Approximation Capabilities of Neural Networks using Morphological Perceptrons and Generalizations

William Chang, Hassan Hamad, Keith M. Chugg
Ming Hsieh Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, United States of America
{chan087, hhamad, chugg}@usc.edu

***Abstract*—Standard artificial neural networks (ANNs) use sum-product or multiply-accumulate node operations with a memoryless nonlinear activation. These neural networks are known to have universal function approximation capabilities. Previously proposed morphological perceptrons use max-sum, in place of sum-product, node processing and have promising properties for circuit implementations. In this paper we show that these max-sum ANNs do not have universal approximation capabilities. Furthermore, we consider proposed signed-max-sum and max-star-sum generalizations of morphological ANNs and show that these variants also do not have universal approximation capabilities. We contrast these variations to log-number system (LNS) implementations which also avoid multiplications, but do exhibit universal approximation capabilities.**

***Index Terms*—Neural networks, morpholgical perceptrons, log number system**

I. Introduction

Deep neural networks are driving the AI revolution. They have led to breakthroughs in various fields ranging from computer vision [1] and natural language processing [2] to protein folding [3] and autonomous driving [4]. The current trend is toward larger models. This has motivated recent work on complexity reductions for both inference and training. Complexity reduction techniques such as pruning [5], sparsity [6], and quantization [7] have been proposed. This is particularly important for models deployed on edge devices that have limited memory and computational resources.

Another approach for complexity reduction is to depart from the Multiply-Accumulate (MAC) (or sum-product) processing used in standard Artificial Neural Networks (ANNs). Radically different network structures, such as the Spiking Neural Network (SNN) [8], [9], have been proposed. Others have proposed to simply replace sum-product operations with a different, and ideally more efficient, operations [10]–[14]. In table I, we list the types of node operations that will be the discussed in this paper, where $\vee$ denotes the $\max$ operator. The morphological perceptron replaces the sum-product by a $\max$-sum operation [10], [12]. The use of the $\max$ function inherently adds non-linearity to the network. The morphological perceptron was extended to use a signed $\max$-sum operation by adding a binary sign parameter in [11]. In the field of digital communications, and specifically in error correction coding literature, the $\max^*$-sum operation[1] is widely used in decoding iterative digital codes [15]. This operation can be seen as a natural extension to the $\max$-sum node. Finally, it is well known that the exact equivalent of a sum-product

TABLE I
Different node operations studied in this paper

Node Operation	Sum-Product Equivalent
sum-product	$z = \sum_i (x_i y_i)$
max-sum	$z = \bigvee_i (x_i + y_i)$
signed max-sum	$z = \bigvee_i a_i (x_i + y_i)$
$\max^*$-sum	$z = \max^*_i (x_i + y_i)$
LNS	$z = \max^*_{\pm,i} (\log \lvert x_i \rvert + \log \lvert y_i \rvert)$ $s_z = s_x \oplus s_y$

This work is supported in part by the National Science Foundation (CCF-1763747).

[1] $\max^*$-sum is also known as the Jacobian Logarithm [15]

can be implemented in the log domain, i.e. using the Logarithmic Number System (LNS). LNS requires the use of $\max^*_+$-sum and $\max^*_-$-sum processing along with tracking the signs of the linear operands.

A neural network using LNS can thus implement the same processing as a standard Artificial Neural Network (ANN). Therefore, LNS-based networks inherit the approximation capabilities of standard ANNs. Specifically, a single layer network in LNS with a non-linear activation is also a universal function approximator [16], [17]. Several previous works successfully trained ANNs using LNS and demonstrated that the performance of these LNS-networks is on a par with standard networks on publicly available datasets [13], [14], [18]. To the best of our knowledge, the approximation capabilities of neural networks with these max-like units have not yet been studied. In this paper, our goal is to characterize the approximation capabilities of neural networks having a max-sum, signed max-sum or a $\max^*$-sum nodes. We prove that these kind of networks are not universal approximators. In addition, we characterize the exact set of functions that they can approximate.

The paper is organized as follows. In section 2, we define our notation and review the universal approximation theorem of neural networks. Section 3 contains the main results of the paper. We conclude in section 4. The proofs of all the theorems are given in the appendices.

II. Background

Consider a scalar-input and scalar-output fully connected neural network with d hidden layers as shown in Fig. 1. Denote the node function at each unit by the $g(.)$. Note that we also lump the activation function, if any, in the definition of $g(.)$. The standard MAC-based node processing $g(.)$ with input activation vector $\boldsymbol{x} = [x_1, x_2, .., x_n]$, weight vector $\boldsymbol{w} = [w_1, w_2, .., w_n]$, bias value b, and a non-linear activation function $\sigma(.)$, is defined as $g(\boldsymbol{x}, \boldsymbol{w}, b) = \sigma(b + \sum_{i=1}^{n} w_i x_i)$.

A well known fact of neural networks is that they are universal approximators. In [17], it was shown that a one-hidden layer neural network $f : \mathbb{R}^k \to \mathbb{R}^l$ with a bounded and nonconstant activation function can uniformly approximate any function h on $\mathbb{R}^k$, given that sufficient hidden units are available. This result has been later extended to networks with unbounded activations, such as the rectified linear activation (Relu) [19].

A class of neural networks $\mathcal{N} : [0,1]^n \to \mathbb{R}$ is said to uniformly approximate a given function $h : [0,1]^n \to \mathbb{R}$ iff for any $\epsilon > 0$, there exists a function $N \in \mathcal{N}$ such that $\forall x \in [0,1]^n : |N(x) - h(x)| < \epsilon$. This class of neural networks is said to be a universal approximator iff it uniformly approximates all continuous functions $h : [0,1]^n \to \mathbb{R}$.

The universal approximation theorems mentioned above concern neural networks with the standard sum-product or MAC processing. In this paper, the approximation capabilities of neural networks with three different non-standard node operations are investigated.

Our proof method is to show neural networks with the non-MAC nodes exhibit bounded first derivatives. By the following lemma, this implies that these networks are not universal approximators.

Lemma 1. *Given a single-input and single-output continuous neural network $f : [0,1] \to \mathbb{R}$. Suppose that $a \leq f'(x) \leq b$ almost everywhere for $a, b \in \mathbb{R}$. Then f can only uniformly approximate functions $h : [0,1] \to \mathbb{R}$ such that $a \leq h'(x) \leq b$ almost everywhere.*

While this result appears intuitive, the same does not hold for higher order derivatives. That is, if $f^{(n)}(x) \in [a, b]$ for some $n \geq 2$, this does not mean that f can only universally approximate functions h such that $a \leq h^{(n)}(x) \leq b$. A simple counter example is a sum-product network with Relu activation on the hidden layers, which is known to be a universal approximator [19] but has a bounded second derivative. In Appendix 1, we show that the second derivative of such a network is zero a.e.

III. Approximation Capabilities

In this section we consider the approximation capabilities of neural networks with max-sum processing in place of standard sum-product processing. We also consider two generalizations to max-sum processing. We show that these non-MAC networks are not universal approximators from $\mathbb{R} \to \mathbb{R}$, and thus they are also not universal approximators for $\mathbb{R}^n \to \mathbb{R}$ for any $n \geq 1$.

We state the results of each case in this section and provide the proofs in appendices.

A. Max-Sum Network

In the case of a *max-sum* network, the node function $g(.)$ in Fig. 1 is defined as

$$\begin{aligned} g(\boldsymbol{x}, \boldsymbol{w}, b) &= b \vee (w_1 + x_1) \vee \cdots \vee (w_n + x_n) \\ &= b \vee \left(\bigvee_{i=1}^{n} (w_i + x_i) \right) \end{aligned} \tag{1}$$

where $\vee$ is the max operator, i.e. $x \vee y = \max(x, y)$. The max-sum node is also known as the morphological perceptron [10]. Note that the *max* operation inherently adds non-linearity to the node and no explicit activation function is used. Next we present our first theorem.

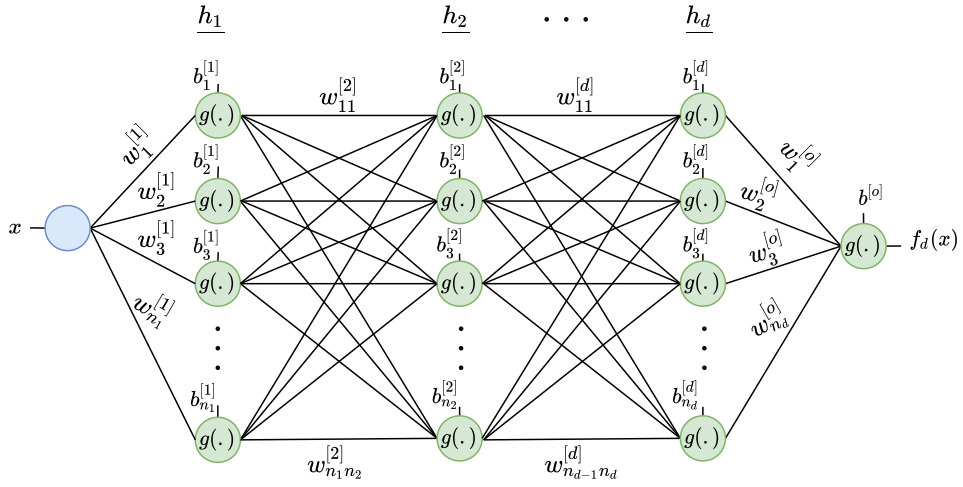

Fig. 1. A d hidden layer neural network with scalar input x and scalar output $f_d(x)$. The node processing at each unit is donated by the function $g(.)$. Note that $g(.)$ also includes the activation function, when applicable.

Theorem 1. *Consider a single-input single-output, d hidden layer neural network with* max-sum *node processing $f_d(x) : \mathbb{R} \to \mathbb{R}$.*

Then $f_d(x) = \max(w, w' + x)$ for some constants $w, w' \in \mathbb{R}$.

Thus, max-sum node processing results in a very limited class of functions that can be realized. In particular, by lemma 1, max-sum networks are not universal approximators. In [10], max-sum layers were combined with standard sum-product layers to obtain effective classifiers.

B. Signed Max-Sum Network

In the case of a *signed max-sum* network, the node function $g(.)$ in Fig. 1 is defined as

$$\begin{aligned} g(\boldsymbol{x}, \boldsymbol{w}, b) &= b \vee a_1(w_1 + x_1) \vee \cdots \vee a_n(w_n + x_n) \\ &= b \vee \left(\bigvee_{i=1}^{n} a_i(w_i + x_i) \right) \end{aligned} \tag{2}$$

where $a_i \in \{-1, 1\}$. Note that the a_i's can either be learnable network parameters or fixed. For our purposes, this choice is irrelevant to the study of the approximation capability of the network since we consider all $a_i \in \{-1, 1\}$.

Theorem 2. *Consider a single-input single-output, d hidden layer neural network with* signed max-sum *node processing $f_d(x) : \mathbb{R} \to \mathbb{R}$.*

Then $f'_d(x) \in \{-1, 0, 1\}$ a.e.

By Lemma 1 *signed max-sum* networks are not universal approximators. In fact, by the above theorem, these networks have very limited approximation capabilties. Note also that the *max-sum* node is a special case of the *signed max-sum* where all the a_i's are set to 1. Theorem 1 implies that the derivative of max-sum networks is limited to the set $\{0, 1\}$ which is a proper subset of the possible derivatives of signed-max-sum networks.

C. Max*-Sum Network

To define the *max*-sum* node, first note the definition of the *max** function:

$$\max{}^*(x, y) = \ln\left(e^x + e^y\right) \tag{3}$$

which can be nested as follows

$$\begin{aligned} \max_i{}^* x_i &= \max{}^*(x_1, x_2, .., x_n) \\ &= \ln\left(e^{x_1} + e^{x_2} + .. + e^{x_n}\right) \\ &= \ln\left(\sum_{i=1}^{n} e_i^x\right) \end{aligned} \tag{4}$$

In the case of a *max*-sum* network, the node function $g(.)$ in Fig. 1 is defined as

$$\begin{aligned} g(\boldsymbol{x}, \boldsymbol{w}, b) &= \sigma(\max{}^*(b, \max_i{}^* x_i + w_i)) \\ &= \sigma\left(\ln\left(e^B + \sum_{i=0}^{n} e^{x_i + w_i}\right)\right) \end{aligned} \tag{5}$$

where $\sigma(x)$ is any activation function such that $\sigma'(x) \in [0, 1]$, e.g. a Relu.

The *max*-sum* node processing may be viewed as doing arithmetic in the log domain. Consider the standard sum-product $c_i = \sum_i a_i b_i$ where $a_i, b_i > 0 \; \forall i$. If we let $C_i = \ln(c_i), B_i = \ln(b_i)$ and $A_i = \ln(a_i)$ then

$$\begin{aligned} C_i &= \ln\left(\sum_i a_i b_i\right) = \ln\left(\sum_i e^{A_i + B_i}\right) \\ &= \max_i{}^*(A_i + B_i) \end{aligned} \tag{6}$$

This indicates that if all data inputs to the network as well as all the weights are non-negative, then working with *max*-sum* nodes is the equivalent of working with the standard sum-product node, but in the log domain. This is the reason for introducing a nonlinear activation function for *max*-sum* networks. Specifically, if a linear mapping with non-negative weights, biases, inputs, and outputs is implemented in the log domain, it would be a network with *max*-sum* node processing and no activation function.

Theorem 3. *Consider a single-input single-output, d hidden layer neural network with* max*-sum *node processing $f_d(x) : \mathbb{R} \to \mathbb{R}$.*

Then $0 \leq f_d'(x) < 1$

Again, by Lemma 1 *max*-sum* networks are not universal approximators.

D. Discussion

In the previous subsections we proved that a neural network with a *max-sum*, *signed max-sum* or *max*-sum* node processing is not a universal approximator. If *max*-sum* were extended to include signed inputs, weights, and biases, one would need to separately track the effects on the sign and magnitude of the quantities. In fact, this is LNS arithmetic.

A neural network with LNS arithmetic, and a nonlinear activation function, is equivalent to a log-domain sum-product ANN, and thus is also a universal approximator. This suggests that it is unlikely that there exists an ANN using *max*-like family of computations that is a universal approximator and substantially less complex than an LNS network.

We note that a recent pre-print paper, posted after the acceptance of this work, proved that networks with sum-product processing and non-negative weights are not universal approximators [20]. Specifically, standard ANNs with non-negative weights can generate sections with negative slopes only. This is complementary to our result for *max*-sum* processing, where *max*-sum* processing can be seen as the log domain equivalent of sum-product with non-negative values.

IV. Conclusion

We showed that an ANN with morpholgical perceptron (i.e., max-sum) node processing has a very limited approximation capabilities. Furthermore, direct extensions to this node processing, such as signed-max-sum and $\max^*$-sum, also are not universal approximators. This suggests that it is unlikely to achieve universal approximation with a max-sum generalization substantially simpler than an LNS implementation.

Appendix A
Proof of Lemma 1

Proof. of Lemma 1

Let $N(x)$ be a neural network that uniformly approximates $h(x) : [0, 1] \to \mathbb{R}$ to ϵ accuracy for any $\epsilon > 0$. For the sake of contradiction, suppose that there is a set of nonzero measure I such that $h'(x) > b$ a.e. for $x \in I$. Without any loss of generality, we can assume $I = [c, d]$ is a closed interval. It follows from the condition $h'(x) > b$ that

$$h(d) = h(c) + b(d - c) + \delta \tag{7}$$

for some constant $\delta > 0$. Since $N(x)$ is at most ϵ away from $h(a)$ it follows that $N(c) < h(c)+\epsilon$. Thus, from the continuity of $N(x)$ as well as the condition $N'(x) \leq b$

$$N(d) < h(a) + \epsilon + (d - c)b. \tag{8}$$

Thus:

$$h(d) - N(d) > \delta - \epsilon \tag{9}$$

When $\epsilon < \frac{\delta}{2}$, $h(d) - N(d) > \frac{\delta}{2} > \epsilon$, which contradicts the fact that $N(x)$ uniformly approximates $h(x)$ to ϵ accuracy.

Similarly, it can be shown that $h'(x) \geq a$ a.e. □

Proposition 1. *A single-input single-output, sum-product Network with Relu activation has its second derivative equal to 0 a.e.*

Proof. Let $\sigma = \max(0, x)$ be the ReLu function and $N_d(x)$ be the resulting d-hidden layer network. Proceed by induction on the number of hidden layers. For one hidden layer network with width n, we have

$$N_1(x) = \sum_{i=1}^{n} w_i^{[o]} \sigma(w_i^{[1]} x) \tag{10}$$

So that

$$\begin{aligned} N_1''(x) &= \sum_{i=1}^{n} w_i^{[o]} (w_i^{[1]})^2 \sigma''(w_i^{[1]} x) && (11) \\ &= 0 && (12) \end{aligned}$$

where the last equality holds a.e. since $\sigma''(x) = 0$ except when $x = 0$. For the inductive step, note that if the final hidden layer has width n, then

$$N_d(x) = \sum_{i=1}^{n} w_i^{[o]} \sigma(g_i(x)) \tag{13}$$

where $w_i^{[o]}$ are the weights at the output layer, and $g_1(x), ..., g_n(x)$ can be thought of sub-networks of $N_d(x)$ with $d-1$ hidden layers, and thus by the inductive hypothesis satisfy $g_i''(x) = 0$ a.e. The next two derivatives of this network are computed as

$$N_d'(x) = \sum_{i=1}^{n} w_i^{[o]} \sigma'(g_i(x)) g_i'(x) \tag{14}$$

$$N_d''(x) = \sum_{i=1}^{n} w_i^{[o]} (\sigma''(g_i(x)) g_i'(x)^2 + \sigma'(g_i(x)) g_i''(x)) \tag{15}$$

The first term in the summation is 0 except when $g_i(x) = 0$ which happens finitely many times, and thus this term is equal to 0 a.e. The second term in the summation is 0 a.e. by the inductive hypothesis. Thus, $N_d''(x) = 0$ a.e., completing our induction. □

Appendix B
Proof of Theorem 1

To prove Theorem 1, the following lemma is needed,

Lemma 2. *For any constants a_1, a_2, b_1, b_2, $\exists w_0, w_1$ such that $(a_1 + (b_1 \vee x)) \vee (a_2 + (b_2 \vee x)) = w_0 \vee (w_1 + x)$*

Proof. We first note that $(a_1 + (b_1 \vee x)) \vee (a_2 + (b_2 \vee x)) = a_1 + (b_1 \vee x) \vee (a_2 - a_1 + b_2 \vee x) = a_1 + (b_1 \vee x) \vee (c + (b_2 \vee x))$. Set

$$g(x) = a_1 + (b_1 \vee x) \vee (c + (b_2 \vee x)) \tag{16}$$

We hope to show that $g(x) = w_0 \vee (w_1 + x)$ for some w_0, w_1 which will depend on a_1, b_1, b_2, c. We now have the following cases based on these values.

Case 1: $b_1 > b_2$, $c < 0$. When $x \leq b_2$, we have $f(x) = a_1 + b_1 \vee (b_2 + c) = a_1 + b_1$. When $x \in (b_2, b_1)$, we have $f(x) = a_1 + b_1 \vee (x + c) = a_1 + b_1$. Finally, when $x \geq b_1$, we have $f(x) = a_1 + x \vee (x + c) = a_1 + x$. Thus we can set $w_0 = a_1 + b_1$ and $w_1 = a_1$, completing this case.

Case 2: $b_1 > b_2$, $c > 0$. When $x \leq b_2$, we have $f(x) = a_1 + b_1 \vee (b_2 + c)$. When $x \in (b_2, b_1)$, we have $f(x) = a_1 + b_1 \vee (x + c)$. Finally, when $x \geq b_1$, we have $f(x) = a_1 + x \vee (x + c) = x + c + a_1$. Thus we can set $w_0 = a_1 + b_1 \vee (b_2 + c)$ and $w_1 = c + a_1$, completing this case.

Since $g(x) = a_1 + (b_1 \vee x) \vee (c + (b_2 \vee x)) = a_1 + c + (-c + (b_1 \vee x)) \vee (b_2 \vee x)$, the lemma also holds when $b_2 \geq b_1$. This completes all the cases. □

Proof. of Theorem 1

We can prove this by induction on the number of hidden layers. Consider a one hidden layer network with n computation units. The resulting function is of the form

$$N_1(x) = b^{[o]} \vee \left(\bigvee_{i=1}^{n} \left(w_i^{[o]} + b_i^{[1]} \vee (w_i^{[1]} + x) \right) \right) \tag{17}$$

for some real valued constants $b^{[o]}, w_i^{[o]}, w_i^{[1]}, b_i^{[1]}$ for $i \in [1, n]$. First note that $b \vee (w + x) = w + (b - w) \vee x$. Then

$$N_1(x) = b^{[o]} \vee \left(\bigvee_{i=1}^{n} (w_i^{[o]} + w_i^{[1]} + (b_i^{[1]} - w_i^{[1]}) \vee x) \right) \tag{18}$$

We can now apply Lemma 2 to equation (18) repeatedly to complete our base case.

For the inductive step, suppose that the theorem is true for $d-1$ hidden layers. For a network with d layers, suppose that the d-th hidden layer had n computation units. Denote the output of each computation unit in the d-th hidden layer as $g_i(x)$ for $i \in [1, n]$. Now $g_i(x)$ can be thought of as a neural network with $d-1$ hidden layers. It is clear from these definitions that

$$N_d(x) = b^{[o]} \vee \left(\bigvee_{i=1}^{n} \left(w_i^{[o]} + g_i(x) \right) \right) \tag{19}$$

for some real valued constants $b^{[o]}, w_i^{[o]}$ for $i \in [1, n]$. By the inductive hypothesis, each $g_i(x) = w_i \vee (w_i' + x)$ for some values w_i, w_i'. As with the base case, Lemma 2 can be repeatedly applied to equation (19) to complete the inductive step.

□

Appendix C
Proof of Theorem 2

To prove Theorem 2, we first need the following lemma, analogous to lemma 2.

Lemma 3. *Consider the class of functions*

$$\mathcal{F} := \{c_1, c_2 - x, c_3 + x, c_1 \vee (c_2 - x), c_1 \vee (c_3 + x), (c_2 - x) \vee (c_3 + x), c_1 \vee (c_2 - x) \vee (c_3 + x) | c_1, c_2, c_3 \in \mathbb{R}\} \tag{20}$$

Then $\forall f, g \in \mathcal{F}$, *we have* $f \vee g \in \mathcal{F}$.

Proof. This is trivial to verify except when f or g takes on the form $(c_2 - x) \vee (c_3 + x), c_1 \vee (c_2 - x) \vee (c_3 + x$. To address this case, let $\sigma_{a_1,a_2,a_3}(x) = (a_1 - x) \vee a_2 \vee (a_3 + x)$. Then it is sufficient to verify the following properties

1) $\sigma_{a_1,a_2,a_3}(x) \vee (b - x) = \sigma_{c_1,c_2,c_3}(x)$ for some $c_1, c_2, c_3 \in \mathbb{R}$.
2) $\sigma_{a_1,a_2,a_3}(x) \vee (b + x) = \sigma_{c_1,c_2,c_3}(x)$ for some $c_1, c_2, c_3 \in \mathbb{R}$.
3) $\sigma_{a_1,a_2,a_3}(x) \vee b = \sigma_{c_1,c_2,c_3}(x)$ for some $c_1, c_2, c_3 \in \mathbb{R}$.
4) $\sigma_{a_1,a_2,a_3}(x) \vee \sigma_{b_1,b_2,b_3}(x) = \sigma_{c_1,c_2,c_3}(x)$ for some constants c_1, c_2, and c_3.

Property (1) can be proved as follows,

$$\begin{aligned} &\sigma_{a_1,a_2,a_3}(x) \vee (b - x) \\ &= ((a_1 - x) \vee (b - x)) \vee a_2 \vee (a_3 + x) \\ &= ((a_1 \vee b) - x) \vee a_2 \vee (a_3 + x) \\ &= \sigma_{a_1 \vee b, a_2, a_3}(x) \end{aligned} \tag{21}$$

Properties (2) and (3) can be treated similarly. Property (4) follows from basic properties of $a \vee b$,

$$\begin{aligned} &\sigma_{a_1,a_2,a_3}(x) \vee \sigma_{b_1,b_2,b_3}(x) \\ &= ((a_1 - x) \vee (b_1 - x)) \vee (a_2 \vee b_2) \\ &\quad \vee ((a_3 + x) \vee (b_3 + x)) \end{aligned} \tag{22}$$

Thus, letting $c_1 = a_1 \vee b_1$, $c_2 = a_2 \vee b_2$, $c_3 = a_3 \vee b_3$ gives us the desired result. □

With this lemma we can now prove Theorem 2.

Proof. of Theorem 2 Since all the functions in $\mathcal{F}$ as given in Lemma 3 have their derivatives in the set $\{-1, 0, 1\}$ a.e., it is sufficient to use induction on the number of hidden layers to prove that the neural networks given by the theorem statement belong in $\mathcal{F}$. Consider a single input single output, one hidden layer network with n computation units. The resulting function is of the form

$$N_1(x) = b^{[o]} \vee \left(\bigvee_{i=1}^{n} a_i^{[o]} \left(w_i^{[o]} + b_i^{[1]} \vee a_i^{[1]}(w_i^{[1]} + x) \right) \right) \tag{23}$$

for some real valued constants $b^{[o]}$, $b_i^{[1]}$, $a_i^{[o]}$, $a_i^{[1]}$, $w_i^{[o]}$, $w_i^{[1]}$ for $i \in [1, n]$. Since $a_i^{[1]}(w_i^{[1]} + x)$ and $w_i^{[o]} \in \mathcal{F}$ as in Lemma 3, note that $w_i^{[o]} + b_i^{[1]} \vee a_i^{[1]}(w_i^{[1]} + x) \in \mathcal{F}$, from which it is clear that $a_i^{[o]}(w_i^{[o]} + b_i^{[1]} \vee a_i^{[1]}(w_i^{[1]} + x)) \in \mathcal{F}$ as well. Thus, $N_1(x) \in \mathcal{F}$, completing the base case.

For the inductive step, suppose that the theorem is true for $d - 1$ hidden layers. For a network with d layers, suppose the d-th hidden layer had n computation units. Denote the output of each computation unit in the d-th hidden layer as $g_i(x)$ for $i \in [1, n]$. Now $g_i(x)$ is effectively a neural network with $d - 1$ hidden layers. It is clear from these definitions that

$$N_d(x) = b^{[o]} \vee \left(\bigvee_{i=1}^{n} a_i^{[o]} \left(w_i^{[o]} + g_i(x) \right) \right) \tag{24}$$

for some real valued constants $b^{[o]}, a_i^{[o]}, w_i^{[o]}$ for $i \in [1, n]$. From the inductive hypothesis, $g_i(x) \in \mathcal{F}$, from which a repeated application of Lemma 3 shows that $N_d(x) \in \mathcal{F}$ as well, completing our induction. □

Appendix D
Proof of Theorem 3

Proof. of Theorem 3 It is easy to see that $N_d'(x)$ is undefined when $\sigma'(x)$ is undefined. Thus there are only finitely many points for which $N_d'(x)$ is undefined. For the values of x for which $N_d'(x)$ is defined, we shall show that $N_d'(x) \leq 1$ using induction on the number of hidden layers. Consider a single-input single-output, one hidden layer network with n computation units. The resulting function is of the form

$$N_1(x) = \ln \left[e^{b^{[o]}} + \sum_{i=1}^{n} e^{w_i^{[o]} + \sigma\left(\ln\left(e^{b_i^{[1]}} + e^{w_i^{[1]} + x} \right) \right)} \right] \tag{25}$$

for some real valued constants $b^{[o]}, w_i^{[o]}, b_i^{[1]}, w_i^{[1]}$ for $i \in [1, n]$. From equation (25) it follows that

$$N_1'(x) = \frac{\alpha(x)}{\beta(x)} \tag{26}$$

where

$$\alpha(x) = \sum_{i=1}^{n} \left[e^{w_i^{[o]} + \sigma(M(x))} \sigma'(M(x)) \frac{e^{w_i^{[1]} + x}}{e^{b_i^{[1]}} + e^{w_i^{[1]} + x}} \right] \tag{27}$$

$$\beta(x) = e^{b^{[o]}} + \sum_{i=1}^{n} e^{w_i^{[o]} + \sigma(M(x))} \tag{28}$$

$$M(x) = \ln \left(e^{b_i^{[1]}} + e^{w_i^{[1]} + x} \right) \tag{29}$$

Since $\sigma'(x) \in [0,1]$, it is immediately clear that $N_1'(x) \geq 0$. $N_1'(x) < 1$ follows from the following inequalities.

$$e^{w_i^{[1]}+x} < e^{b_i^{[1]}} + e^{w_i^{[1]}+x} \tag{30}$$

and

$$\sum_{i=1}^{n} \left[e^{w_i^{[o]}+\sigma(M(x))} \sigma'(M(x)) \right] < e^{b^{[o]}} + \sum_{i=1}^{n} e^{w_i^{[o]}+\sigma(M(x))} \tag{31}$$

completing the base case. For the inductive step, suppose the theorem is true for $d-1$ hidden layers. Consider a network with d hidden layers where the dth hidden layer has n computation units. Denote the output of each computation unit in the d-th hidden layer as $g_i(x)$ for $i \in [1,n]$. Now $g_i(x)$ is effectively a neural network with $d-1$ hidden layers. Since there is no activation at the output layer, $N_d(x)$ has the following form

$$N_d(x) = \ln \left[e^{b^{[o]}} + \sum_{i=1}^{n} e^{w_i^{[o]}+\sigma(g_i(x))} \right] \tag{32}$$

for some real valued constants $b^{[o]}, w_i^{[o]}$ for $i \in [1,n]$. From equation (32) it follows that

$$N_d'(x) = \frac{\sum_{i=1}^{n} e^{w_i^{[o]}+\sigma(g_i(x))} \sigma'(g_i(x)) g_i'(x)}{e^{b^{[o]}} + \sum_{i=1}^{n} e^{w_i^{[o]}+\sigma(g_i(x))}} \tag{33}$$

From the inductive hypothesis, $g_i'(x) \geq 0$. Therefore $N_d'(x) \geq 0$. $N_d'(x) < 1$ follows from the following inequality which is trivially true since $g_i'(x) < 1$,

$$e^{w_i^{[o]}+\sigma(g_i(x))} \sigma'(g_i(x)) g_i'(x) < e^{w_i^{[o]}+\sigma(g_i(x))} \tag{34}$$

This completes the proof. □

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[3] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, and O. Ronneberger, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[6] S. Kundu, M. Nazemi, M. Pedram, K. M. Chugg, and P. A. Beerel, "Pre-defined sparsity for low-complexity convolutional neural networks," *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 1045–1058, 2020.

[7] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018.

[8] K. Roy and A. J. . P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[9] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[10] V. Charisopoulos and P. Maragos, "Morphological perceptrons: geometry and training algorithms," in *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*. Springer, 2017, pp. 3–15.

[11] P. Sussner, "Morphological perceptron learning," in *Proceedings of the 1998 IEEE International Symposium on Intelligent Control (ISIC) held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA) Intell.* IEEE, 1998, pp. 477–482.

[12] G. X. Ritter and P. Sussner, "An introduction to morphological neural networks," in *Proceedings of 13th International Conference on Pattern Recognition*, vol. 4. IEEE, 1996, pp. 709–717.

[13] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *CoRR*, vol. abs/1603.01025, 2016.

[14] A. Sanyal, P. A. Beerel, and K. M. Chugg, "Neural network training with approximate logarithmic computations," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 3122–3126.

[15] P. Robertson, E. Villebrum, and P. Hoeher, "A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain," in *Proceeding, IEEE International Conference on Communications*, Seattle, WA, 1995, pp. 1009–1013.

[16] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural networks*, vol. 3, no. 5, pp. 551–560, 1990.

[17] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[18] M. Arnold, E. Chester, and C. Johnson, "Training neural nets using only an approximate tableless lns alu," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020, pp. 69–72.

[19] D. Yarotsky, "Error bounds for approximations with deep relu networks," *Neural Networks*, vol. 94, pp. 103–114, 2017.

[20] Q. Wang, M. A. Powell, A. Geisa, E. Bridgeford, and J. T. Vogelstein, "Why do networks need negative weights?" 2022. [Online]. Available: https://arxiv.org/abs/2208.03211