The Effect of Weight Precision on the Neuron Count in Deep ReLU Networks

Songhua He [*1] **Periklis A. Papakonstantinou** [*2]

Abstract

Deep neural networks (DNNs) have become pivotal in machine learning, but the impact of weight precision, such as in networks with rectified linear units (ReLU), remains underexplored. We analytically investigate the interplay of three key factors: the precision of ReLU network weights, the number of neurons, and the time of the preprocessing algorithm that generates the network description. Our study, which, to the best of our knowledge, is the first formal work on weight precision, yields three main results.

(1) We present an exponential time preprocessing algorithm that showcases the possibility of trading ReLU nodes for weight precision. Specifically, our method achieves an exponential reduction in neuron count when computing any function of high complexity with boolean input encoding.

What is the implication of the above result in theoretical and practical works?

(2) In theory of computing, in general, there is no free lunch. In our case, if you significantly reduce the number of neurons then you should pay the cost in weight precision. To address this, we introduce a notion of network size that considers weight precision in addition to the network's number of neurons. We establish that under this redefined notion of network size, it is generally impossible to exchange neurons for weight precision in ReLU networks of the same (redefined) size.

(3) In practice, we show that high weight precision alone cannot help in reducing the neuron count. If instead of our exponential time preprocessing algorithm one uses any polynomial time algorithm, then it is impossible to non-trivially reduce the neuron count, regardless of the high weight precision.

[*]Equal contribution [1]Department of Computer Science, Rutgers University, New Brunswick, NJ, USA [2]Department of Management Science and Information Systems, Rutgers University, New Brunswick, NJ, USA. Correspondence to: Songhua He <songhua0110@gmail.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Deep Neural Networks (DNNs) have been intensively studied and, in the last decade, have become widely popular. In this work, we revisit the resources a DNN is using. We show that the standard notion of network size, which is the number of neurons in a ReLU network, allows networks to have unreasonably high computational power using only a small number of neurons and high weight precision. This means that in addition to the number of neurons, one should take into account the numerical precision of the weights of the network.

A ReLU network is an edge-weighted graph whose nodes (neurons) are ReLU functions ($\sigma(x) = x, x > 0$ and $\sigma(x) = 0, x \leq 0$) evaluated over the weighted sum of their inputs. If we view it 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. In the real world, all real-valued problems are compiled in binary. This means that for every ReLU network with real-weights there exists an equivalent ReLU network with boolean inputs. Thus, it formally suffices to study the computational capabilities of neural networks on boolean problems.

Prior to our work, there were many excellent works that studied the computational power of various neural networks architectures both theoretically, e.g. (Siegelmann & Sontag, 1992; 1994; Siegelmann, 1999; Chen et al., 2018; Pérez et al., 2019) and empirically in finite precision settings, e.g. (Weiss et al., 2018). To the best of our knowledge, our work is the first to consider formally and in a comprehensive fashion the computational power of neural networks in terms of the precision of their weights.

Allowing high precision or equivalently large-number arithmetic often makes a computational model unreasonably

strong. Such unreasonably powerful models charge a single step per operation; for example, integer addition on large numbers. In particular, the unit-cost RAM model (Random Access Machine) can solve NP-hard and even PSPACE-hard problems in polynomial time (Schönhage, 1979); see also (Hartmanis & Simon, 1974; Bertoni et al., 1985; van Emde Boas, 1990). Similarly, for the notoriously difficult 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 (Tarasov & Vyalyi, 2008; Allender et al., 2009). 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" problem 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 (O'Rourke, 1981). As of now, this problem is not even known to be in NP.

In the standard Neural Networks model, ReLU nodes can possess any real-number weights. We investigate the consequences when these weights have high but finite precision or can be large integers. Theorem 1 (below) suggests that this adaptation confers unreasonable power to the model. Interestingly, this holds true even for inputs consisting of small numbers, such as binary input vectors. Compared to theorems about unit-cost RAM models, Theorem 1 has a significantly different statement and proof but conveys a narrative of similar flavor.

Theorem 1 (informal). *Every hard boolean function can be computed in the standard model of ReLU networks (i.e. with unbounded weights) using a number of nodes (or edges), which is exponentially smaller than the size of the smallest combinatorial circuit (or ReLU network with bounded weights) for this function. Note that this exponentially smaller ReLU network is generated by an algorithm that runs in exponential time.*

To put this result in the proper context, we recall that Shannon showed (Shannon, 1949) that almost every boolean function requires a boolean circuit of size $\Omega(2^n/n)$ (also see (Riordan & Shannon, 1942)). The $2^n/n$ bound holds for circuits of constant fan-in. To provide a meaningful comparison between ReLU networks and combinatorial circuits, we will also consider ReLU networks with constant fan-in. Shannon's lower bound is asymptotically tight, as Lupanov (Rus, 1958) showed that every boolean function can be computed in size $O(2^n/n)$. Later, when we formally restate Theorem 1, we will see that we can compute every boolean function using a number of network edges smaller than Shannon's bound. Specifically, we show that every boolean function can be computed in size $O(\sqrt{2^n}) = o(2^n/n)$ with bounded fan-in. Going exponentially below the lower bound encapsulates the meaning of the phrase "the model has unreasonable computing power." Our proof leverages a somewhat similar idea to the construction in (Dančík, 1996). However, unlike (Dančík, 1996), we do not blow up exponentially the fan-in, which is a critical step in (Dančík, 1996). We also note that our construction of ReLU networks are as deep as its size (i.e., of depth $O(\sqrt{2^n})$). We do not know if there is an algorithm that can trade precision for a similar neuron count in a shallow network.

This discussion does not mean that NNs are super-powerful in practice. The reason is that the algorithm that generates the network description runs in time exponential in the input size of the network. However, this result shows that in a NN it is possible to trade the number of nodes (or edges) for the precision of weights. Therefore, the number of the ReLU nodes/edges alone is insufficient in describing the complexity of the NN. To that end, we introduce a natural definition of *size*, which takes into account the precision and the magnitude of NN weights. Under this new "weight-sensitive ReLU size" definition we show the following.

Theorem 2 (informal). *The model of ReLU networks under our "weight-sensitive ReLU size" has the same computing power as standard combinatorial circuits of the same size.*

We formalize the above result by showing a variant of Shannon's lower bound for ReLU networks where "size" is our "precision-sensitive size". In the variant of Shannon's theorem, we obtain asymptotically the same result as for combinatorial circuits (but, in our case "size" is something different).

Theorem 3 (informal). *The set of all problems computable in polynomial time is the set of problems computable by ReLU networks (with unbounded weights) whose description is generated by an algorithm that runs in polynomial time.*

Theorem 3 complements Theorem 1. Theorem 1 tells us that the weight precision matters. Theorem 3 tells us that the weight precision does not matter if one invests only polynomial time to output the network description. In other words, if the algorithm that outputs the network description does not use too much time, then there is no advantage in utilizing high-precision weights.

We note that an analogous phenomenon to Theorem 3 is not true for other high-precision models of computation. For example, consider circuits with real-valued gates doing the arithmetic of the unit-cost RAM model. The algorithm that outputs the circuit description is efficient (polynomial time), but the circuit it outputs is computationally extremely powerful; it can compute the entire PSPACE with only a polynomial number of gates.

We conclude the introduction with two remarks regarding our results in the context of machine learning.

First, we do not know how a training algorithm (learning algorithm) could utilize high-precision weights to give more-than-usual power to a learned ReLU model. For example, we do not know how one can devise a training algorithm for neural networks, which for a fixed number of ReLU nodes the more it trains the more it can take advantage of high precision weights. Exploring different ways to update the weights (other than this, using the same training algorithm) could lead to new insights in the practice of machine learning. This seems to be an interesting question that can be studied both theoretically and, importantly, empirically.

Second, our study highlights the importance of considering weight precision in theoretical works on ReLU networks, particularly in works proving lower bounds. This consideration should extend to both the statements of such theorems and their proofs to ensure sound arguments.

The rest of the paper is organized as follows. First, we introduce the necessary notation and the new NN size definition in Section 2. We also define gadgets useful for constructing ReLU networks in Section 3. In the last three sections, we introduce and prove our three main results: we prove that ReLU networks are strictly more powerful than standard circuit models with the same number of gates (by using an exponential time algorithm that generates the network description); we show that ReLU networks under our new definition of size match Shannon's circuit lower-bound; and finally, we show that ReLU networks that are generated by a polynomial time algorithm compute exactly the polynomial-time computable problems.

2. Definitions and Notation

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. $\text{LCM}(x_1, ..., x_k)$ is the least common multiple of the integer x_is, and $\text{GCD}(x_1, ..., x_k)$ is their greatest common divisor. All logarithms are of base 2. For any two natural numbers p, q, $p|q$ indicates that p divides q.

2.1. ReLU Networks and Other Models

The formal definition of ReLU networks is as follows:

Definition 2.1 (see (Arora et al., 2019))**.** 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 $0, 1$. To differentiate with sources, non-source vertices are called neurons (nodes). The value $\text{val}(x)$ of a neuron x is given by ReLU functions:

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

where E is the edge set of the ReLU network, $w_{(y,x)} \in \mathbb{R}$ is the weight of edge (y, x), and σ is the ReLU function defined as $\sigma(x) =_{\text{def}} \max(x, 0)$. 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, the 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. We note that a boolean circuit and a ReLU network where the weights are of at most polynomial precision or magnitude are polynomially equivalent. That is, when we say that "we compare boolean circuits and ReLU networks", this is the same as saying that "we compare ReLU networks of reasonable weight-precision and ReLU networks of arbitrary precision".

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

2.2. Network Size

Prior to this work, in neural networks, the size is defined as the number of neurons (or the number of edges/connections between neurons). The number of neurons is at most the product of the network's depth and width. As for boolean circuits, the size of circuits is either defined as the number of nodes or the number of edges, where for notational convenience we choose the latter one as the definition of circuit size. For the size of boolean circuits with unbounded fan-in and ReLU networks, we make the same choice to apply Shannon's circuit lower-bound on these models.

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

Definition 2.2. 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:

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

where E is the edge set of C, and $p_{x,y}/q_{x,y} = w_{x,y}$, where $p_{x,y}$ 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 network. It is equal to the number of binary bits we need to write all the integers $(|p_{x,y}|), (|q_{x,y}|)$ for $(x, y) \in E$.

3. Gadgets for ReLU Networks

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

Lemma 3.1. *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: $y := \sigma(\frac{1}{c-c'}x - \frac{c'}{c-c'})$. y is non-zero if and only if $x > c'$, and is greater than or equal to 1 if and only if $x \geq c$. We have

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

□

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

$$[x \leq c] = [(-x) \geq (-c)]$$

$$[x = c] = [[x \geq c] + [x \leq c] \geq 2]$$

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

Fact 3.2. *For input $x, y \in \mathbb{R}_{\geq 0}$ 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 switches" the same way as above.

Lemma 3.3. *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:

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

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

□

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

Lemma 3.3 says that ReLU networks are at least as powerful as boolean circuits, while boolean circuits are composed of AND, OR, and NOT gates (nodes). 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. Let us formally restate our first result.

Theorem 4.1 (restatement of 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. Furthermore, the network description is computed in time $2^{O(n)}$.*

This upper-bound is exponentially smaller than the $\Omega(2^n/n)$ lower-bound given by Shannon for the number of gates/wires in a constant fan-in combinatorial circuit.

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 network 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 $x_1, ..., x_n \in \{0, 1\}$,

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

To formalize this argument, we first need 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 4.2. *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.*

Algorithm 1 2^n-sized network 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 + $([T_f \geq 2^k] \wedge [(x_1...x_n)_2 = k])$
  // $[T_f \geq 2^k]$ decides whether the k-th bit is on or off.
  $T_f := ([T_f \geq 2^k]\ ?\ T_f - 2^k : T_f)$
  // Remove the k-th bit.
end for
Output := Sum

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

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

$T := 0$
for $k := 0$ to $2^{\lfloor n/2 \rfloor} - 1$ **do**
  $T := T + ([(x_1...x_{\lfloor n/2 \rfloor})_2 = k]\ ?\ T_{f(k_1,...,k_{\lfloor n/2 \rfloor})} : 0)$
  // Assign the $(x_1...x_{\lfloor n/2 \rfloor})_2$-th truth table to T.
end for
Output := the $(x_{\lfloor n/2 \rfloor+1}...x_n)$-th bit of T
// This is done using Algorithm 1.

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 network. The operators involved can all be implemented in constant size by Lemmas 3.1, 3.2 and 3.3. By precomputing $(x_1x_2...x_n)_2 = \sum_{i=1}^{n} x_i \cdot 2^{n-i}$, the formula $[(x_1x_2...x_n)_2 = k]$ also costs constant size as stated in Lemma 3.1. The total size of the 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. We will come to this in the proof of Theorem 4.1, when T_f will be a variable.

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 4.2 after we partition the truth table using a different algorithmic idea.

Proof of Theorem 4.1. 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.

Once more, we give the construction as pseudocode in Algorithm 2.

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 4.2. The total size of the network is still $O(2^{n/2})$. □

Remark. Previously it was shown (Dančík, 1996) that computing any boolean function in *unbounded* fan-in boolean circuits can be done in $O(2^{n/2})$ nodes as well, which is proved by a similar construction as ours. Importantly, the construction in (Dančík, 1996) 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 4.1 shows that ReLU networks are strictly stronger than boolean circuits. In contrast, Shannon's Theorem (Shannon, 1949) staets that almost every n-ary boolean function requires boolean circuits of size $2^n/n$, for sufficiently big n.

We will discuss open questions related to Theorem 4.1 in Section 7.

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

In this section, we show in a combinatorial way that under Definition 2.2, ReLU networks have the same computational power as regular models, i.e., boolean circuits. We know from Lemma 3.3 that functions computed by boolean circuits can efficiently translate to ReLU networks. Combined with Lupanov's construction (Rus, 1958), every function can be computed by ReLU networks of p-SIZE $O(2^n/n)$. To show the other direction, we prove a matching lower bound in Theorem 5.1.

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

Theorem 5.1 (restatement of Theorem 2)**.** *For any sufficiently large n,*

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

In fact, we prove a stronger proposition: for any sufficiently large n, almost every boolean function requires ReLU networks of size greater than $2^n/n$.

Proof. We proceed with a counting argument.

- Count the number of different ReLU networks over n variables and of p-SIZE at most $2^n/n$.

- Compare it with the total number 2^{2^n} of boolean functions on n variables.

As we saw in Definition 2.2, the p-SIZE of a network is the total number of bits to store numerators and denominators of the rational-valued weights. Let $t = 2^n/n$. To fully characterize a ReLU network we need the following: 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. Together, all the conditions above determine a unique ReLU network.

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, and $\binom{t+2i}{2i}$ is the number of different ways to allocate t bits to i numerators and i denominators. Here i is bounded by t because each edge at least costs 1 bit (the lowest p-SIZE of an edge is 1, corresponds to the weight $0/1$). The number of neurons m is no more than t as well because otherwise the network is not connected.

Besides, we can assume that no two neurons in the ReLU network compute the same function, or there will be another network of smaller 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 network computing the same boolean function. Therefore, we can update and bound the number of different ReLU networks:

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

where the second inequality comes from the inequalities $m! \geq \frac{m^m}{e^m}$ and $\binom{n}{k} \leq (\frac{en}{k})^k$, the third inequality comes from $(1+\frac{1}{x})^x \leq e$, and the fifth and sixth inequality comes from the observation that $m^{2i-m} i^{-i}$ takes its maximum at $i = t$ and m somewhere in the range $(2t/\ln t, 4t/\ln t)$.

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

$$\begin{aligned}
C_t &\leq t^t \cdot (t^{-2t/\ln t + 2} \cdot (\ln t)^{2t/\ln t - 2t} \cdot 2^{8t - 2t/\ln t} \cdot e^{5t}) \\
&= 2^{2^n(1 - \log n/n)} \cdot o(1) = o(2^{2^n})
\end{aligned}$$

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

Therefore, the lower bound under our new size definition matches Shannon's $2^n/n$ boolean circuit lower bound.

6. ReLU Networks whose Description is Computed in Polynomial Time Decide Exactly Problems in Polynomial Time

We formally state and prove Theorem 3. We will show that when the description of the networks can be printed in polynomial time, then these ReLU networks can exactly compute problems in P (the set of polynomial-time computable problems). This is done by showing that we can simulate the arithmetic in executing the ReLU networks efficiently, in which the intermediate values could be exponentially large.

A family of networks $\{C_n : n \in \mathbb{N}\}$ is *polynomial time printable*[1] 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 input descriptions. In this case, we encode all the information (e.g., the connection of the network, the weights in binary) of a ReLU network in binary strings. We let the description of the ReLU network be the concatenation of the binary strings.

Theorem 6.1 (restatement of Theorem 3)**.** *For every language $L \subseteq \{0,1\}^n$, L is decided by a family of ReLU networks $\{C_n\}$ whose description is computable in polynomial time if and only if $L \in$ P.*

The easy direction is that polynomial time printable ReLU networks contain P. This follows by the Cook-Levin Theorem (Cook, 1971), 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 nodes can simulate the AND/OR/NOT boolean operators with a constant-size construction.

[1] In classical computational complexity theory we say that "the circuit is P-uniform".

To show the other direction, we first bound the magnitude of all the weights and intermediate values. Fix an arbitrary network C_n with an underlying graph $G = (V, E)$. Let $p_e/q_e = w_e$ the weight of an edge $e \in E$, and $p'_x/q'_x = \text{val}(x)$ to be the value of vertex x, where all the fractions are irreducible.

We first bound the length of the intermediate variables p'_xs (numerators) and q'_xs (denominators).

Lemma 6.2. *For every ReLU network C_n, the value of every vertex x has $q'_x \big| \prod_{e\in E} q_e$.*

Proof. Define

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

We prove a stronger proposition:

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

by induction on the depth of x.

This is to say that the denominator of the value of a vertex x can divide the product of denominators of the weights of edges above x (recall that C_n is a directed acyclic graph).

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

$$\begin{aligned}\text{val}(x) = \frac{p'_x}{q'_x} &= \sigma\left(\sum_{y\in\text{prev(x)}} w_{(y,x)} \cdot \text{val}(y)\right) \\ &= \sigma\left(\sum_{y\in\text{prev(x)}} \frac{p_{(y,x)} \cdot p'_y}{q_{(y,x)} \cdot q'_y}\right)\end{aligned}$$

Now we show that q'_x divides the product of q_e.

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

□

Lemma 6.2 implies that the length of q'_x (denominators) will not be longer than the description of the ReLU network, as q_es are all included in the description.

To bound the length of p'_x (numerators), it suffices to bound the magnitude of $\text{val}(x)$, while $p'_x = q'_x \cdot \text{val}(x)$.

Lemma 6.3. *For a ReLU network 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. We still prove it by induction on depth, that is, for any vertex x at depth i we have $|\text{val}(x)| \le T^i \cdot (2^T)^i$.

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

$$\big|\text{val}(x)\big| \le 1 = T^0 \cdot (2^T)^0$$

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

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

□

As a result, we get

$$\big|p'_x\big| = \big|q'_x \cdot \text{val}(x)\big| \le \left(\prod_{e\in E} q_e\right) \cdot T^T \cdot \left(2^T\right)^T \qquad (1)$$

Now, we are ready to prove Theorem 6.1.

Proof of Theorem 6.1. For any boolean language L decided by a polynomial time printable 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

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

Here we give the polynomial-time simulator of C_n, it goes by executing the network in topological order. For each vertex x, we compute $\text{val}(x) = \frac{p'_x}{q'_x}$ in the following procedure:

- For each $y \in \text{prev}(x)$, respectively multiply p'_y and $p_{(y,x)}$, q'_y and $q_{(y,x)}$, then $\text{val}(y)\cdot w_{(y,x)} = \frac{p'_y\cdot p_{(y,x)}}{q'_y\cdot q_{(y,x)}}$.
- 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 $\text{val}(x) = \sigma\left(\frac{\sum_y p''_{(y,x)}}{\prod_{e\in E} q_e}\right)$.
- Run Euclid's algorithm on $\text{val}(x)$ to its irreducible form $\text{val}(x) = \frac{p'_x}{q'_x}$.

Notice that we amplify the fractions in step two for the convenience of summing them up.

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, addition, and Euclid's algorithm are all running in time polynomial to the length of the numbers:

- For the length of $\prod_{e\in E} q_e$, by (2),
$$\log(\prod_{e\in E} q_e) \le T(n)$$
- For q'_x, by Lemma 6.2,
$$\log_2(|q'_x|) \le \log(\prod_{e\in E} q_e) \le T(n)$$
- 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 $T(n)$.

□

7. Open Questions

This work explores the computational capabilities of very high weight precision deep ReLU networks, as well as their limits given restrictions in the precision and/or the running time of the pre-processing algorithm. There are still important questions we have not answered.

1. Perhaps the most important open question is whether there is a training algorithm (maybe one that takes a lot of time to train), which takes advantage of higher weight precision.
2. If one focuses only on the number of neurons we do not know of any non-trivial lower bound on the neuron count when the ReLU network is of unrestricted precision. The upper bound we obtained is $O(\sqrt{2^n})$. It is interesting to close the gap. Can we obtain a better upper bound? Can we obtain any lower bound?
3. Our construction in Theorem 4.1 involves networks depth similar to the neuron count (of order $O(\sqrt{2^n})$). Is there a construction of smaller depth?
4. This work focuses on neural networks with ReLU activation, a standard activation function. It remains open if a similar result can be obtained for neural networks with other activation functions. We note that our result takes advantage of the magnitude of the weights. It is not clear if the same idea can be applied to networks with softmax or sigmoid activation.

Impact Statement

This paper aims to advance the field of Machine Learning. While there are numerous potential societal implications of our work, we do not believe any specific impact needs to be highlighted at this time.

Acknowledgements

We would like to thank Ryan Williams for the useful suggestions. We would also like to thank Chengyuan Deng for the suggestions on the write-up of the paper. Finally, we also want to thank the anonymous reviewers for their comments, which helped in improving the exposition.

References

Allender, E., Bürgisser, P., Kjeldgaard-Pedersen, J., and Miltersen, P. B. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R., and Wang, R. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems*, 32, 2019.

Bertoni, A., Mauri, G., and Sabadini, N. Simulations among classes of random access machines and equivalence among numbers succinctly represented. *Ann. Discrete Math.*, 25:65–90, 1985.

Chen, Y., Gilroy, S., Maletti, A., May, J., and Knight, K. Recurrent neural networks as weighted language recognizers. In Walker, M. A., Ji, H., and Stent, A. (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pp. 2261–2271. Association for Computational Linguistics, 2018. doi: 10.18653/v1/n18-1205. URL https://doi.org/10.18653/v1/n18-1205.

Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, 1971.

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

Hartmanis, J. and Simon, J. On the power of multiplciation in random-access machines. In *Proc. 15th Annu. IEEE Sympos. Switching Automata Theory*, pp. 13–23, 1974.

O'Rourke, J. Advanced problem 6369. *Amer. Math. Monthly*, 88(10):769, 1981.

Pérez, J., Marinković, J., and Barceló, P. On the turing completeness of modern neural network architectures. In *ICLR*, 2019.

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

The synthesis of contact circuits, volume 119, 1958. Russian Academy of Sciences.

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

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

Siegelmann, H. T. *Neural networks and analog computation - beyond the Turing limit*. Progress in theoretical computer science. Birkhäuser, 1999. ISBN 978-0-8176-3949-5.

Siegelmann, H. T. and Sontag, E. D. On the computational power of neural nets. In Haussler, D. (ed.), *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992*, pp. 440–449. ACM, 1992. doi: 10.1145/130385.130432. URL https://doi.org/10.1145/130385.130432.

Siegelmann, H. T. and Sontag, E. D. Analog computation via neural networks. *Theor. Comput. Sci.*, 131(2):331–360, 1994. doi: 10.1016/0304-3975(94)90178-3. URL https://doi.org/10.1016/0304-3975(94)90178-3.

Tarasov, S. P. and Vyalyi, M. N. Semidefinite programming and arithmetic circuit evaluation. *Discrete Applied Mathematics*, 156(11):2070–2078, 2008.

van Emde Boas, P. Machine models and simulation. In van Leeuwen, J. (ed.), *Handbook of Theoretical Computer Science*, volume A, pp. 1–66. Elsevier, Amsterdam, 1990.

Weiss, G., Goldberg, Y., and Yahav, E. On the practical computational power of finite precision rnns for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 740–745, 2018.