $(\alpha(x), \beta(x, y)) \in R'$;*
- *For every $(\alpha(x), z) \in R'$, it holds that $(x, \gamma(\alpha(x), z)) \in R$.*

Furthermore, the functions β and γ are injective maps with the property that $\gamma(\alpha(x), \beta(x, y)) = y$.

Note that for any class of functions $\mathcal{C}$, we can define the binary relation $R_{\mathcal{C}}$ as follows : $(f, x) \in R_{\mathcal{C}}$ if and only if $f(x) = 1$ and $f \in \mathcal{C}$. In this section, whenever we say that there is an invertible Levin reduction from class $\mathcal{C}_1$ to class $\mathcal{C}_2$, we mean that there is an invertible Levin reduction between the corresponding binary relations $R_{\mathcal{C}_1}$ and $R_{\mathcal{C}_2}$.

F.1.1 A general hardness result based on signature schemes.

We now state and prove our main theorem relating signature schemes to hardness of distribution learning:

Theorem 56. *Let (G, S, V) be a (t, ϵ)-RMA secure signature scheme. Let $\mathcal{C}$ be a class of n-variable Boolean functions such that there is a Levin reduction from* CIRCUIT-SAT *to $\mathcal{C}$ running in time $t'(n)$. Let κ_1 and κ_2 be such that $\kappa_1 \leq 1 - 2 \cdot t'(n)/|\mathcal{M}|$, $\kappa_2 \leq 1$ and $\epsilon \leq (1 - \kappa_1)(1 - \kappa_2)/4$. If $t_1(\cdot)$ is a time function such that $2t_1(t'(n)) \leq t(n)$, then $\mathcal{C}$ is $(t_1(n), \kappa_1, \kappa_2)$-hard for distribution learning.*

The idea of the proof is simple: Suppose there were an efficient distribution learning algorithm for $\mathcal{C}$. Because of the invertible Levin reduction from CIRCUIT-SAT to $\mathcal{C}$, there is a signature scheme for which the verification algorithm (using any given public key) corresponds to a function in $\mathcal{C}$. The signed messages $(m_1, \sigma_1), \ldots, (m_t, \sigma_t)$ correspond to points from $\mathcal{U}_{f^{-1}(1)}$ where $f \in \mathcal{C}$. Now the existence of an efficient distribution learning algorithm for $\mathcal{C}$ (i.e. an algorithm which, given points from $\mathcal{U}_{f^{-1}(1)}$, can generate more such points) translates into an algorithm which, given a sample of signed messages, can generate a new signed message. But this violates the existential unforgeability under RMA of the signature scheme. We now proceed to the formal proof.

Proof. Assume towards a contradiction that there is a distribution learning algorithm A which runs in time t_1 such that with probability $1 - \kappa_2$, the output distribution is κ_1-close to the target distribution. We will now use algorithm A to construct an adversary which breaks the signature scheme for key pairs (pk, sk).

Towards this, fix a key pair (pk, sk) and consider the function $V_{pk} : \mathcal{M} \times \{0,1\}^* \to \{0,1\}$ defined as $V_{pk}(m, \sigma) = V(m, pk, \sigma)$. Clearly, V_{pk} is an instance of CIRCUIT-SAT (i.e. V_{pk} is computed by a satisfiable polynomial-size Boolean circuit). Since there is an invertible Levin reduction from CIRCUIT-SAT to $\mathcal{C}$, given pk, the adversary in time $t'(n)$ can compute $\Phi_{pk} \in \mathcal{C}$ with the following properties (let β and γ be the corresponding algorithms in the definition of the Levin reduction):

- For every (m, σ) such that $V_{pk}(m, \sigma) = 1$, $\Phi_{pk}(\beta(V_{pk}, (m, \sigma))) = 1$.
- For every x such that $\Phi_{pk}(x) = 1$, $V_{pk}(\gamma(\Phi_{pk}, x)) = 1$.

Let $m \in \mathcal{M}$ be a randomly chosen message and let σ_m be the corresponding signature. It is obvious that $\beta(V_{pk}, (m, \sigma_m))$ has the same distribution as a randomly chosen element from $\Phi_{pk}^{-1}(1)$. Recall that the adversary receives random message-signature pairs $\{(m_i, \sigma_i)\}_{i=1}^{t'(n)}$. Define $x_i = \beta(V_{pk}, (m_i, \sigma_{m_i}))$. Observe that $x_1, \ldots, x_{t'(n)}$ are distributed exactly as $t'(n)$ random elements from $\Phi_{pk}^{-1}(1)$. Also, the instances x_i are of length bounded by $t'(n)$. We run the algorithm A on the inputs $x_1, \ldots, x_{t'(n)}$. Note that the total running time of this procedure is bounded by $t(n)$. Let the output of the algorithm be a sampler $\mathcal{S}$. Now, we run the sampler $\mathcal{S}$ and apply the map β to its output. Observe that the total running time is bounded by $2t(n)$.

Now observe that with probability $1 - \kappa_2$, the output distribution of the sampler is κ_1 close to being uniform on $\Phi_{pk}^{-1}(1)$. For this sampler, when we apply the map β to its output, we obtain a valid message-signature pair $(m', \sigma_{m'})$ with probability $1 - \kappa_1$. Furthermore, m' is distinct from all $\{m_i\}_{i=1}^{t'(n)}$ with probability $1 - t'(n)/|\mathcal{M}|$. Thus, the adversary succeeds in breaking the security of the signature scheme with probability $1 - \kappa_1 - t'(n)/|\mathcal{M}|$.

Thus, with overall probability $(1-\kappa_1)(1-\kappa_2)/4 \geq \epsilon$, the adversary succeeds in producing $z = g(m, \sigma)$ such that $\forall i \in [t'], m_i \neq m$. Applying the map γ on (Φ_{pk}, z), the adversary gets the pair (m, σ). Also, note that the total running time of the adversary is $t_1(t'(n)) + t'(n) \leq 2t_1(t'(n)) \leq t(n)$ which contradicts the (t, ϵ)-RMA security of the signature scheme. □

F.1.2 A specific hardness assumption. At this point, we consider a particular instantiation of a unique signature scheme from the literature which meets our requirements. There are many other assumptions under which such signature schemes can be constructed (as we remark shortly). To state our cryptographic assumption, we need the following notation: PRIMES_k is the set of k-bit prime numbers. RSA_k is the set of all products of two primes of length $\lfloor (k-1)/2 \rfloor$. The following cryptographic assumption (a slight variant of the standard RSA assumption) appears in [MRV99].

Assumption 1. ***The RSA′ $s(k)$ assumption****: Fix any $m \in \text{RSA}_k$ and let $x \in_U \mathbb{Z}_m^*$ and $p \in_U \text{PRIMES}_{k+1}$. Let A be any probabilistic algorithm running in time $s(k)$. Then,*

$$\mathbf{Pr}_{(x,p)}[A(m, x, p) = y \text{ and } y^p = x \ (mod\ m)] \leq \frac{1}{s(k)}.$$

As mentioned in [MRV99], given the present state of computational number theory, it is plausible to conjecture the RSA′ $s(k)$ assumption for $s(k) = 2^{k^\delta}$ for some absolute constant $\delta > 0$. For the sake of conciseness, for the rest of this section we write "Assumption 1 holds true" to mean that Assumption 1 holds true with $s(k) = 2^{n^\delta}$ for some fixed constant $\delta > 0$. (We note, though, that all our hardness results go through giving superpolynomial hardness using only $s(k) = k^{\omega(1)}$.) Micali *et al.* [MRV99] give a construction of a unique signature scheme using Assumption 1:

Theorem 57. *If Assumption 1 holds true, then there is a $(t = 2^{n^\delta}, \epsilon = 1/t)$-RMA secure unique signature scheme (G, S, V).*

Remark 58. *It is important to note here that there are other constructions of unique signature schemes known in literature. For example, Lysyanskaya [Lys02] constructed a unique signature scheme using a strong version of the Diffie–Hellman assumption. Similarly, Hohenberger and Waters [HW10] constructed a unique signature scheme using a variant of the Diffie–Hellman assumption on bilinear groups.*

Instantiating Theorem 56 with the signature scheme from Theorem 57, we obtain the following corollary:

Corollary 59. *Suppose that Assumption 1 holds true. Then the following holds : Let $\mathcal{C}$ be a function class such that there is a polynomial time (n^k-time) invertible Levin reduction from CIRCUIT-SAT to $\mathcal{C}$. Then $\mathcal{C}$ is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for distribution learning for some constant $c > 0$ (depending only on the "δ" in Assumption 1 and on k).*

F.1.3 Hardness results for specific function classes whose satisfiability problem is NP-complete. In this subsection we use Corollary 59 to prove distribution learning hardness results for specific function classes $\mathcal{C}$ for which there are invertible Levin reductions from CIRCUIT-SAT to $\mathcal{C}$.

Fact 60. *There is a polynomial time invertible Levin reduction from CIRCUIT-SAT to 3-CNF-SAT.*

As a corollary, we have the following result.

Corollary 61. *If Assumption 1 holds true, then there exists an absolute constant $c > 0$ such that the class 3-CNF is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for distribution learning.*

Corollary 61 is interesting in light of the well known fact that the class of all 3-CNF formulas is efficiently PAC learnable from uniform random examples (in fact under any distribution). The next theorem can be proven by following the standard reduction from CIRCUIT-SAT to SUBSET-SUM.

Theorem 62. *If Assumption 1 holds true, then there exists an absolute constant $c > 0$ such that $\mathcal{C} = \{$all intersections of two halfspaces over n Boolean variables$\}$ is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for distribution learning.*

F.1.4 A hardness result where the satisfiability problem is in P. The last two hardness results have been for classes $\mathcal{C}$ of NP-complete languages. As Theorem 56 requires a reduction from CIRCUIT-SAT to $\mathcal{C}$, this theorem cannot be directly used to prove hardness for classes $\mathcal{C}$ which are not NP-hard. We next give an extension of Theorem 56 which can apply to classes $\mathcal{C}$ for which the satisfiability problem is in P. Using this result we will show hardness of distribution learning for MONOTONE-2-CNF-SAT. We begin by defining by a notion of invertible one-many reductions that we will need.

Definition 63. *CIRCUIT-SAT is said to have an η-*almost invertible one-many reduction to a function class *$\mathcal{C}$ if the following conditions hold:*

- *There is a polynomial time computable function f such that given an instance Φ of CIRCUIT-SAT (i.e. Φ is a satisfiable circuit), $\Psi = f(\Phi)$ is an instance of $\mathcal{C}$ (i.e. $\Psi \in \mathcal{C}$ and Ψ is satisfiable).*
- *Fix any instance Φ of CIRCUIT-SAT and let $\mathcal{A} = \Psi^{-1}(1)$ denote the set of satisfying assignments of Ψ. Then $\mathcal{A}$ can be partitioned into sets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $|\mathcal{A}_2|/|\mathcal{A}| \leq \eta$ and there is an efficiently computable function $g : \mathcal{A}_1 \to \Phi^{-1}(1)$ such that $g(x)$ is a satisfying assignment of Φ for every $x \in \mathcal{A}_1$.*

- *For every y which is a satisfying assignment of Φ, the number of pre-images of y under g is exactly the same, and the uniform distribution over $g^{-1}(y)$ is polynomial time samplable.*

We next state the following simple claim which will be helpful later.

Claim 64. *Suppose there is an η-almost invertible one-many reduction from CIRCUIT-SAT to $\mathcal{C}$. Let f and g be the functions from Definition 63. Let Φ be an instance of CIRCUIT-SAT and let $\Psi = f(\Phi)$ be the corresponding instance of $\mathcal{C}$. Define distributions D_1 and D_2 as follows :*

- *A draw from D_1 is obtained by choosing y uniformly at random from $\Phi^{-1}(1)$ and then outputting z uniformly at random from $g^{-1}(y)$.*
- *A draw from D_2 is obtained by choosing z' uniformly at random from $\Psi^{-1}(1)$.*

Then we have $d_{TV}(D_1, D_2) \leq \eta$.

Proof. This is an immediate consequence of the fact that D_1 is uniform over the set $\mathcal{A}_1$ while D_2 is uniform over the set $\mathcal{A}$ (from Definition 63). □

We next have the following extension of Corollary 59.

Theorem 65. *Suppose that Assumption 1 holds true. Then if $\mathcal{C}$ is a function class such that there is an η-almost invertible one-many reduction (for $\eta = 2^{-\Omega(n)}$) from CIRCUIT-SAT to $\mathcal{C}$, then $\mathcal{C}$ is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for distribution learning for some absolute constant $c > 0$.*

Proof. The proof is similar to the proof of Corollary 59. Assume towards a contradiction that there is a distribution learning algorithm A for $\mathcal{C}$ which runs in time t_1 such that with probability $1 - \kappa_2$, the output distribution is κ_1-close to the target distribution. (We will set t_1, κ_1 and κ_2 later to 2^{n^c}, $1 - 2^{-n^c}$ and $1 - 2^{-n^c}$ respectively.)

Let (G, S, V) be the RMA-secure signature scheme constructed in Theorem 57. Note that (G, S, V) is a (T, ϵ)-RMA secure signature scheme where $T = 2^{n^\delta}$, $\epsilon = 1/T$ and $|\mathcal{M}| = 2^{n^\mu}$ for constant $\delta, \mu > 0$. Let (pk, sk) be a choice of key pair. We will us A to contradict the security of (G, S, V). Towards this, consider the function $V_{pk} : \mathcal{M} \times \{0,1\}^* \to \{0,1\}$ defined as $V_{pk}(m, \sigma) = V(m, pk, \sigma)$. Clearly, V_{pk} is an instance of CIRCUIT-SAT. Consider the η-invertible one-many reduction from CIRCUIT-SAT to $\mathcal{C}$. Let α and β have the same meaning as in Definition 63. Let $\Psi = \alpha(V_{pk})$ and let $\mathcal{A}$, $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same meaning as in Definition 63. The adversary receives message-signature pairs $(m_1, \sigma_1) \ldots (m_{t_1}, \sigma_{t_1})$ where $m_1, \ldots, m_{t_1}$ are chosen independently at random from $\mathcal{M}$. For any i, (m_i, σ_i) is a satisfying assignment of V_{pk}. By definition, in time $t_2 = t_1 \cdot \mathrm{poly}(n)$, the adversary can sample $(z_1, \ldots, z_{t_1})$ such that $z_1, \ldots, z_{t_1}$ are independent and $z_i \sim \mathcal{U}_{\beta^{-1}(m_i, \sigma_i)}$. Note that this means that each z_i is an independent sample from $\mathcal{A}_1$ and $|z_i| = \mathrm{poly}(n)$. Note that $(z_1, \ldots, z_{t_1})$ is a t_1-fold product distribution such that if D' denotes the distribution of z_i, then by Claim 64, $d_{TV}(D', \mathcal{U}_{\Psi^{-1}(1)}) \leq \eta$. Hence, if D is the distribution of $(z_1, \ldots, z_{t_1})$, then $d_{TV}(D, \mathcal{U}^t_{\Psi^{-1}(1)}) \leq t_1 \eta$.

Hence, the adversary can now run A_{rec} on the samples $z_1, \ldots, z_{t_1}$ and as long as $1 - \kappa_2 - t_1\eta \geq (1 - \kappa_2)/2$, succeeds in producing a sampler with probability $(1 - \kappa_2)/2$ whose output distribution (call it Z) is κ_1 close to the distribution $\mathcal{U}_{\Psi^{-1}(1)}$. Note that as $\eta = 2^{-\Omega(n)}$, for any $c > 0$, $t_1 = 2^{n^c}$ and $\kappa_2 = 1 - 2^{-n^c}$ satisfies this condition. Hence, we get that $d_{TV}(Z, D') \leq \kappa_1 + \eta$. Now, observe that

$$\mathbf{Pr}_{\rho \in D'}[\beta(\rho) = (m, \sigma) \text{ and } m \neq m_i] = 1 - \frac{t_1}{|\mathcal{M}|}.$$

The above uses the fact that every element in the range of β has the same number of pre-images. This of course implies that

$$\mathbf{Pr}_{\rho \in Z}[\beta(\rho) = (m, \sigma) \text{ and } m \neq m_i] \geq 1 - \frac{t_1}{|\mathcal{M}|} - (\kappa_1 + \eta).$$

Again as long as $\kappa_1 \leq 1 - 2(\eta + t_1/|\mathcal{M}|)$, the adversary succeeds in getting a valid message signature pair (m, σ) with $m \neq m_i$ for any $1 \leq i \leq t_1$ with probability $(1 - \kappa_1)/2$. Again, we can ensure $\kappa_1 \leq 1 - 2(\eta + t_1/|\mathcal{M}|)$ by choosing c sufficiently small compared to μ. The total probability of success is $(1 - \kappa_1)(1 - \kappa_2)/4$ and the total running time is $t_1(\text{poly}(n)) + \text{poly}(n)$. Again if c is sufficiently small compared to μ and δ, then the total running time is at most $t_1(\text{poly}(n)) + \text{poly}(n) < T$ and the success probability is at least $(1 - \kappa_1)(1 - \kappa_2)/4 > \epsilon$, resulting in a contradiction. □

We now demonstrate a polynomial time η-invertible one-many reduction from CIRCUIT-SAT to MONOTONE-2-CNF-SAT for $\eta = 2^{-\Omega(n)}$. The reduction uses the "blow-up" idea used to prove hardness of approximate counting for MONOTONE-2-CNF-SAT in [JVV86]. We will closely follow the instantiation of this technique in [Wat12].

Lemma 66. *There is a polynomial time η-almost invertible one-many reduction from CIRCUIT-SAT to MONOTONE-2-CNF-SAT where $\eta = 2^{-\Omega(n)}$.*

Proof. We begin by noting the following simple fact.

Fact 67. *If there is a polynomial time invertible Levin reduction from CIRCUIT-SAT to a class $\mathcal{C}_1$ and an η-almost invertible one-many reduction from $\mathcal{C}_1$ to $\mathcal{C}_2$, then there is a polynomial time η-almost invertible one-many reduction from CIRCUIT-SAT to $\mathcal{C}_2$.*

Since there is an invertible Levin reduction from CIRCUIT-SAT to 3-CNF-SAT, by virtue of Fact 67, it suffices to demonstrate a polynomial time η-almost invertible one-many reduction from 3-CNF-SAT to MONOTONE-2-CNF-SAT. To do this, we first construct an instance of VERTEX-COVER from the 3-CNF-SAT instance. Let $\Phi = \bigwedge_{i=1}^{m} \Phi_i$ be the instance of 3-CNF-SAT. Construct an instance of VERTEX-COVER by introducing seven vertices for each clause Φ_i (one corresponding to every satisfying assignment of Φ_i). Now, put an edge between any two vertices of this graph if the corresponding assignments to the variables of Φ conflict on some variable. We call this graph G. We observe the following properties of this graph :

- G has exactly $7m$ vertices.
- Every vertex cover of G has size at least $6m$.
- There is an efficiently computable and invertible injection ℓ between the satisfying assignments of Φ and the vertex covers of G of size $6m$. To get the vertex cover corresponding to a satisfying assignment, for every clause Φ_i, include the six vertices in the vertex cover which conflict with the satisfying assignment.

We next do the blow-up construction. We create a new graph G' by replacing every vertex of G with a cloud of $10m$ vertices, and for every edge in G we create a complete bipartite graph between the corresponding clouds in G'. Clearly, the size of the graph G' is polynomial in the size of the 3-CNF-SAT formula. We define a map g_1 between vertex covers of G' and vertex covers of G as follows : Let S' be a vertex cover of G'. We define the set $S = g_1(S')$ in the following way. For every vertex v in the graph G, if all the vertices in the corresponding cloud in G' are in S', then include $v \in S$, else do not include v in S. It is easy to observe that g_1 maps vertex covers of G' to vertex covers of G. It is also easy to observe that a vertex cover of G of size s has $(2^{10m} - 1)^{7m-s}$ pre-images under g_1.

Now, observe that we can construct a MONOTONE-2-CNF-SAT formula Ψ which has a variable corresponding to every vertex in G' and every subset S' of G' corresponds to a truth assignment $y_{S'}$ to Ψ such that $\Psi(y_{S'}) = 1$ if and only if S' is a vertex cover of G'. Because of this correspondence, we can construct a map g_1' which maps satisfying assignments of Ψ to vertex covers of G. Further, a vertex cover of size s in graph G has $(2^{10m}-1)^{7m-s}$ pre-images under g_1'. Since the total number of vertex covers of G of size s is at most $\binom{7m}{s}$, the total number of satisfying assignments of Ψ which map to vertex covers of G of size more than $6m$ can be bounded by :

$$\sum_{s=6m+1}^{7m} \binom{7m}{s} \cdot (2^{10m}-1)^{7m-s} \leq m \cdot \binom{7m}{6m+1} \cdot (2^{10m}-1)^{m-1} \leq (2^{10m}-1)^m \cdot \frac{2^{7m}}{2^{10m}-1}$$

On the other hand, since Φ has at least one satisfying assignment, hence G has at least one vertex cover of size $6m$ and hence the total number of satisfying assignments of Ψ which map to vertex covers of G of size $6m$ is at least $(2^{10m}-1)^m$. Thus, if we let $\mathcal{A}$ denote the set of satisfying assignments of Ψ and $\mathcal{A}_1$ be the set of satisfying assignment of Ψ which map to vertex covers of G of size exactly $6m$ (under g_1), then $|\mathcal{A}_1|/|\mathcal{A}| \geq 1 - 2^{-\Omega(n)}$. Next, notice that we can define the map g mapping $\mathcal{A}_1$ to the satisfying assignments of Φ in the following manner : $g(x) = \ell^{-1}(g_1(x))$. It is easy to see that this map satisfies all the requirements of the map g from Definition 63 which concludes the proof. □

Combining Lemma 66 with Theorem 65, we have the following corollary.

Corollary 68. *If Assumption 1 holds true, then* MONOTONE-2-CNF-SAT *is* $(2^{n^c}, 1-2^{-n^c}, 1-2^{-n^c})$ *hard for distribution learning for some absolute constant* $c > 0$.

As a consequence of the above result, we also get hardness of distribution learning of degree-2 *polynomial threshold functions (PTFs)*; these are functions of the form $\mathrm{sign}(q(x))$ where $q(x)$ is a degree-2 multilinear polynomial over $\{0,1\}^n$.

Corollary 69. *If Assumption 1 holds true, then the class of all* n*-variable degree-2 polynomial threshold functions is* $(2^{n^c}, 1-2^{-n^c}, 1-2^{-n^c})$ *hard for distribution learning for some absolute constant* $c > 0$.

Proof. This follows immediately from the fact that every monotone 2-CNF formula can be expressed as a degree-2 PTF. To see this, note that if $\Phi = \bigwedge_{i=1}^{m}(x_{i1} \vee x_{i2})$ where each x_{ij} is a 0/1 variable, then $\Phi(x)$ is true if and only if $\sum_{i=1}^{m} x_{i1} + x_{i2} - x_{i1} \cdot x_{i2} \geq m$. This finishes the proof. □