Property Testing of Computational Networks

Artur Czumaj
Department of Computer Science and DIMAP, University of Warwick, Coventry, UK
University of Cologne, Germany

Christian Sohler
University of Cologne, Germany

Abstract

In this paper we initiate the study of *property testing of weighted computational networks viewed as computational devices*. Our goal is to design property testing algorithms that for a given computational network with oracle access to the weights of the network, accept (with probability at least $\frac{2}{3}$) any network that computes a certain function (or a function with a certain property) and reject (with probability at least $\frac{2}{3}$) any network that is *far* from computing the function (or any function with the given property). We parameterize the notion of being far and want to reject networks that are (ε, δ)-*far*, which means that one needs to change an ε-fraction of the description of the network to obtain a network that computes a function that differs in at most a δ-fraction of inputs from the desired function (or any function with a given property).

To exemplify our framework, we present a case study involving simple neural Boolean networks with ReLU activation function. As a highlight, we demonstrate that for such networks, any near-constant function is testable in query complexity independent of the network's size. We also show that a similar result cannot be achieved in a natural generalization of the distribution-free model to our setting, and also in a related vanilla testing model. While the analysis of neural Boolean networks with ReLU activation function is rigorous and technically demanding, we consider it a testament to the robustness of our framework and the depth of our findings.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Computing methodologies → Neural networks

Keywords and phrases Property Testing, Testing Computational Networks

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2026.65

Category RANDOM

Related Version *Full Version*: `https://arxiv.org/abs/2512.07577` [19]

Funding *Artur Czumaj*: Research partially supported by the Key Profile Area (KPA): "Intelligent Methods for Earth System Sciences" at the University of Cologne and by the Centre for Discrete Mathematics and its Applications (DIMAP) at the University of Warwick.

Acknowledgements We would like to thank Gereon Frahling for his valuable discussions regarding the computational model. We would like to thank Nithin Varma for helpful discussions during the initial phase of research that lead to this paper.

1 Introduction

Since its development in the late 90s [46, 29], the area of *property testing* has emerged as an important area of modern theoretical computer science. It considers the relaxation of classical decision problems by distinguishing inputs that belong to a given set (have a given property $\mathcal{P}$) from those that are *"far"* from any input that belongs to the set (are *far* from having property $\mathcal{P}$). In its most classical setting (see, e.g., survey expositions in [28, 9]), a property $\mathcal{P}$ is a collection of objects (binary strings, graphs, etc), and being *"far"* is measured by the Hamming distance, namely, in how many places of its representation should an input object

be changed so as to have the property $\mathcal{P}$. As a useful and powerful way of relaxation of the classical decision problems, property testing has found numerous applications and established itself as a significant topic in the study of function properties, graph properties, and beyond. Examples include testing properties of functions such as being a low degree polynomial, being monotone, depending on a specified number of attributes, testing properties of graphs such as being bipartite and being triangle-free, testing properties of strings, and testing properties of geometric objects and visual images such as being well-clustered and being convex (for a more complete picture, see, e.g., [28, 9, 16, 18, 45] and the references therein). Furthermore, property testing is in the center of recent advances in *sublinear algorithms*, as property testing algorithms are frequently "superfast" and rely on inspecting small portions of objects and making the evaluation based on such an inspection.

In property testing one assumes an indirect access to the input and its representation that is provided by an *oracle*; e.g., one can query the value of a function on a *specific input*, or the existence of a *specific edge* of a graph. The *complexity of the tester* is the number of queries to the oracle.

In this paper, *we study property testing in the context of complex computational devices* (e.g., *computational networks*). We consider a setting in which one wants to study functions that are represented in an indirect way (e.g., as circuits or computational networks, or via complicated algebraic expressions) by computational devices used to determine the value of the functions. One may think about functions that are given *implicitly*, and in order to compute f one has to perform a possibly non-trivial computation on the input. For example, we may have a computational network that *computes* a function $f : \mathbb{R}^n \to \{0,1\}$; or we may have a function $f : \{0,1\}^n \to \mathbb{R}^r$ defined by a matrix $A \in \mathbb{R}^{r\times n}$ as $f(x) := Ax$, or $f : \{0,1\}^n \to \{0,1\}^r$ defined as[1] $f(x) := \mathrm{sgn}(\mathrm{ReLU}(Ax))$.

Our goal is to determine whether a given computational device computes a function having a given property or if it is *far* from any computational device computing function with a given property. For example, if for a matrix $A \in \mathbb{R}^{r\times n}$ we have $f(x) := Ax$, then, informally, we would call the "device" *Ax far from having property $\mathcal{P}$* if there is no matrix $A^* \in \mathbb{R}^{r\times n}$ that differs from A only in a small fraction of its entries (i.e., $\|A^* - A\|_0$ is small) such that function $f^*(x) := A^*x$ has the property $\mathcal{P}$. Furthermore, to make our study more general and applicable, we will allow to enhance our definition to devices that err on a small number of inputs.

Observe that while our model can be seen as a special case of the classical property testing framework (see, e.g., references to a similar setting in [28, Chapter 1]), the approach presented in our paper offers a different perspective than that most frequently studied: *our framework focuses on the computational device* – and its analysis should depend on specific parameters of the device rather than on the inputs or the function f. We want to determine if one has to make many changes in the device's representations in order to compute some function having desirable property $\mathcal{P}$, or only a small number of changes suffices. And so, for example, if for a matrix $A \in \mathbb{R}^{r\times n}$ we define $f(x) := Ax$ or $f(x) := \mathrm{sgn}(\mathrm{ReLU}(Ax))$ for any $x \in \mathbb{R}^n$, then it is natural to define the oracle to access individual entries from A (rather than to allow queries to the value of $f(x)$ for any specific input x).

[1] We use standard **ReLU** and **signum** functions to ensure that f has range $\{0,1\}^r$; recall that $\mathrm{ReLU}(z) := \max\{z, 0\}$ and $\mathrm{sgn}(z) := 1$ if $z > 0$, $\mathrm{sgn}(0) := 0$, $\mathrm{sgn}(z) := -1$ if $z < 0$. For vectors, the operations are for each coordinate. (Note that the use of these standard definitions make the combination $\mathrm{sgn}(\mathrm{ReLU}(\cdot))$ *not* redundant: $\mathrm{sgn}(\mathrm{ReLU}(Ax)) \not\equiv \mathrm{sgn}(Ax)$ for matrices A containing *negative* values – as it is the case for matrices and vectors studied in our paper.)

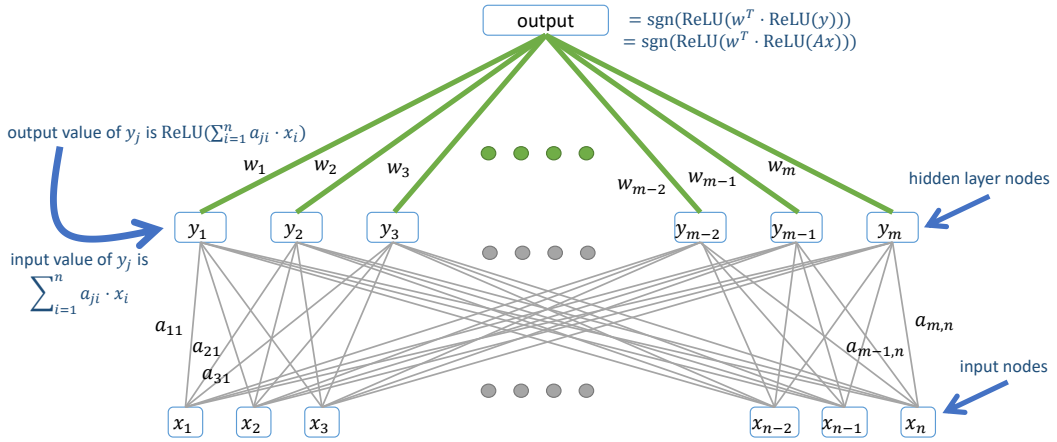

Figure 1 A ReLU network with n input nodes with inputs $x_1, \ldots, x_n$, a single hidden layer with m hidden layer nodes with values $y_1, \ldots, y_m$, and one output node. Every input node $x_i \in \{0,1\}$ is connected to every hidden layer node y_j by an edge with real weight $a_{ji} \in [-1,1]$ and every hidden layer node y_j is connected to the output node by an edge of real weight $w_j \in [-1,1]$. The value of node y_j is $\sum_{i=1}^n a_{ji}x_i$, which after applying the ReLU activation function gives $\mathrm{ReLU}(\sum_{i=1}^n a_{ji}x_i)$. The Boolean function computed by the network is equal to $\mathrm{sgn}\left(\mathrm{ReLU}\left(\sum_{j=1}^m \left(w_j \cdot \mathrm{ReLU}(\sum_{i=1}^n a_{ji}x_i)\right)\right)\right) \equiv \mathrm{sgn}\left(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(Ax))\right)$. A more general ReLU network with multiple inputs, multiple outputs, and multiple layers is depicted on Figure 2.

Case study: Testing basic neural ReLU networks. Our setting is very open-ended, and in order to make our study more concrete, we will consider our framework for a specific example of complex computational networks. Our goal is to take a representative example of computational networks that is generic and simple, yet complex enough to go beyond the study relying on random sampling. We will study an example of the most basic yet highly non-trivial *neural ReLU networks* seen as computational devices from *the point of view of property testing*. Consider a trained feedforward network and view it as a circuit that receives an input vector $x \in \{0,1\}^n$ and computes some output. In the most basic case of the network with one output, one hidden layer, and the ReLU activation function (see also Figure 1), when a network is interpreted as a binary classifier, for a vector $w \in [-1,1]^m$ and matrix $A \in [-1,1]^{m \times n}$ it computes a Boolean function of the form

$$f(x) := \mathrm{sgn}\left(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(Ax))\right).$$

(Our setting can be naturally extended to the study of more complex neural networks (functions) with *multiple outputs* and *multiple hidden layers*, where for matrices $W_0, \ldots, W_\ell$ with $W_i \in [-1,1]^{m_{i+1} \times m_i}$, we compute a function $f : \{0,1\}^{m_0} \to \{0,1\}^{m_{\ell+1}}$ defined as $f(x) := \mathrm{sgn}(\mathrm{ReLU}(W_\ell \cdot \ \ldots \ \cdot \mathrm{ReLU}(W_1 \cdot \mathrm{ReLU}(W_0 \cdot \mathrm{ReLU}(x))) \ldots))$, see Section 3.4.)

In order to facilitate our analysis in the property testing framework, we will consider property testing algorithms that are given the oracle access to the weights of the network[2] and the objective is to accept networks that compute functions that have a certain property of interest and reject networks that compute functions that are far from having that property for almost all inputs.

[2] Which for networks with one output and one hidden layer, is to query for the entries of A and w.

Our motivation. Our study is motivated by the modeling of computational networks in the framework of property testing, and to instantiate our framework in the context of basic neural networks. Taking the point of view of property testing and complexity theory, our goal is to obtain a better understanding of the relation between the (local) structure of a computational network and the function it computes. Our motivation stems from the fact that property testing has been a successful tool to reveal relations between the local structure of an object (here, a computational device) and its global properties that can provide combinatorial insights about the structure of an object. Examples of such insights include the characterization of property testing for dense graphs and its relation to the regularity lemma [1, 3], the relation of property testing of hyperfinite graphs to the theory of amenable groups [23, 39], or the line of research that started with testing connectivity [30] and then studied sublinear algorithms for approximating MST cost [13, 15, 17] and whose structural findings formed a building block in finding the currently fastest approximation algorithms for low dimensional MST approximation [4].

To showcase the versatility and power of our framework, we study *networks that are simple*, yet whose *computational properties are complex*. The reliance on *non-linear ReLU operators* is not only something that makes neural deep networks so powerful, but this also makes their study challenging (observe that, say, for $f(x) := Ax$, simple random sampling of entries of matrix A would provide sufficient information to understand many properties of such function; the non-linear ReLU operators make the study and the analysis more complex and interesting). Feedforward neural networks, as those studied in the simplest form in our paper, form a fundamental part of modern deep networks [34] such as transformers and the attention mechanism [49], which in turn are the main building block of large language models such as ChatGPT [43]. Unfortunately, despite their relatively simple setting, we still lack a good understanding of the computation process of these networks, leading to serious questions on the trust and transparency of the underlying learning algorithms. Therefore, even though *our main focus is on the complexity study*, we hope that the lens of property testing will contribute to a better understanding of the relation between the network structure and the computed function or properties of that function. Since the analysis of property testing algorithms relies on establishing close links between the object's local structure and the tested property, we believe that the study of simple yet fundamental models of ReLU networks may provide a new insight into such networks. Additionally, the basic notion of a relaxed decision problem in property testing has some appealing properties in the context of understanding neural networks: Typically, we would like a neural network to be robust to small changes in the network, as otherwise there is a higher risk of adversarial attacks [48]. Furthermore, during network training, *dropout* [47] is often used to improve generalization, i.e., a random fraction of input nodes is dropped during training for each training example.

1.1 Overview of our results

In this paper, we *introduce a property testing model for computational networks* through a study of *basic neural networks*. We first focus on a simple network structure: a fully connected feedforward network with the ReLU activation function, one hidden layer, and one output node (see Figure 1 and Definition 1). Later, we generalize our model to a more complex setting with multiple hidden layers and multiple output nodes (see Section 2.1 and Figure 2). We view the network as an *arithmetic circuit* that computes a Boolean function. Our objective is to design property testing algorithms that sample weights of the network and accept, if the network computes a function with a specific property, and reject, if the network is (ε, δ)-far from computing any function with the property, by which we mean that one has to change more than an ε-fraction of the network to obtain a function that differs from the target functions on less than a δ-fraction of the inputs.

An important contribution of this paper is *introduction of a model that allows a non-trivial study of computational networks (e.g., neural networks) from the lens of property testing.*

Next, we provide a *comprehensive study of our model*, focusing on the example of the constant 0 and the OR-function, for which we design constant-complexity property testers. Further, we consider other natural variants of our model for which we provide super-constant lower bounds demonstrating their complexity limitations. This shows that our model introduces a useful notion of being far that leads to interesting questions and results from the algorithmic and complexity perspective. Our results extend beyond the 0/OR-functions and we give some positive results showing the testability of some Boolean functions and classes of functions.

Our *main technical contributions* are (see Section 3 for more details):

(1) study of our model on the problem of *testing some very simple yet fundamental functions*,
(2) study of limitations of possibility of extending our results to a *distribution-free model*,
(3) *first steps towards understanding of constant time testable properties* in our model,
(4) extending the model to *more complex networks with multiple outputs and layers.*

In the following, after reviewing related work, we introduce our property testing model in detail. Then, in Section 3, we summarize our results and provide a technical overview. Due to space constraints, we omit the full technical details for the claims made in Section 3. However, to illustrate our analytical approach, Section 4 provides a complete proof of one key result. All remaining proofs and a comprehensive analysis are deferred to the full version [19].

1.2 Related work

Property testing. Since its development in the late 90s [46, 29], the area of property testing has emerged as an important area of modern theoretical computer science, see surveys in [9, 28, 44]. Although a close relationship between property testing and properties of functions, and in the context of learning has been known since the beginning of the area (these topics were explored as early as in [46, 29]), *the setting presented in this paper has received little attention.* The classical framework of testing properties of functions typically assumes an oracle access to the functions and/or the inputs rather than a computational device, see [9, 28]. (For example, most function property testers assume oracle access to the values of a given function on a specific input, and typically, graph testers assume access to the edges of the input graph.) Similarly, one normally studies networks in the context of their combinatorial properties rather than by analyzing "functions" they compute. The only work that we are aware of that considers a model similar to ours is the analysis of testing mixing property of Markov chains by Batu et al. [7]. We are not aware of any property testing work for basic logical or arithmetic circuits (even a related testing satisfiability in [2] uses a different setting), though some works on testing branching programs [25, 38] or whether a function can be represented by a small/shallow circuit/branching program [22, 24, 42], are of similar flavor. Other related works include property testing of some matrix properties (e.g., [5], where testing the rank of a matrix A has implications for some properties of Ax), but these works use a different setting, rely on linear operators, and do not apply to non-linear operators like those studied in our paper.

Neural networks. The complexity of neural networks has received significant attention in the past. In particular, threshold circuits have been studied as a model of computation and for corresponding complexity class TC^0 we know $\mathsf{AC}^0 \subsetneq \mathsf{TC}^0 \subseteq \mathsf{NC}^1$ [50]. The problem of verifying whether there exist weights such that an input circuit computes an input Boolean

function (threshold circuit loading) is known to be NP-complete [33], even when the circuit is restricted to three neurons [11]. For more details on the computational complexity of networks, see, e.g., [40].

In general, feedforward networks (acyclic networks) are universal approximators when viewed as computing a continuous function, i.e., they can arbitrarily well approximate any Borel-measurable function between finite dimensional spaces provided a sufficient number of neurons (nodes) in the hidden layer exist [32]. Furthermore, width-bounded networks (but generally with a large number of layers) are universal approximators [35]. The VC-dimension of neural networks with linear threshold gates has been studied and there is an upper bound of $O(m \log m)$ where m is the number of linear threshold gates [8]. These results imply bounds on the number of samples required to train the network. However, for our setting they do not seem to be directly relevant since one cannot evaluate a training example without querying the whole network.

In a related study, [20] investigated the problem of deciding if a neural network with real inputs is monotone and [37] considered monotone neural networks.

There has been extensive research on the computational complexity of training constant depth ReLU networks. It is known that even for a single ReLU unit, the problem of finding weights minimizing the squared error of a given training set is hard to approximate [27, 36, 21] (cf. [12] for a slightly different proof/result). Further results show that even for a single neuron, the training problem parameterized by the dimension d of the training data is $W[1]$-hard and there is a $n^{\Omega(d)}$ conditional lower bound assuming the exponential time hypothesis [26]. Studies have also explored learning linear combinations of ReLUs from Gaussian distributions (e.g., an algorithm for a constant number of ReLU units has been recently developed in [14]).

2 Property testing model

We begin our study with a feedforward network with one hidden layer and ReLU activation function. The network (see Figure 1) is fully connected between the layers, and has n input nodes labeled $\{1, \ldots, n\}$, a hidden layer with m nodes labeled $\{1, \ldots, m\}$, and one output node. (We later generalize our model to multiple hidden layers and output nodes, see Section 3.4.) We associate a *binary vector* $x = (x_1, \ldots, x_n)$ with the input nodes, so that input node i has value x_i. Between the input nodes and the hidden layer there is a *complete bipartite graph with real edge weights* from $[-1, 1]$. We define a_{ij} to be the weight between input node i and node j of the hidden layer. The *value of an input node is* x_i. The *value computed at a hidden layer node* j *is* $\sum_i a_{ji} x_i$. Thus, we can write the values at the hidden layer by a matrix multiplication Ax for an $m \times n$ matrix A. To the values at the hidden layer, we apply the *ReLU activation function*, which is defined as $\mathrm{ReLU}(y) := \max\{0, y\}$. For a vector $y = (y_1, \ldots, y_m)^T$, we define $\mathrm{ReLU}(y)$ to be $\mathrm{ReLU}(y) := (\mathrm{ReLU}(y_1), \ldots, \mathrm{ReLU}(y_m))^T$. We can now write the output of the hidden layer nodes as $\mathrm{ReLU}(Ax)$. Between the hidden layer and the output node there is a complete bipartite graph with real weights from $[-1, 1]$. Let w_j denote the weight of the edge connecting hidden layer node j to the output node. If $z = \mathrm{ReLU}(Ax)$ is the vector of hidden layer outputs, then $w^T z$ is the *value of output node*, where $w = (w_1, \ldots, w_m)^T$. Thus, on input $x \in \{0, 1\}^n$ the value at the output node is $w^T \cdot \mathrm{ReLU}(Ax)$. For further intuition, we refer the reader to Figure 1.

ReLU network as a binary classifier. We interpret the network as a *binary classifier*: the network assigns each input vector to one of two classes, defined as 0 and 1. If the value at the output node is at most 0 we assign class 0, if it is larger than 0 we assign class 1. In this paper

we focus on the setting when $x \in \{0,1\}^n$ is a binary vector. Thus, we view the neural network as a function $f : \{0,1\}^n \to \{0,1\}$ that computes 1 if the output value of the network is greater than 0, and 0 if the output value is at most 0. We write[3] $f(x) = \mathrm{sgn}(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(Ax)))$. For a given input $x \in \{0,1\}^n$, we call $f(x)$ the *output of the network*.

Our goal is to understand how the local structure of the network is related to the function f or to properties of that function. We will address this question by studying the function from a *property testing* point of view. This leads us to the following definition.

▶ **Definition 1 (ReLU network with *one hidden layer* and *single output*).** *A ReLU network with n input nodes, m hidden layer nodes, and one output node is a pair (A, w), where $A \in [-1,1]^{m \times n}$ and $w \in [-1,1]^m$. The binary function $f : \{0,1\}^n \to \{0,1\}$ computed by the ReLU network (A, w) is defined as*

$$f(x) := \mathrm{sgn}(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(Ax))).$$

2.1 Property testing: ReLU networks with *one hidden layer/one output*

We introduce a new property testing model for computational networks on an example of neural ReLU networks. We view (see Figure 1) the network as a weighted graph $G = (V, E)$, where $V = I \cup H \cup O$ with I being the set of *input nodes*, H the set of *hidden layer nodes*, and O the set of *output nodes* (frequently $|O| = 1$). The nodes in I, H and O are labelled from 1 to $|I|, |H|$ and $|O|$, respectively. Furthermore, E contains all edges between I and H, and between H and O. The edges between I and H are also referred to as *first layer edges* and the edges from H to O as *second layer edges*. We assume that we have *query access to the edge weights*: we can specify nodes number i in I and number j from H, and query the value a_{ij} of that edge in $O(1)$ time. Similarly, we can access the entries of w by specifying the corresponding hidden layer node.

Our objective is to design (randomized) sampling algorithms that decide whether a given neural network has a certain property or is far away from the property. To achieve this goal, we must first define a distance measure of neural networks. We view the network as a weighted graph and use the (relative) edit distance. However, since we want to take into consideration that there is a different number of edges on each layer, *for two networks (A_1, w_1) and (A_2, w_2)* with n input nodes, m hidden layer nodes, and one output node, we *define their distance* as $\max\left\{\frac{\|A_1 - A_2\|_0}{mn}, \frac{\|w_1 - w_2\|_0}{m}\right\}$, where we use the standard notation $\|\cdot\|_0$ to denote the number of non-zero entries in a matrix/vector.

In this paper, we view neural networks as computing Boolean functions and study the question whether a neural network computes a function that has a certain *property*. We begin with the study of networks that compute a specific function. Since there are often many networks that compute the same function, the question of whether a given network computes a certain function may be viewed as testing whether an input has a certain specific property in standard property testing.

Given a function $f : \{0,1\}^n \to \{0,1\}$, a *property tester* is an algorithm with query access to the weights of a network (with n input nodes and m hidden layer nodes) that accepts with probability at least $\frac{2}{3}$ every network that computes f and rejects with probability at least $\frac{2}{3}$

[3] Note: the outer ReLU is *not redundant* in $\mathrm{sgn}(\mathrm{ReLU}(\cdot))$ because the argument $w^T \cdot \mathrm{ReLU}(Ax)$ can be negative, in which case $\mathrm{sgn}(\cdot)$ would return $-1 \notin \{0,1\}$; see also the discussion in footnote 1.

every network that is *far* from computing f, where we parameterize the notion of being far with parameters ε, δ. Parameter ε refers to the distance between the networks and δ refers to the distance between the computed function with respect to the uniform distribution[4].

The model. Let us recall the notion of *property of Boolean functions*, which is any family $\mathcal{P} = \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$, where $\mathcal{P}_n$ is a set of Boolean functions $f : \{0,1\}^n \to \{0,1\}$.

▶ **Definition 2 (ReLU network far from a property of functions).** *Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes. (A, w) is called* ***(ε, δ)-close to computing a function $f : \{0,1\}^n \to \{0,1\}$ with property $\mathcal{P} = \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$****, if one can change the matrix A in at most εnm places and the weight vector w in at most εm places to obtain a ReLU network that computes a function g, such that there exists a function $f \in \mathcal{P}_n$ for which $\Pr[g(x) \neq f(x)] \leq \delta$, where x is chosen uniformly at random from $\{0,1\}^n$. If (A, w) is not (ε, δ)-close to computing f with property $\mathcal{P}$ then we say that it is* ***(ε, δ)-far from computing f with property $\mathcal{P}$****.*

If $|\mathcal{P}_n| = 1$ and $f \in \mathcal{P}_n$, then in Definition 2 we say that a *network is (ε, δ)-close/far from computing f.*

▶ Remark 3. It might seem highly undesirable that the notion of (ε, δ)-distance employed in this paper does not satisfy the triangle inequality. However, our model is a natural relaxation of the model with $\delta = 0$, which is exactly the standard model of property testing with a property $\mathcal{P}$ being the set of networks computing a given function (like the 0-function) [28]. *It is an inherent feature of property testing that the distance to a property/function often does not satisfy the triangle inequality*: Just as two graphs G and H being close to 3-colorable does not imply that G and H are close to each other, or two networks close to computing the 0-function are not necessarily close.

▶ **Definition 4.** *A property $\mathcal{P} = \bigcup_{n\in\mathbb{N}} \mathcal{P}_n$ of Boolean functions is* ***testable*** *(for ReLU networks)* ***with query complexity $q(\varepsilon, \delta, n, m)$****, if for every $n, m \in \mathbb{N}$, every $\varepsilon \in (0,1)$, and every $\delta \in (0,1)$ there exists an algorithm that receives parameters $n, m, \varepsilon, \delta$ as input and is given query access to an arbitrary ReLU network with n input nodes, m hidden layer nodes and one output node, and that*

- *queries the ReLU network in at most $q(\varepsilon, \delta, n, m)$ many places,*
- *accepts with probability at least $\frac{2}{3}$ every network that computes a function from $\mathcal{P}_n$, and*
- *rejects with probability at least $\frac{2}{3}$ every network that computes a function (ε, δ)-far from $\mathcal{P}$.*

If the function q depends only on ε and δ, then we just say that the property $\mathcal{P}$ is ***testable****.*[5]

A tester that accepts every function from $\mathcal{P}$ with probability 1 is called ***one-sided tester****.*

---

[4] While distance to a property is often parameterized using only the parameter ε, a relaxation with another parameter has been studied before. In fact, a very similar model has been used in [7] to test if a Markov chain is mixing. Similarly to our problem, the graph/Markov chain can be seen as a computational model parameterized in terms of ε and δ, where ε refers to the edit distance of the Markov chain and δ to the ℓ_1-distance between the t-step distributions. We believe that a similar relaxation is useful in our case as the relation between the network and the function is rather complex, and we have little hope to obtain results without this second parameter δ.

[5] In property testing, traditionally (cf. [9, 28]), the ultimate goal of the analysis is to *obtain testers whose query complexity is independent of the input size*, depending only on ε, δ. Testers with query complexity $o(mn)$ are of interest, but often not as the main objective. The importance of having the query complexity independent of ε and δ is especially amplified in our setting: if we query the ReLU network in $q(\varepsilon, \delta, n, m)$ many places, then normally the tester will be computing the function on the subnetwork determined by the tested edges *for all* 2^n *inputs*. However, the number of *relevant* input bits is always upper bounded by $q(\varepsilon, \delta, n, m)$ (since only input bits adjacent to the queried edges impact the function computed), and therefore if the query complexity is independent of the input size, then the number of inputs to be considered is at most $2^{q(\varepsilon,\delta)}$, which is independent on n and m. Hence, *testers with query complexity independent of n and m have their running time independent of n and m* too.

3 Technical overview

We initiate the study of property testing of computational networks on an example of neural networks. Below we presented a detailed discussion of our findings and of the technical challenges involved in our analysis. Due to space constraints, we omit the full technical details for the claims made here. All remaining proofs and a comprehensive analysis are deferred to the full version [19], except for one key result proven in Section 4.

3.1 Studying variants of the model on a simple function

We begin with the study of the constant 0-function (the same results are true for the OR function) in our basic model of feedforward *networks with ReLU activation function, one hidden layer, and single output*. We start by showing that the constant 0-function is testable in such model. This is the case even though, as we can easily show, the problem of verifying if the network computes the constant 0-function is NP-hard (note that if we had no ReLU we could simply compute $w^T A$ and check whether $w^T A$ has positive entries; so here ReLU makes the problem more difficult); see [19] for a formal proof. Our result is summarized in the following theorem (for a proof of part (1), see Section 4; for a proof of part (2) see [19]):

▶ **Theorem 5.** *Let (A, w) be a ReLU network with n input nodes, m hidden layer nodes, and a single output. Let $\delta \geq e^{-n/16}$, $\frac{1}{m} < \varepsilon < \frac{1}{2}$, and $0 < \lambda < \frac{1}{2}$.*

(1) *There is a tester that queries $O(\frac{\ln^2(1/\varepsilon\lambda)}{\varepsilon^6})$ entries from A and w, and (i) accepts with probability at least $\geq 1-\lambda$, if the ReLU network (A, w) computes the constant 0-function, and (ii) rejects with probability at least $1-\lambda$, if (A, w) is (ε, δ)-far from computing the constant 0-function.*

(2) *There is a tester that queries $O(\frac{\ln^2(1/\varepsilon\lambda)}{\varepsilon^6})$ entries from A and w, and (i) accepts with probability at least $1-\lambda$, if the ReLU network (A, w) computes the OR-function, and (ii) rejects with probability at least $1-\lambda$, if (A, w) is (ε, δ)-far from computing the OR-function.*

The query complexity of our tester does not depend on δ and we only require $\delta \geq e^{-\Omega(n)}$. *Assuming that n and m are polynomially related, one could also remove this dependence* by introducing a factor $\mathrm{polylog}(1/\delta)$ in the complexity, since for $\delta < e^{-n/16}$ we can read the whole network.

Observe that the *query complexity of our tester in Theorem 5 is independent of the network size*. Further, for constant ε and λ, the query complexity of our tester is $O(1)$.

Our property tester for the constant 0-function samples a subset of $\mathrm{poly}(1/\varepsilon)\cdot\mathrm{polylog}(1/\lambda)$ input and hidden layer nodes and evaluates whether the induced network with a *suitable bias* at the output node computes the constant 0-function. While this approach seems natural, the analysis and *its use of the bias* is non-standard.

The first step of our analysis shows that if the network is (ε, δ)-far from computing the constant 0-function then there is an input $x_0 \in \{0,1\}^n$ on which the network outputs value $\Omega(\varepsilon nm)$. Then, we show that sampling a constant number of hidden layer nodes approximates the scaled output value for this particular x_0 with sufficient accuracy, giving the scaled output value $\Omega(\varepsilon nm)$. In the next step, we show that we can sample a constant number of input nodes and approximate the value at the sampled hidden layer nodes so that the value at the output node is positive. Combining all the steps yields that if the network is (ε, δ)-far then our tester rejects with sufficient probability.

In order to show that the algorithm accepts a network computing the constant 0-function, we will argue that for all inputs $x \in \{0,1\}^n$ the sampled network computes the constant 0-function. For that, we take a different view and consider the sample from the input nodes

as being fixed. We then observe that restricting our network to the sampled input nodes can also be done by restricting the input to vectors $x \in \{0,1\}^n$ that are 0 for all input nodes that do not belong to the sample. This allow us to interpret our sampling approach as only sampling from the hidden layer. Hence, we can apply a concentration bound and take a union bound over all choices of input vectors that are zero on the nodes not in the sample. We also observe that while the analysis considers n-dimensional input vectors we know that these are 0 except for the sampled input nodes. Thus, in order to compute the output value we only need to query edges between sampled nodes.

Comparison to a vanilla testing algorithm. The above result implies that a network far from computing the 0-function (or from computing the OR-function) already exhibits this property across most of its small subnetworks. While this may not seem to be too surprising in the first place, it turns out to be a useful addition to a *vanilla testing algorithm* that *samples inputs uniformly at random* and *evaluates the network on these inputs*. This vanilla testing algorithm is arguably the most natural and routinely used approach to test the functionality of neural networks. In fact, it corresponds to the most *standard setting in property testing of functions* [28, 9] in which *the oracle allows to query the value of the network on a given input* x, rather than to query the network's weights. In our paper (see [19], we demonstrate the versatility of our property testing model and show that there are networks that *can be recognized as* (ε,δ)*-far* (for constant ε and exponentially small δ) from computing the 0-function *by our tester*, but the *vanilla tester in expectation requires an exponential in* n *number of samples* in order to find an input that does not evaluate to 0.

▶ **Theorem 6.** *Let* $n \in \mathbb{N}$, $0 < \varepsilon < \frac{1}{1000}$, $\delta < 2^{-(16\sqrt{\varepsilon}n+1)}$, *and* $\sqrt{\varepsilon}n$ *be integral.*

(1) *There is a ReLU network* (A_L, w_L) *with* n *hidden layer nodes and* n *input nodes such that with probability at most* $e^{-10\sqrt{\varepsilon}n/8}$, *a vector* x *chosen uniformly at random from* $\{0,1\}^n$ *satisfies* $w_L^T \cdot \mathrm{ReLU}(A_L x) > 0$.

(2) *The ReLU network* (A_L, w_L) *is* (ε,δ)*-far from computing the constant* 0*-function.*

Testing with one-sided error. We also consider the testability of the constant 0-function and the OR-function with *one-sided error*. We design a tester with one-sided error and query complexity of $\widetilde{O}(m/\varepsilon^2)$. The one-sided error property tester samples a subset of $O(\log m/\varepsilon^2)$ input nodes S and rejects only if there exists an input that is zero on all nodes outside of S and that evaluates to 1 or 0, respectively. If such an input is found, then it is a counter-example to computing the constant 0-function (or the OR-function, respectively) and the algorithm can reject.

Next, we give a near matching lower bound: any tester (for any constant ε) for the constant 0-function (or for the OR-function) that has one-sided error has query complexity of $\Omega(m)$. The lower bound follows from the fact that if too few weights are known, one can always "complete" the network to enforce it to compute the constant 0-function. Thus, the tester always accepts, but there are inputs that are (ε,δ)-far from computing the 0-function.

▶ **Theorem 7.**

(1) *Let* (A,w) *be a ReLU network with* n *input nodes and* m *hidden layer nodes. Let* $e^{-n/16} \leq \delta < 1$, $\frac{1}{m} < \varepsilon < \frac{1}{2}$ *and* $0 < \lambda < \frac{1}{2}$. *There is a tester that queries* $O(\frac{m \cdot \ln(m/\lambda)}{\varepsilon^2})$ *entries from* A *and* w *and always accepts, if the ReLU network* (A,w) *computes the constant* 0*-function, and rejects with probability at least* $1-\lambda$, *if the ReLU network* (A,w) *is* (ε,δ)*-far from computing the constant* 0*-function. Furthermore, the same result holds for the OR-function.*

(2) *Any tester for the constant* 0*-function (or for the OR-function) that has one-sided error has query complexity of* $\Omega(m)$.

Why testing of the 0-function? We consider the constant 0- and the OR-functions to be representative for natural, simple, yet rich, and highly non-trivial functions in our setting. Furthermore, note that for functions f and g, $f \equiv g$ iff the network $\mathrm{ReLU}(f-g)+\mathrm{ReLU}(g-f)$ computes the 0 function, and therefore testing the 0-function is closely related to the question of two networks computing the same output values for all inputs. Other related problems that might be in reach are dictatorship and juntas (which look like essentially equivalent to 0-testing, but they are not).

3.2 Lower bound in a *distribution-free model* (or on the model's choice)

It is natural to try to extend our framework to a *distribution-free model* (see, e.g., [29, Section 2], [28, Chapter 12.4]), where the notion of being (ε, δ)-far from a property defines the farness with respect to the second parameter δ in terms of *arbitrary distribution*. In the distribution-free model the distance between Boolean functions is measured with respect to an unknown distribution rather than the uniform distribution, as in our model considered above. It is also assumed that a property tester has sample access to this distribution. In our case, we will assume *query access to bits of the sample* since our goal is to be sublinear in the input size n or even to have a constant query complexity. While the study of this model is very tempting, unfortunately, we show a *lower bound that for any constant $c < 1$, rules out $O(n^c)$-time testers even for the constant* 0 *function*:

▶ **Theorem 8.** *For every constant $k > 1$, every property tester for the constant* 0*-function in the distribution-free model has a query complexity of $\Omega(n^{1-1/k})$.*

We will sketch here the main idea of the proof for $k = 2$ (where for purpose of exposition we use different constants than in the general proof, see [19]). We define two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ on pairs of a network and an input distribution such that the networks from the first distribution are (whp.) computing the constant 0-function and such that the pairs of network and input distribution $\mathcal{D}_I$ from the second distribution are (whp.) (ε, δ)-far from computing the constant 0-function with respect to $\mathcal{D}_I$.

For $\mathcal{D}_1$ the network will (whp.) compute the constant 0-function, so the underlying distribution is not relevant. To construct the network we partition the hidden layer nodes into sets N and P such that the nodes in N are connected to the output by edges of weight -1 and the nodes in P by edges of weight 1. In the first layer, the edges to nodes from N are 1 with probability $\frac{2}{3}$ and -1, otherwise, and the edges to nodes from P are 1 with probability $\frac{1}{2}$. Ignoring the effect of the ReLU it is not too hard to see that the network will evaluate (whp.) to 0 on any fixed input and one can apply a union bound to show that this holds also for all inputs. As we show in the proof, this intuition still holds under the ReLU function.

Networks from $\mathcal{D}_2$ are similar, except that there is a random matching of input nodes and the connections to the nodes from P are sampled for one node of the matching pair and copied for the other. This implies that with probability $\frac{1}{2}$ the pair of nodes contributes 2 to the hidden layer node, while the expected contribution to N is $\frac{8}{9}$ (here the edge weights are sampled independently and the matched pair of input nodes contributes only if both first layer edges are 1, which happens with probability $\frac{4}{9}$ and the contribution is 2). Thus, for any matching pair the expected contribution to the output is linear in the size of the hidden layer. We now define the unknown input distribution to be the uniform distribution on the pairs of matched inputs. The network is far from computing the 0-function since "fixing" one of the $\Omega(n)$ matched pairs requires $\Omega(m)$ changes in the network.

Finally, we observe that distinguishing between the two distributions is expensive as the answers to the queries are identically distributed as long as no matched pair is queried. Having access to samples from the distribution does not help as it is impossible to "find"

input bits that are 1. Since the matching of input bits is random, we prove that one needs $\Omega(\sqrt{n})$ samples to find a matching pair. We formalize this intuition and parameterize it by replacing pairs of inputs by k-sets, where the first $k-1$ inputs are chosen randomly with probability $\frac{1}{2}$ to be 1 (and -1, otherwise) and the last one is 1, if there is an odd number of 1s among the first $k-1$. We then modify our construction to get a lower bound for any k. The lower bound is one of our main technical contributions, relying on a novel controlling of the expectation of ReLU of sums of binomially distributed random variables.

3.3 First steps towards a classification of testable properties?

Our next goal is to try to understand *which properties of ReLU networks are testable*. Our first result shows that some classes of functions, *symmetric Boolean functions* or *monotone Boolean functions* have trivial testers (that always accept). We prove this by showing that every network is close to computing the constant 0 function or is close to the OR-function (with ε, δ not too small).[6]

▶ **Theorem 9.** *Let (A, w) be a ReLU network with n input nodes, m hidden layer nodes, and a single output. Let $0 < \delta \leq \frac{1}{2}$ and $\max\{\frac{1}{m}, 2\sqrt{\log(2/\delta)/n}\} \leq \varepsilon$. Then (A, w) is (ε, δ)-close to computing the constant 0-function or it is (ε, δ)-close to computing the OR-function.*

The proof first argues that the value at the output (before we apply sgn and ReLU) typically deviates from its expectation by at most $O(\sqrt{nm})$. Then we show that we can modify the network in an ε-fraction of its inputs and increase (or decrease) the expected value at the output bit by more than $\frac{1}{2}\varepsilon mn$. These two facts can be shown to yield Theorem 9.

Note that the property of monotone functions includes both the 0-function *and* the OR-function. Thus an immediate consequence of Theorem 9 is that *monotonicity has a trivial tester* that always accepts if $\varepsilon \geq \max\{\frac{1}{m}, 2\sqrt{\log(2/\delta)/n}\}$, and reads the whole input if $\varepsilon < \max\{\frac{1}{m}, 2\sqrt{\log(2/\delta)/n}\}$. When n and m are polynomially related, the query complexity of the tester is $\mathrm{poly}(1/\varepsilon) \cdot \mathrm{polylog}(1/\delta)$.

Finally, we study *monotone properties* (which are different objects than monotone functions). A property $\mathcal{P}$ is *monotone*, if it is closed under flipping bits in the truth table of any function having the property $\mathcal{P}$ from 0 to 1. One can think of monotone properties as being defined by the set of minimal functions under the operation of flipping zeros to ones. We call such a set a *generator*.

We show that every monotone property is testable with query complexity logarithmically on the size of the generator (we assume that the algorithm gets the generator as input; this assumption is justified as this depends on the property only). We prove the following result.

▶ **Theorem 10.** *Let $\frac{1}{m} < \varepsilon < 1$, $2e^{-n/16} \leq \delta \leq 1$. Let $\mathcal{P} = \bigcup_n \mathcal{P}_n$ be an $f(n)$-generatible monotone property with generator $G = \bigcup_n G_n$ with $|G_n| \leq f(n)$ for all n. There is a property tester with query access to a ReLU network $\mathcal{N}$ with n input nodes and m hidden layer nodes that (i) accepts with probability at least $1-\lambda$, if $\mathcal{N}$ computes a function in $\mathcal{P}$, (ii) rejects with probability at least $1-\lambda$, if $\mathcal{N}$ is (ε, δ)-far from computing a function in $\mathcal{P}$. The tester has query complexity $O(\ln(\frac{f(n)}{\delta\varepsilon\lambda})/(\delta\varepsilon^4))$.*

[6] We would like to stress that if two networks are close to computing a given function, then *they are not necessarily* pairwise close to each other (cf. Remark 3). Similarly, notice that while the closeness result in Theorem 9 may on the first glance seem to be unnatural, similar and even stronger claims are known in various scenarios in property testing. For example, in the classical dense graph model (see [28]), for $\varepsilon > \frac{1}{n}$, every graph is *both*, ε-close to be connected and ε-close to be disconnected (every graph on n vertices can be made connected using at most n edge changes, and can be made disconnected using at most n edge changes). Therefore, the fact that in our property testing framework all properties are close to computing the constant 0-function or close to computing the OR-function, does not mean that testing of these two properties is trivial, nor it shows that the framework considered here is unnatural.

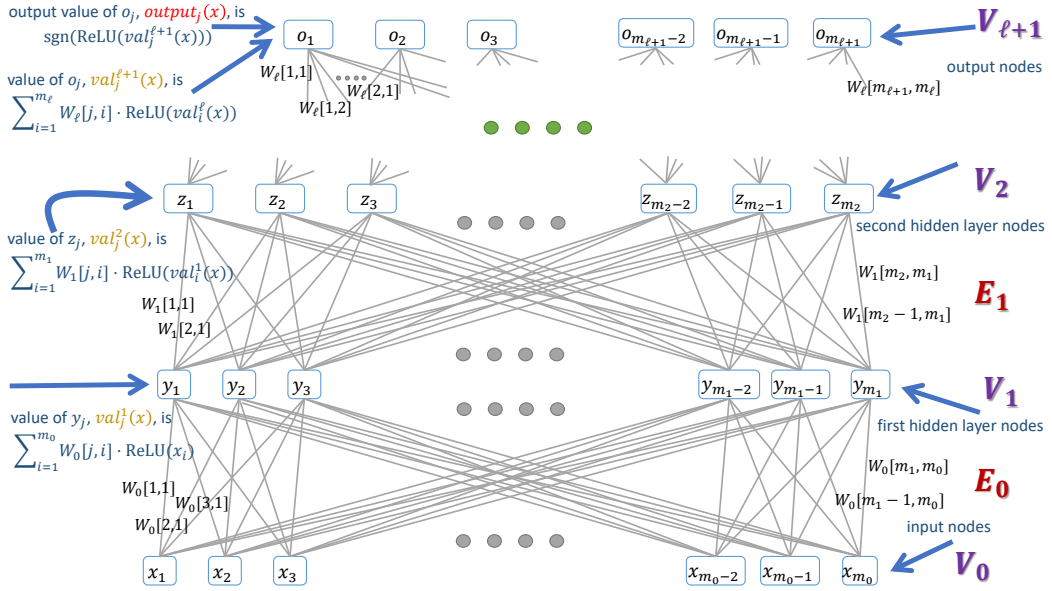

Figure 2 A general ReLU network with multiple inputs, multiple outputs, and multiple layers, extending the network depicted on Figure 1. ReLU network $(W_0, \ldots, W_\ell)$ has $\ell + 2$ layers:

- layer zero V_0 containing m_0 *input nodes*,
- ℓ *hidden layers* $1, \ldots, \ell$, with layer $1 \le k \le \ell$ containing m_k nodes V_k, and
- layer $(\ell + 1)$ containing $m_{\ell+1}$ *output nodes* $V_{\ell+1}$.

Weights: The network has *weighted edges* connecting pairs of consecutive layers. For every $0 \le k \le \ell$ and for every $1 \le i \le m_k$ and $1 \le j \le m_{k+1}$, there is an edge with weight $W_k[j,i] \in [-1,1]$ connecting the i-th node at layer k with the j-th node at layer $k+1$.
Values: For a given input $x = (x_1, \ldots, x_{m_0})^T \in \{0,1\}^{m_0}$, for any node j at layer 0, the *value* of that node $\mathrm{val}_j^0(x) = x_j$ (which is also the j-th row of $f_0(x)$). For any node j at layer $1 \le k \le \ell + 1$, the *value* of that node is $\mathrm{val}_j^k(x) = \sum_{i=1}^{m_{k-1}} W_{k-1}[j,i] \cdot \mathrm{ReLU}(\mathrm{val}_i^{k-1}(x))$; notice that this is the j-th row of $f_k(x)$.
Output: For any node j at layer $\ell + 1$ (i.e., for the j-th output node), the binary *output* computed by that node equals $\mathrm{output}_j(x) = \mathrm{sgn}(\mathrm{ReLU}(\mathrm{val}_j^{\ell+1}(x)))$; notice that this is the j-th row of $f(x)$.

3.4 Extension to ReLU networks with *multiple hidden layers/outputs*

For simplicity of presentation, we focused here mainly on the model for ReLU networks only with one output and one hidden layer. Our study, however, can be naturally extended to ReLU networks with multiple outputs and multiple hidden layers (see also Figure 2).

▶ **Definition 11 (ReLU networks with multiple hidden layers and multiple outputs).** *A ReLU network with n input nodes, $\ell \ge 1$ hidden layers with $m_1, \ldots, m_\ell$ hidden layer nodes each, and r output nodes is a tuple $(W_0, W_1, \ldots, W_\ell)$, where $W_i \in [-1,1]^{m_{i+1} \times m_i}$ for every $0 \le i \le \ell$, with $m_0 = n$ and $m_{\ell+1} = r$. The function $f : \{0,1\}^n \to \{0,1\}^r$ computed by a ReLU network $(W_0, \ldots, W_\ell)$ is defined recursively as $f(x) := \mathrm{sgn}(\mathrm{ReLU}(f_{\ell+1}(x)))$,*

$$f_i(x) := \begin{cases} W_{i-1} \cdot \mathrm{ReLU}(f_{i-1}(x)) & \text{if } 1 \le i \le \ell + 1, \\ x & \text{if } i = 0. \end{cases}$$

Observe that unfolding the recurrence in Definition 11 gives a non-recursive version of the value $f(x)$ computed by the network (in a form resembling Definition 1) is

$$f(x) := \mathrm{sgn}(\mathrm{ReLU}(W_\ell \cdot \ \ldots \ \cdot \mathrm{ReLU}(W_2 \cdot \mathrm{ReLU}(W_1 \cdot \mathrm{ReLU}(W_0 \cdot \mathrm{ReLU}(x)))) \ldots)).$$

One can also easily extend Definitions 2 and 4 to ReLU networks with multiple hidden layers and multiple outputs (for more details, see the full version of the paper [19]).

We can extend our framework to more complex neural networks and demonstrate that our methodology can be applied to a natural generalization of our setting allowing *multiple outputs* and *multiple hidden layers*. We show the applicability of our techniques by extending the approach for testing the constant 0- and the OR-function to testing near-constant functions in ReLU networks with multiple hidden layers and outputs: testing for $f(x) = \mathrm{sgn}(\mathrm{ReLU}(W_\ell \cdot \ \ldots \ \cdot \mathrm{ReLU}(W_0 \cdot \mathrm{ReLU}(x)) \ldots))$, where $W_i \in [-1,1]^{m_{i+1} \times m_i}$ and $x \in \{0,1\}^{m_0}$.

We begin with the analysis of ReLU networks with multiple outputs and *one hidden layer*. Before we proceed, let us introduce the notion of *near-constant functions* with multiple outputs[7], which are constant functions on every input x except $x = \mathbf{0}$. We first present a general reduction to single layer networks. The following theorem, central for our analysis, demonstrates that if a ReLU network with r outputs is (ε, δ)-far from computing near-constant function $\mathbf{b}$, then for at least an $\frac{\varepsilon}{2}$-fraction of the output bits i, the ReLU network restricted to the output node i is (ε', δ')-far from computing an appropriate near-constant function, where $\delta' \sim \delta/r$ and $\varepsilon' = \varepsilon^2/1025$.

▶ **Theorem 12.** *Let $\mathcal{N}$ be a ReLU network (A, W) with n input nodes, m hidden layer nodes, and r output nodes. Let $e^{-n/16} \leq \delta < 1$ and $16 \cdot \sqrt{\frac{\ln(2m)}{m}} \leq \varepsilon < \frac{1}{2}$. Let $\mathbf{b} = (\mathfrak{b}_1, \ldots, \mathfrak{b}_r) \in \{0,1\}^r$. If $\mathcal{N}$ is (ε, δ)-far from computing a near-constant function $\mathbf{b}$ then there are more than $\frac{1}{2}\varepsilon r$ output nodes such that for any such output node i, the ReLU network $\mathcal{N}$ restricted to the output node i is $(\frac{\varepsilon^2}{1025}, \frac{\delta - e^{-n/16}}{r})$-far from computing near-constant function $\mathfrak{b}_i$.*

One can then combine Theorems 5 and 12 to obtain the following theorem for testing near-constant functions in ReLU networks with *multiple outputs and one hidden layer*.

▶ **Theorem 13.** *Let $\mathcal{N}$ be a ReLU network (A, W) with n input nodes, m hidden layer nodes, and r output nodes. Let $\mathbf{b} \in \{0,1\}^r$, $(r+1)e^{-n/16} \leq \delta < 1$, and $c \cdot \sqrt{\ln(2m)/m} < \varepsilon < \frac{1}{2}$ for a sufficiently large constant c. There is a property tester that queries $O(\frac{\ln^2(1/\varepsilon\lambda)}{\varepsilon^{13}})$ entries from A and W and*

- *accepts with probability at least $\frac{2}{3}$, if $\mathcal{N}$ computes near-constant function $\mathbf{b}$, and*
- *rejects with probability at least $\frac{2}{3}$, if $\mathcal{N}$ is (ε, δ)-far from computing near-constant function $\mathbf{b}$.*

Observe that the *query complexity of our tester in Theorem 13 is independent of the network size*. Further, for constant ε and λ, the query complexity is $O(1)$, and similarly as in Theorem 5, one could remove the dependency for δ by introducing a small factor term in the query complexity.

Our analysis so far has studied the model of ReLU networks with a single hidden layer. We can extend our analysis of this setting to allow *multiple hidden layers*. A simplified claim is as follows.

[7] We need this notion to exclude trivial cases; notice that since for any ReLU network $f(\mathbf{0}) = \mathbf{0}$ (and so there is no ReLU network with $\forall_x f(x) = 1$), we consider only *near-constant functions*.

▶ **Theorem 14.** *Let* $0 < \lambda < \frac{1}{\ell+1}$, $\frac{1}{n} < \varepsilon < \frac{1}{2}$, *and* $\delta \geq e^{-n/16} + e^{-(\varepsilon/2)^{\ell} n/(342(\ell+1)^2)}$. *Let* $\mathcal{N}$ *be a ReLU network* $(W_0, \ldots, W_\ell)$ *with* n *input nodes,* $\ell \geq 1$ *hidden layers with* n *hidden layer nodes each, and a single output node. Further, suppose that* $n \geq 171 \cdot (\ell+1)^2 \cdot (2/\varepsilon)^{2\ell} \cdot (\ln(2/\delta) + \ell \ln n)$. *Assuming that* ℓ *is constant, there is a tester that queries* $\Theta(\varepsilon^{-8\ell} \cdot \ln^4(1/\lambda\varepsilon))$ *entries from* $W_0, \ldots, W_\ell$, *and*

- **(i)** *rejects with probability at least* $1 - \lambda$, *if* $\mathcal{N}$ *is* (ε, δ)*-far from computing the constant* 0*-function,*
- **(ii)** *accepts with probability at least* $1 - \lambda$, *if* $\mathcal{N}$ *computes the constant* 0*-function.*

The approach used in the proof of Theorem 14 follows the method presented in the analysis of a single hidden layer, but the need of dealing with multiple layers makes the analysis significantly more complex. The approach relies on two steps of the analysis, first proving part (i) about ReLU networks that are (ε, δ)-far from computing the constant 0-function and then part (ii) about ReLU networks that compute the constant 0-function.

The analysis of part (i) is in four steps. We first show that the use of randomly sampled nodes in each layer of the network results in a modified ReLU network that, after scaling, for any fixed input returns the value close to the value returned by the original network. Next, we extend this claim to show that there is always a modified ReLU network that, after scaling, for all but a small fraction of the inputs, returns the value close to the value returned by the original network. We then study ReLU networks $(W_0, \ldots, W_\ell)$ that are (ε, δ)-far from computing the constant 0-function. In that case, we first show that there is always an input $x_0 \in \{0,1\}^{m_0}$ on which the (ε, δ)-far network returns a value which is (sufficiently) large. Then, using the result that the sampled network approximates the output of the original network, we will argue that if $(W_0, \ldots, W_\ell)$ is (ε, δ)-far from computing the constant 0-function then randomly sampled network on input x_0 returns a (sufficiently) large.

The study of part (ii) relies on a similar approach as that in part (i). We use the fact that the sampled network computes a function which on a fixed input is (after scaling) close to the original function on that input, and hence that value cannot be too large. An important difference with part (i) is that one must prove the claim to *hold for all inputs*, making the arguments more complex (the straightforward arguments would require the size of the sample to depend on m_0).

A consequence of Theorem 14 is that it gives a tester with query complexity $1/\mathrm{poly}(\varepsilon)$ for any constant number of layers. Furthermore, as in our study for one hidden layer, essentially identical analysis as for the constant 0-function can be applied to test the OR-function.

Finally, we can extend the analysis above (Theorems 13 and 14) to networks with *multiple hidden layers* and *multiple outputs*. As in Theorem 13 (for ReLU networks with a single hidden layer and multiple outputs) the proof of the following theorem relies on a reduction of the problem to testing a near-constant function to testing the 0- and the OR-function.

▶ **Theorem 15** (Testing a near-constant function)**.** *Let* $\mathcal{N}$ *be a ReLU network* $(W_0, \ldots, W_\ell)$ *with* m_0 *input nodes,* $\ell \geq 1$ *hidden layers with* $m_1, \ldots, m_\ell$ *hidden layer nodes each, and* $m_{\ell+1} \geq 2$ *output nodes. Let* $0 < \lambda < \frac{1}{\ell+1}$, $2\sqrt[\ell]{\frac{17(\ell+1)}{\max_{0 \leq k \leq \ell}\{m_k\}}} < \varepsilon < \frac{1}{2}$, *and* $\delta \geq 2m_{\ell+1}(\xi + e^{-m_0/16})$. *Let* $\mathbf{b} \in \{0,1\}^{m_{\ell+1}}$ *and* $c = c(\ell)$ *be a sufficiently large positive constant. Suppose that for an arbitrary parameter* $0 < \xi < \frac{1}{\ell+1}$ *it holds that* $m_k \geq c \cdot (2/\varepsilon)^{2\ell^2} \cdot \ln(\frac{1}{\xi} \prod_{i=1}^{\ell} m_i)$ *for every* $0 \leq k \leq \ell$. *If* ℓ *is constant then there is a tester that queries* $\Theta(\varepsilon^{-(8\ell^2+1)} \cdot \ln^4(1/\lambda\varepsilon))$ *entries from* $W_0, \ldots, W_\ell$, *and*

- *accepts with probability at least* $1 - \lambda$, *if* $\mathcal{N}$ *computes near-constant function* $\mathbf{b}$, *and*
- *rejects with probability at least* $1 - \lambda$, *if* $\mathcal{N}$ *is* (ε, δ)*-far from computing near-constant function* $\mathbf{b}$.

As with our earlier results in Section 3.4, the *query complexity of our tester in Theorem 15 is independent of the network size*, and it is $O(1)$ for constant ε, λ, ℓ, and $m_{\ell+1}$.

4 Testing constant 0-function in ReLU networks with one hidden layer

To give readers a flavor of our analysis, we present here our analysis of ReLU networks with one hidden layer and a single output – Theorem 5 (1), deferring the study of other claims and of more complex neural networks with multiple outputs and multiple hidden layers to the full version of our paper [19]. We consider the simplest function, the **constant 0-function** $f_n^{(0)} : \{0,1\}^n \to \{0,1\}$ defined as $f_n^{(0)}(x) = 0$ for all $x \in \{0,1\}^n$, and study it from a property testing point of view. We define $P_n = \{f_n^{(0)}\}$ and the property $\Pi^{(0)} = \bigcup_{n\in\mathbb{N}} P_n$. Our objective is to design a property tester for $\Pi^{(0)}$.

While one can show (see [19]) that it is hard to decide if a given ReLU network computes the constant 0-function or not, in this section we prove that a *relaxed* version of the problem, namely, that of *(property) testing* whether a given ReLU network computes the constant 0-function, has significantly lower complexity. Below we prove the first part of Theorem 5 and show that property $\Pi^{(0)}$ is testable for ReLU networks with one hidden layer.

Testing 0-function. Our property testing algorithm AllZeroTester$(\varepsilon, \lambda, n, m)$ below works by focusing its attention on a small "subnetwork" induced by randomly sampled nodes. The algorithm samples a subset of $\text{poly}(1/\varepsilon) \cdot \ln(1/\lambda)$ nodes from the input layer and from the hidden layer. Then, we scale the weights of all edges between the sampled input and the sampled hidden layer nodes by a factor of $\frac{n}{s}$ and all edges from sampled hidden layer nodes to the output node by $\frac{m}{t}$. We then introduce a *bias* $b = -\frac{1}{16}\varepsilon nm$ to the output node, i.e., we subtract $\frac{1}{16}\varepsilon nm$ before pplying the ReLU function. For the resulting network we check if it computes the constant 0-function. If this is the case, we accept; otherwise, we reject.

Algorithm 1 AllZeroTester$(\varepsilon, \lambda, n, m)$.

1 Sample $s = \lceil \frac{2^{20}}{\varepsilon^2} \cdot \ln(1/\varepsilon\lambda) \rceil$ nodes from the input layer and $t = \lceil \frac{2^{30}}{\varepsilon^4} \cdot \ln(1/\varepsilon\lambda) \rceil$ nodes from the hidden layer uniformly at random without replacement.
2 Let $S \in \mathbb{N}_0^{n\times n}$ and $T \in \mathbb{N}_0^{m\times m}$ be the corresponding sampling matrices.
3 **if** there is $x \in \{0,1\}^n$ with $\frac{nm}{st} \cdot w^T \cdot \text{ReLU}(TASx) - \frac{1}{16}\varepsilon nm > 0$ **then reject**;
4 **else accept**.

In the following, we use matrix notation and write S and T for sampling matrices that select s and t input/hidden layer nodes at random. That is, S is an $n \times n$ diagonal matrix whose diagonal entries (i,i) are equal to 1 iff x_i is in the sample set (and is 0 otherwise), and T is an $m \times m$ diagonal matrix whose diagonal entries (j,j) are equal to 1 iff hidden layer node j is in the second sample set (and is 0 otherwise). With this notation we can describe our algorithm as follows (we did not optimize constants and assume $\varepsilon \in (\frac{1}{m}, \frac{1}{2}), \lambda \in (0, \frac{1}{2})$).

Let us first observe that the query complexity of AllZeroTester is $(s+1)\cdot t = \Theta(st)$, and hence the query complexity is as stated. Indeed, after sampling s input nodes and t nodes from the hidden layer, the diagonal matrices S and T have s and t non-zero entries, respectively. Therefore TAS can be computed by querying only $s \cdot t$ entries from matrix A and it will have at most t non-zero rows and at most s non-zero columns. Hence, in order to compute $TASx$ one has to consider only s bits from $\{0,1\}^n$, and to compute $\frac{nm}{st} \cdot w^T \cdot \text{ReLU}(TASx)$ one has to consider only t entries from w^T.

Properties of the output. Next, we will study the properties of the output. First, notice that our sampling network without a bias would compute

$$\text{sgn}\left(\text{ReLU}\left(\frac{m}{t} \cdot w^T \cdot T \cdot \text{ReLU}\left(\frac{n}{s} \cdot ASx\right)\right)\right).$$

Since the ReLU function is positive homogeneous, T is a fixed matrix with non-negative entries after the sampling, $\frac{n}{s} > 0$, and $\frac{m}{t} > 0$, we have that

$$\operatorname{sgn}\left(\operatorname{ReLU}\left(\frac{m}{t}\cdot w^T\cdot T\cdot \operatorname{ReLU}\left(\frac{n}{s}\cdot ASx\right)\right)\right) = \operatorname{sgn}\left(\operatorname{ReLU}\left(\frac{nm}{st}\cdot w^T\cdot \operatorname{ReLU}(TASx)\right)\right).$$

Introducing in AllZeroTester the bias to the output node leads to a network computing

$$\operatorname{sgn}\left(\operatorname{ReLU}\left(\frac{nm}{st}\cdot w^T\cdot \operatorname{ReLU}(TASx) - \frac{1}{16}\varepsilon nm\right)\right)$$

and we notice that one can drop sgn and the outer ReLU, if we check whether the computed value is greater than 0.

4.1 One hidden layer: If (ε, δ)-far then there is a bad input

To analyze the correctness of AllZeroTester, we first present our structural lemma.

▶ **Lemma 16.** *Let $\delta \geq e^{-n/16}$ and $\frac{1}{m} < \varepsilon < \frac{1}{2}$. Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes. If (A, w) is (ε, δ)-far from computing the constant 0-function then there exists an input $x \in \{0,1\}^n$ such that $w^T \cdot \operatorname{ReLU}(Ax) > \frac{1}{8}\varepsilon mn$.*

Proof. The proof of Lemma 16 is by contradiction: we will assume that for every input x the output value is at most $\frac{1}{8}\varepsilon nm$ and then we will show that the ReLU network is (ε, δ)-close to computing the constant 0-function; this would imply contradiction.

First, suppose that there are less than $\lfloor \varepsilon m \rfloor$ positively weighted edges connecting the hidden layer to the output node. Then we set all of them to 0 and the resulting network computes the constant 0-function. Thus the network is (ε, δ)-close, which is a contradiction.

Otherwise, suppose that there are at least $\lfloor \varepsilon m \rfloor$ positively weighted edges connecting the hidden layer to the output node. Select an arbitrary subset E of $\lfloor \varepsilon m \rfloor$ positively weighted edges and let U denote the set of hidden layer nodes incident to the edges from E. Let E' be the subset of first layer edges connecting input nodes to the nodes from U. Notice that $|E| = |U| = \lfloor \varepsilon m \rfloor$ and $|E'| = n \cdot |U| = \lfloor \varepsilon m \rfloor n$. Now we modify the weights of $|E| = \lfloor \varepsilon m \rfloor$ edges between the hidden layer nodes and the output, and $|E'| = \lfloor \varepsilon m \rfloor n$ edges between input nodes and hidden layer nodes: we assign weight -1 to every edge from E and weight 1 to every edge from E'.

Observe that originally, before the modifications, every hidden layer node in U had a non-negative contribution to the output. After the modifications, every hidden layer node from U contributes $-k$ to the output node, where k is the number of 1s in the input vector x, i.e., $k = \|x\|_1$. Thus all nodes from U contribute $-|U| \cdot \|x\|_1 = -\lfloor \varepsilon m \rfloor \cdot \|x\|_1$, and the changes of the weights of the edges incident to U *reduce the output value* by at least $\lfloor \varepsilon m \rfloor \cdot \|x\|_1$.

Next, let us consider a random input vector $x \in \{0,1\}^n$. Using Chernoff bounds, with probability at least $1 - e^{-n/16} \geq 1 - \delta$, we have $\|x\|_1 > \frac{1}{4}n$. Conditioned on this event, after modifying the weights in the network, the sum of inputs to the output node is *reduced* by at least $\frac{1}{4}n \cdot \lfloor \varepsilon m \rfloor \geq \frac{1}{8}\varepsilon nm$ for our range of parameters. However, we have assumed that for every input x the output value is at most $\frac{1}{8}\varepsilon nm$, and thus the network computes the constant 0-function. Hence, we modified an ε-fraction of the network's edge weights and obtained a function that disagrees with the constant 0-function only on inputs with $\|x\|_0 \leq n/4$, which is at most a δ-fraction of all inputs. This is a contradiction to the assumption that the network is (ε, δ)-far from that function. ◀

4.2 One hidden layer: Random sampling concentration

Next, we consider the impact of the random sampling matrices S and T on the output value of a ReLU network for any fixed x (and hence, for x whose existence is argued in Lemma 16).

▶ **Lemma 17.** *Let $w \in [-1,1]^m$, $A \in [-1,1]^{m\times n}$. Let S, T be sampling matrices corresponding to samples of size $t \geq \frac{2048\ln(4/\lambda)}{\varepsilon^2}$ and $s \geq \frac{2048\ln(4t/\lambda)}{\varepsilon^2}$. Then for any fixed $x \in \{0,1\}^n$*

$$\Pr\left[\left|\frac{mn}{st}\cdot w^T\cdot \mathrm{ReLU}(TASx) - w^T\cdot\mathrm{ReLU}(Ax)\right| > \tfrac{1}{16}\varepsilon nm\right] \leq \lambda\ .$$

Proof. Fix an arbitrary $x \in \{0,1\}^n$. We first show that

$$\Pr\left[\left|\frac{m}{t}\cdot w^T\cdot T\cdot \mathrm{ReLU}(Ax) - w^T\cdot\mathrm{ReLU}(Ax)\right| > \tfrac{1}{32}\varepsilon nm\right] \leq \tfrac{1}{2}\lambda\ . \tag{1}$$

Observe that $w^T\cdot T\cdot\mathrm{ReLU}(Ax)$ is the sum of t entries in $w\odot\mathrm{ReLU}(Ax)$ (here $\odot$ denotes the Hadamard product) sampled uniformly without replacement. Next, we notice that $\mathbf{E}[w^T\cdot T\cdot\mathrm{ReLU}(Ax)] = \frac{t}{m}\cdot w^T\cdot\mathrm{ReLU}(Ax)$. Rescaling by $\frac{1}{n}$ ensures that the entries in $w\odot\mathrm{ReLU}(Ax)$ are in $[-1,1]$ and so we can apply Hoeffding bounds [31] (note that the Hoeffding bound also holds for sampling without replacement [6, Proposition 1.2]) to show

$$\Pr\left[\left|\frac{1}{n}w^T T\,\mathrm{ReLU}(Ax) - \frac{t}{mn}w^T\,\mathrm{ReLU}(Ax)\right| > \tfrac{1}{32}\varepsilon t\right] =$$
$$\Pr\left[\left|\frac{1}{n}w^T T\,\mathrm{ReLU}(Ax) - \mathbf{E}\left[\frac{1}{n}w^T\cdot T\cdot\mathrm{ReLU}(Ax)\right]\right| > \tfrac{1}{32}\varepsilon t\right] \leq 2e^{-2\varepsilon^2(t/32)^2/(4t)} \leq \tfrac{1}{2}\lambda$$

for our choice of $t \geq \frac{2048\ln(4/\lambda)}{\varepsilon^2}$. Multiplying both sides with $\frac{mn}{t}$ proves inequality (1).

We can view the sampling process as first sampling t nodes from the hidden layer and then s nodes from the input layer. Thus, we can condition on T being fixed and assume that the s nodes from the input layer are sampled uniformly at random. We will now prove:

$$\Pr\left[\left|\frac{mn}{ts}\cdot w^T\cdot\mathrm{ReLU}(TASx) - \frac{m}{t}w^T\cdot T\cdot\mathrm{ReLU}(Ax)\right| > \tfrac{1}{32}\varepsilon nm\ \middle|\ T\right] \leq \tfrac{1}{2}\lambda\ . \tag{2}$$

Let a_i be the i-th row of matrix A. The value $a_i\cdot x$ is the value at the i-th hidden layer node before applying ReLU. We may view a_iSx as the sum of s random entries from $a_i\odot x$. Since these entries are from $[-1,1]$ and $\mathbf{E}[a_iSx] = \frac{s}{n}\cdot a_i\cdot x$, Hoeffding's bound implies that

$$\Pr\left[\left|a_iSx - \frac{s}{n}\cdot a_ix\right| > \varepsilon s/32\right] \leq 2e^{-2\varepsilon^2(s/32)^2/(4s)} \leq \frac{\lambda}{2t},$$

where the last inequality follows from our assumption that $s \geq \frac{2024\ln(4t/\lambda)}{\varepsilon^2}$.

Now we observe that for $w_i \in [-1,1]$ we have

$$\left|w_i\cdot\left(\mathrm{ReLU}(a_iSx) - \mathrm{ReLU}\left(\frac{s}{n}a_ix\right)\right)\right| \leq \left|w_i\cdot\left(a_iSx - \frac{s}{n}a_ix\right)\right| \leq \left|a_iSx - \frac{s}{n}a_ix\right|.$$

Following our arguments above, for each of the rows we can bound the contribution of the edge with weight w_i that connects the i-th hidden layer node to the output node to get

$$\Pr\left[\left|w_i\cdot\left(\mathrm{ReLU}(a_iSx) - \mathrm{ReLU}\left(\frac{s}{n}\cdot a_ix\right)\right)\right| > \frac{1}{32}\varepsilon s\right] \leq 2e^{-2\varepsilon^2/(s/32)^2/(4s)} \leq \frac{\lambda}{2t}.$$

Next, we observe that TA equals A in exactly t rows (the sampled rows) and is 0 otherwise. Taking a union bound over the contribution of these t rows and observing that for the remaining rows the product with Sx is 0 for every S, we obtain

$$\Pr\left[\left|w^T\cdot\mathrm{ReLU}(TASx) - \frac{s}{n}\cdot w^T\cdot\mathrm{ReLU}(TAx)\right| > \frac{1}{32}\varepsilon st\ \middle|\ T\right] \leq \frac{\lambda}{2}.$$

Multiplying with $\frac{mn}{st}$ and observing that $\mathrm{ReLU}(TAx) = T \cdot \mathrm{ReLU}(Ax)$ (since ReLU is positively homogeneous), we obtain (2):

$$\Pr\left[\left|\frac{mn}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx) - \frac{m}{t} \cdot w^T \cdot T \cdot \mathrm{ReLU}(Ax)\right| > \varepsilon mn/32 \;\middle|\; T\right] \leq \frac{\lambda}{2}.$$

Next observe that inequalities (1) and (2) are simultaneously satisfied with probability at least $1 - \lambda$. Thus, the lemma follows from

$$\begin{aligned}
&\left|\frac{mn}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx) - w^T \cdot \mathrm{ReLU}(Ax)\right| \leq \\
&\leq \left|\frac{mn}{st} w^T \,\mathrm{ReLU}(TASx) - \frac{m}{t} w^T T \,\mathrm{ReLU}(Ax)\right| + \left|\frac{m}{t} w^T T \,\mathrm{ReLU}(Ax) - w^T \,\mathrm{ReLU}(Ax)\right| \\
&\leq \frac{1}{16}\varepsilon mn
\end{aligned}$$

with probability at least $1 - \lambda$. ◀

4.3 One hidden layer: If (ε, δ)-far then we reject

Our next lemma incorporates Lemmas 16 and 17 to summarize the analysis of our testers for ReLU networks that are (ε, δ)-far from computing the constant 0-function.

▶ **Lemma 18.** *Let $\delta \geq e^{-n/16}$, $\frac{1}{m} < \varepsilon < \frac{1}{2}$, and $0 < \lambda < \frac{1}{2}$. Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes. If (A, w) is (ε, δ)-far from computing the constant 0-function then algorithm* AllZeroTester *rejects with probability at least $1 - \lambda$.*

Proof. By Lemma 16, since (A, w) is (ε, δ)-far from computing the constant 0-function, there exists an $x_0 \in \{0,1\}^n$ such that $w^T \cdot \mathrm{ReLU}(Ax_0) > \frac{1}{8}\varepsilon mn$. Next, by Lemma 17, with probability at least $1 - \lambda$ (over the random choice of S and T in the algorithm) we have

$$\left|\frac{mn}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx_0) - w^T \cdot \mathrm{ReLU}(Ax_0)\right| \leq \frac{1}{16}\varepsilon nm \ .$$

This implies that with probability $1 - \lambda$ we obtain

$$\frac{mn}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx_0) - \frac{1}{16}\varepsilon mn > 0,$$

and so AllZeroTester rejects with probability at least $1 - \lambda$. This concludes the proof. ◀

4.4 One hidden layer: If network computes 0 then we accept

Our final lemma of this section summarizes the second part of the analysis of AllZeroTester for ReLU networks and deals with the networks that correctly compute the 0-function.

▶ **Lemma 19.** *Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes. Let S, T be sampling matrices corresponding to samples of size $t \geq \frac{512 \ln(2^{s+1}/\lambda)}{\varepsilon^2}$ and $s \geq 1$. If (A, w) computes the constant 0-function then with probability at least $1 - \lambda$ (over the random choice of S and T as defined by the algorithm), for all $x \in \{0,1\}^n$*

$$\frac{nm}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx) - \frac{1}{16}\varepsilon nm < 0,$$

and so algorithm AllZeroTester *accepts.*

Proof. Let (A, w) be a ReLU network that computes the constant 0-function. Let I_S be the set of indices of the s nodes sampled by our algorithm from the input layer and let $X_S = \{x = (x_1, \ldots, x_n)^T \in \{0,1\}^n : x_i = 0, \forall i \notin S\}$ be the set of different inputs to the sampled network. In what follows, we will condition on the choice of S being fixed. Then

$$\frac{nm}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx) - \frac{1}{16}\varepsilon nm < 0$$

holds for all $x \in \{0,1\}^n$ if and only if this inequality holds for all $x \in X_S$. Therefore, in what follows we will show that the output value of a fixed $x \in X_S$ will be approximated with high probability and then we will apply a union bound over all $x \in X_S$ to conclude the analysis.

Let x be an arbitrary fixed vector from X_S. Observe that $\frac{1}{s} \cdot w^T \cdot \mathrm{ReLU}(ASx)$ is an m-dimensional vector with entries from $[-1, 1]$. We know that $w^T \mathrm{ReLU}(ASx) \leq 0$ since the network computes the constant 0-function. Since ReLU is positively homogeneous we can move T out of the ReLU function. The fact that T is a sampling matrix and that $w^T \mathrm{ReLU}(ASx) \leq 0$ yields

$$\mathbf{E}\left[\frac{1}{s} \cdot w^T \cdot T \cdot \mathrm{ReLU}(ASx) \;\middle|\; S\right] \leq 0. \tag{3}$$

Then, applying Hoeffding bound gives us

$$\Pr\left[\left|\frac{1}{s} \cdot w^T \cdot T \cdot \mathrm{ReLU}(ASx) - \mathbf{E}\left[\frac{1}{s} \cdot w^T \cdot T \cdot \mathrm{ReLU}(ASx)\right]\right| \geq \frac{1}{16}\varepsilon t \;\middle|\; S\right] \leq 2e^{\frac{-2(\varepsilon t/16)^2}{4t}},$$

which is upper bounded by $\frac{\lambda}{2^s}$ for our choice of t, since $t \geq \frac{512 \ln(2^{s+1}/\lambda)}{\varepsilon^2}$. By the union bound over the $|X_S| = 2^s$ vectors in X_S, this implies the following inequality for all $x \in \{0,1\}^n$:

$$\Pr\left[\left|\frac{mn}{st} \cdot w^T \cdot T \cdot \mathrm{ReLU}(ASx) - \mathbf{E}\left[\frac{nm}{st} \cdot w^T \cdot T \cdot \mathrm{ReLU}(ASx)\right]\right| \geq \frac{1}{16}\varepsilon nm\right] \leq \lambda.$$

Lemma 19 follows from inequality (3) and the fact that in the algorithm we are subtracting $\frac{1}{16}\varepsilon nm$ from the result by taking a union bound over the $|X_S| = 2^s$ vectors in X_S. ◀

4.5 One hidden layer: Completing the proof of Theorem 5 (1)

Now Theorem 5 (1) follows immediately from Lemmas 18 and 19 and from our analysis of the query complexity after the description of AllZeroTester on page 16. ◀

5 Conclusions

In this paper, we introduce a property testing model to study computational networks used as computational devices and illustrate our setting on a case study of simple ReLU neural networks. We believe that the property testing model developed in this paper establishes a novel and useful theoretical framework for the study of testing algorithms and offers a new and interesting viewpoint with a potential of advancing our understanding of the structure of neural networks (and, in general, of computational networks). While, due to space constraints, we have omitted full technical details behind the claims made in Section 3, we refer the reader for the proofs and a comprehensive analysis to the full version of this paper [19].

In the context of neural networks, given our poor understanding of the computational processes in modern deep networks (leading to serious questions on the trust and transparency of the underlying learning algorithms), we hope that the lens of property testing as developed in the framework considered in this paper will contribute to a better understanding of the relation of the network structure and the computed function or properties of that function.

Indeed, since the analysis of property testing algorithms relies on establishing close links between the object's local structure and the tested property, we believe the study of simple yet fundamental models of ReLU networks may provide a new insight into such networks.

Although our paper makes the first step in the analysis of ReLU networks in the context of property testing, there is still a large number of open questions (from the property testing point of view) regarding such networks, including, among others, the following:

- What is the query complexity of testing dictatorship and juntas (see [10] for a survey on testing juntas)?
- Classify the constant time testable properties with one- and two-sided error.
- Study tolerant testing [41] in our model.
- Extend the results to inputs over the reals.

These are just a small number of immediate questions one can ask about our model. We believe that there is a wide variety of interesting questions and are confident to see further research in this direction.

References

1 Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: It's all about regularity. *SIAM Journal on Computing*, 39(1):143–167, 2009. doi:10.1137/060667177.

2 Noga Alon and Asaf Shapira. Testing satisfiability. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 645–654, 2002. URL: https://dl.acm.org/doi/10.5555/545381.545467.

3 Noga Alon and Asaf Shapira. A characterization of the (natural) graph properties testable with one-sided error. *SIAM Journal on Computing*, 37(6):1703–1727, 2008. doi:10.1137/06064888X.

4 Sunil Arya and David M. Mount. A fast and simple algorithm for computing approximate Euclidean minimum spanning trees. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1220–1233, 2016. doi:10.1137/1.9781611974331.CH85.

5 Maria-Florina Balcan, Yi Li, David P. Woodruff, and Hongyang Zhang. Testing matrix rank, optimally. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 727–746, 2019. doi:10.1137/1.9781611975482.46.

6 Rémi Bardenet and Odalric-Ambrym Maillard. Concentration inequalities for sampling without replacement. *Bernoulli*, 21(3):1361–1385, 2015. doi:10.3150/14-BEJ605.

7 Tugkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing closeness of discrete distributions. *Journal of the ACM*, 60(1):4:1–4:25, 2013. doi:10.1145/2432622.2432626.

8 Eric Baum and David Haussler. What size net gives valid generalization? *Advances in Neural Information Processing Systems (NIPS)*, 1:81–90, 1988. URL: https://proceedings.neurips.cc/paper/1988/hash/1d7f7abc18fcb43975065399b0d1e48e-Abstract.html.

9 Arnab Bhattacharyya and Yuichi Yoshida. *Property Testing: Problems and Techniques*. Springer Verlag, Berlin, 2022. doi:10.1007/978-981-16-8622-1.

10 Eric Blais. Testing juntas: A brief survey. In Oded Goldreich, editor, *Property Testing: Current Research and Surveys*, pages 32–40. Springer Verlag, Berlin, 2010. doi:10.1007/978-3-642-16367-8_4.

11 Avrim Blum and Ronald Rivest. Training a 3-node neural network is NP-complete. *Advances in Neural Information Processing Systems (NIPS)*, 1:494–501, 1988. URL: https://proceedings.neurips.cc/paper_files/paper/1988/hash/3def184ad8f4755ff269862ea77393dd-Abstract.html.

12 Digvijay Boob, Santanu S. Dey, and Guanghui Lan. Complexity of training ReLU neural network. *Discrete Optimization*, 44:100620, 2022. doi:10.1016/j.disopt.2020.100620.

13 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005. doi:10.1137/S0097539702403244.

14 Sitan Chen, Zehao Dou, Surbhi Goel, Adam R. Klivans, and Raghu Meka. Learning narrow one-hidden-layer ReLU networks. In *Proceedings of the 36th Annual Conference on Learning Theory (COLT)*, pages 5580–5614, 2023. URL: https://proceedings.mlr.press/v195/chen23a.html.

15 Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the Euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1):91–109, 2005. doi:10.1137/S0097539703435297.

16 Artur Czumaj and Christian Sohler. Sublinear-time algorithms. *Bulletin of EATCS*, 89:23–47, 2006.

17 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009. doi:10.1137/060672121.

18 Artur Czumaj and Christian Sohler. Sublinear-time algorithms. In Oded Goldreich, editor, *Property Testing — Current Research and Surveys*, volume 6390 of *Lecture Notes in Computer Science*, pages 41–64. Springer, 2010. doi:10.1007/978-3-642-16367-8_5.

19 Artur Czumaj and Christian Sohler. Property testing of computational networks. *CoRR arXiv*, abs/2512.07577, 2025. Full version of the current paper. doi:10.48550/arXiv.2512.07577.

20 Hennie Daniels and Marina Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, June 2010. doi:10.1109/TNN.2010.2044803.

21 Santanu S Dey, Guanyi Wang, and Yao Xie. Approximation algorithms for training one-node ReLU neural networks. *IEEE Transactions on Signal Processing*, 68:6696–6706, 2020. doi:10.1109/TSP.2020.3039360.

22 Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A. Servedio, and Andrew Wan. Testing for concise representations. In *Proceedings of the 48th Symposium on Foundations of Computer Science (FOCS)*, pages 549–558, 2007. doi:10.1109/FOCS.2007.32.

23 Gábor Elek. Finite graphs and amenability. *Journal of Functional Analysis*, 263(9):2593–2614, 2012. doi:10.1016/j.jfa.2012.08.021.

24 Eldar Fischer, Guy Kindler, Dana Ron, Shmuel Safra, and Alex Samorodnitsky. Testing juntas. *Journal of Computer and System Sciences*, 68(4):753–787, 2004. doi:10.1016/J.JCSS.2003.11.004.

25 Eldar Fischer, Ilan Newman, and Jirí Sgall. Functions that have read-twice constant width branching programs are not necessarily testable. *Random Structures and Algorithms*, 24(2):175–193, 2004. doi:10.1002/RSA.10110.

26 Vincent Froese, Christoph Hertrich, and Rolf Niedermeier. The computational complexity of ReLU network training parameterized by data dimensionality. *Journal of Artificial Intelligence Research*, 74:1775–1790, 2022. doi:10.1613/jair.1.13547.

27 Surbhi Goel, Adam R. Klivans, Pasin Manurangsi, and Daniel Reichman. Tight hardness results for training depth-2 ReLU networks. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 22:1–22:14, 2021. doi:10.4230/LIPIcs.ITCS.2021.22.

28 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

29 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

30 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/S00453-001-0078-7.

31 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi:10.2307/2282952.

32 Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi:10.1016/0893-6080(89)90020-8.

33 Stephen Judd. On the complexity of loading shallow neural networks. *Journal of Complexity*, 4(3):177–192, 1988. doi:10.1016/0885-064X(88)90019-2.

34 Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi:10.1038/nature14539.

35 Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Proceedings of the 30th Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 6231–6239, 2017. URL: https://papers.nips.cc/paper_files/paper/2017/hash/32cbf687880eb1674a07bf717761dd3a-Abstract.html.

36 Pasin Manurangsi and Daniel Reichman. The computational complexity of training ReLU(s). *CoRR arXiv*, abs/1810.04207, 2018. arXiv:1810.04207.

37 Dan Mikulincer and Daniel Reichman. Size and depth of monotone neural networks: Interpolation and approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 36(4):6314–6325, April 2025. doi:10.1109/TNNLS.2024.3387878.

38 Ilan Newman. Testing membership in languages that have small width branching programs. *SIAM Journal on Computing*, 31(5):1557–1570, 2002. doi:10.1137/S009753970038211X.

39 Ilan Newman and Christian Sohler. Every property of hyperfinite graphs is testable. *SIAM Journal on Computing*, 42(3):1095–1112, 2013. doi:10.1137/120890946.

40 Pekka Orponen. Neural networks and complexity theory. *Nordic Journal of Computing*, 1(1):94–110, 1994. URL: https://www.cs.helsinki.fi/njc/njc1_papers/number1/survey.pdf.

41 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006. doi:10.1016/j.jcss.2006.03.002.

42 Michal Parnas, Dana Ron, and Alex Samorodnitsky. Testing basic Boolean formulae. *SIAM Journal on Discrete Mathematics*, 16(1):20–46, 2002. doi:10.1137/S0895480101407444.

43 Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI, 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

44 Dana Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008. doi:10.1561/2200000004.

45 Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Mathematics*, 25(4):1562–1588, 2011. doi:10.1137/100791075.

46 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996. doi:10.1137/S0097539793255151.

47 Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. doi:10.5555/2627435.2670313.

48 Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR arXiv*, abs/1312.6199, 2013. arXiv:1312.6199.

49 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 30th Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

50 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer Verlag, Berlin, 1999. doi:10.1007/978-3-662-03927-4.