Electronic Colloquium on Computational Complexity, Report No. 159 (2022)

Deep Neural Networks: The Missing Complexity Parameter

Songhua He
Rutgers University

Periklis A. Papakonstantinou
Rutgers University

November 18, 2022

Abstract

Deep neural networks are the dominant machine learning model. We show that this model is missing a crucial complexity parameter. Today, the standard neural network (NN) model is a circuit whose gates (neurons) are ReLU units. The complexity of a NN is quantified by the depth (number of layers) and the size (number of neurons = depth times width). This work shows that this alone is insufficient, resulting in NNs with unreasonable computing power. We show that the correct way to talk about the size complexity of a NN is besides the number of neurons to consider the precision (or magnitude) of the weights of the ReLU units. The main message of this work is that if the precision of the weights is not considered in the complexity of the NN then one can engineer weights to "buy" exponentially many neurons for free. In summary, we make three theoretical contributions, potentially affecting many theoretical works on NNs.

- Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed with $O(\sqrt{2^n})$ many neurons and constant fan-in per neuron; i.e. exponential times less than Shannon's classic lower bound for usual combinatorial circuits.
- We give a new definition of circuit size that takes into account the precision/magnitude of the weights. Under this new definition of size we asymptotically match Shannon's bound for NNs.
- We complement the above results showing that P-uniform NNs decide exactly P.

1 Introduction

Neural networks (NNs) have been intensively studied and in the last decade have taken the world by storm. In this work, we revisit the deep neural networks model. We show that the standard model, that is, the networks whose nodes are associated with ReLU function as the activation function, has unreasonably high power. We call these networks as "ReLU networks" or "ReLU circuits".

A ReLU network is an edge-weighted circuit whose gates (neurons) are ReLU functions ($\sigma(x) = x, x > 0$ and $\sigma(x) = 0, x \leq 0$) evaluated over the weighted sum of their inputs. As a circuit, a ReLU network encodes a real-valued function. To compare with boolean models, we study ReLU networks on boolean inputs, which is not an actual restriction in a world of finite precision.

Allowing high precision or equivalently large-number arithmetic often makes a computational model stronger than expected. This is true even for problems that have small inputs. Here, "small" means that every element listed in the input has bit representation at most a polynomial in the input length parameter n. For example, even if the input is small, such as a CNF formula for the SAT problem (satisfiability), an arbitrary-precision computing model is able to internally

ISSN 1433-8092

trade computing power for numerical precision. For instance, the polynomial-time unit-cost RAM (Random Access Machine) model is as powerful as polynomial-space Turing machines [Sch79]; see also [HS74, BMS85, vE90]. Similarly, for the long-standing open question of solving a semidefinite program (SDP) exactly. Although, SDPs are known to be solvable in polynomial time for a tolerance parameter $\varepsilon > 0$, the exact version of the same problem is conjectured to be NP-hard [TV08, ABKPM09]. The same thing holds for the "sum of square roots" problem. In this problem the input consists of two lists of positive integers $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ and one wants to decide whether $\sum_{i=1}^{n} \sqrt{a_i} > \sum_{i=1}^{n} \sqrt{b_i}$. The "sum of square roots" problems has an obvious linear time algorithm in the real-RAM model. However, its complexity is a long-standing open question in a practical model of computation such as a polynomial time Turing Machine [O1]. As of now, this problem is not even known to be in NP.

The standard Neural Networks model allows for weights in the ReLU units which are arbitrary real numbers. We ask what happens when these numbers are real numbers with high (but finite) precision or equivalently when they can be arbitrarily large integers. Theorem 1 (see below) says that this way we give unreasonable power to the computing model. This theorem is true for inputs, which are small, such as a binary input of length n. The technical reasons (proof of Theorem 1) why typical ReLU networks are unreasonably powerful are rather different than the similar situation in e.g. the unit-cost RAM. However, conceptually, we get a similar conclusion.

Theorem 1 (informal)**.** *Every function can be computed in the standard model of ReLU circuits (i.e. with unbounded weights) with exponentially smaller number of neurons (or number of wires) compared to the regular combinatorial circuits (and with the same fan-in in both models).*

To put this theorem in proper context we recall that Shannon proved [Sha49] that almost every boolean function requires boolean circuit of size $\Omega(2^n/n)$ (see also [RS42]). This bound holds for circuits of constant fan-in – to give a fair comparison we will also consider NNs of constant fan-in. Shannon's lower bound is asymptotically tight, since Lupanov [Lup58] showed that every boolean function can be computed in size $O(2^n/n)$. Later on, when we state Theorem 1 formally we will see that we can compute every boolean function using a number of edges which is smaller than Shannon's bound. In particular, we show that every boolean function can be computed in size $O(\sqrt{2^n}) = o(2^n/n)$ with bounded fan-in. Going asymptotically below the lower bound is what appears to give the model unreasonable power. Our proof uses a similar idea to the construction in [Dan96] (however, unlike [Dan96], we do not blow up exponentially the fan-in[1]). Philosophically, our argument beats a version of the extended Church-Turing thesis. This is because the proof of Theorem 1 contains an algorithm and the same theorem can be restated for various degrees of uniformity of the underlying ReLU network.

This discussion does not mean that NNs are super-powerful in practice. We believe that nothing in the classical world is more powerful than a Turing Machine within polynomial factors. It means that the complexity resource cannot only be the number of its ReLU units or the number of edges/wires. To that end, we introduce a new definition of "size", which takes into account the precision and the magnitude of NN weights. Under this new "precision-sensitive ReLU size" definition we show the following.

[1]In other words, [Dan96] shows that if we define as "size" the number of gates then there are savings, however if the size is the number of wires then nothing changes. Our NNs will be of constant fan-in and thus we can focus on the number of NNs and address the issue about the magnitude of the weights.

Theorem 2 (informal)**.** *The model of ReLU networks using the "precision-sensitive ReLU size" has the same computing power as usual combinatorial circuits.*

Our third and last result states that when the NN is not precomputed, but rather printed out efficiently by a polynomial time Turing Machine, then the conclusion of Theorem 1 is no longer true.

Theorem 3 (informal)**.** *The set of all problems computable in polynomial time is the set of problems computable by poly-time uniform ReLU circuits.*

We note that a similar conclusion as Theorem 3 is not true for other models of computation.

Let us conclude with two remarks regarding our results in the context of Machine Learning. We do not know how (or if there are any) learning algorithms that give more-than-usual power to a learned ReLU model. For example, we do not know if there is a learning algorithm for neural networks, which for a fixed number of ReLU units the more it trains the more it can make use of the better precision of the weights. This seems to be an interesting open question that can be studied both theoretically and empirically. Our results seem to be of relevance for learning theory works that prove lower bounds for neural networks.

In the rest of the paper we first introduce necessary notation as well as the new NN size definition in Section 2. We also construct gadgets useful for constructing ReLU networks in Section 3. In Section 4, we prove that ReLU networks are strictly more powerful than standard circuit models with the same number of gates. In Section 5, we show that ReLU networks under our new definition of size match Shannon's circuit lower-bound. Finally, in Section 6, we show that P-uniform ReLU networks exactly computes problems in P.

2 Preliminaries

In this section, we provide definitions and notations.

For any predicate P, define the Iverson bracket $[P]$ to be 1 iff P is true, and $[P] = 0$ otherwise. $\mathrm{LCM}(x_1, ..., x_k)$ is the least common multiple of the integer x_is, and $\mathrm{GCD}(x_1, ..., x_k)$ is their greatest common divisor. All logarithms are of base 2.

2.1 ReLU Networks and Other Models

The formal definition of ReLU networks is as follows:

Definition 4 (see [ADH+19])**.** *A ReLU network C is an edge-weighted acyclic directed graph $G = (V, E)$. The sources (i.e. vertices with in-degree 0) of the graph are either inputs or constants of real values. To differentiate with sources, non-source vertices are called neurons (gates). The value v_x of a neuron x is given by ReLU functions:*

$$v_x =_{def} \sigma\Big(\sum_{(y,x)\in E} w_{(y,x)} \cdot v_y\Big)$$

where $w_{(y,x)} \in \mathbb{R}$ is the weight of edge (y, x), and σ is the ReLU function defined as $\sigma(x) =_{def} \max(x, 0)$. Besides, the sinks (i.e. vertices with out-degree 0) are the outputs of the ReLU network.

In this work, the inputs are either 0 or 1. We shall omit the activation function $\sigma(x)$ if it is clear that x is non-negative.

Other usual computational models such as Turing machines, RAM model, boolean circuits, and so on, have the same power in the sense that each one of them can simulate other models with an at most polynomial loss. In this work, we will mainly compare ReLU networks to boolean circuits.

To show that ReLU networks can simulate boolean circuits efficiently, we shall first provide the definition of boolean circuits. A *boolean circuit* is a directed acyclic graph, whose inputs are bits, and gates are boolean operators. For boolean circuits with bounded fan-in (i.e., in-degree), they only use the binary and unary operators $\wedge, \vee, \neg$ as gates. Boolean circuits with unbounded fan-in use $\wedge_n, \vee_n, \neg$ for any n.

2.2 Circuit Size

Prior to our work, in neural networks, the size is defined as the product of depth and width, which is the number of neurons. As for boolean circuits, the size of circuits are either defined as the number of gates or the number of edges, where for notational convenience we choose the latter one as a definition of circuit size. For boolean circuits with unbounded fan-in and ReLU networks, we use the latter definition in order to apply Shannon's circuit lower-bound on these models.

To address the issue of limited precision in the weigths we introduce the following definition.

Definition 5. *For a ReLU network C whose weights $\{w_{x,y}\}$ are all rational numbers, we define the precision size of C to be the sum of bits to describe each weight:*

$$p\text{-}SIZE(C) = \sum_{(x,y)\in E} \Big(\lceil \log(|p_{x,y}| + 1) \rceil + \lceil \log(|q_{x,y}| + 1) \rceil \Big)$$

where E is the edge set of C, and $p_{x,y}/q_{x,y} = w_{x,y}$ in which $p_{x,y}$, $q_{x,y}$ are integers.

Henceforth, for clarity, we shall assume that $p_{x,y}$ and $q_{x,y}$ form an irreducible fraction $w_{x,y}$.

The above definition quantifies the number of bits we need to describe the weights of a circuit. It is equal to the number of binary bits we need to write all the integers $(|p_{x,y}|), (|q_{x,y}|)$.

3 Gadgets for ReLU Networks

In this section, we give gadgets for building ReLU networks. These gadgets are used throughout our paper.

Lemma 6. *For input $x \in S$ where $S \subset \mathbb{R}$ is a finite set, there is a ReLU network of constant size computing $[x \geq c]$.*

Proof. Since S is a finite set, there exists a $c' < c \in \mathbb{R}$ such that if $x < c$ then $x < c'$. We first compute: $y := \sigma(\frac{1}{c-c'}x - \frac{c'}{c-c'})$. It's non-zero iff $x > c'$, and equals to 1 iff $x = c$. We have

$$[x \geq c] = \sigma(y - \sigma(y - 1))$$

□

To put things in context, here is an example of constructing $[x \le c]$ and $[x = c]$ by $[x \ge c]$:

$$[x \le c] = [(-x) \ge (-c)]$$

$$[x = c] = [[x \ge c] + [x \le c] \ge 2]$$

Other relations between x and c can be obtained similarly from the gadget above.

Fact 7. *For input $x, y \in \mathbb{R}_{\ge 0}$ bounded by a sufficiently big constant C, and a switch $z \in \{0, 1\}$, there exists a ReLU network computing the formula below in constant size*

$$(z \ ? \ x : y) =_{def} \begin{cases} x, & \text{if } z = 0 \\ y, & \text{if } z = 1 \end{cases}$$

Proof. We construct it directly:

$$(z \ ? \ x : y) = \sigma(x - z \cdot C) + \sigma(y - (1 - z) \cdot C)$$

□

We can construct "if swithes" the same way as above.

Lemma 8. *The boolean functions $\neg$ and $\wedge_n, \vee_n$ for any n can be implemented by ReLU networks of constant neurons.*

Proof. Negation function $\neg x$ is $1 - x$.

For the remaining functions, we first use a single ReLU neuron to get the sum of the n input bits $s = \sigma(\sum_{i=1}^{n} x_i)$. Then we can compute them by comparing s to particular numbers:

$$\wedge_n = [s = n]$$

$$\vee_n = [s \ge 1]$$

□

4 ReLU Networks Are Strictly Stronger than Boolean Circuits with the Same Number of Gates (and the Same Fan-in)

Lemma 8 says that ReLU networks are at least as powerful as boolean circuits, while boolean circuits are composed of AND, OR and NOT gates. In this section, we show that ReLU networks are strictly stronger than boolean circuits, by showing that ReLU networks can compute particular functions with asymptotically less number of neurons and wires. We restate Theorem 1 here formally.

Theorem 1. *For any boolean function $f : \{0, 1\}^n \to \{0, 1\}$, there exists a ReLU network of size $O(2^{n/2})$ that computes f.*

This upper-bound is exponentially smaller than the $\Theta(2^n/n)$ lower-bound given by Shannon.

Algorithm 1 2^n-sized circuit for looking up the truth table

Input: $x_1, ..., x_n, T_f$
Output: The $(x_1...x_n)_2$-th bit of T_f

```
Sum := 0
for k := 2^n − 1 to 0 do
    Sum := Sum + ([T_f ≥ 2^k] ∧ [(x_1...x_n)_2 = k])
    T_f := ([T_f ≥ 2^k] ? T_f − 2^k : T_f)
end for
Output := Sum
```

Overview of the algorithm that beats Shannon's bound. Denote by the inputs of the network $x_1, x_2, ..., x_n \in \{0, 1\}$. The algorithm consists of two parts. First, we print the truth table of the function $f(x_1, x_2, ..., x_{\lfloor n/2 \rfloor})$ by fixing the first $\lfloor n/2 \rfloor$ inputs. Then we "lookup" and output the $(x_{\lfloor n/2 \rfloor+1}...x_n)_2$-th item of the truth table.

Here are the details of our argument.

We construct the circuit by encoding the whole truth table T_f of a boolean function f into the weights. Formally, T_f is a big constant whose binary representation encodes the function f:

$$T_f = (t_0 t_1 ... t_{2^n-1})_2$$

where for all $x_1, ..., x_n \in \{0, 1\}$,

$$t_{(x_1...x_n)_2} = f(x_1, ..., x_n)$$

The way the proof goes should have the ability to "lookup" whether a specific entry in the truth-table is 0 or 1. However, we note that it is hard to do so by a small number of neurons.

Lemma 9. *Given $x_1, ..., x_n \in \{0, 1\}$ and $T_f \in \{0, 1, ..., 2^{2^n} - 1\}$ as inputs, there exists a ReLU network of size $O(2^n)$ whose output is the $(x_1...x_n)_2$-th bit of T_f.*

Proof. We give the construction as pseudocode in Algorithm 1.

The procedure slices T_f bit-by-bit, and determines whether or not the highest bit of T_f is 1.

The for-loop can be simply implemented sequentially in the circuit. The operators involved can all be implemented in constant size by Lemmas 6, 7 and 8. By precomputing $(x_1 x_2 ... x_n)_2 = \sum_{i=1}^{n} x_i \cdot 2^{n-i}$, the formula $[(x_1 x_2 ... x_n)_2 = k]$ also costs constant size as stated in Lemma 6. The total size of construction above is $O(2^n)$. □

It is worth noting that T_f is not necessarily an input in the lemma above, as the truth table T_f is a constant not depending on the input. But we will come to the case T_f is a variable in the proof of Theorem 1 later.

We will not use the above lemma directly on the given function. Note that this size is worse than the $2^n/n$ size in Lupanov's construction for boolean circuits. However, we will use Lemma 9 after we partition the truth table using a different algorithmic idea.

Proof of Theorem 1. Let $T_{f(x_1,...,x_i)} \in \{0, 1, ..., 2^{2^{n-i}} - 1\}$ be the truth table of f fixing the first i bits.

Still, we give the construction as pseudocode in Algorithm 2.

Algorithm 2 $2^{n/2}$-sized circuit for computing f

Input: $x_1, ..., x_n$
Output: $f(x_1, ..., x_n)$

1: $T := 0$
2: **for** $k := 0$ to $2^{\lfloor n/2 \rfloor} - 1$ **do**
3: $\quad T := T + ([(x_1...x_{\lfloor n/2 \rfloor})_2 = k]\ ?\ T_{f(k_1,...,k_{\lfloor n/2 \rfloor})} : 0)$
4: **end for**
5: Output := the $(x_{\lfloor n/2 \rfloor+1}...x_n)$-th bit of T

In the construction we first iteratively search the truth table of $T_{f(x_1,...,x_{\lfloor n/2 \rfloor})}$ in $O(2^{n/2})$ size, then lookup the truth table using Lemma 9. The total size of the circuit is still $O(2^{n/2})$. □

Remark. A relevant work [Dan96] showed that computing any boolean function in unbounded fan-in boolean circuits can be done in $O(2^{n/2})$ gates as well, which is proved by a similar construction as ours. Importantly, the construction in [Dan96] uses $O(2^n)$ many wires, whereas ours uses $O(2^{n/2})$. And our construction above consists of neurons of only constant fan-in. This still implies a huge gap between ReLU networks and unbounded fan-in boolean circuits.

Theorem 1 shows that ReLU networks, the standard model of neural networks, are strictly stronger than boolean circuits. In contrast, Shannon's Theorem [Sha49] told us that most n-ary boolean functions require boolean circuits of size $2^n/n$, when n is big enough.

5 A ReLU Lower Bound that Matches Shannon: The New ReLU Size Definition

We show that under Definition 5, ReLU networks have the same computational power as regular models, i.e., boolean circuits.

Let S_f denote the minimal p-SIZE of a ReLU network computing a boolean function f. We obtain that

Theorem 2. *For any sufficiently large n,*

$$\max_{f:\{0,1\}^n \to \{0,1\}} S_f > 2^{n-1}/n$$

The theorem implies that, any n-ary boolean function can be computed by a ReLU network of p-SIZE at most $2^{n-1}/n$.

Proof. We proceed with a counting argument.

- Count the number of different ReLU networks over n variables and of p-SIZE at most $2^{n-1}/n$.
- Compare it with the total number 2^{2^n} of boolean functions on n variables.

As we saw in Definition 5, the p-SIZE of a circuit is the total number of bits to store numerators and denominators of the rational-valued weights. Let $t = 2^{n-1}/n$. What else we need to characterize a

ReLU network are: the number of neurons m, the edge set E, how many bits we allocate to each numerator and denominator, whether or not each weight is positive, and the 2^t possible values of the t bits. All the conditions above together determines a unique ReLU circuit.

Denote by C_t the number of different ReLU networks within p-SIZE t, C_t is at most

$$2^t \cdot \sum_{m=1}^{t} \sum_{i=m}^{t} 2^i \cdot \binom{\binom{m+n}{2}}{i} \cdot \binom{t+2i}{2i}$$

where i is the size of the edge set. Here i is bounded by t because each edge at least costs 1 bit (i.e., the case 0/1). And the number of neurons m is no more than t as well because otherwise the circuit is not connected.

Also, we can assume that no two neurons in the ReLU circuit computes the same function, or there will be another circuit of small size computing the same function, by eliminating repeated neurons. Thus, permuting the labels of the t neurons gives us a different description of a ReLU circuit computing the same boolean function. Therefore, we can update and bound the number of different ReLU circuits now:

$$\begin{aligned}
C_t \leq & 2^t \cdot \sum_{m=1}^{t} \sum_{i=m}^{t} \frac{2^i}{m!} \cdot \binom{\binom{m+n}{2}}{i} \cdot \binom{t+2i}{2i} \\
\leq & 2^t \cdot \sum_{m=1}^{t} \sum_{i=m}^{t} \frac{2^i \cdot e^m}{m^m} \cdot \left(\frac{e(m+n)^2}{i} \right)^i \cdot \left(\frac{e(t+2i)}{2i} \right)^{2i} \\
= & 2^t \cdot \sum_{m=1}^{t} \sum_{i=m}^{t} \frac{2^i \cdot e^{m+3i} \cdot (m+n)^{2i}}{i^i \cdot m^m} \cdot \left(1 + \frac{t}{2i} \right)^{2i} \\
\leq & (2e)^t \cdot \sum_{m=1}^{t} \sum_{i=m}^{t} \frac{2^i \cdot e^{m+3i} \cdot (m+n)^{2i}}{i^i \cdot m^m} \\
\leq & t \cdot (4e^5)^t \cdot \sum_{m=1}^{t} \frac{(m+n)^{2t}}{m^{2m}} \\
\leq & t^2 \cdot (8e^5)^t \cdot e^{2n} \cdot t^{2t}
\end{aligned}$$

As $t = 2^n/n$, we have

$$C_t \leq 2^{2^n(1-\frac{\log n}{n})+6\cdot\frac{2^n}{n}+O(n)} = o(2^{2^n})$$

which implies that there exists n-ary boolean functions that can not be computed by ReLU networks of p-SIZE at most $2^{n-1}/n$, when n is large enough. □

Therefore, the lower bound under the new size definition asymptotically matches the Shannon's $\Omega(2^n/n)$ circuit lower-bound.

6 P-uniform ReLU Circuits Decide Exactly Problems in P

A family of circuits $\{C_n : n \in \mathbb{N}\}$ is P-uniform if there is a polynomial-time Turing machine that outputs the description of C_n giving 1^n as input.

We consider only rational-valued ReLU networks (which have finite descriptions). In this case, we encode all the information of a ReLU network in binary strings. We obtain the description of the ReLU network by concatenating the binary strings together.

Theorem 3. *For any boolean language $L \subseteq \{0,1\}^n$, L is decided by a family of P-uniform ReLU networks $\{C_n\}$ if and only if $L \in P$.*

The easy direction is that P-uniform ReLU networks contains P. This follows by the Cook-Levin Theorem [Coo71], where every function in P that can be decided in time $t(n)$ has a boolean circuit of size $O(t(n)^2)$, and the fact that ReLU units can simulate the AND/OR/NOT boolean gates with a constant-size construction.

Lemma 10. *For any $L \in P$, L is decided by a family of P-uniform ReLU networks.*

Proof. By the proof of Cook-Levin Theorem, for any language $L \in$ P there exists a P-uniform family of boolean circuits $\{C_n\}$ deciding L and a Turing machine M printing $\{C_n\}$. Then, by Lemma 8 there exists another polynomial-time Turing machine M' translating any boolean circuit C_n to a ReLU circuit C'_n whose size is linear in C_n. By putting together M and M' we can get a polynomial-time Turing machine outputs the description of ReLU circuits $\{C'_n\}$ deciding L. □

To prove the remaining direction, it is sufficient to bound the weights and values, and show that all the arithmetic can be done in polynomial-time.

For any circuit C_n running on graph $G = (V, E)$, define p_e/q_e to be the parameter of an edge e, p'_x/q'_x to be the value of vertex x, where all the fractions are irreducible.

We first bound the length of the intermediate invariables p'_xs and q'_xs.

Lemma 11. *For every ReLU network C_n, the value of every vertex x has $q'_x \big| \prod_{e \in E} q_e$.*

Proof. Define

$$\text{conn}(x) =_{\text{def}} \{e | \exists \text{ path whose first edge is } e \text{ to vertex } x\}$$
$$\text{prev}(x) =_{\text{def}} \{y | (y, x) \in E\}$$

We prove a stronger proposition:

$$q'_x \Big| \prod_{e \in \text{conn}(x)} q_e$$

by induction on the depth of x.

For the base case, x is a source and $q'_x = 1 \big| 1$.

Suppose for all the vertices y of depth less than i it holds: $q'_y \big| \prod_{e \in \text{conn}(y)} q_e$. For any vertex x at depth i, we know that

$$\text{val}(x) = \frac{p'_x}{q'_x} = \sum_{y \in \text{prev(x)}} w_{(y,x)} \cdot \text{val}(y) = \sum_{y \in \text{prev(x)}} \frac{p_{(y,x)} \cdot p'_y}{q_{(y,x)} \cdot q'_y}$$

Now, we bound q'_x.

$$
\begin{aligned}
q'_x \;\Big|\; & \mathrm{LCM}_{y\in\mathrm{prev}(x)}\left(q_{(y,x)}\cdot q'_y\right)\\
\Big|\; & \mathrm{LCM}_{y\in\mathrm{prev}(x)}\left(q_{(y,x)}\cdot \prod_{e\in\mathrm{conn}(y)} q_e\right)\\
\Big|\; & \mathrm{LCM}_{y\in\mathrm{prev}(x)}\left(q_{(y,x)}\cdot \prod_{e\in\mathrm{conn}(x)\wedge e\notin\{(z,x)\}} q_e\right)\\
\Big|\; & \left(\prod_{e\in\mathrm{conn}(x)\wedge e\notin\{(z,x)\}} q_e\right)\cdot \mathrm{LCM}_{y\in\mathrm{prev}(x)}\left(q_{(y,x)}\right)\\
\Big|\; & \prod_{e\in\mathrm{conn}(x)} q_e
\end{aligned}
$$

□

Lemma 11 implies that the length of q'_x will not be longer than the description of the ReLU network, as q_es are all included in the description.

To bound p'_x, it is equivalent to bound $\mathrm{val}(x)$, while $p'_x = q'_x \cdot \mathrm{val}(x)$.

Lemma 12. *For a ReLU circuit C_n whose length of description is T, the value of any neuron x has*

$$|val(x)| \le T^T \cdot (2^T)^T$$

Proof. Still prove it by induction on depth, that is, for any vertex x at depth i we have $|\mathrm{val}(x)| \le T^i \cdot (2^T)^i$.

For the base case where x is a source, obviously we have

$$\left|\mathrm{val}(x)\right| \le 1 = T^0 \cdot (2^T)^0$$

By the inductive hypothesis, all the depth-$(i-1)$ vertices y have $|\mathrm{val}(y)| \le T^{i-1}\cdot(2^T)^{i-1}$. Since all the weights have $|w_e| \le 2^T$, and the size of edge set $|E| \le T$, we have

$$
\begin{aligned}
\left|\mathrm{val}(x)\right| &= \left|\sum_{y\in\mathrm{prev}(x)} w_{(y,x)}\cdot \mathrm{val}(y)\right|\\
&\le \sum_{y\in\mathrm{prev}(x)} 2^T \cdot T^{i-1}\cdot (2^T)^{i-1}\\
&\le T^i \cdot (2^T)^i
\end{aligned}
$$

□

As a result, we get

$$\left|p'_x\right| = \left|q'_x \cdot \mathrm{val}(x)\right| \le \left(\prod_e q_e\right)\cdot T^T \cdot \left(2^T\right)^T \tag{1}$$

Now, we continue with the rest of the proof of Theorem 3.

Proof of Theorem 3. For any boolean language L decided by a P-uniform family of ReLU circuits $\{C_n\}$, there exists a polynomial-time Turing machine M printing $\{C_n\}$. Suppose the running time of M is $T(n)$ polynomial in n, then we know that the length of the description is bounded by $T(n)$. We have the length of all the parameters

$$\sum_{e\in E}\big(\log(|p_e|+1)+\log(|q_e|+1)\big)\le T(n)$$

which means that

$$\begin{aligned}\prod_{e\in E} q_e &= 2^{\Sigma_{e\in E}\log_2 q_e}\\ &\le 2^{\Sigma_{e\in E}\left(\log_2(|p_e|+1)+\log_2(|q_e|+1)\right)}\\ &\le 2^{T(n)}\end{aligned}\tag{2}$$

To show how to simulate C_n, we first we give the procedure computing $\mathrm{val}(x)=\frac{p'_x}{q'_x}=\sum_{y\in\mathrm{prev}(x)}\mathrm{val}(y)\cdot w_{(y,x)}$:

- For each $y\in\mathrm{prev}(x)$, respectively multiply p'_y and $p_{(y,x)}$, q'_y and $q_{(y,x)}$, then $\mathrm{val}(y)\cdot w_{(y,x)}=\frac{p'_y\cdot p_{(y,x)}}{q'_y\cdot q_{(y,x)}}$.
- Amplify each fraction $\frac{p'_y\cdot p_{(y,x)}}{q'_y\cdot q_{(y,x)}}$ to $\frac{p''_{(y,x)}}{\prod_{e\in E}q_e}$, where $p''_{(y,x)}=\frac{p'_y\cdot p_{(y,x)}}{q'_y\cdot q_{(y,x)}}\cdot\prod_{e\in E}q_e$ is an integer.
- Add all the $p''_{(y,x)}$s together, we get $\mathrm{val}(x)=\frac{\sum_y p''_{(y,x)}}{\prod_{e\in E}q_e}$.
- Run Euclid's algorithm on $\mathrm{val}(x)$ to its irreducible form $\mathrm{val}(x)=\frac{p'_x}{q'_x}$.

It remains to show that the running time of the above procedure is polynomial in $T(n)$. Equivalently, we bound the length of the numbers involved instead, since integer multiplication, division, and addition, and Euclid's algorithm are all running in time polynomial to the length of the numbers:

- For $\prod_{e\in E}q_e$, by (2),
$$\log(\prod_{e\in E}q_e)\le T(n)$$
- For q'_x, by Lemma 11,
$$\log_2(|q'_x|)\le\log(\prod_{e\in E}q_e)\le T(n)$$
- For p'_x, by (1),
$$\log(|p'_x|+1)\le T(n)\log(T(n))+T(n)^2+T(n)+1$$
- For $p_{(y,x)}$ and $q_{(y,x)}$, they are included in the description and of course have length no more than $T(n)$.

Thus we can perform each ReLU neuron in $\mathrm{poly}(T(n))$ time. Whilst the circuit has size at most $T(n)$, there exists a Turing machine M' simulating circuit C_n in time $\mathrm{poly}(T(n))$. By concatenating M and M' we get a Turing machine deciding L in polynomial time. □

Remark. In the real world, we usually use one or several neural networks rather than "an infinite family of" neural networks to solve a problem. In this case, Theorem 3 states that, the running time executing a trained ReLU network is polynomial in the length of its description. In this sense, Theorem 1 does not affect practical, polynomial time computation. However, every theoretical work that proves lower bounds on the size of NNs should carefully consider the complexity parameter studied in this work.

Acknowledgements

We would like to thank Eric Allender and Ryan Williams for their remarks and for bringing to our attention related previous work.

References

[ABKPM09] Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009.

[ADH+19] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems*, 32, 2019.

[BMS85] Alberto Bertoni, Giancarlo Mauri, and Nicoletta Sabadini. Simulations among classes of random access machines and equivalence among numbers succinctly represented. *Ann. Discrete Math.*, 25:65–90, 1985.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[Dan96] Vlado Dančík. Complexity of boolean functions over bases with unbounded fan-in gates. *Information processing letters*, 57(1):31–34, 1996.

[HS74] Juris Hartmanis and Janos Simon. On the power of multiplciation in random-access machines. In *Proc. 15th Annu. IEEE Sympos. Switching Automata Theory*, pages 13–23, 1974.

[Lup58] Oleg Borisovich Lupanov. The synthesis of contact circuits. In *Doklady Akademii Nauk*, volume 119, pages 23–26. Russian Academy of Sciences, 1958.

[O1] Joseph O'Rourke. Advanced problem 6369. *Amer. Math. Monthly*, 88(10):769, 1981.

[RS42] John Riordan and Claude E Shannon. The number of two-terminal series-parallel networks. *Journal of Mathematics and Physics*, 21(1-4):83–93, 1942.

[Sch79] Arnold Schönhage. On the power of random access machines. In *Proc. 6th Internat. Colloq. Automata Lang. Program.*, volume 71 of *Lecture Notes Comput. Sci.*, pages 520–529. Springer-Verlag, 1979.

[Sha49] Claude E Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949.

[TV08] Sergey P Tarasov and Mikhail N Vyalyi. Semidefinite programming and arithmetic circuit evaluation. *Discrete Applied Mathematics*, 156(11):2070–2078, 2008.

[vE90] Peter van Emde Boas. Machine models and simulation. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 1–66. Elsevier, Amsterdam, 1990.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il