The Power of Max Pooling Layer

Jiří Šíma[1[0000−0001−8248−9425]] (✉) and Jérémie Cabessa[1,2[0000−0002−5394−5249]]

[1] Institute of Computer Science of the Czech Academy of Sciences, Prague, Czechia
sima@cs.cas.cz
[2] DAVID Laboratory, UVSQ – University Paris-Saclay, Versailles, France
jeremie.cabessa@uvsq.fr

Abstract. Max pooling layers are the basic building blocks of convolutional neural networks. The theoretical characterization of their computational power is therefore a question of central interest. This paper deals with the representability of the max pooling layer by neural networks (NNs) employing the ReLU activation function. We provide two upper bounds on the size (number of ReLU neurons) and depth (number of layers) of the NNs that implement the maximum MAX_n of n nonnegative numbers. We show that the MAX_n function can be computed either by a NN of size n and logarithmic depth, or by a NN of quadratic size and constant depth for bounded input numbers of limited precision, where the constant depth depends on the magnitude of the weights. As a lower bound, we prove that no NN of depth 2 can compute the maximum of more than two nonnegative numbers. This confirms that the max pooling layer cannot be replaced by just two convolutional layers.

Keywords: Convolutional neural network · Max pooling · ReLU · Computational power

1 Introduction

Convolutional neural networks (CNNs) have revolutionized the field of computer vision and have since been applied to various other domains [8]. A critical component within these architectures are pooling layers that progressively downsample the spatial dimensions of feature maps. The pooling layers primarily enhance efficiency in terms of computation and memory, improve the network's robustness to variations, distortions and noise in input data, remove redundant information, and mitigate overfitting [3]. Among various pooling strategies, max pooling is the most widely used. The practical effectiveness and theoretical benefits of max pooling have been investigated in numerous studies, e.g. [11, 4, 9].

A natural question arises whether pooling layers can be expressed using the same unified mathematical formulation as ReLU neurons in convolutional layers (with fully connected layers as a special case) [9]. Such a formulation would make the computation of CNNs as well as their theoretical analysis more uniform. For average pooling, this is straightforward: the average of n real-valued inputs can be computed as a weighted sum, with each input assigned a weight of $1/n$. In this paper, we focus on whether max pooling layers can be expressed similarly.

Specifically, we investigate whether the maximum MAX_n of n *nonnegative* real numbers can be computed by a neural network (NN) that employs the ReLU (rectified linear unit) activation function. A formal definition of NNs which allow layer-skipping connections [12], is introduced Section 2. Note that the inputs to max pooling in CNNs are nonnegative reals since they represent the outputs from the previous convolutional layer produced by the nonnegative ReLU function.

We present two upper bounds on the depth (number of layers) and size (number of ReLU neurons) of the NNs that implement MAX_n (Section 3), providing a basic estimate of the computational complexity of max pooling. We show that MAX_n can be computed either by a NN of size n and depth $\lceil \log_2 n \rceil + 1$ (Theorem 1), or by a NN of quadratic size and constant depth for bounded input numbers of limited precision (Theorem 2), where the constant depth depends on the magnitude of the weights. By using Theorem 2, we present examples of NNs that compute MAX_n with decreasing weights for different b-bit unsigned integer and floating-point data types (specified in the IEEE 754 standard). These constructions have already been employed in the practical AppMax method of estimating the error of energy-efficient CNNs (with reduced bitwidth) based on linear programming where max pooling layers are replaced by convolutional ones [15]. As a lower bound (Section 4), we prove that no NN of depth 2 can compute the maximum of more than two nonnegative numbers (Theorem 3). This confirms that the max pooling layer cannot be replaced even by two convolutional layers.

Recent work has extensively studied the expressivity and approximation power of deep neural networks with ReLU activations (DNNs), both in theory [1, 2, 7, 9, 14] and in practical settings with floating-point arithmetic [13]. Arora et al. [1] showed that DNNs which compute exactly continuous piecewise linear functions, can represent any such function on $\mathbb{R}^n$, including MAX_n extended to negative inputs, using a DNN of depth $\lceil \log_2(n+1) \rceil + 1$. In Theorem 1, we derive a similar upper bound using a compact formula for MAX_2 that involves only one ReLU, improving formally previous constructions [1, Lemma D.3], [7, Theorem 1.1], and [9, Theorem 2], by reducing the required network size from $4(2n-1)$ to n for nonnegative inputs and with skip-layer connections.

Hertrich et al. [7] conjecture that the classes of functions computable by neural networks (NNs) for increasing number of layers, form a strict hierarchy up to the mentioned logarithmic upper bound which would thus be tight. They show this conjecture to be equivalent to the logarithmic lower bound on the depth of NNs that has to be greater than $k+1$ to implement the maximum $\max(x_1, \ldots, x_n, 0)$ of $n = 2^k$ real numbers $x_1, \ldots, x_n$ (including negative ones) and zero. This lower bound is known only for $k = 1$ [10] (i.e. $\max(x_1, x_2, 0)$ cannot be computed in depth 2) which is closely related to our Theorem 3 providing a lower bound for MAX_3 restricted to three *nonnegative* reals (more relevant to CNNs) and thus representing a stronger result. The lower bound was improved to $k = 2$ (i.e. $\max(x_1, x_2, x_3, x_4, 0)$ cannot be implemented in depth 3) by a computer-based proof [7], but only for the so-called H-conforming NNs whose neurons realize linear functions when restricted to input numbers $x_1, x_2, x_3, x_4, 0$

with a fixed ordering. This result for H-conforming NNs was achieved also by a combinatorial proof not requiring excessive computational verification and further strengthened to nonconstant depth $\Omega(\log\log n)$ [5]. Finally, the matching lower bound $\lceil \log_2(n+1) \rceil + 1$ was achieved for integer weights [6] whereas the depth $\lceil \log_3(n+1) \rceil + 1$ is required for NNs whose weights are decimal fractions [2].

2 A Formal Model of NNs with the Rectifier

In this section, we define a formal model of (artificial) feedforward neural networks (NNs) with the ReLU (rectified linear unit) activation function. From a formal point of view, this model covers any CNN without max pooling layers, which consists of ReLU neurons.

The architecture of a NN $\mathcal{N}$ is thus a general connected directed acyclic graph (V, E), where V is a set of computational units, called (ReLU) neurons, and $E \subset V \times V$ is a set connections (edges). The set V can be partitioned in a unique minimal way into a sequence of $d+1$ pairwise disjoint layers counted from zero so that neurons in any layer are connected only to neurons in subsequent layers, where d is the *depth* of $\mathcal{N}$. The zeroth input layer $X = \{1, \ldots, n\} \subset V$ is composed of $n = |X|$ input neurons which are not counted in the *size* $s = |V \setminus X|$ of $\mathcal{N}$. For our purposes of computing the maximum of inputs we assume that the dth output layer contains only one output neuron $m \in V$. Each connection $(i, j) \in E$ is labeled with a real weight parameter $w_{ji} \in \mathbb{R}$ and each neuron $j \in V$ is associated with a real bias $w_{j0} \in \mathbb{R}$. We define the *weight* of $\mathcal{N}$ as $\max_{(i,j)\in E} |w_{ji}|$. For any neuron $j \in V$, we denote by $j_{\leftarrow} = \{i \in V \mid (i, j) \in E\}$ the set of units in $\mathcal{N}$ from which connections lead to j, which represent the inputs to j. Thus, $j_{\leftarrow} = \emptyset$ for every $j \in X$, and $m \notin j_{\leftarrow}$ for all $j \in V$.

The NN $\mathcal{N}$ computes a function $\mathcal{N} : \mathbb{R}^n_{\geq 0} \to \mathbb{R}_{\geq 0}$ from a set of n-tuples of nonnegative real numbers to nonnegative reals by evaluating the nonnegative real states (outputs) $y_j \in \mathbb{R}_{\geq 0}$ of neurons $j \in V$ layer by layer. At the beginning, the argument $(x_1, \ldots, x_n) \in \mathbb{R}^n_{\geq 0}$ is presented as an external input to $\mathcal{N}$ via its input neurons, that is, $y_j = x_j$ for every $j \in X$. The real excitation $\xi_j \in \mathbb{R}$ of a non-input neuron $j \in V \setminus X$ in the next layer is computed as a weighted sum of its inputs including its bias:

$$\xi_j = w_{j0} + \sum_{i \in j_{\leftarrow}} w_{ji} y_i \,, \tag{1}$$

provided that the states y_i have already been determined for all units $i \in j_{\leftarrow}$ in previous layers. Then the output y_j from neuron j is computed by applying the ReLU activation function (rectifier) $R : \mathbb{R} \to \mathbb{R}$ to its excitation ξ_j,

$$y_j = R(\xi_j) = \max(0, \xi_j) \,. \tag{2}$$

By the connectivity of $\mathcal{N}$, the state y_m of the output neuron $m \in V$ is eventually evaluated, which provides the output $y_m = \mathcal{N}(x_1, \ldots, x_n)$ from $\mathcal{N}$, for the input $(x_1, \ldots, x_n)$.

3 Upper Bounds

In the following theorem, we show a simple construction of a NN that computes the MAX_n function returning the maximum of n nonnegative real numbers. It is based on a observation that the MAX_2 function can be easily calculated by ReLU neurons, which is recursively used to implement MAX_4, MAX_8, MAX_{16}, etc. This results in a logarithmic number of layers and linear number of neurons having unit weights, which provides a basic upper bound on the computational complexity of max pooling.

Theorem 1. *The maximum of $n \geq 2$ nonnegative real numbers $x_1, \ldots, x_n \in \mathbb{R}_{\geq 0}$ can be computed by a unit-weight NN of size n and depth $\lceil \log_2 n \rceil + 1$.*

Proof. For simplicity of notation, assume that $n = 2^k$ is the power of two for some integer $k \geq 1$, while the argument is similar if it is not the case. The idea of the proof is based on the fact that the maximum of two nonnegative real numbers $x, y \geq 0$ can be simply calculated by the ReLU function R as

$$\max(x, y) = R(x - y) + y \tag{3}$$

according to (2). This is used in the NN $\mathcal{N}$ to compute the maximum of n nonnegative reals $x_1, \ldots, x_n \geq 0$, which form the leaves of a binary tree of logarithmic depth. Its internal nodes recursively determine the maximum of two numbers by (3) over pairs, fours, eights, etc., as illustrated in Figure 1 for $n = 8$. We also provide a precise formal specification of $\mathcal{N}$, including complex index formulas suitable for computer implementation [15]. Let $x_{0i} = x_i$ for $i \in \{1, \ldots, n\}$ and

$$x_{\ell i} = \max\left(x_{\ell-1,2i-1}, x_{\ell-1,2i}\right) \text{ for } \ell \in \{1, \ldots, k\},\ i \in \left\{1, \ldots, \tfrac{n}{2^\ell}\right\} \tag{4}$$

which ensures $x_{k1} = \max(x_1, \ldots, x_n)$. By plugging (3) into (4), we obtain $y_i = y_{0i} = x_i$ for $i \in X = \{1, \ldots, n\}$,

$$y_{\ell i} = R\left(\sum_{j=0}^{\ell-1} \left(y_{j,(2i-1)2^{\ell-j-1}} - y_{j,i2^{\ell-j}}\right)\right) \text{ for } \ell \in \{1, \ldots, k\},\ i \in \left\{1, \ldots, \tfrac{n}{2^\ell}\right\}, \tag{5}$$

$$\text{and} \quad y_m = \sum_{j=0}^{k} y_{j,2^{k-j}} = R\left(\sum_{j=0}^{k} y_{j,2^{k-j}}\right) = \max(x_1, \ldots, x_n)\,. \tag{6}$$

For each $\ell \in \{0, \ldots, k\}$, this establishes $\frac{n}{2^\ell}$ neurons $(\ell, i) \in V$ in the ℓth layer of $\mathcal{N}$ for $i \in \{1, \ldots, \frac{n}{2^\ell}\}$, and one output neuron $m \in V$, which provides the maximum $y_m = \mathcal{N}(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n)$ according to (6). Clearly, the size of unit-weight $\mathcal{N}$ is $\sum_{\ell=1}^{k} \frac{n}{2^\ell} + 1 = n$ (excluding the inputs) and its depth is $k + 1 = \log_2 n + 1$, which completes the proof. □

The main drawback of the NN in Theorem 1 is its logarithmic depth and sparse connections, making it inefficient for evaluation, e.g. via matrix operations. A natural question is whether the depth can be reduced to a constant

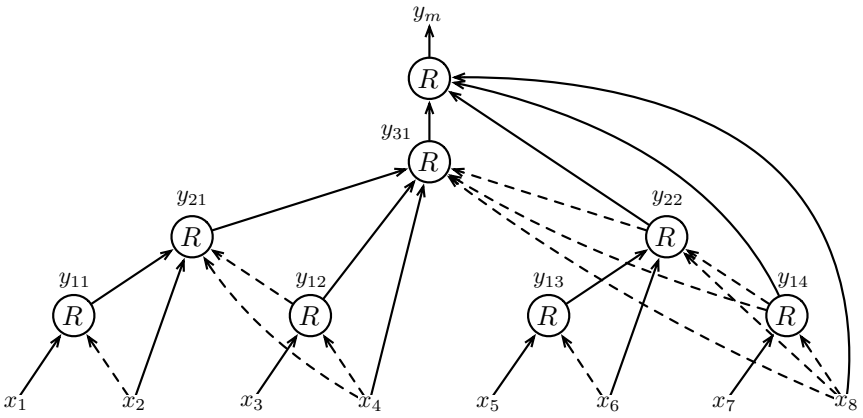

Fig. 1. An example of a depth-4 NN of 8 (non-input) neurons, that computes the maximum of $n = 8$ nonnegative real numbers according to Theorem 1, where the solid and dashed arrows depict the connections with the weights 1 and -1, respectively.

number of layers. The following theorem shows this is indeed possible with a quadratic number of neurons, but only under an additional assumption that the maximum value and the gap between the largest and second largest numbers are bounded. These bounds then control the tradeoff between the depth and weight of NNs implementing the MAX_n function.

Theorem 2. *Let $x_1, \ldots, x_n \in \mathbb{R}_{\geq 0}$ be $n \geq 2$ nonnegative real numbers. Denote by $\mu_1 = \max\{x_1, \ldots, x_n\}$ and $\mu_2 = \max\big(\{0\} \cup (\{x_1, \ldots, x_n\} \setminus \{\mu_1\})\big)$ their largest and second largest (or zero) values, respectively. Then for any integer $r \geq 0$, the maximum μ_1 of $x_1, \ldots, x_n$ can be computed by a NN $\mathcal{N}_r$ of size $rn^2 + n + 1$, depth $2r + 2$, and weight $\max(1, \sqrt{w})$ such that*

$$(w+1)^r \geq \frac{\mu_1}{\mu_1 - \mu_2} \quad \text{if } \mu_1 > 0\,, \quad \text{or } w = 1 \text{ otherwise.} \tag{7}$$

Proof. Let $y_i = y_{0i} = x_i$ for $i \in X = \{1, \ldots, n\}$,

$$y_{\ell j} = R(\xi_{\ell j}) \quad \text{where} \quad \xi_{\ell j} = y_{\ell-1,j} - \sqrt{w} \sum_{i \in X \setminus \{j\}} R\big(\sqrt{w}\,(y_{\ell-1,i} - y_{\ell-1,j})\big)\,, \tag{8}$$

for every $\ell \in \{1, \ldots, r\}$ and $j \in \{1, \ldots, n\}$, and

$$y_m = \sum_{k=1}^{n} R\left(y_{rk} - \sum_{j=1}^{k-1} y_{rj}\right)\,. \tag{9}$$

We will show that the formulas (8) and (9) including the rectifier (2), implement the maximum $y_m = \max(x_1, \ldots, x_n)$.

We first prove that

$$\mu_1 \in \{y_{r1}, \ldots, y_{rn}\} \subseteq \{0, \mu_1\}\,. \tag{10}$$

For $\mu_1 = 0$, we have $\{x_1, \ldots, x_n\} = \{0\}$ implying $\{y_{r1}, \ldots, y_{rn}\} = \{0\}$ by (8), which ensures (10). Thus, let $\mu_1 > 0$. For $r = 0$, condition (7) entails $\mu_2 = 0$, which proves (10). Moreover, let $r \geq 1$. For notational simplicity, we assume the index notation that meets

$$\mu_1 = x_n = x_{n-1} = \cdots = x_{j_0} > \mu_2 = x_{j_0-1} \geq \cdots \geq x_1 \geq 0 \tag{11}$$

for some $j_0 \in \{2, \ldots, n\}$, without loss of generality as the following argument for (10) holds for any ordering of the inputs. By induction on ℓ, we extend the assumption (11) to

$$\mu_1 = y_{\ell n} = y_{\ell,n-1} = \cdots = y_{\ell,j_0} > y_{\ell,j_0-1} \geq \cdots \geq y_{\ell 1} \geq 0\,. \tag{12}$$

for every $\ell \in \{0, \ldots, r\}$. Clearly, (11) implies (12) for $\ell = 0$.

Further assume that (12) holds for $\ell - 1$ where $0 < \ell \leq r$, that is,

$$\mu_1 = y_{\ell-1,n} = y_{\ell-1,n-1} = \cdots = y_{\ell-1,j_0} > y_{\ell-1,j_0-1} \geq \cdots \geq y_{\ell-1,1} \geq 0\,. \tag{13}$$

Then equation in (8) reduces to

$$\xi_{\ell j} = (w(n-j)+1)y_{\ell-1,j} - w \sum_{i=j+1}^{n} y_{\ell-1,i} \quad \text{for } j \in \{1, \ldots, n\}\,, \tag{14}$$

since

$$R(\sqrt{w}\,(y_{\ell-1,i} - y_{\ell-1,j})) = \begin{cases} 0 & \text{for } i \in \{1, \ldots, j-1\} \\ \sqrt{w}\,(y_{\ell-1,i} - y_{\ell-1,j}) & \text{for } i \in \{j+1, \ldots, n\} \end{cases} \tag{15}$$

by induction hypothesis (13) according to (2). It follows from (14) and (13) that $\xi_{\ell j} = (w(n-j)+1)\mu_1 - w(n-j)\mu_1 = \mu_1$ for $j \in \{j_0, j_0+1, \ldots, n\}$ and

$$\xi_{\ell j} - \xi_{\ell,j-1} = (w(n-j+1)+1)(y_{\ell-1,j} - y_{\ell-1,j-1}) \geq 0 \quad \text{for } j \in \{2, \ldots, n\}\,, \tag{16}$$

which implies that (12) holds for ℓ, due to the rectifier (2) is nondecreasing. This completes the induction.

It follows from (16) for $j_0 \leq n$ that for each $\ell \in \{1, \ldots, r\}$,

$$\xi_{\ell j_0} - \xi_{\ell,j_0-1} \geq (w+1)\,(y_{\ell-1,j_0} - y_{\ell-1,j_0-1}) \tag{17}$$

which reduces to

$$\xi_{\ell,j_0-1} \leq (w+1)\,y_{\ell-1,j_0-1} - w\mu_1 \tag{18}$$

due to $y_{0j_0} = y_{\ell j_0} = \xi_{\ell j_0} = \mu_1 > 0$ for all $\ell \in \{1, \ldots, r\}$, according to (12). Suppose first there is $\ell_0 \in \{1, \ldots, r\}$ such that $y_{\ell_0-1,j_0-1} = 0$. Then $\xi_{\ell_0,j_0-1} \leq -w\mu_1 < 0$ by (18), which ensures $y_{\ell_0,j_0-1} = R(\xi_{\ell_0,j_0-1}) = 0$. By applying this argument inductively, we obtain $y_{\ell,j_0-1} = 0$ for all $\ell \in \{\ell_0, \ldots, r\}$, which implies (10) according to (12).

On the other hand, if $y_{\ell-1,j_0-1} > 0$ for each $\ell \in \{1, \ldots, r\}$, then $y_{\ell-1,j_0-1} = \xi_{\ell-1,j_0-1}$ due to (2). This means $y_{\ell-1,j_0} - y_{\ell-1,j_0-1} = \xi_{\ell-1,j_0} - \xi_{\ell-1,j_0-1}$ for all

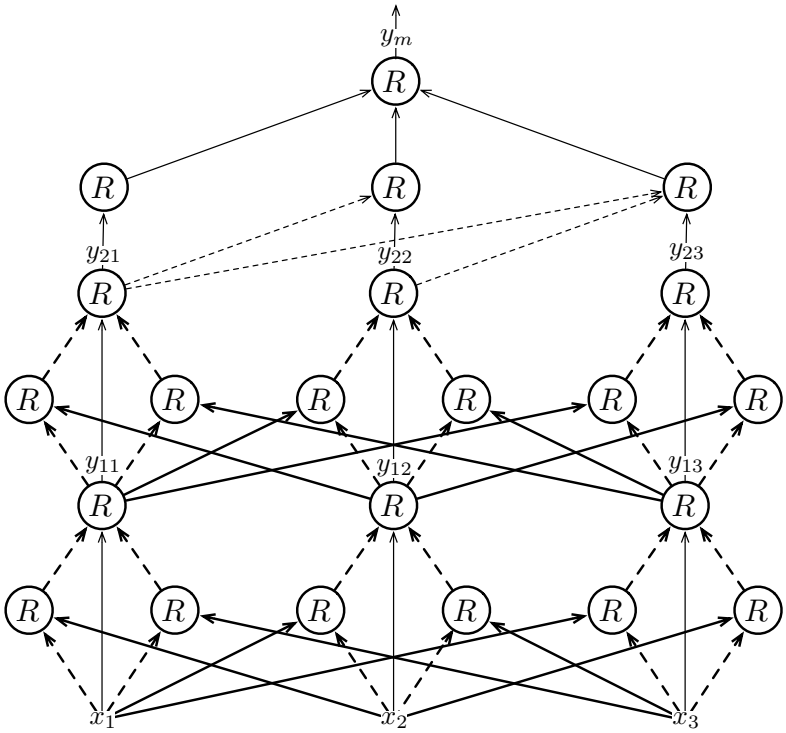

Fig. 2. An example of a depth-6 NN $\mathcal{N}_2$ ($r = 2$) of 22 (non-input) neurons, that computes the maximum of $n = 3$ nonnegative real numbers according to Theorem 2, where the solid and dashed, thick or thin arrows depict the connections with the weights $\sqrt{w}$ and $-\sqrt{w}$, or 1 and -1, respectively.

$\ell \in \{2, \ldots, r\}$, which entails $\xi_{rj_0} - \xi_{r,j_0-1} \geq (w+1)^r(\mu_1 - \mu_2)$ according to (17) and (11). Hence, $\xi_{r,j_0-1} \leq \mu_1 - (w+1)^r(\mu_1 - \mu_2) \leq 0$ due to (7), implying $y_{r,j_0-1} = 0$ which ensures (10) by (12) also in this case. This completes the proof of (10) and terminates the validity of notational assumption (11).

According to (10), there exists $k_0 = \min\{k \in \{1, \ldots, n\} \mid y_{rk} = \mu_1\}$ and $y_{rj} = 0$ for all $j \in \{1, \ldots, k_0 - 1\}$, which means $y_{rk} - \sum_{j=1}^{k-1} y_{rj} \leq 0$ for all $k \in \{1, \ldots, n\} \setminus \{k_0\}$, whereas $y_{rk_0} - \sum_{j=1}^{k_0-1} y_{rj} = y_{rk_0} = \mu_1 > 0$. This shows that formula (9) provides the correct maximum $y_m = \mu_1$ due to (2).

Finally, the formulas (8) and (9) can be evaluated by a NN $\mathcal{N}_r$ of depth $2r + 2$ with the input layer $X \subset V$ and one output $m \in V$ implementing the outer sum in (9) so that $y_m = \mathcal{N}_r(x_1, \ldots, x_n) = \mu_1$. Namely, for each $\ell \in \{1, \ldots, r\}$, the $(2\ell - 1)$th layer in $\mathcal{N}_r$ contains $n^2 - n$ neurons that implement $R\big(\sqrt{w}\,(y_{\ell-1,i} - y_{\ell-1,j})\big)$ from (8) for every $j \in \{1, \ldots, n\}$ and $i \in X \setminus \{j\}$, which are used by n neurons in the (2ℓ)th layer for computing $y_{\ell,j}$ according to (8) for $j \in \{1, \ldots, n\}$. The $(2r + 1)$th layer of $\mathcal{N}_r$ is composed of n neurons that evaluate the summands $R\big(y_{rk} - \sum_{j=1}^{k-1} y_{rj}\big)$ in (9) for $k \in \{1, \ldots, n\}$. Figure 2

depth of $\mathcal{N}_r$	4 ($\mathcal{N}_1$)	6 ($\mathcal{N}_2$)	8 ($\mathcal{N}_3$)	10 ($\mathcal{N}_4$)	12 ($\mathcal{N}_5$)	22 ($\mathcal{N}_{10}$)	32 ($\mathcal{N}_{15}$)
16-bit ushort	256.00	15.97	6.28	3.88	2.87	1.43	1.05
32-bit uint	65536.00	256.00	40.31	15.97	9.14	2.87	1.85
64-bit ulong	4294967296.00	65536.00	1625.50	256.00	84.45	9.14	4.28

Table 1. The weights (rounded up to hundredths) of $\mathcal{N}_r$ that computes MAX_n by Theorem 2, in terms of its depth, for different b-bit unsigned integer data types.

depth of $\mathcal{N}_r$	8 ($\mathcal{N}_3$)	22 ($\mathcal{N}_{10}$)	44 ($\mathcal{N}_{20}$)	102 ($\mathcal{N}_{50}$)	302 ($\mathcal{N}_{150}$)	1002 ($\mathcal{N}_{500}$)
16-bit half ($e = 4$)	101.59	3.88	1.74	1	1	1
32-bit single ($e = 7$)	7.90E13	14766.09	121.52	6.75	1.62	1
64-bit double ($e = 10$)	1.83E105	3.79E31	6.16E15	2068279.89	127.41	4.17

Table 2. The weights (rounded up to hundredths or to three significant digits) of $\mathcal{N}_r$ that computes MAX_n according to Theorem 2, in terms of its depth, for different b-bit floating-point data types with e-bit exponents.

shows an example of $\mathcal{N}_2$ ($r = 2$) for $n = 3$ inputs. Altogether, the size of $\mathcal{N}_r$ is $r(n^2 - n + n) + n + 1 = rn^2 + n + 1$ and its weight is $\max(1, \sqrt{w})$ according to (8) and (9), which completes the proof. □

For example, Theorem 2 provides a depth-4 NN $\mathcal{N}_1$ of size $n^2 + n + 1$ that computes the MAX_n function and has the weight $\max\left(1, \sqrt{\mu_2/(\mu_1 - \mu_2)}\right)$ due to (7). In general, however, the assumption (7) of Theorem 2 cannot be guaranteed in advance. This means that the NN $\mathcal{N}_r$ in Theorem 2 can fail to compute the maximum for large numbers or very close numbers violating this assumption under which $\mathcal{N}_r$ was constructed. Nevertheless, the condition (7) can be ensured when the input real values to $\mathcal{N}_r$ are represented in the computer using data types within a constant number b of bits. Tables 1 and 2 show the tradeoffs between the weight and depth of $\mathcal{N}_r$ for increasing r when the unsigned integer and floating-point data types, respectively, are used with different usual b-bit precision where $b = 16, 32, 64$ (including $e = 4, 7, 10$ bits, respectively, for the floating-point exponent). Namely, for the b-bit unsigned integer data types, we know that $\mu_1 \leq 2^b - 1$ and $\mu_1 - \mu_2 \geq 1$, which means that the weight $\max(1, \sqrt{w})$ of $\mathcal{N}_r$ in Theorem 2 meets its assumption (7) for $w = \sqrt[r]{2^b - 1} - 1$. For the b-bit floating-point data types in the IEEE 754 standard, including e bits for the exponent, we have $\mu_1 \leq 2^{2^e}(1 - 2^{-b+e+1})$ and $\mu_1 - \mu_2 \geq 2^{-2^b - b + e + 4}$, which ensures that the weight $\max(1, \sqrt{w})$ of $\mathcal{N}_r$ in Theorem 2 satisfies its assumption (7) for $w = \sqrt[r]{2^{2^{e+1}-3}(2^{b-e-1} - 1)} - 1$. For even lower bitwidths employed in approximate energy-efficient DNNs [15], max pooling can be implemented using only a few convolutional layers with small weights. For example, for $b = 4$ bits using either integer or microfloat data types, Theorem 2 provides a four-layer NN $\mathcal{N}_1$ with the weight $w < 4$.

4 Lower Bound

In this section, we present a lower bound on the depth of NNs computing the MAX_n function. In the following theorem, we will prove that two layers of ReLU neurons are not enough to implement MAX_3, which means that the maximum pooling layer cannot be replaced in CNNs even with two convolutional layers.

Theorem 3. *There is no depth-2 NN that computes the maximum of more than two nonnegative real numbers.*

Proof. On the contrary, suppose there is a depth-2 NN $\mathcal{N}$ which contains three input neurons $X = \{1, 2, 3\} \subset V$ reading nonnegative real inputs $x_i \geq 0$ for $i \in X$, one hidden layer $U \subset V$, and one output neuron in the second layer, that computes

$$\max(x_1, x_2, x_3) = \mathcal{N}(x_1, x_2, x_3) = R\left(w_0 + \sum_{j \in U} w_j R(\xi_j)\right) \tag{19}$$

including the excitation

$$\xi_j = w_{j0} + \sum_{i \in X} w_{ji} x_i \quad \text{for } j \in U\,, \tag{20}$$

where $w_0, w_{j0} \in \mathbb{R}$ and $w_j, w_{ji} \in \mathbb{R}$ for $j \in U$ and $i \in X$, are real biases and weights, respectively. For simplicity of initial notation, formula (19) assumes without loss of generality that there is no direct connection in $\mathcal{N}$ leading from its input to its output (this case will be resolved later anyway), because otherwise such an edge could be split into two connections by a hidden neuron added to U, one with the original weight and the other with the unit weight. Clearly, $U \neq \emptyset$. In addition, let $w_j \neq 0$ and $\{w_{ji} \mid i \in X\} \neq \{0\}$ for all $j \in U$. Moreover, we assume that x_1, x_2, x_3 are positive reals since if (19) cannot be achieved for $x_1, x_2, x_3 > 0$, then even more so for $x_1, x_2, x_3 \geq 0$. Hence, formula (19) reduces to

$$\max(x_1, x_2, x_3) = w_0 + \sum_{j \in U} w_j R(\xi_j) \tag{21}$$

due to (2), $\mathcal{N}(x_1, x_2, x_3) \in \{x_1, x_2, x_3\}$, and $x_1, x_2, x_3 > 0$.

In the three-dimensional Euclidean space $\mathbb{E}^3$, the planes $w_{j0} + \sum_{i \in X} w_{ji} x_i = 0$ for every $j \in U$, cut its positive octant ($x_i > 0$ for all $i \in X$) into p open convex polytopes $P_r \subset \mathbb{E}^3$ for $r \in \{1, \ldots, p\}$. Each such a polytope P_r is thus an intersection of open half-spaces

$$w_{j0} + \sum_{i \in X} w_{ji} x_i \begin{cases} > 0 & \text{if } j \in U_r \\ < 0 & \text{if } j \in U \setminus U_r \end{cases} \tag{22}$$

for some $U_r \subset U$. For each $r \in \{1, \ldots, p\}$, equation (21) can be rewritten by (2), (20), and (22) as $\max(x_1, x_2, x_3) = W_{r0} + \sum_{i \in X} W_{ri} x_i$ for all $(x_1, x_2, x_3) \in P_r$,

$$\text{where} \quad W_{r0} = w_0 + \sum_{j \in U_r} w_j w_{j0} \quad \text{and} \quad W_{ri} = \sum_{j \in U_r} w_j w_{ji} \ \text{ for } i \in X\,. \tag{23}$$

Since P_r is open and $\max(x_1, x_2, x_3) \in \{x_1, x_2, x_3\}$, there is $i_r \in X$ such that $x_{i_r} = \max(x_1, x_2, x_3)$ for all $(x_1, x_2, x_3) \in P_r$, which means

$$W_{ri} = \begin{cases} 1 & \text{if } i = i_r \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i \in X \cup \{0\}\,. \tag{24}$$

Consider that one moves from a polytope P_r to an adjacent polytope P_s for some $r, s \in \{1, \ldots, p\}$, through a single plane $w_{k0} + \sum_{i \in X} w_{ki} x_i = 0$ where $k \in U$ belongs to the symmetric difference $U_r \triangle U_s$ due to (22), say $k \in U_r \setminus U_s$ (possibly after exchanging the indices r and s). This plane can be represented by more neurons $j \in U_r \triangle U_s$ whose weights differ by multiplicative real constants $C_{jk} \neq 0$:

$$w_{ji} = C_{jk} w_{ki} \quad \text{for } i \in X \cup \{0\}\,. \tag{25}$$

We know by (22) that

$$C_{jk} \begin{cases} > 0 & \text{if } j \in U_r \setminus U_s \\ < 0 & \text{if } j \in U_s \setminus U_r\,. \end{cases} \tag{26}$$

It follows from (23) and (25) that

$$\begin{aligned} W_{ri} - W_{si} &= \sum_{j \in U_r} w_j w_{ji} - \sum_{j \in U_s} w_j w_{ji} \\ &= \sum_{j \in U_r \setminus U_s} w_j C_{jk} w_{ki} - \sum_{j \in U_s \setminus U_r} w_j C_{jk} w_{ki} = C_k w_{ki} \end{aligned} \tag{27}$$

for $i \in X \cup \{0\}$, where

$$C_k = \sum_{j \in U_r \setminus U_s} w_j C_{jk} - \sum_{j \in U_s \setminus U_r} w_j C_{jk}\,. \tag{28}$$

First suppose that $i_r \neq i_s$, then $W_{ri_r} - W_{si_r} = 1$ due to (24), which ensures $C_k \neq 0$ by (27). Thus,

$$w_{ki} = \frac{W_{ri} - W_{si}}{C_k} = \begin{cases} \frac{1}{C_k} & \text{if } i = i_r \\ -\frac{1}{C_k} & \text{if } i = i_s \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i \in X \cup \{0\}\,, \tag{29}$$

according to (27) and (24). Hence, $w_{ki_r} = -w_{ki_s}$ and $w_{k0} = w_{ki} = 0$ for $i \in X \setminus \{i_r, i_s\}$, which implies $w_{ji_r} = -w_{ji_s}$ and $w_{j0} = w_{ji} = 0$ for $i \in X \setminus \{i_r, i_s\}$, for every $j \in U_r \triangle U_s$, due to (25). By (20) and (2), this gives

$$R(\xi_j) = \begin{cases} w_{ji_r} R(x_{i_r} - x_{i_s}) & \text{if } w_{ji_r} > 0 \\ -w_{ji_r} R(x_{i_s} - x_{i_r}) & \text{if } w_{ji_r} < 0 \end{cases} \quad \text{for } j \in U_r \triangle U_s\,. \tag{30}$$

On the other hand, for $i_r = i_s$, we know $W_{ri} - W_{si} = 0$ for $i \in X \cup \{0\}$, due to (24), which means $C_k = 0$ by (27), since $\{w_{ki} \mid i \in X\} \neq \{0\}$. Hence,

$$C'_k = \sum_{j \in U_r \setminus U_s} w_j C_{jk} = \sum_{j \in U_s \setminus U_r} w_j C_{jk} \tag{31}$$

follows from (28). We have

$$\sum_{j\in U_r\triangle U_s} w_j R(\xi_j) = \sum_{j\in U_r\setminus U_s} w_j C_{jk} R(\xi_k) - \sum_{j\in U_s\setminus U_r} w_j C_{jk} R(-\xi_k)$$
$$= C'_k \left(R(\xi_k) - R(-\xi_k)\right) = C'_k \xi_k = C'_k w_{k0} + \sum_{i\in X} C'_k w_{ki} x_i \quad (32)$$

according to (20), (25), (26), (2), and (31).

Finally, for the remaining neurons $j \in U$ which represent the planes $w_{j0} + \sum_{i\in X} w_{ji} x_i = 0$ that do not intersect the positive octant, we have either $\xi_j < 0$ for all inputs to $\mathcal{N}$, which means $R(\xi_j) = 0$, or $\xi_j > 0$ for all inputs, implying

$$R(\xi_j) = \xi_j = w_{j0} + \sum_{i\in X} w_{ji} x_i \,. \quad (33)$$

We plug (30), (32), and (33) into (21) which reduces to

$$\max(x_1, x_2, x_3) = \sum_{i\in X} a_i x_i + \sum_{\substack{i,j\in X \\ i\neq j}} a_{ij} R(x_i - x_j) \quad (34)$$

for some nine real constants $a_1, a_2, a_3, a_{12}, a_{21}, a_{13}, a_{31}, a_{23}, a_{32} \in \mathbb{R}$, due to (24).

Now, consider the following seven inputs $(1,1,2)$, $(1,1,3)$, $(2,2,1)$, $(2,3,2)$, $(2,3,3)$, $(3,1,3)$, and $(3,2,2)$ to $\mathcal{N}$, which must satisfy (34). This generates the following system of seven linear equations in the nine real variables a_1, a_2, a_3, a_{12}, $a_{21}, a_{13}, a_{31}, a_{23}, a_{32} \in \mathbb{R}$:

$$\begin{aligned}
a_1 + a_2 + 2a_3 + a_{31} + a_{32} &= 2 & 2a_1 + 3a_2 + 3a_3 + a_{21} + a_{31} &= 3 \\
a_1 + a_2 + 3a_3 + 2a_{31} + 2a_{32} &= 3 & 3a_1 + a_2 + 3a_3 + 2a_{12} + 2a_{32} &= 3 \\
2a_1 + 2a_2 + a_3 + a_{13} + a_{23} &= 2 & 3a_1 + 2a_2 + 2a_3 + a_{12} + a_{13} &= 3 \\
2a_1 + 3a_2 + 2a_3 + a_{21} + a_{23} &= 3
\end{aligned} \quad (35)$$

that can be checked by the Rouché-Capelli theorem to have no solution, which provides a contradiction and completes the proof. □

5 Conclusion

We investigated the computational power of max pooling in CNNs by expressing the MAX_n function using ReLU-based NNs. Theorem 1 constructs a NN of size n and depth $\lceil \log_2 n \rceil + 1$ that implements MAX_n, successfully used in the practical AppMax method [15]. Theorem 2 shows that for bounded input numbers with limited precision (constant-bit data types), MAX_n can be computed with quadratic size and constant depth which decreases as weight magnitude increases, quantifying a depth-weight trade-off in CNNs. This construction can also be applied to approximate, energy-efficient DNNs with reduced bitwidths for floating-point numbers [15]. Furthermore, Theorem 3 proves that no NN of

depth 2 can compute the maximum of 3 nonnegative numbers. Extending this result to constant depth is the main open problem, which would establish the strict depth hierarchy of NNs [7]. Overall, these findings address whether max pooling layers offer greater computational power and efficiency than convolutional ones.

Acknowledgments. This research was institutionally supported by RVO: 67985807, J. Šíma was partially supported by the research programme of the Strategy AV21 "AI: Artificial Intelligence for Science and Society," and J. Cabessa was partially supported by the Czech Science Foundation grant GA25-15490S. J. Šíma thanks his son Dominik for implementing computer calculations to create Tables 1, 2, and system (35).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. In: Proc. ICLR 2018. OpenReview.net (2018)
2. Averkov, G., Hojny, C., Merkert, M.: On the expressiveness of rational ReLU neural networks with bounded depth. In: Proc. ICLR 2025. OpenReview.net (2025)
3. Boureau, Y., Ponce, J., LeCun, Y.: A theoretical analysis of feature pooling in visual recognition. In: Proc. ICML 2010. pp. 111–118. Omnipress (2010)
4. Christlein, V., Spranger, L., Seuret, M., Nicolaou, A., Král, P., Maier, A.: Deep generalized max pooling. In: Proc. ICDAR 2019. pp. 1090–1096. IEEE (2019)
5. Grillo, M., Hertrich, C., Loho, G.: Depth-bounds for neural networks via the braid arrangement. CoRR, arXiv:2502.09324 [cs.LG] (2025)
6. Haase, C., Hertrich, C., Loho, G.: Lower bounds on the depth of integral ReLU neural networks via lattice polytopes. In: Proc. ICLR 2023. OpenReview.net (2023)
7. Hertrich, C., Basu, A., Summa, M.D., Skutella, M.: Towards lower bounds on the depth of ReLU neural networks. SIAM J. Discret. Math. **37**(2), 997–1029 (2023)
8. Li, Z., Liu, F., Yang, W., Peng, S., Zhou, J.: A survey of convolutional neural networks: Analysis, applications, and prospects. IEEE Trans. Neural Netw. Learn. Syst. **33**(12), 6999–7019 (2022)
9. Matoba, K., Dimitriadis, N., Fleuret, F.: Benefits of max pooling in neural networks: Theoretical and experimental evidence. Trans. Mach. Learn. Res. (2023)
10. Mukherjee, A., Basu, A.: Lower bounds over Boolean inputs for deep neural networks with ReLU gates. CoRR, arXiv:1711.03073 [cs.CC] (2017)
11. Murray, N., Perronnin, F.: Generalized max pooling. In: Proc. CVPR 2014. pp. 2473–2480. IEEE (2014)
12. Oyedotun, O.K., Ismaeil, K.A., Aouada, D.: Why is everyone training very deep neural network with skip connections? IEEE Trans. Neural Netw. Learn. Syst. **34**(9), 5961–5975 (2023)
13. Park, Y., Hwang, G., Lee, W., Park, S.: Expressive power of ReLU and step networks under floating-point operations. Neural Networks **175**, 106297 (2024)
14. Safran, I., Reichman, D., Valiant, P.: How many neurons does it take to approximate the maximum? In: Proc. SODA 2024. pp. 3156–3183. SIAM (2024)
15. Šíma, J., Vidnerová, P.: Weight-rounding error in deep neural networks. (To appear at ECML PKKD 2025), `https://www.cs.cas.cz/sima/rnderr.pdf`