Proceedings of Machine Learning Research vol 134:1–29, 2021 34th Annual Conference on Learning Theory

Size and Depth Separation in Approximating Benign Functions with Neural Networks

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

Daniel Reichman DANIEL.REICHMAN@GMAIL.COM
Worcester Polytechnic Institute

Toniann Pitassi TONI@CS.TORONTO.EDU
University of Toronto and IAS

Ohad Shamir OHAD.SHAMIR@WEIZMANN.AC.IL
Weizmann Institute of Science, Israel

Editors: Mikhail Belkin and Samory Kpotufe

Abstract

When studying the expressive power of neural networks, a main challenge is to understand how the size and depth of the network affect its ability to approximate real functions. However, not all functions are interesting from a practical viewpoint: functions of interest usually have a polynomially-bounded Lipschitz constant, and can be computed efficiently. We call functions that satisfy these conditions "benign", and explore the benefits of size and depth for approximation of benign functions with ReLU networks. As we show, this problem is more challenging than the corresponding problem for non-benign functions. We give complexity-theoretic barriers to showing depth-lower-bounds: Proving existence of a benign function that cannot be approximated by polynomial-sized networks of depth 4 would settle longstanding open problems in computational complexity. It implies that beyond depth 4 there is a barrier to showing depth-separation for benign functions, even between networks of constant depth and networks of nonconstant depth. We also study size-separation, namely, whether there are benign functions that can be approximated with networks of size $\mathcal{O}(s(d))$, but not with networks of size $\mathcal{O}(s'(d))$. We show a complexity-theoretic barrier to proving such results beyond size $\mathcal{O}(d\log^2(d))$, but also show an explicit benign function, that can be approximated with networks of size $\mathcal{O}(d)$ and not with networks of size $o(d/\log d)$. For approximation in the L_∞ sense we achieve such separation already between size $\mathcal{O}(d)$ and size $o(d)$. Moreover, we show superpolynomial size lower bounds and barriers to such lower bounds, depending on the assumptions on the function. Our size-separation results rely on an analysis of size lower bounds for Boolean functions, which is of independent interest: We show linear size lower bounds for computing explicit Boolean functions (such as set disjointness) with neural networks and threshold circuits.

1. Introduction

The *expressive power* of feedforward neural networks is a central topic in the theory of deep learning. It is well-known that sufficiently large depth-2 neural networks, using reasonable activation functions, can approximate any continuous function on a bounded domain (Cybenko (1989); Funahashi (1989); Hornik (1991); Barron (1994)). However, the required size of such networks (namely, the overall number of neurons) can be impractically large, e.g., exponential in the input dimension. From a learning perspective, both theoretically and in practice, the main interest is in neural networks whose size is at most polynomial in the input dimension.

© 2021 G. Vardi, D. Reichman, T. Pitassi & O. Shamir.

Many works in recent years have studied the expressive power of polynomial-size neural networks, and the beneficial effect of depth for approximating real functions. However, not all functions are interesting from a practical viewpoint: For example, in practice we are interested in functions that can be efficiently computed. Moreover, in learning tasks, it is usually sufficient to consider prediction functions which have some polynomially-bounded Lipschitz parameter: Otherwise, it means that the learning task crucially requires a function that varies at a superpolynomial rate, which is generally not the case. In addition, functions with very large Lipschitz constants tend to be more difficult to learn with standard methods (cf. Safran et al. (2019); Malach et al. (2021)).

Motivated by this, the main goal of our paper is to explore the benefits of size and depth for approximation of *benign* functions, which do satisfy the conditions above. Specifically, we say that a function $f : [0,1]^d \to [0,1]$ is benign if it satisfies the following conditions (stated slightly informally): (1) it is $\text{poly}(d)$-Lipschitz; (2) there is an algorithm that for $\mathbf{x} \in [0,1]^d$ given in binary representation, computes $f(\mathbf{x})$ in at most *exponential* time, within $1/\text{poly}(d)$ precision. Clearly, this computability requirement is very mild. A stronger (and still mild) assumption is to replace the exponential-time requirement by a polynomial-time requirement, in which case we will call such functions *polynomial-time benign*. We provide several results, both positive and negative, on the benefits of size and depth for approximating benign functions with ReLU networks.

Depth separation. Overwhelming empirical evidence indicates that deeper networks tend to perform better than shallow ones. Quite a few theoretical works in recent years have explored the beneficial effect of depth on increasing the expressiveness of neural networks (e.g., Martens et al. (2013); Eldan and Shamir (2016); Telgarsky (2016); Liang and Srikant (2016); Daniely (2017); Safran and Shamir (2017); Yarotsky (2017); Safran et al. (2019); Vardi and Shamir (2020); Bresler and Nagaraj (2020)). A main focus is on *depth separation*, namely, showing that there is a function $f : \mathbb{R}^d \to \mathbb{R}$ that can be approximated by a $\text{poly}(d)$-sized network of a given depth, with respect to some input distribution, but cannot be approximated by $\text{poly}(d)$-sized networks of a smaller depth. Depth separation between depth 2 and 3 is known (Eldan and Shamir, 2016; Daniely, 2017)[1] already for benign functions. A complexity-theoretic barrier to proving separation between two constant depths beyond depth 4 was established in Vardi and Shamir (2020). A construction shown by Telgarsky (2016) gives separation between $\text{poly}(d)$-sized networks of a constant depth, and $\text{poly}(d)$-sized networks of some nonconstant depth. Thus, restricting the depth hurts the expressiveness of $\text{poly}(d)$-sized networks. However, the function of Telgarsky (2016) is highly oscillatory, and its Lipschitzness is superpolynomial in d. Hence, an interesting question is whether $\text{poly}(d)$-sized networks are more powerful than $\text{poly}(d)$-sized networks of constant depth, in their ability to approximate *benign* functions. We show:

- Proving existence of a benign function that cannot be approximated by $\text{poly}(d)$-sized networks of constant depth $k \geq 4$, would settle a longstanding open problem in computational complexity (namely, $\mathsf{EXP} \not\subseteq \mathsf{TC}^0_{k-2}$, where TC^0_{k-2} is the class of threshold circuits of polynomial size and depth $k-2$). Moreover, if we wish to prove the result for depth $k \geq 6$, it would require overcoming a *natural-proofs barrier*, a concept from computational complexity which indicates that such a proof would be very hard to find. We note that Vardi and Shamir (2020) gave a barrier to proving depth separation, and we give a barrier already to proving depth lower bounds. Thus,

1. The result of Daniely (2017) holds only for depth-2 networks whose weights magnitudes are upper-bounded by an exponential.

while the barrier of Vardi and Shamir (2020) holds only for separation between two constant depths, our barrier applies also to separation between constant and nonconstant depths.

- Interestingly, we show that this barrier crucially relies on both "benignness" requirements: If we allow exponential Lipschitz constants, then a depth lower bound is known (Telgarsky, 2016). Moreover, using counting arguments, we prove a lower bound for a function that is 1-Lipschitz, but not necessarily computable in exponential time.

Size separation. We study *size separation*, namely, whether there are benign functions that can be approximated by networks of size $\mathcal{O}(s(d))$, but cannot be approximated by networks of size $\mathcal{O}(s'(d))$. Here, we consider the overall number of neurons in the network, regardless of its depth/width. Recall that the motivation behind the study of depth separation is to achieve a theoretical understanding of the empirical success of deeper networks compared to shallow ones. This motivation applies also to size separation, since often large networks are required in practice in order achieve good performance, and we are missing a theoretical understanding of this phenomenon. We show both size-separation results, and barriers to size-separation:

- Proving existence of a polynomial-time benign function that cannot be approximated by networks of size $\mathcal{O}(d\log^2(d))$, would settle the longstanding open problem in circuit complexity, on whether there is a Boolean function in P that cannot be computed by threshold circuits of linear size.
- We show a polynomial-time benign function, that can be approximated by a network of size $\mathcal{O}(d)$, but cannot be approximated by networks of size $o(d/\log d)$.
- We also consider size-separation in the L_∞ sense (where we wish to approximate the function uniformly rather than on average), and show a polynomial-time benign function that can be computed by a network of size $\mathcal{O}(d)$, but cannot be approximated by networks of size $o(d)$.

Superpolynomial size lower bounds. Many works in recent years have studied approximation of classes of smooth functions with ReLU neural networks (e.g., Gühring et al. (2020); Yarotsky (2017); Petersen and Voigtlaender (2018); Yarotsky (2018); Yarotsky and Zhevnerchuk (2019); Shen et al. (2019); Lu et al. (2020)). The upper bounds on the required size of the network are at least exponential in the input dimension. Yarotsky (2017) gave a lower bound for the required size for approximation in the L_∞ sense, which is exponential in the input dimension. We show:

- There is a 1-Lipschitz function $f : [0,1]^d \to [0,1]$ that cannot be approximated by $\mathrm{poly}(d)$-sized networks whose weights are represented by a $\mathrm{poly}(d)$ number of bits. Thus, we obtain a superpolynomial size lower bound for approximating 1-Lipschitz functions in the L_2 sense. However, this function is obtained by a counting argument and is not known to be benign.
- We give a barrier to proving superpolynomial size lower bounds, already for *semi-benign* functions, namely, functions with an *exponential* Lipschitz constant, that are computable in exponential time. We show that proving such a lower bound would imply that $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$.

Size lower bounds for Boolean functions. Our size-separation results for benign functions rely on an analysis of the corresponding problem for Boolean functions. Namely, we show size lower-bounds and upper-bounds for computing certain Boolean functions with ReLU networks, and then use these bounds to obtain size-separation for real functions. We consider the Boolean functions that compute *disjointness* and *inner product*, and show linear size lower bounds. Our lower bounds are based on results from communication complexity, and hold also for networks with k-piecewise-linear activation. We note that these results are also of independent interest. Indeed, the study of the computational power of neural networks in the context of Boolean functions has received ample attention in the past decades (e.g., Maass et al. (1991); Koiran (1996); Maass (1997); Martens et al. (2013); Kane and Williams (2016); Mukherjee and Basu (2017); Williams (2018)). Our linear size lower bounds for disjointness and inner-product hold also for threshold circuits. This bound for threshold circuits was already shown with different methods for the inner-product function (Groeger and Turán, 1993; Jukna, 2012; Roychowdhury et al., 1994), but is new for disjointness.

Connection to threshold circuits. In order to establish our results, we explore the connection between ReLU networks and threshold circuits. These are essentially neural networks with a threshold activation function in all neurons (including the output neuron), and where the inputs are in $\{0,1\}^d$. Size and depth lower bounds for threshold circuits were extensively studied in the context of circuit complexity over the past decades. It is natural to ask whether the results on size and depth lower bounds in threshold circuits have implications on the analogous questions for neural networks, and indeed we study such implications in our work. However, we emphasize that in general, it is not obvious how to "import" separation results (or barriers) of threshold circuits to the realm of neural networks. This is because unlike threshold circuits, neural networks have real-valued inputs and outputs, and a continuous activation function. Thus, it might be possible to come up with a separation result, which crucially utilizes some function and inputs in Euclidean space. In fact, this can already be seen in existing results: For example, separation between threshold circuits of polynomial size and constant depth (TC^0) and threshold circuits of polynomial size of any depth (which equals the complexity class $\mathsf{P/poly}$) is not known, but Telgarsky (2016) showed such a result for neural networks. His construction is based on the observation that for one dimensional data, a network of depth k is able to express a sawtooth function on the interval $[0,1]$ which oscillates $\mathcal{O}(2^k)$ times. Clearly, this utilizes the continuous structure of the domain, in a way that is not possible with Boolean inputs. Also, depth-separation results for neural networks (Eldan and Shamir, 2016; Daniely, 2017) rely on harmonic analysis of real functions. Finally, the result of Eldan and Shamir (2016) does not make any assumption on the weight magnitudes, whereas relaxing this assumption for the parallel result on threshold circuits is a longstanding open problem (Razborov, 1992b).

2. Preliminaries

Notations. We use bold-faced letters to denote vectors, e.g., $\mathbf{x} = (x_1, \ldots, x_d)$. For $\mathbf{x} \in \mathbb{R}^d$ we denote by $\|\mathbf{x}\|$ the Euclidean norm. For a real function f and a distribution $\mathcal{D}$, we denote by $\|f\|_{L_2(\mathcal{D})}$ the L_2 norm weighted by $\mathcal{D}$, namely $\|f\|^2_{L_2(\mathcal{D})} = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}(f(\mathbf{x}))^2$. For a set A, we denote by $\mathcal{U}(A)$ the uniform distribution over A. For an integer $d \geq 1$ we denote $[d] = \{1, \ldots, d\}$. We use $\mathrm{poly}(d)$ as a shorthand for "some polynomial in d". Let μ be the density function of a continuous distribution on $[0,1]^d$. For $i \in [d]$ we denote by μ_i the marginal density of the i-th component. We say that μ has a *polynomially-bounded marginal density* if there is $M = \mathrm{poly}(d)$ such that $\mu_i(t) \leq M$ for every $i \in [d]$ and $t \in [0,1]$.

benign functions. We say that a function $f : [0,1]^d \to [0,1]$ is *benign* if it satisfies the following conditions: (1) It is $\mathrm{poly}(d)$-Lipschitz; (2) It is exponential-time computable: For every $c = \mathcal{O}(\log(d))$, there is an algorithm $\mathcal{A}$ that runs in time exponential in d, such that for every input $\mathbf{x} \in [0,1]^d$ where each component is given by a binary representation with c bits, it returns

$f(\mathbf{x})$ within precision of c bits. Namely, the algorithm $\mathcal{A}$ returns a c-bits binary representation of $\mathbf{y} \in [0,1]$ such that $|\mathbf{y} - f(\mathbf{x})| \leq \frac{1}{2^c} = \frac{1}{\text{poly}(d)}$.

We say that a benign function f is *polynomial-time benign* (respectively, *polynomial-space benign*), if it is computable in polynomial time (resp., space), i.e., for every $c = \mathcal{O}(\log(d))$, there is an algorithm $\mathcal{A}$ that runs in $\text{poly}(d)$ time (resp., space), such that for every input $\mathbf{x} \in [0,1]^d$ where each component is given by c bits, it returns $f(\mathbf{x})$ within precision of c bits.

In the above definitions we use $c = \mathcal{O}(\log(d))$ bits since it corresponds to precision of $1/\text{poly}(d)$. Standard mathematical operations such as addition, multiplication, division, square root, $\exp$, $\log$, $\sin$ and $\cos$, can all be computed within precision of m bits in $\text{poly}(m)$ time, as well as combinations of such operations. Thus, the requirement that we can compute f within precision of a logarithmic number of bits in polynomial time is standard. Note that a function may be expressible by an exponential-size (or even polynomial-size) neural network but not exponential-time computable, since it may not be possible to find the neural network in exponential time. Namely, expressiveness by bounded-size neural networks corresponds to *non-uniform computability*, while time-computability is in the *uniform* sense.

The Lipschitzness assumption is also standard, since in learning tasks it is usually sufficient to consider prediction functions with a bounded Lipschitz constant, as they tend to be more robust and not very sensitive to small changes in the input. E.g., if we slightly perturb the pixel values in some given image we usually do not expect the target distribution over the possible image labels to change dramatically. Note that the Lipschitzness assumption is w.r.t. the target function that we want to approximate, and the neural network that approximates the function is not limited in its Lipschitzness. Moreover, target functions with very large Lipschitz constants are usually harder to learn with gradient-based methods.

Consider, for example, the function given in Daniely (2017) to establish depth-separation between depth 2 and 3. This function is of the form[2] $f(\mathbf{x}) = \sin(\frac{1}{2}\pi d^3 \|\mathbf{x}\|)$, and hence it is clearly $\text{poly}(d)$-Lipschitz and computable in polynomial time. Thus, it is polynomial-time benign.

Neural networks. We consider feedforward neural networks, computing functions from $\mathbb{R}^d$ to $\mathbb{R}$. A neural network is composed of layers of neurons, where each neuron, except for the output neuron, has an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. We focus on the ReLU activation, namely, $\sigma(z) = [z]_+ := \max\{0, z\}$. When we consider other activation functions we explicitly mention it. We define the *depth* of the network as the number of layers. Denoting the number of neurons in the i-th layer by n_i, we define the *width* of a network as $\max_i n_i$, and the *size of the network* as $\sum_i n_i$. We sometimes consider networks with an activation in the output neuron, and with multiple outputs.

Threshold circuits. A threshold circuit is a neural network with the following restrictions: (1) The activation function in all neurons is $\sigma(z) = \text{sign}(z)$. We define $\text{sign}(z) = 0$ for $z \leq 0$, and $\text{sign}(z) = 1$ for $z > 0$. A neuron in a threshold circuit is called a *threshold gate*. The function computed by a threshold gate is called *linear threshold function (LTF)*, and is denoted by $L_{\mathbf{a}=(a_1 \ldots a_m),\theta}$, where $\mathbf{a}$ are the weights and θ is the bias term. (2) The output gates also have a sign activation function. Hence, the output is binary. (3) We always assume that the input to a threshold circuit is a binary vector $\mathbf{x} \in \{0,1\}^d$. (4) Since every threshold circuit with real weights can be expressed by a threshold circuit of the same size with integer weights bounded by $2^{\mathcal{O}(d \log(d))}$ (cf. Goldmann and Karpinski (1998)), we assume that all weights are represented by $\text{poly}(d)$ bits.

2. The function in Daniely (2017) is defined on the sphere and in a slightly different way, but it is easily reduced to a function of this form on the unit ball (see Vardi and Shamir (2020); Safran et al. (2019)).

We denote by TC^0 the class of polynomial-sized threshold circuits of constant depth, and by TC^0_k the class of polynomial-sized threshold circuits of depth k.

Functions approximation. We say that a function f can be *approximated by a* $\operatorname{poly}(d)$*-sized neural network of depth* k (with respect to a distribution $\mathcal{D}$) if for every $\epsilon = \frac{1}{\operatorname{poly}(d)}$ we have $\|f - N\|_{L_2(\mathcal{D})} \leq \epsilon$ for some depth-k network N of size $\operatorname{poly}(d)$. For $\epsilon = \epsilon(d)$ and $m = m(d)$, we say that f can be *ϵ-approximated by a neural network of size* m (with respect to a distribution $\mathcal{D}$) if $\|f - N\|_{L_2(\mathcal{D})} \leq \epsilon$ for some network N of size m. While we focus on approximation in the L_2 sense, we note that our results apply also to L_p for every $1 \leq p < \infty$. For a function $f : [0,1]^d \to [0,1]$ and a neural network N, we denote $\|f - N\|_\infty = \sup_{\mathbf{x} \in [0,1]^d} |f(\mathbf{x}) - N(\mathbf{x})|$.

Depth-separation and size-separation. We say that there is *depth-separation* between networks of depth k and depth k' for some integers $k' > k$, if there is a distribution $\mathcal{D}$ on $[0,1]^d$ and a function $f : [0,1]^d \to [0,1]$ that can be approximated (with respect to $\mathcal{D}$) by a $\operatorname{poly}(d)$-sized neural network of depth k' but cannot be approximated by $\operatorname{poly}(d)$-sized networks of depth k. We note that our definition of depth-separation is a bit weaker than most existing depth-separation results, which actually show difficulty of approximation even up to constant accuracy (and not just $1/\operatorname{poly}(d)$ accuracy). However, depth separation in that sense implies depth separation in our sense. Hence, the barriers we show here to depth separation imply similar barriers under this other (or any stronger) notion of depth separation. Our definition is similar to the definition in Vardi and Shamir (2020).

We say that there is *size-separation* between networks of size $\mathcal{O}(m)$ and size $\mathcal{O}(m')$, if there is a distribution $\mathcal{D}$ on $[0,1]^d$, a function $f : [0,1]^d \to [0,1]$, and $\epsilon = \frac{1}{\operatorname{poly}(d)}$, such that f can be ϵ-approximated (with respect to $\mathcal{D}$) by a neural network of size $\mathcal{O}(m')$ but cannot be ϵ-approximated by networks of size $\mathcal{O}(m)$. Thus, networks of size $\mathcal{O}(m')$ are more powerful than networks of size $\mathcal{O}(m)$ in their ability to approximate f within some reasonable accuracy ϵ.

Natural-proofs barrier. The study of circuit lower bounds is a central challenge in theoretical computer science, but despite many attempts the results in this field are limited (Arora and Barak, 2009). In a seminal work, Razborov and Rudich (1997) described a main technical limitation of current approaches for proving circuit lower bounds: They defined a notion of "natural proofs" for a circuit lower bound (which include current proof techniques), and showed that obtaining lower bounds with such proof techniques would violate a widely accepted conjecture on the existence of pseudorandom functions. This *natural-proofs barrier* (partially) explains the lack of progress on circuit lower bounds. More formally, if a class $\mathcal{C}$ of circuits contains a family of pseudorandom functions, then showing for some function f that $f \notin \mathcal{C}$ cannot be done with a natural proof.

3. Barriers to depth lower bounds and to depth-separation

Telgarsky (2016) showed that there exists a family of univariate functions $\{\varphi_k\}_{k=1}^\infty$ on the interval $[0,1]$, such that the function φ_k is 2^k-Lipschitz, it can be expressed by a network of depth k and width $\mathcal{O}(1)$, but it cannot be ϵ-approximated (for some constant ϵ) by any $o(k/\log(k))$-depth, $\operatorname{poly}(k)$-width network with respect to the uniform distribution on $[0,1]$. The function φ_k consists of 2^{k-1} identical triangles of height 1[3]. Consider the functions $\{f_d\}_{d=1}^\infty$ where $f_d : [0,1]^d \to [0,1]$ is such that $f_d(\mathbf{x}) = \varphi_d(x_1)$. Thus, f_d depends only on the first component of $\mathbf{x}$. The result of Telgarsky (2016) implies that the function f_d can be expressed by a network of width $\mathcal{O}(1)$ and depth d, but cannot be approximated by networks of width $\operatorname{poly}(d)$ and constant depth w.r.t. the

3. The function φ_k is obtained by composing the function $z \mapsto [2z]_+ - [4z-2]_+$ with itself k times.

uniform distribution on $[0, 1]^d$. Hence, there is separation between constant and nonconstant depths. Namely, there are functions that can be computed by a neural network of $\text{poly}(d)$ size, but cannot be approximated by networks of $\text{poly}(d)$ size and constant depth. Note that f_d is 2^d-Lipschitz.

As we discussed in the introduction, the main weakness of the above result, is that the Lipschitzness of f_d is superpolynomial. Hence, an interesting question is whether such a result can be obtained for benign functions. The following theorem implies barriers to depth-lower-bounds.

Theorem 1 *If there exists a benign function $f : [0, 1]^d \to [0, 1]$, that cannot be approximated by a neural network of size $\text{poly}(d)$ and constant depth $k \geq 4$, w.r.t. a distribution μ with a polynomially-bounded marginal density, then* $\mathsf{EXP} \not\subseteq \mathsf{TC}^0_{k-2}$.

Proof idea (for complete proof see Appendix A.1) Let $c = \mathcal{O}(\log(d))$. Since f is exponential-time computable then there is an exponential-time algorithm $\mathcal{A}$, such that for an input $\mathbf{x}$ given by c bits for each component, it returns $f(x)$ within precision of c bits. Let $\hat{f} : \{0, 1\}^{c \cdot d} \to \{0, 1\}^c$ be the function that $\mathcal{A}$ computes. Assume that $\hat{f}$ can be computed by a threshold circuit T of size $\text{poly}(d)$ and depth $k - 2$. We construct a neural network N of size $\text{poly}(d)$ and depth k that approximates f and thus reach a contradiction. It implies that the function $\hat{f}$ can be computed in exponential time but cannot be computed by a threshold circuit of size $\text{poly}(d)$ and depth $k - 2$, and hence we are able to obtain $\mathsf{EXP} \not\subseteq \mathsf{TC}^0_{k-2}$. The network N first transforms w.h.p. over $\mathbf{x} \sim \mu$ the input $\mathbf{x} \in [0, 1]^d$ to a binary representation $\hat{\mathbf{x}} \in \{0, 1\}^{c \cdot d}$, then it simulates the threshold circuit T to obtain $T(\hat{\mathbf{x}}) = \hat{f}(\hat{\mathbf{x}})$, and finally it converts the output of T from a binary representation to the corresponding real value. Note that since f is $\text{poly}(d)$-Lipschitz, then for an appropriate c, transforming the input to the binary representation does not hurt the approximation too much. ■

Remark 2 (Barrier to depth lower bounds) *It is a longstanding open problem whether* $\mathsf{EXP} \not\subseteq \mathsf{TC}^0_2$ *(and even whether* $\mathsf{NEXP} \not\subseteq \mathsf{TC}^0_2$*) (Razborov, 1992b; Oliveira, 2015; Chen, 2018). By Theorem 1, proving existence of a benign function that cannot be approximated by a network of depth $k \geq 4$ would imply that* $\mathsf{EXP} \not\subseteq \mathsf{TC}^0_{k-2}$ *and thus solve this open problem.*

Remark 3 (A stronger barrier for "more benign" functions) *For polynomial-time benign functions and polynomial-space benign functions, we can obtain even stronger barriers to depth lower bounds: Proving existence of a polynomial-time (respectively, polynomial-space) benign function that cannot be approximated by a network of depth $k \geq 4$, would imply that* $\mathsf{P} \not\subseteq \mathsf{TC}^0_{k-2}$ *(respectively,* $\mathsf{PSPACE} \not\subseteq \mathsf{TC}^0_{k-2}$*). Since* $\mathsf{P} \subseteq \mathsf{P/poly}$*, establishing* $\mathsf{P} \not\subseteq \mathsf{TC}^0_{k-2}$ *would also imply that* $\mathsf{TC}^0_{k-2} \neq \mathsf{P/poly}$*. Moreover, we can also conclude that proving existence of a polynomial-time benign function that cannot be approximated by networks of polynomial size and any constant depth, would solve the open problem of whether* $\mathsf{TC}^0 \neq \mathsf{P/poly}$*.*

Remark 4 (Natural-proofs barrier for $k \geq 6$) *Naor and Reingold (2004) and Krause and Lucks (2001) showed a candidate pseudorandom function family in* TC^0_4*. By Razborov and Rudich (1997), it implies that there is a natural-proofs barrier to proving circuit lower bounds for threshold circuits of depth at least 4. Since by Theorem 1 proving existence of a benign function that cannot be approximated by networks of depth $k \geq 6$ would imply a lower bound for threshold circuits of depth $k - 2 \geq 4$, then such depth lower bounds would need to overcome the natural-proofs barrier.*

Remark 5 (Barrier to depth separation) *Theorem 1 clearly implies a barrier to showing depth-separation results for benign functions. Thus, there is a barrier already to showing depth separation* *for a benign function between* $\mathrm{poly}(d)$*-sized networks of depth* 4 *and* $\mathrm{poly}(d)$*-sized networks of unbounded depth. We note that Vardi and Shamir (2020) established a complexity-theoretic barrier to showing depth-separation, but their result applies only to separation between two constant depths.*

By its definition, the benign function f in Theorem 1 satisfies two requirements: $\mathrm{poly}(d)$-Lipschitzness and exponential-time computability. As we already discussed, the construction of Telgarsky (2016) gives a function that cannot be approximated by $\mathrm{poly}(d)$-sized networks of any constant depth. This function is efficiently computable, but is not $\mathrm{poly}(d)$-Lipschitz. We now show that a similar result can be obtained with a function that is 1-Lipschitz, but we do not have any guarantees on its computability.

Theorem 6 *There exists a* 1*-Lipschitz function* $f : [0,1]^d \to [0,1]$ *that cannot be approximated w.r.t. a distribution with a polynomially-bounded marginal density, by a* $\mathrm{poly}(d)$*-sized neural network whose weights are represented by a* $\mathrm{poly}(d)$ *number of bits.*

We note that Theorem 1 holds already for networks whose weights are represented by a $\mathrm{poly}(d)$ number of bits. That is, if the benign function f cannot be approximated by a network of size $\mathrm{poly}(d)$ and depth k, whose weights are represented by $\mathrm{poly}(d)$ bits, then $\mathsf{EXP} \not\subseteq \mathsf{TC}^0_{k-2}$. Hence the barriers to depth lower bounds (for benign functions) hold already for the network considered in Theorem 6. The proof of Theorem 6 follows essentially from a counting argument: we show that in order to cover the set of all 1-Lipschitz functions $f : [0,1]^d \to [0,1]$ with balls of radius $\epsilon = \frac{1}{\mathrm{poly}(d)}$ (w.r.t. norm $L_2(\mu)$), the number of balls required is larger than the number of $\mathrm{poly}(d)$-sized networks whose weights are represented by $\mathrm{poly}(d)$ bits. See Appendix A.2 for the proof.

4. Barriers to size lower bounds and to size separation

4.1. Barriers to superpolynomial lower bounds

In Section 3 we studied which functions cannot be approximated by neural networks of polynomial size and bounded depth. Here, we study which functions cannot be approximated by neural networks of polynomial size without restricting its depth. The barriers from Section 3 apply also here. Indeed, if a benign function cannot be approximated by any polynomial-sized network, then it clearly cannot be approximated by any polynomial-sized network of a bounded depth. Thus, there is a barrier to proving superpolynomial size lower bounds for benign functions. We now show that a barrier can be obtained already for functions that satisfy a weaker requirement.

We say that a function $f : [0,1]^d \to [0,1]$ is *semi-benign* if it satisfies the following conditions: (1) It is $2^{\mathrm{poly}(d)}$-Lipschitz; (2) It is exponential-time computable for $\mathrm{poly}(d)$-bits inputs: For every $c = \mathrm{poly}(d)$ and $c' = \mathcal{O}(\log(d))$, there is an algorithm $\mathcal{A}$ that runs in time exponential in d, such that for every input $\mathbf{x} \in [0,1]^d$ where each component is given by a binary representation with c bits, it returns $f(\mathbf{x})$ within precision of c' bits. Namely, the algorithm $\mathcal{A}$ returns a c'-bits binary representation of $\mathbf{y} \in [0,1]$ such that $|\mathbf{y} - f(\mathbf{x})| \leq \frac{1}{2^{c'}} = \frac{1}{\mathrm{poly}(d)}$.

Note that, unlike benign functions, the Lipschitz constant of semi-benign functions can be exponential in d. We show a barrier to size lower bounds for semi-benign functions.

Theorem 7 *If there exists a semi-benign function* $f : [0,1]^d \to [0,1]$*, that cannot be approximated by neural networks of size* $\mathrm{poly}(d)$ *w.r.t. a distribution* μ *with a polynomially-bounded marginal density, then* $\mathsf{EXP} \not\subseteq \mathsf{P/poly}$.

The proof of the theorem (in Appendix B.1) follows roughly a similar idea to the proof of Theorem 1. However, since the Lipschitz constant of f is exponential, then we need to use a binary representation with a $\mathrm{poly}(d)$ number of bits. Transforming w.h.p. the input $\mathbf{x} \in [0,1]^d$ to such a representation can be done with a $\mathrm{poly}(d)$-sized network using the construction of Telgarsky (2016).

Remark 8 (Barrier to superpolynomial size lower bounds) *It is a longstanding open problem whether EXP $\not\subseteq$ P/poly. Also, as we discussed in Remark 4, there is a natural-proofs barrier to solving this problem. Hence, Theorem 7 implies a barrier to showing that there exists a semi-benign function that cannot be approximated by polynomial-sized networks.*

Recall that by Telgarsky (2016), there exists a family of univariate functions $\{\varphi_k\}_{k=1}^{\infty}$ on the interval $[0,1]$, such that the function φ_k is 2^k-Lipschitz, and it cannot be approximated by any $o(k/\log(k))$-depth, $\mathrm{poly}(k)$-width network w.r.t. the uniform distribution on $[0,1]$. Consider the functions $\{f_d\}_{d=1}^{\infty}$ where $f_d : [0,1]^d \to [0,1]$ is such that $f_d(\mathbf{x}) = \varphi_{d^{\log(d)}}(x_1)$. The result of Telgarsky (2016) implies that f_d cannot be approximated by a network of depth $\mathrm{poly}(d)$ and width $\mathrm{poly}(d)$ w.r.t. the uniform distribution on $[0,1]^d$. Hence, it cannot be approximated by any network of size $\mathrm{poly}(d)$. The function f_d is exponential-time computable for $\mathrm{poly}(d)$-bits inputs. However, note that f_d is $2^{d^{\log(d)}}$-Lipschitz. Thus, it is not semi-benign.

We note that the barrier implied by Theorem 7 holds already for networks of size $\mathrm{poly}(d)$ whose weights are represented by a $\mathrm{poly}(d)$ number of bits. By Theorem 6 there is a 1-Lipschitz function that cannot be approximated by a $\mathrm{poly}(d)$-sized network whose weights have $\mathrm{poly}(d)$ bits. However, we do not have any guarantees on the time complexity of computing this function.

4.2. Barriers to $\omega(d\log^2(d))$ lower bounds

Here, we establish a barrier to showing $\omega(d\log^2(d))$-size lower bounds with polynomial-time benign functions. The proof follows ideas from the proofs of Theorems 1 and 7 (see Appendix B.2).

Theorem 9 *If there exist a polynomial-time benign function $f : [0,1]^d \to [0,1]$, a distribution μ with a polynomially-bounded marginal density, and $\epsilon = \frac{1}{\mathrm{poly}(d)}$, such that f cannot be ϵ-approximated by neural networks of size $\mathcal{O}(d\log^2(d))$ w.r.t. μ, then there is a function $g : \{0,1\}^{d'} \to \{0,1\}$ in P that cannot be computed by threshold circuits of size $\mathcal{O}(d')$.*

Remark 10 (Barrier to $\omega(d\log^2(d))$-size lower bounds and to size separation) *It is a longstanding open problem whether there is a function in P (or even in NP) that cannot be computed by threshold circuits (or even Boolean circuits) of linear size (cf. Find et al. (2016); Arora and Barak (2009)). Hence, Theorem 9 implies a barrier to proving that there exists a polynomial-time benign function that cannot be approximated by networks of size $\mathcal{O}(d\log^2(d))$. Thus, it also implies a barrier to showing size-separation results for polynomial-time benign function, between size $\mathcal{O}(d\log^2(d))$ and some larger size.*

Let $\{\varphi_k\}_{k=1}^{\infty}$ be the functions from Telgarsky (2016), and recall that φ_k is 2^k-Lipschitz, and that it cannot be approximated by any $o(k/\log(k))$-depth, $\mathrm{poly}(k)$-width network with respect to w.r.t. the uniform distribution on $[0,1]$. Consider the functions $\{f_d\}_{d=1}^{\infty}$ where $f_d : [0,1]^d \to [0,1]$ is such that $f_d(\mathbf{x}) = \varphi_{d\log^4(d)}(x_1)$. Thus, f_d cannot be approximated by networks of depth $\mathcal{O}(d\log^2(d))$ and width $\mathrm{poly}(d)$ w.r.t. the uniform distribution on $[0,1]^d$. Therefore, it cannot be approximated by any network of size $\mathcal{O}(d\log^2(d))$. The function f_d is polynomial-time computable. However, note that f_d is $2^{d\log^4(d)}$-Lipschitz. Thus, it is not benign.

Finally, we note that the barrier implied by Theorem 9 holds already for networks of size $\mathcal{O}(d\log^2(d))$ whose weights are represented by a $\mathrm{poly}(d)$ number of bits. By Theorem 6 there is a 1-Lipschitz function that cannot be approximated by such networks. However, we do not have any guarantees on the time complexity of computing this function.

5. Lower bounds for Boolean functions

In this section we establish size lower bounds for computing certain explicit Boolean functions with neural networks. Our lower bounds are with respect to neural networks that exactly interpolate a Boolean function f. Namely, for every Boolean input the network outputs the *exact* Boolean value of f. The same lower bounds (with nearly identical proofs) apply also to neural networks where the output neuron has a threshold activation function. Such thresholding is often used when considering Boolean functions implemented by neural networks (e.g., Mukherjee and Basu (2017); Martens et al. (2013); Maass (1997); Koiran (1996); Maass et al. (1991)).

5.1. $\Omega(d/\log d)$ lower bound for approximation in $L_2(\mathcal{U}(\{0,1\}^d))$

The lower bounds in this section are based on communication complexity, in the worst-case partition setting (cf. Chapter 7 in Kushilevitz and Nisan (1997)). In this setting there is a Boolean function f with d inputs. An input $(y_1,...,y_d)$ is partitioned between two players, Alice and Bob, with unbounded computational power. In other words, Alice has the set of bits $\{y_i|i\in I\}$ and Bob has the set of bits $\{y_j|j\in[d]\setminus I\}$ (where I is a nonempty subset of $[d]$). The goal of Alice and Bob is to compute $f(y_1,\ldots,y_d)$ using a predefined protocol with as few bits exchanged between the players. In every round of the protocol a single player can send one bit of communication to the other party. The *cost* of a communication protocol on a given input and partition is the number of bits exchanged between the players. The cost of a given protocol is its maximum cost over all possible inputs and partitions. We consider randomized protocols, where Alice and Bob can use random bits and have access to a common source of random bits. The randomized (worst case) communication complexity of f, denoted by $R(f)$, is the minimal cost of a protocol that results with computing f correctly with probability at least $2/3$ on every possible input and partition.

Nisan (1993) observed that a threshold circuit C for a Boolean function f can be used for a communication protocol evaluating f, whose cost is not much larger than the size of C, as any threshold gate can be evaluated with a protocol of logarithmic cost (in the number of inputs to the gate). Therefore, lower bounds on the communication complexity of f imply lower bounds on the threshold circuit complexity of f. When trying to use this idea for neural networks, a difficulty is that the outputs of the neurons are real numbers, as opposed to the case of threshold circuits where the output of every gate is Boolean. We circumvent this problem by noticing that for two parties who have the parameters of a ReLU network computing a Boolean function, and want to determine the sign of the output for a given Boolean input, the parties can determine the sign of the output of each neuron in the network recursively by a low-communication protocol. That is, once the players know for all neurons in layers $1,\ldots,j-1$ whether their outputs are zero or positive, we show that they can determine with the protocol of Nisan (1993) for neurons in layer j whether their outputs are zero of positive, while using a logarithmic number of bits for each neuron. This idea is formalized in the following theorem, and extended to the more general case of neural networks with k-piecewise-linear activation functions. See Appendix C.1 for a proof.

Theorem 11 *Let $h : \{0,1\}^d \to \{0,1\}$ be such that $R(h) = \Omega(d)$. Any ReLU network computing h has size $\Omega(d/\log d)$. More generally, any neural network with a k-piecewise-linear activation function computing h has size $\Omega(d/(\log d \cdot \log k))$.*

Two classical polynomial-time computable Boolean functions with $\Omega(d)$ randomized communication complexity are *disjointness* and *inner product*. The disjointness function $f = \text{DISJ}_d$ evaluates to 1 on Boolean inputs $(x_1, x_2 \dots x_{2d})$ iff the two subsets of $[d]$ whose characteristic vectors are $(x_1, x_2, \dots, x_d), (x_{d+1}, \dots, x_{2d})$ are disjoint. Thus, f evaluates to 1 iff there is no index $j \in [d]$ where $x_j = x_{d+j} = 1$. It is known that $R(f) = \Omega(d)$ (Kalyanasundaram and Schintger, 1992; Razborov, 1992a; Bar-Yossef et al., 2004). The inner product function $g = \text{IP}_d$ with Boolean inputs $(x_1, x_2, \dots, x_d, y_1, y_2, \dots, y_d)$ evaluates to $\sum_{i=1}^{d} x_i y_i \mod 2$. This function is known to satisfy $R(g) = \Omega(d)$ as well (Babai et al., 1986). Thus, Theorem 11 implies the following corollary:

Corollary 12 *Any ReLU network computing $DISJ_d$ or IP_d has size $\Omega(d/\log d)$.*

In the next section we will improve Corollary 12 and establish a linear lower bound. However, since the results here are based on randomized communication complexity, then we are able to obtain a $\Omega(d/\log d)$ lower bound already for approximation in the L_2 sense. Intuitively, it is shown as follows. Assume that there is a neural network N that approximates IP_d in the L_2 sense w.r.t. the uniform distribution over $\{0,1\}^{2d}$. It implies that there is a neural network N' that computes IP_d correctly on a large fraction of the inputs. Using N', the protocol from Theorem 11, and properties of the function IP_d, we show a randomized protocol, that computes w.h.p. $\text{IP}_d(\mathbf{x}, \mathbf{y})$ for every $\mathbf{x}, \mathbf{y}$. Then, the lower bound on the randomized communication complexity of $\text{IP}_d(\mathbf{x}, \mathbf{y})$ implies a lower bound on the sizes of N' and N. See Appendix C.2 for the proof.

Theorem 13 *Let $\epsilon = \frac{1}{20}$. Let N be a ReLU network that ϵ-approximates the function IP_d w.r.t. the uniform distribution over $\{0,1\}^{2d}$. Then, N has size $\Omega(d/\log d)$.*

5.2. $\Omega(d)$ lower bound for exact computation

We now utilize the real communication model introduced by Kraíček (1998) (see also de Rezende et al. (2016)) in order to establish linear size lower bounds for neural networks. Consider a Boolean function $f : \{0,1\}^d \to \{0,1\}$ whose input is split between two players, Alice and Bob. We define the following real (deterministic) communication protocol. In each round, each player outputs a real number, based on its input ($\mathbf{x}$ for Alice and $\mathbf{y}$ for Bob) and a word $\mathbf{w} \in \{0,1\}^*$ (accessible to both players) defined as follows. Before the first round, w is the empty word. At round i Alice outputs $\alpha \in \mathbb{R}$ and Bob outputs $\beta \in \mathbb{R}$. A referee receives α, β and alters the word $\mathbf{w}$ to $\mathbf{w}1$ if $\alpha > \beta$ and to $\mathbf{w}0$ if $\alpha \leq \beta$. The *cost* of the protocol with respect to a given input to f and its bipartition is the final length of $\mathbf{w}$, and the cost of an arbitrary protocol is the maximal cost for every input and bipartition. The real communication complexity of a function f, denoted by $CC^{\mathbb{R}}(f)$, is the minimal cost of a protocol, such that when the protocol halts, f can be computed (deterministically with zero error) from the word $\mathbf{w}$ attained at the termination of the protocol. This model is equivalent to a communication protocol with an access to a greater-than oracle (cf. Chattopadhyay et al. (2019)).

It can be easily shown that the real communication complexity of a linear threshold function is 1. Using a similar reasoning to the proof of Theorem 11, we show that lower bounds on the real communication complexity of a function f imply lower bounds on the size of a neural network that computes f. The argument holds also for threshold circuits, and can be extended to neural networks with a k-piecewise-linear activation. Formally, we have (see Appendix C.3 for a proof):

Theorem 14 *Let $f : \{0,1\}^d \to \{0,1\}$ be such that $CC^{\mathbb{R}}(f) = \Omega(d)$. Any ReLU network or threshold circuit computing f has size $\Omega(d)$. Any neural network with a k-piecewise-linear activation function computing f has size $\Omega(d/\log k)$.*

By Lemma 4.9 in Chattopadhyay et al. (2019), we have $CC^{\mathbb{R}}(\text{DISJ}_d) = \Omega(d)$. A similar lower bound for IP_d is known to experts, but we are not aware of a previous proof. In Appendix C.4 we give a proof of this fact based on Chattopadhyay et al. (2019). Thus, we have the following:

Corollary 15 *Any ReLU network or threshold circuit computing $DISJ_d$ or IP_d has size $\Omega(d)$. Any neural network with a k-piecewise-linear activation computing $DISJ_d$ or IP_d has size $\Omega(d/\log k)$.*

We note that for the case of computing IP_d with threshold circuits, the above result was already shown with different methods (Groeger and Turán, 1993; Jukna, 2012; Roychowdhury et al., 1994). The first proof of this linear lower bound in Groeger and Turán (1993) is based on a gate elimination argument, whereas the proof of Roychowdhury et al. (1994) uses combinatorial properties of communication matrices of threshold circuits. However, Corollary 15 is the first linear lower bound for computing DISJ_d with threshold circuits, and for computing DISJ_d or IP_d with neural networks.

5.3. Linear upper bounds

Recall that by Corollary 15, DISJ_d and IP_d cannot be computed by a neural network of size $o(d)$, and by Theorem 13, IP_d cannot be approximated in the L_2 sense by a network of size $o(d/\log d)$. In the following theorem we show a linear upper bound for computing DISJ_d and IP_d. The theorem follows by straightforward constructions (see Appendix C.5 for a proof).

Theorem 16 *The functions $DISJ_d$ and IP_d can be computed by ReLU networks of size $\mathcal{O}(d)$.*

6. Size separation for benign functions

We utilize our results on Boolean functions in order to establish size separation for polynomial-time benign functions. The following proposition allows us to translate our results from the Boolean setting to the continuous setting (see proof in Appendix D.1).

Proposition 17 *Let $g : \{0,1\}^d \to \{0,1\}$. There is a 4-Lipschitz function $f : [0,1]^d \to [0,1]$ that agrees with g on $\{0,1\}^d$, and a distribution μ on $[0,1]^d$ with a polynomially-bounded marginal density, such that:*

1. *If g cannot be ϵ-approximated by neural networks of size $\mathcal{O}(m)$ w.r.t. the uniform distribution over $\{0,1\}^d$, then f cannot be ϵ-approximated by networks of size $\mathcal{O}(m)$ w.r.t. μ.*
2. *If g can be computed by a neural network of size $\tilde{m}$, then there is a neural network $\tilde{N}$ of size $\tilde{m} + 2d$ such that $\|\tilde{N} - f\|_{L_2(\mu)} = 0$.*
3. *If $g \in \mathsf{P}$ then f can be computed in polynomial time.*

By Theorem 13, the function IP_d cannot be $\frac{1}{20}$-approximated w.r.t. the uniform distribution over $\{0,1\}^{2d}$ by neural networks of size $o(d/\log d)$. By Theorem 16, IP_d can be computed by a network of size $\mathcal{O}(d)$. Also, IP_d is clearly in P. Combining these results with Proposition 17, we obtain size-separation between networks of size $o(d/\log d)$ and size $\mathcal{O}(d)$:

Corollary 18 *There is a polynomial-time benign function $f : [0,1]^d \to [0,1]$ and a distribution μ on $[0,1]^d$ with a polynomially-bounded marginal density, such that f cannot be $\frac{1}{20}$-approximated by neural networks of size $o(d/\log d)$ w.r.t. μ, but there is a network $\tilde{N}$ of size $\mathcal{O}(d)$ such that $\|\tilde{N} - f\|_{L_2(\mu)} = 0$.*

Recall that by Remark 10, there is a barrier to showing size separation for polynomial-time benign functions, between size $\mathcal{O}(d\log^2(d))$ and some larger size. Closing the gap between the above size-separation result and the barrier is an interesting topic for future research.

Finally, our lower bounds for exact computation of Boolean functions (Corollary 15) allow us to obtain size separation also in the L_∞ sense. Namely, we show a polynomial-time benign function $f : [0,1]^d \to [0,1]$ that can be computed by a network of size $\mathcal{O}(d)$, but cannot be approximated in the L_∞ sense by networks of size $o(d)$. The function f is the function computed by the neural networks from Theorem 16.

Theorem 19 *There is a polynomial-time benign function $f : [0,1]^d \to [0,1]$ that can be computed by a neural network of size $\mathcal{O}(d)$, and every network N such that $\|f - N\|_\infty \leq \frac{1}{3}$ has size $\Omega(d)$.*

We prove the theorem in Appendix D.2. Note that the barrier to size separation from Remark 10 does not apply to approximation in the L_∞ sense.

Acknowledgments

We would like to thank Mika Goos, Thomas Watson, Sasha Golovnev, Arkadev Chattopadhyay, Suhail Sherif and T.S. Jayram for useful discussions. This research is supported in part by European Research Council (ERC) grant 754705.

References

Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 337–347. IEEE, 1986.

Ziv Bar-Yossef, Thathachar S Jayram, Ravi Kumar, and D Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14(1):115–133, 1994.

Guy Bresler and Dheeraj Nagaraj. Sharp representation theorems for relu networks with precise dependence on depth. *arXiv preprint arXiv:2006.04048*, 2020.

Arkadev Chattopadhyay, Shachar Lovett, and Marc Vinyals. Equality alone does not simulate randomness. In *34th Computational Complexity Conference (CCC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

Lijie Chen. Toward super-polynomial size lower bounds for depth-two threshold circuits. *arXiv preprint arXiv:1805.10698*, 2018.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Amit Daniely. Depth separation for neural networks. *arXiv preprint arXiv:1702.08489*, 2017.

Susanna F de Rezende, Jakob Nordström, and Marc Vinyals. How limited interaction hinders real communication (and what it means for proof and circuit complexity). In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 295–304. IEEE, 2016.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.

Magnus Gausdal Find, Alexander Golovnev, Edward A Hirsch, and Alexander S Kulikov. A better-than-3n lower bound for the circuit complexity of an explicit function. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 89–98. IEEE, 2016.

Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

Mikael Goldmann and Marek Karpinski. Simulating threshold circuits by majority circuits. *SIAM Journal on Computing*, 27(1):230–246, 1998.

Mikael Goldmann, Johan Håstad, and Alexander Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity*, 2(4):277–300, 1992.

Hans Dietmar Groeger and György Turán. A linear lower bound for the size of threshold circuits. *Bulletin-European Association For Theoretical Computer Science*, 50:220–220, 1993.

Ingo Gühring, Mones Raslan, and Gitta Kutyniok. Expressivity of deep neural networks. *arXiv preprint arXiv:2007.04759*, 2020.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2):251–257, 1991.

Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.

Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

Daniel M. Kane and Ryan Williams. Super-linear gate and super-quadratic wire lower bounds for depth-two and depth-three threshold circuits. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 633–643. ACM, 2016.

Pascal Koiran. Vc dimension in circuit complexity. In *Computational Complexity, 1996. Proceedings., Eleventh Annual IEEE Conference on*, pages 81–85. IEEE, 1996.

Jan Kraíček. Interpolation by a game. *Mathematical Logic Quarterly*, 44(4):450–458, 1998.

Matthias Krause and Stefan Lucks. Pseudorandom functions in tc^0 and cryptographic limitations to proving lower bounds. *computational complexity*, 10(4):297–313, 2001.

Eyal Kushilevitz and Noam Nisan. Communication complexity. 1997.

Shiyu Liang and Rayadurgam Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

Jianfeng Lu, Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network approximation for smooth functions. *arXiv preprint arXiv:2001.03040*, 2020.

Wolfgang Maass. Bounds for the computational power and learning complexity of analog neural nets. *SIAM Journal on Computing*, 26(3):708–732, 1997.

Wolfgang Maass, Georg Schnitger, and Eduardo D Sontag. On the computational power of sigmoid versus boolean threshold circuits. In *FOCS*, pages 767–776, 1991.

Eran Malach, Gilad Yehudai, Shai Shalev-Shwartz, and Ohad Shamir. The connection between approximation, depth separation and learnability in neural networks. *To appear*, 2021.

James Martens, Arkadev Chattopadhya, Toni Pitassi, and Richard Zemel. On the representational efficiency of restricted boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2877–2885, 2013.

Anirbit Mukherjee and Amitabh Basu. Lower bounds over boolean inputs for deep neural networks with relu gates. *arXiv preprint arXiv:1711.03073*, 2017.

Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)*, 51(2):231–262, 2004.

Noam Nisan. The communication complexity of threshold gates. *Combinatorics, Paul Erdos is Eighty*, 1:301–315, 1993.

Igor C Oliveira. *Unconditional lower bounds in complexity theory*. PhD thesis, Columbia University, 2015.

Philipp Petersen and Felix Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018.

Alexander A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992a.

Alexander A Razborov. On small depth threshold circuits. In *Scandinavian Workshop on Algorithm Theory*, pages 42–52. Springer, 1992b.

Alexander A Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

Vwani P Roychowdhury, Alon Orlitsky, and Kai-Yeung Siu. Lower bounds on threshold and related circuits via communication complexity. *IEEE Transactions on Information Theory*, 40(2):467–474, 1994.

Itay Safran and Ohad Shamir. Depth-width tradeoffs in approximating natural functions with neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2979–2987. JMLR. org, 2017.

Itay Safran, Ronen Eldan, and Ohad Shamir. Depth separations in neural networks: What is actually being separated? *arXiv preprint arXiv:1904.06984*, 2019.

Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network approximation characterized by number of neurons. *arXiv preprint arXiv:1906.05497*, 2019.

Matus Telgarsky. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*, 2016.

Gal Vardi and Ohad Shamir. Neural networks with small weights and depth-separation barriers. *arXiv preprint arXiv:2006.00625*, 2020.

R Ryan Williams. Limits on representing boolean functions by linear combinations of simple functions: thresholds, relus, and low-degree polynomials. *arXiv preprint arXiv:1802.09121*, 2018.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.

Dmitry Yarotsky. Optimal approximation of continuous functions by very deep relu networks. *arXiv preprint arXiv:1802.03620*, 2018.

Dmitry Yarotsky and Anton Zhevnerchuk. The phase diagram of approximation rates for deep neural networks. *arXiv preprint arXiv:1906.09477*, 2019.

Appendix A. Proofs for Section 3

A.1. Proof of Theorem 1

Let $f : [0,1]^d \to [0,1]$ be a benign function. Assume that f is L-Lipschitz for $L = \text{poly}(d) \geq 1$. Let $\epsilon = \frac{1}{\text{poly}(d)}$, and assume that for every neural network $\mathcal{N}$ of size $\text{poly}(d)$ and depth k we have $\|f - \mathcal{N}\|_{L_2(\mu)} > \epsilon$ (for some d). Let $p(d) = \frac{4L\sqrt{d}}{\epsilon}$, and let $c = \lceil \log(p(d)) \rceil$. Thus, $2^c \geq p(d)$. Let $\mathcal{I} = \{\frac{j}{2^c} : 0 \leq j \leq 2^c - 1, j \in \mathbb{Z}\}$. For $\tilde{x} \in \mathcal{I}$ we denote by $\text{bin}(\tilde{x}) \in \{0,1\}^c$ the binary representation of $0 \leq j \leq 2^c - 1$ such that $\tilde{x} = \frac{j}{2^c}$. For $\tilde{\mathbf{x}} \in \mathcal{I}^d$ we denote by $\text{bin}(\tilde{\mathbf{x}}) \in \{0,1\}^{c \cdot d}$ the concatenation of $\text{bin}(\tilde{x}_i)$ for $i = 1, \ldots, d$. For $\hat{\mathbf{x}} \in \{0,1\}^c$ we denote $\text{real}(\hat{\mathbf{x}}) = \frac{j}{2^c} \in \mathcal{I}$, where j is the integer whose binary representation is $\hat{\mathbf{x}}$. For $\hat{\mathbf{x}} \in \{0,1\}^{c \cdot d}$ we denote $\text{real}(\hat{\mathbf{x}}) \in \mathcal{I}^d$, such that the i-th component of $\mathrm{real}(\hat{\mathbf{x}})$ is $\mathrm{real}(\hat{x}_{(i-1)\cdot c+1}, \ldots, \hat{x}_{i\cdot c})$. Finally, for $x \in [0,1]$ we denote by $\mathrm{trunc}(x) \in \mathcal{I}$ the maximal $\tilde{x} \in \mathcal{I}$ such that $\tilde{x} \leq x$. Likewise, for $\mathbf{x} \in [0,1]^d$ we denote $\mathrm{trunc}(\mathbf{x}) = (\mathrm{trunc}(x_1), \ldots, \mathrm{trunc}(x_d)) \in \mathcal{I}^d$.

Since f is benign, there is an exponential-time algorithm $\mathcal{A}$, such that given $\hat{\mathbf{x}} \in \{0,1\}^{c\cdot d}$ it returns $\mathcal{A}(\hat{\mathbf{x}}) \in \{0,1\}^c$, such that

$$|f(\mathrm{real}(\hat{\mathbf{x}})) - \mathrm{real}(\mathcal{A}(\hat{\mathbf{x}}))| \leq \frac{1}{2^c} \leq \frac{1}{p(d)} . \tag{1}$$

Let $\hat{f} : \{0,1\}^{c\cdot d} \to \{0,1\}^c$ be the function that this algorithm computes. That is, $\hat{f}(\hat{\mathbf{x}}) = \mathcal{A}(\hat{\mathbf{x}})$. Assume that the function $\hat{f}$ can be computed by a threshold circuit T of size $\mathrm{poly}(d)$ and depth $k-2$. We will construct a neural network N of size $\mathrm{poly}(d)$ and depth k such that $\|f - N\|_{L_2(\mu)} \leq \epsilon$ and thus reach a contradiction. It implies that $\hat{f}$ can be computed in exponential time but cannot be computed by a threshold circuit of size $\mathrm{poly}(d)$ and depth $k-2$. Then, the theorem follows from the following lemma (see proof in Section A.1.1).

Lemma 20 *Let $l(d) = \mathcal{O}(\log(d))$ be monotonically non-decreasing and let $g : \{0,1\}^{d\cdot l} \to \{0,1\}^l$ be a function that can be computed in exponential time, and cannot be computed by a threshold circuit of size $\mathrm{poly}(d)$ and constant depth m. Then, there is a function $g' : \{0,1\}^{d'} \to \{0,1\}$ that can be computed in exponential time, and cannot be computed by a threshold circuit of size $\mathrm{poly}(d')$ and depth m, i.e., $g' \in \mathsf{EXP} \setminus \mathsf{TC}^0_m$.*

Let $\tilde{f} : [0,1]^d \to \mathcal{I}^d$ be such that $\tilde{f}(\mathbf{x}) = \mathrm{real}(\hat{f}(\mathrm{bin}(\mathrm{trunc}(\mathbf{x}))))$. Thus, $\tilde{f}$ transforms $\mathbf{x}$ to a $(c \cdot d)$-bits binary representation, runs $\hat{f}$, and converts the output from binary to a real value. Let $\mathbf{x} \in [0,1]^d$, let $\tilde{\mathbf{x}} = \mathrm{trunc}(\mathbf{x})$ and let $\hat{\mathbf{x}} = \mathrm{bin}(\tilde{\mathbf{x}})$. By Eq. 1 we have

$$|\tilde{f}(\mathbf{x}) - f(\tilde{\mathbf{x}})| = |\,\mathrm{real}(\hat{f}(\hat{\mathbf{x}})) - f(\mathrm{real}(\hat{\mathbf{x}}))| \leq \frac{1}{p(d)} = \frac{\epsilon}{4L\sqrt{d}} \leq \frac{\epsilon}{4} .$$

Also, since f is L-Lipschitz then we have

$$|f(\tilde{\mathbf{x}}) - f(\mathbf{x})| \leq L \cdot \|\tilde{\mathbf{x}} - \mathbf{x}\| \leq L \cdot \frac{\sqrt{d}}{2^c} \leq L \cdot \frac{\sqrt{d}}{p(d)} = \frac{L\sqrt{d}\cdot\epsilon}{4L\sqrt{d}} = \frac{\epsilon}{4} .$$

Thus, $|\tilde{f}(\mathbf{x}) - f(\mathbf{x})| \leq |\tilde{f}(\mathbf{x}) - f(\tilde{\mathbf{x}})| + |f(\tilde{\mathbf{x}}) - f(\mathbf{x})| \leq \frac{\epsilon}{2}$, and therefore $\|f - \tilde{f}\|_{L_2(\mu)} \leq \frac{\epsilon}{2}$.

We now construct a network N of size $\mathrm{poly}(d)$ and depth k such that $\|\tilde{f} - N\|_{L_2(\mu)} \leq \frac{\epsilon}{2}$. It implies that $\|f - N\|_{L_2(\mu)} \leq \|f - \tilde{f}\|_{L_2(\mu)} + \|\tilde{f} - N\|_{L_2(\mu)} \leq \epsilon$ and thus completes the proof. The network N consists of three parts. First, it transforms the input $\mathbf{x} \in [0,1]^d$ w.h.p. to $\mathrm{bin}(\mathrm{trunc}(\mathbf{x})) \in \{0,1\}^{c\cdot d}$. Then, it simulates the threshold circuit T. Finally, it converts the output of T from a binary representation to the corresponding real value. We now describe these parts in more details.

For the transformation from $\mathbf{x} \in [0,1]^d$ to $\mathrm{bin}(\mathrm{trunc}(\mathbf{x})) \in \{0,1\}^{c\cdot d}$ we will need the following lemma (see proof in Section A.1.2).

Lemma 21 *Let $\delta = \frac{1}{\mathrm{poly}(d)}$. There is a neural network $\mathcal{N}$ of depth 2, size $\mathrm{poly}(d)$, and $(c \cdot d)$ outputs, such that*

$$\Pr_{\mathbf{x}\sim\mu}\left[\mathcal{N}(\mathbf{x}) = \mathrm{bin}(\mathrm{trunc}(\mathbf{x}))\right] \geq 1 - \delta .$$

Also, for the simulation of the threshold circuit T we will need the following lemma (see proof in Section A.1.3).

Lemma 22 *Let T be a threshold circuit with d inputs, q outputs, depth m and size s. There is a neural network $\mathcal{N}$ with q outputs, depth $m+1$ and size $2s+q$, such that for every $\mathbf{x} \in \{0,1\}^d$ we have $\mathcal{N}(\mathbf{x}) = T(\mathbf{x})$. Moreover, for every input $\mathbf{x} \in \mathbb{R}^d$ the outputs of $\mathcal{N}$ are in $[0,1]$.*

We note that lemmas with a similar idea to Lemmas 21 and 22 where also shown in Vardi and Shamir (2020). The construction of N proceeds as follows. Let $\delta = \frac{\epsilon^2}{4}$. First N transforms w.p. at least $1-\delta$ the input $\mathbf{x} \in [0,1]^d$ to $\hat{\mathbf{x}} = \mathrm{bin}(\mathrm{trunc}(\mathbf{x})) \in \{0,1\}^{c \cdot d}$. By Lemma 21 it can be done by a depth-2 network $\mathcal{N}_1$. Second, N computes $T(\hat{\mathbf{x}})$. By Lemma 22 it can be done by a network $\mathcal{N}_2$ of depth $k-1$. Note that

$$T(\hat{\mathbf{x}}) = \hat{f}(\hat{\mathbf{x}}) = \mathrm{bin}(\mathrm{real}(\hat{f}(\hat{\mathbf{x}}))) = \mathrm{bin}(\tilde{f}(\mathbf{x})) \; .$$

Third, N transforms the output of $\mathcal{N}_2$ to the corresponding value in $\mathcal{I}$, and thus obtains $\tilde{f}(\mathbf{x})$. It can be done by a single layer, since if $\hat{\mathbf{z}} \in \{0,1\}^c$ is a binary representation of $z \in \mathcal{I}$, then

$$z = \sum_{j \in [c]} \hat{z}_j \cdot \frac{2^{j-1}}{2^c} \; . \tag{2}$$

Since the final layers in $\mathcal{N}_1$ and $\mathcal{N}_2$ do not have activations and can be combined with the next layers, and since the third part of N is simply a linear transformation, then the depth of N is k.

Given an input $\mathbf{x} \sim \mu$, the network N computes $\tilde{f}(\mathbf{x})$ w.p. at least $1-\delta$. However, it is possible (w.p. at most δ) that $\mathcal{N}_1$ fails to transform the input $\mathbf{x}$ to $\mathrm{bin}(\mathrm{trunc}(\mathbf{x}))$, and therefore N fails to compute $\tilde{f}(\mathbf{x})$. Still, even in this case we can bound the output of N as follows. If $\mathcal{N}_1$ fails to transform the input $\mathbf{x}$ to $\mathrm{bin}(\mathrm{trunc}(\mathbf{x}))$, then the input to $\mathcal{N}_2$ may contain values other than $\{0,1\}$. However, by Lemma 22, the network $\mathcal{N}_2$ outputs only values in $[0,1]$. Hence, when computing the output of N using Eq. 2, the resulting value is at least 0 and at most $(2^c - 1) \cdot \frac{1}{2^c} \leq 1$. Therefore, we have $N(\mathbf{x}) \in [0,1]$. Since $\tilde{f}(\mathbf{x}) \in [0,1]$, then $|\tilde{f}(\mathbf{x}) - N(\mathbf{x})| \leq 1$. We have

$$\mathop{\mathbb{E}}_{\mathbf{x} \sim \mu} \left(\tilde{f}(\mathbf{x}) - N(\mathbf{x}) \right)^2 \leq \delta \cdot 1^2 + (1-\delta) \cdot 0 = \delta = \frac{\epsilon^2}{4} \; .$$

Hence, $\|\tilde{f} - N\|_{L_2(\mu)} \leq \frac{\epsilon}{2}$ as required.

A.1.1. PROOF OF LEMMA 20

Let $g' : \{0,1\}^{d'} \to \{0,1\}$ be a function such that if $d' = d \cdot l + l$ then we have the following. Let $\mathbf{x} \in \{0,1\}^{d'}$ and denote $\mathbf{x}^1 = (x_1, \ldots, x_{d \cdot l})$ and $\mathbf{x}^2 = (x_{d \cdot l+1}, \ldots x_{d \cdot l + l})$. If $\mathbf{x}^2$ has a 1-bit in the i-th coordinate and all other bits are 0, then we say that $\mathbf{x}^2$ is the *i-selector*. For $\mathbf{x} \in \{0,1\}^{d'}$ such that $\mathbf{x}^2$ is i-selector, we have $g'(\mathbf{x}) = (g(\mathbf{x}^1))_i$. Namely, g' returns the i-th output bit of $g(\mathbf{x}^1)$. Since g can be computed in exponential time then clearly g' can also be computed in exponential time. Assume that g' can be computed by a threshold circuit T' of size $s(d') = \mathrm{poly}(d')$ and depth m. Then, g can also be computed by a $\mathrm{poly}(d)$-sized threshold circuit T of depth m as follows. The circuit T consists of l circuits $T_1, \ldots, T_l$, such that T_i computes the i-th output bit. The circuit T_i has input dimension $d \cdot l$, and is obtained from T' by hardwiring the input bits $\mathbf{x}^2$ to be the i-selector. That is, let n be a threshold gate in the first layer of T', and assume that the weight from the i-th component of $\mathbf{x}^2$ to n is w, and that the bias of n is b. Then, in T_i we change the bias of n to $b+w$. Note that T has size $l \cdot s(dl + l) = \mathrm{poly}(d)$ and depth m, and that T computes g.

A.1.2. PROOF OF LEMMA 21

Let $\mathbf{x} \in [0,1]^d$. In order to construct $\mathcal{N}$, we need to show how to compute $\text{bin}(\text{trunc}(x_i))$ for every $i \in [d]$. We will show a depth-2 network $\mathcal{N}'$ such that given $x_i \sim \mu_i$ it outputs $\text{bin}(\text{trunc}(x_i))$ w.p. $\geq 1 - \frac{\delta}{d}$. Then, the network $\mathcal{N}$ consists of d copies of $\mathcal{N}'$, and satisfies

$$\Pr_{\mathbf{x}\sim\mu}\left[\mathcal{N}(\mathbf{x}) \neq \text{bin}(\text{trunc}(\mathbf{x}))\right] \leq \sum_{i\in[d]} \Pr_{x_i\sim\mu_i}\left[\mathcal{N}'(x_i) \neq \text{bin}(\text{trunc}(x_i))\right] \leq \frac{\delta}{d}\cdot d = \delta\,.$$

We denote $\tilde{x}_i = \text{trunc}(x_i)$. For $j \in [c]$ let $I_j \subseteq \{0,\ldots,2^c-1\}$ be the integers such that the j-th bit in their binary representation is 1. Hence, given x_i, the network $\mathcal{N}'$ should output in the j-th output $\mathbb{1}_{I_j}(2^c \cdot \tilde{x}_i)$, where $\mathbb{1}_{I_j}(z) = 1$ if $z \in I_j$ and $\mathbb{1}_{I_j}(z) = 0$ otherwise.

Since μ has a polynomially-bounded marginal density, then there is $\Delta = \frac{1}{\text{poly}(d)}$ such that for every $i \in [d]$ and every $t \in [0,1]$ we have

$$\Pr_{\mathbf{x}\sim\mu}\left[x_i \in \left[t - \frac{\Delta}{2^c}, t\right]\right] \leq \frac{\delta}{2^c\cdot d}\,. \tag{3}$$

For an integer $0 \leq l \leq 2^c - 1$, let $g_l : \mathbb{R} \to \mathbb{R}$ be such that

$$g_l(t) = \left[\frac{1}{\Delta}(t - l + \Delta)\right]_+ - \left[\frac{1}{\Delta}(t-l)\right]_+\,.$$

Note that $g_l(t) = 0$ if $t \leq l - \Delta$, and that $g_l(t) = 1$ if $t \geq l$. Let $g'_l(t) = g_l(t) - g_{l+1}(t)$. Note that $g'_l(t) = 0$ if $t \leq l - \Delta$ or $t \geq l+1$, and that $g'_l(t) = 1$ if $l \leq t \leq l + 1 - \Delta$.

Let $h_j(t) = \sum_{l\in I_j} g'_l(t)$. Note that for every $l \in \{0,\ldots,2^c-1\}$ and $l \leq t \leq l+1-\Delta$ we have $h_j(t) = 1$ if $l \in I_j$ and $h_j(t) = 0$ otherwise. Therefore, if $h_j(2^c x_i) \neq \mathbb{1}_{I_j}(2^c\tilde{x}_i)$ then $2^c x_i \in [l+1-\Delta, l+1]$ for some integer $0 \leq l \leq 2^c - 1$.

Let $\mathcal{N}'$ be such that $\mathcal{N}'(x_i) = (h_1(2^c x_i), \ldots, h_c(2^c x_i))$. Note that $\mathcal{N}'$ can be implemented by a depth-2 neural network. We have:

$$\begin{aligned}
\Pr_{x_i\sim\mu_i}\left[\mathcal{N}'(x_i) \neq \text{bin}(\tilde{x}_i)\right] &= \Pr_{x_i\sim\mu_i}\left(\exists j \in [c] \text{ s.t. } h_j(2^c x_i) \neq (\text{bin}(\tilde{x}_i))_j\right) \\
&= \Pr_{x_i\sim\mu_i}\left[\exists j \in [c] \text{ s.t. } h_j(2^c x_i) \neq \mathbb{1}_{I_j}(2^c\tilde{x}_i)\right] \\
&\leq \Pr_{x_i\sim\mu_i}\left[2^c x_i \in [l+1-\Delta, l+1], 0 \leq l \leq 2^c-1\right] \\
&\leq \sum_{0\leq l\leq 2^c-1} \Pr_{x_i\sim\mu_i}\left[x_i \in \left[\frac{l}{2^c} + \frac{1}{2^c} - \frac{\Delta}{2^c}, \frac{l}{2^c} + \frac{1}{2^c}\right]\right] \\
&\overset{(Eq.\ 3)}{\leq} 2^c \cdot \frac{\delta}{2^c\cdot d} = \frac{\delta}{d}\,.
\end{aligned}$$

A.1.3. PROOF OF LEMMA 22

Let g be a gate in T, and let $\mathbf{w} \in \mathbb{Z}^l$ and $b \in \mathbb{Z}$ be its weights and bias. Let n_1 be a neuron with weights $\mathbf{w}$ and bias b, and let n_2 be a neuron with weights $\mathbf{w}$ and bias $b-1$. Let $\mathbf{y} \in \{0,1\}^l$. Since $(\langle\mathbf{w},\mathbf{y}\rangle + b) \in \mathbb{Z}$, we have $[\langle\mathbf{w},\mathbf{y}\rangle + b]_+ - [\langle\mathbf{w},\mathbf{y}\rangle + b - 1]_+ = \text{sign}(\langle\mathbf{w},\mathbf{y}\rangle + b)$. Hence, the gate g can be replaced by the neurons n_1, n_2. We replace all gates in T by neurons and obtain a network $\mathcal{N}$. Since each output gate of T is also replaced by two neurons, $\mathcal{N}$ has $m+1$ layers and size $2s+q$. Since for every $\mathbf{x} \in \mathbb{R}^d$, weight vector $\mathbf{w}$ and bias b we have $[\langle\mathbf{w},\mathbf{x}\rangle+b]_+ - [\langle\mathbf{w},\mathbf{x}\rangle+b-1]_+ \in [0,1]$ then for every input $\mathbf{x} \in \mathbb{R}^d$ the outputs of $\mathcal{N}$ are in $[0,1]$.

A.2. Proof of Theorem 6

Every neural network can be represented in a standard way by a binary vector, such that if the network has $\mathrm{poly}(d)$ neurons and the binary representation of each weight is of length at most $\mathrm{poly}(d)$, then the binary representation of the network is of length $\mathrm{poly}(d)$. In the following lemma we show that for a sufficiently large d, even the set of networks whose binary representations are of length $d^{\log(d)}$ does not suffice to approximate all 1-Lipschits functions.

Lemma 23 *Let $\epsilon = \frac{1}{d}$. There is a distribution μ with a polynomially-bounded marginal density, such that for every sufficiently large d we have the following: There is a 1-Lipscitz function $g : [0,1]^d \to [0,1]$ such that for every neural network N whose binary representation has $d^{\log(d)}$ bits, we have $\|g - N\|_{L_2(\mu)} > \epsilon$.*

Proof For $\mathbf{z} \in \{0,1\}^d$ we denote $A_{\mathbf{z}} = \{\mathbf{x} \in [0,1]^d : \forall i \in [d],\ |x_i - z_i| \leq \frac{1}{4}\}$. Thus, $A_{\mathbf{z}}$ is a cube with volume $\left(\frac{1}{4}\right)^d$. For $\mathbf{x} \in [0,1]^d$ we denote $\mathrm{dist}(\mathbf{x}, A_{\mathbf{z}}) = \min\{\|\mathbf{x} - \mathbf{a}\| : \mathbf{a} \in A_{\mathbf{z}}\}$. For $\mathbf{z} \in \{0,1\}^d$, let $h_{\mathbf{z}} : [0,1]^d \to [0,1]$ be such that $h_{\mathbf{z}}(\mathbf{x}) = \max\{0, \frac{1}{4} - \mathrm{dist}(\mathbf{x}, A_{\mathbf{z}})\}$. Note that if $\mathbf{x} \in A_{\mathbf{z}}$ then $h_{\mathbf{z}}(\mathbf{x}) = \frac{1}{4}$, if there is $i \in [d]$ such that $|x_i - z_i| \geq \frac{1}{2}$ then $h_{\mathbf{z}}(\mathbf{x}) = 0$, and $h_{\mathbf{z}}$ is 1-Lipschitz. For $\psi : \{0,1\}^d \to \{0,1\}$ let $f_\psi : [0,1]^d \to [0,1]$ be such that $f_\psi(\mathbf{x}) = \sum_{\mathbf{z} \in \{0,1\}^d} \psi(\mathbf{z}) h_{\mathbf{z}}(\mathbf{x})$. Note that for every $\mathbf{z} \in \{0,1\}^d$ and $\mathbf{x} \in A_{\mathbf{z}}$ we have $f_\psi(\mathbf{x}) = \frac{1}{4}\psi(\mathbf{z})$. Moreover, since for every $\mathbf{x} \in [0,1]$ there is at most one $\mathbf{z} \in \{0,1\}^d$ such that $h_{\mathbf{z}}(\mathbf{x}) \neq 0$, then f_ψ is also 1-Lipschitz. Let $\mathcal{F} = \{f_\psi : \psi \in \{0,1\}^{(\{0,1\}^d)}\}$. Let μ be the uniform distribution over $([0,1/4] \cup [3/4,1])^d = \bigcup_{\mathbf{z} \in \{0,1\}^d} A_{\mathbf{z}}$. Note that μ has a polynomially-bounded marginal density.

Let $\mathcal{G}$ be the set of all 1-Lipschitz functions $g : [0,1]^d \to [0,1]$. Note that $\mathcal{F} \subseteq \mathcal{G}$. Let $\mathcal{H}$ be a set of functions, such that for every $g \in \mathcal{G}$ there exists a function $h \in \mathcal{H}$ such that $\|g - h\|_{L_2(\mu)} \leq \epsilon$. We now show a lower bound on the size of $\mathcal{H}$. Then, we will use this bound to show that the set of networks whose binary representations are of length $d^{\log(d)}$ does not suffice to approximate $\mathcal{G}$.

Since $\mathcal{F} \subseteq \mathcal{G}$, then for every $f \in \mathcal{F}$ there exists a function $h \in \mathcal{H}$ such that $\|f - h\|_{L_2(\mu)} \leq \epsilon$. For a function $\varphi \in \mathcal{H} \cup \mathcal{F}$ and $\delta > 0$, we denote $B_\delta(\varphi) = \{f \in \mathcal{F} : \|f - \varphi\|_{L_2(\mu)} \leq \delta\}$. Let $h \in \mathcal{H}$. We will first bound the size of $B_\epsilon(h)$ and then use it to obtain a lower bound for $|\mathcal{H}|$. For every $f_1, f_2 \in B_\epsilon(h)$, we have $\|f_1 - f_2\|_{L_2(\mu)} \leq 2\epsilon$. Hence, for every $f' \in B_\epsilon(h)$, we have $B_\epsilon(h) \subseteq B_{2\epsilon}(f')$. Let $\psi' \in \{0,1\}^{(\{0,1\}^d)}$ be such that $f' = f_{\psi'}$. Let $\psi \in \{0,1\}^{(\{0,1\}^d)}$ be such that $f_\psi \in B_{2\epsilon}(f')$. We have

$$
\begin{aligned}
(2\epsilon)^2 \geq \left\|f_{\psi'} - f_\psi\right\|^2_{L_2(\mu)} &= \int_{\mathbf{x} \in [0,1]^d} \left(f_{\psi'}(\mathbf{x}) - f_\psi(\mathbf{x})\right)^2 \mu(\mathbf{x}) d\mathbf{x} \\
&= \sum_{\mathbf{z} \in \{0,1\}^d} \int_{\mathbf{x} \in A_{\mathbf{z}}} \left(f_{\psi'}(\mathbf{x}) - f_\psi(\mathbf{x})\right)^2 \cdot \frac{1}{2^d \cdot \left(\frac{1}{4}\right)^d} d\mathbf{x} \\
&= 2^d \cdot \sum_{\mathbf{z} \in \{0,1\}^d} \int_{\mathbf{x} \in A_{\mathbf{z}}} \frac{1}{16} \left(\psi'(\mathbf{z}) - \psi(\mathbf{z})\right)^2 d\mathbf{x} \\
&= \frac{2^d}{16} \cdot \sum_{\mathbf{z} \in \{0,1\}^d} \left(\frac{1}{4}\right)^d \mathbb{1}(\psi'(\mathbf{z}) \neq \psi(\mathbf{z})) \,.
\end{aligned}
$$

Therefore, $(2\epsilon)^2 \cdot 16 \cdot 2^d \geq \sum_{\mathbf{z} \in \{0,1\}^d} \mathbb{1}(\psi'(\mathbf{z}) \neq \psi(\mathbf{z}))$. Thus, for every $f_\psi \in B_{2\epsilon}(f')$, the function ψ disagrees with ψ' in at most $(2\epsilon)^2 \cdot 16 \cdot 2^d$ points. Hence, we have

$$|B_\epsilon(h)| \leq |B_{2\epsilon}(f')| \leq \sum_{j=0}^{(2\epsilon)^2 \cdot 16 \cdot 2^d} \binom{2^d}{j} \leq (2^d+1)^{(2\epsilon)^2 \cdot 16 \cdot 2^d} \leq 2^{(d+1) \cdot (2\epsilon)^2 \cdot 16 \cdot 2^d} .$$

Since $\mathcal{F} = \bigcup_{h \in \mathcal{H}} B_\epsilon(h)$, then

$$|\mathcal{H}| \geq \frac{|\mathcal{F}|}{|B_\epsilon(h)|} \geq \frac{2^{2^d}}{2^{(d+1) \cdot (2\epsilon)^2 \cdot 16 \cdot 2^d}} \geq \frac{2^{2^d}}{2^{8 \cdot 16 d \epsilon^2 \cdot 2^d}} = 2^{\left(1 - 8 \cdot 16 d \epsilon^2\right) 2^d} .$$

By plugging-in $\epsilon = \frac{1}{d}$, we have for a suffciently large d that

$$|\mathcal{H}| \geq 2^{\left(1 - \frac{8 \cdot 16}{d}\right) 2^d} \geq 2^{2^{d-1}} . \tag{4}$$

Let $\mathcal{N}$ be the set of all functions that can be expressed by a neural network whose representation has $d^{\log(d)}$ bits. By Eq. 4, if for every 1-Lipschitz function $g \in \mathcal{G}$ there exists a network $N \in \mathcal{N}$ such that $\|g - N\|_{L_2(\mu)} \leq \epsilon$, then $|\mathcal{N}| \geq 2^{2^{d-1}}$. However, for a sufficiently large d we have

$$|\mathcal{N}| = 2^{d^{\log(d)}} = 2^{2^{\log^2(d)}} < 2^{2^{d-1}} .$$

Therefore, for every sufficiently large d, there is a 1-Lipschitz function $g : [0,1]^d \to [0,1]$ such that for every network N whose binary representation has $d^{\log(d)}$ bits we have $\|g - N\|_{L_2(\mu)} > \epsilon$. ■

Finally, in order to prove the theorem we need to obtain a sequence of functions $\{f_d\}_{d=1}^\infty$ where $f_d : [0,1]^d \to [0,1]$, and some $\epsilon = \frac{1}{\text{poly}(d)}$ and distribution μ with a polynomially-bounded marginal density. Then, we need to show that for every polynomial $p(d)$ and every sequence of neural networks $\{N_d\}_{d=1}^\infty$, where the input dimension of N_d is d and its size and number of bits in the weights are bounded by $p(d)$, we have for some d that $\|f_d - N_d\|_{L_2(\mu)} > \epsilon$. By Lemma 23, for every sufficiently large d there is a 1-Lipschitz function $g_d : [0,1]^d \to [0,1]$ such that for every network N_d whose binary representation has $d^{\log(d)}$ bits we have $\|g_d - N_d\|_{L_2(\mu)} > \epsilon$. Consider a sequence $\{f_d\}_{d=1}^\infty$ where for every sufficiently large d we choose $f_d = g_d$. Let $\{N_d\}_{d=1}^\infty$ be a sequence of networks such that their sizes and number of bits in the weights are bounded by some $\text{poly}(d)$. The length of the binary representation of N_d is bounded by some $\text{poly}(d)$, and hence for a sufficiently large d it is smaller than $d^{\log(d)}$. Hence, for a sufficiently large d we have $\|f_d - N_d\|_{L_2(\mu)} > \epsilon$.

Appendix B. Proofs for Section 4

B.1. Proof of Theorem 7

Let $f : [0,1]^d \to [0,1]$ be a semi-benign function. Assume that f is L-Lipschitz for $L = 2^{\text{poly}(d)}$. Let $\epsilon = \frac{1}{\text{poly}(d)}$, let $q(d) = \frac{4L\sqrt{d}}{\epsilon}$, and let $c = \lceil \log(q(d)+1) \rceil$. Let $\mathcal{I} = \{\frac{j}{2^c} : 0 \leq j \leq 2^c - 1, j \in \mathbb{Z}\}$. Note that since L is exponential, then $q(d)$ is also exponential, and that $c = \text{poly}(d)$. We use the notations $\text{bin}(\cdot), \text{real}(\cdot)$ and $\text{trunc}(\cdot)$ in an analogous way to the proof of Theorem 1.

Let $c' = \log(4/\epsilon)$. Since f is semi-benign, there is an exponential-time algorithm $\mathcal{A}$, such that given $\hat{\mathbf{x}} \in \{0,1\}^{c \cdot d}$ it computes $f(\text{real}(\hat{\mathbf{x}}))$ within precision of c' bits. In order to simplify notations, we assume that $\mathcal{A}(\hat{\mathbf{x}}) \in \{0,1\}^c$. Namely, $\mathcal{A}$ computes a c'-bits approximation of $f(\text{real}(\hat{\mathbf{x}}))$ and then pads it with zeros to obtain the c-bits output. Thus, we have

$$|\,\text{real}(\mathcal{A}(\hat{\mathbf{x}})) - f(\text{real}(\hat{\mathbf{x}}))| \leq \frac{1}{2^{c'}} = \frac{\epsilon}{4} \,. \tag{5}$$

Let $\hat{f} : \{0,1\}^{c \cdot d} \to \{0,1\}^c$ be the function that this algorithm computes. That is, $\hat{f}(\hat{\mathbf{x}}) = \mathcal{A}(\hat{\mathbf{x}})$. Assume that the function $\hat{f}$ can be computed by a threshold circuit T of size $\text{poly}(d)$. We will construct a neural network N of size $\text{poly}(d)$ such that $\|f - N\|_{L_2(\mu)} \leq \epsilon$ and thus reach a contradiction. It implies that $\hat{f}$ can be computed in exponential time but cannot be computed by a $\text{poly}(d)$-sized threshold circuit. Then, the theorem follows from the following lemma (whose proof is similar to the proof of Lemma 20).

Lemma 24 *Let $l(d) \leq \text{poly}(d)$ be monotonically non-decreasing and let $g : \{0,1\}^{d \cdot l} \to \{0,1\}^l$ be a function that can be computed in exponential time, and cannot be computed by a threshold circuit of size $\text{poly}(d)$. Then, there is a function $g' : \{0,1\}^{d'} \to \{0,1\}$ that can be computed in exponential time, and cannot be computed by a threshold circuit of size $\text{poly}(d')$, i.e., $g' \in$ EXP $\setminus$ P/poly.*

Let $\tilde{f} : [0,1]^d \to \mathcal{I}^d$ be such that $\tilde{f}(\mathbf{x}) = \text{real}(\hat{f}(\text{bin}(\text{trunc}(\mathbf{x}))))$. Thus, $\tilde{f}$ transforms $\mathbf{x}$ to a $(c \cdot d)$-bits binary representation, computes $\hat{f}$, and converts the output from binary to a real value. Let $\mathbf{x} \in [0,1]^d$, let $\tilde{\mathbf{x}} = \text{trunc}(\mathbf{x})$ and let $\hat{\mathbf{x}} = \text{bin}(\tilde{\mathbf{x}})$. By Eq. 5 we have

$$|\tilde{f}(\mathbf{x}) - f(\tilde{\mathbf{x}})| = |\,\text{real}(\hat{f}(\hat{\mathbf{x}})) - f(\text{real}(\hat{\mathbf{x}}))| \leq \frac{\epsilon}{4} \,.$$

Also, since f is L-Lipschitz then we have

$$|f(\tilde{\mathbf{x}}) - f(\mathbf{x})| \leq L \cdot \|\tilde{\mathbf{x}} - \mathbf{x}\| \leq L \cdot \frac{\sqrt{d}}{2^c} \leq L \cdot \frac{\sqrt{d}}{q(d)} = \frac{L\sqrt{d} \cdot \epsilon}{4L\sqrt{d}} = \frac{\epsilon}{4} \,.$$

Thus, $|\tilde{f}(\mathbf{x}) - f(\mathbf{x})| \leq |\tilde{f}(\mathbf{x}) - f(\tilde{\mathbf{x}})| + |f(\tilde{\mathbf{x}}) - f(\mathbf{x})| \leq \frac{\epsilon}{2}$, and therefore $\|f - \tilde{f}\|_{L_2(\mu)} \leq \frac{\epsilon}{2}$.

We now construct a $\text{poly}(d)$-sized network N such that $\|\tilde{f} - N\|_{L_2(\mu)} \leq \frac{\epsilon}{2}$. It implies that $\|f - N\|_{L_2(\mu)} \leq \|f - \tilde{f}\|_{L_2(\mu)} + \|\tilde{f} - N\|_{L_2(\mu)} \leq \epsilon$ and thus completes the proof.

The construction of N follows a similar idea to the proof of Theorem 1: First, N transforms w.p. at least $1 - \frac{\epsilon^2}{4}$ the input $\mathbf{x} \in [0,1]^d$ to $\hat{\mathbf{x}} = \text{bin}(\text{trunc}(\mathbf{x})) \in \{0,1\}^{c \cdot d}$. Then, it computes $T(\hat{\mathbf{x}})$ by simulating the threshold circuit T using Lemma 22. Finally, it transforms $T(\hat{\mathbf{x}})$ from a binary representation to the corresponding real value. The main difference from the proof of Theorem 1, is that since L is exponential, then the number of bits in $\hat{\mathbf{x}}$ is polynomial in d (rather than logarithmic), and hence computing this binary representation with a $\text{poly}(d)$-sized network requires a more clever construction. In the following lemma, we show that the transformation from $\mathbf{x} \in [0,1]^d$ to $\text{bin}(\text{trunc}(\mathbf{x})) \in \{0,1\}^{c \cdot d}$ can be implemented, using the construction of Telgarsky (2016), with a $\text{poly}(d)$-sized network. Then, the construction of N and the proof that it approximates $\tilde{f}$ follow similar arguments to the proof of Theorem 1.

Lemma 25 *Let $\delta = \frac{1}{\text{poly}(d)}$. There is a neural network $\mathcal{N}$ of size $\text{poly}(d)$ and $(c \cdot d)$ outputs, such that*

$$\Pr_{\mathbf{x} \sim \mu} \left[\mathcal{N}(\mathbf{x}) = \text{bin}(\text{trunc}(\mathbf{x}))\right] \geq 1 - \delta \,.$$

Proof Let $\mathbf{x} \in [0,1]^d$. In order to construct $\mathcal{N}$, we need to show how to compute $\text{bin}(\text{trunc}(x_i))$ for every $i \in [d]$. We construct a network $\mathcal{N}'$ such that for every $i \in [d]$, given $x_i \sim \mu_i$ it outputs $\text{bin}(\text{trunc}(x_i))$ w.p. at least $1 - \frac{\delta}{d}$. Then, the network $\mathcal{N}$ consists of d copies of $\mathcal{N}'$, and satisfies

$$\Pr_{\mathbf{x}\sim\mu}\left[\mathcal{N}(\mathbf{x}) \neq \text{bin}(\text{trunc}(\mathbf{x}))\right] \leq \sum_{i\in[d]} \Pr_{x_i\sim\mu_i}\left[\mathcal{N}'(x_i) \neq \text{bin}(\text{trunc}(x_i))\right] \leq \frac{\delta}{d} \cdot d = \delta \,.$$

The network $\mathcal{N}'$ consists of c networks $\mathcal{N}_1, \ldots, \mathcal{N}_c$, such that for every $i \in [d]$ the network $\mathcal{N}_j$ computes the j-th bit of $\text{bin}(\text{trunc}(x_i))$ w.p. at least $1 - \frac{\delta}{d \cdot c}$ over μ_i. Hence, the network $\mathcal{N}'$ satisfies

$$\Pr_{x_i\sim\mu_i}\left[\mathcal{N}'(x_i) \neq \text{bin}(\text{trunc}(x_i))\right] \leq \sum_{j\in[c]} \Pr_{x_i\sim\mu_i}\left[\mathcal{N}_j(x_i) \text{ fails}\right] \leq \frac{\delta}{d \cdot c} \cdot c = \frac{\delta}{d} \,.$$

We now construct $\mathcal{N}_j$. Note that in order to compute the j-th bit of $\text{bin}(\text{trunc}(x_i))$, the network $\mathcal{N}_j$ needs to oscillate $\mathcal{O}(2^j)$ many times. Hence, unlike the depth-2 construction from Lemma 21, the network $\mathcal{N}_j$ requires $\text{poly}(d)$ depth. We note that a similar construction was used in Safran and Shamir (2017).

In the following, we assume that $x_i \notin \mathcal{I} \cup \{1\}$, i.e., $x_i \cdot 2^c$ is not an integer. Note that since μ has a polynomially-bounded marginal density then the probability that $x_i \in \mathcal{I} \cup \{1\}$ is 0, and hence we can ignore this case. Let $\varphi(z) = [2z]_+ - [4z-2]_+$. Telgarsky (2016) observed that the composition of φ with itself j times, denoted by φ^j, yields a highly oscillatory triangle wave function. In the domain $[0,1]$, the function φ^j consists of 2^{j-1} identical triangles of height 1. Note that for $z \leq 0$ we have $\varphi^j(z) = 0$. Given $z \in (0,1) \setminus \mathcal{I}$, note that the j-th bit of $\text{bin}(\text{trunc}(z))$ is 1 iff the following expression is at least $\frac{1}{2}$:

$$\varphi^j\left(z - \frac{1}{2} \cdot \frac{1}{2^j}\right) .$$

Hence, given $x_i \sim \mu_i$, the network $\mathcal{N}_j$ should return w.p. at least $1 - \frac{\delta}{d \cdot c}$ the expression

$$\mathbb{1}_{\geq \frac{1}{2}}\left(\varphi^j\left(x_i - 2^{-j-1}\right)\right) ,$$

where $\mathbb{1}_{\geq \frac{1}{2}}(y) = 1$ if $y \geq \frac{1}{2}$ and is 0 otherwise. While the function $\mathbb{1}_{\geq \frac{1}{2}}$ cannot be expressed by a ReLU network, it can be approximated by

$$h_\Delta(y) = \left[\frac{1}{\Delta}\left(y - \frac{1}{2} + \frac{\Delta}{2}\right)\right]_+ - \left[\frac{1}{\Delta}\left(y - \frac{1}{2} - \frac{\Delta}{2}\right)\right]_+ .$$

Note that $h_\Delta(y) = 0$ for every $y \leq \frac{1}{2} - \frac{\Delta}{2}$, and $h_\Delta(y) = 1$ for every $y \geq \frac{1}{2} + \frac{\Delta}{2}$. Since μ has a polynomially-bounded marginal density, then by choosing a sufficiently small $\Delta = \frac{1}{\text{poly}(d)}$, we have w.p. at least $1 - \frac{\delta}{d \cdot c}$ over $x_i \sim \mu_i$, that

$$h_\Delta\left(\varphi^j\left(x_i - 2^{-j-1}\right)\right) = \mathbb{1}_{\geq \frac{1}{2}}\left(\varphi^j\left(x_i - 2^{-j-1}\right)\right) .$$

Finally, the l.h.s. of the above equation can be implemented by a $\text{poly}(d)$-sized neural network $\mathcal{N}_j$. The construction of such a network is straightforward, since it is a composition of a $\text{poly}(d)$ number of functions, that can be implemented by ReLU networks of size $\text{poly}(d)$. ■

B.2. Proof of Theorem 9

The proof uses ideas from the proofs of Theorems 1 and 7 with some necessary modifications. Let $f : [0,1]^d \to [0,1]$ be a polynomial-time benign function. Assume that f is L-Lipschitz for $L = \text{poly}(d)$. Let $\epsilon = \frac{1}{\text{poly}(d)}$, let $p(d) = \frac{4L\sqrt{d}}{\epsilon}$, and let $c = \lceil \log(p(d)+1) \rceil$. Let $\mathcal{I} = \{\frac{j}{2^c} : 0 \leq j \leq 2^c - 1, j \in \mathbb{Z}\}$. We use the notations $\text{bin}(\cdot), \text{real}(\cdot)$ and $\text{trunc}(\cdot)$ in a similar way to the proof of Theorem 1.

Since f is polynomial-time benign, there is a polynomial-time algorithm $\mathcal{A}$, such that given $\hat{\mathbf{x}} \in \{0,1\}^{c \cdot d}$ it returns $\mathcal{A}(\hat{\mathbf{x}}) \in \{0,1\}^c$, such that

$$|f(\text{real}(\hat{\mathbf{x}})) - \text{real}(\mathcal{A}(\hat{\mathbf{x}}))| \leq \frac{1}{2^c} \leq \frac{1}{p(d)} .$$

Let $\hat{f} : \{0,1\}^{c \cdot d} \to \{0,1\}^c$ be the function that this algorithm computes. That is, $\hat{f}(\hat{\mathbf{x}}) = \mathcal{A}(\hat{\mathbf{x}})$. Assume that the function $\hat{f}$ can be computed by a threshold circuit T of size $\mathcal{O}(d \log^2(d))$. We will construct a neural network N of size $\mathcal{O}(d \log^2(d))$ such that $\|f - N\|_{L_2(\mu)} \leq \epsilon$ and thus reach a contradiction. It implies that $\hat{f}$ can be computed in polynomial time but cannot be computed by threshold circuits of size $\mathcal{O}(d \log^2(d))$. Then, the theorem follows from the following lemma.

Lemma 26 *Let $l(d) = \mathcal{O}(\log(d))$ be monotonically non-decreasing and let $g : \{0,1\}^{d \cdot l} \to \{0,1\}^l$ be a function that can be computed in polynomial time, and cannot be computed by threshold circuits of size $\mathcal{O}(d \log^2(d))$. Then, there is a function $g' : \{0,1\}^{d'} \to \{0,1\}$ in P, that cannot be computed by threshold circuits of size $\mathcal{O}(d')$.*

Proof We first define g' in a similar manner to the proof of Lemma 20. Let $g' : \{0,1\}^{d'} \to \{0,1\}$ be a function such that if $d' = d \cdot l + l$ then we have the following. Let $\mathbf{x} \in \{0,1\}^{d'}$ and denote $\mathbf{x}^1 = (x_1, \ldots, x_{d \cdot l})$ and $\mathbf{x}^2 = (x_{d \cdot l+1}, \ldots x_{d \cdot l+l})$. If $\mathbf{x}^2$ has a 1-bit in the i-th coordinate and all other bits are 0, then we say that $\mathbf{x}^2$ is the i-*selector*. For $\mathbf{x} \in \{0,1\}^{d'}$ such that $\mathbf{x}^2$ is i-selector, we have $g'(\mathbf{x}) = (g(\mathbf{x}^1))_i$. Namely, g' returns the i-th output bit of $g(\mathbf{x}^1)$.

Since g can be computed in polynomial time then clearly g' can also be computed in polynomial time. Assume that g' can be computed by a threshold circuit T' of size $c' \cdot d'$ for some constant c'. Then, g can be computed by a threshold circuit T of size $\mathcal{O}(d \log^2(d))$ as follows. The circuit T consists of l circuits $T_1, \ldots, T_l$, such that T_i computes the i-th output bit. The circuit T_i has input dimension $d \cdot l$, and is obtained from T' by hardwiring the input bits $\mathbf{x}^2$ to be the i-selector. That is, let n be a threshold gate in the first layer of T', and assume that the weight from the i-th component of $\mathbf{x}^2$ to n is w, and that the bias of n is b. Then, in T_i we change the bias of n to $b + w$. Note that T has size $l \cdot c'd' = c'l(dl + l) = \mathcal{O}(d \log^2(d))$, and that T computes g. ■

The construction of the network N is done in a similar manner to the proof of Theorem 1, with some necessary modifications. Note that here the size of N should be $\mathcal{O}(d \log^2(d))$. However, the transformation from $\mathbf{x} \in [0,1]^d$ to $\text{bin}(\text{trunc}(\mathbf{x}))$ from Lemma 21 requires a larger size. Hence, we use here the construction from the proof of Lemma 25. Then, transforming $\mathbf{x} \in [0,1]^d$ to $\text{bin}(\text{trunc}(\mathbf{x})) \in \{0,1\}^{c \cdot d}$ requires only $\mathcal{O}(d \cdot c^2)$ neurons. Indeed, for every $i \in [d]$, the computation of each bit in $\text{bin}(\text{trunc}(x_i))$ requires at most $\mathcal{O}(c)$ neurons ($\mathcal{O}(c)$ layers of constant width). Since $c = \mathcal{O}(\log(d))$, then the total size required for this transformation is $\mathcal{O}(d \log^2(d))$. Now, note that the simulation of the threshold circuit T can be done by Lemma 22. Since T is of size $\mathcal{O}(d \log^2(d))$ then simulating T with a neural network requires size $\mathcal{O}(d \log^2(d))$. Overall, the size of N is $\mathcal{O}(d \log^2(d))$.

Appendix C. Proofs for Section 5

C.1. Proof of Theorem 11

For a Boolean function f, we denote by $R_\epsilon(f)$ the randomized communication complexity with error ϵ. Namely, the minimal cost of a protocol that computes f correctly with probability at least $1-\epsilon$ on every input and partition. Note that $R(f) = R_{1/3}(f)$. The following lemma, which provides an upper bound on the communication complexity of LTFs, was shown by Nisan (1993) :

Lemma 27 *Let $g = L_{\mathbf{a}=(a_1 \dots a_m),\theta}$ be an LTF. Then $R_\epsilon(g) = \mathcal{O}(\log m + \log \epsilon^{-1})$.*

We stress that no assumption in this Lemma is made on the real weights of the LTF g. The proof builds on a classical result that every LTF with m inputs can be computed by an LTF with integer weights whose absolute values are at most $\mathcal{O}(2^{m \log m})$ (cf. Goldmann and Karpinski (1998); Goldmann et al. (1992)).

We first handle the case of ReLU networks. Then, we will explain how to extend the proof to the case of k-piecewise-linear activation functions.

Lemma 28 *Let N be a ReLU network, computing a function $h : \{0,1\}^d \to \{0,1\}$ with $R(h) = \Omega(d)$. Then, N has size $\Omega(d/\log d)$.*

Proof We first outline the proof for depth-2 networks. Suppose that there is a depth-2 ReLU network with s hidden neurons computing the function h . Assume w.l.o.g. that $s = \mathcal{O}(d)$. Using Lemma 27, the two parties can determine whether the output of each of the s hidden neurons is zero or positive, where the probability of error for each neuron is $\mathcal{O}(d^{-2})$ and the total amount of communication is $\mathcal{O}(s \log d)$. With this information each active ReLU neuron becomes a linear function and the whole network "collapses" to a single linear function, for which we wish to determine the sign of the output. Thus, it remain to compute a single threshold gate. The parties can infer the "updated" weights of this threshold gate, namely the real coefficient of every input x_i in the threshold gate (observe that we may and do assume that both players know the weights of all neurons). Then, by using Lemma 27 again, the parties can determine the sign of the output, with an additional communication cost of $\mathcal{O}(\log d)$ bits. Overall, the cost of the protocol is $\mathcal{O}(s \log d)$, and it succeeds with probability $1 - o(1)$. Since by our assumption the cost of the protocol must be $\Omega(d)$, then $s = \Omega(d/\log d)$.

As an illustrative example consider a ReLU network with 6 inputs $x_1, \dots x_6$ and three ReLUs at the hidden layer $[ax_1 + a'x_2]_+$, $[bx_3 + b'x_4]_+$ and $[cx_5 + c'x_6]_+$ (where a, a', b, b', c, c' are real constants) feeding to an output neuron that computes the sum (i.e., all weights are 1). If we know that the first gate evaluates to 0 and the others are positive, then the output of the network is positive iff $bx_3 + b'x_4 + cx_5 + c'x_6 > 0$.

For depth greater than 2 we can use the same method to deduce whether each ReLU neuron has positive or zero output starting with the neurons in the first hidden layer and then doing the same evaluation (positive vs. zero) for all neurons, evaluating all neurons of depth i before neurons of depth $i+1$ (and evaluating the neurons in a given layer in an arbitrary order). Once we know for all hidden neurons whether their output is positive or zero, the players can infer the linear function feeding into the topmost output neuron and evaluate its sign. ■

Neural networks with the ReLU activation function are a special case of more general networks where the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ of each neuron is *piecewise linear*. Namely, there are k

real numbers $c_1 < c_2 < ... < c_k$ such that if $c_i < x \leq c_{i+1}$ (for $1 \leq i < k$) then $\sigma(x) = a_i x + b_i$, if $x \leq c_1$ then $\sigma(x) = a_0 x + b_0$, and else $\sigma(x) = a_k x + b_k$, where for every i the parameters a_i, b_i are real numbers[4]. We now explain how to extend the communication complexity argument from Lemma 28 to prove lower bounds for such networks. Suppose that the network N that computes h has size $s = \mathcal{O}(d)$ and k-piecewise-linear activation functions. Using binary search on $[k]$ and Lemma 27, we can determine for each neuron (starting with the neurons in the first hidden layer and moving upward), which of the k linear functions is used in the output of the neuron, using $\mathcal{O}(\log d \log k)$ bits communicated per neuron. Once we determine this information for all neurons we have the linear function of the output neuron and we can evaluate the sign of the output with additional $\mathcal{O}(\log d)$ bits. The overall communication (also ensuring that the probability of error at every neuron is at most $\mathcal{O}(d^{-2})$) is $\mathcal{O}(s \log d \log k)$ and by the union bound[5], the probability of error is $o(1)$. Since by our assumption the cost of the protocol must be $\Omega(d)$, then $s = \Omega(d/(\log d \log k))$.

C.2. Proof of Theorem 13

Let $f = \text{IP}_d$, and let N be a ReLU network of size s such that $\|N - f\|_{L_2(\mathcal{U}(\{0,1\}^{2d}))} \leq \epsilon$. Let N' be a neural network of size $s + 2$ such that for every $\mathbf{z} \in \{0,1\}^{2d}$ we have: if $N(\mathbf{z}) \leq \frac{1}{3}$ then $N'(\mathbf{z}) = 0$, and if $N(\mathbf{z}) \geq \frac{2}{3}$ then $N'(\mathbf{z}) = 1$. Such a network can be obtained from N by adding two neurons, namely,

$$N'(\mathbf{z}) = [3N(\mathbf{z}) - 1]_+ - [3N(\mathbf{z}) - 2]_+ \ .$$

Note that for every $\mathbf{z} \in \{0,1\}^{2d}$ such that $|N(\mathbf{z}) - f(\mathbf{z})| \leq \frac{1}{3}$, we have $N'(\mathbf{z}) = f(\mathbf{z})$. Also, we have

$$\begin{aligned} \epsilon^2 \geq \|N - f\|^2_{L_2(\mathcal{U}(\{0,1\}^{2d}))} &= \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathcal{U}(\{0,1\}^{2d})} (N(\mathbf{z}) - f(\mathbf{z}))^2 \\ &\geq \left(\frac{1}{3}\right)^2 \cdot \Pr_{\mathbf{z} \sim \mathcal{U}(\{0,1\}^{2d})} \left[|N(\mathbf{z}) - f(\mathbf{z})| > \frac{1}{3} \right] \ , \end{aligned}$$

and therefore $\Pr_{\mathbf{z} \sim \mathcal{U}(\{0,1\}^{2d})} \left[|N(\mathbf{z}) - f(\mathbf{z})| > \frac{1}{3} \right] \leq 9\epsilon^2$. Thus, with probability at least $1 - 9\epsilon^2$ over $\mathbf{z} \sim \mathcal{U}(\{0,1\}^{2d})$ we have $N'(\mathbf{z}) = f(\mathbf{z})$.

We now use N' to obtain a protocol that computes f w.h.p. for every input $\mathbf{z} \in \{0,1\}^{2d}$ and every partition. Let $\mathbf{x}, \mathbf{y} \in \{0,1\}^d$. In order to compute $\text{IP}_d(\mathbf{x}, \mathbf{y})$, the players first use their shared randomness to generate $\mathbf{x}', \mathbf{y}' \sim \mathcal{U}(\{0,1\}^d)$. Note that

$$\text{IP}_d(\mathbf{x}, \mathbf{y}) = \text{IP}_d(\mathbf{x} + \mathbf{x}', \mathbf{y} + \mathbf{y}') + \text{IP}_d(\mathbf{x} + \mathbf{x}', \mathbf{y}') + \text{IP}_d(\mathbf{x}', \mathbf{y} + \mathbf{y}') + \text{IP}_d(\mathbf{x}', \mathbf{y}') \mod 2 \ . \quad (6)$$

Also, note that $\mathbf{x}+\mathbf{x}'$ and $\mathbf{y}+\mathbf{y}'$ are distributed uniformly on $\{0,1\}^d$ (where the addition is $\mod 2$). Thus, by computing $\text{IP}_d(\mathbf{x} + \mathbf{x}', \mathbf{y} + \mathbf{y}'), \text{IP}_d(\mathbf{x} + \mathbf{x}', \mathbf{y}'), \text{IP}_d(\mathbf{x}', \mathbf{y} + \mathbf{y}'), \text{IP}_d(\mathbf{x}', \mathbf{y}')$ the players can compute $\text{IP}_d(\mathbf{x}, \mathbf{y})$. Note that by the union bound, with probability at least $1 - 4 \cdot 9\epsilon^2$ over the choice of $\mathbf{x}', \mathbf{y}'$ we have $N'(\mathbf{x}+\mathbf{x}', \mathbf{y}+\mathbf{y}') = \text{IP}_d(\mathbf{x}+\mathbf{x}', \mathbf{y}+\mathbf{y}')$, $N'(\mathbf{x}+\mathbf{x}', \mathbf{y}') = \text{IP}_d(\mathbf{x}+\mathbf{x}', \mathbf{y}')$, $N'(\mathbf{x}', \mathbf{y} + \mathbf{y}') = \text{IP}_d(\mathbf{x}', \mathbf{y} + \mathbf{y}')$ and $N'(\mathbf{x}', \mathbf{y}') = \text{IP}_d(\mathbf{x}', \mathbf{y}')$.

Since both players know $\mathbf{x}', \mathbf{y}'$, then they can compute $N'(\mathbf{x}', \mathbf{y}')$ without communicating. Now, the players compute the signs of $N'(\mathbf{x}+\mathbf{x}', \mathbf{y}+\mathbf{y}')$, $N'(\mathbf{x}+\mathbf{x}', \mathbf{y}')$, $N'(\mathbf{x}', \mathbf{y}+\mathbf{y}')$ using the protocol

4. No further assumptions are made on σ: it does not have to be continuous nor monotone.

5. We can assume $k = 2^{\mathcal{O}(d/\log d)}$ as otherwise the claim in the theorem is trivial.

described in the proof of Lemma 28 (we assume here w.l.o.g. that $s = \mathcal{O}(d)$). It will succeed with probability $1 - o(1)$, and its cost is $\mathcal{O}(s \log d)$. Finally, the players compute $\text{IP}_d(\mathbf{x}, \mathbf{y})$ with Eq. 6. Overall, the probability for an error is at most $4 \cdot 9\epsilon^2 + o(1)$. By plugging in $\epsilon = \frac{1}{20}$, we obtain that this probability is at most $1/3$. Since there is a linear lower bound on the randomized communication complexity of IP_d, and since we obtained a protocol with cost $\mathcal{O}(s \log d)$, then we have $s = \Omega(d/\log d)$.

C.3. Proof of Theorem 14

Lemma 29 *Let $g = L_{\mathbf{a}=(a_1 \ldots a_d),\theta}$ be an LTF. Then $CC^{\mathbb{R}}(g) = 1$.*

Proof As before we may assume that the players know the weights and the bias of the LTF. Suppose that $A \subseteq [d]$ is the set of indices of bits Alice gets and $B \subseteq [d]$ is the set of indices of bits Bob gets. Alice sends $\alpha = \sum_{i \in A} a_i x_i$ and Bob sends $\beta = \theta - \sum_{j \in B} a_j y_j$. The output of f can be decided based on whether $\alpha > \beta$, concluding the proof. ■

Since we can use real communication to evaluate the sign of the output of a ReLU neuron and of a threshold gate with one round of communication, we get using a similar argument to the proof of Lemma 28:

Corollary 30 *Let f be a Boolean function with d inputs. Suppose that $CC^{\mathbb{R}}(f) = \Omega(d)$. Then, any threshold circuit or ReLU network computing f has size $\Omega(d)$.*

Moreover, for a k-piecewise-linear activation function, we can determine which linear function is active in the output of a neuron with $\mathcal{O}(\log k)$ cost in the real communication model, using binary search. A similar reasoning to the proof of Theorem 11 yields:

Corollary 31 *Let f be a Boolean function with d inputs. Suppose that $CC^{\mathbb{R}}(f) = \Omega(d)$. Then, any neural network with a k-piecewise-linear activation function computing f has size $\Omega(d/\log k)$.*

C.4. Real communication complexity of IP_d

Theorem 32 *We have $CC^{\mathbb{R}}(IP_d) = \Omega(d)$.*

Proof Recall that the *communication matrix* of a Boolean function $f(\mathbf{x}, \mathbf{y})$ where $\mathbf{x}$ and $\mathbf{y}$ are binary vectors of length d is a $2^d \times 2^d$ matrix M_f where $M_f(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}, \mathbf{y})$. By Chattopadhyay et al. (2019) (Lemma 3.5 and Lemma 3.7) if M_f has $\alpha 2^{2d}$ ones and any 1-monochromatic rectangle R has $|R| \leq \beta 2^{2d}$, it holds that $CC^{\mathbb{R}}(f) = \Omega(\log(\alpha(\beta^{\eta-1}))$ for any $\eta \in (1/2, 1)$. By Lindsey's Lemma, any 1-monochromatic rectangle R satisfies $|R| \leq 2^d$. It follows that for $f = \text{IP}_d$ we have $\beta \leq 2^{-d}$. As α is roughly $1/2$ we have that $CC^{\mathbb{R}}(\text{IP}_d) = \Omega(d)$. ■

C.5. Proof of Theorem 16

Given input $\mathbf{x}, \mathbf{y} \in \{0, 1\}^d$, we have:

$$\text{DISJ}_d(\mathbf{x}, \mathbf{y}) = \neg \bigvee_{i \in [d]} (x_i \wedge y_i) \; .$$

Since $x_i \wedge y_i = [x_i + y_i - 1]_+$ then implementing $x_i \wedge y_i$ requires a single neuron for every $i \in [d]$. Implementing $\neg \bigvee_i z_i = [\sum_i(-z_i) + 1]_+$ also requires a single neuron.

Likewise, we have

$$\text{IP}_d(\mathbf{x}, \mathbf{y}) = \bigoplus_{i \in [d]} (x_i \wedge y_i) .$$

Implementing $x_i \wedge y_i$ requires a single neuron for every $i \in [d]$. Implementing $\bigoplus_i z_i$ requires evaluating the parity of $\sum_i z_i$. Computing the parity bit of an integer $1 \leq j \leq d$ can be done by a network of size $\mathcal{O}(d)$ using a straightforward construction, similar to the construction from the proof of Lemma 21.

Appendix D. Proofs for Section 6

D.1. Proof of Proposition 17

For $\mathbf{z} \in \{0,1\}^d$ we denote $A_{\mathbf{z}} = \{\mathbf{x} \in [0,1]^d : \forall i \in [d],\ |x_i - z_i| \leq \frac{1}{4}\}$. Thus, $A_{\mathbf{z}}$ is a cube with volume $\left(\frac{1}{4}\right)^d$. For $\mathbf{x} \in [0,1]^d$ we denote $\text{dist}(\mathbf{x}, A_{\mathbf{z}}) = \min\{\|\mathbf{x} - \mathbf{a}\| : \mathbf{a} \in A_{\mathbf{z}}\}$. For $\mathbf{z} \in \{0,1\}^d$, let $h_{\mathbf{z}} : [0,1]^d \to [0,1]$ be such that $h_{\mathbf{z}}(\mathbf{x}) = \max\{0, 1 - 4 \cdot \text{dist}(\mathbf{x}, A_{\mathbf{z}})\}$. Note that if $\mathbf{x} \in A_{\mathbf{z}}$ then $h_{\mathbf{z}}(\mathbf{x}) = 1$, if there is $i \in [d]$ such that $|x_i - z_i| \geq \frac{1}{2}$ then $h_{\mathbf{z}}(\mathbf{x}) = 0$, and $h_{\mathbf{z}}$ is 4-Lipschitz. Let $f : [0,1]^d \to [0,1]$ be such that $f(\mathbf{x}) = \sum_{\mathbf{z} \in \{0,1\}^d} g(\mathbf{z}) h_{\mathbf{z}}(\mathbf{x})$. Note that for every $\mathbf{z} \in \{0,1\}^d$ and $\mathbf{x} \in A_{\mathbf{z}}$ we have $f(\mathbf{x}) = g(\mathbf{z})$. Moreover, since for every $\mathbf{x} \in [0,1]$ there is at most one $\mathbf{z} \in \{0,1\}^d$ such that $h_{\mathbf{z}}(\mathbf{x}) \neq 0$, then f is also 4-Lipschitz. Let μ be the uniform distribution on $([0,1/4] \cup [3/4,1])^d = \bigcup_{\mathbf{z} \in \{0,1\}^d} A_{\mathbf{z}}$. Note that μ has a polynomially-bounded marginal density.

Part (1). Let $g' : \{0,3/4\}^d \to \{0,1\}$ be a function that corresponds to g, namely, $g'(\mathbf{z}) = g(\frac{4}{3}\mathbf{z})$ for every $\mathbf{z} \in \{0,3/4\}^d$. Since g cannot be ϵ-approximated by networks of size $\mathcal{O}(m)$ w.r.t. $\mathcal{U}(\{0,1\}^d)$, then g' cannot be ϵ-approximated by networks of size $\mathcal{O}(m)$ w.r.t. $\mathcal{U}(\{0,3/4\}^d)$. Assume that there is a neural network N of size $\mathcal{O}(m)$ such that $\|N - f\|_{L_2(\mu)} \leq \epsilon$. We show that there exists a network N' of the same size such that $\|N' - g'\|_{L_2(\mathcal{U}(\{0,3/4\}^d))} \leq \epsilon$. Thus, if f can be ϵ-approximated by a network of size $\mathcal{O}(m)$ then g' can also be ϵ-approximated by a network of size $\mathcal{O}(m)$, and hence we reach a contradiction.

For every $\mathbf{c} \in [0,1/4]^d$ we denote by $N_{\mathbf{c}}$ the neural network of size $\mathcal{O}(m)$ such that for every $\mathbf{x}$ we have $N_{\mathbf{c}}(\mathbf{x}) = N(\mathbf{x} + \mathbf{c})$. The network $N_{\mathbf{c}}$ is obtained from N by adding the appropriate bias terms to the neurons in the first hidden layer, and hence has size $\mathcal{O}(m)$. We now show that there exists $\mathbf{c} \in [0,1/4]^d$ such that $\|N_{\mathbf{c}} - g'\|_{L_2(\mathcal{U}(\{0,3/4\}^d))} \leq \epsilon$.

We have

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathbf{c} \sim \mathcal{U}([0,1/4]^d)} \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathcal{U}(\{0,3/4\}^d)} \left(N_{\mathbf{c}}(\mathbf{z}) - g'(\mathbf{z})\right)^2 &= \mathop{\mathbb{E}}_{\mathbf{c} \sim \mathcal{U}([0,1/4]^d)} \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathcal{U}(\{0,3/4\}^d)} \left(N(\mathbf{z} + \mathbf{c}) - f(\mathbf{z} + \mathbf{c})\right)^2 \\
&= \mathop{\mathbb{E}}_{\mathbf{x} \sim \mu} \left(N(\mathbf{x}) - f(\mathbf{x})\right)^2 \\
&= \|N - f\|_{L_2(\mu)}^2 \leq \epsilon^2 .
\end{aligned}$$

Hence, there exists $\mathbf{c} \in [0,1/4]^d$ such that $\mathbb{E}_{\mathbf{z} \sim \mathcal{U}(\{0,3/4\}^d)} \left(N_{\mathbf{c}}(\mathbf{z}) - g'(\mathbf{z})\right)^2 \leq \epsilon^2$, and therefore $\|N_{\mathbf{c}} - g'\|_{L_2(\mathcal{U}(\{0,3/4\}^d))} \leq \epsilon$.

Part (2). Let N be a neural network of size $\tilde{m}$ such that for every $\mathbf{z} \in \{0,1\}^d$ we have $N(\mathbf{z}) = g(\mathbf{z})$. Let $\tilde{N}$ be a network, that first transforms the input $\mathbf{x} \in ([0,1/4] \cup [3/4,1])^d$ to $\mathbf{z} \in \{0,1\}^d$ by rounding each component, and then computes $N(\mathbf{z})$. Note transforming $x_i \in [0,1/4] \cup [3/4,1]$ to the corresponding $z_i \in \{0,1\}$ can be done with two neurons as follows:

$$z_i = \left[2x_i - \frac{1}{2}\right]_+ - \left[2x_i - \frac{3}{2}\right]_+ .$$

Hence, the size of $\tilde{N}$ is $\tilde{m} + 2d$. Also, we have

$$\|\tilde{N} - f\|^2_{L_2(\mu)} = \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{U}(([0,1/4] \cup [3/4,1])^d)} \left(\tilde{N}(\mathbf{x}) - f(\mathbf{x})\right)^2 = \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathcal{U}(\{0,1\}^d)} \left(N(\mathbf{z}) - g(\mathbf{z})\right)^2 = 0 .$$

Part (3). If $g \in \mathsf{P}$, then f can be computed in polynomial time as follows. Given an input $\mathbf{x}$ we find the nearest $\mathbf{z} \in \{0,1\}^d$, compute $g(\mathbf{z})$, and return $f(\mathbf{x}) = g(\mathbf{z}) \cdot \max\{0, 1 - 4 \cdot \text{dist}(\mathbf{x}, A_{\mathbf{z}})\}$.

D.2. Proof of Theorem 19

Let $g : \{0,1\}^d \to \{0,1\}$ be either the disjointness function or the inner product function. Let $f : [0,1]^d \to [0,1]$ be the function computed by the neural network $\mathcal{N}$ from Theorem 16 that corresponds to g. Thus, for every $\mathbf{x} \in \{0,1\}^d$ we have $f(\mathbf{x}) = g(\mathbf{x})$.

Given an input $\mathbf{x} \in [0,1]^d$, we can construct in time polynomial in d the network $\mathcal{N}$, as we describe in the proof of Theorem 16, and hence we can compute $f(\mathbf{x})$ in polynomial time. Moreover, by the construction in Theorem 16, the network $\mathcal{N}$ is of a constant depth and size $\mathcal{O}(d)$, and the absolute values of its weights are bounded by $\text{poly}(d)$, and therefore it computes a $\text{poly}(d)$-Lipschitz function. Hence, f is polynomial-time benign, and it is computed by a network of size $\mathcal{O}(d)$.

Let N be a neural network such that $\|f - N\|_\infty \leq \frac{1}{3}$. Let N' be a network such that if $N(\mathbf{x}) \leq \frac{1}{3}$ then $N'(\mathbf{x}) = 0$, and if $N(\mathbf{x}) \geq \frac{2}{3}$ then $N'(\mathbf{x}) = 1$. Such a network can be easily obtained from N. Indeed, let $y = N(\mathbf{x})$, then we define

$$N'(\mathbf{x}) = [3y - 1]_+ - [3y - 2]_+ .$$

Since for every $\mathbf{x} \in [0,1]^d$ we have $|f(\mathbf{x}) - N(\mathbf{x})| \leq \frac{1}{3}$, then for every $\mathbf{x}$ such that $f(\mathbf{x}) \in \{0,1\}$ we have $N'(\mathbf{x}) = f(\mathbf{x})$. The function f is such that for every $\mathbf{x} \in \{0,1\}^d$ we have $f(\mathbf{x}) = g(\mathbf{x}) \in \{0,1\}$, and thus $N'(\mathbf{x}) = f(\mathbf{x}) = g(\mathbf{x})$. Hence, N' computes g. By Corollary 15, the size of N' is $\Omega(d)$, and therefore the size of N is also $\Omega(d)$.