

Interval Universal Approximation for Neural Networks

ZI WANG, University of Wisconsin–Madison

AWS ALBARGHOUTHI, University of Wisconsin–Madison

GAUTAM PRAKRIYA, Chinese University of Hong Kong

SOMESH JHA, University of Wisconsin–Madison

To verify safety and robustness of neural networks, researchers have successfully applied *abstract interpretation*, primarily using the interval abstract domain. In this paper, we study the theoretical *power and limits* of the interval domain for neural-network verification.

First, we introduce the *interval universal approximation* (IUA) theorem. IUA shows that neural networks not only can approximate any continuous function f (universal approximation) as we have known for decades, *but* we can find a neural network, using any well-behaved activation function, whose interval bounds are an arbitrarily close approximation of the set semantics of f (the result of applying f to a set of inputs). We call this notion of approximation *interval approximation*. Our theorem generalizes the recent result of Baader et al. [2020] from ReLUs to a rich class of activation functions that we call *squashable functions*. Additionally, the IUA theorem implies that we can always construct provably robust neural networks under ℓ_∞ -norm using almost any practical activation function.

Second, we study the computational complexity of constructing neural networks that are amenable to precise interval analysis. This is a crucial question, as our constructive proof of IUA is exponential in the size of the approximation domain. We boil this question down to the problem of approximating the range of a neural network with squashable activation functions. We show that the range approximation problem (RA) is a Δ_2 -intermediate problem, which is strictly harder than NP-complete problems, assuming $\text{coNP} \not\subseteq \text{NP}$. As a result, *IUA is an inherently hard problem*: No matter what abstract domain or computational tools we consider to achieve interval approximation, there is no efficient construction of such a universal approximator. This implies that it is hard to construct a provably robust network, even if we have a robust network to start with.

1 INTRODUCTION

Neural networks and approximation. Over the past decade, machine learning with neural networks has revolutionized a vast array of tasks—from computer vision [Krizhevsky et al. 2012], to natural-language processing [Mikolov et al. 2013], to program-analysis tasks [Raychev et al. 2015], and beyond. While these advances are recent, it has been well-known that neural networks are a powerful class of models: The *universal approximation theorem* [Cybenko 1989; Hornik et al. 1989] states that neural networks can approximate any continuous function with arbitrary precision. Moreover, we only need a single hidden layer of neurons to realize this theorem. By adding more neurons, one gets a more and more precise approximation. The intuition is that each neuron can encode a step function. So, by adding more neurons, one gets a finer-grained, step-like approximation of a continuous function (see Nielsen [2015, Ch.4] for an interactive visualization).

Abstract interpretation of neural networks. With the wide adoption of neural networks, new safety and security concerns arose. The most prominent property of study has been *robustness* [Goodfellow et al. 2015]: small perturbations to the input of a network should not change the prediction. For example, a small change to an image of a stop sign should not cause a classifier to think it is a speed-limit sign. A number of researchers have proposed the use of *abstract*

Authors' addresses: Zi Wang, University of Wisconsin–Madison, zw@cs.wisc.edu; Aws Albarghouthi, University of Wisconsin–Madison, aws@cs.wisc.edu; Gautam Prakriya, Chinese University of Hong Kong, gautamprakriya@gmail.com; Somesh Jha, University of Wisconsin–Madison, jha@cs.wisc.edu.

2018. 2475-1421/2018/1-ART1 \$15.00

<https://doi.org/>

IUA theorem (semi-formally): For a function $f : \mathbb{R}^m \rightarrow \mathbb{R}$ that we wish to approximate and error $\delta > 0$, there is a neural network N that has the following behavior:

Let $B \subset \mathbb{R}^m$ be a hyperrectangle (box) in Euclidean space. The red interval (top) is the tightest interval that contains all outputs of f when applied to elements of the set B .

If we abstractly interpret N on the box B , we may get the black interval (bottom) $N^\#(B)$, whose lower/upper bounds are up to δ away from those of the red interval. Note that $N^\#(B)$ may not necessarily subsume the top interval, since N is an approximation of f .

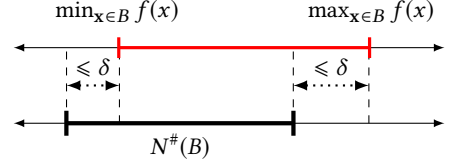


Fig. 1. Illustration and semi-formal statement of the interval universal approximation (IUA) theorem (Right is adapted from Baader et al. [2020])

interpretation [Cousot and Cousot 1977] techniques to prove robustness of neural networks to small perturbations [Anderson et al. 2019; Gehr et al. 2018; Wang et al. 2018] and to train robust models [Gowal et al. 2018; Huang et al. 2019; Mirman et al. 2018].

Suppose we want to verify robustness of a neural network to small changes in the brightness of an image. We can represent a large set of images, with varying brightness, as an element of some abstract domain, and propagate it through the network, effectively “executing” the network on an intractably large number of images. If all images lead to the same prediction, then we have a proof that the network is robust on the original image.

The simplest abstract domain that leads to practical verification results in this setting is the interval domain. In our example above, if each pixel in a monochrome image is a real number r , then the pixel can be represented as an interval $[r - \epsilon, r + \epsilon]$, where ϵ denotes the range of brightness we wish to be robust on. Then, the *box* representing the interval of each pixel is propagated through the network using interval arithmetic operations and other custom abstract transformers.

The power of the interval domain. The interval abstract domain has been successfully used for verifying properties of neural networks for image classification [Gehr et al. 2018; Gowal et al. 2018], natural-language processing [Huang et al. 2019], as well as cyber-physical systems [Wang et al. 2018]. Why does the interval domain work for verifying neural networks?

In investigating this question, Baader et al. [2020] demonstrated a surprising connection between the universal approximation theorem and interval-based verification. Their theorem states that not only can neural networks approximate any function f , but we can find a neural network, using *rectified linear unit* (ReLU) activation functions [Nair and Hinton 2010], whose interval abstract interpretation is arbitrarily close to the *collecting* (or set) semantics of f .

Interval universal approximation theorem. In this paper, our first goal is to deepen our understanding of the power of interval analysis of neural networks, broadly construed. Specifically, we set out to answer the following question:

Can we always construct a neural network, with any activation function, whose interval abstract interpretation is arbitrarily close to the *collecting* (or set) semantics of f ?

The theorem of Baader et al. [2020] is restricted to networks that use ReLU activations. In this work, we generalize the result of Baader et al. [2020] to neural networks that use arbitrary well-behaved activation functions. Specifically, we prove what we call the *interval universal approximation*

theorem, or IUA theorem for short: Let f be the function we wish to approximate, and let $\delta > 0$ be the tolerated error. Then, there exists a neural network N , built using *any* activation function, such that for any box of inputs B , the abstract interpretation of N on B is δ close to the collecting semantics of f over B . If the box of inputs is a single point in Euclidean space, then the IUA theorem reduces to the universal approximation theorem; thus, IUA generalizes universal approximation. *The IUA theorem is illustrated in more detail in Fig. 1.*

We define a rich class of activation functions, which we call *squashable functions*, for which our IUA theorem holds. This class includes popular activation functions, like ReLU, sigmoid, tanh, ELU, and other activation functions that have been shown to be useful for training robust neural networks [Xie et al. 2020]. The key idea behind squashable activation functions is that they have left and right limits (or we can use them to construct functions with limits); we exploit limits to approximate step functions, and therefore construct step-like approximations of f , while controlling approximation error δ .

Existence of provably robust networks. While our results are theoretical in nature, they shed light on the existence of provably correct neural networks. Suppose there is some ideal robust image classifier f using the ℓ_∞ -norm, which is typically used to define a set of images in the neighborhood of a given image. The classical universal approximation theorem tells us that, for any desired precision, there is a neural network that can approximate f . The IUA theorem further tells us that there exists a neural network for which we can automatically construct proofs of robustness using the interval domain. In addition, this neural network can be built using almost any activation function in the literature, and more.

Hardness of range approximation. Our proof of IUA, like that of Baader et al. [2020], is constructive. Given f and δ , one can construct a neural network that δ -interval approximates f . However, the constructions are exponential in the size of the function's domain. A key open problem is whether there is an efficient construction of such neural networks; therefore, the second question we set out to answer in this paper is

Can we efficiently build an interval universal approximator for any continuous function f ?

We answer this question by boiling it down to studying the hardness of what we call the *range approximation* (RA) problem: Given a function f , how hard it is to approximate the range of f . Specifically, we consider the case where f is given as a neural network N with domain $[0, 1]^m$ and codomain $[0, 1]$, and our goal is to approximate the range of N with tolerance δ . We show a dichotomy result: if $\delta \geq 1/2$, then this is a trivial task; if $\delta < 1/2$, then this is a Δ_2 -intermediate (Definition 5.6) problem, where Δ_2 is the smallest class in the polynomial hierarchy that contains both the NP and coNP classes. As a consequence, there is no efficient construction of the interval universal approximating neural network, and the verification of robustness using the interval domain is hard. If one can approximate the collecting semantics of a neural network using the interval domain as required for verifying robustness, then one can immediately approximate the range of the network.

Contributions. Our contributions can be summarized as follows:

- (1) We characterize a rich class of activation functions, which we call *squashable functions*, that includes most activation functions used in neural networks (ReLU, sigmoid, tanh, ELU, etc.). We show that squashable functions can arbitrarily approximate step functions. Since neural networks using step functions can encode Boolean formulas, interpreting any activation function as a squashable function provides a unified view of neural networks. We believe

that it will benefit future researchers in understanding the theoretical and formal properties of neural networks. (Section 2)

- (2) We prove the *interval universal approximation* (IUA) theorem: Given a continuous f over a compact domain, one can always construct a neural network N with *any* squashable function whose interval semantics is arbitrarily close to the set semantics of f . Our result generalizes the work of Baader et al. [2020], which is specialized for ReLU activations. Our proof follows the general framework put forth by Baader et al. [2020], which can be viewed as a careful design of summation of indicator functions. Baader et al. [2020] use a construction from He et al. [2018] to construct the min function with ReLU units. We present a smaller construction that is simpler to analyze and applies to *any* squashable activation function. (Sections 3 and 4)
- (3) We demonstrate that the IUA theorem implies the existence of provably robust neural networks for any classification task at hand that has a robust solution f . Specifically, there exists a neural network for which we can automatically construct proofs of robustness using the interval domain and whose classifications match those of f . In addition, this neural network can be built using any squashable activation. (Section 7)
- (4) We study the hardness of building neural networks for IUA. We show a dichotomy result, that it is either trivial or Δ_2 -intermediate to approximate the range of a neural network that is polynomial-time executable. As a consequence, there is no efficient construction of the interval universal approximator. To our best knowledge, this is the first work to classify the complexity of a verification task of neural networks.¹ (Sections 5 and 6)

2 SQUASHABLE ACTIVATION FUNCTIONS

In this section, we define squashable functions, and how they can be used to build other functions that are essential in our analysis of neural networks.

2.1 Neural Networks and Squashable Activation Functions

Neural networks. A neural network in our setting is a function in $\mathbb{R}^m \rightarrow \mathbb{R}$, where m is the number of inputs to the network. We will take a general view and define a network N following a simple grammar, a composition of primitive arithmetic operations and *activation functions*. Throughout, we will use $\mathbf{x} \in \mathbb{R}^m$ to denote a vector, and use x_1, \dots, x_m to denote the m elements of \mathbf{x} .

Definition 2.1 (Neural network grammar). Let \mathbf{x} be the input to the neural network. A neural network N is defined as follows

$$\begin{array}{l}
 N \quad := \quad c \\
 \quad \quad | \quad x_i \\
 \quad \quad | \quad N_1 + N_2 \\
 \quad \quad | \quad c * N_1 \\
 \quad \quad | \quad t(N_1)
 \end{array}$$

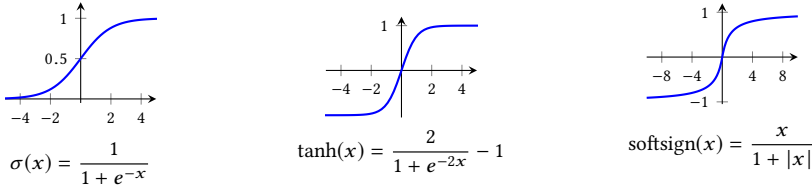
where $c \in \mathbb{R}$, x_i is the i th input to the network, and $t : \mathbb{R} \rightarrow \mathbb{R}$ is an *activation function*. Whenever we discuss neural networks, we will fix a single activation function t to be used in the grammar. ■

This grammar is rich enough to encode standard feed-forward neural networks, convolutional neural networks, and other non-recurrent architectures.

Activation Functions. In Fig. 2, we define and plot a number of popular activation functions, and other more recent ones: *sigmoid*, *tanh*, *rectified linear units* (ReLU) [Nair and Hinton 2010], *exponential linear unit* (ELU) [Clevert et al. 2016], *softplus* [Glorot et al. 2011], *softsign* [Bergstra et al. 2009], and *smooth ReLU* [Xie et al. 2020].

¹Katz et al. [2017] and Weng et al. [2018] study the complexity of bug finding (falsification) instead of verification.

Activation functions that satisfy Eq. (1)



Activation functions that do not directly satisfy Eq. (1)

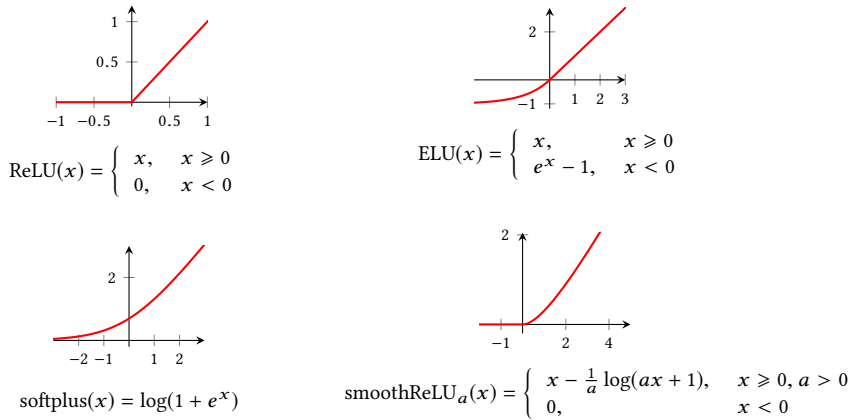
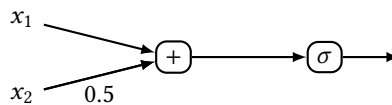


Fig. 2. Example activation functions. Smooth ReLU (smoothReLU_a) is parameterized by $a > 0$ ($a = 1$ is plotted).

Example 2.2. Consider the following simple neural network with 2-dimensional input $\mathbf{x} = (x_1, x_2)$ and a sigmoid activation function: $N(\mathbf{x}) = \sigma(x_1 + 0.5x_2)$. This is typically depicted as:



Observe that the coefficient of x_2 is shown on the arrow. ■

Squashable activation functions. We provide the definitions of activation functions above to ground our discussion. Our results, however, are more general: they apply to a general class of activation functions that we will call *squashable* activation functions:

Definition 2.3 (Squashable activation functions). $t : \mathbb{R} \rightarrow \mathbb{R}$ is squashable iff

(1) there is $a < b \in \mathbb{R}$ such that

$$\lim_{x \rightarrow -\infty} t(x) = a, \quad \lim_{x \rightarrow \infty} t(x) = b, \quad \text{and} \quad \forall x < y. t(x) \leq t(y) \quad (1)$$

(2) or a function $t' : \mathbb{R} \rightarrow \mathbb{R}$ that satisfies Eq. (1) and can be expressed using the grammar in Definition 2.1 with copies of t . For example, $t'(x) = t(2 * t(x) - t(x + 10))$.

In other words, squashable functions are the smallest set of functions that can use the grammar in Definition 2.1 to build a function that satisfies Eq. (1). ■

Informally, an activation function is in this class if we can use it to construct a monotonically increasing function that has limits in the left and right directions, $-\infty$ and ∞ .² Squashable activation functions extend the *squashing* functions used by Hornik et al. [1989]. All of the activation functions we have defined in Fig. 2 are squashable.

Fig. 2 (top, blue) shows all activation functions that satisfy Eq. (1), and are therefore squashable. For example, sigmoid and tanh easily satisfy Eq. (1): both have limits and are monotonically increasing. What about activation functions like ReLU, ELU, etc., shown in Fig. 2 (bottom, red)? It is easy to see that they do not satisfy Eq. (1): none of them have a right limit. However, by point (2) of Definition 2.3, given an activation function t , if we can construct a new activation function t' that is squashable, using the operations in the grammar in Definition 2.1, then t is squashable. In the following proposition, we give a general and simple construction that works for all activation functions in Fig. 2 (bottom, red).

PROPOSITION 2.4 (SQUASHABLE ACTIVATIONS). *Let*

$$t \in \{\text{ReLU}, \text{softplus}, \text{smoothReLU}_a, \text{ELU}\}$$

The function $t'(x) = t(1 - t(-x))$ satisfies Eq. (1). Therefore, ReLU, softplus, Smooth ReLU, and ELU, are squashable.

PROOF. It is easy to see that all the activation functions t are monotonically increasing with

$$\lim_{x \rightarrow -\infty} t(x) = l \quad \text{and} \quad \lim_{x \rightarrow \infty} t(x) = \infty.$$

for some $l \in \mathbb{R}$.

Because t is increasing, $t(-x)$ and $t(1-x)$ are both decreasing; thus, their composition $t(1-t(-x))$ is increasing.

$$\lim_{x \rightarrow -\infty} t(1 - t(-x)) = t\left(\lim_{x \rightarrow -\infty} (1 - t(-x))\right) = l$$

$$\lim_{x \rightarrow \infty} t(1 - t(-x)) = t(1 - \lim_{x \rightarrow \infty} t(-x)) = t(1 - l)$$

ReLU. : $l = 0$, and $t(1 - l) = \text{ReLU}(1 - 0) = 1$.

ELU. : $l = -1$, and $t(1 - l) = \text{ELU}(2) = 2$.

softplus. : $l = 0$, and $t(1 - l) = \text{softplus}(1) = \log(1 + e)$.

smoothReLU. : $l = 0$, and $t(1 - l) = \text{smoothReLU}_a(1) = 1 - \frac{1}{a} \log(a + 1)$. (Note that $\frac{1}{a} \log(a + 1) < 1$ for $a \neq 0$). \square

Throughout this paper, we will work with neural networks with squashable activation functions. Proposition 2.4 guarantees that our results are general enough to account for many different neural networks, including ReLU networks.

Example 2.5. Fig. 3 shows $t(1 - t(-x))$, for $t = \text{ReLU}$ and $t = \text{softplus}$. Both have left/right limits and are monotonic. Thus, they satisfy Eq. (1) and therefore ReLU and softplus are squashable. \blacksquare

²In our construction and proof, we do not need the function to be monotonic; however, in practice, most activation functions are monotonic and abstractly interpreting arbitrary functions is impractical.



Fig. 3. Two activation functions after applying construction in Proposition 2.4. Observe that the resulting function satisfies Eq. (1), and therefore ReLU and softplus are squashable.

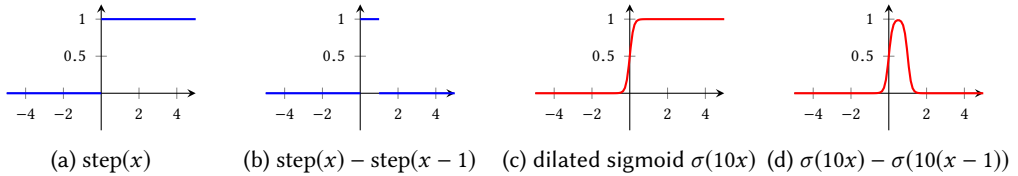


Fig. 4. Approximations of step function and indicator function

2.2 Squashable-Function Constructions

In this section, we will show some constructions using squashable and step functions. This is a key idea of the whole paper, and essential for proving the IUA and the hardness of range approximation theorems. As we will demonstrate in the subsequent sections, we will use squashable functions to approximate some gadgets that are fundamental in mathematics and complexity theory. We believe that these constructions are important in understanding the computational and formal properties of neural networks in the future.

Remark. In reality, ReLU is believed to improve neural network training because it prevents gradient vanishing during training [Nair and Hinton 2010]. Our treatment to ReLU, i.e., as a squashable function, allows gradient vanishing. However, we are not concerned about neural network training; instead, we work on function approximations, and our goal is an *existence* proof. Using the indicator functions is a classical idea in approximation theory, allowing us to approximate each piece of a function. These constructions are useful for investigating the existence of function approximators and worst-case behaviors. Similarly, the approximation proof in Baader et al. [2020]; Lin and Jegelka [2018]; Nielsen [2015] also used bump/indicator functions built by ReLU, via different approaches from what we did, which also potentially introduces gradient vanishing.

Step Function. A step function is

$$\text{step}(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

One can view the step function as the indication of whether x is a positive number. The step function can be used to build indicator functions, which is a fundamental tool in mathematical analysis. For example, the standard way of defining integration with respect to the probability measure is using the summation of indicator functions [Durrett 2010]. Also, neural networks with step functions as the activation function can encode Boolean formula as we will show later.

Squashable Function. Squashable functions do not have an explicit formula, as it is a very expressive class of functions. All we know is that it is monotonic and has both left and right limits (see Definition 2.3). Proving the properties of neural network with squashable functions might appear a challenging task.

However, the key observation is that if we dilate t properly, i.e., multiply the input with a large number μ to get $t(\mu x)$, we will obtain an approximation of the step function. See Figs. 4a and 4c on how one can use the sigmoid function to approximate the step function.

Indicator Function. An indicator function associated with a set $S \subset \mathbb{R}^m$ is defined as

$$\text{ind}_S = \begin{cases} 1, & \mathbf{x} \in S \\ 0, & \mathbf{x} \notin S \end{cases}$$

Note how the value is 1 if \mathbf{x} is in the set S , and 0 otherwise.

We can use step functions to build indicator functions. For example,

$$\text{step}(x) - \text{step}(x - 1)$$

returns 1 for $x \in (0, 1]$, and 0 otherwise. See Figs. 4b and 4d for illustrations of the (approximating) indicator function of $(0, 1]$.

Encoding Boolean Formula. A Boolean formula is a composition of operators \neg , \wedge and \vee and variables that take values $\{0, 1\}$. This is one of the most fundamental objects in logic and computer science, and has been extensively studied. If the Boolean formula is expressible using neural networks, then we can understand the properties of the neural networks from their Boolean formula counterparts. To simulate a Boolean formula, for each variable, we can build an input node corresponding to the variable. We only need to encode the logical operators.

- (1) For $\neg\phi_x$, we only need to use $1 - X$, where X is the neural network node corresponds to the expression ϕ_x .
- (2) For $\phi_x \wedge \phi_y$, we can use $\text{step}(X+Y-1.5)$. For X and Y that takes values in $\{0, 1\}$, $\text{step}(X+Y-1.5)$ evaluates to 1 only when both X and Y are 1.
- (3) For $\phi_x \vee \phi_y$, we can use $\text{step}(X+Y-0.5)$. For X and Y that takes values in $\{0, 1\}$, $\text{step}(X+Y-0.5)$ evaluates to 0 only when both X and Y are 0.

Note that the constants used in the encodings are not unique. In the appendix, we will give an inductive proof on the correctness of this encoding.

One can then build a neural network that encodes a Boolean formula recursively according to the syntactic composition of the formula. In Section 5, we will consider Boolean formulas of special forms, i.e., 3CNF (conjunction normal form) and 3DNF (disjunction normal form). We will present encodings of the 3CNF and 3DNF formulas using neural networks with squashable functions, and the construction essentially captures the computation of corresponding logical operators. (Section 6)

Remark. As we shall see later, the step-function formulation serves as the intuition for understanding the neural network with squashable activations. However, because we do not have the perfect step function as the activation, and the values are continuous rather than $\{0, 1\}$, to rigorously prove the results, we need to carefully control the imprecision introduced by the approximation of squashable functions, and the construction that works for discrete values might not work for continuous values directly. Nevertheless, one can expect the formal property of neural networks with squashable activations will not be fundamentally different from its Boolean formula or step-function neural network counterparts. Moreover, the step function gadget can still guide the design for the network with squashable functions. In fact, we will show that the complexity result in Katz et al. [2017]—that it is NP-hard to falsify correctness³ of a neural network—can be easily proved using the squashable function idea, as a corollary of the result we will present (Corollary 6.8).

³Katz et al. [2017] prove complexity of falsifying, rather than verifying, correctness properties presented as conjunctions of linear inequalities over inputs and outputs of a network.

3 THE INTERVAL UNIVERSAL APPROXIMATION THEOREM

In this section, we present the interval universal approximation (IUA) theorem. We begin with background on abstract interpretation for neural networks.

3.1 Interval Abstraction

We will now define the interval abstract domain and use it to abstractly interpret the semantics of neural networks.

Set semantics. Given a function $f : \mathbb{R}^m \rightarrow \mathbb{R}$, we will use $f^s : \mathcal{P}(\mathbb{R}^m) \rightarrow \mathcal{P}(\mathbb{R})$ to define its *collecting* (or *set*) semantics. Formally, given a set $S \subseteq \mathbb{R}^m$,

$$f^s(S) = \{f(\mathbf{x}) \mid \mathbf{x} \in S\}$$

Henceforth, we will simply use $f(S)$ to denote the collecting semantics version, $f^s(S)$, as it will be clear from context that we are applying the function f to a set.

The interval abstract domain. Evaluating the set semantics on elements of $\mathcal{P}(\mathbb{R}^m)$, the *concrete domain*, is generally infeasible. The *abstract interpretation* framework [Cousot and Cousot 1977] enables constructing sound approximations of collecting semantics by restricting operations to sets of a certain shape—*abstract domains*. In this work, we consider the well-known *interval* abstract domain, where the kinds of sets are limited to *boxes* in \mathbb{R}^m . An m -dimensional box B is a tuple of intervals, defining the lower and upper bounds of each dimension:

$$\langle [l_1, u_1], \dots, [l_m, u_m] \rangle$$

where $l_i, u_i \in \mathbb{R}$ (we do not need to consider unbounded boxes because we only consider bounded input space, e.g., $u_i = \infty$).

The *abstraction function* α transforms an element of the concrete domain to a box. Let $S \in \mathcal{P}(\mathbb{R}^m)$.

$$\alpha(S) = \langle [\inf S_i, \sup S_i]_{i=1}^m \rangle$$

where $S_i = \{x_i \mid \mathbf{x} \in S\}$ and x_i refers to the i th element of \mathbf{x} . In other words, S_i is a projection of vectors in S onto their i th element.

The *concretization function* γ transforms boxes into their concrete domain counterparts.

$$\gamma(\langle [l_1, u_1], \dots, [l_m, u_m] \rangle) = \{\mathbf{x} \in \mathbb{R}^m \mid l_i \leq x_i \leq u_i\}$$

For clarity of presentation, we will often drop the use of the concretization operator, and treat a box B as a subset of \mathbb{R}^m .

Abstract transformers for neural operations. We can now define abstract versions of the operations of a neural network. We start with primitive arithmetic operations, where we use superscript $\#$ to denote the abstract transformer. Since all of our operations are over scalars, we define arithmetic abstract transformers over 1-dimensional boxes.

Definition 3.1 (Arithmetic abstract transformers). Let B be an m -dimensional box input to the neural network. We follow the grammar in Definition 2.1 to define the abstract transformers.

$$\begin{aligned} c^\# &= [c, c] \\ x_i^\# &= [l_i, u_i], \quad \text{where } l_i, u_i \text{ are the } i\text{th lower and upper bounds of } B \\ [l_1, u_1] +^\# [l_2, u_2] &= [l_1 + l_2, u_1 + u_2] \\ [c, c] *^\# [l, u] &= [\min(c * l, c * u), \max(c * l, c * u)] \end{aligned}$$

■

We also need to define abstract transformers for activation functions. We give a general definition that works for any function satisfying Eq. (1).

Definition 3.2 (Abstract transformer for activations [Gehr et al. 2018]). Let $B = \langle [l, u] \rangle$ be a 1-dimensional box.

$$t^\#(B) = \left\langle \left[\min_{l \leq x \leq u} t(x), \max_{l \leq x \leq u} t(x) \right] \right\rangle$$

Intuitively, we simply take the minimum and maximum values of t over the interval defined by the box B . This may not generally be easy to compute, as it involves solving a constrained optimization problem; however, for monotonically increasing activation functions (all activation functions in Fig. 2), we can simplify the definition as follows:

$$t^\#(B) = \langle [t(l), t(u)] \rangle$$

where we only apply t to the lower and upper bounds of B , since by monotonicity we know that $t(\gamma(B)) \subseteq [t(l), t(u)]$. ■

Example 3.3. Recall the neural network $N(x) = \sigma(x_1 + 0.5x_2)$, defined in Example 2.2. Suppose we want to abstractly interpret it on the 2-dimensional box $B = \langle [0, 1], [0.6, 1] \rangle$, i.e., the set of all values where $x_1 \in [0, 1]$ and $x_2 \in [0.6, 1]$.

$$\begin{aligned} N^\#(B) &= \sigma^\#([0, 1] +^\# [0.5, 0.5] *^\# [0.6, 1]) \\ &= \sigma^\#([0, 1] +^\# [0.3, 0.5]) && \text{(evaluate } *^\# \text{)} \\ &= \sigma^\#([0.3, 1.5]) && \text{(evaluate } +^\# \text{)} \\ &= [\sigma(0.3), \sigma(1.5)] && \text{(evaluate } \sigma^\#; \sigma \text{ is monotonic)} \end{aligned}$$

Soundness. Finally, we shall use $N^\#$ to denote the abstract version of a neural network N . The following theorem establishes soundness of our abstract transformers.

THEOREM 3.4 (SOUNDNESS OF ABSTRACT TRANSFORMERS). *Let $N : S \rightarrow \mathbb{R}$ be a neural network with domain $S \subseteq \mathbb{R}^m$. Let B be an m -dimensional box such that $\gamma(B) \subseteq S$. Then, $N(\gamma(B)) \subseteq \gamma(N^\#(B))$.*

Incompleteness. Even though abstract interpretation is sound, it is not necessarily complete, i.e., the robustness of some inputs might not be certified by abstract interpretation even though they are robust. This is because abstract interpretation could potentially lose precision, and the certification result for some robust inputs becomes unknown. Consider the following example,

$$y = x - x \quad \text{and} \quad x^\# = [0, 1]$$

The abstract interpretation rule gives

$$y^\# = x^\# -^\# x^\# = [0, 1] -^\# [0, 1] = [-1, 1]$$

whereas $y = 0$ is always true.

The soundness of abstract interpretation enables the verification of robustness and other correctness properties. However, because abstract interpretation is not necessarily complete, and therefore for some correctness properties we may fail to construct proofs. As we shall see, the interval universal approximation theorem that we will present in Section 3.2 implies that in fact it is possible to verify certain robustness definitions (ℓ_∞) using interval abstract interpretation.

3.2 The Interval Universal Approximation Theorem

In this section, we state the *interval universal approximation* (IUA) theorem.

Interval approximation. We begin by defining what it means to approximate a function using a neural network. We assume some fixed continuous function $f : C \rightarrow \mathbb{R}$, with a compact domain $C \subseteq \mathbb{R}^m$, that we wish to approximate.

Definition 3.5 (δ -approximation). Let $\delta > 0$. A neural network N δ -approximates f iff for all $\mathbf{x} \in C$, we have $f(\mathbf{x}) - \delta \leq N(\mathbf{x}) \leq f(\mathbf{x}) + \delta$. ■

We now *generalize* this point-wise approximation definition to elements of our abstract domain.

Definition 3.6 (δ -interval approximation). Let $\delta > 0$. A neural network N δ -interval approximates f iff for every box $B \subseteq C$, we have

$$[l + \delta, u - \delta] \subseteq N^\#(B) \subseteq [l - \delta, u + \delta]$$

where $l = \min f(B)$ and $u = \max f(B)$. ■

Informally, δ -interval approximation says that the box output of abstract interpretation $N^\#(B)$ is up to δ away from the tightest bounding box around the collecting semantics $f(B)$. Revisit Fig. 1 from Section 1 for an illustration of δ -interval approximation. Observe that δ -approximation is a special case of δ -interval approximation, when the box B is a point in C , i.e., $\gamma(B)$ is a singleton set.

Interval universal approximation (IUA). We now state the IUA theorem:

THEOREM 3.7 (INTERVAL UNIVERSAL APPROXIMATION). *Let $f : C \rightarrow \mathbb{R}$ be a continuous function on a compact domain $C \subset \mathbb{R}^m$. Let t be a squashable activation function. For all $\delta > 0$, there exists a neural network N , using only activations t , that δ -interval approximates f .*

Informally, the theorem says that we can always find a neural network whose abstract interpretation is arbitrarily close to the collecting semantics of the approximated function. Note also that there exists such a neural network for any fixed squashable activation function t .

As we discuss in Section 7, the IUA theorem has very exciting implications: We can show that one can always construct provably *robust* neural networks using any squashable activation function (Theorem 7.4). The robustness property, which states that small perturbations in the input result in the same classification by a neural network, has been heavily studied recently, and the interval domain has been used to prove robustness in a range of domains [Anderson et al. 2019; Gehr et al. 2018; Wang et al. 2018]. Our result hints at a very close theoretical connection between robust neural networks and proofs using interval-based abstract interpretation.

In the supplementary materials, we give a generalization of the IUA theorem to functions and networks with multiple outputs. In reality, neural networks are used as multi-label classifiers, therefore, the multiple-output case demonstrates this real-world scenerio because each output corresponds to a predicted label. This generalization shows that we can always construct provably robust neural networks for the real-world use case.

4 PROOF OF IUA THEOREM

We will show the IUA theorem, as stated in Theorem 3.7. Our proof uses the framework of Baader et al. [2020], which is a delicate design of a summation of indicator functions. Though constructing indicator functions is a classical idea in approximation theory (see Nielsen [2015, Ch.4] for an interactive visualization), we are working in interval approximation, which is harder than pointwise approximation because interval approximation implies pointwise approximation. The interval approximation construction differs from the usual pointwise approximation one in the following two ways:

- (1) In the pointwise case, we only need to grid the input domain. As long as one can approximate the target function within each grid using an indicator function, the pointwise approximation is achieved. However, this does not work in the interval case because the input can be a box

which might span over several grids. Baader et al. [2020] discovered an ingenious observation that if one *slices* the domain of a function, and approximate each slice, then the usual indicator approximation works because one can control the loss of precision of each slice.

- (2) The interval semantics and the pointwise semantics can be vastly different, therefore, the pointwise indicator function might not remain an indicator in the interval semantics. As we demonstrate in Theorem 5.8, it is in general a hard task to build a network whose interval semantics approximate another network's set semantics. Baader et al. [2020] use a construction from He et al. [2018] to build the indicator function from the ReLU units, and carefully analyze that this construction is indeed an indicator under the interval semantics, which is in fact among the most technical and involved parts. We instead use ideas of squashable functions introduced in Section 2 to approximate the indicator function. This results in a technique that is simpler to analyze and also works for a larger set of functions, including ReLU.

To summarize, we extend the IUA restricted to ReLU-network shown in Baader et al. [2020] to a more general class of neural networks, and provide a simple-to-analyze indicator construction from squashable functions. If we only consider ReLU network, our construction will achieve a linear factor reduction in the usage of activation units to build the approximation network compared to Baader et al. [2020].

4.1 Overview of Complete Proof of IUA

We will give a brief overview of the whole proof, following the technique of Baader et al. [2020] for ReLU networks. Because we use an arbitrary squashable function rather than ReLU to approximate the step function, this introduces extra imprecision in comparison with ReLUs. In what follows, we outline on how to build the network N that satisfies the IUA theorem. We provide the detailed analysis in the supplementary materials.

Slicing f . Let $f : C \rightarrow \mathbb{R}$ be the continuous function we need to approximate, and δ be the approximation tolerance, as per IUA theorem statement (Theorem 3.7). Assume $\min f(C) = 0$.⁴ Let $u = \max f(C)$. In other words, the range of f is $[0, u]$.

Let $\tau = \frac{\delta}{3}$. We will decompose f into a sequence of *function slices* f_i , whose values are restricted to $[0, \tau]$. Let $K = \lfloor u/\tau \rfloor$. The sum of the sequence of function slices is f . The sequence of functions $f_i : C \rightarrow [0, \tau]$, for $i \in \{0, \dots, K\}$, is:

$$f_i(\mathbf{x}) = \begin{cases} f(\mathbf{x}) - i\tau, & i\tau < f(\mathbf{x}) \leq (i+1)\tau \\ 0, & f(\mathbf{x}) \leq i\tau \\ \tau, & (i+1)\tau < f(\mathbf{x}) \end{cases}$$

Notice that in Baader et al. [2020], the slicing size is $\delta/2$ instead of $\delta/3$. This is because squashable functions might introduce extra imprecision compared to ReLU. For example, in Fig. 3, in the ReLU case, we know $t(x) = 0$ when $x < -5$ but we only know $t(x) \approx 0$ when $x < -5$ in the softplus case. We thus need a finer function slicing to accommodate it. Recall that the goal is to show that in the final result, the interval semantics of the constructed neural network satisfy the inequality in Definition 3.6. From the detailed analysis we provide in the supplementary materials, we can find that $\delta/2$ leads to a sharp inequality in the final result for the ReLU case, and thus we have this finer ($\delta/3$ instead of $\delta/2$) slicing for general squashable functions.

Example 4.1. See Fig. 5 for slicing $f(x) = \sin(2x) + 1$ with $\delta = 1.2$. ■

⁴Otherwise, we can shift f such that $\min f(C) = 0$.

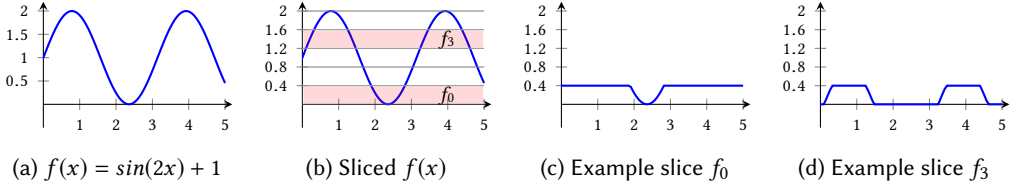


Fig. 5. Slicing $f(x) = \sin(2x) + 1$ with approximation tolerance $\delta = 1.2$.

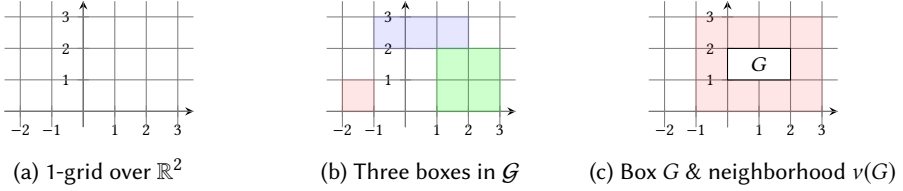


Fig. 6. A grid illustration

Approximating f_i . Fix $\epsilon > 0$. Consider a standard *grid* of vertices over a compact set C , where any two neighboring vertices are axis-aligned and of distance ϵ ; we will call this an ϵ -grid. Let $[a_1, b_1] \times \dots \times [a_m, b_m]$ be a box G on the grid, where $[a_i, b_i]$ is the range of G at dimension i . In other words, $b_i - a_i$ is a multiple of ϵ . Let \mathcal{G} be the set of boxes whose vertices are in the grid. The *neighborhood* $v(G)$ of G is $[a_1 - \epsilon, b_1 + \epsilon] \times \dots \times [a_m - \epsilon, b_m + \epsilon]$. Our goal is to construct an indicator function whose value is close to 1 within G , and close to 0 outside G 's neighborhood $v(G)$. The idea of using grid is similar to the nodal basis in He et al. [2018]. See Fig. 6 for an example of grid and boxes in the grid. Constructing the indicator function for boxes in the grid is essential for the proof, and we will provide the precise construction in Section 4.2. For now, let's assume we have the construction N_G (Eq. (3)) with $N_G(x) \approx 1$ in $x \in G$ and $N_G \approx 0$ for $x \notin v(G)$.

We will use the indicator approximation N_G to construct a neural network N_i that approximates f_i . Because C is compact, $|\mathcal{G}|$ is finite. Consider $\frac{1}{\tau} f_i(x)$; it is roughly similar to an indicator function for the set $S = \{x \in C \mid f(x) > (i+1)\tau\}$, i.e., indicating when $f(x)$ is greater than the upper bound of the i th slice. To approximate $\frac{1}{\tau} f_i(x)$, we will consider all boxes in \mathcal{G} that are subsets of S , and construct an indicator function to tell us whether an input \mathbf{x} is in those boxes. Let $\mathcal{G}_i = \{G \in \mathcal{G} \mid f(G) > (i+1)\tau\}$. Now construct $N_i(\mathbf{x})$ that approximates $\frac{1}{\tau} f_i(\mathbf{x})$ as

$$N_i(\mathbf{x}) = t \left(\mu \left(\sum_{G \in \mathcal{G}_i} N_G(\mathbf{x}) - 0.5 \right) \right).$$

Sum all N_i . Because $\sum_{i=0}^K f_i(\mathbf{x}) = f(\mathbf{x})$, and $N_i(\mathbf{x})$ approximates $\frac{1}{\tau} f_i(\mathbf{x})$, we define N as

$$N(\mathbf{x}) = \tau \sum_{i=0}^K N_i(\mathbf{x})$$

N δ -interval approximates f ; therefore, the IUA theorem holds.

4.2 Approximating Indicator Functions

In this section, we will give the precise construction of the approximating indicator function and the rigorous proof of correctness. Because we will grid the input space (recall the ϵ -grid defined in Section 4.1), we need an indicator function for each grid cell, i.e., high-dimensional box. We

start from building the one-dimensional indicator function and then use that to build the high-dimensional one. We will use the idea introduced in Section 2.2 on how to construct the indicator function from squashable functions.

4.2.1 One-dimensional indicator function. We will first show how to construct an indicator function for a 1-dimensional box, using a squashable activation function as we have seen in Section 2. The main challenge is choosing the dilation factor that results in small precision loss when abstractly interpreting the neural network.

By the IUA theorem statement, we are given some squashable activation function t . Without loss of generality, we make the following two assumptions about t :

- (1) We assume that t already satisfies Eq. (1) (Definition 2.3):

$$\lim_{x \rightarrow -\infty} t(x) = a \quad \text{and} \quad \lim_{x \rightarrow \infty} t(x) = b \quad \text{and} \quad \forall x \in \mathbb{R}. t(x) \in [a, b]$$

Otherwise, by Definition 2.3, we can use t to build a t' that satisfies Eq. (1).

- (2) We assume that the left and right limits of t are 0 and 1, respectively. (If not, we can apply an affine transformation to the results of t to make the left and right limits 0 and 1.)

Loss of precision from limits. The activation function t has limits at both sides, but the function might never reach the limit. For example, the right limit of the sigmoid function, σ , is 1, but $\forall x. \sigma(x) \neq 1$. This will lead to a loss of precision when we use t to model a step function. However, we can carefully apply mathematical analysis to rigorously bound this imprecision.

Dilation to approximate step function. We now discuss how to dilate t to get a step-function-like behavior. By definition of limit, we know the following lemma, which states that by sufficiently increasing the input of t , we can get θ close to the right limit of 1, and analogously for the left limit.

LEMMA 4.2. $\forall \theta > 0, \exists D > 0$ such that:

- (1) If $x \geq D$, then $t(x) \in (1 - \theta, 1]$.
- (2) If $x \leq -D$, then $t(x) \in [0, \theta)$.

Because the grid size is ϵ , we want the step-function approximation to achieve a transition from ≈ 0 to ≈ 1 within ϵ . Let μ be the dilation factor. Following Lemma 4.2, we would like the following:

- (1) if $x \geq 0.5\epsilon$, then $t(\mu x) \in (1 - \theta, 1]$;
- (2) if $x \leq -0.5\epsilon$, then $t(\mu x) \in [0, \theta)$.

From Lemma 4.2, we only need $\mu x > D$ when $x > 0.5\epsilon$; therefore, $\mu = 2D/\epsilon$ suffices as the dilation factor.

LEMMA 4.3. From Lemma 4.2, $\forall \theta > 0$, let $\mu = 2D/\epsilon$. The following is true:

- (1) if $x \geq 0.5\epsilon$, then $t(\mu x) \in (1 - \theta, 1]$;
- (2) if $x \leq -0.5\epsilon$, then $t(\mu x) \in [0, \theta)$.

Example 4.4. Fig. 7 illustrates the loss of precision θ incurred by our construction. ■

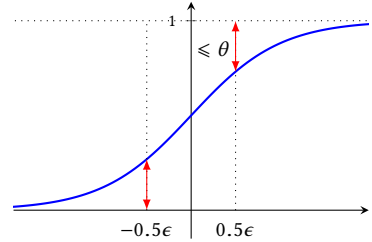


Fig. 7. Illustrating the loss of precision θ incurred through using a squashable activation to approximate a step function. The length of the red arrows is $\leq \theta$.

Indicator function on dimension i . Now that we have discussed how to approximate a step function, we are ready to show how to approximate an indicator function for one dimension of a box G in the grid.

Suppose the projection of a box G on dimension i is $[a_i, b_i]$. Because G is in the ϵ -grid, $b_i - a_i \geq \epsilon$; and the projection of neighborhood $\nu(G)$ on dimension i is $[a_i - \epsilon, b_i + \epsilon]$. We want to build an

indicator function that has value close to 1 on $[a_i, b_i]$, and value close to 0 on $\mathbb{R} \setminus [a_i - \epsilon, b_i + \epsilon]$. Notice how we may lose precision within the neighborhood of G ; this is expected, because our approximation may not be able to exactly tell if we are in G or its neighborhood.

Inspired by how to construct an indicator function from a step function, we will take the difference between two shifted step functions. Let

$$\hat{t}(x) = t(\mu(x + 0.5\epsilon - a_i)) - t(\mu(x - 0.5\epsilon - b_i)) \quad (2)$$

Properties of \hat{t} . The following lemmas show that \hat{t} roughly behaves like an indicator function: its value within a box's i th dimension $[a_i, b_i]$ is ≈ 1 ; its value outside of the neighborhood is ≈ 0 ; its value globally is bounded by 1. We will analyze the values of the two terms in \hat{t} .

The following lemma states that if x is within the box's i th dimension, then the first term is close to 1 and the second term is close to 0, resulting in $\hat{t}(x) \approx 1$.

LEMMA 4.5. *If $x \in [a_i, b_i]$, then the following is true:*

- (1) $t(\mu(x + 0.5\epsilon - a_i)) \in (1 - \theta, 1]$.
- (2) $t(\mu(x - 0.5\epsilon - b_i)) \in [0, \theta)$.

The next two lemmas state that if x is outside the neighborhood, then the two terms are similar, resulting in a $\hat{t}(x) \approx 0$.

LEMMA 4.6. *If $x \leq a_i - \epsilon$, then the following is true:*

- (1) $t(\mu(x + 0.5\epsilon - a_i)) \in [0, \theta)$.
- (2) $t(\mu(x - 0.5\epsilon - b_i)) \in [0, \theta)$.

LEMMA 4.7. *If $x \geq b_i + \epsilon$, then the following is true:*

- (1) $t(\mu(x + 0.5\epsilon - a_i)) \in (1 - \theta, 1]$.
- (2) $t(\mu(x - 0.5\epsilon - b_i)) \in (1 - \theta, 1]$.

Abstract precision of \hat{t} . We are now ready to prove properties about the abstract interpretation of our 1-dimensional indicator approximation, \hat{t} . The following lemma states that the abstract interpretation of \hat{t} , $\hat{t}^\#(B)$, is quite precise: if the 1-dimensional input box B is outside the neighborhood of G , on G 's i th dimension, then the output box is within θ from 0; if the input box B is within the i th dimension of G , then the output box is within 2θ from 1.

LEMMA 4.8. *For a 1-dimensional box B , the following is true:*

- (1) $\hat{t}^\#(B) \subset (-\infty, 1]$.
- (2) If $B \subseteq (-\infty, a_i - \epsilon]$ or $B \subseteq [b_i + \epsilon, \infty)$, then $\hat{t}^\#(B) \subseteq (-\theta, \theta)$.
- (3) If $B \subseteq [a_i, b_i]$, then $\hat{t}^\#(B) \subseteq (1 - 2\theta, 1]$.

4.2.2 Approximating an m -dimensional indicator. We saw how to construct an indicator approximation for a 1-dimensional box. We will now show how to construct an indicator function approximation N_G for an m -dimensional box.

Throughout, we assume a box $G = [a_1, b_1] \times \dots \times [a_m, b_m]$. So, if $\mathbf{x} \in G$, then $x_i \in [a_i, b_i]$ for all $i \in \{1, \dots, m\}$; if $\mathbf{x} \notin \nu(G)$, i.e., not in the neighborhood, then $\exists i$ such that $x_i \leq a_i - \epsilon$ or $x_i \geq b_i + \epsilon$.

Constructing N_G . We want to construct an indicator function whose value within a box G is close to 1 and outside the neighborhood $\nu(G)$ is close to 0. In the multi-dimensional case, $m \geq 2$, we do not know at which, if any, dimension j of an input is outside the neighborhood of G . The 1-dimensional indicator approximation, \hat{t} , which we constructed earlier, can be used to tell us, for each dimension j , whether x_j is within the bounds of the neighborhood of G . Therefore we can

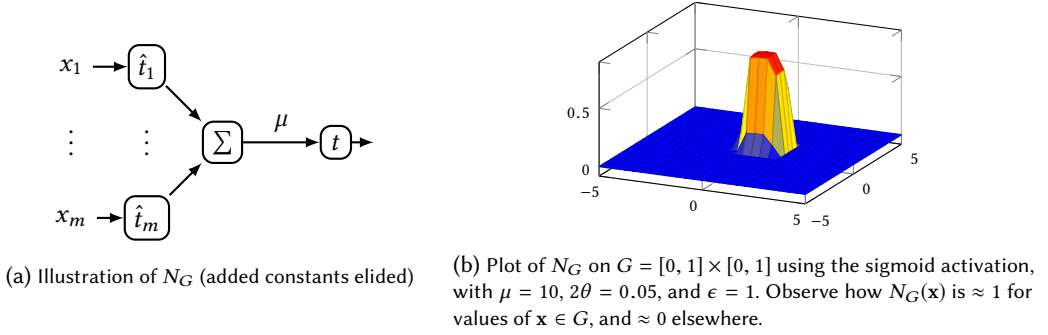


Fig. 8. **Step 2** Illustration of neural network N_G

construct a logical OR approximation that applies \hat{t} to each dimension and takes the OR of the results. Specifically,

- (1) We will construct a function that applies \hat{t} to each dimension, and sums the results such that the answer is > 0 if $\mathbf{x} \in G$, and < 0 if $\mathbf{x} \notin v(G)$.
- (2) Then, we can use the step-function approximation to indicate the step of the answer.

Formally, we define the neural network N_G as follows:

$$N_G(\mathbf{x}) = t \left(\mu \left(\sum_{i=1}^m H_i(x_i) + 0.5\epsilon \right) \right) \quad (3)$$

where $H_i(x) = \hat{t}_i(x) - (1 - 2\theta)$, and \hat{t}_i is \hat{t} using the range $[a_i, b_i]$ of the i th dimension of G . The neural network N_G is graphically depicted in Fig. 8a.

The function term $\sum_{i=1}^m H_i(x_i)$ evaluates to a positive value if $\mathbf{x} \in G$ and to a negative value if $\mathbf{x} \notin v(G)$. Observe that we need to shift the result of \hat{t} by $(1 - 2\theta)$ to ensure a negative answer if one of the dimensions is outside the neighborhood. Then, we use t to approximate the step function, as we did in the 1-dimensional case, giving ≈ 1 if $\mathbf{x} \in G$, and ≈ 0 if $\mathbf{x} \notin v(G)$.

Example 4.9. Fig. 8b shows a plot of N_G for $\mathbf{x} \in \mathbb{R}^2$. ■

Abstract precision of N_G . We are now ready to analyze the abstract precision of N_G . We first consider H_i in the following lemma. For any box $B \subseteq C$, let B_i be its projection on dimension i , which is an interval.

The following lemma states that if B is in the box G , then $\sum_i H_i^\#$ is positive; otherwise, if B is outside the neighborhood of G , then $\sum_i H_i^\#$ is negative.

LEMMA 4.10 (ABSTRACT INTERPRETATION OF H_i). *For any box $B \subseteq C$, the following is true:*

- (1) If $B \subseteq G$, then $\sum_{i=1}^m H_i^\#(B_i) \subseteq (0, \infty)$.
- (2) If $B \subseteq C \setminus v(G)$, then $\sum_{i=1}^m H_i^\#(B_i) \subseteq (-\infty, -\epsilon)$.

The following theorem states the precision of the abstract interpretation of N_G : if the input box is in G , then the output box is within θ from 1; if B is outside the neighborhood of G , then the output box is within θ from 0.

THEOREM 4.11 (ABSTRACT INTERPRETATION OF N_G). *For any box $B \subseteq C$, the following is true:*

- (1) $N_G^\#(B) \subseteq [0, 1]$.
- (2) If $B \subseteq G$, then $N_G^\#(B) \subseteq (1 - \theta, 1]$.

(3) If $B \subseteq C \setminus v(G)$, then $N_G^\#(B) \subseteq [0, \theta]$.

PROOF.

STATEMENT (1): See definition of N_G in Eq. (3). The outer function of N_G is t , whose range is $[0, 1]$ by the definition of squashable functions and our assumption that the left and right limits are 0 and 1. Therefore, $N_G^\#(B) \subseteq [0, 1]$.

STATEMENT (2): If $B \subseteq G$, from Lemma 4.10, we know that $\sum_{i=1}^m H_i^\#(B_i) \subseteq (0, \infty)$. Then,

$$\sum_i^m H_i^\#(B_i) +^\# (0.5\epsilon)^\# \subseteq (0, \infty) +^\# (0.5\epsilon)^\# \subseteq (0.5\epsilon, \infty)$$

From Lemma 4.3, we know that if $x \geq 0.5\epsilon$, then $1 - \theta < t(\mu x) \leq 1$. Therefore,

$$N_G^\#(B) = t^\#(\mu^\# *^\# (0.5\epsilon, \infty)) \subseteq (1 - \theta, 1]$$

STATEMENT (3): If $B \subseteq C \setminus v(G)$, from Lemma 4.10, we know that $\sum_{i=1}^m H_i^\#(B_i) \subseteq (-\infty, -\epsilon)$. Then,

$$\sum_{i=1}^m H_i^\#(B_i) +^\# (0.5\epsilon)^\# \subseteq (-\infty, -\epsilon) +^\# (0.5\epsilon)^\# \subseteq (-\infty, -0.5\epsilon)$$

From Lemma 4.3, we know that if $x \leq -0.5\epsilon$, then $0 \leq t(\mu x) < \theta$. Therefore,

$$N_G^\#(B) = t^\#(\mu^\# *^\# (-\infty, -0.5\epsilon)) \subseteq [0, \theta]$$

□

Complexity of construction. To construct a single indicator function, we use $2m + 1$ activation functions, with depth 2 and width $2m$. If we restrict ourselves to ReLU activations, we use $4m + 2$ neurons, with depth 4 and width $2m$; in contrast, Baader et al. [2020] use $10m - 3$ ReLU functions, with depth $3 + \log_2(m)$, and width $4m$.

5 HARDNESS OF RANGE APPROXIMATION

In this section, we will present the range approximation (RA) problem and some basics of computational complexity theory. By studying the complexity of RA, one can understand the hardness of IUA.

5.1 The Polynomial Hierarchy

The polynomial hierarchy generalizes the definitions of P, NP, coNP. Let L be a language.

Definition 5.1 (The NP and coNP classes). L is an NP language if there exists a polynomial-time Turing machine M , and a polynomial q such that

$$x \in L \text{ if and only if } \exists u \in \{0, 1\}^{q(|x|)}. M(x, u) = 1.$$

coNP languages are similarly defined, but with a universal (\forall) quantifier instead. ■

Example 5.2. Deciding whether a Boolean formula is satisfiable is an NP problem. Deciding whether a Boolean formula is a tautology is a coNP problem. ■

We write $\Sigma_1 = \text{NP}$, and $\Pi_1 = \text{coNP}$. Notice that the difference between Σ_1 and Π_1 is the leading quantifier. Indeed, the definitions of Σ_n and Π_n have similar structure as NP and coNP, with n alternating quantifiers rather than a single quantifier. In this paper, we only need to consider Σ_2 and Π_2 , which we define below.

Definition 5.3 (The Σ_2 class). L is a Σ_2 language if there exists a polynomial-time Turing machine M , and a polynomial q such that

$$x \in L \text{ if and only if } \exists u_1 \in \{0, 1\}^{q(|x|)} \forall u_2 \in \{0, 1\}^{q(|x|)}. M(x, u_1, u_2) = 1.$$

Definition 5.4 (The Π_2 class). L is a Π_2 language if there exists a polynomial-time Turing machine M , and a polynomial q such that

$$x \in L \text{ if and only if } \forall u_1 \in \{0, 1\}^{q(|x|)} \exists u_2 \in \{0, 1\}^{q(|x|)}. M(x, u_1, u_2) = 1.$$

Definition 5.5 (The Δ_2 class). $\Delta_2 = \Sigma_2 \cap \Pi_2$.

Note that $\text{NP}, \text{coNP} \subseteq \Delta_2$, because one can substitute an empty string to u_1 or u_2 in Definitions 5.3 and 5.4. The polynomial hierarchy is the union of all Σ_n languages.

Definition 5.6 (Δ_2 -intermediate language). A set of languages \mathbb{L} is Δ_2 -intermediate if $\text{NP} \cup \text{coNP} \subseteq \mathbb{L}$ and $\mathbb{L} \subseteq \Delta_2$.

Remark. By definition, $\Delta_2 = \Pi_2 \cap \Sigma_2$, and $\text{NP} \cup \text{coNP} \subseteq \Delta_2$ because both NP and coNP are subsets of Δ_2 . It is unknown whether $\text{NP} \cup \text{coNP} = \Delta_2$ or not. However, if $\text{coNP} \not\subseteq \text{NP}$ as is commonly believed, $\text{NP} \subseteq \mathbb{L}$ when \mathbb{L} is Δ_2 -intermediate.

5.2 The Range Approximation Problem

In this section, we present the RA problem. This will reveal one of the fundamental differences between the classical UA and IUA because the interval approximation can be studied via decision problems (Section 6.3) and therefore one can attempt to understand its computational complexity. In the appendix, we will give a precise description of the real-valued computational model.

We restrict our attention to neural networks that map $[0, 1]^m$ to codomain $[0, 1]$, i.e., $f : [0, 1]^m \rightarrow [0, 1]$ is a neural network. Throughout the paper, when we use polynomial-time executable, we mean polynomial in terms of m .

Definition 5.7 (δ -range approximation). Let $\delta > 0$ and $f : [0, 1]^m \rightarrow [0, 1]$ be a neural network. We can δ -range approximate f if we find $a \leq b \in [0, 1]$ such that

$$[l + \delta, u - \delta] \subseteq [a, b] \subseteq [l - \delta, u + \delta]$$

where $l = \min f([0, 1]^m)$ and $u = \max f([0, 1]^m)$.

Note that δ -range approximation (Definition 5.7) is weaker than δ -interval approximation (IA) (Definition 3.6) in the following ways:

- (1) In RA, we only need f to be a neural network, while in IA, we aim to approximate any continuous function;
- (2) In IA, we require the approximation holds for any B in the domain, while in RA, we only need it holds for the domain $[0, 1]^m$;
- (3) In IA, we need to find a neural network that approximates f , but in RA, we do not require any specific ways to find a, b . If one can find the δ -interval approximation neural network, abstractly executing the neural network will return a, b automatically.

As a result, if we show that the δ -range approximation problem is hard, then building a δ -interval approximation neural network that can be polynomial-time executable has to be hard.

We now state a dichotomy theorem on the δ -range approximation problem:

THEOREM 5.8 (DICHOTOMY OF δ -RANGE APPROXIMATION). *Let $f : [0, 1]^m \rightarrow [0, 1]$ be a neural network with any squashable functions. Then*

- (1) *If $\delta \geq 1/2$, it is trivial to δ -range approximate f .*
- (2) *If $\delta < 1/2$, it is NP-hard and coNP-hard to δ -range approximate f . Moreover, if we assume that the neural network takes polynomial time to execute and the input has finite precision, then it is Δ_2 -intermediate to δ -range approximate f .*

The first statement is trivial, because if $\delta \geq 1/2$, we can choose $a = b = 1/2$. Because we have $0 \leq l \leq u \leq 1$, then it is always true that when $\delta \geq 1/2$,

$$u - \delta \leq 1/2 \leq u + \delta$$

and

$$l - \delta \leq 1/2 \leq l + \delta.$$

We will show the second statement in Section 5. The idea is to reduce the problem of determining the range of a Boolean formula to the δ -range approximation problem, by encoding the Boolean formula as a neural network with step functions as the activation functions. Since squashable units can arbitrarily approximate the step function, the δ -range approximation problem for the neural network is also hard.

Implications of RA hardness. Even though we knew that exactly finding the range is hard, δ -range approximation might be much easier. As an analogy, many NP-complete optimization problems have polynomial-time approximation algorithms [Vazirani 2003]. Theorem 5.8 is surprising because it shows a dichotomy that it is either trivial or very hard to achieve the approximation of the set semantics depending on how close/tolerant one demands the approximation to be. As we will reveal in Section 6, exactly deciding or approximating the range of a neural network are not very different from the complexity-theoretical view, even though the former appears a harder task because it implies the latter (Corollary 6.4).

Theorem 5.8 also implies that even if we have a neural network N_0 that approximates some function in the pointwise sense, it does not help build the interval approximator because one cannot simply build another network N whose abstract interpretation approximates the set semantics of N_0 . This shows the non-triviality of the IUA, even though UA is a classical topic and has been studied extensively.

6 PROOF OF HARDNESS OF RA

In this section, we will show the second statement of Theorem 5.8, that it is Δ_2 -intermediate to δ -range approximate f for $\delta < 1/2$ and $f : [0, 1]^m \rightarrow [0, 1]$, where f is a neural network with any squashable functions. Before showing this, we will consider the Boolean formula counterpart of this problem. This will provide us with an intuition for the original neural network version of the range-approximation problem.

Because a Boolean formula is only valued in $\{0, 1\}$, δ -range approximating a Boolean formula effectively decides the exact range of the Boolean formula. Therefore, let's consider the following problem: deciding exactly the range of a Boolean formula. We show that this problem is Δ_2 -intermediate.

6.1 Deciding the Range of a Boolean Formula

Let ϕ be an arbitrary Boolean formula. Let R_ϕ be the range of ϕ . To decide the range of ϕ amounts to deciding whether $R_\phi = \{0\}$, $R_\phi = \{1\}$, or $R_\phi = \{0, 1\}$. To show a problem is Δ_2 -intermediate, we

need to show that it is in $\Delta_2 = \Sigma_2 \cap \Pi_2$; and it is both Σ_1 -hard and Π_1 -hard. Recall that $\Sigma_1 = \text{NP}$ and $\Pi_1 = \text{coNP}$.

$\mathbb{R}_\phi = \{0\}$ can be expressed as

$$\forall x. \phi(x) = 0.$$

Similarly, $\mathbb{R}_\phi = \{1\}$ can be expressed as

$$\forall x. \phi(x) = 1.$$

$\mathbb{R}_\phi = \{0, 1\}$ can be expressed as

$$\exists x, y. \phi(x) = 1 \wedge \phi(y) = 0.$$

All of them can be expressed within both Σ_2 and Π_2 languages. Therefore, deciding the range of a Boolean formula is in $\Sigma_2 \cap \Pi_2 = \Delta_2$.

The canonical NP-hard problem is deciding whether a Boolean formula is satisfiable, and the canonical coNP-hard problem is deciding whether a Boolean formula is a tautology. Indeed, if one can decide the range of a Boolean formula, then one can easily tell whether the Boolean formula is satisfiable or not, and whether the Boolean formula is a tautology or not.

Therefore, deciding the range of a Boolean formula is Δ_2 -intermediate.

6.2 Range Approximating a Neural Network

Showing that deciding the range of a formula is a Δ_2 -intermediate problem provides an intuition for why the range approximation problem is Δ_2 -intermediate. Our proof of the hardness of RA consists of 3 parts:

- (1) The RA problem is in $\Sigma_2 \cap \Pi_2$. We show that (1) we can express deciding the range of a polynomial-time executable neural network using Π_2 and Σ_2 languages. Because deciding the range of a function is harder than approximating the range, this shows that approximating the range is also in Δ_2 .
- (2) The RA problem is NP-hard.
- (3) The RA problem is coNP-hard.

We will build reductions from the NP-hard and coNP-hard problems to the RA problem, and this shows that both exactly deciding and approximating the range of a neural network are NP- and coNP-hard.

The NP-hard problem is whether a Boolean formula in 3CNF is satisfiable. The coNP-hard problem is whether a Boolean formula in 3DNF is a tautology. In particular, we will encode the 3CNF and 3DNF formulas computation using neural networks with squashable functions, which comes from on how to encode a Boolean formula using neural networks with perfect step functions.

RA is in Δ_2 . We first show that the RA problem is Δ_2 . We will need the assumptions that the neural network is polynomial-time executable in terms of m and the input precision is finite, otherwise, we cannot use a polynomial-time Turing machine to simulate the execution of the neural network. We require finite input-precision because we want to ensure there are only exponentially many inputs. Note that the NP-hardness and coNP-hardness of RA do not need these assumptions, so if $\text{coNP} \not\subseteq \text{NP}$ as commonly believed, RA is always harder than any NP-complete problem.

LEMMA 6.1. *The δ -range approximation problem as defined in Theorem 5.8 is in Δ_2 for $\delta < 1/2$, if the neural network is polynomial-time executable and the input has finite precision.*

PROOF. We will show that exactly deciding the range of f is in Δ_2 . Because exactly deciding a range is harder than the approximating it, this also shows that approximating the range of a polynomial-time executable neural network is in Δ_2 .

Because f is a continuous function, deciding the range of f is $[a, b]$ can be written as

$$\exists x, y. \forall z. f(x) = a \wedge f(y) = b \wedge f(z) \leq b \wedge f(z) \geq a. \quad (4)$$

In Eq. (4), because z is not dependent on x, y , we can also switch the order of the quantifier. Therefore, deciding the range of f is in both Σ_2 and Π_2 , and thus in Δ_2 . \square

Hardness of RA. We need to show that the RA problem is both NP-hard and coNP-hard, which is formally stated in the following lemmas:

LEMMA 6.2. *The δ -range approximation problem as defined in Theorem 5.8 is NP-hard for $\delta < 1/2$.*

LEMMA 6.3. *The δ -range approximation problem as defined in Theorem 5.8 is coNP-hard for $\delta < 1/2$.*

We will present a decision problem formulation of approximating the maximum value of the neural network, and show a reduction from the SAT problem to the decision problem in Section 6.3. This shows that the δ -range approximation problem is NP-hard.

The idea of the reduction is to use neural networks with squashable functions to encode a 3CNF formula as discussed in Section 2. Approximating the range of the neural network also approximates the range of the Boolean formula. As discussed in Section 6.1, given the range of the Boolean formula, it is easy to know its satisfiability. Then we know the δ -range approximation problem is NP-hard.

The proof of Lemma 6.3 very much resembles that of Lemma 6.2. We encode a 3DNF formula instead of a 3CNF formula, and delegate the proof to the supplementary materials.

Range decision of neural networks. Lemmas 6.2 and 6.3 also imply that deciding the range of a neural network is both NP-hard and coNP-hard. Together with the result that deciding the range of the neural network is in Δ_2 , we have the following corollary:

COROLLARY 6.4. *Let $f : [0, 1]^m \rightarrow [0, 1]$ be a neural network that takes polynomial time to execute and the input has finite precision, then it is Δ_2 -intermediate to decide the range of f .*

6.3 The NP-Hardness of Range Approximation

In this section, we will prove Lemma 6.2. $f : [0, 1]^m \rightarrow [0, 1]$ is a neural network with any squashable units. We will show that for $\delta < 1/2$, it is NP-hard to approximate the maximum value of f over $[0, 1]^m$ up to an additive factor of δ (Lemma 6.5). This is accomplished by a reduction from the 3SAT problem, and we show that there is an gap between the maximums of neural networks that encode either satisfiable formulas or unsatisfiable formulas (Proposition 6.7). This gap enables us to show that approximating of maximum of neural network is NP-hard because approximating the maximum can tell whether the 3CNF formula is satisfiable or not. Because if one can approximate the range of a neural network, one can also approximate its maximum. This implies that approximating the range of a neural network is NP-hard (Lemma 6.2).

Our reduction maps 3CNF formulas ϕ over m variables to a neural network f on m variables, such that if ϕ is satisfiable then the maximum value attained by f over $[0, 1]^m$ lies in $(1/2 + \delta, 1]$, and if ϕ is unsatisfiable, the maximum value of f over $[0, 1]^m$ lies in $[0, 1/2 - \delta]$. The neural network returned by the reduction can be built using any squashable activation.

Decision Problem Formulation. Let F_m be the set of neural networks over m variables that map $[0, 1]^m$ to codomain $[0, 1]$, and let $F = \bigcup_{m \geq 1} F_m$. For $\delta < 1/2$, let

$$F_\delta^+ = \bigcup_{m \geq 1} \left\{ f \in F_m \mid \max_{\mathbf{x} \in [0, 1]^m} (f(\mathbf{x})) > 1/2 + \delta \right\} \quad (5)$$

$$F_{\delta}^{-} = \bigcup_{m \geq 1} \left\{ f \in F_m \mid \max_{\mathbf{x} \in [0,1]^m} (f(\mathbf{x})) \leq 1/2 - \delta \right\} \quad (6)$$

LEMMA 6.5. Given $f \in F_{\delta}^{+} \cup F_{\delta}^{-}$, it is NP-hard to determine whether $f \in F_{\delta}^{+}$ or $f \in F_{\delta}^{-}$.

Since an efficient algorithm for δ -range approximating a neural network also approximates its maximum value, Lemma 6.2 is an immediate consequence of Lemma 6.5.

SAT reduction. Let X_1, \dots, X_m be Boolean variables, and $L_i = (\neg)X_i$ is called a *literal* (with the negation operator, it is called a negative literal). A 3CNF instance ϕ is a conjunction of *clauses* of the form $C_1 \wedge \dots \wedge C_k$, where each clause C_j is a disjunction of 3 literals. To distinguish the 3CNF instance and its simulation using the neural network, we will use uppercase letters to denote components in the 3CNF instance, and lowercase letters to denote the corresponding construction in the neural network.

Simulation of 3CNF. We will need to simulate the logical operations using the neural network operations. If we have perfect step functions as the activations and the input values are discrete, then the 3CNF instance can be easily simulated, as we have demonstrated in Section 2.2 on how to encode Boolean formula. Using the idea presented in Section 2, we will scale and shift the activations to simulate the step function. Define the following three activation functions that will be used in the reduction:

$$t_1(z) = \begin{cases} \geq -0.2 \text{ and } \leq -0.1, & z \leq 0.6 \\ \geq 0.5 \text{ and } \leq 0.6, & z \geq 0.7 \end{cases} \quad (7)$$

$$t_2(z) = \begin{cases} \geq \frac{1}{2k} \text{ and } \leq \frac{1}{k}, & z \geq 0.1 \\ \leq -1, & z \leq 0 \end{cases} \quad (8)$$

$$t_3(z) = \begin{cases} > 1/2 + \delta \text{ and } \leq 1, & z \geq 0.5 \\ < 1/2 - \delta \text{ and } \geq 0, & z \leq 0 \end{cases} \quad (9)$$

Recall that we use upper case letters for Boolean variables and lower case letters for neural network variables. The goal is to show Lemma 6.5, and it can be proved via Proposition 6.7, i.e., for satisfiable or unsatisfiable instances, the neural networks have different upper bounds. On one hand, if the instance ϕ is satisfiable, we can take the satisfiable assignment \mathbf{X} of values 0s and 1s as an input to the neural network and show that $f(\mathbf{x}) > 1/2 + \delta$. On the other hand, if $f(\mathbf{x}) > 1/2 - \delta$, we can use \mathbf{x} to construct a satisfiable assignment for ϕ . The output values in Eq. (9) are chosen to generate the gap as in Lemma 6.5. The choice for other values in Eqs. (7) to (9) are not unique. We only need to ensure that for satisfiable or unsatisfiable instances, we can produce the gap.

We will simulate the 3CNF instance using a neural network in the following way.

- For each variable X_i , construct an input node x_i .
- Simulate the negation operator using $l_i = 1 - x_i$. If there is no negation operator for l_i , we use $l_i = x_i$ directly. Then transform each literal using t_1 .
- For each disjunction operator, we will use t_2 to control the output value. For example, if $C_j = L_{j1} \vee L_{j2} \vee L_{j3}$, build the gadget $c_j = t_2(t_1(l_{j1}) + t_1(l_{j2}) + t_1(l_{j3}))$.
- For the conjunction operator, we will use t_3 . For example, if $\phi = \bigwedge_{i=1}^k C_i$, then let $y = t_3(\sum_{i=1}^k c_i)$.

Example 6.6. Fig. 9 shows an example of the neural network corresponds to the 3SAT instance $(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_4)$. ■

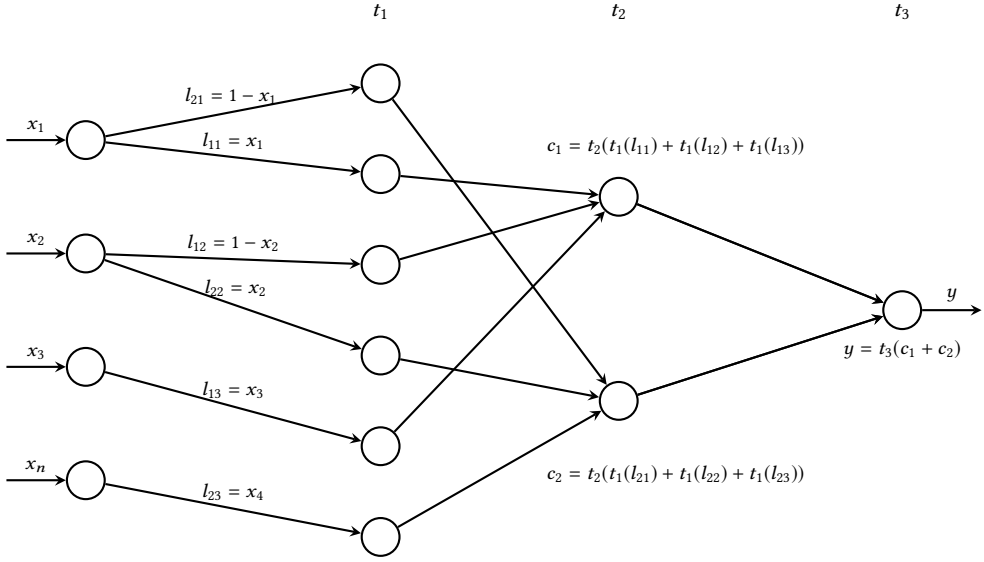


Fig. 9. The neural network encoding for $(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_4)$

Gap in upper bounds. We need to ensure that there is a gap between the upper bound of neural networks obtained from satisfiable or unsatisfiable 3CNF instances. This shows that even the approximation of the upper bound can differentiate satisfiable and unsatisfiable 3CNF instances.

PROPOSITION 6.7. *For a 3CNF instance ϕ , let N_ϕ be the encoding neural network. Let $y_u = \max N_\phi([0, 1]^m)$. The following two statements are true:*

- (1) *If the 3CNF instance ϕ is satisfiable, then $y_u > 1/2 + \delta$.*
- (2) *If ϕ is unsatisfiable, then $y_u \leq 1/2 - \delta$.*

PROOF. STATEMENT (1): If ϕ is satisfiable, let v_i be a satisfying assignment of X_i and use them as the input to N_ϕ . For each clause C_j , at least one literal is valued 1. WLOG, assume $L_{j1} = 1$. Therefore, $t_1(l_{j1}) \geq 0.5$, the remaining two literals are valued either 0 or 1, then $t_1(l_{jk}) \geq -0.2$ for the gadgets corresponding to the two literals. Thus, $t_1(l_{j1}) + t_1(l_{j2}) + t_1(l_{j3}) \geq 0.5 - 0.2 - 0.2 \geq 0.1$, and $c_j = t_2(t_1(l_{j1}) + t_1(l_{j2}) + t_1(l_{j3})) \geq \frac{1}{2k}$. Therefore, $\sum_{i=1}^k c_i \geq 1/2$, then $y = t_3(\sum_{i=1}^k c_i) > 1/2 + \delta$, and so $y_u > 1/2 + \delta$.

STATEMENT (2): We will prove that if $y_u > 1/2 - \delta$, then ϕ is satisfiable. Let \mathbf{z} be such that $N_\phi(\mathbf{z}) > 1/2 - \delta$. For each $i \in \{1, \dots, m\}$, if $z_i \geq 0.6$, let $x_i = 1$; otherwise, let $x_i = 0$. We will show that \mathbf{x} is a satisfying assignment for ϕ .

We need to show that for each clause C_i , the assignment makes C_i true. Equivalently, at least one of the literals in C_i is true. Let us consider the corresponding gadget c_i in the network. Because $N_\phi(\mathbf{z}) > 1/2 - \delta$, from Eq. (9) and the construction $y = t_3(\sum_{i=1}^k c_i)$, then $\sum_{i=1}^k c_i(\mathbf{z}) > 0$.

This implies that $c_i(\mathbf{z}) > -1$ for every i . Otherwise, the gadgets corresponding to the remaining $k - 1$ clauses are valued at most $\frac{1}{k}$ (see Eq. (8)), if $c_i(\mathbf{z}) \leq -1$, then $\sum_{i=1}^k c_i(\mathbf{z}) \leq -1 + (k - 1)\frac{1}{k} < 0$.

Because $c_i(\mathbf{z}) > -1$ and $c_i = t_2(t_1(l_{i1}) + t_1(l_{i2}) + t_1(l_{i3}))$, $[t_1(l_{i1}) + t_1(l_{i2}) + t_1(l_{i3})](\mathbf{z}) > 0$ (see Eq. (8)).

For all the three literals L_{i1}, L_{i2}, L_{i3} in C_i , consider the three gadgets corresponding to them in the neural network. Because $[t_1(l_{i1}) + t_1(l_{i2}) + t_1(l_{i3})](\mathbf{z}) > 0$, at least one of the literals l_{i1}, l_{i2}, l_{i3}

is evaluated to > 0.6 , otherwise $t_1(l_{ij})(\mathbf{z}) \leq -0.1$ for $j = 1, 2, 3$, and it is impossible that $[t_1(l_{i1}) + t_1(l_{i2}) + t_1(l_{i3})](\mathbf{z}) > 0$.

WLOG, let's assume $l_{i1}(\mathbf{z}) > 0.6$. We can consider the corresponding literal L_{i1} in C_i . Let L_{i1} come from variable X_j . Either $L_{i1} = X_j$ or $L_{i1} = \neg X_j$. In the former case, because $l_{i1}(\mathbf{z}) > 0.6$, then $l_{i1}(\mathbf{z}) = l_{i1}(z_j) = z_j > 0.6$. According to our assignment rule, $x_j = 1$ and X_j is evaluated true, and so is C_i . In the latter case, $l_{i1}(\mathbf{z}) = l_{i1}(z_j) = 1 - z_j > 0.6$, so $z_j < 0.4$. According to the assignment rule, $x_j = 0$, and so X_j is evaluated to false. C_i is still evaluated to true.

We have shown that the assignment \mathbf{x} satisfies all clauses, and so the 3-SAT instance ϕ . \square

Consequences of NP-hardness. We have shown that the decision problem in Lemma 6.5 is NP-hard. Because RA implies the decision problem, RA is NP-hard. Also, as described in Section 5.2, IUA implies RA, so IUA is also NP-hard. Additionally, we also have the following result:

COROLLARY 6.8. *It is NP-hard to falsify correctness⁵ of neural networks with any squashable activation function.*

PROOF. To show that this is NP-hard, given a 3CNF instance ϕ , let's build the neural network N_ϕ in Section 6.3. From Proposition 6.7 we know that $N_\phi \in F_\delta^+ \cup F_\delta^-$. Let the correctness constraint be $\bigwedge_{i=1}^m (x_i \leq 1) \wedge \bigwedge_{i=1}^m (x_i \geq 0) \wedge (y > 0.5)$. If one can decide the satisfiability of this constraint, then one can decide $N_\phi \in F_\delta^+$ or $N_\phi \in F_\delta^-$. Therefore, the falsification of neural networks with any squashable activation functions is NP-hard. \square

Because ReLU is also squashable, the result of Katz et al. [2017], showing that falsifying ReLU networks is NP-hard, is a special case of Corollary 6.8.

7 PROVABLY ROBUST NEURAL NETWORKS

In this section, we discuss the connection between the IUA theorem and robust classifiers. Because of the soundness of abstract interpretation, we can use it to verify the robustness of neural networks. However, abstract interpretation is not complete, so some robust points might not be verified. One consequence of the IUA is that not only a neural network can arbitrarily approximate any continuous function on a compact domain, as we knew from classical universal approximation, but one could also construct a neural network as shown in the proof of IUA, where all the robust inputs can be verified using interval abstract interpretation. We begin with some definitions on robustness and provably robust neural networks, and then show how IUA implies the existence of provably robust neural networks.

Robust classifiers. We begin by defining a robust classifier using ℓ_∞ -norm. Throughout this section, we assume that $f : C \rightarrow \mathbb{R}$ is a continuous function over compact domain C . We treat f as a *binary* classifier, where an output < 0.5 represents one class and ≥ 0.5 represents another.

We start by defining the notion of an ϵ -ball, which can represent, for example, a set of copies of the same image but with varying brightness. Recall that ℓ_∞ -norm is defined as $\|\mathbf{z}\|_\infty = \max_i |z_i|$.

Definition 7.1 (ϵ -Ball). Let $\mathbf{x} \in \mathbb{R}^m$ and $\epsilon > 0$. The ϵ -ball of \mathbf{x} is $R_\epsilon(\mathbf{x}) = \{\mathbf{z} \mid \|\mathbf{z} - \mathbf{x}\|_\infty \leq \epsilon\}$. \blacksquare

Next, we define an ϵ -robust classifier. Informally, given a set of points M , for each $\mathbf{x} \in M$, an ϵ -robust classifier returns the same classification for all points in the ϵ -ball of \mathbf{x} .

Definition 7.2 (ϵ -Robustness). Let $M \subseteq C$ and $\epsilon > 0$. We say that f is ϵ -robust on set M iff, for all $\mathbf{x} \in M$ and $\mathbf{z} \in R_\epsilon(\mathbf{x})$, we have $f(\mathbf{x}) < 0.5$ iff $f(\mathbf{z}) < 0.5$. \blacksquare

⁵Falsification is defined as deciding whether the conjunction of linear constraints on the input and output of the network is satisfiable, as used by Katz et al. [2017] for verification.

Provably robust neural networks. Next, we define provably robust neural networks. These are neural networks for which we can automatically prove ϵ -robustness. Note that an ϵ -ball is a box in \mathbb{R}^m , and so there is no loss of precision while using the interval domain, i.e., $\gamma(\alpha(R_\epsilon(\mathbf{x}))) = R_\epsilon(\mathbf{x})$.

Definition 7.3 (Provably robust networks). A neural network N is ϵ -provably robust on M iff, for all $\mathbf{x} \in M$, we have $N^\#(B) \subseteq (-\infty, 0.5)$ or $N^\#(B) \subseteq [0.5, \infty)$, where $B = \alpha(R_\epsilon(\mathbf{x}))$. ■

From an automation perspective, the set M is typically a finite set of points, e.g., images. For every $\mathbf{x} \in M$, the verifier abstract interprets N on the ϵ -ball of \mathbf{x} , deriving a lower bound and upper bound of the set of predictions $N(R_\epsilon(\mathbf{x}))$. If the lower bound is ≥ 0.5 or the upper bound is < 0.5 , then we have proven that all images in the ϵ -ball have the same classification using N .

Existence of provably robust networks. The following theorem states the existence of provably robust neural networks. Specifically, assuming there is some ideal robust classifier, then, following the IUA theorem, we can construct a neural network, using any squashable activation function, that matches the classifier's predictions and is provably robust.

THEOREM 7.4 (EXISTENCE OF ROBUST NETWORKS). *Let $f : C \rightarrow R$ be ϵ -robust on set $M \subseteq C$. Assume that $\forall \mathbf{x} \in M, \mathbf{z} \in R_\epsilon(\mathbf{x}), f(\mathbf{z}) \neq 0.5$.⁶ Let t be a squashable activation function. Then, there exists a neural network N , using activation functions t , that*

- (1) agrees with f on M , i.e., $\forall \mathbf{x} \in M, N(\mathbf{x}) < 0.5$ iff $f(\mathbf{x}) < 0.5$, and
- (2) is ϵ -provably robust on M .

PROOF. Let set $Z = \bigcup_{\mathbf{x} \in M} R_\epsilon(\mathbf{x})$. Let $\delta' = \min_{\mathbf{z} \in Z} |f(\mathbf{z}) - 0.5|$. That is, $\delta' > 0$ is the smallest distance from the classification boundary. Following the IUA theorem, we know that there is a neural network N that δ -interval approximates f , for any $\delta < \delta'$. Fix such network N .

STATEMENT (1): Pick any $\mathbf{x} \in M$. Suppose that $f(\mathbf{x}) < 0.5$. Then, we know that $0.5 - f(\mathbf{x}) \geq \delta'$. By the IUA theorem, we know that $|N(\mathbf{x}) - f(\mathbf{x})| \leq \delta < \delta'$. It follows that $N(\mathbf{x}) < 0.5$. The case where $f(\mathbf{x}) > 0.5$ is symmetric.

STATEMENT (2): Let $\mathbf{x} \in M$. Suppose that $f(\mathbf{x}) < 0.5$. Because f is robust, $\forall \mathbf{z} \in R_\epsilon(\mathbf{x}), f(\mathbf{z}) < 0.5$. Then, we know that $0.5 - \max f(R_\epsilon(\mathbf{x})) \geq \delta'$. By the IUA theorem, we know that $N^\#(R_\epsilon(\mathbf{x})) = \langle [l, u] \rangle$, where $|u - \max f(R_\epsilon(\mathbf{x}))| \leq \delta < \delta'$. It follows that $N^\#(R_\epsilon(\mathbf{x})) \subseteq (-\infty, 0.5)$. The case where $f(\mathbf{x}) > 0.5$ is symmetric. So, N is ϵ -provably robust on M . □

n -ary classifiers. The above theorem can be extended to n -ary classifiers, for $n > 2$, in an analogous fashion. Please refer to the supplementary materials for the formalization and proof.

8 RELATED WORK

Universal approximation. The classical universal approximation (UA) theorem has been established for decades. In contrast to IUA, UA states that a neural network with one single hidden layer can approximate any continuous function on a compact domain. One of the first versions goes back to [Cybenko \[1989\]](#); [Hornik et al. \[1989\]](#), who showed that the standard feed-forward neural network with sigmoidal or squashing activations is a universal approximator. The most general version of UA was discovered by [Leshno et al. \[1993\]](#), who showed that the feed-forward neural network is a universal approximator if and only if the activation function is non-polynomial. Because IUA implies UA, this means IUA cannot hold beyond non-polynomial activation functions. There are also other variants of UA. Some of them study the expressiveness of neural networks with structural constraints, such as restricted width per layer [[Kidger and Lyons 2019](#); [Lu et al.](#)].

⁶Informally, this assumption eliminates the corner case where a point sits exactly on the classification boundary, 0.5.

2017], or specific neural network architectures [Lin and Jegelka 2018]. Another line of work focuses on specific functions that one wants to approximate rather than arbitrary continuous functions, such as Anil et al. [2019]; Cohen et al. [2019], who study approximation of Lipschitz functions.

Baader et al. [2020] showed the first UA theorem adapted to interval analysis, and our high-level construction resembles theirs. However, we proved that the neural networks with any squashable activation functions can be an interval universal approximator. In contrast, they only showed the IUA theorem restricted to ReLU activation functions.

Neural-network verification. Neural-network verification has received a lot of attention in recent years. Consult Albarghouthi [2021] for an introduction. Most techniques are either based on decision procedures, like SMT solvers [Ehlers 2017; Katz et al. 2017] and integer linear programming (ILP) solvers [Tjeng et al. 2019], or abstract interpretation. The former class can often provide sound and complete verification on neural networks with piecewise-linear operations, like ReLU, but is not scalable due to the complexity of the problem and the size of the networks. Abstract-interpretation-based techniques sacrifice completeness for efficient verification. We have considered the simplest non-trivial numerical domain, intervals, that has been shown to produce strong results, both for robustness verification and adversarial training [Anderson et al. 2019; Gehr et al. 2018; Huang et al. 2019; Mirman et al. 2018; Wang et al. 2018; Zhang et al. 2020, 2021]. Researchers have considered richer domains [Singh et al. 2018, 2019], like zonotopes [Ghorbal et al. 2009] and forms of polyhedra [Cousot and Halbwachs 1978]. Since such domains are strictly more precise than intervals, the IUA theorem holds for them.

Complexity of Neural Network Verification. Katz et al. [2017] proved that the falsification of ReLU neural networks is NP-complete. It introduced a reduction from 3SAT to the ReLU neural network falsification problem. Our result implies theirs as we have shown. Weng et al. [2018] proved the inapproximability of finding the optimal ℓ_1 -distortion of ReLU networks, using a reduction from the set cover problem, a well-known hard-to-approximate problem. However, they are working on ℓ_1 ReLU robustness falsification problem and their reduction does not imply our result.

9 CONCLUSION

We identify a set of activation functions, squashable functions, which includes most commonly used activation functions. We prove that neural networks with any squashable functions are interval universal approximators. We further study the computational complexity to range-approximate a neural network, which implies that building the interval universal approximator is in general a hard task. Our proof uses the idea that squashable functions can arbitrarily approximate step functions and neural networks with step functions are formally well-behaved objects. We believe that this perspective can be important to understand the formal aspect of neural network in the future.

ACKNOWLEDGMENTS

This work is partially supported by Air Force Grant FA9550-18-1-0166, the National Science Foundation (NSF) Grants CCF-FMitF-1836978, SaTC-Frontiers-1804648 and CCF-1652140 and ARO grant number W911NF-17-1-0405.

REFERENCES

- Aws Albarghouthi. 2021. Introduction to Neural Network Verification. *CoRR* abs/2109.10317 (2021). arXiv:2109.10317 <https://verifieddeeplearning.com>
- Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. 2019. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 731–744.

- Cem Anil, James Lucas, and Roger Grosse. 2019. Sorting Out Lipschitz Function Approximation. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 291–301. <http://proceedings.mlr.press/v97/anil19a.html>
- Maximilian Baader, Matthew Mirman, and Martin Vechev. 2020. Universal Approximation with Certified Networks. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=B1gX8kBTPr>
- James Bergstra, Guillaume Desjardins, Pascal Lamblin, and Yoshua Bengio. 2009. Quadratic polynomials learn better image features. *Technical report*, 1337 (2009).
- Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). <http://arxiv.org/abs/1511.07289>
- Jeremy E. J. Cohen, Todd Huster, and Ra Cohen. 2019. Universal Lipschitz Approximation in Bounded Depth Neural Networks. arXiv:1904.04861 [cs.LG]
- Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (Los Angeles, California) (POPL '77)*. Association for Computing Machinery, New York, NY, USA, 238–252. <https://doi.org/10.1145/512950.512973>
- Patrick Cousot and Nicolas Halbwachs. 1978. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 84–96.
- George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems 2* (1989), 303–314.
- Rick Durrett. 2010. *Probability: Theory and Examples* (4 ed.). Cambridge University Press. <https://doi.org/10.1017/CBO9780511779398>
- Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 269–286.
- T. Gehr, M. Mirman, D. Drachler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. 3–18.
- Khalil Ghorbal, Eric Goubault, and Sylvie Putot. 2009. The zonotope abstract domain taylor1+. In *International Conference on Computer Aided Verification*. Springer, 627–633.
- Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 315–323.
- Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*. <http://arxiv.org/abs/1412.6572>
- Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. 2018. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715* (2018).
- Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. 2018. ReLU deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973* (2018).
- Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. 1989. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.
- Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. 2019. Achieving Verified Robustness to Symbol Substitutions via Interval Bound Propagation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. 4081–4091. <https://doi.org/10.18653/v1/D19-1419>
- Kai Jia and Martin Rinard. 2021. Exploiting Verified Neural Networks via Floating Point Numerical Error. arXiv:2003.03021 [cs.LG]
- Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.
- Patrick Kidger and Terry Lyons. 2019. Universal approximation with deep narrow networks. *arXiv preprint arXiv:1905.08539* (2019).
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
- Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks 6*, 6 (1993), 861 – 867. [https://doi.org/10.1016/S0893-6080\(05\)80131-5](https://doi.org/10.1016/S0893-6080(05)80131-5)

- Hongzhou Lin and Stefanie Jegelka. 2018. ResNet with one-neuron hidden layers is a Universal Approximator. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Curran Associates, Inc., 6169–6178. <http://papers.nips.cc/paper/7855-resnet-with-one-neuron-hidden-layers-is-a-universal-approximator.pdf>
- Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The Expressive Power of Neural Networks: A View from the Width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) (*NIPS'17*). Curran Associates Inc., Red Hook, NY, USA, 6232–6240.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.
- Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *International Conference on Machine Learning (ICML)*. <https://www.icml.cc/Conferences/2018/Schedule?showEvent=2477>
- Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.
- Michael A Nielsen. 2015. *Neural networks and deep learning*. Vol. 2018. Determination press San Francisco, CA.
- Veselin Raychev, Martin Vechev, and Andreas Krause. 2015. Predicting program properties from "big code". *ACM SIGPLAN Notices* 50, 1 (2015), 111–124.
- W. Rudin. 1986. *Principles of Mathematical Analysis*. McGraw - Hill Book C. <https://books.google.com/books?id=frDNAACA AJ>
- Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*. 10802–10813.
- Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–30.
- Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. <https://openreview.net/forum?id=HyGldiRqtm>
- V.V. Vazirani. 2003. *Approximation Algorithms*. Springer Berlin Heidelberg. <https://doi.org/10.1007/978-3-662-04565-7>
- Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*. 1599–1614.
- Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S. Dhillon, and Luca Daniel. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *International Conference on Machine Learning (ICML)*.
- Cihang Xie, Mingxing Tan, Boqing Gong, Alan Yuille, and Quoc V Le. 2020. Smooth Adversarial Training. *arXiv preprint arXiv:2006.14536* (2020).
- Yuhao Zhang, Aws Albarghouthi, and Loris D'Antoni. 2020. Robustness to Programmable String Transformations via Augmented Abstract Training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 11023–11032. <http://proceedings.mlr.press/v119/zhang20b.html>
- Yuhao Zhang, Aws Albarghouthi, and Loris D'Antoni. 2021. Certified Robustness to Programmable Transformations in LSTMs. *CoRR abs/2102.07818* (2021). arXiv:2102.07818 <https://arxiv.org/abs/2102.07818>

A VECTOR-VALUED NETWORKS AND ROBUSTNESS

In this section, we extend the IUA theorem to vector-valued functions. We also extend our robustness results to n -ary classifiers.

A.1 Higher-Dimensional Functions

Vector-valued neural networks. So far we have considered scalar-valued neural networks. We can generalize the neural-network grammar (Definition 2.1) to enable vector-valued neural networks. Simply, we can compose a sequence of n scalar-valued neural networks to construct a neural network whose range is \mathbb{R}^n . Formally, we extend the grammar as follows, where E_i are the scalar-valued sub-neural networks.

Definition A.1 (Vector-valued neural network grammar). A neural network $N : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is defined as follows

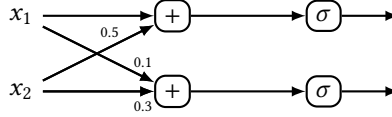
$$\begin{aligned} N &:= (E_1, \dots, E_n) \\ E &:= c \\ &\quad | \quad x_i \\ &\quad | \quad E_1 + E_2 \\ &\quad | \quad c * E_2 \\ &\quad | \quad t(E_1, \dots, E_k) \end{aligned}$$

where $c \in \mathbb{R}$, x_i is one of the m inputs to the network, and t is an activation function. ■

Example A.2. Consider the following neural network $N : \mathbb{R}^2 \rightarrow \mathbb{R}^2$:

$$N(\mathbf{x}) = (\sigma(x_1 + 0.5x_2), \sigma(0.1x_1 + 0.3x_2))$$

which we can pictorially depict as the following graph:



Generalized IUA theorem. We now generalize the IUA theorem to show that we can δ -interval approximate vector-valued functions.

THEOREM A.3. Let $f : C \rightarrow \mathbb{R}^n$ be a continuous function with compact domain $C \subset \mathbb{R}^m$. Let $\delta > 0$. Then, there exists a neural network $N : \mathbb{R}^n \rightarrow \mathbb{R}^m$ such that for every box $B \subseteq C$, and for all $i \in [1, m]$,

$$[l_i + \delta, u_i - \delta] \subseteq N^\#(B)_i \subseteq [l_i - \delta, u_i + \delta] \quad (10)$$

where

- (1) $N^\#(B)_i$ is the i th interval in the box $N^\#(B)$, and
- (2) $l_i = \min S_i$ and $u_i = \max S_i$, where $S = f(B)$ (recall that S_i is the set of i th element of every vector in S).

PROOF. From the IUA theorem, we know that there exists a neural network N_i that δ -interval approximates $f_i : C \rightarrow \mathbb{R}$, which is like f but only returns the i th output. We can then construct the network $N = (N_1, \dots, N_n)$. Since each N_i satisfies Eq. (10) separately, then N δ -interval approximates f . □

A.2 Robustness in n -ary classification

We now extend the definition of ϵ -robustness to n -ary classifiers. We use a function $f : C \rightarrow \mathbb{R}^n$ to denote an n -class classifier. f returns a value for each of the n classes; the class with the largest value is the result of classification. We assume there are no ties. Formally, for a given $\mathbf{x} \in C$, we denote classification by f as $\text{class}(f(\mathbf{x}))$, where

$$\text{class}(\mathbf{y}) = \arg \max_{i \in \{1, \dots, m\}} y_i$$

Definition A.4 (n -ary robustness). Let $M \subset C$. We say that f is ϵ -robust on M , where $\epsilon > 0$, iff for all $\mathbf{x} \in M$ and $\mathbf{x}' \in R_\epsilon(\mathbf{x})$, we have $\text{class}(f(\mathbf{x})) = \text{class}(f(\mathbf{x}'))$. ■

We now extend the provably robust neural networks definition to the n -class case. Recall that $R_\epsilon(\mathbf{x}) = \{\mathbf{x}' \mid \|\mathbf{x} - \mathbf{x}'\| \leq \epsilon\}$.

Definition A.5 (Provably robust networks). A neural network N is ϵ -provably robust on M iff, for all $\mathbf{x} \in M$, for all $\mathbf{y}, \mathbf{y}' \in \gamma(N^\#(\alpha(R_\epsilon(\mathbf{x}))))$, we have $\text{class}(\mathbf{y}) = \text{class}(\mathbf{y}')$. ■

Existence of robust networks. We now show existence of robust networks that approximate some robust n -ary classifier f .

THEOREM A.6 (EXISTENCE OF ROBUST NETWORKS). *Let $f : C \rightarrow \mathbb{R}^n$ be a continuous function that is ϵ -robust on set M . Then, there exists a neural network that*

- (1) *agrees with f on M , i.e., $\forall \mathbf{x} \in M. \text{class}(N(\mathbf{x})) = \text{class}(f(\mathbf{x}))$, and*
- (2) *is ϵ -provably robust on M .*

PROOF. First, we need to post-process the results of f as follows: For all $\mathbf{x} \in C$,

$$\hat{f}(\mathbf{x}) = (0, \dots, |y_i|, \dots, 0)$$

where $\mathbf{y} = f(\mathbf{x})$ and $\text{class}(f(\mathbf{x})) = i$. In other words, \hat{f} is just like f , but it zeroes out the values of all but the output class i . This is needed since the interval domain is non-relational, and therefore it cannot capture relations between values of different classes, namely, keeping track which one is larger. Note that if f is continuous, then \hat{f} is continuous.

Let δ' be the smallest non-zero element of any vector in the set $\{\hat{f}(\mathbf{x}) \mid \mathbf{x} \in C\}$. Following the IUA theorem, let N be a neural network that δ -interval approximates \hat{f} , where $\delta < 0.5\delta'$.

STATEMENT (1): Pick any $\mathbf{x} \in M$. Let the i th element of $\hat{f}(\mathbf{x}) \neq 0$; call it c . By construction $i = \text{class}(f(\mathbf{x}))$. Let $N(\mathbf{x}) = (y_1, \dots, y_n)$. By IUA theorem, we know that $0 \leq y_j < 0.5\delta'$, for $j \neq i$, and $y_i \geq c - 0.5\delta'$. Since $c \geq \delta'$, $\text{class}(N(\mathbf{x})) = \text{class}(f(\mathbf{x})) = i$.

STATEMENT (2): Let $\mathbf{x} \in M$. Let $S = \hat{f}(R_\epsilon(\mathbf{x}))$. Let S_i be the projection of all vectors in S on their i th element, where $i = \text{class}(\hat{f}(\mathbf{x}))$. We know that $\min S_i \geq \delta'$. By construction of \hat{f} and the fact that f is robust, all other elements of vectors of S are zero, i.e., $S_j = \{0\}$, for $j \neq i$.

Let $N^\#(\alpha(R_\epsilon(\mathbf{x}))) = \langle [l_j, u_j] \rangle_j$. By IUA theorem and its proof, for $j \neq i$, we have $[l_j, u_j] \subset [0, 0.5\delta']$. Similarly, $[l_i, u_i] \subseteq [\min S_i - 0.5\delta', u_i] \subseteq [0.5\delta', u_i]$. It follows that for all $\mathbf{y}, \mathbf{y}' \in \gamma(N^\#(\alpha(R_\epsilon(\mathbf{x}))))$, we have $\text{class}(\mathbf{y}) = \text{class}(\mathbf{y}') = i$. This is because any value in $[\delta' - 0.5\delta', u_i]$ is larger than any value in $[0, 0.5\delta']$. □

B APPENDIX: ELIDED DEFINITIONS AND PROOFS

B.1 Correctness of Boolean Formula Encoding

Let ϕ be a Boolean formula and N_ϕ be its encoding neural network as described in Section 2.2. We will show that ϕ has the same semantics as N_ϕ , i.e.,

$$\text{for any } x \in \{0, 1\}^m, \phi(x) = N_\phi(x)$$

PROOF. We will prove the proposition by an induction on the number of operators (AND, OR, NOT), n , in the Boolean formula.

Base case. $n = 0$, the formula is only a variable X . The encoding network has no activation units and will output X .

Inductive hypothesis. Suppose all the formulas using $< k$ operators are captured by the corresponding neural network.

Inductive step. Let ϕ be a boolean formula with k operators, then ϕ can be in the following form:

$$\phi = \neg\phi_0 \text{ or } \phi = \phi_1 \wedge \phi_2 \text{ or } \phi = \phi_1 \vee \phi_2.$$

Notice that the number of operators in ϕ_0 , ϕ_1 and ϕ_2 are $< k$, so their corresponding encoding networks are correct according to the inductive hypothesis.

- (1) For $\phi = \neg\phi_0$, by the encoding rule, $N_\phi = 1 - N_{\phi_0}$. For any $x \in \{0, 1\}^m$, by definition, $\phi(x) = \neg\phi_0(x) = 1 - \phi_0(x)$.
By inductive hypothesis, we have $1 - \phi_0(x) = 1 - N_{\phi_0}(x) = N_\phi(x)$.
- (2) For $\phi = \phi_1 \wedge \phi_2$, by the encoding rule, $N_\phi = \text{step}(N_{\phi_1} + N_{\phi_2} - 1.5)$.
For any $x \in \{0, 1\}^m$, $\phi(x) = \phi_1(x) \wedge \phi_2(x)$. $\phi(x) = 1$ iff $\phi_1(x) = 1$ and $\phi_2(x) = 1$.
Because, by the inductive hypothesis, N_{ϕ_1} and N_{ϕ_2} have the same semantics as ϕ_1 and ϕ_2 , so the outputs of N_{ϕ_1} and N_{ϕ_2} are in $\{0, 1\}$. $N_\phi = \text{step}(N_{\phi_1} + N_{\phi_2} - 1.5)$ evaluate to 1 iff N_{ϕ_1} and N_{ϕ_2} both evaluate to 1.
- (3) For $\phi = \phi_1 \vee \phi_2$, the reasoning is essentially the same as the case $\phi = \phi_1 \wedge \phi_2$.

□

B.2 Proof of Theorem 3.4

All of the interval arithmetic operations we have defined are standard and are sound.

The only non-standard abstract transformers are $t^\#$. We start with the general definition and prove its soundness:

$$t^\#(B) = \left\langle \left[\min_{l \leq x \leq u} t(x), \max_{l \leq x \leq u} t(x) \right] \right\rangle$$

Let $B = \langle [l, u] \rangle$ be a 1-dimensional box. Since t satisfies Eq. (1), the lower bound and upper bound above exist. The collecting semantics $t(\gamma(B)) = \{t(x) \mid l \leq x \leq u\}$. It follows that $t(\gamma(B)) \subseteq t^\#(B)$.

If t is monotonically increasing, we defined the transformer

$$t^\#(B) = \langle [t(l), t(u)] \rangle$$

By monotonicity of t , we have $\forall x \in [l, u]. t(l) \leq t(x) \leq t(u)$. Therefore, $t(\gamma(B)) \subseteq [t(l), t(u)]$. It follows that $t(\gamma(B)) \subseteq t^\#(B)$.

Therefore all abstract transformers are sound. Soundness of $N^\#$ follows compositionally from soundness of all operators.

B.3 Choice of Parameters θ and ϵ

Because our construction works for any fixed θ and ϵ , we will choose $\theta = \min(\frac{1}{K+1}, \frac{1}{4m+2}, \frac{1}{4|\mathcal{G}|})$, where τ , K and \mathcal{G} are defined in Section 4.1; and $\epsilon < 0.5$ be such that if $\|x - y\|_\infty \leq \epsilon$, then $|f(x) - f(y)| < \tau$. The latter is achievable from the Heine–Cantor Theorem (see Rudin [1986]), so f is uniformly continuous on C .

B.4 Proof of Lemma 4.5

PROOF. STATEMENT (1): Because $x \geq a_i$, $x + 0.5\epsilon - a_i \geq 0.5\epsilon$. From Lemma 4.3, $t(\mu(x + 0.5\epsilon - a_i)) \in (1 - \theta, 1]$.

STATEMENT (2): Because $x \leq b_i$, $x - 0.5\epsilon - b_i \leq -0.5\epsilon$. From Lemma 4.3, $t(\mu(x - 0.5\epsilon - b_i)) \in [0, \theta)$.

□

B.5 Proof of Lemma 4.6

PROOF. STATEMENT (1): Because $x \leq a_i - \epsilon$, $x + 0.5\epsilon - a_i \leq -0.5\epsilon$. From Lemma 4.3, $t(\mu(x + 0.5\epsilon - a_i)) \in [0, \theta)$.

STATEMENT (2): Because $x \leq a_i - \epsilon$ and $a_i < b_i$, $x \leq b_i - \epsilon$. Then $x - 0.5\epsilon - b_i \leq -0.5\epsilon$. From Lemma 4.3, $t(\mu(x - 0.5\epsilon - b_i)) \in [0, \theta)$.

□

B.6 Proof of Lemma 4.7

PROOF. STATEMENT (1): Because $x \geq b_i + \epsilon$ and $b_i \geq a_i$, $x \geq a_i + \epsilon$. Then $x + 0.5\epsilon - a_i \geq 0.5\epsilon$. From Lemma 4.3, $t(\mu(x + 0.5\epsilon - a_i)) \in (1 - \theta, 1]$.

STATEMENT (2): Because $x \geq b_i + \epsilon$, $x - 0.5\epsilon - b_i \geq 0.5\epsilon$. From Lemma 4.3, $t(\mu(x - 0.5\epsilon - b_i)) \in (1 - \theta, 1]$.

□

B.7 Proof of Lemma 4.8

PROOF. We begin the proof by simplifying the expression $\hat{t}^\#(B)$. Recall that $\hat{t}(x) = t(\mu(x + 0.5\epsilon - a_i)) - t(\mu(x - 0.5\epsilon - b_i))$. Let $B = \langle [a, b] \rangle$. By applying abstract transformer $t^\#$ (Definition 3.2) and subtracting the two terms, we get $\hat{t}^\#(B) = [T_1 - T_4, T_2 - T_3]$, where

$$\begin{aligned} T_1 &= \min_{x \in [a, b]} t(\mu(x + 0.5\epsilon - a_i)) & T_2 &= \max_{x \in [a, b]} t(\mu(x + 0.5\epsilon - a_i)) \\ T_3 &= \min_{x \in [a, b]} t(\mu(x - 0.5\epsilon - b_i)) & T_4 &= \max_{x \in [a, b]} t(\mu(x - 0.5\epsilon - b_i)) \end{aligned}$$

We are now ready to prove the three statements.

STATEMENT (1): By the limits of t , $\forall x. t(x) \in [0, 1]$, so $T_1, T_2, T_3, T_4 \in [0, 1]$. Therefore, the upper bound of $\hat{t}^\#(B)$ is $T_2 - T_3 \leq 1$.

STATEMENT (2):

Case 1: $B \subseteq (-\infty, a_i - \epsilon]$. From Lemma 4.6, $T_1, T_2, T_3, T_4 \in [0, \theta]$, then $T_2 - T_3 < \theta$, and $T_1 - T_4 > -\theta$.

Case 2: $B \subseteq [b_i + \epsilon, \infty)$. From Lemma 4.7, $T_1, T_2, T_3, T_4 \in (1 - \theta, 1]$, then $T_2 - T_3 < \theta$, and $T_1 - T_4 > -\theta$.

In either case, $\hat{t}^\#(B) \subseteq (-\theta, \theta)$.

STATEMENT (3): If $B \subseteq [a_i, b_i]$, $a, b \in [a_i, b_i]$. From Lemma 4.5(1), $T_1, T_2 \in (1 - \theta, 1]$. From Lemma 4.5(2), $T_3, T_4 \in [0, \theta]$. Then $T_1 - T_4 > 1 - 2\theta$ and $T_2 - T_3 \leq 1$.

Therefore, $\hat{t}^\#(B) \subseteq (1 - 2\theta, 1]$.

□

B.8 Proof of Lemma 4.10

PROOF. STATEMENT (1): If $B \subseteq G$, then $\forall i. B_i \subseteq [a_i, b_i]$. From Lemma 4.8 (3), $\hat{t}_i^\#(B_i) \subseteq (1 - 2\theta, 1]$; thus,

$$\begin{aligned} H_i^\#(B_i) &= \hat{t}_i^\#(B_i) +^\# -(1 - 2\theta)^\# \\ &\subseteq (0, 2\theta] \\ &\subset (0, \infty) \end{aligned}$$

Sum over all m dimensions, $\sum_{i=1}^m H_i^\#(B_i) \subseteq \sum_{i=1}^m (0, \infty) = (0, \infty)$.

STATEMENT (2): If $B \subseteq C \setminus v(G)$, then there is a dimension j such that either $B_j \subseteq (-\infty, a_j - \epsilon]$ or $B_j \subseteq [b_j + \epsilon, \infty)$. From Lemma 4.8 (2), we know that $\hat{t}^\#(B_j) \subseteq (-\theta, \theta)$. Therefore,

$$\begin{aligned} H_j^\#(B_j) &= \hat{t}^\#(B_j) +^\# -(1 - 2\theta)^\# \\ &\subseteq (\theta - 1, 3\theta - 1) \end{aligned} \tag{11}$$

For the remaining $m - 1$ dimensions, from Lemma 4.8 (1), we know that $\hat{t}^\#(B_i) \subset (-\infty, 1]$ when $i \neq j$. Therefore,

$$\begin{aligned} H_i^\#(B_i) &= \hat{t}^\#(B_i) +^\# -(1 - 2\theta)^\# \\ &\subseteq (-\infty, 2\theta] \end{aligned} \tag{12}$$

Take the sum of all the $m - 1$ dimensions,

$$\begin{aligned} \sum_{i \in \{1, \dots, m\} \setminus \{j\}} H_i^\#(B_i) &\subseteq \sum_{i \in \{1, \dots, m\} \setminus \{j\}} (-\infty, 2\theta] && \text{(substitute Eq. (12))} \\ &= [m - 1, m - 1] \# (-\infty, 2\theta] && \text{(turn sum into } \# \text{)} \\ &= (-\infty, 2(m - 1)\theta] && \text{(apply } \# \text{)} \end{aligned} \quad (13)$$

Now, take sum over all the m dimensions,

$$\begin{aligned} \sum_{i=1}^m H_i^\#(B_i) &= \sum_{i \in \{1, \dots, m\} \setminus \{j\}} H_i^\#(B_i) +^\# H_j^\#(B_j) && \text{(decompose sum)} \\ &\subseteq (-\infty, 2(m - 1)\theta] +^\# (\theta - 1, 3\theta - 1) && \text{(substitute Eqs. (11) and (13))} \\ &= (-\infty, (2m + 1)\theta - 1) && \text{(apply } \# \text{)} \end{aligned}$$

Because of our choice of θ , $\theta \leq \frac{1}{4m+2}$ (see Appendix B.3). Then $(2m + 1)\theta \leq \frac{2m+1}{4m+2} = 0.5$, and therefore

$$\sum_{i=1}^m H_i^\#(B_i) \subseteq (-\infty, -0.5)$$

Also we have assumed that $\epsilon < 0.5$ (see Appendix B.3); therefore

$$\sum_{i=1}^m H_i^\#(B_i) \subseteq (-\infty, -\epsilon)$$

□

B.9 Abstract Interpretation of N_i

Observe how for any box $B \subseteq C$ from the abstract domain, it is overapproximated by a larger box $G \supseteq B$ from the finitely many boxes in the ϵ -grid. Intuitively, our abstract approximation of N_i incurs an error when the input B is not in the grid. We formalize this idea by extending the notion of neighborhood (Section 4.2) to boxes from the abstract domain. For a box $B \subseteq C$, if $B \in \mathcal{G}$, then B 's neighborhood $G_B = v(B)$; otherwise, let G_B be the smallest $G \in \mathcal{G}$, by volume, such that $B \subseteq G$. Note that G_B is uniquely defined.

The following lemma says that considering the neighborhood of B only adds up to τ of imprecision to the collecting semantics of f .

LEMMA B.1 (PROPERTIES OF G_B). *The following is true:*

- (1) If $f(B) \geq \beta$, then $f(G_B) \geq \beta - \tau$.
- (2) If $f(B) \leq \beta$, then $f(G_B) \leq \beta + \tau$.

PROOF. Both of the statements follow from our choice of ϵ in constructing the grid (see Appendix B.3). If $\|\mathbf{x} - \mathbf{y}\|_\infty \leq \epsilon$, then $|f(\mathbf{x}) - f(\mathbf{y})| < \tau$. Consider the B and its neighborhood G_B . By definition of neighborhood, $\forall \mathbf{x} \in G_B, \exists \mathbf{y} \in B$, such that $\|\mathbf{x} - \mathbf{y}\|_\infty \leq \epsilon$.

STATEMENT (1) Because $f(B) \geq \beta$, then $f(\mathbf{y}) \geq \beta$, so $f(\mathbf{x}) \geq f(\mathbf{y}) - \tau \geq \beta - \tau$. Then $\forall \mathbf{x} \in G_B, f(\mathbf{x}) \geq \beta - \tau$.

STATEMENT (2) Because $f(B) \leq \beta$, then $f(\mathbf{y}) \leq \beta$, so $f(\mathbf{x}) \leq f(\mathbf{y}) + \tau \leq \beta + \tau$. Then $\forall \mathbf{x} \in G_B, f(\mathbf{x}) \leq \beta + \tau$. □

THEOREM B.2 (ABSTRACT INTERPRETATION OF N_i). *For any box $B \subseteq C$, let $u = \max f(B)$, and $l = \min f(B)$. The following is true:*

- (1) $N_i^\#(B) \subseteq [0, 1]$.
- (2) If $l \geq (i + 2)\tau$, then $\exists u_i \in (1 - \theta, 1]$ such that $[u_i, u_i] \subseteq N_i^\#(B) \subseteq (1 - \theta, 1]$.
- (3) If $u \leq (i - 1)\tau$, then $\exists l_i \in [0, \theta)$ such that $[l_i, l_i] \subseteq N_i^\#(B) \subseteq [0, \theta)$.

PROOF. We begin by noting that in Statement (2), $[u_i, u_i] \subseteq N_i^\#(B)$ for some $u_i \in (1 - \theta, 1]$ is a direct corollary of $N_i^\#(B) \subseteq (1 - \theta, 1]$. Because if $N_i^\#(B) \subseteq (1 - \theta, 1]$, and $N_i^\#(B) \neq \emptyset$, then $N_i^\#(B)$ contains at least one point in $(1 - \theta, 1]$. Similarly, in Statement (3), $[l_i, l_i] \subseteq N_i^\#(B)$ for some $l_i \in [0, \theta)$ is a direct corollary of $N_i^\#(B) \subseteq [0, \theta)$.

In Appendix B.3, we have chosen that $\theta \leq \frac{1}{4|\mathcal{G}|}$, a fact we will use later in the proof.

STATEMENT (1): The outer function of N_i is t , whose range is $[0, 1]$, by the definition of squashable function and our construction, so $N_i^\#(B) \subseteq [0, 1]$.

STATEMENT (2): Because $f(B) \geq (i + 2)\tau$, by Lemma B.1, $f(G_B) \geq (i + 1)\tau$, so $G_B \in \mathcal{G}_i$. Thus, we can break up the sum as follows:

$$\sum_{G \in \mathcal{G}_i} N_G(\mathbf{x}) = \left(\sum_{G \in (\mathcal{G}_i \setminus \{G_B\})} N_G(\mathbf{x}) \right) + N_{G_B}(\mathbf{x})$$

From Theorem 4.11, $N_{G_B}^\#(B) \subseteq (1 - \theta, 1]$, and $N_G^\#(B) \subseteq [0, 1]$ for $G \in \mathcal{G}_i \setminus \{G_B\}$. Therefore, we can conclude the following two facts:

$$\sum_{G \in \mathcal{G}_i} N_G^\#(B) \subseteq (1 - \theta, \infty) \quad \text{and} \quad \sum_{G \in \mathcal{G}_i} N_G^\#(B) +^\# [-0.5, -0.5] \subseteq (0.5 - \theta, \infty) \subset (0.5\epsilon, \infty)$$

The second inequality follows from the fact that we assumed $\theta \leq \frac{1}{4|\mathcal{G}|} \leq 0.25$ (above) and $\epsilon < 0.5$ (see Appendix B.3). Therefore, $0.5 - \theta > 0.25 > 0.5\epsilon$.

It follows from Lemma 4.3 that

$$N_i^\#(B) = t^\# \left(\mu^\# *^\# \left(\sum_{G \in \mathcal{G}_i} N_G^\#(B) +^\# [-0.5, -0.5] \right) \right) \subseteq (1 - \theta, 1]$$

STATEMENT (3): If $u \leq (i - 1)\tau$, we will show that $\forall G \in \mathcal{G}_i. B \subset C \setminus \nu(G)$.

Pick any $G \in \mathcal{G}_i$, then we have $f(G) \geq (i + 1)\tau$. Thus, from Lemma B.1, $f(G_B) \geq i\tau$. Recall that if $B \in \mathcal{G}$, then $G_B = \nu(B)$. Hence, $f(\nu(G)) \geq i\tau$. However, $f(B) \leq u \leq (i - 1)\tau$, so $B \cap \nu(G) = \emptyset$. Equivalently, $B \subset C \setminus \nu(G)$.

From Theorem 4.11, $\forall G \in \mathcal{G}_i. N_G^\#(B) \subseteq [0, \theta)$, so

$$\sum_{G \in \mathcal{G}_i} N_G^\#(B) \subseteq [0, |\mathcal{G}_i|\theta] \subseteq [0, |\mathcal{G}|\theta)$$

We assumed that $\theta \leq \frac{1}{4|\mathcal{G}|}$ and $\epsilon < 0.5$ (see Appendix B.3), so $|\mathcal{G}|\theta \leq 0.25$, and $|\mathcal{G}|\theta - 0.5 \leq -0.25 \leq -0.5\epsilon$. Hence,

$$\sum_{G \in \mathcal{G}_i} N_G^\#(B) \subseteq [0, 0.25) \quad \text{and} \quad \sum_{G \in \mathcal{G}_i} N_G^\#(B) +^\# [-0.5, -0.5] \subseteq [-0.5, -0.25) \subseteq (-\infty, -0.5\epsilon)$$

It follows from Lemma 4.3 that

$$N_i^\#(B) = t^\# \left(\mu^\# *^\# \left(\sum_{G \in \mathcal{G}_i} N_G^\#(B) +^\# [-0.5, -0.5] \right) \right) \subseteq [0, \theta)$$

□

B.10 Abstract Interpretation of N

Because $\sum_{i=0}^K f_i(\mathbf{x}) = f(\mathbf{x})$, and $N_i(\mathbf{x})$ approximates $\frac{1}{\tau} f_i(\mathbf{x})$, we will construct the neural network N as $N(\mathbf{x}) = \tau \sum_{i=0}^K N_i(\mathbf{x})$.

Before proceeding with the proof, we give a general lemma that will be useful in our analysis. The lemma follows from the fact that, by construction, $\theta \leq \frac{1}{K+1}$.

LEMMA B.3. *If $\eta_0, \dots, \eta_K \in [-\theta, \theta]$, then $\sum_{i=0}^K \eta_i \in [-1, 1]$.*

PROOF. This simply follow from the choice of $\theta \leq \frac{1}{K+1}$. \square

Proof outline of the existence of δ -interval approximating neural networks. Our proof involves three pieces, outlined below:

- (A) Because $N^\#(B) = \tau \sum_{i=0}^K N_i^\#(B)$, we need only analyze $\sum_{i=0}^K N_i^\#(B)$. We will decompose the sum into five sums and analyze each separately, arriving at five results of the form:

$$[\tilde{L}_{1j}, \tilde{U}_{1j}] \subseteq \sum_{i \in S_j} N_i^\#(B) \subseteq [\tilde{L}_{2j}, \tilde{U}_{2j}]$$

for $j \in \{1, \dots, 5\}$, where $\bigcup_j S_j = \{0, \dots, K\}$ and S_j are mutually disjoint sets.

- (B) Then, we sum over all five cases, getting

$$\left[\sum_{j=1}^5 \tilde{L}_{1j}, \sum_{j=1}^5 \tilde{U}_{1j} \right] \subseteq \sum_{i=0}^K N_i^\#(B) \subseteq \left[\sum_{j=1}^5 \tilde{L}_{2j}, \sum_{j=1}^5 \tilde{U}_{2j} \right]$$

- (C) Let $L_i = \tau \sum_{j=1}^5 \tilde{L}_{ij}$ and $U_i = \tau \sum_{j=1}^5 \tilde{U}_{ij}$. Then, we get the bound $[L_1, U_1] \subseteq N^\#(B) \subseteq [L_2, U_2]$. Finally, we show that $[L_2, U_2] \subseteq [l - \delta, u + \delta]$ and $[l + \delta, u - \delta] \subseteq [L_1, U_1]$. Equivalently, we will show that

$$l - \delta \leq L_2 \leq L_1 \leq l + \delta \quad \text{and} \quad u - \delta \leq U_1 \leq U_2 \leq u + \delta$$

Proof assumptions. We will assume that $l \in [p\tau, (p+1)\tau]$ and $u \in [q\tau, (q+1)\tau]$, for some $p \leq q \leq K$. Additionally, let $c, d \in B$ be such that $f(c) = l$ and $f(d) = u$.

Step A: Decompose sum and analyze separately. We begin by decomposing the sum into five terms.

This is the most important step of the proof. We want to show that most $N_i^\#$'s in $\sum_{i=0}^K N_i^\#(B)$ are (almost) precise. By almost we mean that their values are ≈ 1 and ≈ 0 . The motivation is then to extract as many precise terms as possible. The only tool used in the analysis is Theorem B.2.

- Consider the function slices represented by Term 1 and 5; for example, Term 1 represents abstractions $N_i^\#$ of function slices f_i , for $i \in [0, p-2]$. The function slices of Term 1 and 5 are referred to in Theorem B.2 (Statements 2 and 3): they have an (almost) precise abstract interpretation. That is, the abstract semantics of $N_i^\#(B)$ and the collecting semantics of $f_i(B)$ agree. For Term 1, the abstract interpretation of all $N_i^\#(B) \approx [1, 1]$ and $f_i(B) = [\tau, \tau]$. For Term 5, the abstract interpretation of all $N_i^\#(B) \approx [0, 0]$ and $f_i(B) = [0, 0]$.
- Now consider function slices f_i , where $i \in [p+2, q-2]$. The abstraction of these function slices is also (almost) precise. We can see $f(c) = l$ is below the lower bound of the slices and $f(d) = u$ is above the upper bound of the slices. Hence, $f_i(d) = \tau$ and $N_i^\#(\{d\}) \approx [1, 1]$. Similarly, $f_i(c) = 0$ and $N_i^\#(\{c\}) \approx [0, 0]$. Because $c, d \in B$, and due to continuity of f , we have $f_i(B) = [0, 1]$, and $N_i^\#(B) \approx [0, 1]$.
- The remaining function slices are those in Term 2 and Term 4, and they are at the neighborhood of the boundary of $[l, u]$. Most precision loss of $N_i^\#(B)$ comes from those two terms.

This drives us to decompose the sum as follows:

$$\sum_{i=0}^K N_i^\#(B) = \underbrace{\sum_{i=0}^{p-2} N_i^\#(B)}_{\text{Term 1}} + \underbrace{\sum_{i=p-1}^{p+1} N_i^\#(B)}_{\text{Term 2}} + \underbrace{\sum_{i=p+2}^{q-2} N_i^\#(B)}_{\text{Term 3}} + \underbrace{\sum_{i=q-1}^{q+1} N_i^\#(B)}_{\text{Term 4}} + \underbrace{\sum_{i=q+2}^K N_i^\#(B)}_{\text{Term 5}} \quad (14)$$

We will analyze the five terms in Eq. (14) separately, and then take their sum to get the final result. For now, assume that $q \geq p+3$; the $q \leq p+2$ case will follow easily.

- (i) Term 1: $\forall i \leq p-2$, we have $p\tau \geq (i+2)\tau$. Because $l = \min f(B)$ and $l \in [p\tau, (p+1)\tau)$, then $f(B) \geq p\tau \geq (i+2)\tau$.

From Theorem B.2, $\exists u_i \in (1-\theta, 1]$ such that $[u_i, u_i] \subseteq N_i^\#(B) \subseteq (1-\theta, 1]$. Then $\sum_{i=0}^{p-2} [u_i, u_i] \subseteq \sum_{i=0}^{p-2} N_i^\#(B) \subseteq \sum_{i=0}^{p-2} (1-\theta, 1]$.

$$\sum_{i=0}^{p-2} [u_i, u_i] \subseteq \sum_{i=0}^{p-2} N_i^\#(B) \subseteq (p-1)^\# *^\# (1-\theta, 1]$$

- (ii) Term 5: $\forall i \geq q+2$, we have $(q+1)\tau \leq (i-1)\tau$. Because $u = \max f(B)$ and $u \in [q\tau, (q+1)\tau)$, then $f(B) < (q+1)\tau \leq (i-1)\tau$.

From Theorem B.2, $\exists l_i \in [0, \theta)$ such that $[l_i, l_i] \subseteq N_i^\#(B) \subseteq [0, \theta)$. Then $\sum_{i=q+2}^K [l_i, l_i] \subseteq \sum_{i=q+2}^K N_i^\#(B) \subseteq \sum_{i=q+2}^K [0, \theta)$.

$$\sum_{i=q+2}^K [l_i, l_i] \subseteq \sum_{i=q+2}^K N_i^\#(B) \subseteq (K-q-1)^\# *^\# [0, \theta)$$

- (iii) Term 3: $\forall i \in [p+2, q-2]$, we have $(p+1)\tau \leq (i-1)\tau$ and $q\tau \geq (i+2)\tau$.

$f(c) = l < (p+1)\tau \leq (i-1)\tau$, and $f(d) = u \geq q\tau \geq (i+2)\tau$.

From Theorem B.2, $N_i^\#(\{c\}) \subseteq [0, \theta)$ and $N_i^\#(\{d\}) \subseteq (1-\theta, 1]$. Because $c, d \in B$, $[\theta, 1-\theta] \subseteq N_i^\#(B)$.

Also by Theorem B.2, $N_i^\#(B) \subseteq [0, 1]$. Hence, $\sum_{i=p+2}^{q-2} [\theta, 1-\theta] \subseteq \sum_{i=p+2}^{q-2} N_i^\#(B) \subseteq \sum_{i=p+2}^{q-2} [0, 1]$.

$$\sum_{i=p+2}^{q-2} [\theta, 1-\theta] \subseteq \sum_{i=p+2}^{q-2} N_i^\#(B) \subseteq (q-p-3)^\# *^\# [0, 1]$$

- (iv) Term 2: $\forall i \in [p-1, p+1]$, since we have assumed that $q \geq p+3$, then $q \geq p+3 \geq i+2$.

Because $f(d) \geq q\tau \geq (i+2)\tau$, from Theorem B.2, $\exists u_i \in (1-\theta, 1]$ such that $[u_i, u_i] \subseteq N_i^\#(\{d\}) \subseteq (1-\theta, 1]$.

Because $d \in B$, $[u_i, u_i] \subseteq N_i^\#(B)$. Hence, $[u_i, u_i] \subseteq N_i^\#(B) \subseteq [0, 1]$ and $\sum_{i=p-1}^{p+1} [u_i, u_i] \subseteq \sum_{i=p-1}^{p+1} N_i^\#(B) \subseteq \sum_{i=p-1}^{p+1} [0, 1]$.

$$\sum_{i=p-1}^{p+1} [u_i, u_i] \subseteq \sum_{i=p-1}^{p+1} N_i^\#(B) \subseteq 3^\# *^\# [0, 1]$$

- (v) Term 4: For $\forall q-1 \leq i \leq q+1$, because $q \geq p+3$, we have $p+1 \leq q-2 \leq i-1$. Then $f(c) = l < (p+1)\tau \leq (i-1)\tau$. From Theorem B.2, $\exists l_i \in [0, \theta)$ such that $[l_i, l_i] \subseteq N_i^\#(\{c\}) \subseteq [0, \theta)$.

Because $c \in B$, $[l_i, l_i] \subseteq N_i^\#(B)$. Thus, $[l_i, l_i] \subseteq N_i^\#(B) \subseteq [0, 1]$.

$$\sum_{i=q-1}^{q+1} [l_i, l_i] \subseteq \sum_{i=q-1}^{q+1} N_i^\#(B) \subseteq 3^\# *^\# [0, 1]$$

Step B: Sum all five cases. We now sum up all five inequalities we derived above to derive an overall bound of the sum in the form $[L'_1, U'_1] \subseteq \sum_{i=0}^K N_i^\#(B) \subseteq [L'_2, U'_2]$. For example,

$$L'_1 = \sum_{i=0}^{p-2} u_i + \sum_{i=q+2}^K l_i + \sum_{i=p+2}^{q-2} \theta + \sum_{i=p-1}^{p+1} u_i + \sum_{i=q-1}^{q+1} l_i$$

Recall that, by Theorem B.2, $\forall i \in \{0, \dots, K\}$, $u_i \in (1 - \theta, 1]$ and $l_i \in [0, \theta)$. Let $\tilde{l}_i = 1 - u_i$, so $\tilde{l}_i \in [0, \theta)$.

We simplify L'_1, L'_2, U'_1 and U'_2 as follows:

$$\begin{aligned} L'_1 &= \sum_{i=0}^{p-2} u_i + \sum_{i=q+2}^K l_i + \sum_{i=p+2}^{q-2} \theta + \sum_{i=p-1}^{p+1} u_i + \sum_{i=q-1}^{q+1} l_i && \text{(sum of the left bound)} \\ &= \sum_{i=0}^{p-2} (1 - \tilde{l}_i) + \sum_{i=q+2}^K l_i + \sum_{i=p+2}^{q-2} \theta + \sum_{i=p-1}^{p+1} (1 - \tilde{l}_i) + \sum_{i=q-1}^{q+1} l_i \\ &= (\sum_{i=0}^{p-2} + \sum_{i=p-1}^{p+1})(1) + \sum_{i=0}^{p-2} (-\tilde{l}_i) + \sum_{i=q+2}^K l_i + \sum_{i=p+2}^{q-2} \theta + \sum_{i=p-1}^{p+1} (-\tilde{l}_i) + \sum_{i=q-1}^{q+1} l_i && \text{(substitute } u_i \text{ with } \tilde{l}_i) \\ &= (p + 2) + \sum_{i=0}^{p+1} (-\tilde{l}_i) + \sum_{i=q-1}^K l_i + \sum_{i=p+2}^{q-2} \theta && \text{(Rearrange the terms)} \\ & && \text{(Sum all the 1's)} \end{aligned}$$

From Lemma B.3, $\sum_{i=0}^{p+1} (-\tilde{l}_i) + \sum_{i=p+2}^{q-2} \theta + \sum_{i=q-1}^K l_i \in [1, 1]$ by plugging in $-\tilde{l}_i, l_i, \theta$ to η_i . So,

$$L'_1 \in [p + 1, p + 3]$$

$$\begin{aligned} U'_1 &= \sum_{i=0}^{p-2} u_i + \sum_{i=q+2}^K l_i + \sum_{i=p+2}^{q-2} (1 - \theta) + \sum_{i=p-1}^{p+1} u_i + \sum_{i=q-1}^{q+1} l_i && \text{(sum of right bound)} \\ &= \sum_{i=0}^{p-2} (1 - \tilde{l}_i) + \sum_{i=q+2}^K l_i + \sum_{i=p+2}^{q-2} (1 - \theta) + \sum_{i=p-1}^{p+1} (1 - \tilde{l}_i) + \sum_{i=q-1}^{q+1} l_i && \text{(substitute } u_i \text{ with } \tilde{l}_i) \\ &= (q - 1) + \sum_{i=0}^{p+1} (-\tilde{l}_i) + \sum_{i=q-1}^K l_i + \sum_{i=p+2}^{q-2} (-\theta) && \text{(sum all the 1's)} \end{aligned}$$

From Lemma B.3, $\sum_{i=0}^{p+1} (-\tilde{l}_i) + \sum_{i=p+2}^{q-2} (-\theta) + \sum_{i=q-1}^K l_i \in [-1, 1]$. Thus,

$$U'_1 \in [q - 2, q]$$

$$\begin{aligned} L'_2 &= (p - 1)(1 - \theta) && \text{(sum of left bound)} \\ &= (p - 1) + (p - 1)(-\theta) && \text{(rearrange terms)} \end{aligned}$$

Because $\theta \leq \frac{1}{K+1}$, and $-K \leq p - 1 \leq K$, we have $(p - 1)(-\theta) \in [-1, 1]$. Hence,

$$L'_2 \in [p - 2, p]$$

$$\begin{aligned} U'_2 &= (p - 1) + (K - q - 1)\theta + (q - p - 3) + 3 + 3 && \text{(sum of right bound)} \\ &= (p - 1 + q - p - 3 + 3 + 3) + (K - q - 1)(\theta) && \text{(rearrange terms)} \\ &= q + 2 + (K - q - 1)(\theta) && \text{(sum all the 1's)} \end{aligned}$$

Because $\theta \leq \frac{1}{K+1}$, and $-K \leq (K - q - 1) \leq K$, we have $(K - q - 1)(\theta) \in [-1, 1]$. Then,

$$U'_2 \in [q + 1, q + 3]$$

Step C: Analyze the bound. It remains to show that $l - \delta \leq L_2 \leq L_1 \leq l + \delta$ and $u - \delta \leq U_1 \leq U_2 \leq u + \delta$.

Recall that we have set that $\delta = 3\tau$. Also $l \in [p\tau, (p + 1)\tau)$, then

$$l - \delta < (p - 2)\tau \quad \text{and} \quad l + \delta \geq (p + 3)\tau$$

Since $u \in [q\tau, (q+1)\tau]$, then

$$u - \delta < (q-2)\tau \quad \text{and} \quad u + \delta \geq (q+3)\tau$$

We have just analyzed L'_1, L'_2, U'_1 and U'_2 above. Now we have:

$$\begin{aligned} L_1 &= \tau L'_1 \leq (p+3)\tau & L_2 &= \tau L'_2 \geq (p-2)\tau \\ U_1 &= \tau U'_1 \geq (q-2)\tau & U_2 &= \tau U'_2 \leq (q+3)\tau \end{aligned}$$

It follows from the above inequalities that

$$l - \delta < (p-2)\tau \leq L_2 \leq L_1 \leq (p+3)\tau \leq l + \delta$$

and

$$u - \delta < (q-2)\tau \leq U_1 \leq U_2 \leq (q+3)\tau \leq u + \delta$$

This concludes the proof for the case where $q \geq p+3$.

Excluded case. Previously, we have shown that Terms 1, 3, and 5 are almost precise. The imprecise terms can only come from Terms 2 and 4. If $q \leq p+2$, the only analyses that will be affected are those of Terms 2 and 4. Since $q \leq p+2$, we have $p+1 \geq q-1$, which means Terms 2 and 4 have potentially less sub-terms in this case. Thus imprecise terms are less than the $q \geq p+3$ case and we can apply the same analysis as above and derive the same bound.

We have thus shown that the neural network N that we construct δ -interval approximates f , and therefore the IUA theorem is true.

B.11 Real-valued computational model

We consider the standard Turing machine, and when we use *polynomial*, we mean polynomial in terms of m , the dimensions of the input space. The input x_i has finite precision, which means the bit-representation of the input takes finite space, say IEEE 754 float. The bit-representation of δ and k are polynomial in terms of m in Eqs. (7) to (9). Note that the floating point arithmetics can invalidate a theoretically sound verifier in practice, as shown in [Jia and Rinard \[2021\]](#); however, because we are considering the theoretical complexity of the RA problem, floating point arithmetics is not a concern. Our goal in the complexity part is the hardness result—range approximating a neural network is both NP-hard and coNP-hard—it does not rely on the assumption that the input has finite precision and the neural network is poly-time executable.

Now each activation function is a conditional function—if-else function, on multiplications and additions of polynomial-sized bits—so the execution of the activation function takes poly-time.

To give a concrete example of the activation, let $k = 2m$. In Eq. (8), we only require that the right limit is between $1/(2k)$ and $1/k$, and there is at least one number in the range that can be represented using finite bits. Without loss of generality, we can assume that $1/(2m)$ can be represented using finite bits. Now we can transform Eq. (8) to the following conditional function:

- (1) if $z \geq 0.1$: return $1/(2m)$;
- (2) elif $z \leq 0$: return -1 ;
- (3) else: return $10(1/(2m) + 1)z - 1$.

The execution of this function takes poly-time because the operations in the function take poly-time in terms of the bit-representations of $1/(2m)$ and z .

Because the 3SAT instance has poly-size, the corresponding neural network also has polynomial size in the bit-representation, and its execution takes polynomial time. The reduction itself also only takes polynomial-time: it is merely scanning the Boolean formula, and then constructing the network accordingly. Again, if we do not impose the finite-precision and polynomial-time-execution assumptions, the NP-hardness and coNP-hardness results are still true.

B.12 coNP-hardness of Range Approximation

To show the coNP-hardness of the range approximation problem, we will show that approximating the minimum value of a neural network can decide whether a 3DNF formula is a tautology. For $\delta < 1/2$, let

$$G_\delta^+ = \bigcup_{m \geq 1} \left\{ f \in F_m \mid \min_{\mathbf{x} \in [0,1]^m} (f(\mathbf{x})) \geq 1/2 + \delta \right\} \quad (15)$$

$$G_\delta^- = \bigcup_{m \geq 1} \left\{ f \in F_m \mid \min_{\mathbf{x} \in [0,1]^m} (f(\mathbf{x})) < 1/2 - \delta \right\} \quad (16)$$

LEMMA B.4. *Given $f \in G_\delta^+ \cup G_\delta^-$, it is coNP-hard to determine whether $f \in G_\delta^+$ or $f \in G_\delta^-$.*

A 3DNF instance ϕ is a disjunction of clauses of the form $C_1 \vee \dots \vee C_k$, where each clause C_j is a conjunction of 3 literals.

$$t_4(z) = \begin{cases} \leq -0.5 \text{ and } \geq -0.6, & z \leq 0.3 \\ \geq 0.1 \text{ and } \leq 0.2, & z \geq 0.4 \end{cases} \quad (17)$$

$$t_5(z) = \begin{cases} \geq -\frac{1}{2k} \text{ and } \leq -\frac{1}{4k}, & z \leq 0 \\ \geq 1, & z \geq 0.1 \end{cases} \quad (18)$$

We simulate the 3DNF instance using a neural network in the following way. For each variable X_i , construct an input node x_i . Simulate the negation operator using $l_i = 1 - x_i$. If there is no negation operator for l_i , we use $l_i = x_i$ directly. Then transform each literal using t_4 .

For each conjunction operator, we will use t_5 to control the output value. For example, if $C_j = L_{j1} \wedge L_{j2} \wedge L_{j3}$, build the gadget $c_j = t_5(t_4(l_{j1}) + t_4(l_{j2}) + t_4(l_{j3}))$. For the disjunction operator, we will use t_3 . For example, if $\phi = \bigvee_{i=1}^k C_i$, then let $y = t_3(\sum_{i=1}^k c_i)$.

PROPOSITION B.5. *For a 3DNF instance ϕ , let N_ϕ be the encoding neural network. Let $y_l = \min N_\phi([0, 1]^m)$. The following two statements are true:*

- (1) *If the 3DNF instance ϕ is not a tautology, then $y_l < 1/2 - \delta$.*
- (2) *If ϕ is a tautology, then $y_l \geq 1/2 + \delta$.*

PROOF. STATEMENT (1): If ϕ is not a tautology, let v_i be an unsatisfying assignment of X_i and use them as the input to N_ϕ . All clauses are evaluated 0, thus at least one literal from each clause are valued 0. WLOG, assume $L_{j1} = 0$ for $j = 1, \dots, k$. Therefore, $t_4(l_{j1}) \leq -0.5$, and $t_4(l_{j1}) + t_4(l_{j2}) + t_4(l_{j3}) \leq -0.5 + 0.2 + 0.2 = -0.1$. $c_j = t_5(t_4(l_{j1}) + t_4(l_{j2}) + t_4(l_{j3})) \leq -\frac{1}{4k}$. Therefore, $\sum_{i=1}^k c_i \leq k \sum_{i \neq j} c_i \leq -1/4$, then $y = t_3(\sum_{i=1}^k c_i) < 1/2 - \delta$, and so $y_l < 1/2 - \delta$.

STATEMENT (2): We will prove that if $y_l < 1/2 + \delta$, then ϕ is not a tautology. Let \mathbf{z} be such that $N_\phi(\mathbf{z}) < 1/2 + \delta$. For each $i \in \{1, \dots, m\}$, if $z_i \geq 0.4$, let $x_i = 1$; otherwise, let $x_i = 0$. We will show that \mathbf{x} is an unsatisfying assignment for ϕ .

We need to show that for all clause C_j , the assignment makes C_j false. Equivalently, there exists at least one literal in C_j that is false. Let us consider the corresponding gadget c_j in the network. Because $N_\phi(\mathbf{z}) < 1/2 + \delta$, from Eq. (9) and the construction $y = t_3(\sum_{i=1}^k c_i)$, then $\sum_{i=1}^k c_i(\mathbf{z}) < 0.5$.

This implies that for all $j \in \{1, \dots, k\}$ such that $c_j(\mathbf{z}) < 1$. Otherwise the remaining $k - 1$ gadgets are at least $-\frac{1}{2k}$, $\sum_{i=1}^k c_i(\mathbf{z}) \geq 1 - (k - 1)\frac{1}{2k} \geq 0.5$.

Because $c_j(\mathbf{z}) < 1$ and $c_j = t_5(t_4(l_{j1}) + t_4(l_{j2}) + t_4(l_{j3}))$, $[t_4(l_{j1}) + t_4(l_{j2}) + t_4(l_{j3})](\mathbf{z}) < 0.1$.

For all the three literals L_{j1}, L_{j2}, L_{j3} in C_j , consider the three gadgets corresponding to them in the neural network. Because $[t_4(l_{j1}) + t_4(l_{j2}) + t_4(l_{j3})](\mathbf{z}) < 0.1$, at least one literal is valued < 0.4 . Otherwise, the $[t_4(l_{j1}) + t_4(l_{j2}) + t_4(l_{j3})](\mathbf{z}) \geq 0.1 + 0.1 + 0.1 = 0.3$. WLOG, let's assume $l_{j1} < 0.4$.

Let the corresponding literal L_{j1} come from variable X_e . Either $L_{j1} = X_e$ or $L_{j1} = \neg X_e$. In the former case, because $l_{j1}(\mathbf{z}) < 0.4$, then $l_{ij}(\mathbf{z}) = l_{ij}(z_e) = z_e < 0.4$. According to our assignment rule, $x_e = 0$ and X_e is evaluated false. In the latter case, $l_{i1}(\mathbf{z}) = l_{i1}(z_j) = 1 - z_j < 0.4$, so $z_j > 0.6$. According to the assignment rule, $x_j = 1$, and so X_j is evaluated to true, and $\neg X_j$ is evaluated to false. In either case, C_j is evaluated to false, because C_j is a conjunction of literals.

We have shown that the assignment \mathbf{x} unsatisfies C_i , and so the 3DNF Boolean instance ϕ is not a tautology. \square