From Local Pseudorandom Generators to Hardness of Learning

Amit Daniely AMIT.DANIELY@MAIL.HUJI.AC.IL
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel and Google Research Tel-Aviv

Gal Vardi GAL.VARDI@WEIZMANN.AC.IL
Weizmann Institute of Science, Israel

Editors: Mikhail Belkin and Samory Kpotufe

Abstract

We prove hardness-of-learning results under a well-studied assumption on the existence of local pseudorandom generators. As we show, this assumption allows us to surpass the current state of the art, and prove hardness of various basic problems, with no hardness results to date.

Our results include: hardness of learning shallow ReLU neural networks under the Gaussian distribution and other distributions; hardness of learning intersections of $\omega(1)$ halfspaces, DNF formulas with $\omega(1)$ terms, and ReLU networks with $\omega(1)$ hidden neurons; hardness of weakly learning deterministic finite automata under the uniform distribution; hardness of weakly learning depth-3 Boolean circuits under the uniform distribution, as well as distribution-specific hardness results for learning DNF formulas and intersections of halfspaces. We also establish lower bounds on the complexity of learning intersections of a constant number of halfspaces, and ReLU networks with a constant number of hidden neurons. Moreover, our results imply the hardness of virtually all improper PAC-learning problems (both distribution-free and distribution-specific) that were previously shown hard under other assumptions.

1. Introduction

The computational complexity of PAC learning has been extensively studied over the past decades. Nevertheless, for many learning problems there is still a large gap between the complexity of the best known algorithms and the hardness results. The situation is even worse for *distribution-specific learning*, namely, where the inputs are drawn from some known distribution (e.g., the uniform or the normal distribution). Since there are very few distribution-specific hardness results, the status of most basic learning problems with respect to natural distributions is wide open.

The main obstacle for achieving hardness results for learning problems, is the ability of a learning algorithm to return a hypothesis which does not belong to the considered hypothesis class (such an algorithm is called *improper learner*). This flexibility makes it very difficult to apply reductions from NP-hard problems, and unless we face a dramatic breakthrough in complexity theory, it seems unlikely that hardness of improper learning can be established on standard complexity assumptions (see Applebaum et al. (2008); Daniely et al. (2014)). Indeed, all currently known lower bounds are based on assumptions from cryptography or average-case hardness.

In this work, we consider hardness of learning under assumptions on the existence of *local pseudorandom generators (PRG)* with polynomial stretch. This type of assumptions was extensively studied in the last two decades. Under such assumptions we extend the current state of the art, and establish new hardness results for several hypothesis classes, for both distribution-free and distribution-specific learning. Our results apply to fundamental classes, such as DNFs, Boolean circuits, intersections of halfspaces, neural networks and automata. Most of the results are based on the mere assumption that *some* local PRG with polynomial stretch exists. The only exceptions are our lower bounds for intersections of a constant number of halfspaces, and neural networks with a constant number of neurons, that are based on a stronger assumption, regarding a specific candidate for such a PRG, that was suggested by Applebaum and Lovett (2016). Below we discuss our results and related work.

DNFs and Boolean circuits. Learning polynomial-size DNF formulas has been a major effort in computational learning theory. The best known upper bound for (distribution-free) learning of polynomial-size DNF formulas over n variables is $2^{\tilde{O}(n^{1/3})}$, due to Klivans and Servedio (2001). Already in Valiant's seminal paper (Valiant, 1984), it is shown that for every constant q, DNF formulas with q terms can be learned efficiently. Hardness of improperly learning DNF formulas is implied by Applebaum et al. (2010) under a combination of two assumptions: the first is related to the planted dense subgraph problem in hypergraphs, and the second is related to local PRGs. Daniely and Shalev-Shwartz (2016) showed hardness of improperly learning DNF formulas with $q(n) = \omega(\log(n))$ terms, under a common assumption, namely, that refuting a random K-SAT formula is hard. We improve this lower bound, and show hardness of learning DNF formulas with $q(n) = \omega(1)$ terms.

Linial et al. (1993) gave a quasi-polynomial ($O(n^{\mathrm{polylog}(n)})$) upper bound for learning constant-depth Boolean circuits (AC^0) on the uniform distribution. Their result was later improved to a slightly better quasi-polynomial bound (Boppana, 1997; Håstad, 2001). Learning AC^0 in quasi-polynomial time under other restricted distributions was studied in, e.g., Furst et al. (1991); Blais et al. (2010). In Kharitonov (1993) it is shown, under a relatively strong assumption on the complexity of factoring random Blum integers, that learning depth-d circuits on the uniform distribution is hard, where d is an unspecified sufficiently large constant. Applebaum and Raykov (2016) showed, under an assumption on a specific candidate for Goldreich's PRG (based on the XOR-MAJ predicate), that learning depth-3 Boolean circuits under the uniform distribution is hard. We prove distribution-specific hardness of improperly learning Boolean circuits of depth-2 (namely, DNFs) and depth-3. For DNF formulas with n^{ϵ} terms, we show hardness of learning on a distribution where each component is drawn i.i.d. from a Bernoulli distribution (which is not uniform). For depth-3 Boolean circuits, we show hardness of weak learning on the uniform distribution (recall that we only assume here the existence of *some* local PRG, rather than a specific candidate).

Intersections of halfspaces. Learning intersections of halfspaces is also a fundamental problem in learning theory. Klivans and Sherstov (2006) showed, assuming the hardness of the shortest vector problem, that improper learning of intersections of n^{ϵ} halfspaces for a constant $\epsilon > 0$, is hard. The hardness result from Daniely and Shalev-Shwartz (2016) for learning DNF formulas with $\omega(\log(n))$ terms, implies hardness of learning intersections of $\omega(\log(n))$ halfspaces, since every DNF formula with $q(n)$ terms can be realized by the complement of an intersection of $q(n)$ halfspaces. Our result on hardness of learning DNF formulas with $\omega(1)$ terms implies hardness of learning intersections of $\omega(1)$ halfspaces, and thus improves the bound from Daniely and Shalev-Shwartz (2016). Learning intersections of halfspaces under some restricted distributions has been studied in, e.g., Baum (1990); Blum and Kannan (1997); Vempala (1997); Klivans et al. (2004, 2009). Our distribution-specific hardness result for DNFs implies a first distribution-specific hardness result for improperly learning intersections of n^{ϵ} halfspaces.

Efficient algorithms for (distribution-free) learning intersections of k halfspaces are not known even for a constant k, and even for $k = 2$. Klivans et al. (2004) showed an algorithm for distribution-free learning k weight-w halfspaces on the hypercube in time $n^{O(k \log(k) \log(w))}$, where the weight of a halfspace is the sum of the absolute values of its components. We study distribution-free improper learning of a constant number of halfspaces, namely, where the number k of halfspaces is independent of n. We show (under our stronger assumption regarding a specific candidate for a local PRG) a $n^{\beta k}$ lower bound. More formally, we show that there is an absolute constant $\beta > 0$ (independent of k, n), such that learning intersections of k halfspaces on the hypercube within a constant error requires time $\Omega(n^{\beta k})$. Also, a conjecture due to Applebaum and Lovett (2016) implies that our lower bound holds for, e.g., $\beta = \frac{1}{11}$. This is the first lower bound for improperly learning intersections of a constant number of halfspaces.

Neural networks. Hardness of improperly learning neural networks (with respect to the square loss) follows from hardness of learning intersection of halfspaces. Hence, the results from Klivans and Sherstov (2006) and Daniely and Shalev-Shwartz (2016) imply hardness of improperly learning depth-2 neural networks with n^{ϵ} and $\omega(\log(n))$ hidden neurons (respectively). Daniely and Vardi (2020) showed, under the assumption that refuting a random K-SAT formula is hard, that improperly learning depth-2 neural networks is hard already if its weights are drawn from some "natural" distribution or satisfy some "natural" properties. While hardness of proper learning is implied by hardness of improper learning, there are some recent works that show hardness of properly learning depth-2 networks under more standard assumptions (cf. Goel et al. (2020c)).

Our hardness results for DNFs and intersections of halfspaces imply new hardness results for learning neural networks. We show hardness of improperly learning depth-2 neural networks with $\omega(1)$ hidden neurons and the ReLU activation function, with respect to the square loss. Thus, we improve the $\omega(\log(n))$ lower bound implied by Daniely and Shalev-Shwartz (2016). Moreover, the lower bound implied by Daniely and Shalev-Shwartz (2016) requires an activation function also in the output neuron, while our lower bound does not. For depth-2 networks with a constant number k of hidden neuron, namely, where the number of hidden neurons is independent of n, we show (under our stronger assumption regarding a specific candidate for a local PRG) a $\Omega(n^{\beta k})$ lower bound, where β is a constant independent of n, k. This is the first lower bound for improperly learning neural networks with a constant number of hidden neurons.

Due to the empirical success of neural networks, there has been much effort to understand under what assumptions neural networks may be learned efficiently. This effort includes making assumptions on the input distribution (Li and Yuan, 2017; Brutzkus and Globerson, 2017; Du et al., 2017a,b; Du and Goel, 2018; Goel et al., 2018), the network's weights (Arora et al., 2014; Das et al., 2019; Agarwal et al., 2020; Goel and Klivans, 2017), or both (Janzamin et al., 2015; Tian, 2017; Bakshi et al., 2019). Hence, distribution-specific learning of neural networks is a central problem. Several works in recent years have shown hardness of distribution-specific learning shallow neural networks using gradient-descent or statistical query (SQ) algorithms (Shamir, 2018; Song et al., 2017; Vempala and Wilmes, 2019; Goel et al., 2020a; Diakonikolas et al., 2020b). We note that while the SQ framework captures the gradient-descent algorithm, it does not capture, for example, stochastic gradient-descent (SGD), which examines training points individually (see a discussion in Goel et al. (2020a)). Distribution-specific hardness of learning a single ReLU neuron in the agnostic setting was studied in Goel et al. (2019, 2020b); Diakonikolas et al. (2020a).

We show hardness of improper distribution-specific learning of depth-2 and depth-3 ReLU neural networks with respect to the square loss. First, our distribution-specific hardness results for Boolean circuits, imply hardness of learning depth-2 networks on a distribution where each component is drawn i.i.d. from a (non-uniform) Bernoulli distribution, and depth-3 networks on the uniform distribution on the hypercube. More importantly, we also show hardness of improperly learning depth-3 networks on the standard Gaussian distribution.

Automata. Deterministic finite automata are an elementary computational model, and their learnability is a classical problem in learning theory. An efficient algorithm due to Angluin (1987) is known for learning deterministic automata with membership and equivalence queries, and was extensively studied over the last decades. Improper learning of deterministic automata with n^ϵ states is known to be harder than breaking the RSA cryptosystem, factoring Blum integers and detecting quadratic residues (Kearns and Valiant, 1994). It is also harder than refuting a random K-SAT formula (Daniely and Shalev-Shwartz, 2016). The question of whether deterministic automata are learnable on the uniform distribution was posed by Pitt (1989) over 30 years ago, and remained open (cf. Fish and Reyzin (2017); Michaliszyn and Otop (2019)). We solve this problem, by showing hardness of weakly learning deterministic automata on the uniform distribution over the hypercube. This is the first distribution-specific hardness result for improperly learning automata.

Other classes. Our lower bound for learning DNF formulas with $\omega(1)$ terms implies hardness of learning $\omega(1)$-sparse polynomial threshold functions over $\{0,1\}^n$, where a q-sparse polynomial has at most q monomials with non-zero coefficients. It improves the lower bound from Daniely and Shalev-Shwartz (2016) for learning $\omega(\log(n))$-sparse polynomial threshold functions. Also, we show hardness of learning $\omega(1)$-sparse $GF(2)$ polynomials over $\{0,1\}^n$. Subexponential-time upper bounds for these problems are given in Hellerstein and Servedio (2007).

Finally, our lower bound for learning DNFs implies hardness of agnostically learning conjunctions, halfspaces and parities. These problems are already known to be hard under other assumptions (Feldman et al., 2006; Daniely, 2016; Blum et al., 2003; Daniely and Shalev-Shwartz, 2016).

A summary of our contribution. Below we summarize our main contributions:

- Hardness of learning DNF formulas with $\omega(1)$ terms.
- Distribution-specific hardness of learning DNFs and weakly learning depth-3 Boolean circuits.
- Hardness of learning intersections of $\omega(1)$ halfspaces.
- Distribution-specific hardness of learning intersections of halfspaces on the hypercube.
- $\Omega(n^{\beta k})$-time lower bound for learning intersections of a constant number k of halfspaces, where β is an absolute constant.
- Hardness of learning depth-2 neural networks with $\omega(1)$ hidden neuron.
- $\Omega(n^{\beta k})$-time lower bound for learning depth-2 neural networks with a constant number k of hidden neurons, where β is an absolute constant.
- Distribution-specific hardness of learning depth-2 and 3 neural networks on the hypercube.
- Distribution-specific hardness of learning depth-3 neural networks on the Gaussian distribution.
- Distribution-specific hardness of weakly learning deterministic automata on the hypercube.

- Hardness of learning $\omega(1)$-sparse polynomial threshold functions and $\omega(1)$-sparse $GF(2)$ polynomials over $\{0,1\}^n$.
- Hardness of agnostically learning conjunctions, halfspaces and parities (these problems are already known to be hard under other assumptions).
- Our results imply the hardness of virtually all[1] improper PAC-learning problems (both distribution-free and distribution-specific) that were previously shown hard (under various complexity assumptions). Moreover, our technique is simple, and we believe that it might be useful for showing hardness of more learning problems in the future.

Our paper is structured as follows: In Section 2 we provide necessary notations and definitions, and discuss our assumptions. The results are stated in Section 3. We informally sketch our proof technique in Section 4, with all formal proofs deferred to the appendix.

2. Preliminaries

2.1. Notations

We use bold-faced letters to denote vectors, e.g., $\mathbf{x} = (x_1, \ldots, x_d)$. For a vector $\mathbf{x}$ and a sequence $S = (i_1, \ldots, i_k)$ of k indices, we let $\mathbf{x}_S = (x_{i_1}, \ldots, x_{i_k})$, i.e., the restriction of $\mathbf{x}$ to the indices S. We denote by $\mathbb{1}(\cdot)$ the indicator function, for example $\mathbb{1}(t \geq 5)$ equals 1 if $t \geq 5$ and 0 otherwise. For an integer $d \geq 1$ we denote $[d] = \{1, \ldots, d\}$. The majority predicate $\mathrm{MAJ}_k : \{0,1\}^k \to \{0,1\}$ is defined by $\mathrm{MAJ}_k(\mathbf{x}) = 1$ iff $\sum_{i \in [k]} x_i > \frac{k}{2}$. We denote $\mathrm{XOR}_k : \{0,1\}^k \to \{0,1\}$ where $\mathrm{XOR}_k(\mathbf{x}) = x_1 \oplus \ldots \oplus x_k$. For $m \in \mathbb{R}$ we let $\mathrm{sign}(m) = 1$ if $m > 0$ and $\mathrm{sign}(m) = 0$ otherwise.

2.2. Local pseudorandom generators

An (n, m, k)-hypergraph is a hypergraph over n vertices $[n]$ with m hyperedges $S_1, \ldots, S_m$, each of cardinality k. Each hyperedge $S = (i_1, \ldots, i_k)$ is ordered, and all the k members of a hyperedge are distinct. We let $\mathcal{G}_{n,m,k}$ be the distribution over such hypergraphs in which a hypergraph is chosen by picking each hyperedge uniformly and independently at random among all the possible $n \cdot (n-1) \cdot \ldots \cdot (n-k+1)$ ordered hyperedges. Let $P : \{0,1\}^k \to \{0,1\}$ be a predicate, and let G be a (n, m, k)-hypergraph. We call *Goldreich's pseudorandom generator (PRG)* (Goldreich, 2000) the function $f_{P,G} : \{0,1\}^n \to \{0,1\}^m$ such that for $\mathbf{x} \in \{0,1\}^n$, we have $f_{P,G}(\mathbf{x}) = (P(\mathbf{x}_{S_1}), \ldots, P(\mathbf{x}_{S_m}))$. The integer k is called the *locality* of the PRG. If k is a constant then the PRG and the predicate P are called *local*. We say that the PRG has *polynomial stretch* if $m = n^s$ for some constant $s > 1$. Let $\mathcal{F}_{P,n,m}$ be the collection of functions $f_{P,G}$ where G is an (n, m, k)-hypergraph. We sample a function from $\mathcal{F}_{P,n,m}$ by choosing a random hypergraph G from $\mathcal{G}_{n,m,k}$.

We denote by $G \overset{R}{\leftarrow} \mathcal{G}_{n,m,k}$ the operation of sampling a hypergraph G from $\mathcal{G}_{n,m,k}$, and by $\mathbf{x} \overset{R}{\leftarrow} \{0,1\}^n$ the operation of sampling $\mathbf{x}$ from the uniform distribution on $\{0,1\}^n$. We say that $\mathcal{F}_{P,n,m}$ is ε-pseudorandom generator (ε-PRG) if for every polynomial-time probabilistic algorithm

[1] It does not imply the hardness result from Daniely and Vardi (2020) for learning depth-2 neural networks whose weights are drawn from some "natural" distribution.

$\mathcal{A}$ the *distinguishing advantage*

$$\left|\Pr_{G\xleftarrow{R}\mathcal{G}_{n,m,k},\mathbf{x}\xleftarrow{R}\{0,1\}^n}[\mathcal{A}(G,f_{P,G}(\mathbf{x}))=1]-\Pr_{G\xleftarrow{R}\mathcal{G}_{n,m,k},\mathbf{y}\xleftarrow{R}\{0,1\}^m}[\mathcal{A}(G,\mathbf{y})=1]\right|$$

is at most ε. Thus, the distinguisher $\mathcal{A}$ is given a random hypergraph G and a string $\mathbf{y}\in\{0,1\}^m$, and its goal is to distinguish between the case where $\mathbf{y}$ is chosen at random, and the case where $\mathbf{y}$ is a random image of $f_{P,G}$. Our main assumption is that local PRGs with polynomial stretch and constant distinguishing advantage exist:

Assumption 1 *For every constant $s>1$, there exists a constant k and a predicate $P:\{0,1\}^k\to\{0,1\}$, such that $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG.*

Note that we assume constant distinguishing advantage. In the literature, a requirement of negligible distinguishing advantage[2] is often considered (cf. Applebaum and Lovett (2016); Applebaum (2016); Couteau et al. (2018)). Thus, our requirement from the PRG is weaker.

Local PRGs have been extensively studied in the last two decades. In particular, local PRGs with polynomial stretch have shown to have remarkable applications, such as secure-computation with constant computational overhead (Ishai et al., 2008; Applebaum et al., 2017), and general-purpose obfuscation based on constant degree multilinear maps (cf. Lin (2016); Lin and Vaikuntanathan (2016)). A significant evidence for Assumption 1 was shown in Applebaum (2013). He showed that Assumption 1 follows from the assumption that for every constant $s>1$, there exists a *sensitive* local predicate[3] P such that $\mathcal{F}_{P,n,n^s}$ is one-way. This is a variant of Goldreich's one-wayness assumption (Goldreich, 2000).

In light of Assumption 1, an important question is which local predicates are secure. O'Donnell and Witmer (2014) showed that a property called *resiliency* yields pseudorandomness against attacks which are based on a large class of semidefinite programs. Feldman et al. (2015) showed that resiliency also ensures pseudorandomness against a wide family of statistical algorithms. Applebaum and Lovett (2016) showed that predicates with high resiliency and high *rational degree* are secure against two classes of distinguishing attacks: *linear attacks* and *algebraic attacks*. These classes include all known attacks against PRGs. Furthermore, they suggested the following predicate as a candidate for local PRG with polynomial stretch: $\text{XOR-MAJ}_{a,b}(\mathbf{z})=(z_1\oplus\ldots\oplus z_a)\oplus\text{MAJ}_b(z_{a+1},\ldots,z_{a+b})$. By their conjecture, for every constant $s>1$ and constants $a\geq 5s$,$b>36s$ the predicate $P=\text{XOR-MAJ}_{a,b}$ is such that the collection $\mathcal{F}_{P,n,n^s}$ is PRG with negligible distinguishing advantage. This predicate has high resiliency and rational degree, and is secured against all known attacks. Its security has been studied also in Couteau et al. (2018); Méaux et al. (2019); Applebaum and Raykov (2016). We make a somewhat weaker assumption:

Assumption 2 *There is a constant $\alpha>0$, such that for every constant $s>1$ there is a constant l such that for the predicate $P=\text{XOR-MAJ}_{\lceil\alpha s\rceil,l}$ the collection $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG.*

Our results on learning intersections of a constant number of halfspaces, and on leaning neural networks with a constant number of hidden neurons, rely on Assumption 2. All other results rely

[2] More formally, that for $1-o_n(1)$ fraction of the hypergraphs, the distinguisher has no more than negligible advantage.

[3] A predicate is sensitive if at least one coordinate i has full influence, i.e., flipping the value of the i-th variable always changes the output.

on Assumption 1. Thus, most of our results assume the existence of a local PRG with polynomial stretch, and do not rely on a specific candidate.

In our assumptions we consider local PRGs that are secure against polynomial-time algorithms. Hence, the hardness results in this paper rule out polynomial-time learning algorithms. We note that our results can be improved by strengthening the assumptions, e.g., by assuming that the local PRGs are secure against some quasi-polynomial time algorithms.

Prior works on the relation between Goldreich's PRG and hardness of learning. First, Goldreich's PRG are closely related to CSP refutation, and there have been many works on the relation between CSP refutation and hardness of learning (e.g., Daniely et al. (2014); Daniely and Shalev-Shwartz (2016); Daniely (2016); Vadhan (2017); Kothari and Livni (2018); Daniely and Vardi (2020)). Moreover, some applications of variants of Goldreich's assumption and local PRGs for hardness of learning are shown in Applebaum et al. (2010); Applebaum and Raykov (2016); Nanashima (2020).

2.3. PAC learning

A *hypothesis class* $\mathcal{H}$ is a series of collections of functions $\mathcal{H}_n \subset \mathcal{Y}^{\mathcal{X}_n}, \; n = 1, 2, \ldots$. We often abuse notation and identify $\mathcal{H}$ with $\mathcal{H}_n$. The domain sets $\mathcal{X}_n$ we consider are $\{0,1\}^n$ or $\mathbb{R}^n$, and the label sets $\mathcal{Y}$ we consider are $\{0,1\}$ or $\mathbb{R}$. Let $\mathcal{Z}_n = \mathcal{X}_n \times \mathcal{Y}$ and let $\mathcal{D}_n$ be a distribution on $\mathcal{Z}_n$. A *loss function* is a mapping $\ell : \mathcal{H}_n \times \mathcal{Z}_n \to \mathbb{R}_+$. We consider the following loss functions. The *0-1 loss* is $\ell_{0-1}(h, (\mathbf{x}, y)) = \mathbb{1}(h(\mathbf{x}) \neq y)$. For $\mathcal{Y} = \mathbb{R}$, the *square loss* is $\ell_{\text{sq}}(h, (\mathbf{x}, y)) = (h(\mathbf{x}) - y)^2$. The *error* of $h : \mathcal{X}_n \to \mathcal{Y}$ is $L_{\mathcal{D}_n}(h) = \mathbb{E}_{\mathbf{z} \in \mathcal{D}_n}[\ell(h, \mathbf{z})]$. Note that for the 0-1 loss we have $L_{\mathcal{D}_n}(h) = \Pr_{(\mathbf{x},y)\sim\mathcal{D}_n}[h(\mathbf{x}) \neq y]$. For a class $\mathcal{H}_n$, we let $L_{\mathcal{D}_n}(\mathcal{H}_n) = \min_{h \in \mathcal{H}_n} L_{\mathcal{D}_n}(h)$. We say that $\mathcal{D}_n$ is *realizable* by h (respectively $\mathcal{H}_n$) if $L_{\mathcal{D}_n}(h) = 0$ (respectively $L_{\mathcal{D}_n}(\mathcal{H}_n) = 0$).

A *learning algorithm* $\mathcal{L}$ is given $\epsilon, \delta \in (0,1)$, as well as an oracle access to examples from an unknown distribution $\mathcal{D}$ on $\mathcal{Z}_n$. It should output a (description of) hypothesis $h : \mathcal{X}_n \to \mathcal{Y}$. We say that $\mathcal{L}$ *(PAC) learns* $\mathcal{H}$, if for every realizable $\mathcal{D}$, with probability at least $1-\delta$, the algorithm $\mathcal{L}$ outputs a hypothesis with error at most ϵ. We say that $\mathcal{L}$ *agnostically learns* $\mathcal{H}$, if for every $\mathcal{D}$, with probability at least $1-\delta$, the algorithm $\mathcal{L}$ outputs a hypothesis with error at most $L_{\mathcal{D}}(\mathcal{H}) + \epsilon$. Note that by these definitions, $\mathcal{L}$ should succeed for every realizable distribution $\mathcal{D}$ (in the former definition) or for every distribution $\mathcal{D}$ (in the later definition). Hence, this setting is called *distribution-free learning*. We now consider *distribution-specific learning*, namely, where the marginal distribution of $\mathcal{D}$ on $\mathcal{X}_n$ is fixed. Let $\mathcal{D}_{\mathcal{X}}$ be a distribution on $\mathcal{X}_n$. We say that $\mathcal{L}$ *learns* $\mathcal{H}$ *on* $\mathcal{D}_{\mathcal{X}}$, if for every realizable $\mathcal{D}$ whose marginal distribution on $\mathcal{X}_n$ is $\mathcal{D}_{\mathcal{X}}$, with probability at least $1-\delta$, the algorithm $\mathcal{L}$ outputs a hypothesis with error at most ϵ.

When the error is defined with respect to the 0-1 loss, we also consider *weak learning*: For $\gamma > 0$ we say that $\mathcal{L}$ *γ-weakly learns* $\mathcal{H}$, if for every realizable $\mathcal{D}$, the algorithm $\mathcal{L}$ is given $\delta \in (0,1)$, and outputs w.p. at least $1-\delta$ a hypothesis with error at most $\frac{1}{2} - \gamma$. We say that $\mathcal{L}$ *γ-weakly learns* $\mathcal{H}$ *on* $\mathcal{D}_{\mathcal{X}}$, if for every realizable $\mathcal{D}$ whose marginal distribution on $\mathcal{X}_n$ is $\mathcal{D}_{\mathcal{X}}$, the algorithm $\mathcal{L}$ is given $\delta \in (0,1)$, and outputs w.p. at least $1-\delta$ a hypothesis with error at most $\frac{1}{2} - \gamma$. Thus, when γ is small, the returned hypothesis needs to be at least slightly better than a random guess.

We say that $\mathcal{L}$ is *efficient* if it runs in time $\text{poly}(n, 1/\epsilon, 1/\delta)$ (or $\text{poly}(n, 1/\delta)$, for weak learning), and outputs a hypothesis that can be evaluated in time $\text{poly}(n, 1/\epsilon, 1/\delta)$ (respectively, $\text{poly}(n, 1/\delta)$). Finally, $\mathcal{L}$ is *proper* if it always outputs a hypothesis in $\mathcal{H}$. Otherwise, we say that $\mathcal{L}$ is *improper*.

By boosting results (Schapire, 1989; Freund, 1995), if there is an efficient algorithm that $\frac{1}{n^c}$-weakly learns $\mathcal{H}$ for some constant $c > 0$, then there is also an efficient improper algorithm that learns $\mathcal{H}$. Hence, in the distribution-free setting, hardness of improper learning implies hardness of improper weak learning. These boosting arguments do not apply to the distribution-specific setting.

2.4. Neural networks and automata

We consider feedforward neural networks, computing functions from $\mathbb{R}^n$ to $\mathbb{R}$. The network is composed of layers of neurons, where each neuron computes a function $\mathbf{x} \mapsto \sigma(\mathbf{w}^\top \mathbf{x} + b)$, where $\mathbf{w}$ is a weight vector, b is a bias term and $\sigma : \mathbb{R} \mapsto \mathbb{R}$ is a non-linear activation function. We focus on the ReLU activation, i.e., $\sigma(z) = [z]_+ = \max\{0, z\}$. For a matrix $W = (\mathbf{w}_1, \ldots, \mathbf{w}_d)$, we let $\sigma(W^\top \mathbf{x} + \mathbf{b})$ be a shorthand for $\left(\sigma(\mathbf{w}_1^\top \mathbf{x} + b_1), \ldots, \sigma(\mathbf{w}_d^\top \mathbf{x} + b_d)\right)$, and define a layer of d neurons as $\mathbf{x} \mapsto \sigma(W^\top \mathbf{x} + \mathbf{b})$. By denoting the output of the i-th layer as O_i, we define a network recursively by $O_{i+1} = \sigma(W_{i+1}^\top O_i + \mathbf{b}_{i+1})$. The *weights vector* of the j-th neuron in the i-th layer is the j-th column of W_i. The *fan-in* of a neuron is the number of non-zero entries in its weights vector. We define the *depth* of the network as the number of layers. Unless stated otherwise, the output neuron also has a ReLU activation function. A neuron which is not an input or output neuron is called a *hidden neuron*. We sometimes consider neural networks with multiple outputs.

A *deterministic finite automaton* (DFA) is a tuple $A = \langle \Sigma, Q, q_0, \delta, F \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is a transition function, and $F \subseteq Q$ are the final states. Given a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_l \in \Sigma^*$, the *run* of A on w is the sequence $r = q_0, q_1, \ldots, q_l$ such that $q_{i+1} = \delta(q_i, \sigma_{i+1})$ for all $i \geq 0$. The run r is accepting if $q_l \in F$, and A *accepts* w iff r is accepting. We sometimes use the notation $A(w) = 1$ (resp., $A(w) = 0$) to indicate that A accepts (resp., rejects) w. The *size* of A is the number of its states.

3. results

3.1. DNFs and Boolean circuits

In the following theorem we show distribution-free hardness for DNF formulas with $\omega(1)$ terms, and distribution-specific hardness for DNF formulas with n^ϵ terms (see proof in Appendix A.1).

Theorem 3 *Under Assumption 1, for every $q(n) = \omega(1)$, there is no efficient algorithm that learns DNF formulas with n variables and $q(n)$ terms. Moreover, for every constant $\epsilon > 0$, there is no efficient algorithm that learns DNF formulas with n^ϵ terms, on a distribution where each component is drawn i.i.d. from a (non-uniform) Bernoulli distribution.*

Theorem 3 gives distribution-specific hardness for learning DNF formulas, namely, depth-2 Boolean circuits, where the input distribution is such that the components are i.i.d. copies from a Bernoulli distribution. For depth-3 Boolean circuits we show hardness of weak learning, where the input distribution is uniform on the hypercube (see proof in Appendix A.2).

Theorem 4 *Under Assumption 1, for every constants $\gamma, \epsilon > 0$, there is no efficient algorithm that γ-weakly learns depth-3 Boolean circuits of size n^ϵ on the uniform distribution over $\{0, 1\}^n$.*

3.2. Intersections of halfspaces

Any function realized by a DNF formula with $q(n)$ terms can be also realized by the complement of an intersection of $q(n)$ halfspaces. Hence, Theorem 3 implies the following corollary.

Corollary 5 *Under Assumption 1, for every $q(n) = \omega(1)$, there is no efficient algorithm that learns intersections of $q(n)$ halfspaces over $\{0,1\}^n$. Moreover, for every constant $\epsilon > 0$, there is no efficient algorithm that learns intersections of n^ϵ halfspaces, on a distribution where each component is drawn i.i.d. from a (non-uniform) Bernoulli distribution.*

We now consider intersections of a constant number k of halfspaces (i.e., k is independent of n), and show a $\Omega(n^{\beta k})$ lower bound (see proof in Appendix A.3).

Theorem 6 *Let $\mathcal{H} \subseteq \{0,1\}^{(\{0,1\}^n)}$ be the functions expressible by intersections of k halfspaces, where k is a constant independent of n. Let $\mathcal{L}$ be a learning algorithm, that for every $\mathcal{H}$-realizable distribution, returns with probability at least $\frac{3}{4}$ a hypothesis with error at most $\frac{1}{10}$. Then, under Assumption 2, there is a universal constant $\beta > 0$ (independent of k, n) such that the time-complexity of $\mathcal{L}$ is $\Omega(n^{\beta k})$.*

Remark 7 *Applebaum and Lovett (2016) conjectured that Assumption 2 holds for $\alpha \geq 5$. It implies that Theorem 6 holds for, e.g., $\beta = \frac{1}{11}$.*

3.3. Neural networks

We consider neural networks with the ReLU activation function. Since neural networks are real-valued, we consider here the square loss rather than the 0-1 loss. Our results hold for networks where the norms of the weights of each neuron are bounded by some $\text{poly}(n)$. By a simple scaling trick (i.e., by increasing the input dimension), it follows that for every constant $\epsilon > 0$, the results also hold for networks where the norms of the weights of every neuron are bounded by n^ϵ.

From Theorems 3 and 4, it is not hard to show the following theorems (see proofs in Appendices A.4 and A.5).

Theorem 8 *Under Assumption 1, we have:*

1. *For every $q(n) = \omega(1)$, there is no efficient algorithm that learns depth-2 neural networks with $q(n)$ hidden neurons, and no activation function in the output neuron, where the input distribution is supported on $\{0,1\}^n$.*
2. *For every constant $\epsilon > 0$, there is no efficient algorithm that learns depth-2 neural networks with n^ϵ hidden neurons, on a distribution where each component is drawn i.i.d. from a (non-uniform) Bernoulli distribution.*
3. *For every constant $\epsilon > 0$, there is no efficient algorithm that learns depth-3 neural networks with n^ϵ hidden neurons, on the uniform distribution over $\{0,1\}^n$.*

Theorem 9 *Let $\mathcal{H} \subseteq \mathbb{R}^{(\{0,1\}^n)}$ be the functions expressible by depth-2 neural networks with k hidden neurons and no activation function in the output neuron, where k is a constant independent of n. Let $\mathcal{L}$ be a learning algorithm, that for every $\mathcal{H}$-realizable distribution, returns with probability at least $\frac{3}{4}$ a hypothesis with error at most $\frac{1}{10}$. Then, under Assumption 2, there is a universal constant $\beta > 0$ (independent of k, n) such that the time-complexity of $\mathcal{L}$ is $\Omega(n^{\beta k})$.*

Remark 10 *Applebaum and Lovett (2016) conjectured that Assumption 2 holds for $\alpha \geq 5$. It implies that Theorem 9 holds for, e.g., $\beta = \frac{1}{21}$.*

We now consider continuous input distributions. We focus here on the normal distribution, but our result can be extended to other continuous distributions (see proof in Appendix A.6).

Theorem 11 *Under Assumption 1, for every constant $\epsilon > 0$, there is no efficient algorithm that learns depth-3 neural networks with n^ϵ hidden neurons on the standard Gaussian distribution.*

3.4. Automata

We show hardness of weakly-learning DFAs on the uniform distribution (see proof in Appendix A.7).

Theorem 12 *Under Assumption 1, for every constants $c, \epsilon > 0$, there is no efficient algorithm that $\frac{1}{n^c}$-weakly learns DFAs of size n^ϵ, on the uniform distribution over $\{0,1\}^n$.*

3.5. Other classes

Our results imply lower bounds for some additional classes. We start with hardness of learning $\omega(1)$-sparse polynomial threshold functions on $\{0,1\}^n$. Recall that a q-sparse polynomial has at most q monomials with non-zero coefficients.

Corollary 13 *Under Assumption 1, for every $q(n) = \omega(1)$, there is no efficient algorithm that learns $q(n)$-sparse polynomial threshold functions over $\{0,1\}^n$.*

Corollary 13 follows from Theorem 3 since any function realized by a DNF formula with $q(n)$ terms can be also realized by a polynomial threshold function over $\{0,1\}^n$ with $q(n)$ monomials. We also consider $\omega(1)$-sparse $GF(2)$ polynomials over $\{0,1\}^n$. Such a polynomial is simply a sum modulo 2 of $\omega(1)$ monomials (see proof in Appendix A.8).

Theorem 14 *Under Assumption 1, for every $q(n) = \omega(1)$, there is no efficient algorithm that learns $q(n)$-sparse $GF(2)$ polynomials over $\{0,1\}^n$.*

Finally, the following corollaries follow from the hardness of learning DNFs (see Daniely and Shalev-Shwartz (2016)). We note that these results are already known under other assumptions (Feldman et al., 2006; Daniely, 2016; Blum et al., 2003; Daniely and Shalev-Shwartz, 2016).

Corollary 15 *Under Assumption 1, no efficient algorithm agnostically learns conjunctions.*

Corollary 16 *Under Assumption 1, no efficient algorithm agnostically learns halfspaces.*

Corollary 17 *Under Assumption 1, no efficient algorithm agnostically learns parities.*

4. Our technique

4.1. Hardness under Assumption 1

We first describe the proof ideas for the case of DNFs. Then, we explain how to apply the method to other classes.

4.1.1. DISTRIBUTION-FREE HARDNESS FOR DNFS

We describe the main ideas in the proof of the first part of Theorem 3. We encode a hyperedge $S = (i_1, \ldots, i_k)$ by $\mathbf{z}^S \in \{0,1\}^{kn}$, where $\mathbf{z}^S$ is the concatenation of k vectors in $\{0,1\}^n$, such that the j-th vector has 0 in the i_j-th component and 1 elsewhere. Thus, $\mathbf{z}^S$ consists of k size-n slices, each encodes a member of S. For a predicate $P : \{0,1\}^k \to \{0,1\}$ and $\mathbf{x} \in \{0,1\}^n$, let $P_{\mathbf{x}} : \{0,1\}^{kn} \to \{0,1\}$ be a function such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = P(\mathbf{x}_S)$.

Let $s > 1$ be a constant. By Assumption 1, there exists a constant k and a predicate $P : \{0,1\}^k \to \{0,1\}$, such that $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG. Assume that there is an efficient algorithm $\mathcal{L}$ that learns DNF formulas with n' variables and $q(n') = \omega_{n'}(1)$ terms. We will use the algorithm $\mathcal{L}$ to obtain an algorithm $\mathcal{A}$ with distinguishing advantage greater than $\frac{1}{3}$ and thus reach a contradiction.

Given a sequence $(S_1, y_1), \ldots, (S_{n^s}, y_{n^s})$, where $S_1, \ldots, S_{n^s}$ are i.i.d. random hyperedges, the algorithm $\mathcal{A}$ needs to distinguish whether $\mathbf{y} = (y_1, \ldots, y_{n^s})$ is random or that we have $\mathbf{y} = (P(\mathbf{x}_{S_1}), \ldots, P(\mathbf{x}_{S_{n^s}})) = (P_{\mathbf{x}}(\mathbf{z}^{S_1}), \ldots, P_{\mathbf{x}}(\mathbf{z}^{S_{n^s}}))$ for a random $\mathbf{x} \in \{0,1\}^n$. We denote $\mathcal{S} = ((\mathbf{z}^{S_1}, y_1), \ldots, (\mathbf{z}^{S_{n^s}}, y_{n^s}))$.

We show that for every predicate $P : \{0,1\}^k \to \{0,1\}$ and $\mathbf{x} \in \{0,1\}^n$, there is a DNF formula ψ over $\{0,1\}^{kn}$ with at most 2^k terms, such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = \psi(\mathbf{z}^S)$. The formula ψ is such that for each satisfying assignment $\mathbf{b} \in \{0,1\}^k$ of P there is a term in ψ that checks whether $\mathbf{x}_S = \mathbf{b}$. Thus, $\psi(\mathbf{z}^S) = P(\mathbf{x}_S) = P_{\mathbf{x}}(\mathbf{z}^S)$. Therefore, if $\mathcal{S}$ is pseudorandom then it is realizable by a DNF formula with at most 2^k terms. Since k is constant, then for a sufficiently large n we have $2^k \leq q(kn)$. Hence, the algorithm $\mathcal{A}$ can distinguish whether $\mathcal{S}$ is pseudorandom or random as follows. It partitions $\mathcal{S}$ to a training set and a test set, and runs $\mathcal{L}$ on the training set (we show that if s is a sufficiently large constant then we can choose a training set large enough for $\mathcal{L}$). Let h be the hypothesis returned by $\mathcal{L}$. If $\mathcal{S}$ is pseudorandom then we show that h will have small error on the test set, and if $\mathcal{S}$ is random then h will have large error on the test set. Hence, $\mathcal{A}$ can distinguish between the cases.

4.1.2. DISTRIBUTION-SPECIFIC HARDNESS FOR DNFS

We turn to describe the main ideas in the proof of the second part of Theorem 3. We show how to distinguish whether the sequence $\mathcal{S}$ from the previous paragraph is random or pseudorandom, given access to a distribution-specific learning algorithm. Let $\mathcal{L}'$ be an efficient algorithm that learns DNF formulas with n' variables and at most $(n')^\epsilon$ terms, on a distribution $\mathcal{D}'$ such that each component is drawn i.i.d. from a Bernoulli distribution where the probability of 1 is p. Assume that p is such that the probability that a random $\mathbf{z} \sim \mathcal{D}'$ is an encoding of a hyperedge is not too small.

We show an algorithm $\mathcal{A}'$ such that given a sequence $\mathcal{S} = ((\mathbf{z}^{S_1}, y_1), \ldots, (\mathbf{z}^{S_{n^s}}, y_{n^s}))$, it distinguishes whether $\mathcal{S}$ is pseudorandom or random. Here, the algorithm $\mathcal{A}'$ has access to $\mathcal{L}'$, which is guaranteed to learn successfully only if the input distribution is $\mathcal{D}'$. Note that for every $i \in [n^s]$ the vector $\mathbf{z}^{S_i}$ is an encoding of a random hyperedge, and does not have the distribution $\mathcal{D}'$. Therefore, the algorithm $\mathcal{A}'$ will run $\mathcal{L}'$ with an examples oracle that essentially works as follows: In the i-th call to the oracle, it chooses $\mathbf{z}_i \sim \mathcal{D}'$. If $\mathbf{z}_i$ is an encoding of a hyperedge then the oracle returns $(\mathbf{z}^{S_i}, y_i)$, and otherwise it returns $(\mathbf{z}_i, 1)$. Namely, if $\mathbf{z}_i$ is an encoding of a hyperedge then we replace it by $\mathbf{z}^{S_i}$, which is an encoding of a random hyperedge, and hence we do not change the distribution. Thus, the oracle uses $\mathcal{S}$ as a source for random encodings.

Let h' be the hypothesis returned by $\mathcal{L}'$. The algorithm $\mathcal{A}'$ now checks h' on a test set created by the examples oracle (we show that if s is large enough then we can create sufficiently large training and test sets). If $\mathcal{S}$ is pseudorandom then we show that the examples returned by the oracle are realized by some DNF formula with an appropriate number of terms, and hence h' will have small error on the test set. Note that this DNF formula needs to return 1 if the input is not an encoding of a hyperedge, and to return $P_{\mathbf{x}}(\mathbf{z}^S)$ if the input is the encoding $\mathbf{z}^S$ of a hyperedge S. If $\mathcal{S}$ is random then h' will be incorrect in roughly half of the examples in the test set that correspond to pairs $(\mathbf{z}^{S_i}, y_i)$ from $\mathcal{S}$, and hence will have larger error on the test set. Therefore, $\mathcal{A}'$ can distinguish between the cases.

4.1.3. DISTRIBUTION-SPECIFIC HARDNESS FOR OTHER CLASSES

While each of the distribution-specific hardness results involves some unique challenges, all the proofs roughly follow a similar method to the one used in the case of DNFs.

Let $\mathcal{H}$ be the hypothesis class for which we want to show hardness and let $\mathcal{D}$ be the input distribution. Assuming that there is an algorithm $\mathcal{L}$ that learns (or weakly learns) $\mathcal{H}$ on the distribution $\mathcal{D}$, we show an algorithm $\mathcal{A}$ that distinguishes whether a sequence $\mathcal{S} = ((S_1, y_1), \dots, (S_{n^s}, y_{n^s}))$ is pseudorandom or random. The algorithm $\mathcal{A}$ runs $\mathcal{L}$ with an examples oracle that can be implemented efficiently, and returns examples $(\mathbf{z}, y)$ such that $\mathbf{z} \sim \mathcal{D}$. The oracle uses $\mathcal{S}$ as a source for labeled random hyperedges, and with sufficiently high probability the returned example $(\mathbf{z}, y)$ corresponds to some (S_i, y_i) in $\mathcal{S}$. Let h be the hypothesis returned by $\mathcal{L}$. If $\mathcal{S}$ is pseudorandom, then we show that the examples returned by the oracle are realizable by $\mathcal{H}$, and hence h has a small error on a test set created by the oracle. If $\mathcal{S}$ is random then h is incorrect in roughly half of the examples in the test set that correspond to pairs (S_i, y_i) from $\mathcal{S}$, and hence has larger error on the test set. Hence, $\mathcal{A}$ can distinguish between the cases.

The implementation details of the above method are different for every class $\mathcal{H}$ that we consider. Thus, in each proof we use different encodings of hyperedges and a different examples oracle. Moreover, in each proof we need to show that the examples returned by the oracle are realizable, and hence we construct a function $h \in \mathcal{H}$ that labels correctly all examples returned by the oracle.

4.2. Lower bounds under Assumption 2

We explain how to apply Assumption 2 in the case intersections of a constant number of halfspaces (we sketch here a proof for Theorem 6). The case of neural networks with a constant number of hidden neurons (Theorem 9) is similar.

It is not hard to show that Assumption 2 implies that there is a constant $\beta > 0$ such that for every constant k, there is l such that for the predicate $P = \text{XOR-MAJ}_{k,l}$ the collection $\mathcal{F}_{P,n,n^{2.1\beta k}}$ is $\frac{1}{3}$-PRG. Assume that there is an efficient algorithm $\mathcal{L}$ that learns intersections of k halfspaces over $\{0,1\}^{\tilde{n}}$. Assume that $\mathcal{L}$ uses a sample of size $m(\tilde{n}) = \tilde{n}^{\beta k}$ and returns with probability at least $\frac{3}{4}$ a hypothesis with error at most $\frac{1}{10}$. We will use the algorithm $\mathcal{L}$ to establish an algorithm $\mathcal{A}$ with distinguishing advantage greater than $\frac{1}{3}$ and thus reach a contradiction. It implies that an efficient algorithm that learns intersections of k halfspaces over $\{0,1\}^{\tilde{n}}$ must use a sample of size greater than $\tilde{n}^{\beta k}$, and therefore runs in time $\Omega(\tilde{n}^{\beta k})$.

Let $\tilde{n} = \frac{(2n)(2n-1)}{2} + 2n + 1$. For $\mathbf{z} \in \{0,1\}^{2n}$, we denote by $\tilde{\mathbf{z}} \in \{0,1\}^{\tilde{n}}$ the vector of all monomials over $\mathbf{z}$ of degree at most 2. We call $\tilde{\mathbf{z}}$ the *monomials encoding* of $\mathbf{z}$. We encode a hyperedge $S = (i_1, \dots, i_{k+l})$ by $\mathbf{z}^S \in \{0,1\}^{2n}$, where $\mathbf{z}^S$ is the concatenation of 2 vectors in $\{0,1\}^n$, such that the first vector has 1-bits in the indices $i_1, \dots, i_k$ and 0 elsewhere, and the second vector has 1-bits in the indices $i_{k+1}, \dots, i_{k+l}$ and 0 elsewhere. We denote by $\tilde{\mathbf{z}}^S$ the monomials encoding of $\mathbf{z}^S$. For $\mathbf{x} \in \{0,1\}^n$, let $P_{\mathbf{x}} : \{0,1\}^{\tilde{n}} \to \{0,1\}$ be a function such that for every hyperedge S we have $P_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = P(\mathbf{x}_S)$.

We show that for every $\mathbf{x} \in \{0,1\}^n$, there is a function $g : \{0,1\}^{\tilde{n}} \to \{0,1\}$ that can be expressed by an intersection of k halfspaces, such that for every hyperedge S we have $g(\tilde{\mathbf{z}}^S) = P_{\mathbf{x}}(\tilde{\mathbf{z}}^S)$. Intuitively, an intersection of k halfspaces over $\tilde{\mathbf{z}}^S$ is an intersection of k degree-2 polynomial threshold functions over $\mathbf{z}^S$, and we show that each degree-2 polynomial threshold function is powerful enough to handle the case where the number of 1-bits in the XOR part of P is i, for some $i \in [k]$. With this claim at hand, we establish the algorithm $\mathcal{A}$ as follows.

Given a sequence $(S_1, y_1), \ldots, (S_{n^{2.1\beta k}}, y_{n^{2.1\beta k}})$, where $S_1, \ldots, S_{n^{2.1\beta k}}$ are i.i.d. random hyperedges, the algorithm $\mathcal{A}$ needs to distinguish whether $\mathbf{y} = (y_1, \ldots, y_{n^{2.1\beta k}})$ is random, or that $\mathbf{y} = (P(\mathbf{x}_{S_1}), \ldots, P(\mathbf{x}_{S_{n^{2.1\beta k}}})) = (P_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_1}), \ldots, P_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_{n^{2.1\beta k}}}))$ for a random $\mathbf{x} \in \{0,1\}^n$. Let $\mathcal{S} = ((\tilde{\mathbf{z}}^{S_1}, y_1), \ldots, (\tilde{\mathbf{z}}^{S_{n^{2.1\beta k}}}, y_{n^{2.1\beta k}}))$. The algorithm $\mathcal{A}$ learns a function $h : \{0,1\}^{\tilde{n}} \to \{0,1\}$ by running $\mathcal{L}$ with an examples oracle that in each call returns the next example from $\mathcal{S}$. Recall that $\mathcal{L}$ uses at most $m(\tilde{n}) = \tilde{n}^{\beta k}$ examples, and hence $\mathcal{S}$ contains at least $n^{2.1\beta k} - \tilde{n}^{\beta k} \geq n^{2.1\beta k} - (2n)^{2\beta k} \geq \ln(n)$ examples that $\mathcal{L}$ cannot view (for a sufficiently large n). We use these examples as a test set. If $\mathcal{S}$ is pseudorandom then it is realizable by an intersection of k halfspaces, and thus w.h.p. h has small error on the test set. If $\mathcal{S}$ is random then h has error of roughly $\frac{1}{2}$ on the test set. Hence, $\mathcal{A}$ can distinguish between the cases.

Acknowledgments

We thank Benny Applebaum and anonymous reviewers for their valuable comments. This research is partially supported by ISF grant 2258/19.

References

Naman Agarwal, Pranjal Awasthi, and Satyen Kale. A deep conditioning treatment of neural networks. *arXiv preprint arXiv:2002.01523*, 2020.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

B. Applebaum, B. Barak, and D. Xiao. On basing lower-bounds for learning on worst-case assumptions. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 211–220. IEEE, 2008.

Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM Journal on Computing*, 42(5):2008–2037, 2013.

Benny Applebaum. Cryptographic hardness of random local functions. *Computational complexity*, 25(3):667–722, 2016.

Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 1087–1100, 2016.

Benny Applebaum and Pavel Raykov. Fast pseudorandom functions based on expander graphs. In *Theory of Cryptography Conference*, pages 27–56. Springer, 2016.

Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 171–180, 2010.

Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In *Annual International Cryptology Conference*, pages 223–254. Springer, 2017.

Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *International Conference on Machine Learning*, pages 584–592, 2014.

Ainesh Bakshi, Rajesh Jayaram, and David P Woodruff. Learning two layer rectified neural networks in polynomial time. In *Conference on Learning Theory*, pages 195–268. PMLR, 2019.

Eric B Baum. A polynomial time algorithm that learns two hidden unit nets. *Neural Computation*, 2(4):510–522, 1990.

Eric Blais, Ryan O'Donnell, and Karl Wimmer. Polynomial regression under arbitrary product distributions. *Machine learning*, 80(2-3):273–294, 2010.

Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

Avrim L Blum and Ravindran Kannan. Learning an intersection of a constant number of halfspaces over a uniform distribution. *Journal of Computer and System Sciences*, 54(2):371–380, 1997.

Ravi B Boppana. The average sensitivity of bounded-depth circuits. *Information processing letters*, 63(5):257–261, 1997.

Alon Brutzkus and Amir Globerson. Globally optimal gradient descent for a convnet with gaussian inputs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 605–614. JMLR. org, 2017.

Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of goldreich's pseudorandom generator. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 96–124. Springer, 2018.

Amit Daniely. Complexity theoretic limitations on learning halfspaces. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 105–117. ACM, 2016.

Amit Daniely and Shai Shalev-Shwartz. Complexity theoretic limitations on learning dnf's. In *Conference on Learning Theory*, pages 815–830, 2016.

Amit Daniely and Gal Vardi. Hardness of learning neural networks with natural weights. *arXiv preprint arXiv:2006.03177*, 2020.

Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. From average case complexity to improper learning complexity. In *STOC*, 2014.

Abhimanyu Das, Sreenivas Gollapudi, Ravi Kumar, and Rina Panigrahy. On the learnability of deep random networks. *arXiv preprint arXiv:1904.03866*, 2019.

Ilias Diakonikolas, Daniel Kane, and Nikos Zarifis. Near-optimal sq lower bounds for agnostically learning halfspaces and relus under gaussian marginals. *Advances in Neural Information Processing Systems*, 33, 2020a.

Ilias Diakonikolas, Daniel M Kane, Vasilis Kontonis, and Nikos Zarifis. Algorithms and sq lower bounds for pac learning one-hidden-layer relu networks. *arXiv preprint arXiv:2006.12476*, 2020b.

Simon S Du and Surbhi Goel. Improved learning of one-hidden-layer convolutional neural networks with overlaps. *arXiv preprint arXiv:1805.07798*, 2018.

Simon S Du, Jason D Lee, and Yuandong Tian. When is a convolutional filter easy to learn? *arXiv preprint arXiv:1709.06129*, 2017a.

Simon S Du, Jason D Lee, Yuandong Tian, Barnabas Poczos, and Aarti Singh. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima. *arXiv preprint arXiv:1712.00779*, 2017b.

Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. New results for learning noisy parities and halfspaces. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 563–574. IEEE, 2006.

Vitaly Feldman, Will Perkins, and Santosh Vempala. On the complexity of random satisfiability problems with planted solutions. In *STOC*, 2015.

Benjamin Fish and Lev Reyzin. Open problem: Meeting times for learning random automata. In *Conference on Learning Theory*, pages 8–11, 2017.

Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121 (2):256–285, 1995.

Merrick L Furst, Jeffrey C Jackson, and Sean W Smith. Improved learning of ac0 functions. In *COLT*, volume 91, pages 317–325, 1991.

Surbhi Goel and Adam Klivans. Learning neural networks with two nonlinear layers in polynomial time. *arXiv preprint arXiv:1709.06010*, 2017.

Surbhi Goel, Adam Klivans, and Raghu Meka. Learning one convolutional layer with overlapping patches. *arXiv preprint arXiv:1802.02547*, 2018.

Surbhi Goel, Sushrut Karmalkar, and Adam Klivans. Time/accuracy tradeoffs for learning a relu with respect to gaussian marginals. In *Advances in Neural Information Processing Systems*, pages 8584–8593, 2019.

Surbhi Goel, Aravind Gollakota, Zhihan Jin, Sushrut Karmalkar, and Adam Klivans. Superpolynomial lower bounds for learning one-layer neural networks using gradient descent. *arXiv preprint arXiv:2006.12011*, 2020a.

Surbhi Goel, Aravind Gollakota, and Adam Klivans. Statistical-query lower bounds via functional gradients. *Advances in Neural Information Processing Systems*, 33, 2020b.

Surbhi Goel, Adam Klivans, Pasin Manurangsi, and Daniel Reichman. Tight hardness results for training depth-2 relu networks. *arXiv preprint arXiv:2011.13550*, 2020c.

Oded Goldreich. Candidate one-way functions based on expander graphs. *IACR Cryptol. ePrint Arch.*, 2000:63, 2000.

Johan Håstad. A slight sharpening of lmn. *Journal of Computer and System Sciences*, 63(3):498–508, 2001.

Lisa Hellerstein and Rocco A Servedio. On pac learning algorithms for rich boolean function classes. *Theoretical Computer Science*, 384(1):66–76, 2007.

Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 433–442, 2008.

Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. *arXiv preprint arXiv:1506.08473*, 2015.

Michael Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the Association for Computing Machinery*, 41(1):67–95, January 1994.

Michael Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 372–381. ACM, 1993.

Adam R Klivans and Rocco Servedio. Learning dnf in time $2^{O(n^{1/3})}$. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 258–265. ACM, 2001.

Adam R. Klivans and Alexander A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. In *FOCS*, 2006.

Adam R Klivans, Ryan O'Donnell, and Rocco A Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer and System Sciences*, 68(4):808–840, 2004.

Adam R Klivans, Philip M Long, and Alex K Tang. Baum's algorithm learns intersections of halfspaces with respect to log-concave distributions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 588–600. Springer, 2009.

Pravesh K Kothari and Roi Livni. Improper learning by refuting. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation. In *Advances in Neural Information Processing Systems*, pages 597–607, 2017.

Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 28–57. Springer, 2016.

Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 11–20. IEEE, 2016.

Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *Journal of the Association for Computing Machinery*, 40(3):607–620, July 1993.

Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. Improved filter permutators: Combining symmetric encryption design, boolean functions, low complexity cryptography, and homomorphic encryption, for private delegation of computations. *IACR Cryptol. ePrint Arch.*, 2019:483, 2019.

Jakub Michaliszyn and Jan Otop. Approximate learning of limit-average automata. *arXiv preprint arXiv:1906.11104*, 2019.

Mikito Nanashima. Extending learnability to auxiliary-input cryptographic primitives and meta-pac learning. In *Conference on Learning Theory*, pages 2998–3029. PMLR, 2020.

Ryan O'Donnell and David Witmer. Goldreich's prg: Evidence for near-optimal polynomial stretch. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 1–12. IEEE, 2014.

Leonard Pitt. Inductive inference, dfas, and computational complexity. In *International Workshop on Analogical and Inductive Inference*, pages 18–44. Springer, 1989.

R.E. Schapire. The strength of weak learnability. In *FOCS*, pages 28–33, October 1989.

Ohad Shamir. Distribution-specific hardness of learning neural networks. *The Journal of Machine Learning Research*, 19(1):1135–1163, 2018.

Le Song, Santosh Vempala, John Wilmes, and Bo Xie. On the complexity of learning neural networks. In *Advances in neural information processing systems*, pages 5514–5522, 2017.

Yuandong Tian. An analytical formula of population gradient for two-layered relu network and its applications in convergence and critical point analysis. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3404–3413. JMLR. org, 2017.

Salil Vadhan. On learning vs. refutation. In *Conference on Learning Theory*, pages 1835–1848. PMLR, 2017.

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

Santosh Vempala. A random sampling based algorithm for learning the intersection of half-spaces. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 508–513. IEEE, 1997.

Santosh Vempala and John Wilmes. Gradient descent for one-hidden-layer neural networks: Polynomial convergence and sq lower bounds. In *Conference on Learning Theory*, pages 3115–3117. PMLR, 2019.

Appendix A. Proofs

A.1. Proof of Theorem 3

We encode a hyperedge $S = (i_1, \ldots, i_k)$ by $\mathbf{z}^S \in \{0,1\}^{kn}$, where $\mathbf{z}^S$ is the concatenation of k vectors in $\{0,1\}^n$, such that the j-th vector has 0 in the i_j-th component and 1 elsewhere. Thus, $\mathbf{z}^S$ consists of k size-n slices, each encodes a member of S. For $\mathbf{z} \in \{0,1\}^{kn}$, $i \in [k]$ and $j \in [n]$, we denote $z_{i,j} = z_{(i-1)\cdot n+j}$. That is, $z_{i,j}$ is the j-th component in the i-th slice in $\mathbf{z}$. For a predicate $P : \{0,1\}^k \to \{0,1\}$ and $\mathbf{x} \in \{0,1\}^n$, let $P_{\mathbf{x}} : \{0,1\}^{kn} \to \{0,1\}$ be a function such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = P(\mathbf{x}_S)$.

Lemma 18 *For every predicate $P : \{0,1\}^k \to \{0,1\}$ and $\mathbf{x} \in \{0,1\}^n$, there is a DNF formula ψ over $\{0,1\}^{kn}$ with at most 2^k terms, such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = \psi(\mathbf{z}^S)$.*

Proof We denote by $\mathcal{B} \subseteq \{0,1\}^k$ the set of satisfying assignments of P. Note that the size of $\mathcal{B}$ is at most 2^k. Consider the following DNF formula over $\{0,1\}^{kn}$:

$$\psi(\mathbf{z}) = \bigvee_{\mathbf{b}\in\mathcal{B}} \bigwedge_{j\in[k]} \bigwedge_{\{l : x_l \neq b_j\}} z_{j,l} \ .$$

For a hyperedge $S = (i_1, \ldots, i_k)$, we have

$$\begin{aligned}
\psi(\mathbf{z}^S) = 1 &\iff \exists \mathbf{b} \in \mathcal{B}\ \forall j \in [k]\ \forall x_l \neq b_j,\ z^S_{j,l} = 1 \\
&\iff \exists \mathbf{b} \in \mathcal{B}\ \forall j \in [k]\ \forall x_l \neq b_j,\ i_j \neq l \\
&\iff \exists \mathbf{b} \in \mathcal{B}\ \forall j \in [k],\ x_{i_j} = b_j \\
&\iff \exists \mathbf{b} \in \mathcal{B},\ \mathbf{x}_S = \mathbf{b} \\
&\iff P(\mathbf{x}_S) = 1 \\
&\iff P_{\mathbf{x}}(\mathbf{z}^S) = 1 \ .
\end{aligned}$$

■

In the following lemma we prove the first part of the theorem.

Lemma 19 *Under Assumption 1, there is no efficient algorithm that learns DNF formulas with n variables and $\omega_n(1)$ terms.*

Proof Assume that there is an efficient algorithm $\mathcal{L}$ that learns DNF formulas with n' variables and $q(n') = \omega_{n'}(1)$ terms. Let $m(n')$ be a polynomial such that $\mathcal{L}$ uses a sample of size at most $m(n')$ and returns with probability at least $\frac{3}{4}$ a hypothesis with error at most $\frac{1}{10}$. Let $s > 1$ be a constant such that $n^s \geq m(n\log(n)) + n$ for every sufficiently large n. By Assumption 1, there exists a constant k and a predicate $P : \{0,1\}^k \to \{0,1\}$, such that $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG. We will show an algorithm $\mathcal{A}$ with distinguishing advantage greater than $\frac{1}{3}$ and thus reach a contradiction.

Given a sequence $(S_1, y_1), \ldots, (S_{n^s}, y_{n^s})$, where $S_1, \ldots, S_{n^s}$ are i.i.d. random hyperedges, the algorithm $\mathcal{A}$ needs to distinguish whether $\mathbf{y} = (y_1, \ldots, y_{n^s})$ is random or that we have $\mathbf{y} = (P(\mathbf{x}_{S_1}), \ldots, P(\mathbf{x}_{S_{n^s}})) = (P_{\mathbf{x}}(\mathbf{z}^{S_1}), \ldots, P_{\mathbf{x}}(\mathbf{z}^{S_{n^s}}))$ for a random $\mathbf{x} \in \{0,1\}^n$. We denote $\mathcal{S} = ((\mathbf{z}^{S_1}, y_1), \ldots, (\mathbf{z}^{S_{n^s}}, y_{n^s}))$. Let $\mathcal{D}$ be a distribution on $\{0,1\}^{kn}$ such that $\mathbf{z} \sim \mathcal{D}$ is an encoding of a random hyperedge. Note that each $\mathbf{z}^{S_i}$ from $\mathcal{S}$ is drawn i.i.d. from $\mathcal{D}$.

We use the efficient algorithm $\mathcal{L}$ in order to obtain distinguishing advantage greater than $\frac{1}{3}$ as follows. The algorithm $\mathcal{A}$ learns a hypothesis $h : \{0,1\}^{kn} \to \{0,1\}$ by running $\mathcal{L}$ with an examples oracle that in each call returns the next example from $\mathcal{S}$. Recall that $\mathcal{L}$ uses at most $m(kn) \leq m(n \log(n))$ examples (assuming n is large enough), and hence $\mathcal{S}$ contains at least n examples that $\mathcal{L}$ cannot view. Denote the indices of these examples by $I = \{m(n \log(n)) + 1, \ldots, m(n \log(n)) + n\}$, and the examples by $\mathcal{S}_I = \{(\mathbf{z}^{S_i}, y_i)\}_{i \in I}$. Let $\ell_I(h) = \frac{1}{|I|} \sum_{i \in I} \mathbb{1}(h(\mathbf{z}^{S_i}) \neq y_i)$. Now, if $\ell_I(h) \leq \frac{2}{10}$, then $\mathcal{A}$ returns 1, and otherwise it returns 0.

Clearly, the algorithm $\mathcal{A}$ runs in polynomial time. We now show that if $\mathcal{S}$ is pseudorandom then $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$, and if $\mathcal{S}$ is random then $\mathcal{A}$ returns 1 with probability less than $\frac{1}{3}$. By Lemma 18, there is a DNF formula $\psi_{\mathbf{x}}$ over $\{0,1\}^{kn}$ with at most $2^k < q(kn)$ terms (for a sufficiently large n), such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = \psi_{\mathbf{x}}(\mathbf{z}^S)$. Thus, $\mathcal{S}$ is realized by $\psi_{\mathbf{x}}$. Hence, if $\mathcal{S}$ is pseudorandom then with probability at least $\frac{3}{4}$ the algorithm $\mathcal{L}$ returns a hypothesis h such that $\mathbb{E}_{\mathbf{z} \sim \mathcal{D}} \mathbb{1}(h(\mathbf{z}) \neq P_{\mathbf{x}}(\mathbf{z})) \leq \frac{1}{10}$. Therefore, $\mathbb{E}_{\mathcal{S}_I} \ell_I(h) = \mathbb{E}_{\mathbf{z} \sim \mathcal{D}} \mathbb{1}(h(\mathbf{z}) \neq P_{\mathbf{x}}(\mathbf{z})) \leq \frac{1}{10}$. If $\mathcal{S}$ is random then for every function $h : \{0,1\}^{kn} \to \{0,1\}$ the events $\{h(\mathbf{z}^{S_i}) = y_i\}_{i \in I}$ are independent from one another, and each has probability $\frac{1}{2}$. Hence, $\mathbb{E}_{\mathcal{S}_I} \ell_I(h) = \frac{1}{2}$.

By the Hoefding bound, for a sufficiently large n we have

$$\Pr_{\mathcal{S}_I}\left[\left|\ell_I(h) - \mathop{\mathbb{E}}_{\mathcal{S}_I} \ell_I(h)\right| \geq \frac{1}{10}\right] \leq \frac{1}{20} .$$

Therefore, if $\mathcal{S}$ is pseudorandom then for a sufficiently large n we have with probability at least $1 - \left(\frac{1}{4} + \frac{1}{20}\right) = \frac{7}{10} > \frac{2}{3}$ that $\mathbb{E}_{\mathcal{S}_I} \ell_I(h) \leq \frac{1}{10}$ and $|\ell_I(h) - \mathbb{E}_{\mathcal{S}_I} \ell_I(h)| < \frac{1}{10}$, and hence $\ell_I(h) \leq \frac{2}{10}$. Thus, the algorithm $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$. If $\mathcal{S}$ is random then $\mathbb{E}_{\mathcal{S}} \ell_I(h) = \frac{1}{2}$ and for a sufficiently large n we have with probability at least $\frac{19}{20}$ that $|\ell_I(h) - \mathbb{E}_{\mathcal{S}_I} \ell_I(h)| < \frac{1}{10}$. Hence, with probability greater than $\frac{2}{3}$ we have $\ell_I(h) > \frac{2}{10}$ and the algorithm $\mathcal{A}$ returns 0. ■

We will use the following lemma throughout our proofs.

Lemma 20 *Let $c \geq 0$ be a constant. Let $\xi_1, \ldots, \xi_{n^{2c+3}}$ be a sequence of i.i.d. random variables and let $\xi = \frac{1}{n^{2c+3}} \sum_{i \in [n^{2c+3}]} \xi_i$. Assume that $\Pr\left[0 \leq \xi_i \leq 1\right] = 1$ for every i. Then for a sufficiently large n we have*

$$\Pr\left[|\xi - \mathbb{E}[\xi]| \geq \frac{1}{n^{c+1}}\right] < \frac{1}{20} .$$

Proof By the Hoefding bound we have

$$\Pr\left[|\xi - \mathbb{E}[\xi]| \geq \frac{1}{n^{c+1}}\right] \leq 2 \exp\left(-\frac{2n^{2c+3}}{n^{(c+1) \cdot 2}}\right) = 2 \exp\left(-2n\right) .$$

Thus, for a sufficiently large n the requirement holds. ■

In the following lemma we prove the second part of the theorem.

Lemma 21 *For every constant $\epsilon > 0$, there is no efficient algorithm that learns DNF formulas with n^ϵ terms, on a distribution such that each component is drawn i.i.d. from a (non-uniform) Bernoulli distribution.*

Proof Consider the distribution $\mathcal{D}$ over $\{0,1\}^{n^{1+3/\epsilon}}$, such that each component is drawn i.i.d. from a Bernoulli distribution where the probability of 0 is $\frac{1}{n}$. Assume that there is an efficient algorithm $\mathcal{L}$ that learns DNF formulas over $\{0,1\}^{n^{1+3/\epsilon}}$ with at most n^3 terms on the distribution $\mathcal{D}$. Let $m(n)$ be a polynomial such that $\mathcal{L}$ uses a sample of size at most $m(n)$ and returns with probability at least $\frac{3}{4}$ a hypothesis with error at most $\frac{1}{n}$. Let $s > 1$ be a constant such that $n^s \geq m(n) + n^3$ for every sufficiently large n. By Assumption 1, there exists a constant k and a predicate $P : \{0,1\}^k \to \{0,1\}$, such that $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG. We will show an algorithm $\mathcal{A}$ with distinguishing advantage greater than $\frac{1}{3}$ and thus reach a contradiction.

We say that $\mathbf{z} \in \{0,1\}^{n^{1+3/\epsilon}}$ is an extended encoding of a hyperedge if $(z_1, \ldots, z_{kn}) = \mathbf{z}^S$ for some hyperedge S. That is, in each of the first k size-n slices in $\mathbf{z}$ there is exactly one 0-bit and each two of the first k slices in $\mathbf{z}$ encode different indices. Assuming that $n^{1+3/\epsilon} \geq kn$, the probability that $\mathbf{z} \sim \mathcal{D}$ is an extended encoding of a hyperedge, is given by

$$\begin{aligned} n \cdot (n-1) \cdot \ldots \cdot (n-k+1) \cdot \left(\frac{1}{n}\right)^k \left(\frac{n-1}{n}\right)^{nk-k} &\geq \left(\frac{n-k}{n}\right)^k \left(\frac{n-1}{n}\right)^{k(n-1)} \\ &= \left(1 - \frac{k}{n}\right)^k \left(1 - \frac{1}{n}\right)^{k(n-1)} . \end{aligned}$$

Since for every $x \in (0,1)$ we have $e^{-x} < 1 - \frac{x}{2}$, then for a sufficiently large n the above is at least

$$\exp\left(-\frac{2k^2}{n}\right) \cdot \exp\left(-\frac{2k(n-1)}{n}\right) \geq \exp(-1) \cdot \exp(-2k) \geq \frac{1}{\log(n)} . \tag{1}$$

Given a sequence $(S_1, y_1), \ldots, (S_{n^s}, y_{n^s})$, where $S_1, \ldots, S_{n^s}$ are i.i.d. random hyperedges, the algorithm $\mathcal{A}$ needs to distinguish whether $\mathbf{y} = (y_1, \ldots, y_{n^s})$ is random or that we have $\mathbf{y} = (P(\mathbf{x}_{S_1}), \ldots, P(\mathbf{x}_{S_{n^s}})) = (P_{\mathbf{x}}(\mathbf{z}^{S_1}), \ldots, P_{\mathbf{x}}(\mathbf{z}^{S_{n^s}}))$ for a random $\mathbf{x} \in \{0,1\}^n$. We denote $\mathcal{S} = (\mathbf{z}^{S_1}, y_1), \ldots, (\mathbf{z}^{S_{n^s}}, y_{n^s})$.

We use the efficient algorithm $\mathcal{L}$ in order to obtain distinguishing advantage greater than $\frac{1}{3}$ as follows. The algorithm $\mathcal{A}$ runs $\mathcal{L}$ with the following examples oracle. In the i-th call to the oracle, it chooses $\mathbf{z}_i \in \{0,1\}^{n^{1+3/\epsilon}}$ according to $\mathcal{D}$. If $\mathbf{z}_i$ is not an extended encoding of a hyperedge (with probability at most $1 - \frac{1}{\log(n)}$ by Eq. 1) then the oracle returns $(\mathbf{z}'_i, y'_i)$ where $\mathbf{z}'_i = \mathbf{z}_i$ and $y'_i = 1$. Otherwise, the oracle obtains a vector $\mathbf{z}'_i$ by replacing the first kn components in $\mathbf{z}_i$ with $\mathbf{z}^{S_i}$, and returns $(\mathbf{z}'_i, y'_i)$ where $y'_i = y_i$. Note that the vector $\mathbf{z}'_i$ returned by the oracle has the distribution $\mathcal{D}$, since replacing a random hyperedge with another random hyperedge does not change the distribution. Let h be the hypothesis returned by $\mathcal{L}$. Recall that $\mathcal{L}$ uses at most $m(n)$ examples, and hence $\mathcal{S}$ contains at least n^3 examples that $\mathcal{L}$ cannot view. We denote the indices of these examples by $I = \{m(n)+1, \ldots, m(n)+n^3\}$, and the examples by $\mathcal{S}_I = \{(\mathbf{z}^{S_i}, y_i)\}_{i \in I}$. By n^3 additional calls to the oracle, the algorithm $\mathcal{A}$ obtains the examples $\mathcal{S}'_I = \{(\mathbf{z}'_i, y'_i)\}_{i \in I}$ that correspond to $\mathcal{S}_I$. Let $\ell_I(h) = \frac{1}{|I|} \sum_{i \in I} \mathbb{1}(h(\mathbf{z}'_i) \neq y'_i)$. Now, if $\ell_I(h) \leq \frac{2}{n}$, then $\mathcal{A}$ returns 1, and otherwise it returns 0. Clearly, the algorithm $\mathcal{A}$ runs in polynomial time. We now show that if $\mathcal{S}$ is pseudorandom then $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$, and if $\mathcal{S}$ is random then $\mathcal{A}$ returns 1 with probability less than $\frac{1}{3}$.

Consider a DNF formula ψ with $k \cdot \frac{n(n-1)}{2} + k + n \cdot \frac{k(k-1)}{2}$ terms such that $\psi(\mathbf{z}) = 1$ iff at least one of the first k size-n slices in $\mathbf{z}$ contains 0 more than once or less than once, or that from the first k slices in $\mathbf{z}$ there are two slices that encode the same index. Namely, ψ return 1 iff $\mathbf{z}$ is not an extended encoding of a hyperedge. The construction of such a formula ψ is straightforward. By Lemma 18, there is a DNF formula $\psi_{\mathbf{x}}$ over $\{0,1\}^{kn}$ with at most 2^k terms, such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = \psi_{\mathbf{x}}(\mathbf{z}^S)$. Let $\psi' = \psi \vee \psi_{\mathbf{x}}$. Note that ψ' consists of $k \cdot \frac{n(n-1)}{2} + k + n \cdot \frac{k(k-1)}{2} + 2^k$ terms, which is at most n^3 (for a sufficiently large n). Also, note that the inputs to ψ' are in $\{0,1\}^{n^{1+3/\epsilon}}$, but it uses only the first kn components of the input.

If $\mathcal{S}$ is pseudorandom then the examples $(\mathbf{z}'_i, y'_i)$ returned by the oracle satisfy $y'_i = \psi'(\mathbf{z}'_i)$. Indeed, if $\mathbf{z}'_i$ is an extended encoding of a hyperedge S_i then $\psi(\mathbf{z}'_i) = 0$ and $y'_i = P_{\mathbf{x}}(\mathbf{z}^{S_i}) = \psi_{\mathbf{x}}(\mathbf{z}^{S_i})$, and otherwise $y'_i = \psi(\mathbf{z}'_i) = 1$. Hence, if $\mathcal{S}$ is pseudorandom then with probability at least $\frac{3}{4}$ the algorithm $\mathcal{L}$ returns a hypothesis h such that $\mathbb{E}_{\mathbf{z}\sim\mathcal{D}} \mathbb{1}(h(\mathbf{z}) \neq \psi'(\mathbf{z})) \leq \frac{1}{n}$. Therefore, $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \leq \frac{1}{n}$.

If $\mathcal{S}$ is random, then for every i such that $\mathbf{z}'_i$ is an extended encoding of a hyperedge S_i, we have $y'_i = 1$ w.p. $\frac{1}{2}$ and $y'_i = 0$ otherwise, and y'_i is independent of S_i. Hence, for every h and $i \in I$ we have

$$\Pr\left[h(\mathbf{z}'_i) \neq y'_i\right] \geq \Pr\left[h(\mathbf{z}'_i) \neq y'_i \mid \mathbf{z}'_i \text{ represents a hyperedge}\right] \cdot \Pr\left[\mathbf{z}'_i \text{ represents a hyperedge}\right]$$
$$\overset{(Eq.\ 1)}{\geq} \frac{1}{2} \cdot \frac{1}{\log(n)} = \frac{1}{2\log(n)} .$$

Thus, $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \geq \frac{1}{2\log(n)}$.

By Lemma 20 (with $c = 0$), we have for a sufficiently large n that

$$\Pr_{\mathcal{S}'_I}\left[\left|\ell_I(h) - \mathop{\mathbb{E}}_{\mathcal{S}'_I} \ell_I(h)\right| \geq \frac{1}{n}\right] < \frac{1}{20} .$$

Therefore, if $\mathcal{S}$ is pseudorandom, then for a sufficiently large n, we have with probability at least $1 - \left(\frac{1}{4} + \frac{1}{20}\right) = \frac{7}{10} > \frac{2}{3}$ that $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \leq \frac{1}{n}$ and $\left|\ell_I(h) - \mathbb{E}_{\mathcal{S}'_I} \ell_I(h)\right| < \frac{1}{n}$, and hence $\ell_I(h) \leq \frac{2}{n}$. Thus, the algorithm $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$. If $\mathcal{S}$ is random then $\mathbb{E}_{\mathcal{S}'} \ell_I(h) \geq \frac{1}{2\log(n)}$ and for a sufficiently large n we have with probability at least $\frac{19}{20}$ that $\left|\ell_I(h) - \mathbb{E}_{\mathcal{S}'_I} \ell_I(h)\right| < \frac{1}{n}$. Hence, with probability greater than $\frac{2}{3}$ we have $\ell_I(h) \geq \frac{1}{2\log(n)} - \frac{1}{n} > \frac{2}{n}$ and the algorithm $\mathcal{A}$ returns 0.

Hence, it is hard to learn DNF formulas with n^3 terms where the input distribution is $\mathcal{D}$. Thus, for $\tilde{n} = n^{1+3/\epsilon}$, we have that it is hard to learn DNF formulas with $\tilde{n}^\epsilon = n^{(1+3/\epsilon)\cdot\epsilon} = n^{\epsilon+3} \geq n^3$ terms on a distribution over $\{0,1\}^{\tilde{n}}$, where each component is drawn i.i.d. from a Bernoulli distribution. ■

A.2. Proof of Theorem 4

Let $\mathcal{D}$ be the uniform distribution on $\{0,1\}^{n^{1+2/\epsilon}}$. Assume that there is an efficient algorithm $\mathcal{L}$ that learns depth-3 Boolean circuits of size n^2 on the distribution $\mathcal{D}$. Let $m(n)$ be a polynomial such that $\mathcal{L}$ uses a sample of size at most $m(n)$ and returns with probability at least $\frac{3}{4}$ a hypothesis h with error at most $\frac{1}{2} - \gamma$. Let $s > 1$ be a constant such that $n^s \geq m(n) + n$ for every sufficiently large n. By Assumption 1, there exists a constant k and a predicate $P : \{0,1\}^k \to \{0,1\}$, such that $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG. We will show an algorithm $\mathcal{A}$ with distinguishing advantage greater than $\frac{1}{3}$ and thus reach a contradiction.

For a hyperedge S we denote by $\mathbf{z}^S \in \{0,1\}^{kn}$ the encoding of S that is defined in the proof of Theorem 3. The *compressed encoding* of S, denoted by $\tilde{\mathbf{z}}^S \in \{0,1\}^{k\log(n)}$, is a concatenation of k size-$\log(n)$ slices, such that the i-th slice is a binary representation of the i-th member in S. We sometimes denote the i-th slice of $\tilde{\mathbf{z}} \in \{0,1\}^{k\log(n)}$ by $\tilde{z}_{i,1},\ldots,\tilde{z}_{i,\log(n)}$. For $\mathbf{x} \in \{0,1\}^n$, let $P_{\mathbf{x}} : \{0,1\}^{kn} \to \{0,1\}$ and $\tilde{P}_{\mathbf{x}} : \{0,1\}^{k\log(n)} \to \{0,1\}$ be such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = \tilde{P}_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = P(\mathbf{x}_S)$. We say that $\tilde{\mathbf{z}} \in \{0,1\}^{n^{1+2/\epsilon}}$ is an *extended compressed encoding* of a hyperedge S, if $(\tilde{z}_1,\ldots,\tilde{z}_{k\log(n)}) = \tilde{\mathbf{z}}^S$, namely, $\tilde{\mathbf{z}}$ starts with the compressed encoding $\tilde{\mathbf{z}}^S$.

If $\tilde{\mathbf{z}}$ is drawn from the uniform distribution on $\{0,1\}^{n^{1+2/\epsilon}}$, then, for a sufficiently large n, the probability that it is an extended compressed encoding of a hyperedge, namely, that each two of the first k size-$\log(n)$ slices encode different indices, is

$$\frac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{n^k} \geq \left(\frac{n-k}{n}\right)^k = \left(1-\frac{k}{n}\right)^k \geq 1-\frac{\gamma}{2}\,. \tag{2}$$

Given a sequence $(S_1,y_1),\ldots,(S_{n^s},y_{n^s})$, where $S_1,\ldots,S_{n^s}$ are i.i.d. random hyperedges, the algorithm $\mathcal{A}$ needs to distinguish whether $\mathbf{y} = (y_1,\ldots,y_{n^s})$ is random or that we have $\mathbf{y} = (P(\mathbf{x}_{S_1}),\ldots,P(\mathbf{x}_{S_{n^s}})) = (\tilde{P}_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_1}),\ldots,\tilde{P}_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_{n^s}}))$ for a random $\mathbf{x} \in \{0,1\}^n$. We denote $\mathcal{S} = (\tilde{\mathbf{z}}^{S_1},y_1),\ldots,(\tilde{\mathbf{z}}^{S_{n^s}},y_{n^s})$.

We use the efficient algorithm $\mathcal{L}$ in order to obtain distinguishing advantage greater than $\frac{1}{3}$ as follows. The algorithm $\mathcal{A}$ runs $\mathcal{L}$ with the following examples oracle. In the i-th call to the oracle, it chooses $\tilde{\mathbf{z}}_i \in \{0,1\}^{n^{1+3/\epsilon}}$ according to $\mathcal{D}$. If $\tilde{\mathbf{z}}_i$ is not an extended compressed encoding of a hyperedge (with probability at most $\frac{\gamma}{2}$, by Eq. 2), then the oracle returns $(\mathbf{z}'_i,y'_i)$ where $\mathbf{z}'_i = \tilde{\mathbf{z}}_i$ and $y'_i = 1$. Otherwise, the oracle obtained a vector $\mathbf{z}'_i$ by replacing the first $k\log(n)$ components in $\tilde{\mathbf{z}}_i$ with $\tilde{\mathbf{z}}^{S_i}$, and returns $(\mathbf{z}'_i,y'_i)$ where $y'_i = y_i$. Note that the vector $\mathbf{z}'_i$ returned by the oracle has the distribution $\mathcal{D}$, since replacing a random hyperedge with another random hyperedge does not change the distribution. Let h be the hypothesis returned by $\mathcal{L}$. Recall that $\mathcal{L}$ uses at most $m(n)$ examples, and hence $\mathcal{S}$ contains at least n examples that $\mathcal{L}$ cannot view. We denote the indices of these examples by $I = \{m(n)+1,\ldots,m(n)+n\}$, and the examples by $\mathcal{S}_I = \{(\tilde{\mathbf{z}}^{S_i},y_i)\}_{i\in I}$. By n additional calls to the oracle, the algorithm $\mathcal{A}$ obtains the examples $\mathcal{S}'_I = \{(\mathbf{z}'_i,y'_i)\}_{i\in I}$ that correspond to $\mathcal{S}_I$. Let $\ell_I(h) = \frac{1}{|I|}\sum_{i\in I}\mathbb{1}(h(\mathbf{z}'_i) \neq y'_i)$. Now, if $\ell_I(h) \leq \frac{1}{2}-\frac{\gamma}{2}$, then $\mathcal{A}$ returns 1, and otherwise it returns 0. Clearly, the algorithm $\mathcal{A}$ runs in polynomial time. We now show that if $\mathcal{S}$ is pseudorandom then $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$, and if $\mathcal{S}$ is random then $\mathcal{A}$ returns 1 with probability less than $\frac{1}{3}$.

Consider the encoding $\mathbf{z}^S$ and the compressed encoding $\tilde{\mathbf{z}}^S$ of a hyperedge S. Note that for every $i \in [k]$ and $j \in [n]$, we have $z^S_{i,j} = 0$ iff $(\tilde{z}^S_{i,1},\ldots,\tilde{z}^S_{i,\log(n)})$ is the binary representation of j. Hence, we can express $\neg z^S_{i,j}$ by a conjunction with the variables $\tilde{\mathbf{z}}^S$, and express $z^S_{i,j}$ by a disjunction with the variables $\tilde{\mathbf{z}}^S$. By Lemma 18, there is a DNF formula $\psi_{\mathbf{x}}$ over $\{0,1\}^{kn}$ with at most 2^k terms, such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = \psi_{\mathbf{x}}(\mathbf{z}^S)$. Let $C_{\mathbf{x}}$ be a depth-3 Boolean circuit such that for every hyperedge S we have $C_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = \psi_{\mathbf{x}}(\mathbf{z}^S)$. The circuit $C_{\mathbf{x}}$ is obtained from $\psi_{\mathbf{x}}$ by replacing every literal $z_{i,j}$ with the appropriate disjunction. Hence, we have $C_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = \psi_{\mathbf{x}}(\mathbf{z}^S) = P_{\mathbf{x}}(\mathbf{z}^S) = \tilde{P}_{\mathbf{x}}(\tilde{\mathbf{z}}^S)$. Note that $C_{\mathbf{x}}$ has $1+2^k+nk \leq \frac{n^2}{2}$ gates (for a sufficiently large n).

Consider a DNF formula ψ over $\{0,1\}^{n^{1+2/\epsilon}}$ with $n\cdot\frac{k(k-1)}{2} \leq \frac{n^2}{2}$ terms (for a sufficiently large n) such that $\psi(\tilde{\mathbf{z}}) = 1$ iff $\tilde{\mathbf{z}}$ is not an extended compressed encoding of a hyperedge, namely, from the first k size-$\log(n)$ slices in $\tilde{\mathbf{z}}$ there are two slices that encode the same index. The construction of such a formula ψ is straightforward. Let C' be a depth-3 Boolean circuit such that $C' = C_{\mathbf{x}} \vee \psi$. Note that the inputs to C' are in $\{0,1\}^{n^{1+2/\epsilon}}$, but it uses only the first $k\log(n)$ components of the input. The circuit C' has at most n^2 gates.

If $\mathcal{S}$ is pseudorandom then the examples $(\mathbf{z}'_i, y'_i)$ returned by the oracle satisfy $y'_i = C'(\mathbf{z}'_i)$. Indeed, if $\mathbf{z}'_i$ is an extended compressed encoding of a hyperedge S_i then $\psi(\mathbf{z}'_i) = 0$ and $y'_i = \tilde{P}_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_i}) = C_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_i})$, and otherwise $y'_i = \psi(\mathbf{z}'_i) = 1$. Hence, if $\mathcal{S}$ is pseudorandom then with probability at least $\frac{3}{4}$ the algorithm $\mathcal{L}$ returns a hypothesis h such that $\mathbb{E}_{\tilde{\mathbf{z}}\sim\mathcal{D}} \mathbb{1}(h(\tilde{\mathbf{z}}) \neq C'(\tilde{\mathbf{z}})) \leq \frac{1}{2} - \gamma$. Therefore, $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \leq \frac{1}{2} - \gamma$.

If $\mathcal{S}$ is random, then for every i such that $\mathbf{z}'_i$ is an extended compressed encoding of a hyperedge S_i, we have $y'_i = 1$ w.p. $\frac{1}{2}$ and $y'_i = 0$ otherwise, and y'_i is independent of S_i. Hence, for every h and $i \in I$ we have

$$\Pr\left(h(\mathbf{z}'_i) \neq y'_i\right) \geq \Pr\left[h(\mathbf{z}'_i) \neq y'_i \mid \mathbf{z}'_i \text{ represents a hyperedge}\right] \cdot \Pr\left[\mathbf{z}'_i \text{ represents a hyperedge}\right]$$
$$\overset{(Eq.~2)}{\geq} \frac{1}{2} \cdot \left(1 - \frac{\gamma}{2}\right) = \frac{1}{2} - \frac{\gamma}{4}\,.$$

Thus, $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \geq \frac{1}{2} - \frac{\gamma}{4}$.

By the Hoefding bound, for a sufficiently large n we have

$$\Pr_{\mathcal{S}'_i}\left[\left|\ell_I(h) - \mathop{\mathbb{E}}_{\mathcal{S}'_i} \ell_I(h)\right| \geq \frac{\gamma}{4}\right] \leq \frac{1}{20}\,.$$

Therefore, if $\mathcal{S}$ is pseudorandom then for a sufficiently large n we have with probability at least $1 - \left(\frac{1}{4} + \frac{1}{20}\right) = \frac{7}{10} > \frac{2}{3}$ that $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \leq \frac{1}{2} - \gamma$ and $\left|\ell_I(h) - \mathbb{E}_{\mathcal{S}'_I} \ell_I(h)\right| < \frac{\gamma}{4}$, and hence $\ell_I(h) < \frac{1}{2} - \frac{3}{4}\gamma < \frac{1}{2} - \frac{\gamma}{2}$. Thus, the algorithm $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$. If $\mathcal{S}$ is random then $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \geq \frac{1}{2} - \frac{\gamma}{4}$, and for a sufficiently large n we have with probability at least $\frac{19}{20}$ that $\left|\ell_I(h) - \mathbb{E}_{\mathcal{S}'_I} \ell_I(h)\right| < \frac{\gamma}{4}$. Hence, with probability greater than $\frac{2}{3}$ we have $\ell_I(h) > \frac{1}{2} - \frac{\gamma}{2}$ and the algorithm $\mathcal{A}$ returns 0.

Hence, it is hard to weakly-learn depth-3 Boolean circuits of size at most n^2 where the input distribution is $\mathcal{D}$. Thus, for $\tilde{n} = n^{1+2/\epsilon}$, we have that it is hard to weakly-learn depth-3 Boolean circuits of size $\tilde{n}^{\epsilon} = n^{(1+2/\epsilon)\cdot\epsilon} = n^{\epsilon+2} \geq n^2$, on the uniform distribution over $\{0,1\}^{\tilde{n}}$.

A.3. Proof of Theorem 6

By Assumption 2, there is a constant $\alpha > 0$ such that for every constant $s > 1$ there is a constant l such that for the predicate $P = \text{XOR-MAJ}_{\lceil \alpha s \rceil, l}$ the collection $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG. Let $\beta = \frac{1}{2.1\alpha}$, and let $k > \alpha$ be an integer constant. By our assumption, for $s = \frac{k}{\alpha}$, there is a constant l such that for the predicate $P = \text{XOR-MAJ}_{\alpha s, l} = \text{XOR-MAJ}_{k,l}$ the collection $\mathcal{F}_{P,n,n^s} = \mathcal{F}_{P,n,n^{k/\alpha}} = \mathcal{F}_{P,n,n^{2.1\beta k}}$ is $\frac{1}{3}$-PRG. Let $\tilde{n} = \frac{(2n)(2n-1)}{2} + 2n + 1$. Assume that there is an efficient algorithm $\mathcal{L}$ that learns intersections of k halfspaces over $\{0,1\}^{\tilde{n}}$. Assume that $\mathcal{L}$ uses a sample of size $m(\tilde{n}) = \tilde{n}^{\beta k}$ and returns with probability at least $\frac{3}{4}$ a hypothesis with error at most $\frac{1}{10}$. We will show an algorithm $\mathcal{A}$ with distinguishing advantage greater than $\frac{1}{3}$ and thus reach a contradiction. It implies that an efficient algorithm that learns intersections of k halfspaces over $\{0,1\}^{\tilde{n}}$ must use a sample of size greater than $\tilde{n}^{\beta k}$, and therefore runs in time $\Omega(\tilde{n}^{\beta k})$. Note that we assume that $k > \alpha$. For $k \leq \alpha$ the claim holds trivially, since learning intersections of k halfspaces on $\{0,1\}^{\tilde{n}}$ clearly requires time $\Omega(\tilde{n})$, and we have $\tilde{n}^{\beta k} \leq \tilde{n}^{\beta\alpha} = \tilde{n}^{1/2.1} \leq \tilde{n}$.

For $\mathbf{z} \in \{0,1\}^{2n}$, we denote by $\tilde{\mathbf{z}} \in \{0,1\}^{\tilde{n}}$ the vector of all monomials over $\mathbf{z}$ of degree at most 2. We call $\tilde{\mathbf{z}}$ the *monomials encoding* of $\mathbf{z}$. We encode a hyperedge $S = (i_1, \ldots, i_{k+l})$ by $\mathbf{z}^S \in \{0,1\}^{2n}$, where $\mathbf{z}^S$ is the concatenation of 2 vectors in $\{0,1\}^n$, such that the first vector has 1-bits in the indices $i_1, \ldots, i_k$ and 0 elsewhere, and the second vector has 1-bits in the indices $i_{k+1}, \ldots, i_{k+l}$ and 0 elsewhere. We denote by $\tilde{\mathbf{z}}^S$ the monomials encoding of $\mathbf{z}^S$. For $\mathbf{x} \in \{0,1\}^n$, let $P_{\mathbf{x}} : \{0,1\}^{\tilde{n}} \to \{0,1\}$ be a function such that for every hyperedge S we have $P_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = P(\mathbf{x}_S)$.

Lemma 22 *For every $\mathbf{x} \in \{0,1\}^n$, there is a function $g : \{0,1\}^{\tilde{n}} \to \{0,1\}$ that can be expressed by an intersection of k halfspaces, such that for every hyperedge S we have $g(\tilde{\mathbf{z}}^S) = P_{\mathbf{x}}(\tilde{\mathbf{z}}^S)$.*

Proof For $\mathbf{z} \in \{0,1\}^{2n}$, we denote $\mathbf{z}^1 = (z_1, \ldots, z_n)$ and $\mathbf{z}^2 = (z_{n+1}, \ldots, z_{2n})$. For every $\mathbf{z} \in \{0,1\}^{2n}$ and even $i \in [k]$, let

$$f_i(\mathbf{z}) = \left(\langle \mathbf{x}, \mathbf{z}^1 \rangle - i\right)^2 \cdot l + \left(\langle \mathbf{x}, \mathbf{z}^2 \rangle - \left\lfloor \frac{l}{2} \right\rfloor\right) ,$$

and for every odd $i \in [k]$ let

$$f_i(\mathbf{z}) = \left(\langle \mathbf{x}, \mathbf{z}^1 \rangle - i\right)^2 \cdot l + 1 - \left(\langle \mathbf{x}, \mathbf{z}^2 \rangle - \left\lfloor \frac{l}{2} \right\rfloor\right) .$$

Then, let $f : \{0,1\}^{2n} \to \{0,1\}$ be such that

$$f(\mathbf{z}) = \bigwedge_{i \in [k]} \operatorname{sign}(f_i(\mathbf{z})) .$$

Note that the functions f_i are degree-2 polynomials. For every $i \in [k]$, let $g_i : \{0,1\}^{\tilde{n}} \to \mathbb{R}$ be a linear function such that for every $\mathbf{z} \in \{0,1\}^{2n}$ we have $g_i(\tilde{\mathbf{z}}) = f_i(\mathbf{z})$, where $\tilde{\mathbf{z}}$ is the monomials encoding of $\mathbf{z}$. Let $g : \{0,1\}^{\tilde{n}} \to \{0,1\}$ be such that

$$g(\tilde{\mathbf{z}}) = \bigwedge_{i \in [k]} \operatorname{sign}(g_i(\tilde{\mathbf{z}})) .$$

Note that the function g is an intersection of k halfspaces, and that for every $\mathbf{z} \in \{0,1\}^{2n}$ we have $f(\mathbf{z}) = g(\tilde{\mathbf{z}})$.

Assume that $\mathbf{z} = \mathbf{z}^S$ for a hyperedge $S = (i_1, \ldots, i_{k+l})$, and let $S^1 = (i_1, \ldots, i_k)$ and $S^2 = (i_{k+1}, \ldots, i_{k+l})$. Note that $\operatorname{sign}\left(\langle \mathbf{x}, \mathbf{z}^2 \rangle - \lfloor \frac{l}{2} \rfloor\right) = \mathrm{MAJ}_l(\mathbf{x}_{S^2})$. We have:

- If i is even and $\langle \mathbf{x}, \mathbf{z}^1 \rangle = i$, then $\operatorname{sign}(f_i(\mathbf{z})) = \operatorname{sign}\left(\langle \mathbf{x}, \mathbf{z}^2 \rangle - \lfloor \frac{l}{2} \rfloor\right) = \mathrm{MAJ}_l(\mathbf{x}_{S^2})$.
- If i is odd and $\langle \mathbf{x}, \mathbf{z}^1 \rangle = i$, then $\operatorname{sign}(f_i(\mathbf{z})) = \operatorname{sign}\left(1 - \left(\langle \mathbf{x}, \mathbf{z}^2 \rangle - \lfloor \frac{l}{2} \rfloor\right)\right) = 1 - \mathrm{MAJ}_l(\mathbf{x}_{S^2})$.
- If $\langle \mathbf{x}, \mathbf{z}^1 \rangle \neq i$ then $\left(\langle \mathbf{x}, \mathbf{z}^1 \rangle - i\right)^2 \cdot l \geq l$. Since we also have $-\lfloor \frac{l}{2} \rfloor \leq \langle \mathbf{x}, \mathbf{z}^2 \rangle - \lfloor \frac{l}{2} \rfloor \leq \lceil \frac{l}{2} \rceil$, then $\operatorname{sign}(f_i(\mathbf{z})) = 1$.

Since for every i such that $\langle \mathbf{x}, \mathbf{z}^1 \rangle \neq i$ we have $\operatorname{sign}(f_i(\mathbf{z})) = 1$, then $f(\mathbf{z}) = \operatorname{sign}(f_{\langle \mathbf{x}, \mathbf{z}^1 \rangle}(\mathbf{z}))$. Hence, if $\langle \mathbf{x}, \mathbf{z}^1 \rangle$ is even then $f(\mathbf{z}) = \mathrm{MAJ}_l(\mathbf{x}_{S^2})$, and otherwise $f(\mathbf{z}) = 1 - \mathrm{MAJ}_l(\mathbf{x}_{S^2})$. Note that $\langle \mathbf{x}, \mathbf{z}^1 \rangle$ is the Hamming weight of $\mathbf{x}_{S^1}$. Therefore, we have

$$\begin{aligned} f(\mathbf{z}) &= [\neg \mathrm{XOR}_k(\mathbf{x}_{S^1}) \wedge \mathrm{MAJ}_l(\mathbf{x}_{S^2})] \vee [\mathrm{XOR}_k(\mathbf{x}_{S^1}) \wedge \neg \mathrm{MAJ}_l(\mathbf{x}_{S^2})] \\ &= \text{XOR-MAJ}_{k,l}(\mathbf{x}_S) = P(\mathbf{x}_S) . \end{aligned}$$

For $\tilde{\mathbf{z}} = \tilde{\mathbf{z}}^S$, we have $g(\tilde{\mathbf{z}}) = f(\mathbf{z}) = P(\mathbf{x}_S) = P_{\mathbf{x}}(\tilde{\mathbf{z}})$. Since g is an intersection of k halfspaces, the lemma follows. ■

Given a sequence $(S_1, y_1), \dots, (S_{n^{2.1\beta k}}, y_{n^{2.1\beta k}})$, where $S_1, \dots, S_{n^{2.1\beta k}}$ are i.i.d. random hyperedges, the algorithm $\mathcal{A}$ needs to distinguish whether $\mathbf{y} = (y_1, \dots, y_{n^{2.1\beta k}})$ is random, or that $\mathbf{y} = (P(\mathbf{x}_{S_1}), \dots, P(\mathbf{x}_{S_{n^{2.1\beta k}}})) = (P_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_1}), \dots, P_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_{n^{2.1\beta k}}}))$ for a random $\mathbf{x} \in \{0,1\}^n$. Let $\mathcal{S} = ((\tilde{\mathbf{z}}^{S_1}, y_1), \dots, (\tilde{\mathbf{z}}^{S_{n^{2.1\beta k}}}, y_{n^{2.1\beta k}}))$. Let $\mathcal{D}$ be a distribution on $\{0,1\}^{\tilde{n}}$ such that $\tilde{\mathbf{z}} \sim \mathcal{D}$ is the monomials encoding of a random hyperedge. Note that each $\tilde{\mathbf{z}}^{S_i}$ from $\mathcal{S}$ is drawn i.i.d. from $\mathcal{D}$.

We use the efficient algorithm $\mathcal{L}$ in order to obtain distinguishing advantage greater than $\frac{1}{3}$ as follows. The algorithm $\mathcal{A}$ learns a function $h : \{0,1\}^{\tilde{n}} \to \{0,1\}$ by running $\mathcal{L}$ with an examples oracle that in each call returns the next example from $\mathcal{S}$. Recall that $\mathcal{L}$ uses at most $m(\tilde{n}) = \tilde{n}^{\beta k}$ examples, and hence $\mathcal{S}$ contains at least

$$\begin{aligned} n^{2.1\beta k} - \tilde{n}^{\beta k} &\geq n^{2.1\beta k} - (2n)^{2\beta k} \geq n^{2.1\beta k} - (n^{1.01})^{2\beta k} = n^{2.1\beta k} - n^{2.02\beta k} \\ &= n^{2.02\beta k}(n^{0.08\beta k} - 1) \geq \ln(n) \end{aligned}$$

examples that $\mathcal{L}$ cannot view (for a sufficiently large n). We denote the indices of these examples by $I = \{m(\tilde{n}) + 1, \dots, m(\tilde{n}) + \ln(n)\}$, and the examples by $\mathcal{S}_I = \{(\tilde{\mathbf{z}}^{S_i}, y_i)\}_{i \in I}$. Let $\ell_I(h) = \frac{1}{|I|} \sum_{i \in I} \mathbb{1}(h(\tilde{\mathbf{z}}^{S_i}) \neq y_i)$. Now, if $\ell_I(h) \leq \frac{2}{10}$, then $\mathcal{A}$ returns 1, and otherwise it returns 0. Clearly, the algorithm $\mathcal{A}$ runs in polynomial time. We now show that if $\mathcal{S}$ is pseudorandom then $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$, and if $\mathcal{S}$ is random then $\mathcal{A}$ returns 1 with probability less than $\frac{1}{3}$.

If $\mathcal{S}$ is pseudorandom, then by Lemma 22, it can be realized by an intersection of k halfspaces. Hence, with probability at least $\frac{3}{4}$ the algorithm $\mathcal{L}$ returns a function h, such that $\mathbb{E}_{\tilde{\mathbf{z}} \sim \mathcal{D}} \mathbb{1}(h(\tilde{\mathbf{z}}) \neq P_{\mathbf{x}}(\tilde{\mathbf{z}})) \leq \frac{1}{10}$. Therefore, $\mathbb{E}_{\mathcal{S}_I} \ell_I(h) \leq \frac{1}{10}$. If $\mathcal{S}$ is random then for every function $h : \{0,1\}^{\tilde{n}} \to \{0,1\}$ the events $\{h(\tilde{\mathbf{z}}_i) = y_i\}_{i \in I}$ are independent from one another, and each has probability $\frac{1}{2}$. Hence, $\mathbb{E}_{\mathcal{S}_I} \ell_I(h) = \frac{1}{2}$.

By the Hoefding bound, for a sufficiently large n we have

$$\Pr_{\mathcal{S}_I}\left[\left|\ell_I(h) - \mathop{\mathbb{E}}_{\mathcal{S}_I} \ell_I(h)\right| \geq \frac{1}{10}\right] \leq \frac{1}{20}\ .$$

Therefore, if $\mathcal{S}$ is pseudorandom then for a sufficiently large n we have with probability at least $1 - \left(\frac{1}{4} + \frac{1}{20}\right) = \frac{7}{10} > \frac{2}{3}$ that $\mathbb{E}_{\mathcal{S}_I} \ell_I(h) \leq \frac{1}{10}$ and $|\ell_I(h) - \mathbb{E}_{\mathcal{S}_I} \ell_I(h)| < \frac{1}{10}$, and hence $\ell_I(h) \leq \frac{2}{10}$. Thus, the algorithm $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$. If $\mathcal{S}$ is random then $\mathbb{E}_{\mathcal{S}} \ell_I(h) = \frac{1}{2}$ and for a sufficiently large n we have with probability at least $\frac{19}{20}$ that $|\ell_I(h) - \mathbb{E}_{\mathcal{S}_I} \ell_I(h)| < \frac{1}{10}$. Hence, with probability greater than $\frac{2}{3}$ we have $\ell_I(h) > \frac{2}{10}$ and the algorithm $\mathcal{A}$ returns 0.

A.4. Proof of Theorem 8

Lemma 23 *Let $\mathcal{D}$ be a distribution on $\{0,1\}^n$ and let $\epsilon > 0$. Let $f : \{0,1\}^n \to \{0,1\}$ and $h : \{0,1\}^n \to \mathbb{R}$ be functions such that $\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}(f(\mathbf{x}) - h(\mathbf{x}))^2 \leq \frac{\epsilon}{4}$. Let $h' : \{0,1\}^n \to \{0,1\}$ be such that for every $\mathbf{x} \in \{0,1\}^n$ we have $h'(\mathbf{x}) = \operatorname{sign}\left(h(\mathbf{x}) - \frac{1}{2}\right)$. Then $\mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{1}(h'(\mathbf{x}) \neq f(\mathbf{x})) \leq \epsilon$.*

Proof For every $\mathbf{x}$ such that $h'(\mathbf{x}) \neq f(\mathbf{x})$, we have $(h(\mathbf{x}) - f(\mathbf{x}))^2 \geq \frac{1}{4}$. Hence,

$$\mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{D}} \mathbb{1}(h'(\mathbf{x}) \neq f(\mathbf{x})) \leq \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{D}} 4(h(\mathbf{x}) - f(\mathbf{x}))^2 \leq 4 \cdot \frac{\epsilon}{4} = \epsilon\ .$$

■

Lemma 24 *Let $\mathcal{H}' \subseteq \{0,1\}^{(\{0,1\}^n)}$ be a hypothesis class, and let $\mathcal{H} \subseteq \mathbb{R}^{(\{0,1\}^n)}$ be a hypothesis class such that $\mathcal{H}' \subseteq \mathcal{H}$. If it is hard to learn $\mathcal{H}'$ with respect to the 0-1 loss, then it is hard to learn $\mathcal{H}$ with respect to the square loss. Moreover, for every distribution $\mathcal{D}$ on $\{0,1\}^n$, if it is hard to learn $\mathcal{H}'$ on $\mathcal{D}$ with respect to the 0-1 loss, then it is hard to learn $\mathcal{H}$ on $\mathcal{D}$ with respect to the square loss.*

Proof Let $\epsilon, \delta \in (0,1)$. Assume that there is an efficient algorithm $\mathcal{L}$ that for every $f \in \mathcal{H}$ and distribution $\mathcal{D}$ on $\{0,1\}^n$, given access to examples $(\mathbf{x}, f(\mathbf{x}))$ where $\mathbf{x} \sim \mathcal{D}$, finds a hypothesis $h : \{0,1\}^n \to \mathbb{R}$ such that with probability at least $1-\delta$ we have $\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}(h(\mathbf{x}) - f(\mathbf{x}))^2 \leq \frac{\epsilon}{4}$. Consider a learning algorithm $\mathcal{L}'$, that given access to examples $(\mathbf{x}, f'(\mathbf{x}))$ where $\mathbf{x} \sim \mathcal{D}$ and $f' \in \mathcal{H}'$, runs $\mathcal{L}$, and returns a hypothesis $h' : \{0,1\}^n \to \{0,1\}$ such that for every $\mathbf{x} \in \mathbb{R}^n$ we have $h'(\mathbf{x}) = \operatorname{sign}\left(h(\mathbf{x}) - \frac{1}{2}\right)$. By Lemma 23, we have $\mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \mathbb{1}(h'(\mathbf{x}) \neq f(\mathbf{x})) \leq \epsilon$. Therefore, $\mathcal{H}'$ can be learned efficiently with respect to the 0-1 loss. The same argument holds also for the case of distribution-specific learning. ■

A.4.1. PROOF OF (1)

Note that in order to express a DNF formula with a depth-2 neural network, the network should have an activation function in the output neuron. Since, this is not allowed here, then the claim does not follow immediately from Lemma 19 and Lemma 24. Nevertheless, the proof follows similar ideas to the proof of Lemma 19, with a few modifications as detailed below.

In the proof of Lemma 19, we consider a sequence $\mathcal{S} = ((\mathbf{z}^{S_1}, y_1), \ldots, (\mathbf{z}^{S_{n^s}}, y_{n^s}))$, and show that if $\mathcal{S}$ is pseudorandom, namely, $y_i = P_{\mathbf{x}}(\mathbf{z}^{S_i})$ for all i, then for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = \psi_{\mathbf{x}}(\mathbf{z}^S)$, where $\psi_{\mathbf{x}}$ is a DNF formula with at most 2^k terms. Then, we use the assumption that there is an efficient algorithm for learning DNF formulas with $\omega(1)$ terms, in order to obtain a hypothesis h such that $\mathbb{E}_{\mathbf{z}\sim\mathcal{D}} \mathbb{1}(h(\mathbf{z}) \neq P_{\mathbf{x}}(\mathbf{z})) \leq \frac{1}{10}$, and we use h in order to obtain distinguishing advantage greater than $\frac{1}{3}$ and reach a contradiction. Here, we will show that for every hyperedge S we also have $P_{\mathbf{x}}(\mathbf{z}^S) = N_{\mathbf{x}}(\mathbf{z}^S)$, where $N_{\mathbf{x}}$ is a depth-2 neural network with 2^k hidden neurons and no activation in the output neuron. Then, if we assume that there is an efficient algorithm for learning depth-2 neural networks with $\omega(1)$ hidden neurons and no activation in the output neuron, with respect to the square loss, then we can obtain a hypothesis h with small error with respect to the square loss. By Lemma 23, we can obtain a hypothesis h' with small error with respect to the 0-1 loss. The arguments from the proof of Lemma 19 then imply that we can use h' in order to obtain distinguishing advantage greater than $\frac{1}{3}$ and reach a contradiction.

We now construct the neural network $N_{\mathbf{x}}$ such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = \psi_{\mathbf{x}}(\mathbf{z}^S) = N_{\mathbf{x}}(\mathbf{z}^S)$. Note that $N_{\mathbf{x}}$ should simulate $\psi_{\mathbf{x}}$ only for inputs that encode hyperedges, and not for all $\mathbf{z} \in \{0,1\}^{kn}$. Each term C_j in $\psi_{\mathbf{x}}$ is a conjunction of positive literals. Let $I_j \subseteq [kn]$ be the indices of these literals. Note that $C_j(\mathbf{z}^S)$ can be expressed by a single ReLU neuron that computes $\left[\left(\sum_{l\in I_j} z^S_l\right) - (|I_j| - 1)\right]_+$. Thus, our neural network $N_{\mathbf{x}}$ includes a hidden neuron for every term in $\psi_{\mathbf{x}}$. By the construction in Lemma 18, each conjunction $C_j(\mathbf{z}^S)$ checks whether $\mathbf{x}_S$ is the j-th satisfying assignment of the predicate P. Hence, it is not possible that more than one term in $\psi_{\mathbf{x}}(\mathbf{z}^S)$ is satisfied. Therefore, the network $N_{\mathbf{x}}$ computes $\psi_{\mathbf{x}}(\mathbf{z}^S)$ by summing the outputs of the hidden neurons, and since this sum is in $\{0,1\}$ then an activation function is not required in the output neuron.

A.4.2. PROOF OF (2) AND (3)

Implementing a depth-d Boolean circuit with a depth-d neural network is straightforward. Hence, the claims follow immediately from Theorems 3 and 4, and from Lemma 24.

A.5. Proof of Theorem 9

Note that in order to express an intersection of halfspaces with a depth-2 neural network, the network should have an activation function in the output neuron. Since, this is not allowed here, then the claim does not follow immediately from Theorem 6 and Lemma 23. Nevertheless, the proof follows similar ideas to the proof of Theorem 6, with a few modifications as detailed below.

In the proof of Theorem 6, we consider a sequence $\mathcal{S} = ((\tilde{\mathbf{z}}^{S_1}, y_1), \ldots, (\tilde{\mathbf{z}}^{S_{n^{2.1\beta k}}}, y_{n^{2.1\beta k}}))$, and show that if $\mathcal{S}$ is pseudorandom, namely, $y_i = P_{\mathbf{x}}(\tilde{\mathbf{z}}^{S_i})$ for all i, then for every hyperedge S we have $P_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = g_{\mathbf{x}}(\tilde{\mathbf{z}}^S)$, where $g_{\mathbf{x}}$ is an intersection of k halfspaces. Then, we use the assumption that there is an efficient algorithm that learns an intersections of k halfspaces and uses a sample of size $\tilde{n}^{\beta k}$, in order to obtain a hypothesis h such that $\mathbb{E}_{\mathbf{z}\sim\mathcal{D}} \mathbb{1}(h(\mathbf{z}) \neq P_{\mathbf{x}}(\mathbf{z})) \leq \frac{1}{10}$, and we use h in order to obtain distinguishing advantage greater than $\frac{1}{3}$ and reach a contradiction. Here, we will show that for every hyperedge S we also have $P_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = N_{\mathbf{x}}(\tilde{\mathbf{z}}^S)$, where $N_{\mathbf{x}}$ is a depth-2 neural networks with $2k$ hidden neurons and no activation in the output neuron. Then, if we assume that there is an efficient algorithm that learns such networks with respect to the square loss and uses a sample of size $\tilde{n}^{\beta k}$, then we can obtain a hypothesis h with small error with respect to the square loss. By Lemma 23, we can obtain a hypothesis h' with small error with respect to the 0-1 loss. The arguments from the proof of Theorem 6 then imply that we can use h' in order to obtain distinguishing advantage greater than $\frac{1}{3}$ and reach a contradiction.

We now construct the neural network $N_{\mathbf{x}}$ such that for every hyperedge S we have $P_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = g_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = N_{\mathbf{x}}(\tilde{\mathbf{z}}^S)$. Note that $N_{\mathbf{x}}$ should simulate $g_{\mathbf{x}}$ only for inputs that encode hyperedges, and not for all $\tilde{\mathbf{z}} \in \{0,1\}^{\tilde{n}}$. By Lemma 22, $g_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = \bigwedge_{i\in[k]} \operatorname{sign}(g_i(\tilde{\mathbf{z}}^S))$, and we show there that for every hyperedge S, there is at most one index i with $\operatorname{sign}(g_i(\tilde{\mathbf{z}}^S)) = 0$. Hence,

$$g_{\mathbf{x}}(\tilde{\mathbf{z}}^S) = \sum_{i\in[k]} \operatorname{sign}(g_i(\tilde{\mathbf{z}}^S)) - (k-1) \ .$$

The network $N_{\mathbf{x}}$ computes $\operatorname{sign}(g_i(\tilde{\mathbf{z}}^S))$ for every $i \in [k]$ using a single nonlinear layer. Then, the computation of $g_{\mathbf{x}}(\tilde{\mathbf{z}}^S)$ does not require an activation function in the output neuron. Note that the output neuron does not have to include a bias term, since the additive term $-(k-1)$ can be implemented by adding a hidden neuron with fan-in 0 and bias $k-1$, that is connected to the output neuron with weight 1.

For every $i \in [k]$ and $\tilde{\mathbf{z}} \in \{0,1\}^{\tilde{n}}$, the network $N_{\mathbf{x}}$ computes $\operatorname{sign}(g_i(\tilde{\mathbf{z}}))$ as follows. We denote $g_i(\tilde{\mathbf{z}}) = \langle \mathbf{w}_i, \tilde{\mathbf{z}} \rangle$. Since $\langle \mathbf{w}_i, \tilde{\mathbf{z}} \rangle$ is an integer, we have

$$\operatorname{sign}(g_i(\tilde{\mathbf{z}})) = [\langle \mathbf{w}_i, \tilde{\mathbf{z}} \rangle]_+ - [\langle \mathbf{w}_i, \tilde{\mathbf{z}} \rangle - 1]_+ \ .$$

Hence, computing $\operatorname{sign}(g_i(\tilde{\mathbf{z}}^S))$ requires 2 hidden neurons. Therefore, the network $N_{\mathbf{x}}$ includes $2k$ hidden neurons.

A.6. Proof of Theorem 11

Let $\mathcal{D}$ be the standard Gaussian distribution on $\mathbb{R}^{n^{1+3/\epsilon}}$. Assume that there is an efficient algorithm $\mathcal{L}$ that learns depth-3 neural networks with n^3 hidden neurons on the distribution $\mathcal{D}$. Let $m(n)$ be a polynomial such that $\mathcal{L}$ uses a sample of size at most $m(n)$ and returns with probability at least $\frac{3}{4}$ a hypothesis h with error at most $\frac{1}{n}$. Let $s > 1$ be a constant such that $n^s \geq m(n) + n^3$ for every sufficiently large n. By Assumption 1, there exists a constant k and a predicate $P : \{0,1\}^k \to \{0,1\}$, such that $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG. We will show an algorithm $\mathcal{A}$ with distinguishing advantage greater than $\frac{1}{3}$ and thus reach a contradiction.

For a hyperedge S we denote by $\mathbf{z}^S \in \{0,1\}^{kn}$ the encoding of S that is defined in the proof of Theorem 3. For $\mathbf{x} \in \{0,1\}^n$, let $P_{\mathbf{x}} : \{0,1\}^{kn} \to \{0,1\}$ be such that for every hyperedge S we have $P_{\mathbf{x}}(\mathbf{z}^S) = P(\mathbf{x}_S)$. We denote by $\mathcal{N}(0,1)$ the standard univariate normal distribution. Let c be a constant such that $\Pr_{t \sim \mathcal{N}(0,1)}[t \leq c] = \frac{1}{n}$. Let μ be the density of $\mathcal{N}(0,1)$, let $\mu_-(t) = n \cdot \mathbb{1}(t \leq c) \cdot \mu(t)$, and let $\mu_+(t) = \frac{n}{n-1} \cdot \mathbb{1}(t \geq c) \cdot \mu(t)$. Let $\Psi : \mathbb{R}^{kn} \to \{0,1\}^{kn}$ be a mapping such that for every $\mathbf{z}' \in \mathbb{R}^{kn}$ and $i \in [kn]$ we have $\Psi(\mathbf{z}')_i = 1$ iff $z'_i \geq c$. For $\tilde{\mathbf{z}} \in \mathbb{R}^{n^{1+3/\epsilon}}$ we denote $\tilde{\mathbf{z}}_{[kn]} = (\tilde{z}_1, \ldots, \tilde{z}_{kn})$, namely, the first kn component of $\tilde{\mathbf{z}}$.

Let $N_1 : \mathbb{R}^{kn} \to [0, 2^k]$ be a depth-3 neural network with at most $\frac{n^3}{3}$ hidden neurons (for a sufficiently large n), and no activation function in the output neuron, that satisfies the following property. Let $\mathbf{z}' \in \mathbb{R}^{kn}$ be such that $\Psi(\mathbf{z}') = \mathbf{z}^S$ for some hyperedge S, and assume that for every $i \in [kn]$ we have $z'_i \notin (c, c + \frac{1}{n^2})$, then $N_1(\mathbf{z}') = P_{\mathbf{x}}(\mathbf{z}^S)$. The construction of the network N_1 is given in Lemma 25. Let $N_2 : \mathbb{R}^{kn} \to \mathbb{R}_+$ be a depth-3 neural network with at most $\frac{n^3}{3}$ hidden neurons (for a sufficiently large n), and no activation function in the output neuron, that satisfies the following property. Let $\mathbf{z}' \in \mathbb{R}^{kn}$ be such that for every $i \in [kn]$ we have $z'_i \notin (c, c + \frac{1}{n^2})$. If $\Psi(\mathbf{z}')$ is an encoding of a hyperedge then $N_2(\mathbf{z}') = 0$, and otherwise $N_2(\mathbf{z}') \geq 2^k$. The construction of the network N_2 is given in Lemma 26. Let $N_3 : \mathbb{R}^{kn} \to \mathbb{R}_+$ be a depth-2 neural network with at most $\frac{n^3}{3}$ hidden neurons (for a sufficiently large n), such that for $\mathbf{z}' \in \mathbb{R}^{kn}$ we have: If there exists $i \in [kn]$ such that $z'_i \in (c, c + \frac{1}{n^2})$ then $N_3(\mathbf{z}') \geq 2^k$, and if for every $i \in [kn]$ we have $z'_i \notin (c - \frac{1}{n^2}, c + \frac{2}{n^2})$ then $N_3(\mathbf{z}') = 0$. The construction of the network N_3 is given in Lemma 27. Note that the network N_1 depends on $\mathbf{x}$, and the networks N_2, N_3 are independent of $\mathbf{x}$. Let N' be a depth-3 neural network such that for every $\mathbf{z}' \in \mathbb{R}^{kn}$ we have $N'(\mathbf{z}') = [N_1(\mathbf{z}') - N_2(\mathbf{z}') - N_3(\mathbf{z}')]_+$. The network N' has at most n^3 hidden neurons. We note that all weights in N' are bounded by some $\mathrm{poly}(n)$ that is independent of k, namely, using weights of magnitude n^k is not allowed. This is crucial since we need to show hardness of learning already where the weights of the network are bounded. Let $\tilde{N} : \mathbb{R}^{n^{1+3/\epsilon}} \to \mathbb{R}$ be a depth-3 neural network such that $\tilde{N}(\tilde{\mathbf{z}}) = N'(\tilde{\mathbf{z}}_{[kn]})$.

Given a sequence $(S_1, y_1), \ldots, (S_{n^s}, y_{n^s})$, where $S_1, \ldots, S_{n^s}$ are i.i.d. random hyperedges, the algorithm $\mathcal{A}$ needs to distinguish whether $\mathbf{y} = (y_1, \ldots, y_{n^s})$ is random or that we have $\mathbf{y} = (P(\mathbf{x}_{S_1}), \ldots, P(\mathbf{x}_{S_{n^s}})) = (P_{\mathbf{x}}(\mathbf{z}^{S_1}), \ldots, P_{\mathbf{x}}(\mathbf{z}^{S_{n^s}}))$ for a random $\mathbf{x} \in \{0,1\}^n$. We denote $\mathcal{S} = ((\mathbf{z}^{S_1}, y_1), \ldots, (\mathbf{z}^{S_{n^s}}, y_{n^s}))$.

We use the efficient algorithm $\mathcal{L}$ in order to obtain distinguishing advantage greater than $\frac{1}{3}$ as follows. The algorithm $\mathcal{A}$ runs $\mathcal{L}$ with the following examples oracle. In the i-th call, the oracle first draws $\mathbf{z} \in \{0,1\}^{kn}$ such that each component is drawn i.i.d. from a Bernoulli distribution where the probability of 0 is $\frac{1}{n}$. If $\mathbf{z}$ is an encoding of a hyperedge then the oracle replaces $\mathbf{z}$ with $\mathbf{z}^{S_i}$. Then, the oracle chooses $\mathbf{z}' \in \mathbb{R}^{kn}$ such that for each component j, if $z_j \geq c$ then z'_j is drawn from μ_+, and otherwise z'_j is drawn from μ_-. Let $\tilde{\mathbf{z}} \in \mathbb{R}^{n^{1+3/\epsilon}}$ be such that $\tilde{\mathbf{z}}_{[kn]} = \mathbf{z}'$, and the other $n^{1+3/\epsilon} - kn$ components of $\tilde{\mathbf{z}}$ are drawn i.i.d. from $\mathcal{N}(0,1)$. Note that the vector $\tilde{\mathbf{z}}$ has the distribution $\mathcal{D}$, due to the definitions of the densities μ_+ and μ_-, and since replacing an encoding of a random hyperedge by an encoding of another random hyperedge does not change the distribution of $\mathbf{z}$. The oracle returns $(\tilde{\mathbf{z}}, \tilde{y})$, where the labels $\tilde{y}$ are chosen as follows:

- If $\Psi(\mathbf{z}')$ is not an encoding of a hyperedge, then $\tilde{y} = 0$.
- If $\Psi(\mathbf{z}')$ is an encoding of a hyperedge:
  - If $\mathbf{z}'$ does not have components in the interval $(c - \frac{1}{n^2}, c + \frac{2}{n^2})$, then $\tilde{y} = y_i$.
  - If $\mathbf{z}'$ has a component in the interval $(c, c + \frac{1}{n^2})$, then $\tilde{y} = 0$.
  - If $\mathbf{z}'$ does not have components in the interval $(c, c + \frac{1}{n^2})$, but has a component in the interval $(c - \frac{1}{n^2}, c + \frac{2}{n^2})$, then $\tilde{y} = [y_i - N_3(\mathbf{z}')]_+$.

Let h be the hypothesis returned by $\mathcal{L}$. Recall that $\mathcal{L}$ uses at most $m(n)$ examples, and hence $\mathcal{S}$ contains at least n^3 examples that $\mathcal{L}$ cannot view. We denote the indices of these examples by $I = \{m(n) + 1, \ldots, m(n) + n^3\}$, and the examples by $\mathcal{S}_I = \{(\mathbf{z}^{S_i}, y_i)\}_{i \in I}$. By n^3 additional calls to the oracle, the algorithm $\mathcal{A}$ obtains the examples $\tilde{\mathcal{S}}_I = \{(\tilde{\mathbf{z}}_i, \tilde{y}_i)\}_{i \in I}$ that correspond to $\mathcal{S}_I$. Let $\ell_I(h) = \frac{1}{|I|} \sum_{i \in I} (h(\tilde{\mathbf{z}}_i) - \tilde{y}_i)^2$. Now, if $\ell_I(h) \leq \frac{2}{n}$, then $\mathcal{A}$ returns 1, and otherwise it returns 0. Clearly, the algorithm $\mathcal{A}$ runs in polynomial time. We now show that if $\mathcal{S}$ is pseudorandom then $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$, and if $\mathcal{S}$ is random then $\mathcal{A}$ returns 1 with probability less than $\frac{1}{3}$.

In Lemma 28, we show that if $\mathcal{S}$ is pseudorandom then the examples $(\tilde{\mathbf{z}}, \tilde{y})$ returned by the oracle are realized by $\tilde{N}$. Hence, with probability at least $\frac{3}{4}$ the algorithm $\mathcal{L}$ returns a hypothesis h such that $\mathbb{E}_{\tilde{\mathbf{z}} \sim \mathcal{D}} (h(\tilde{\mathbf{z}}) - \tilde{N}(\tilde{\mathbf{z}}))^2 \leq \frac{1}{n}$. Therefore, $\mathbb{E}_{\tilde{\mathcal{S}}_I} \ell_I(h) \leq \frac{1}{n}$.

Let $\tilde{\mathcal{Z}} \subseteq \mathbb{R}^{n^{(1+3/\epsilon)}}$ be such that $\tilde{\mathbf{z}} \in \tilde{\mathcal{Z}}$ if $\tilde{\mathbf{z}}_{[kn]}$ does not have components in the interval $(c - \frac{1}{n^2}, c + \frac{2}{n^2})$, and $\Psi(\tilde{\mathbf{z}}_{[kn]}) = \mathbf{z}^S$ for a hyperedge S. If $\mathcal{S}$ is random, then for every i such that $\tilde{\mathbf{z}}_i \in \tilde{\mathcal{Z}}$, we have $\tilde{y}_i = 1$ w.p. $\frac{1}{2}$ and $\tilde{y}_i = 0$ otherwise. Also, by the definition of the oracle, $\tilde{y}_i$ is independent of S_i and independent of the choice of the vector $\tilde{\mathbf{z}}_i$ that corresponds to $\mathbf{z}^{S_i}$. Hence, for every h and $i \in I$ we have

$$\Pr\left[(h(\tilde{\mathbf{z}}_i) - \tilde{y}_i)^2 \geq \frac{1}{4}\right] \geq \Pr\left[(h(\tilde{\mathbf{z}}_i) - \tilde{y}_i)^2 \geq \frac{1}{4} \;\middle|\; \tilde{\mathbf{z}}_i \in \tilde{\mathcal{Z}}\right] \cdot \Pr\left[\tilde{\mathbf{z}}_i \in \tilde{\mathcal{Z}}\right] \geq \frac{1}{2} \cdot \Pr\left(\tilde{\mathbf{z}}_i \in \tilde{\mathcal{Z}}\right) .$$

In Lemma 29 we show that $\Pr\left[\tilde{\mathbf{z}}_i \in \tilde{\mathcal{Z}}\right] \geq \frac{1}{2\log(n)}$. Hence,

$$\Pr\left[(h(\tilde{\mathbf{z}}_i) - \tilde{y}_i)^2 \geq \frac{1}{4}\right] \geq \frac{1}{4\log(n)} .$$

Thus,

$$\mathop{\mathbb{E}}_{\tilde{\mathcal{S}}_I} \ell_I(h) \geq \frac{1}{4} \cdot \frac{1}{4\log(n)} = \frac{1}{16\log(n)} .$$

By Lemma 20 (with $c = 0$), we have for a sufficiently large n that

$$\Pr_{\tilde{\mathcal{S}}_I}\left[\left|\ell_I(h) - \mathop{\mathbb{E}}_{\tilde{\mathcal{S}}_I} \ell_I(h)\right| \geq \frac{1}{n}\right] < \frac{1}{20} .$$

Therefore, if $\mathcal{S}$ is pseudorandom, then for a sufficiently large n, we have with probability at least $1 - \left(\frac{1}{4} + \frac{1}{20}\right) = \frac{7}{10} > \frac{2}{3}$ that $\mathbb{E}_{\tilde{\mathcal{S}}_I} \ell_I(h) \leq \frac{1}{n}$ and $\left|\ell_I(h) - \mathbb{E}_{\tilde{\mathcal{S}}_I} \ell_I(h)\right| < \frac{1}{n}$, and hence $\ell_I(h) \leq \frac{2}{n}$. Thus, the algorithm $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$. If $\mathcal{S}$ is random then $\mathbb{E}_{\tilde{\mathcal{S}}} \ell_I(h) \geq \frac{1}{16\log(n)}$

and for a sufficiently large n we have with probability at least $\frac{19}{20}$ that $\left|\ell_I(h) - \mathbb{E}_{\tilde{\mathcal{S}}_I}\, \ell_I(h)\right| < \frac{1}{n}$. Hence, with probability greater than $\frac{2}{3}$ we have $\ell_I(h) > \frac{1}{16\log(n)} - \frac{1}{n} > \frac{2}{n}$ and the algorithm $\mathcal{A}$ returns 0.

Hence, it is hard to learn depth-3 neural networks with n^3 hidden neurons on the distribution $\mathcal{D}$. Thus, for $\tilde{n} = n^{1+3/\epsilon}$, we have that it is hard to learn depth-3 neural networks with $\tilde{n}^\epsilon = n^{(1+3/\epsilon)\cdot\epsilon} = n^{\epsilon+3} \geq n^3$ hidden neurons on a standard Gaussian distribution over $\mathbb{R}^{\tilde{n}}$.

Lemma 25 *There exists a depth-3 neural network $N_1 : \mathbb{R}^{kn} \to [0, 2^k]$ with at most $2kn+2^k$ hidden neurons and no activation function in the output neuron, that satisfies the following property. Let $\mathbf{z}' \in \mathbb{R}^{kn}$ be such that $\Psi(\mathbf{z}') = \mathbf{z}^S$ for some hyperedge S, and assume that for every $i \in [kn]$ we have $z'_i \notin (c, c+\frac{1}{n^2})$, then $N_1(\mathbf{z}') = P_{\mathbf{x}}(\mathbf{z}^S)$.*

Proof Let $N_{\mathbf{x}}$ be the depth-2 neural network from the proof of Theorem 8 (part 1). The network $N_{\mathbf{x}}$ is such that for every hyperedge S, we have $N_{\mathbf{x}}(\mathbf{z}^S) = P_{\mathbf{x}}(\mathbf{z}^S)$. Also, the network $N_{\mathbf{x}}$ is such that for every $\mathbf{z} \in \mathbb{R}^{kn}$, we have

$$N_{\mathbf{x}}(\mathbf{z}) = \sum_{1 \leq j \leq J} \left[\left(\sum_{l \in I_j} z_l \right) - (|I_j| - 1) \right]_+ ,$$

where $J \leq 2^k$, and $I_j \subseteq [kn]$. Therefore, for every $\mathbf{z} \in [0,1]^{kn}$ we have $N_{\mathbf{x}}(\mathbf{z}) \in [0, 2^k]$.

Next, we construct a depth-2 neural network $N_\Psi : \mathbb{R}^{kn} \to [0,1]^{kn}$ with a single layer of non-linearity, such that for every $\mathbf{z}' \in \mathbb{R}^{kn}$ with $z'_i \notin (c, c+\frac{1}{n^2})$ for every $i \in [kn]$, we have $N_\Psi(\mathbf{z}') = \Psi(\mathbf{z}')$. The network N_Ψ has $2kn$ hidden neurons, and computes $N_\Psi(\mathbf{z}') = (f(z'_1), \ldots, f(z'_{kn}))$, where $f : \mathbb{R} \to [0,1]$ is such that

$$f(t) = n^2 \cdot \left([t-c]_+ - \left[t - \left(c + \frac{1}{n^2} \right) \right]_+ \right) .$$

Note that if $t \leq c$ then $f(t) = 0$, if $t \geq c + \frac{1}{n^2}$ then $f(t) = 1$, and if $c < t < c + \frac{1}{n^2}$ then $f(t) \in (0,1)$.

The network N_1 is obtained by combining the networks N_Ψ and $N_{\mathbf{x}}$. Note that N_1 has at most $2kn + 2^k$ hidden neurons, and satisfies the requirements. ■

Lemma 26 *There exists a depth-3 neural network $N_2 : \mathbb{R}^{kn} \to \mathbb{R}_+$ with at most $2kn + k \cdot \frac{n(n-1)}{2} + k + n \cdot \frac{k(k-1)}{2}$ hidden neurons, and no activation function in the output neuron, that satisfies the following property. Let $\mathbf{z}' \in \mathbb{R}^{kn}$ be such that for every $i \in [kn]$ we have $z'_i \notin (c, c+\frac{1}{n^2})$. If $\Psi(\mathbf{z}')$ is an encoding of a hyperedge then $N_2(\mathbf{z}') = 0$, and otherwise $N_2(\mathbf{z}') \geq 2^k$.*

Proof By the proof of Lemma 21, there is a DNF formula φ over $\{0,1\}^{kn}$ with $k \cdot \frac{n(n-1)}{2} + k + n \cdot \frac{k(k-1)}{2}$ terms such that $\varphi(\mathbf{z}) = 1$ iff $\mathbf{z}$ is not an encoding of a hyperedge. Each term in φ can be implemented by a single ReLU neuron. By summing the outputs of these neurons and multiplying by 2^k we obtain a depth-2 neural network N_φ, such that if $\mathbf{z} \in \{0,1\}^{kn}$ is an encoding of a hyperedge then $N_\varphi(\mathbf{z}) = 0$, and otherwise $N_\varphi(\mathbf{z}) \geq 2^k$. For every $\mathbf{z} \in \mathbb{R}^{kn}$ we have $N_\varphi(\mathbf{z}) \geq 0$.

Let $N_\Psi : \mathbb{R}^{kn} \to [0,1]^{kn}$ be the depth-2 neural network from the proof of Lemma 25, with a single layer of non-linearity of $2kn$ hidden neurons, such that for every $\mathbf{z}' \in \mathbb{R}^{kn}$ with $z'_i \notin (c, c+\frac{1}{n^2})$ for every $i \in [kn]$, we have $N_\Psi(\mathbf{z}') = \Psi(\mathbf{z}')$. By combining N_Ψ and N_φ we obtain a depth-3 network N_2 with $2kn + k \cdot \frac{n(n-1)}{2} + k + n \cdot \frac{k(k-1)}{2}$ hidden neurons that satisfies the requirements. ■

Lemma 27 *There exists a depth-2 neural network $N_3 : \mathbb{R}^{kn} \to \mathbb{R}_+$ with at most $4kn$ hidden neurons, such that for $\mathbf{z}' \in \mathbb{R}^{kn}$ we have: If there exists $i \in [kn]$ such that $z'_i \in (c, c+\frac{1}{n^2})$ then $N_3(\mathbf{z}') \geq 2^k$, and if for every $i \in [kn]$ we have $z'_i \notin (c-\frac{1}{n^2}, c+\frac{2}{n^2})$ then $N_3(\mathbf{z}') = 0$.*

Proof We construct a depth-2 network $N_3 : \mathbb{R}^{kn} \to [0, 2^k \cdot kn]$ with $4kn$ hidden neurons, such that $N_3(\mathbf{z}') = 2^k \cdot \sum_{i\in[kn]} m_i$, where

- If $z'_i \in (c, c+\frac{1}{n^2})$ then $m_i = 1$.
- If $z'_i \notin (c-\frac{1}{n^2}, c+\frac{2}{n^2})$ then $m_i = 0$.
- If $z'_i \in (c-\frac{1}{n^2}, c]$ then $m_i = \left(z'_i - c + \frac{1}{n^2}\right) \cdot n^2 \in [0,1]$.
- If $z'_i \in [c+\frac{1}{n^2}, c+\frac{2}{n^2})$ then $m_i = 1 - \left(z'_i - c - \frac{1}{n^2}\right) \cdot n^2 \in [0,1]$.

The construction now follows immediately from the fact that for every $i \in [kn]$ we have

$$\begin{aligned} m_i =& (n^2)\left(\left[z'_i - \left(c - \frac{1}{n^2}\right)\right]_+ - \left[z'_i - c\right]_+\right) - \\ & (n^2)\left(\left[z'_i - \left(c + \frac{1}{n^2}\right)\right]_+ - \left[z'_i - \left(c + \frac{2}{n^2}\right)\right]_+\right) . \end{aligned}$$

■

Lemma 28 *If $\mathcal{S}$ is pseudorandom then the examples $(\tilde{\mathbf{z}}, \tilde{y})$ returned by the oracle are realized by $\tilde{N}$.*

Proof Let $\mathbf{z}' = \tilde{\mathbf{z}}_{[kn]}$. Thus, $\tilde{N}(\tilde{\mathbf{z}}) = N'(\mathbf{z}')$. We show that $\tilde{y} = N'(\mathbf{z}')$.

- If $\Psi(\mathbf{z}')$ is not an encoding of a hyperedge, then:
  - If $\mathbf{z}'$ does not have components in the interval $(c, c+\frac{1}{n^2})$, then $N_1(\mathbf{z}') \in [0, 2^k]$, $N_2(\mathbf{z}') \geq 2^k$, and $N_3(\mathbf{z}') \geq 0$. Therefore, $N'(\mathbf{z}') = 0 = \tilde{y}$.
  - If $\mathbf{z}'$ has a component in the interval $(c, c+\frac{1}{n^2})$, then $N_1(\mathbf{z}') \in [0, 2^k]$, $N_2(\mathbf{z}') \geq 0$, and $N_3(\mathbf{z}') \geq 2^k$. Therefore, $N'(\mathbf{z}') = 0 = \tilde{y}$.
- If $\Psi(\mathbf{z}')$ is an encoding of a hyperedge S, then:
  - If $\mathbf{z}'$ does not have components in the interval $(c-\frac{1}{n^2}, c+\frac{2}{n^2})$, then $N_1(\mathbf{z}') = P_{\mathbf{x}}(\mathbf{z}^S)$, $N_2(\mathbf{z}') = N_3(\mathbf{z}') = 0$. Therefore, $N'(\mathbf{z}') = P_{\mathbf{x}}(\mathbf{z}^S) = \tilde{y}$.

- If $\mathbf{z}'$ has a component in the interval $(c, c+\frac{1}{n^2})$, then $N_1(\mathbf{z}') \in [0, 2^k]$, $N_2(\mathbf{z}') \geq 0$, and $N_3(\mathbf{z}') \geq 2^k$. Therefore, $N'(\mathbf{z}') = 0 = \tilde{y}$.
- If $\mathbf{z}'$ does not have components in the interval $(c, c+\frac{1}{n^2})$, but has a component in the interval $(c - \frac{1}{n^2}, c + \frac{2}{n^2})$, then $N_1(\mathbf{z}') = P_{\mathbf{x}}(\mathbf{z}^S)$ and $N_2(\mathbf{z}') = 0$. Therefore, $N'(\mathbf{z}') = [P_{\mathbf{x}}(\mathbf{z}^S) - N_3(\mathbf{z}')]_+ = \tilde{y}$.

■

Lemma 29 *Let $\tilde{\mathbf{z}} \in \mathbb{R}^{n^{1+3/\epsilon}}$ be the vector returned by the oracle. We have*

$$\Pr\left[\tilde{\mathbf{z}} \in \tilde{\mathcal{Z}}\right] \geq \frac{1}{2\log(n)} .$$

Proof Let $\mathbf{z}' = \tilde{\mathbf{z}}_{[kn]}$. We have

$$\begin{aligned}&\Pr\left[\tilde{\mathbf{z}} \in \tilde{\mathcal{Z}}\right] \\ &\quad = \Pr\left[\mathbf{z}' \text{ does not have components in } \left(c - \frac{1}{n^2}, c + \frac{2}{n^2}\right) \Bigg| \Psi(\mathbf{z}') \text{ represents a hyperedge}\right] \cdot \\ &\quad\quad \Pr\left[\Psi(\mathbf{z}') \text{ represents a hyperedge}\right] .\end{aligned}$$

Let $\mathbf{z} = \Psi(\mathbf{z}')$. By the definition of the oracle, the probability that $\mathbf{z}$ is an encoding of a hyperedge, equals to the probability that a random vector whose components are drawn i.i.d. from the Bernoulli distribution encodes a hyperedge. In the proof of Lemma 21, we showed that the probability that such vector is an encoding of a hyperedge is at least $\frac{1}{\log(n)}$. Thus, it remains to show that

$$\Pr\left[\mathbf{z}' \text{ does not have components in } \left(c - \frac{1}{n^2}, c + \frac{2}{n^2}\right) \Bigg| \mathbf{z} \text{ represents a hyperedge}\right] \geq \frac{1}{2} .$$

Note that the density μ_- is bounded by $\frac{n}{2\pi}$, and that μ_+ is bounded by $\frac{n}{(n-1)2\pi}$. Hence, for a sufficiently large n, we have

$$\begin{aligned}&\Pr\left[\mathbf{z}' \text{ has a component in } \left(c - \frac{1}{n^2}, c + \frac{2}{n^2}\right) \Bigg| \mathbf{z} \text{ represents a hyperedge}\right] \\ &\quad \leq k \cdot \frac{1}{n^2} \cdot \frac{n}{2\pi} + (nk - k) \cdot \frac{2}{n^2} \cdot \frac{n}{(n-1)2\pi} = \frac{k}{2\pi n} + \frac{k}{\pi n} \leq \frac{1}{2} .\end{aligned}$$

■

A.7. Proof of Theorem 12

Let $\mathcal{D}$ be the uniform distribution on $\{0,1\}^{n^{1+3/\epsilon}}$. Let $c' = c\left(1 + \frac{3}{\epsilon}\right)$. Assume that there is an efficient algorithm $\mathcal{L}$ that learns DFAs with n^3 states on the distribution $\mathcal{D}$. Let $m(n)$ be a polynomial such that $\mathcal{L}$ uses a sample of size at most $m(n)$ and returns with probability at least $\frac{3}{4}$ a hypothesis h with error at most $\frac{1}{2} - \frac{1}{n^{c'}}$. Let $s > 1$ be a constant such that $n^s \geq m(n) + n^{2c'+3}$ for every sufficiently large n. By Assumption 1, there exists a constant k and a predicate $P : \{0,1\}^k \to \{0,1\}$, such that $\mathcal{F}_{P,n,n^s}$ is $\frac{1}{3}$-PRG. We will show an algorithm $\mathcal{A}$ with distinguishing advantage greater than $\frac{1}{3}$ and thus reach a contradiction.

For a hyperedge $S = (i_1, \ldots, i_k)$, we denote by $\mathbf{z}^S \in \{0,1\}^{nk}$ an encoding of S, which consists of n slices of size k, where the j-th bit in the l-th slice is 1 iff $l = i_j$, namely, if the index l is the j-th member in S. We call $\mathbf{z}^S$ the *short encoding* of S. For $\mathbf{z} \in \{0,1\}^{nk}$, we index the coordinates by $[n] \times [k]$, thus $z_{l,j} = z_{(l-1)k+j}$. For $\tilde{\mathbf{z}} \in \{0,1\}^{nk\log(n)}$, we index the coordinates by $[n] \times [k] \times [\log(n)]$, thus, $\tilde{z}_{l,j,i} = \tilde{z}_{(l-1)(k\log(n))+(j-1)\log(n)+i}$. Let $\Psi : \{0,1\}^{nk\log(n)} \to \{0,1\}^{nk}$ be a mapping, such that $\Psi(\tilde{\mathbf{z}})_{l,j} = 1$ iff $\tilde{z}_{l,j,i} = 1$ for every $i \in [\log(n)]$. If $\tilde{\mathbf{z}} \in \{0,1\}^{nk\log(n)}$ is such that $\Psi(\tilde{\mathbf{z}}) = \mathbf{z}^S$ for a hyperedge S, then we say that $\tilde{\mathbf{z}}$ is a *long encoding* of S. Note that a hyperedge S has a single short encoding $\mathbf{z}^S$, but many long encodings, since every 0-bit in $\mathbf{z}^S$ can be represented in the long encoding by any vector in the set $B = \{0,1\}^{\log(n)} \setminus \{(1,\ldots,1)\}$. Hence, given S, a random long encoding of S can be obtained by replacing every 1-bit in $\mathbf{z}^S$ by the size-$\log(n)$ vector $(1,\ldots,1)$, and replacing every 0-bit by a random vector from B.

Let $\mathbf{z} \in \{0,1\}^{c'\log^2(n)\cdot nk}$ be a vector that consists of $c'\log^2(n)$ slices of size nk. If $\mathbf{z}$ has a size-nk slice that is a short encoding of a hyperedge S, and all preceding size-nk slices do not encode hyperedges, then we say that $\mathbf{z}$ is a *multi-short encoding* of S. Note that if all $c'\log^2(n)$ slices do not encode hyperedges then $\mathbf{z}$ is not a multi-short encoding of any hyperedge. For $\mathbf{z} \in \{0,1\}^{c'\log^2(n)nk}$, we index the coordinates by $[c'\log^2(n)] \times [n] \times [k]$, thus $z_{d,l,j} = z_{(d-1)nk+(l-1)k+j}$. For $\tilde{\mathbf{z}} \in \{0,1\}^{c'\log^2(n)nk\cdot\log(n)}$, we index the coordinates by $[c'\log^2(n)] \times [n] \times [k] \times [\log(n)]$, thus, $\tilde{z}_{d,l,j,i} = \tilde{z}_{(d-1)nk\log(n)+(l-1)(k\log(n))+(j-1)\log(n)+i}$. Let $\Psi' : \{0,1\}^{c'\log^2(n)nk\log(n)} \to \{0,1\}^{c'\log^2(n)nk}$ be a mapping, such that $\Psi'(\tilde{\mathbf{z}})_{d,l,j} = 1$ iff $\tilde{z}_{d,l,j,i} = 1$ for every $i \in [\log(n)]$. Thus, $\Psi'(\tilde{\mathbf{z}})$ is obtained by applying Ψ to every size-$nk\log(n)$ slice in $\tilde{\mathbf{z}}$. If $\tilde{\mathbf{z}} \in \{0,1\}^{c'\log^2(n)nk\log(n)}$ is such that $\Psi'(\tilde{\mathbf{z}})$ is a multi-short encoding of a hyperedge S, then we say that $\tilde{\mathbf{z}}$ is a *multi-long encoding* of S. Note that a hyperedge S has a single short encoding $\mathbf{z}^S$, but many multi-short encodings. Also, each multi-short encoding corresponds to many multi-long encodings. We say that $\tilde{\mathbf{z}} \in \{0,1\}^{n^{1+3/\epsilon}}$ is an *extended multi-long encoding* of a hyperedge S, if $(\tilde{z}_1, \ldots, \tilde{z}_{c'\log^3(n)nk})$ is a multi-long encoding of S, namely, $\tilde{\mathbf{z}}$ starts with a multi-long encoding of S. We assume that $n^{1+3/\epsilon} \geq c'\log^3(n)nk$. For $\mathbf{x} \in \{0,1\}^n$, let $P_{\mathbf{x}} : \{0,1\}^{n^{1+3/\epsilon}} \to \{0,1\}$ be such that for every hyperedge S, if $\tilde{\mathbf{z}}$ is an extended multi-long encoding of S, then $P_{\mathbf{x}}(\tilde{\mathbf{z}}) = P(\mathbf{x}_S)$.

Let $\tilde{\mathbf{z}} \in \{0,1\}^{nk\log(n)}$ be a random vector drawn from the uniform distribution. The probability that $\tilde{\mathbf{z}}$ is a long encoding of a hyperedge is

$$n \cdot (n-1) \cdot \ldots \cdot (n-k+1) \cdot \left(\left(\frac{1}{2}\right)^{\log(n)} \right)^k \left(1 - \left(\frac{1}{2}\right)^{\log(n)}\right)^{nk-k}$$
$$\geq \left(\frac{n-k}{n}\right)^k \left(1 - \frac{1}{n}\right)^{k(n-1)} = \left(1 - \frac{k}{n}\right)^k \left(1 - \frac{1}{n}\right)^{k(n-1)} .$$

Since for every $x \in (0,1)$ we have $e^{-x} < 1 - \frac{x}{2}$ then for a sufficiently large n the above is at least

$$\exp\left(-\frac{2k^2}{n}\right) \cdot \exp\left(-\frac{2k(n-1)}{n}\right) \geq \exp(-1) \cdot \exp(-2k) \geq \frac{1}{\log(n)} .$$

Hence, the probability that $\tilde{\mathbf{z}} \sim \mathcal{D}$ is an extended multi-long encoding of a hyperedge is at least

$$1 - \left(1 - \frac{1}{\log(n)}\right)^{c' \log^2(n)} \geq 1 - \exp\left(-\frac{1}{\log(n)} \cdot c' \log^2(n)\right) \geq 1 - \exp\left(-c' \ln(n)\right) = 1 - \frac{1}{n^{c'}} . \tag{3}$$

Given a sequence $\mathcal{S} = (S_1, y_1), \dots, (S_{n^s}, y_{n^s})$, where $S_1, \dots, S_{n^s}$ are i.i.d. random hyperedges, the algorithm $\mathcal{A}$ needs to distinguish whether $\mathbf{y} = (y_1, \dots, y_{n^s})$ is random or that $\mathbf{y} = (P(\mathbf{x}_{S_1}), \dots, P(\mathbf{x}_{S_{n^s}}))$ for a random $\mathbf{x} \in \{0,1\}^n$. We use the efficient algorithm $\mathcal{L}$ in order to obtain distinguishing advantage greater than $\frac{1}{3}$ as follows. The algorithm $\mathcal{A}$ runs $\mathcal{L}$ with the following examples oracle. In the i-th call to the oracle, it chooses $\tilde{\mathbf{z}}_i \in \{0,1\}^{n^{1+3/\epsilon}}$ according to $\mathcal{D}$. If $\tilde{\mathbf{z}}_i$ is not an extended multi-long encoding of a hyperedge (with probability at most $\frac{1}{n^{c'}}$, by Eq. 3), then the oracle returns $(\mathbf{z}'_i, y'_i)$ where $\mathbf{z}'_i = \tilde{\mathbf{z}}_i$ and $y'_i = 0$. Otherwise, the oracle chooses a random long encoding $\tilde{\mathbf{z}}^{S_i}$ of S_i, obtains $\mathbf{z}'_i$ by replacing the first size-$nk\log(n)$ slice in $\tilde{\mathbf{z}}_i$ that encodes a hyperedge with $\tilde{\mathbf{z}}^{S_i}$, and returns $(\mathbf{z}'_i, y'_i)$ where $y'_i = y_i$. Note that the vector $\mathbf{z}'_i$ returned by the oracle has the distribution $\mathcal{D}$, since replacing a random long encoding of a random hyperedge with a random long encoding of another random hyperedge does not change the distribution (see Lemma 30 for a more formal proof). Let h be the hypothesis returned by $\mathcal{L}$. Recall that $\mathcal{L}$ uses at most $m(n)$ examples, and hence $\mathcal{S}$ contains at least $n^{2c'+3}$ examples that $\mathcal{L}$ cannot view. We denote the indices of these examples by $I = \{m(n)+1, \dots, m(n)+n^{2c'+3}\}$, and denote $\mathcal{S}_I = \{(S_i, y_i)\}_{i \in I}$. By $n^{2c'+3}$ additional calls to the oracle, the algorithm $\mathcal{A}$ obtains the examples $\mathcal{S}'_I = \{(\mathbf{z}'_i, y'_i)\}_{i \in I}$ that correspond to $\mathcal{S}_I$. Let $\ell_I(h) = \frac{1}{|I|}\sum_{i \in I} \mathbb{1}(h(\mathbf{z}'_i) \neq y'_i)$. Now, if $\ell_I(h) \leq \frac{1}{2} - \frac{3}{4n^{c'}}$, then $\mathcal{A}$ returns 1, and otherwise it returns 0. Clearly, the algorithm $\mathcal{A}$ runs in polynomial time. We now show that if $\mathcal{S}$ is pseudorandom then $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$, and if $\mathcal{S}$ is random then $\mathcal{A}$ returns 1 with probability less than $\frac{1}{3}$.

If $\mathcal{S}$ is pseudorandom, then by Lemma 33, the examples $(\mathbf{z}'_i, y'_i)$ returned by the oracle satisfy $y'_i = A(\mathbf{z}'_i)$, where A is a DFA with at most n^3 states. Indeed, if $\mathbf{z}'_i$ is an extended multi-long encoding of a hyperedge S_i then $y'_i = P(\mathbf{x}_{S_i}) = P_{\mathbf{x}}(\mathbf{z}'_i) = A(\mathbf{z}'_i)$, and otherwise $y'_i = A(\mathbf{z}'_i) = 0$. Hence, if $\mathcal{S}$ is pseudorandom then with probability at least $\frac{3}{4}$ the algorithm $\mathcal{L}$ returns a hypothesis h such that $\mathbb{E}_{\tilde{\mathbf{z}} \sim \mathcal{D}} \mathbb{1}(h(\tilde{\mathbf{z}}) \neq A(\tilde{\mathbf{z}})) \leq \frac{1}{2} - \frac{1}{n^{c'}}$. Therefore, $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \leq \frac{1}{2} - \frac{1}{n^{c'}}$.

If $\mathcal{S}$ is random, then for the indices i such that $\mathbf{z}'_i$ is an extended multi-long encoding of a hyperedge, the labels y'_i are independent uniform Bernoulli random variables. Hence, for every h and $i \in I$ we have

$$\begin{aligned} \Pr\left[h(\mathbf{z}'_i) \neq y'_i\right] &\geq \Pr\left[h(\mathbf{z}'_i) \neq y'_i \mid \mathbf{z}'_i \text{ represents a hyperedge}\right] \cdot \Pr\left[\mathbf{z}'_i \text{ represents a hyperedge}\right] \\ &\overset{(Eq.\ 3)}{\geq} \frac{1}{2} \cdot \left(1 - \frac{1}{n^{c'}}\right) = \frac{1}{2} - \frac{1}{2n^{c'}} . \end{aligned}$$

Thus, $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \geq \frac{1}{2} - \frac{1}{2n^{c'}}$.

By Lemma 20, for a sufficiently large n, we have

$$\Pr_{\mathcal{S}'_I}\left[\left|\ell_I(h) - \mathbb{E}_{\mathcal{S}'_I} \ell_I(h)\right| \geq \frac{1}{4n^{c'}}\right] \leq \Pr_{\mathcal{S}'_I}\left[\left|\ell_I(h) - \mathbb{E}_{\mathcal{S}'_I} \ell_I(h)\right| \geq \frac{1}{n^{c'+1}}\right] < \frac{1}{20} .$$

Therefore, if $\mathcal{S}$ is pseudorandom, then for a sufficiently large n, we have with probability at least $1 - \left(\frac{1}{4} + \frac{1}{20}\right) = \frac{7}{10} > \frac{2}{3}$ that $\mathbb{E}_{\mathcal{S}'_I} \ell_I(h) \leq \frac{1}{2} - \frac{1}{n^{c'}}$ and $\left|\ell_I(h) - \mathbb{E}_{\mathcal{S}'_I} \ell_I(h)\right| < \frac{1}{4n^{c'}}$, and hence

$\ell_I(h) \leq \frac{1}{2} - \frac{3}{4n^{c'}}$. Thus, the algorithm $\mathcal{A}$ returns 1 with probability greater than $\frac{2}{3}$. If $\mathcal{S}$ is random then $\mathbb{E}_{\mathcal{S}'}\, \ell_I(h) \geq \frac{1}{2} - \frac{1}{2n^{c'}}$ and for a sufficiently large n we have with probability at least $\frac{19}{20}$ that $\left|\ell_I(h) - \mathbb{E}_{\mathcal{S}'_I}\, \ell_I(h)\right| < \frac{1}{4n^{c'}}$. Hence, with probability greater than $\frac{2}{3}$ we have $\ell_I(h) > \frac{1}{2} - \frac{3}{4n^{c'}}$ and the algorithm $\mathcal{A}$ returns 0.

Hence, it is hard to learn DFAs with n^3 states and error at most $\frac{1}{2} - \frac{1}{n^{c'}}$, where the input distribution is $\mathcal{D}$. Thus, for $\tilde{n} = n^{1+3/\epsilon}$, we have that it is hard to learn DFAs with $\tilde{n}^\epsilon = n^{(1+3/\epsilon)\cdot\epsilon} = n^{\epsilon+3} \geq n^3$ states and error at most $\frac{1}{2} - \frac{1}{\tilde{n}^c} = \frac{1}{2} - \frac{1}{n^{(1+3/\epsilon)\cdot c}} = \frac{1}{2} - \frac{1}{n^{c'}}$, on the uniform distribution over $\{0,1\}^{\tilde{n}}$.

Lemma 30 *The distribution of an example $\mathbf{z}' \in \{0,1\}^{n^{1+3/\epsilon}}$ returned by the oracle is $\mathcal{D}$.*

Proof Let $\tilde{n} = n^{1+3/\epsilon}$. Recall that the oracle first chooses $\tilde{\mathbf{z}} \in \{0,1\}^{\tilde{n}}$ according to $\mathcal{D}$. If $\tilde{\mathbf{z}}$ is not an extended multi-long encoding of a hyperedge then $\mathbf{z}' = \tilde{\mathbf{z}}$. Otherwise, the oracle chooses a random long encoding $\tilde{\mathbf{z}}^S$ of a random hyperedge S, and obtains $\mathbf{z}'$ by replacing the first size-$nk\log(n)$ slice in $\tilde{\mathbf{z}}$ that encodes a hyperedge with $\tilde{\mathbf{z}}^S$.

Let $\mathbf{z}^0 \in \{0,1\}^{\tilde{n}}$. We show that the probability of $\mathbf{z}' = \mathbf{z}^0$ is $\frac{1}{2^{\tilde{n}}}$. If $\mathbf{z}^0$ is not an extended multi-long encoding of a hyperedge, then $\Pr[\mathbf{z}' = \mathbf{z}^0] = \Pr[\tilde{\mathbf{z}} = \mathbf{z}^0] = \frac{1}{2^{\tilde{n}}}$. Assume that $\mathbf{z}^0$ is an extended multi-long encoding of a hyperedge S_0, and the first size-$nk\log(n)$ slice in $\mathbf{z}^0$ that encodes a hyperedge is the d-th slice, for some $d \in [c'\log^2(n)]$. Thus, $z^0_{(d-1)nk\log(n)+1}, \ldots, z^0_{dnk\log(n)}$ is a long encoding of S_0. Let $B_0 \subseteq \{0,1\}^{\tilde{n}}$ be the set of all extended multi-long encodings that can be obtained from $\mathbf{z}^0$ by replacing the d-th size-$nk\log(n)$ slice with some long encoding of some hyperedge. Note that $\mathbf{z}' = \mathbf{z}^0$ iff the oracle chooses $\tilde{\mathbf{z}} \in B_0$, and then chooses $S = S_0$, and then chooses the long encoding $\tilde{\mathbf{z}}^S = z^0_{(d-1)nk\log(n)+1}, \ldots, z^0_{dnk\log(n)}$. For every $\mathbf{z} \in B_0$, by replacing the d-th size-$nk\log(n)$ slice with a random long encoding of a random hyperedge, we obtain a random (uniformly distributed) vector in B_0. Hence, $\mathbf{z}' = \mathbf{z}^0$ iff we have: (1) the oracle first chooses $\tilde{\mathbf{z}} \in B_0$, (2) the oracle chooses $\mathbf{z}^0$ as the random vector in B_0. Therefore, we have

$$\Pr\left[\mathbf{z}' = \mathbf{z}^0\right] = \frac{|B_0|}{2^{\tilde{n}}} \cdot \frac{1}{|B_0|} = \frac{1}{2^{\tilde{n}}} .$$

■

Lemma 31 *For a sufficiently large n, there exists a DFA A_E with at most $\log(n)$ states such that A_E accepts a word $\mathbf{z} \in \{0,1\}^{nk}$ iff $\mathbf{z}$ is a short encoding of a hyperedge.*

Proof A word $\mathbf{z} \in \{0,1\}^{nk}$ is a short encoding of a hyperedge iff the following conditions hold:

- Every size-k slice in $\mathbf{z}$ includes at most one 1-bit.
- There are no two size-k slices in $\mathbf{z}$ that have 1-bit in the same index (and thus correspond to the same member in the hyperedge).
- For every $j \in [k]$ there is a size-k slice in $\mathbf{z}$ with 1-bit in index j.

We construct a DFA $A_E = \langle \Sigma, Q, q_0, \delta, F \rangle$ that checks these conditions. We have $\Sigma = \{0,1\}$, $Q = \{q_{\text{rej}}\} \cup ([k] \times \{0,1\} \times 2^{[k]})$, $q_0 = (1,0,\emptyset)$, and $F = \{(1,0,[k])\}$. Note that Q is of size at most $\log(n)$ (for a sufficiently large n). The states in Q are such that the first component keeps the current location in the size-k slice, the second component keeps whether a 1-bit already appeared in the current slice, and the third component keeps the subset of indices in $[k]$ that are already occupied. For $i \in [k-1]$, $b \in \{0,1\}$ and $I \subseteq [k]$, we have

- $\delta((i,b,I),0) = (i+1,b,I)$.
- $\delta((k,b,I),0) = (1,0,I)$.
- $\delta((i,1,I),1) = \delta((k,1,I),1) = q_{\text{rej}}$.
- $\delta((i,0,I),1) = (i+1,1,I \cup \{i\})$ if $i \not\in I$, and $\delta((i,0,I),1) = q_{\text{rej}}$ otherwise.
- $\delta((k,0,I),1) = (1,0,I \cup \{k\})$ if $k \not\in I$, and $\delta((k,0,I),1) = q_{\text{rej}}$ otherwise.
- $\delta(q_{\text{rej}},0) = \delta(q_{\text{rej}},1) = q_{\text{rej}}$.

■

Lemma 32 *For every $\mathbf{x} \in \{0,1\}^n$ and a sufficiently large n, there is a DFA A_P with at most $n\log(n)$ states such that A_P accepts a short encoding of a hyperedge S iff $P(\mathbf{x}_S) = 1$.*

Proof Let $\mathbf{z}^S$ be a short encoding of a hyperedge S. We construct $A_P = \langle \Sigma, Q, q_0, \delta, F \rangle$ that accepts $\mathbf{z}^S$ iff $P(\mathbf{x}_S) = 1$. Let $\Sigma = \{0,1\}$, let $B = \{0,1,\_\}^k$ and $Q = \{q_0\} \cup ([n] \times [k] \times B)$, and let $F = \{n\} \times \{k\} \times \{\mathbf{b} \in \{0,1\}^k : P(\mathbf{b}) = 1\}$. Note that Q is of size at most $n\log(n)$ (for a sufficiently large n). The states in Q are such that the first two components keep the current location in the short encoding, and the third component keeps the information on $\mathbf{x}_S$. The transitions are

- $\delta(q_0,0) = (1,1,(\_,\ldots,\_))$.
- $\delta(q_0,1) = (1,1,(x_1,\_,\ldots,\_))$.
- For $i \in [n]$, $j \in [k-1]$, $\mathbf{b} \in B$ we have:
  - $\delta((i,j,\mathbf{b}),0) = (i,j+1,\mathbf{b})$.
  - $\delta((i,j,\mathbf{b}),1) = (i,j+1,(b_1,\ldots,b_j,x_i,b_{j+2},\ldots,b_k))$.
  - $\delta((i,k,\mathbf{b}),0) = ((i \mod n)+1,1,\mathbf{b})$.
  - $\delta((i,k,\mathbf{b}),1) = ((i \mod n)+1,1,(x_{(i \mod n)+1},b_2,\ldots,b_k))$.

■

Lemma 33 *For every $\mathbf{x} \in \{0,1\}^n$ and a sufficiently large n, there is a function $f : \{0,1\}^{n^{1+3/\epsilon}} \to \{0,1\}$ that can be expressed by a DFA with at most n^3 states, such that:*

- *For every hyperedge S and every extended multi-long encoding $\tilde{\mathbf{z}}$ of S, we have $f(\tilde{\mathbf{z}}) = P_{\mathbf{x}}(\tilde{\mathbf{z}})$.*

- *For every $\tilde{\mathbf{z}} \in n^{1+3/\epsilon}$ that is not an extended multi-long encoding of a hyperedge, we have $f(\tilde{\mathbf{z}}) = 0$.*

Proof Let $d \geq c' \log^2(n) \cdot nk$. We first construct a DFA A' such that for every $\mathbf{z} \in \{0,1\}^d$ we have: If $\mathbf{z}$ starts with a multi-short encoding of a hyperedge S then A' accepts $\mathbf{z}$ iff $P(\mathbf{x}_S) = 1$, and if $\mathbf{z}$ does not start with a multi-short encoding of a hyperedge then A' rejects $\mathbf{z}$. Let A_E and A_P be the DFAs from Lemmas 31 and 32. Thus, A_E checks whether a word is a short encoding of a hyperedge, and A_P checks whether a short encoding $\mathbf{z}^S$ is such that $P(\mathbf{x}_S) = 1$. The DFA A' runs A_E and A_P in parallel on the first size-nk slice. If both A_E and A_P accept then A' accepts, if A_E accepts and A_P rejects then A' rejects, and if A_E rejects then A' continues to the next size-nk slice in a similar manner. Also, A' keeps a counter and stops after $c' \log^2(n)$ slices. Constructing such a DFA is straightforward. Moreover, since A_E has at most $\log(n)$ states and A_P has at most $n \log(n)$ states, then A' has at most $\log(n) \cdot n \log(n) \cdot (c' \log^2(n) nk + 1) \leq n^2 \log^5(n)$ states (for a sufficiently large n).

Next, we construct a DFA A such that for every $\tilde{\mathbf{z}} \in \{0,1\}^{n^{1+3/\epsilon}}$ we have: If $\tilde{\mathbf{z}}$ starts with a multi-long encoding of a hyperedge S then A accepts $\tilde{\mathbf{z}}$ iff $P(\mathbf{x}_S) = 1$, and if $\tilde{\mathbf{z}}$ does not start with a multi-long encoding of a hyperedge then A rejects $\tilde{\mathbf{z}}$. Note that such a DFA A satisfies the lemma's requirements. The DFA A is obtained from A' by replacing each state q in A' by the DFA $A^q = \langle \Sigma, Q^q, q, \delta^q, F^q \rangle$ such that $Q^q = \{q\} \cup ([\log(n) - 1] \times \{0,1\})$, $\delta^q(q,0) = (1,0)$, $\delta^q(q,1) = (1,1)$, and for every $i \in [\log(n) - 2]$ we have $\delta^q((i,1),1) = (i+1,1)$, and $\delta^q((i,0),0) = \delta^q((i,0),1) = \delta^q((i,1),0) = (i+1,0)$. Then, for the transitions $\delta'(q,0) = q'$ and $\delta'(q,1) = q''$ in the DFA A', the DFA A includes the appropriate transitions from the states $(\log(n) - 1, 0)$ and $(\log(n) - 1, 1)$ of A^q, namely, $\delta((\log(n) - 1, 1), 0) = \delta((\log(n) - 1, 0), 0) = \delta((\log(n) - 1, 0), 1) = q'$ and $\delta((\log(n) - 1, 1), 1) = q''$. Also, if q is an accepting state in A' then we set $F^q = \{q\}$ and otherwise $F^q = \emptyset$. Thus, A' and A have the same accepting states. Note that A has at most $n^2 \log^5(n) \cdot 2 \log(n) \leq n^3$ states. ■

A.8. Proof of Theorem 14

In the proof of Theorem 3, we constructed a DNF formula $\psi_{\mathbf{x}}$ such that for every encoding $\mathbf{z}^S \in \{0,1\}^{kn}$ of a hyperedge S we have $\psi_{\mathbf{x}}(\mathbf{z}^S) = P_{\mathbf{x}}(\mathbf{z}^S) = P(\mathbf{x}_S)$. We now show that there is a 2^k-sparse $GF(2)$ polynomial $h : \{0,1\}^{kn} \to \{0,1\}$, such that for every hyperedge S we have $h(\mathbf{z}^S) = P_{\mathbf{x}}(\mathbf{z}^S)$. Namely, h agrees with $\psi_{\mathbf{x}}$ on inputs that encode hyperedges. Then, the theorem follows from the arguments in the proof of Theorem 3.

By Lemma 18, the DNF $\psi_{\mathbf{x}}$ has at most 2^k terms. Each term C_j in $\psi_{\mathbf{x}}$ is a conjunction of positive literals, such that $C_j(\mathbf{z}^S) = 1$ iff $\mathbf{x}_S$ is the j-th satisfying assignment of the predicate P. Hence, it is not possible that more than one term in $\psi_{\mathbf{x}}(\mathbf{z}^S)$ is satisfied. Let h be the $GF(2)$ polynomial induced by $\psi_{\mathbf{x}}$, i.e., each monomial in h corresponds to a term C_j from $\psi_{\mathbf{x}}$. Since at most one term in $\psi_{\mathbf{x}}(\mathbf{z}^S)$ is satisfied, then we have: If $\psi_{\mathbf{x}}(\mathbf{z}^S) = 1$ then exactly one term in $\psi_{\mathbf{x}}(\mathbf{z}^S)$ is satisfied, and therefore $h(\mathbf{z}^S) = 1$. Also, if $\psi_{\mathbf{x}}(\mathbf{z}^S) = 0$ then all terms in $\psi_{\mathbf{x}}(\mathbf{z}^S)$ are unsatisfied, and therefore $h(\mathbf{z}^S) = 0$.