Testing Neural Networks

Artur Czumaj

Department of Computer Science

Centre for Discrete Mathematics and its Applications (DIMAP)

University of Warwick

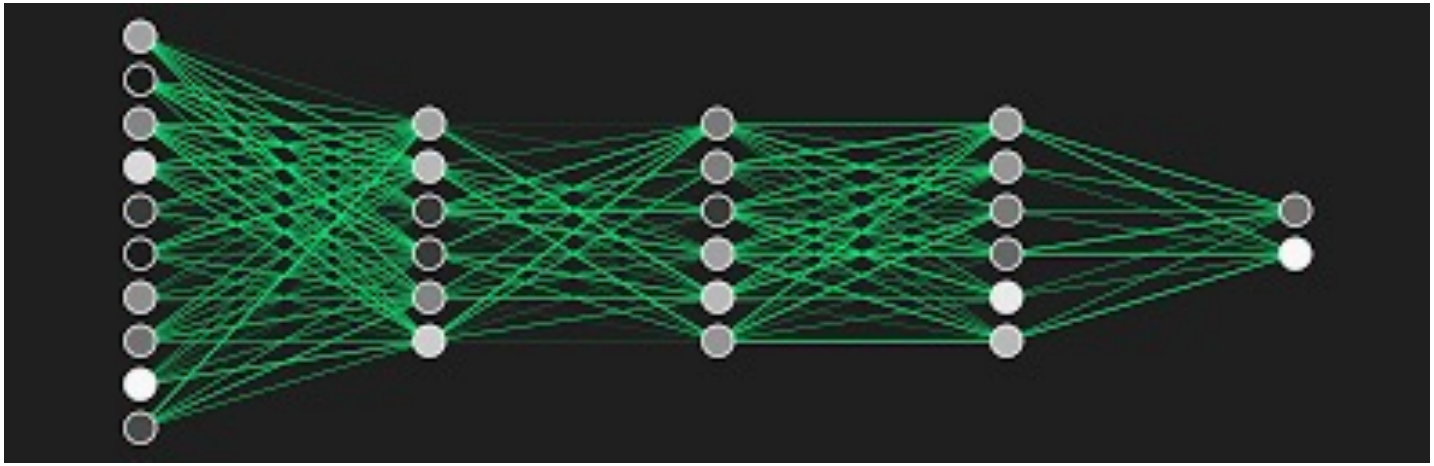
Christian Sohler

Computer Science Institute

University of Cologne

Central question

Does a given neural network compute in a robust way?



Testing a Neural Network

We will view a (trained, feedforward) neural network as a circuit:

- receives an input vector and
- computes an output

A neural network may be viewed as computing a function f of the inputs

Testing a Neural Network

Goal: formulate an approach that allows us to obtain
a mathematical understanding of the correspondence
of the (local) structure of the networks
with the function of its input it computes.

Our method of choice: framework of **property testing**

Underlying idea behind property testing:

- solve a *relaxed decision problem* by inspecting only a small part of the input
  - for example, by sampling

Property Testing

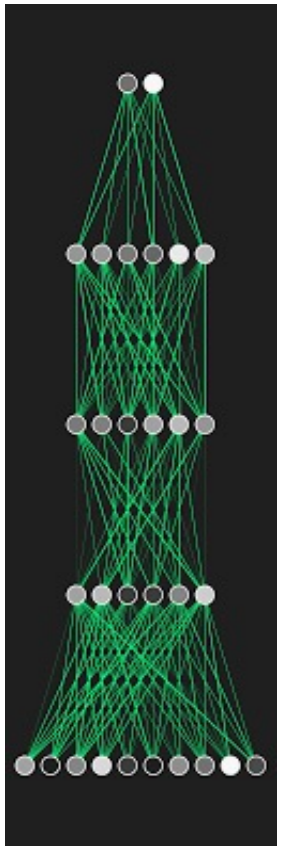
- Distinguish inputs that have specific property
  from those that are far from having the property (Rubinfeld-Sudan'96)

Benefits:

- May be a natural question to ask
- May be just as good when data constantly changing
- Gives fast sanity check to rule out very "bad" inputs (i.e., restaurant bills) or to decide when expensive processing is worth it

Testing a Neural Network

Underlying idea behind property testing: solve a relaxed decision problem by inspecting only a small part of the input (for example, by sampling).



A **property testing** algorithm is given

- query access to an object and
- the objective is to
  - accept objects that have a certain property of interest and
  - reject objects that are ***far*** from having that property.

In this talk:

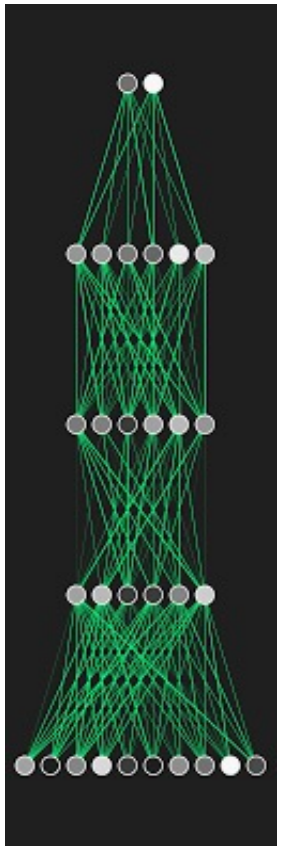
- objects = neural networks
- properties = functions or properties of functions

Testing a Neural Network

Why property testing?

- Not necessarily because neural networks are massive and we need sublinear-time algorithms
- Our view: property testing is a tool to reveal relations between the local structure of an object and its global structure
  - this in turn: can provide combinatorial insights about the structure of an object

- Our goal: improve our understanding of the relation between the network structure and the computed function

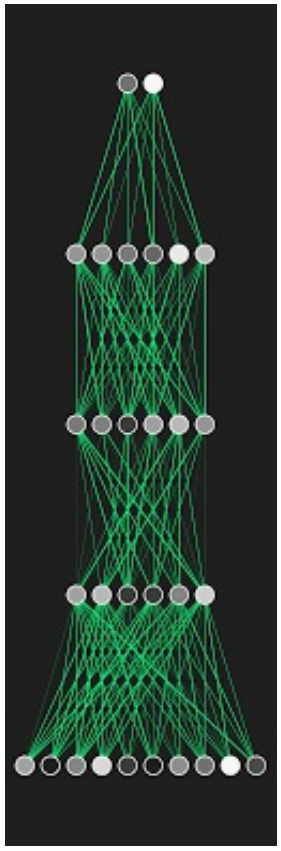


Testing a Neural Network

Why property testing?

- *Relaxed decision* setting in property testing has some appealing properties in the context of understanding neural networks:
  - typically, we'd like a neural network to be robust to small changes in the network;
  - to cope with risks of adversarial attacks
  - to accommodate network training of *dropout* used to improve generalization

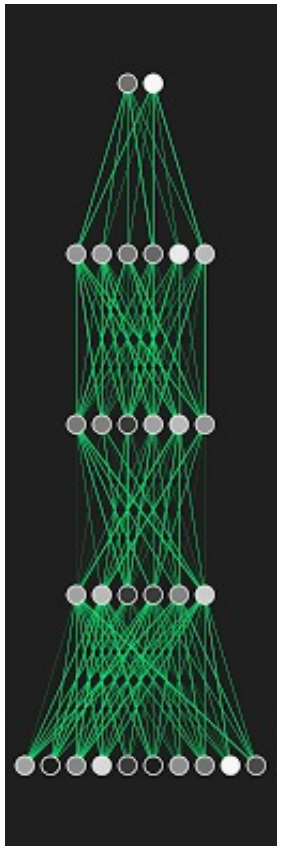
    e.g., a random fraction of input nodes is dropped during training for each training example



Testing a Neural Network

We introduce a property testing model for a fully connected feedforward network with
ReLU activation function (without bias) and hidden layers

- We will view the network as a circuit that computes a Boolean function
- Our objective: design property testing algorithms that
  - sample weights of the network and
  - accept, if the network computes a certain function and
  - reject, if the network is (ε, δ)-far from computing the function



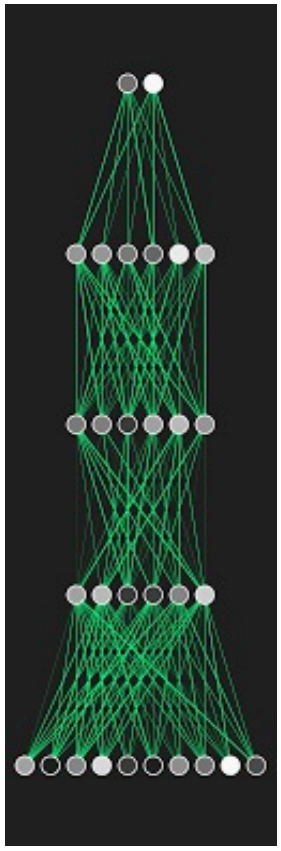
Network is (ε, δ)-far from computing the function f if

- one has to change the network in more than an ε-fraction of its size
- to obtain a function that differs from the target function on $< \delta$-fraction of inputs

Testing a Neural Network

We introduce a property testing model for a fully connected feedforward network with
ReLU activation function (without bias) and one hidden layer

- We will view the network as a circuit that computes a Boolean function
- Our objective: design property testing algorithms that
  - sample weights of the network and
  - accept, if the network computes a certain function and
  - reject, if the network is (ε, δ)-far from computing the function



We will not consider a bias in the activation function

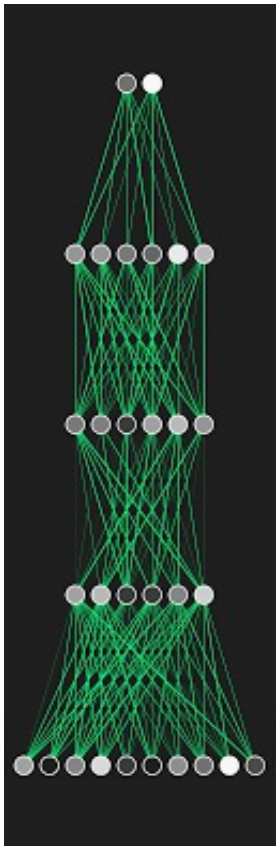
- we believe the main challenge is to understand the non-linearity of the ReLU function;
- one can easily simulate a bias in a network by allowing some inputs that are always 1 (and in practice, a network that is trained without bias will typically learn the bias)

Testing a Neural Network

We introduce a property testing model for a fully connected feedforward network with
ReLU activation function (without bias) and one hidden layer

- We will view the network as a circuit that computes a Boolean function
- Our objective: design property testing algorithms that
  - sample weights of the network and
  - accept, if the network computes a certain function and
  - reject, if the network is (ε, δ)-far from computing the function

We will assume most of the time that the number of input neurons and neurons at the hidden layer will be polynomially related (in practice, they are typically within a constant factor of each other)



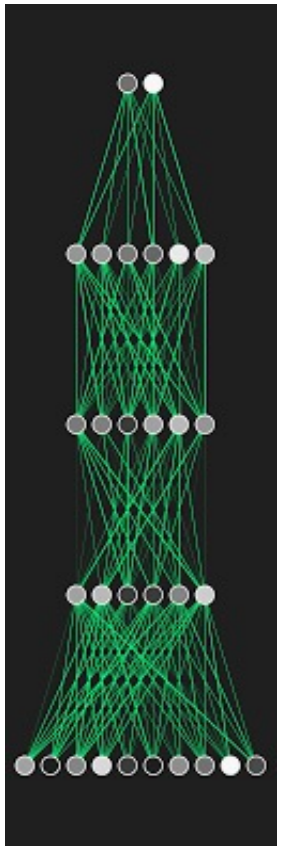
Testing a Neural Network

Main conceptual contribution:

- a model that allows a nontrivial study of neural networks from the lens of property testing

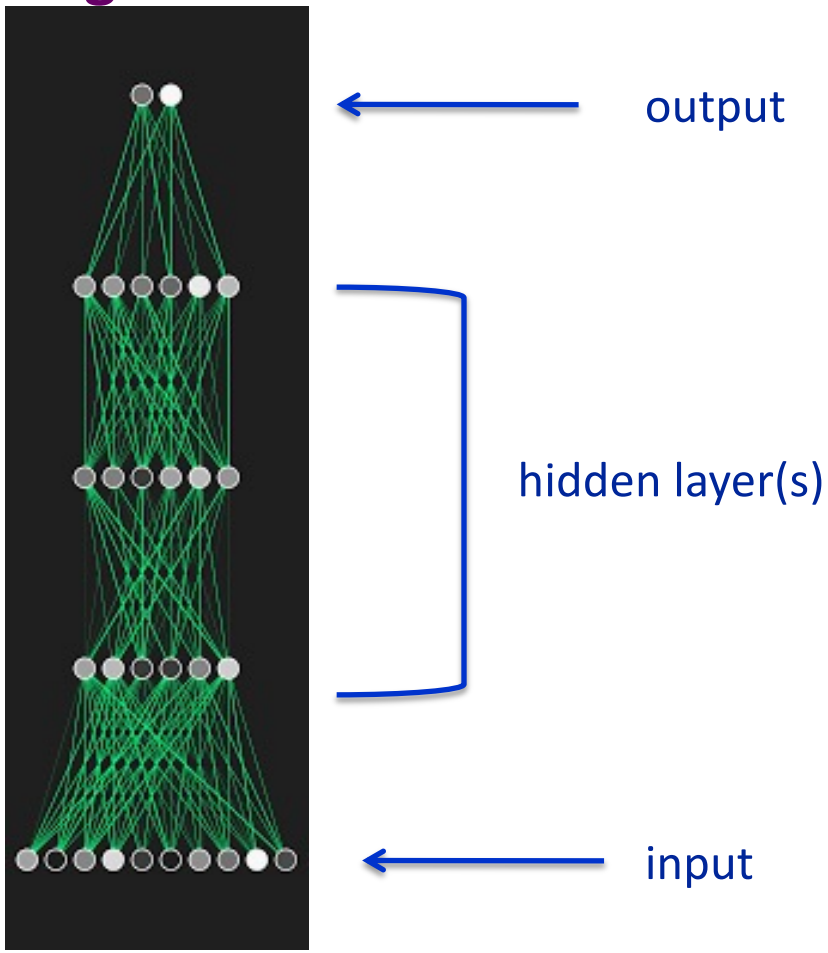
Main technical contribution:

- study of basic Boolean functions and properties
- showing that some of them can be efficiently tested



Testing a Neural Network

- Underlying model:



A simple feedforward network with ReLU activation function

We consider a feedforward network with one hidden layer and ReLU activation function.

- The network is fully connected,
- has n input nodes labeled $\{1, \ldots, n\}$,
- a hidden layer with m nodes labeled $\{1, \ldots, m\}$, and
- one output node.

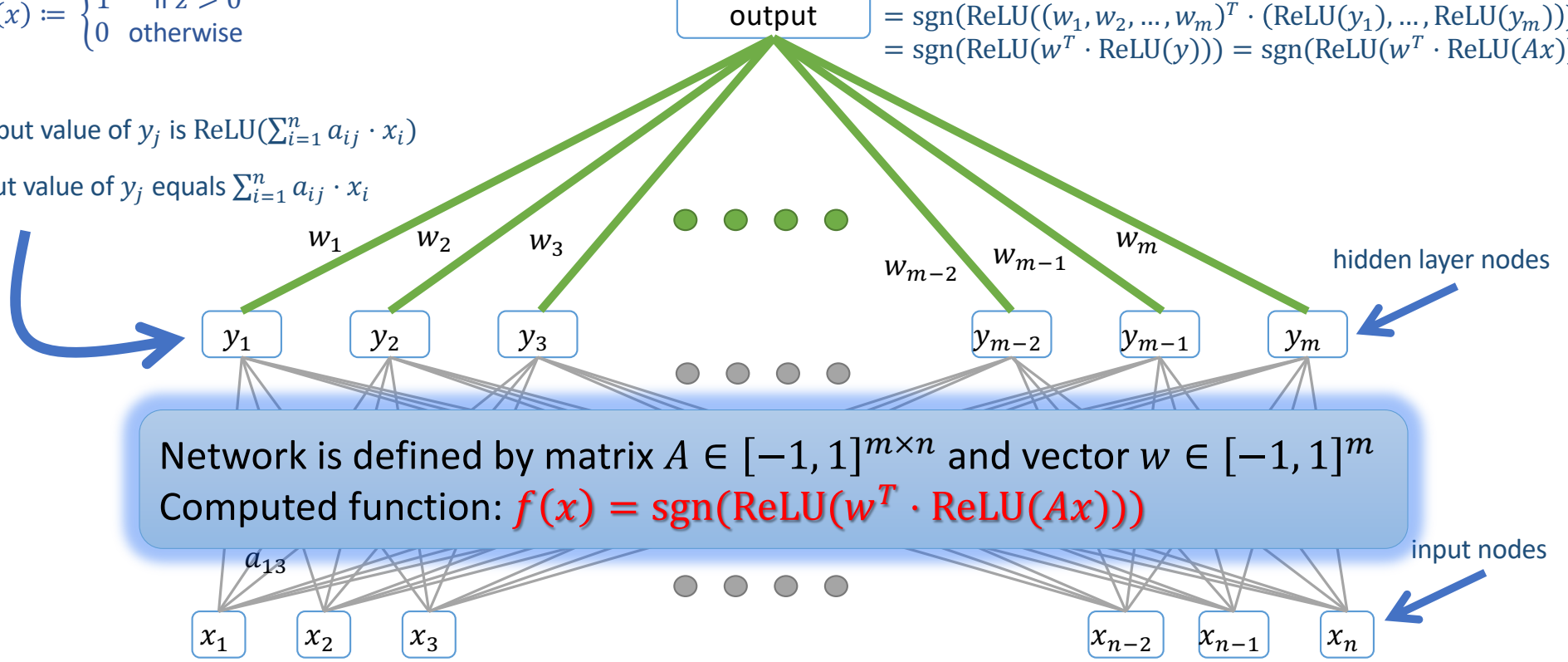
output value is equal to sign of $\mathrm{ReLU}(\sum_{j=1}^{m} w_j \cdot y_j)$

$$\mathrm{sgn}(x) := \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

output $= \mathrm{sgn}(\mathrm{ReLU}((w_1, w_2, \dots, w_m)^T \cdot (\mathrm{ReLU}(y_1), \dots, \mathrm{ReLU}(y_m))))$
$= \mathrm{sgn}(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(y))) = \mathrm{sgn}(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(Ax)))$

output value of y_j is $\mathrm{ReLU}(\sum_{i=1}^{n} a_{ij} \cdot x_i)$

input value of y_j equals $\sum_{i=1}^{n} a_{ij} \cdot x_i$



$a_{ij}, w_i \in [-1,1]$

ReLU activation function: $ReLU(u) = \max\{0, u\}$

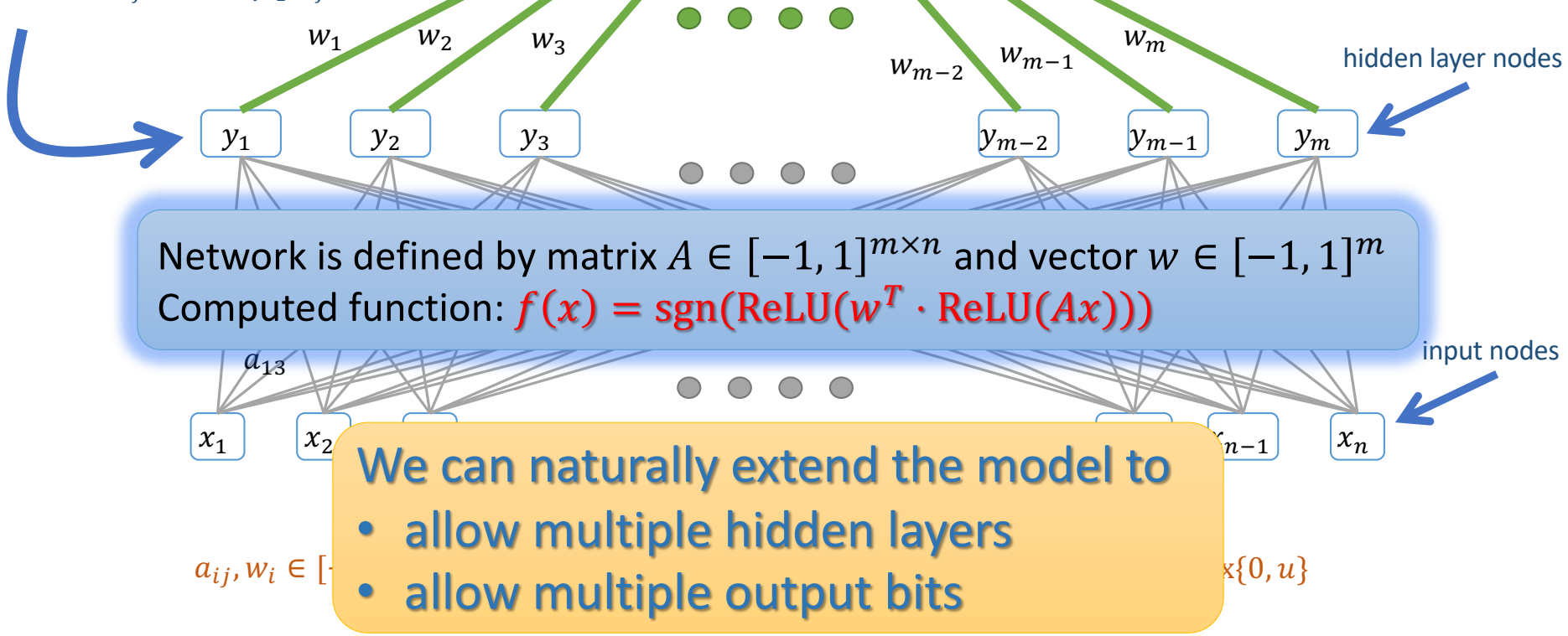
output value is equal to sign of $\mathrm{ReLU}(\sum_{j=1}^{m} w_j \cdot y_j)$

$$\mathrm{sgn}(x) := \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

output $= \mathrm{sgn}(\mathrm{ReLU}((w_1, w_2, \dots, w_m)^T \cdot (\mathrm{ReLU}(y_1), \dots, \mathrm{ReLU}(y_m))))$
$= \mathrm{sgn}(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(y))) = \mathrm{sgn}(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(Ax)))$

output value of y_j is $\mathrm{ReLU}(\sum_{i=1}^{n} a_{ij} \cdot x_i)$

input value of y_j equals $\sum_{i=1}^{n} a_{ij} \cdot x_i$



Network is defined by matrix $A \in [-1,1]^{m \times n}$ and vector $w \in [-1,1]^m$
Computed function: $f(x) = \mathrm{sgn}(\mathrm{ReLU}(w^T \cdot \mathrm{ReLU}(Ax)))$

We can naturally extend the model to

- allow multiple hidden layers
- allow multiple output bits

$a_{ij}, w_i \in [$ … $x\{0, u\}$

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, which we define as 0 and 1.

If the output value of the network is ≤ 0 we assign class 0, if it is > 0 we assign class 1.

We focus on the setting when $x \in \{0,1\}^n$ is a binary vector.

ReLU network is defined by a pair (A, w) where $A \in [-1,1]^{m \times n}$ and $w \in [-1,1]^m$

➔ the neural network defines a function $f : \{0,1\}^n \to \{0,1\}$ that computes

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

Testing a Neural Network

Properties of Boolean functions:

A property of Boolean functions is a family $P = \bigcup_{n\in\mathbb{N}} P_n$,

- where P_n is a set of Boolean functions $f: \{0,1\}^n \to \{0,1\}$

ReLU network being far from computing a function:

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 $f: \{0,1\}^n \to \{0,1\}$, if

- one can change the matrix A in $\leq \varepsilon nm$ places and the weight vector w in $\leq \varepsilon m$ places
- to obtain a ReLU network that computes a function g such that

$$\Pr[g(x) \neq f(x)] \leq \delta,$$

where x is chosen uniformly at random from $\{0,1\}^n$.

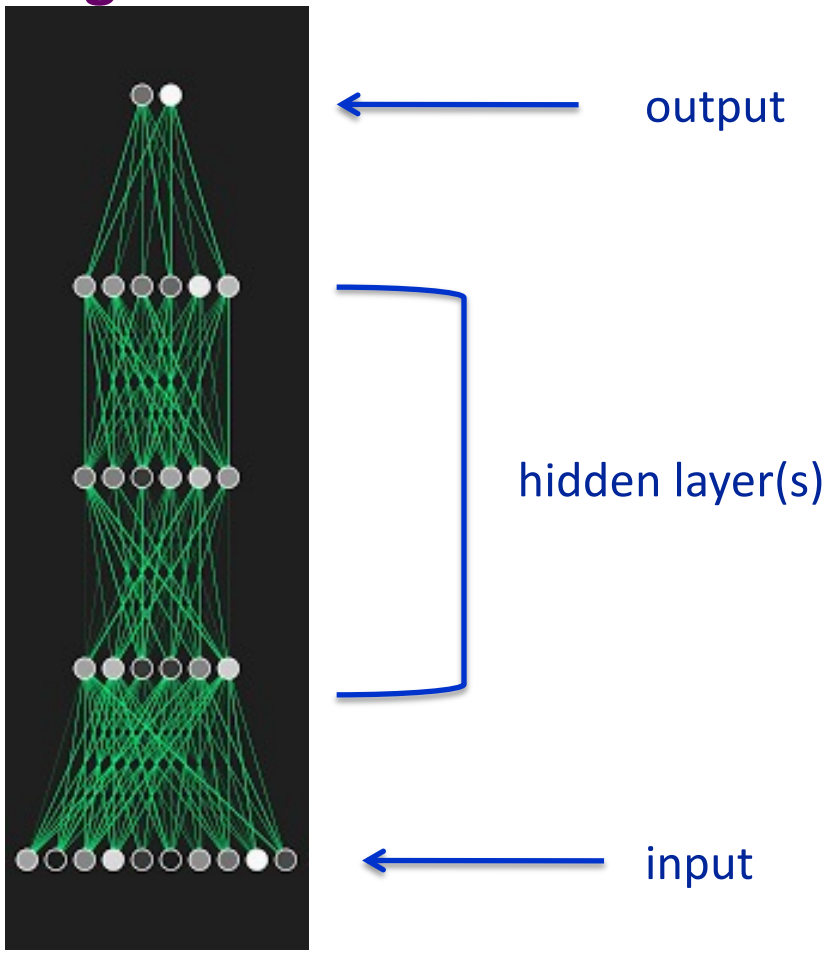
If (A, w) is not (ε, δ)-close to computing f ➔ (A, w) is (ε, δ)-far from computing f

Testing a Neural Network

Network is (ε, δ)-close to computing a function f :

we can edit the network to

- make it to compute f on $(1-\delta)$-fraction of the inputs
- by modifying an ε-fraction of weights at every layer



Testing a Neural Network

Properties of Boolean functions:

A property of Boolean functions is a family $P = \bigcup_{n\in\mathbb{N}} P_n$,

- where P_n is a set of Boolean functions $f: \{0,1\}^n \to \{0,1\}$

ReLU network being far from computing a function:

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 $f: \{0,1\}^n \to \{0,1\}$, if

- one can change the matrix A in $\leq \varepsilon nm$ places and the weight vector w in $\leq \varepsilon m$ places
- to obtain a ReLU network that computes a function g such that

$$\Pr[g(x) \neq f(x)] \leq \delta,$$

where x is chosen uniformly at random from $\{0,1\}^n$.

If (A, w) is not (ε, δ)-close to computing f ➔ (A, w) is (ε, δ)-far from computing f

Testing a Neural Network

Properties of Boolean functions:

Extension to properties of functions

A property of Boolean functions is a family $P = \bigcup_{n\in\mathbb{N}} P_n$,

- where P_n is a set of Boolean functions $f: \{0,1\}^n \to \{0,1\}$

ReLU network being far from computing a function:

Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes.

(A, w) is (ε, δ)-close to computing a function $f: \{0,1\}^n \to \{0,1\}$ with property P, if

- one can change the matrix A in $\leq \varepsilon nm$ places and the weight vector w in $\leq \varepsilon m$ places
- to obtain a ReLU network that computes a function g such that

$$\Pr[g(x) \neq f(x)] \leq \delta,$$

where x is chosen uniformly at random from $\{0,1\}^n$.

If (A, w) is not (ε, δ)-close to computing f ➔ (A, w) is (ε, δ)-far from computing f with P

Testing a Neural Network

Property $P = \bigcup_{n \in \mathbb{N}} P_n$ of Boolean functions is testable with query complexity $q(\varepsilon, \delta, n, m)$

- if for every $n, m \in N$, $\varepsilon, \delta \in (0, 1)$
- there exists an algorithm that
- receives parameters $n, m, \varepsilon, \delta$ as input and
- is given query access to a ReLU network with n input nodes and m hidden layer nodes,
- that queries the ReLU network in at most $q(\varepsilon, \delta, n, m)$ many places,
- accepts with probability at least $2/3$ if the network computes a function from P_n, and
- rejects with probability at least $2/3$ every network that computes a function that is (ε, δ)-far from P_n.

A tester that accepts every function from P with probability 1 is called one-sided tester.

If the function q depends only on ε and δ we just say that the property P is testable.

Testing a Neural Network: What can we show?

Our key findings:

- study of basic functions
  - understanding testability of constant 0-function, OR-function, and near-constant functions – for one and multiple hidden layer networks
- distribution-free model is too strong
- we can test functions faster than in the vanilla testing algorithm
- …

Testing a Neural Network: does it compute constant 0 function?

- We consider most basic functions – to understand which models are suitable
- Determining whether a network computes the constant 0-function is NP-hard

Testing a Neural Network: does it compute constant 0 function?

Vanilla testing algorithm:

- Sample many inputs at random and check if the network computes 0 on all of them

- There are networks that are is (ε, δ)-far from computing the constant 0-function (for constant ε and $\delta = 2^{-\Theta(n)}$) that require $2^{\Omega(n)}$ samples

- In our model: we can do it MUCH better

Testing a Neural Network: constant 0-function

Theorem: Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes.

Let $e^{-n/16} \leq \delta < 1$ and let $0 < \varepsilon, \lambda < 1/2$.

Then A sample nodes and evaluate the restricted network with a suitable bias at the output node whether it computes the constant 0-function

- acc ion;
- rejects with prob $\geq 1 - \lambda$, if the ReLU network (A, w) is (ε, δ)-far from computing the 0-function;

Furthermore, the algorithm queries $O(\log^2(1/\varepsilon\lambda)/\varepsilon^6)$ entries from A and w.

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

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 $\frac{nm}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx) - \varepsilon nm/8 > 0$ **then reject**;

4 **else accept**.

Dependence $e^{-n/16} \leq \delta$ can be "moved" into the query complexity if n and m are polynomially related.

Testing a Neural Network: constant 0-function

Theorem: Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes.

Let $e^{-n/16} \leq \delta < 1$ and let $0 < \varepsilon, \lambda < 1/2$.

Then AllZeroTester$(\varepsilon, \delta, \lambda, n, m)$ with query access to A and w

- accepts with prob $\geq 1 - \lambda$ if the ReLU network (A, w) computes the 0-function;
- rejects with prob $\geq 1 - \lambda$ if the ReLU network (A, w) is (ε, δ)-far from computing

1) if the network is (ε, δ)-far from computing the constant 0-function then there exists an input $x \in \{0,1\}^n$ on which the network has an output value of $\Omega(\varepsilon nm)$
   - sampling a constant number of hidden layer nodes one can approximate the output value for this particular x with sufficient accuracy so that the output value is $\Omega(\varepsilon nm)$
2) we can also sample a constant number of input nodes and approximate the value at the sampled hidden layer nodes in such a way that the value at the output node is positive

3 **if** there is $x \in \{0,1\}^n$ with $\frac{nm}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx) - \varepsilon nm/8 > 0$ **then reject**;
4 **else accept**.

Dependence $e^{-n/16} \leq \delta$ can be "moved" into the query complexity if n and m are polynomially related.

Testing a Neural Network: constant 0-function

Theorem: Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes.

Let $e^{-n/16} \leq \delta < 1$ and let $0 < \varepsilon, \lambda < 1/2$.

Then AllZeroTester$(\varepsilon, \delta, \lambda, n, m)$ with query access to A and w

- accepts with prob $\geq 1 - \lambda$ if the ReLU network (A, w) computes the 0-function;
- rejects with prob $\geq 1 - \lambda$ if the ReLU network (A, w) is (ε, δ)-far from computing

Algorithm accepts a network that computes the constant 0-function?
Claim: for all input vectors x the sampled network computes the constant 0-function.
Consider the sample from the input nodes as being fixed
Restricting our network to the sampled input nodes can also be done by restricting the input to vectors x that are 0 for all input nodes that do not belong to the sample.
➔ our sampling approach = only sampling from the hidden layer
➔ concentration bound + union bound over all ➔ proof

4 else accept.

Dependence $e^{-n/16} \leq \delta$ can be "moved" into the query complexity if n and m are polynomially related.

Testing a Neural Network: constant 0-function

Theorem: Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes.

Let $e^{-n/16} \leq \delta < 1$ and let $0 < \varepsilon, \lambda < 1/2$.

Then AllZeroTe

A similar result can be shown for the OR-function

- accepts with prob $\geq 1 - \lambda$ if the ReLU network (A, w) computes the 0-function;
- rejects with prob $\geq 1 - \lambda$, if the ReLU network (A, w) is (ε, δ)-far from computing the 0-function;

Furthermore, the algorithm queries $O(\log^2(1/\varepsilon\lambda)/\varepsilon^6)$ entries from A and w.

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

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 $\frac{nm}{st} \cdot w^T \cdot \mathrm{ReLU}(TASx) - \varepsilon nm/8 > 0$ **then reject**;

4 **else accept**.

Dependence $e^{-n/16} \leq \delta$ can be "moved" into the query complexity if n and m are polynomially related.

Testing a Neural Network: constant 0-function

Algorithm above has two-sided error (may err while accepting and rejecting)

Can we do the same with 1-sided error?

Theorem: Any tester for the constant 0-function that has one-sided error has query complexity of $\Omega(m)$.

Similar results can be shown for the OR-function

At t

Follows from the fact that if too few weights are specified, one can always "complete" the network in such a way that it computes the constant 0-function

---

1 Sample $s = \lceil \frac{2^{20} \log m}{\varepsilon^2} \cdot \ln(\frac{1}{\lambda}) \rceil$ nodes from the input layer uniformly at random without replacement.

2 Let $S \in \mathbb{N}_0^{n \times n}$ be the corresponding sampling matrix.

3 **if** there is $x \in \{0,1\}^n$ with $w^T \cdot \mathrm{ReLU}(ASx) > 0$ **then reject**;

4 **else accept**.

Testing a Neural Network: 0 and OR are central …

Theorem: Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes.

Let $0 < \delta \leq 1/2$ and let $\varepsilon \geq 2\sqrt{\frac{\log(1/\delta)}{n}}$.

Then (A, w) is (ε, δ)-close to computing the OR-function or it is (ε, δ)-close to computing the constant 0-function.

This does not imply that all functions are pairwise $(2\varepsilon, 2\delta)$-close as our distance measure does not satisfy the triangle inequality.

Testing a Neural Network: 0 and OR are central …

Theorem: Let (A, w) be a ReLU network with n input nodes and m hidden layer nodes.

Let $0 < \delta \leq 1/2$ and let $\varepsilon \geq 2\sqrt{\frac{\log(1/\delta)}{n}}$.

Then (A, w) is (ε, δ)-close to computing the OR-function or it is (ε, δ)-close to computing the constant 0-function.

Corollary: Every property that contains the constant 0-function and the OR-function is trivially testable.

A Boolean function f is ***monotone***, if for every $x, y \in \{0,1\}^n$, $x \leq y$ implies $f(x) \leq f(y)$.

A Boolean function is ***symmetric***, if its value does not depend on the order of its arguments.

Corollary: Monotone/symmetric functions are testable.

Testing monotone properties

We consider also monotone properties.

A property P is monotone, if it is closed under flipping bits in the truth table of any function from the property 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.

Theorem: Every monotone property is testable with query complexity logarithmically on the size of the generator.

ReLU networks with multiple output bits

- Our main focus was on a Boolean functions
- We can extend some of the study to multiple output bits

For a vector $u = (u_1, \dots, u_r)$, let $\mathrm{sgn}(u) = (\mathrm{sgn}(u_1), \dots, \mathrm{sgn}(u_r))$

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

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

- All networks are close to a near-constant function
- Testing a near-constant function can be done with constant query complexity

Since for input 0, the output must be 0 ➔ near-constant function = constant except for input 0

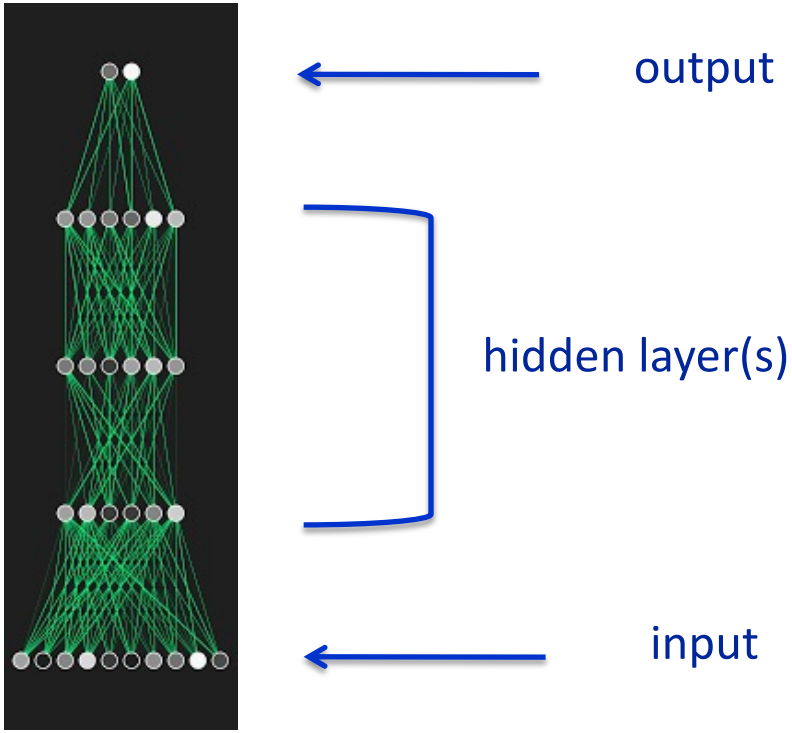
ReLU networks with multiple hidden layers

- Our main focus was on a Boolean functions with one hidden layer
- We can extend some of the study to multiple hidden layers

Network is (ε, δ)-close to computing a function f :

we can edit the network to

- make it to compute f on $(1-\delta)$-fraction of the inputs
- by modifying an ε-fraction of weights at every layer



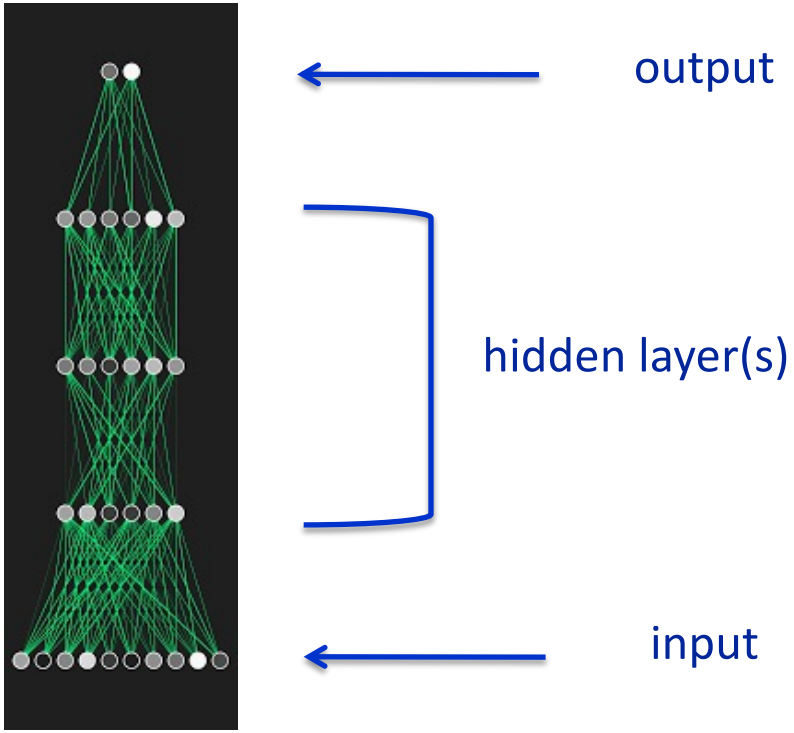
ReLU networks with multiple hidden layers and multiple output bits

- Testing a near-constant function can be done with constant query complexity

Network is (ε, δ)-close to computing a function f :

we can edit the network to

- make it to compute f on $(1-\delta)$-fraction of the inputs
- by modifying an ε-fraction of weights at every layer



Distribution-free model?

Our ma… respect to the s…

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

Distribution-free model:

- an unknown distribution D over $\{0,1\}^n$
- query access: $\mathrm{query}(i, j)$ – sample input $y_i \in \{0,1\}^n$ according to D and return jth bit of y_i

(A, w) is called (ε, δ)-close to computing a function $f: \{0,1\}^n \to \{0,1\}$, if

- one can change the matrix A in $\leq \varepsilon nm$ places and the weight vector w in $\leq \varepsilon m$ places
- to obtain a ReLU network that computes a function g such that

$$\Pr[g(x) \neq f(x)] \leq \delta,$$

where x is chosen at random according to distribution D.

Distribution-free model?

Our ma… respect to the s…

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

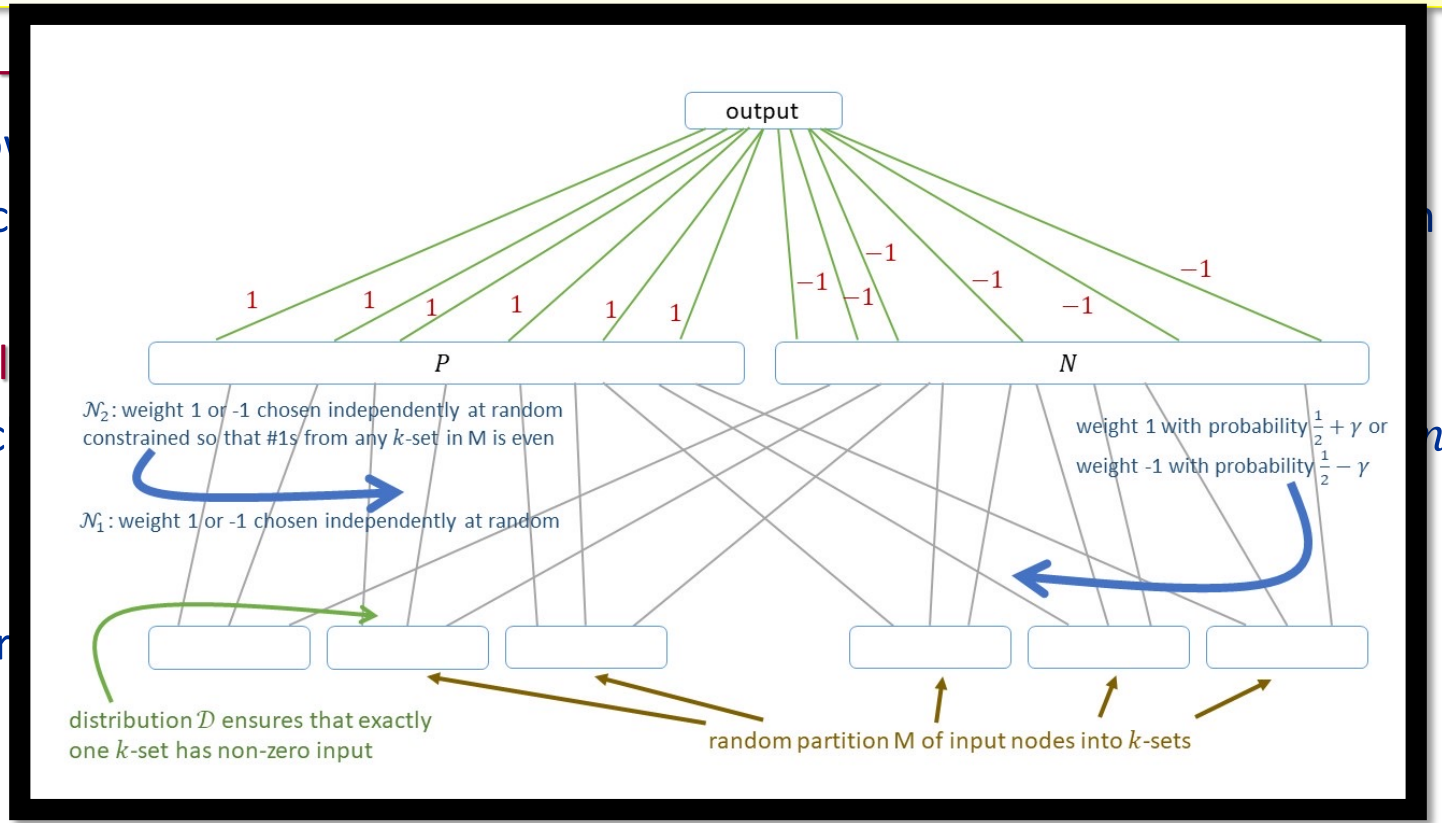
Distribution-…

- an unkno…
- query acc… jth bit of y_i

(A, w) is call…

- one can c… places
- to obtain …

wher…



Summary

- We introduced a property testing model to study neural networks
- Gave some first results in this model

- Large number of open questions regarding ReLU networks
  - query complexity of testing dictatorship and juntas
  - classify constant time testable properties with one- and two-sided error
  - extend the results to inputs over the reals
  - going beyond ReLU networks
  - …

THANK YOU!