EPTCS 421

Proceedings of the

19th International Workshop on

Logical and Semantic Frameworks, with Applications

Goiânia, Brazil, 18th-20th September 2024

Edited by: Cynthia Kop and Helida Salles Santos

Published: 6th June 2025
DOI: 10.4204/EPTCS.421
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Regional, Lattice and Logical Representations of Neural Networks

Sandro Preto
Center for Mathematics, Computing and Cognition
Federal University of ABC, Brazil
Institute of Mathematics and Statistics
University of São Paulo, Brazil
sandro.preto@ufabc.edu.br

Marcelo Finger
Institute of Mathematics and Statistics
University of São Paulo, Brazil
mfinger@ime.usp.br

A possible path to the interpretability of neural networks is to (approximately) represent them in the regional format of piecewise linear functions, where regions of inputs are associated to linear functions computing the network outputs. We present an algorithm for the translation of feedforward neural networks with ReLU activation functions in hidden layers and truncated identity activation functions in the output layer. We also empirically investigate the complexity of regional representations outputted by our method for neural networks with varying sizes. Lattice and logical representations of neural networks are straightforward from regional representations as long as they satisfy a specific property. So we empirically investigate to what extent the translations by our algorithm satisfy such property.

1 Introduction

Neural networks are computational models that aim to generalize patterns found in datasets from which they are determined by means of a learning algorithm [8]. Despite the undeniable advancement in the state of the art of intelligent systems promoted by neural networks, their lack of interpretability is subject to criticism. Neural networks suffer from the *black box problem* due to the lack of justification for their results and the impossibility to directly inspect their learned information [3, 5].

As several architectures of neural networks realize piecewise linear functions or may be approximated by them, a path towards interpretability is through *regional format* representations of such neural networks and functions by explicit sets of pairs $\langle p, \Omega \rangle$ of a linear piece p and a region Ω such that, for a vector of input values $\mathbf{x} \in \Omega$, the output is given by $p(\mathbf{x})$. An algorithm for establishing regional representations from feedforward neural networks with rectified linear units as activation functions is proposed in [15].

The main goal of this work is to introduce an algorithm for computing regional format representations of *ReLU–TId neural networks*, which are feedforward neural networks with rectified linear units as activation functions in hidden layers and truncated identity as activation functions in the output layer. Such algorithm outputs representations in the *pre-closed regional format*, where regions are polyhedra. Rather than just adapting the iterative method in [15], we present a novel recursive approach that allows a correctness proof by a straightforward induction argument.

An important feature of neural networks is that they are compact representations of functions. Then, although regional representations might provide interpretability of neural networks, they also might be exponential in the size of their traditional representation as graphs. In Section 4, we empirically measure the complexity of regional representations determined by our method for randomly generated ReLU–TId neural networks with varying numbers of neurons and layers and varying layer sizes.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 64–79, doi:10.4204/EPTCS.421.4

Lattice representation is another possibility for representing neural networks and is achieved by combining maximum and minimum operations over linear pieces. Such representations further enable the codification of ReLU–TId neural networks in logical systems as Łukasiewicz infinitely-valued logic ($Ł_\infty$) and its extensions [4, 7, 11, 12], leading to yet another path to interpretability. Lattice and logical representations find applications in the formal verification of neural networks in attempts to circumvent the black box problem and allow their use in critical tasks; for instance, in aircraft collision avoidance alerts and autonomous vehicles. There are methods for formal verification using the lattice representation of neural networks [1] and methods that codify properties of neural networks in the language of $Ł_\infty$ departing from their logical representation [13, 14].

Lattice and logical representations may be built in polynomial time from ReLU–TId neural networks given in the pre-closed regional format as long as such encodings satisfy the so-called *lattice property* (Section 2) [12]. In this case, the regional representations are said to be in the *closed regional format*. This work also aims at empirically experimenting how far from satisfying lattice property are randomly generated neural networks.

The rest of this work is organized as follows. Section 2 introduces neural networks and their graph, regional, lattice and logical representations. Section 3 presents an algorithm for translating ReLU–TId neural networks into the pre-closed regional format. Section 4 presents the results of experiments where we measure the complexity of representations in pre-closed regional format and their degree of satisfiability of lattice property.

2 Preliminaries: Some Neural Networks and Their Representations

Traditionally, a *feedforward neural network* N is given (and represented) by a graph whose nodes are partitioned into an ordered family of ordered sets $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, where each L_i is a *layer*. All nodes in layer L_i, for $i \in \{0, \ldots, \Lambda - 1\}$, are linked by an edge to all nodes in layer L_{i+1} establishing a computational circuit such that all output values of nodes in L_i are input values to each node in L_{i+1}. There is a linear function $f^i_j : \mathbb{R}^{|L_{i-1}|} \to \mathbb{R}$ associated to each node n^i_j in layer L_i, for $j \in \{1, \ldots, |L_i|\}$ and $i \in \{1, \ldots, \Lambda\}$. For a tuple of input values $\mathbf{x} = \langle x_1, \ldots, x_{|L_{i-1}|}\rangle \in \mathbb{R}^{|L_{i-1}|}$ to node n^i_j, it has as output the value $n^i_j(\mathbf{x}) = \rho_i \circ f^i_j(\mathbf{x})$, where $\rho_i : \mathbb{R} \to \mathbb{R}$ is an *activation function*. Thus, for $\mathbf{x} = \langle x_1, \ldots, x_{|L_{i-1}|}\rangle$ as input to layer L_i, it has as outputs the values in the tuple

$$L_i(\mathbf{x}) = \langle \rho_i \circ f^i_1(\mathbf{x}), \ldots, \rho_i \circ f^i_{|L_i|}(\mathbf{x})\rangle.$$

Input values to N in a tuple $\mathbf{x} = \langle x_1, \ldots, x_{|L_0|}\rangle \in \mathbb{R}^{|L_0|}$ are neatly associated to the nodes $n^0_1, \ldots, n^0_{|L_0|} \in L_0$, called *input nodes*; thus, from such inputs, N produces the output values in the $|L_\Lambda|$-tuple

$$N(\mathbf{x}) = \langle N(\mathbf{x})_1, \ldots, N(\mathbf{x})_{|L_\Lambda|}\rangle = L_\Lambda \circ \cdots \circ L_1(\mathbf{x}).$$

Nodes in L_Λ are called *output nodes*. We say that each output node n^Λ_j in a neural network N computes a function for which the value $N(\mathbf{x})_j$ is given in function of the input values $\mathbf{x} \in \mathbb{R}^{|L_0|}$.

We might restrict the input values of a neural network to some set $R \subseteq \mathbb{R}$. In this work, we focus on *ReLU–TId neural networks*: they accept input values from $[0,1]$ and have as activation functions the *rectified linear unit* $\rho_i = \mathrm{ReLU} : \mathbb{R} \to \mathbb{R}$, given by $\mathrm{ReLU}(x) = \max(0, x)$, for $i \in \{1, \ldots, \Lambda - 1\}$, and the *truncated identity function* $\rho_\Lambda = \mathrm{TId} : \mathbb{R} \to \mathbb{R}$, given by $\mathrm{TId}(x) = \max(0, \min(1, x))$. Such activation

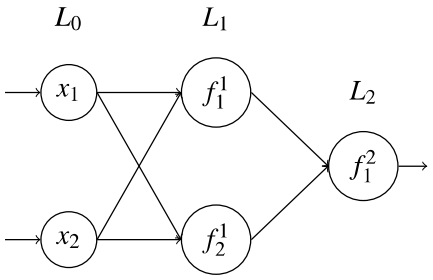


Figure 1: Graph of the ReLU–TId neural network E

functions may be given by piecewise linear definitions as follows:

$$\mathrm{ReLU}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \qquad \mathrm{TId}(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \leq x \leq 1 \\ 1, & x > 1 \end{cases} \tag{1}$$

Example 1 Let E be a ReLU–TId neural network with $\mathscr{L}_E = \{L_0, L_1, L_2\}$, where $L_0 = \{n_1^0, n_2^0\}$, $L_1 = \{n_1^1, n_2^1\}$ and $L_2 = \{n_1^2\}$. The graph of E depicted in Figure 1 highlights the input values x_1 and x_2 in layer L_0 and the functions $f_1^1, f_2^1, f_1^2 : \mathbb{R}^2 \to \mathbb{R}$ in layers L_1 and L_2, which are given by:

- $f_1^1(x_1, x_2) = \frac{4}{3}x_1 - x_2$;
- $f_2^1(x_1, x_2) = x_1 - x_2 + \frac{1}{2}$;
- $f_1^2(x_1, x_2) = x_1 + x_2 + \frac{1}{2}$.

For a tuple of inputs $\mathbf{e} = \langle \frac{1}{8}, \frac{1}{2} \rangle$ to E, we have $L_1(\mathbf{e}) = \langle \mathrm{ReLU}(-\frac{1}{3}), \mathrm{ReLU}(\frac{1}{8}) \rangle = \langle 0, \frac{1}{8} \rangle$ and, thus, $E(\mathbf{e}) = L_2(L_1(\mathbf{e})) = \mathrm{TId}(\frac{5}{8}) = \frac{5}{8}$. □

Before introducing another type of neural network, let us define a *rational McNaughton function* $f : [0,1]^n \to [0,1]$, which is a function that satisfies the following conditions:

- f is continuous with respect to the usual topology of $[0,1]$ as an interval of the real number line;
- There are linear polynomials $p_1, \ldots, p_m$ over $[0,1]^n$ with rational coefficients such that, for each point $\mathbf{x} \in [0,1]^n$, there is an index $i \in \{1, \ldots, m\}$ with $f(\mathbf{x}) = p_i(\mathbf{x})$. Polynomials $p_1, \ldots, p_m$ are the *linear pieces* of f.

A neural network whose $\nu = |L_\Lambda|$ output nodes exactly compute rational McNaughton functions in function of its input nodes is called a *ν-rational McNaughton neural network (ν-RMcN^3)*; a 1-RMcN^3 is also called *rational McNaughton neural network (RMcN^3)*.

A possibility to represent rational McNaughton functions (consequently, ν-rational McNaughton neural networks) is through the *regional formats* discussed in the following. Let Ω° denote the topological interior of $\Omega \subseteq [0,1]^n$ and say that, given functions $f, g : [0,1]^n \to [0,1]$, f is *above* g over the set Ω if $f(\mathbf{x}) \geq g(\mathbf{x})$, for all $\mathbf{x} \in \Omega$. A given rational McNaughton function $f : [0,1]^n \to [0,1]$ is said to be encoded in the *closed regional format* if it is given by m (not necessarily distinct) linear pieces

$$p_i(\mathbf{x}) = \gamma_{i0} + \gamma_{i1}x_1 + \cdots + \gamma_{in}x_n, \tag{2}$$

where $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in [0,1]^n$, $\gamma_{ij} \in \mathbb{Q}$ and $i \in \{1, \ldots, m\}$, such that each p_i is identical to f over a polyhedron $\Omega_i \subseteq [0,1]^n$, called *region*. These regions are determined by the finite intersection of half-spaces given by linear inequalities[1] as

$$\Omega_i = \left\{ \mathbf{x} \in [0,1]^n \;\middle|\; \omega_{j0} + \omega_{j1} x_1 + \cdots + \omega_{jn} x_n \geq 0,\ j \in \{1, \ldots, \lambda_{\Omega_i}\} \right\} \tag{3}$$

and such setting of linear pieces and regions satisfy the following properties:

- $\bigcup_{i=1}^{m} \Omega_i = [0,1]^n$;
- $\Omega_{i'}^{\circ} \cap \Omega_{i''}^{\circ} = \varnothing$, for $i' \neq i''$; and
- The *lattice property*: for $i \neq j$, there is k such that linear piece p_i is above linear piece p_k over region Ω_i and linear piece p_k is above linear piece p_j over region Ω_j.

Such an encoding is called *closed* because regions Ω_i are closed sets in the topological sense. As regions are given by such polyhedra described by (3), there is a polynomial procedure to establish whether a linear piece p is above q over region Ω: find the minimum value m of $p - q$ over Ω, which is a linear program and may be solved in polynomial time [2]; then, if $m \geq 0$, p is above q over Ω, otherwise, it is not.

The lattice property yields the possibility to represent rational McNaughton functions and ν-RMcN[3]s by *lattice representations*—i.e., based on operations of maximum and minimum over functions—as follows. First, let $f_{\Omega_j} : [0,1]^n \to [0,1]$ be the function given by

$$f_{\Omega_j}(\mathbf{x}) = \min \left\{ p_k(\mathbf{x}) \;\middle|\; p_k \text{ is above } p_j \text{ over } \Omega_j \right\}.$$

Note that $f_{\Omega_j}(\mathbf{x}) \leq f(\mathbf{x})$, for all $\mathbf{x} \in [0,1]^n$, which is obvious for $\mathbf{x} \in \Omega_j$ and follows from the lattice property for $\mathbf{x} \in \Omega_i$ where $i \neq j$. In this way, we have that

$$f(\mathbf{x}) = \max \left\{ f_{\Omega_j}(\mathbf{x}) \;\middle|\; j \in \{1, \ldots, m\} \right\}.$$

In [13], we find an example of the one-variable rational McNaughton function $f : [0,1] \to [0,1]$, whose graph is depicted in Figure 2. Function f has a lattice representation given by

$$f(x) = \max \left\{ f_{\Omega_1}(x),\ f_{\Omega_2}(x),\ f_{\Omega_3}(x),\ f_{\Omega_4}(x) \right\},$$

where $f_{\Omega_1}(x) = \min\{p_1(x), p_3(x)\}$, $f_{\Omega_2}(x) = f_{\Omega_3}(x) = \min\{p_2(x), p_3(x)\}$ and $f_{\Omega_4}(x) = \min\{p_2(x), p_4(x)\}$. Note that the lattice encoding just introduced may be employed in representing piecewise linear functions in general, not only rational McNaughton functions.

When a rational McNaughton function is given in an encoding that almost completely agrees with the closed regional format, with the sole exception that there is no guarantee that such encoding satisfies the lattice property (although it may still satisfy), we say that it is in the *pre-closed regional format*.

Unfortunately, lack of lattice property might entail the failure of lattice representation as in the following example taken from [12]. The rational McNaughton function f_E with graph in Figure 3a may have an encoding based on regions in Figure 3b; a linear piece p_i is associated to each region Ω_i. The dotted line in Figure 3b is the projection over $[0,1]^2$ of where p_3 intercepts p_5; note that such line passes

[1] We occasionally abuse notation by using the same symbol to refer both to a set of inequalities and to the polyhedron it determines.

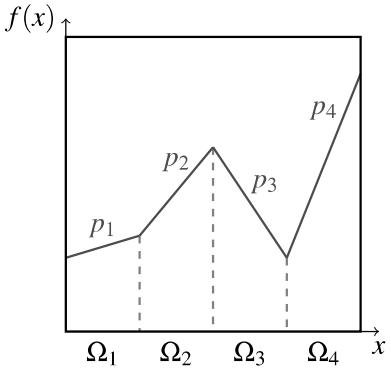
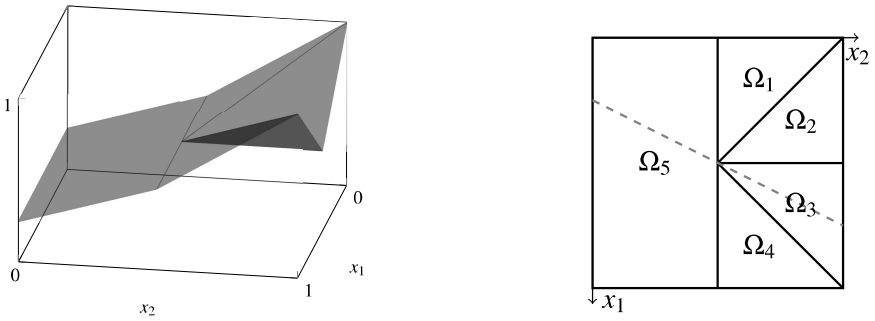


Figure 2: One-variable piecewise linear function



(a) Graph (b) Region configuration

Figure 3: Function encoded in the pre-closed regional format

through the interior of both Ω_3 and Ω_5. There is no linear piece p_k such that p_3 is above p_k over Ω_3 and p_k is above p_5 over Ω_5. So an encoding for f_E based on regions Ω_1–Ω_5 may be at most encoded in pre-closed regional format.

Now, there is $\mathbf{x}_0 \in \Omega_3^\circ$ such that $p_5(\mathbf{x}_0) > p_3(\mathbf{x}_0)$. Therefore, $f_{\Omega_5}(\mathbf{x}_0) = p_5(\mathbf{x}_0) > p_3(\mathbf{x}_0) = \min\{p_1(\mathbf{x}_0), p_3(\mathbf{x}_0)\} = f_{\Omega_3}(\mathbf{x}_0)$, yielding that $\max\{f_{\Omega_j}(\mathbf{x}_0)\} > f_{\Omega_3}(\mathbf{x}_0)$, which eliminates the possibility of lattice representation. Such an issue may be circumvented by splitting region Ω_5, according to the dotted line in Figure 3b, in regions $\Omega_5' = \Omega_5 \cap \{p_5 - p_3 \geq 0\}$ and $\Omega_5'' = \Omega_5 \cap \{p_5 - p_3 \leq 0\}$. In general, repeatedly splitting a region according to projections of linear pieces intersections eventually achieves closed regional format [12, Theorem 7].

We may also represent rational McNaughton functions and ν-RMcN^{3}s in logical systems. For that, let us introduce the Łukasiewicz infinitely-valued logic (Ł$_\infty$). The basic language $\mathscr{L}$ of Ł$_\infty$ comprehends formulas freely generated from a countable set of propositional variables $\mathbb{P}$, a disjunction operator $\oplus$ and a negation operator $\neg$. A *valuation* is a function $v : \mathscr{L} \to [0,1]$, such that, for $\varphi, \psi \in \mathscr{L}$:

$$v(\varphi \oplus \psi) = \min(1, v(\varphi) + v(\psi)); \tag{4}$$

$$v(\neg\varphi) = 1 - v(\varphi). \tag{5}$$

From disjunction and negation we derive the following operators:

$$\begin{array}{rll}
\text{Conjunction:} & \varphi \odot \psi =_{\text{def}} \neg(\neg\varphi \oplus \neg\psi) & v(\varphi \odot \psi) = \max(0, v(\varphi) + v(\psi) - 1) \\
\text{Implication:} & \varphi \to \psi =_{\text{def}} \neg\varphi \oplus \psi & v(\varphi \to \psi) = \min(1, 1 - v(\varphi) + v(\psi)) \\
\text{Maximum:} & \varphi \vee \psi =_{\text{def}} \neg(\neg\varphi \oplus \psi) \oplus \psi & v(\varphi \vee \psi) = \max(v(\varphi), v(\psi)) \\
\text{Minimum:} & \varphi \wedge \psi =_{\text{def}} \neg(\neg\varphi \vee \neg\psi) & v(\varphi \wedge \psi) = \min(v(\varphi), v(\psi)) \\
\text{Bi-implication:} & \varphi \leftrightarrow \psi =_{\text{def}} (\varphi \to \psi) \wedge (\psi \to \varphi) & v(\varphi \leftrightarrow \psi) = 1 - |v(\varphi) - v(\psi)|
\end{array}$$

Note that, as lattice operations are expressed in Ł$_\infty$ by the minimum and maximum operators, piecewise linear functions might have a lattice representation in Ł$_\infty$ as far as their linear pieces are representable in this system. Indeed, the formulas of Ł$_\infty$ represent all the *McNaughton functions*, which are rational McNaughton functions constrained to allow only integer coefficients in their linear pieces [9, 10].

Unfortunately, Ł$_\infty$ cannot express rational McNaughton functions. For that, one possible path is to extend the language of Ł$_\infty$, which is done, for instance, by [7]. Another possibility is to implicitly represent such functions in plain Ł$_\infty$ using the technique of *representation modulo satisfiability*, which we introduce in the following [6, 11, 12].

Let us denote the Ł$_\infty$*-semantics*, that is the set of all valuations, by **Val**. Let us also denote by $\mathbf{Val}_\Phi$ the set of valuations $v \in \mathbf{Val}$ that satisfy a set of formulas Φ; we call such a restricted set of valuations a *semantics modulo satisfiability*. Given a rational McNaughton function $f : [0,1]^n \to [0,1]$, a formula φ_f and a set of formulas Φ_f, we say that φ_f *represents* f *modulo* Φ_f*-satisfiable* or that the pair $\langle \varphi_f, \Phi_f \rangle$ *represents* f *(in the system Ł$_\infty$-MODSAT)* if, for distinguished propositional variables $X_1, \ldots, X_n \in \mathbb{P}$:

- For all $\langle x_1, \ldots, x_n \rangle \in [0,1]^n$, there exists some valuation $v \in \mathbf{Val}_{\Phi_f}$, such that $v(X_i) = x_i$, for $i = 1, \ldots, n$;
- For all valuations $v, v' \in \mathbf{Val}_{\Phi_f}$ such that $v(X_i) = v'(X_i)$, for $i = 1, \ldots, n$, we have $v(\varphi_f) = v'(\varphi_f)$; and
- $f(v(X_1), \ldots, v(X_n)) = v(\varphi_f)$, for all $v \in \mathbf{Val}_{\Phi_f}$.

As an example, any constant function that takes value $\frac{1}{b}$, with $b \in \mathbb{N}^*$, may be represented by the pair

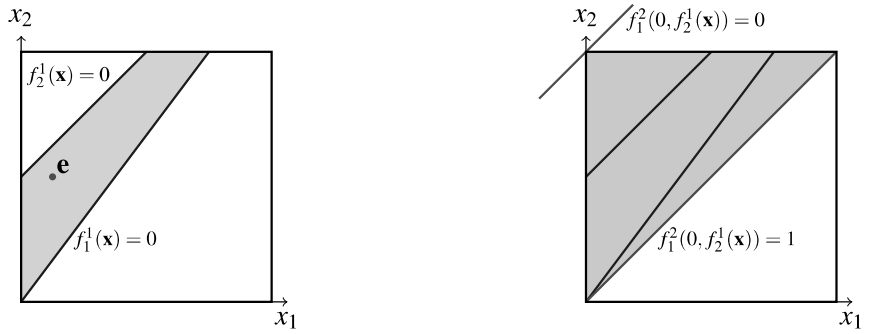
$$\langle \varphi, \Phi \rangle = \left\langle \; Z_{\frac{1}{b}}, \; \left\{ Z_{\frac{1}{b}} \leftrightarrow \neg(b-1) Z_{\frac{1}{b}} \right\} \; \right\rangle, \tag{6}$$

where formula φ is only the propositional variable $Z_{1/b}$ and set Φ is a singleton comprehending formula $Z_{1/b} \leftrightarrow \neg(b-1)Z_{1/b}$, which we denote by $\varphi_{1/b}$. In fact, for any valuation $v \in \mathbf{Val}_{\varphi_{1/b}} \neq \varnothing$, we have that $v(Z_{1/b}) = \frac{1}{b}$. Also, functions that take constant value $\frac{a}{b}$, with $a \in \mathbb{N}$, may be represented by the pair $\langle a\varphi, \Phi \rangle$.

Any rational McNaughton function may be represented in Ł$_\infty$-MODSAT. Moreover, there is a polynomial algorithm for the translation from a rational McNaughton function in closed regional format to its representation in such system [11, 12].

3 Neural Networks into Pre-Closed Regional Format

Given a ReLU–TId neural network N for which $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, we provide an algorithm to translate it into a tuple $\Xi_N = \langle \Xi_1, \ldots, \Xi_{|L_\Lambda|} \rangle$, where each Ξ_k, $k \in \{1, \ldots, |L_\Lambda|\}$, is the codification for a rational McNaughton function in pre-closed regional format, which we will show to be the function computed



(a) $f_1^1(\mathbf{x}) \leq 0$, $f_2^1(\mathbf{x}) \geq 0$ (b) $f_1^2(0, f_2^1(\mathbf{x})) \geq 0$, $f_1^2(0, f_2^1(\mathbf{x})) \leq 1$

Figure 4: Determining a region for neural network E

by N through the path to its k-th output node. Each Ξ_k is a set of pairs $\langle p, \Omega \rangle$, where p is a linear piece and Ω is its associated region.

Let us begin by analyzing the computation that takes place in each node of N. Given a tuple of input values $\mathbf{v} \in \mathbb{R}^{|L_{i-1}|}$ to node n_j^i of N, where $i \in \{1, \ldots, \Lambda - 1\}$ and so $\rho_i = \text{ReLU}$, the computation proceeds in two steps: first, the linear function $f_j^i(\mathbf{x})$ is evaluated for $\mathbf{x} = \mathbf{v}$; second, the activation function $\text{ReLU}(x)$ is evaluated for $x = f_j^i(\mathbf{v})$. In the second step, one from the two possible values highlighted in the piecewise definition of ReLU in (1) is chosen as the output of n_j^i. Such choice depends on the position of point $\mathbf{v} \in \mathbb{R}^{|L_{i-1}|}$ in relation to the hyperplane given by equation $f_j^i(\mathbf{x}) = 0$, since it may be a point lying either in the half-space given by $f_j^i(\mathbf{x}) \leq 0$ or in the half-space given by $f_j^i(\mathbf{x}) \geq 0$. For instance, for $\mathbf{e} = \langle \frac{1}{8}, \frac{1}{2} \rangle$ as input to the node n_1^1 of E in Example 1, the fact that $f_1^1(\mathbf{e}) \leq 0$ indicates that $\mathbf{e}$ lies in one of the half-spaces determined by $f_1^1(\mathbf{x}) = 0$ where, according to $\rho_1 = \text{ReLU}$, n_1^1 outputs 0. On the other hand, if $\mathbf{e}$ is given as input to the node n_2^1, as $f_2^1(\mathbf{e}) \geq 0$, $\mathbf{e}$ lies in the half-space determined by $f_2^1(\mathbf{x}) = 0$ where, according to $\rho_1 = \text{ReLU}$, n_2^1 outputs $f_2^1(\mathbf{e}) = \frac{1}{8}$. Figure 4a depicts the position of $\mathbf{e}$ in relation to these hyperplanes.

Similarly, in case $i = \Lambda$, we have that $\rho_i = \text{TId}$ and, then, one from three possible values is chosen for $\text{TId}(f_j^i(\mathbf{v}))$ depending on the position of $\mathbf{v} \in \mathbb{R}^{|L_{\Lambda-1}|}$ in relation to the hyperplanes $f_j^\Lambda(\mathbf{x}) = 0$ and $f_j^\Lambda(\mathbf{x}) = 1$. It may be a point lying either in the half-space given by $f_j^\Lambda(\mathbf{x}) \leq 0$, or in the half-space given by $f_j^\Lambda(\mathbf{x}) \geq 1$ or in the intersection of half-spaces $f_j^\Lambda(\mathbf{x}) \geq 0$ and $f_j^\Lambda(\mathbf{x}) \leq 1$. Again, for $\mathbf{e} = \langle \frac{1}{8}, \frac{1}{2} \rangle$ as input to the neural network E in Example 1, we have $\mathbf{e}_1 = L_1(\mathbf{e}) = \langle 0, \frac{1}{8} \rangle$. Since $f_1^2(\mathbf{e}_1) \geq 0$ and $f_1^2(\mathbf{e}_1) \leq 1$, $\mathbf{e}_1$ lies in a position relative to $f_1^2(\mathbf{x}) = 0$ and $f_1^2(\mathbf{x}) = 1$ where, according to $\rho_2 = \text{TId}$, n_1^2 outputs $f_1^2(\mathbf{e}_1) = \frac{5}{8}$.

Now, still considering Example 1, let $\mathbf{x} \in [0,1]^{|L_0|}$ be any point that, as $\mathbf{e} = \langle \frac{1}{8}, \frac{1}{2} \rangle$, satisfies both inequalities

$$f_1^1(\mathbf{x}) \leq 0 \quad \text{and} \quad f_2^1(\mathbf{x}) \geq 0. \tag{7}$$

Then, to $\mathbf{x}_1 = L_1(\mathbf{x})$ satisfy inequalities $f_1^2(\mathbf{x}_1) \geq 0$ and $f_1^2(\mathbf{x}_1) \leq 1$ is equivalent to $\mathbf{x}$ satisfy inequalities

$$f_1^2(0, f_2^1(\mathbf{x})) \geq 0 \quad \text{and} \quad f_1^2(0, f_2^1(\mathbf{x})) \leq 1. \tag{8}$$

As the former inequalities (7), the latter inequalities (8) are also linear over tuples from $[0,1]^{|L_0|}$ of input values to the neural network E. Moreover, for $\mathbf{x} \in [0,1]^{|L_0|}$ satisfying inequalities (7) and (8), we have

that $E(\mathbf{x}) = f_1^2(0, f_2^1(\mathbf{x}))$. In this way, we have just devised a region and its associated linear piece for the rational McNaughton function computed by neural network E.

Generalizing the observations above, the idea behind the base algorithm for building Ξ_k, for $k \in \{1,\dots,|L_\Lambda|\}$, is to compute each pair $\langle p, \Omega\rangle \in \Xi_k$ beginning by: associating a symbol between $\leq$ and $\geq$ to each node n_j^i, for $i < \Lambda$, alluding to one of the two possible positions of an input to n_j^i in relation to the hyperplane $f_j^i(\mathbf{x}) = 0$—i.e., lying in the half-space $f_j^i(\mathbf{x}) \leq 0$ or in the half-space $f_j^i(\mathbf{x}) \geq 0$—; and associating a symbol among $\leq$, $\geq$ and $\lesseqgtr$ to the node n_k^Λ alluding to one of the three possible positions of an input to n_k^Λ in relation to the hyperplanes $f_k^\Lambda(\mathbf{x}) = 0$ and $f_k^\Lambda(\mathbf{x}) = 1$—i.e., lying in the half-space $f_k^\Lambda(\mathbf{x}) \leq 0$, or in the half-space $f_k^\Lambda(\mathbf{x}) \geq 1$ or in the intersection of the half-spaces $f_k^\Lambda(\mathbf{x}) \geq 0$ and $f_k^\Lambda(\mathbf{x}) \leq 1$. These associations of symbols to all the nodes in layers $L_1,\dots,L_\Lambda$ determine a *configuration of symbols*. Then, the algorithm proceeds by defining $\Omega \subseteq [0,1]^{|L_0|}$ as an intersection of half-spaces based on such configuration of symbols and establishing a linear expression for p such that $N(\mathbf{x})_k = p(\mathbf{x})$, for $\mathbf{x} \in \Omega$.

Example 2 For the neural network E in Example 1, in a configuration of symbols where we associate $\leq$ to n_1^1 and $\geq$ to n_2^1, the consequent region Ω should comprehend the inequalities $\frac{4}{3}x_1 - x_2 \leq 0$ and $x_1 - x_2 + \frac{1}{2} \geq 0$ (shaded area in Figure 4a). For an input $\mathbf{x} \in [0,1]^2$ that satisfies these inequalities, we have the outputs $n_1^1(\mathbf{x}) = 0$ and $n_2^1(\mathbf{x}) = f_2^1(\mathbf{x})$ which, composed with f_1^Λ, gives us the expression $x_1 - x_2 + 1$. In this way, in case we complete the configuration of symbols by associating $\leq$ to n_1^Λ, Ω should comprehend the inequality $x_1 - x_2 + 1 \leq 0$ and p should be given by $p(x_1, x_2) = 0$. In case we associate $\geq$ to n_1^Λ, Ω should comprehend the inequality $x_1 - x_2 + 1 \geq 1$ and p should be given by $p(x_1, x_2) = 1$. In the last case, if we associate $\lesseqgtr$ to n_1^Λ, Ω should comprehend both inequalities $x_1 - x_2 + 1 \leq 1$ and $x_1 - x_2 + 1 \geq 0$ (shaded area in Figure 4b) and p should be given by $p(x_1, x_2) = x_1 - x_2 + 1$. Note that the last case is the only one where region Ω would be non-empty. □

In order to build the entire representative tuple $\Xi_N = \langle \Xi_1, \dots, \Xi_{|L_\Lambda|}\rangle$, the algorithm needs to compute all the pairs $\langle p, \Omega\rangle$, each one associated to a different configuration of symbols, for all possible configurations of symbols. Thus, the entire computation of Ξ_N ends up with $2^{|L_1|} \times \cdots \times 2^{|L_{\Lambda-1}|} \times 3|L_\Lambda|$ pairs $\langle p, \Omega\rangle$. Later, we introduce methods meant to be combined with the base algorithm that might circumvent such high complexity.

For establishing the base translation algorithm, we first fix some notation. Let κ_0^n and κ_1^n be the constant linear functions with domain $[0,1]^n$ and ranges equal to $\{0\}$ and $\{1\}$, respectively; and let $\chi_n : \{\leq, \geq\} \to \{\kappa_0^n, \kappa_1^n\}$ be the functions given by $\chi_n(\leq) = \kappa_0^n$ and $\chi_n(\geq) = \kappa_1^n$. Also, let $\pi_n^m : [0,1]^m \to \mathbb{R}$ be the projection functions given by $\pi_n^m(x_1,\dots,x_m) = x_n$, for $m \in \mathbb{N}$ and $1 \leq n \leq m$. The base translation algorithm is split into Algorithms 1 and 2.

Algorithm 1 treats the tuple $\Xi_N = \langle \Xi_1, \dots, \Xi_{|L_\Lambda|}\rangle$ as a variable to be updated as it runs; thus, it first sets each of the $\Xi_1, \dots, \Xi_{|L_\Lambda|}$ to the empty set as their initial values (lines 1 and 2). Then, it defines $\Omega_{[0,1]}$ as a set of inequalities common to all regions (line 3) and π as a tuple of projection functions (line 4), which will be suitable for compositions with functions f_j^1 related to the first layer L_1. It proceeds by calling the recursive routine NN2PWL-R($\Xi_N \parallel L_1, \Omega, \pi$) (line 5), where Ξ_N is an argument **passed by reference**, which means that whenever NN2PWL-R modifies the value of Ξ_N, it will also be modified in the scope of the calling function. Finally, Algorithm 1 returns Ξ_N with its final value (line 6).

Algorithm 2 describes the recursive routine NN2PWL-R that has as inputs a tuple $\Xi_N = \langle \Xi_1, \dots, \Xi_{|L_\Lambda|}\rangle$ to be updated, a ReLU–TId neural network N with a distinguished layer $L_i \in \mathscr{L}_N$, a set of inequalities Ω and a tuple of functions $f = \langle f_1, \dots, f_{|L_{i-1}|}\rangle$. If $L_i \neq L_\Lambda$, for all possible association of symbols $\leq$ and $\geq$ to the nodes $n_1^i, \dots, n_{|L_i|}^i$ summarized in the tuple of symbols $\bowtie = \langle \bowtie_1, \dots, \bowtie_{|L_i|}\rangle$, NN2PWL-R($\Xi_N \parallel L_i, \Omega, f$) proceeds by:

- Computing $\Omega^i_{\bowtie}$ as Ω extended by the half-spaces $f^i_j \circ f(\mathbf{x}) \bowtie_j 0$, for $j \in \{1,\ldots,|L_i|\}$, where $f = \langle f_1,\ldots,f_{|L_{i-1}|}\rangle$ is a tuple of linear functions such that $f(\mathbf{x}) = \langle f_1(\mathbf{x}),\ldots,f_{|L_{i-1}|}(\mathbf{x})\rangle = L_{i-1}\circ\cdots\circ L_1(\mathbf{x})$, for $\mathbf{x}\in\Omega$ (line 3);
- Computing the tuple of linear functions $f^i_{\bowtie}$ that is identical to the output of $L_i\circ\cdots\circ L_1$ for inputs $\mathbf{x}\in\Omega^i_{\bowtie}$, with assistance of functions $\chi_{|L_0|}$ (line 4);
- And calling itself again by NN2PWL-R$(\Xi_N \parallel L_{i+1}, \Omega^i_{\bowtie}, f^i_{\bowtie})$ (line 5).

If $L_i = L_\Lambda$, for each of the output nodes $n^\Lambda_1,\ldots,n^\Lambda_{|L_\Lambda|}$, NN2PWL-R$(\Xi_N \parallel L_i, \Omega, f)$ proceeds by:

- Computing $\Omega_{\leq}$, $\Omega_{\geq}$ and $\Omega_{\leqslant}$ as Ω extended, respectively, by the half-space $f^\Lambda_k\circ f(\mathbf{x}) \leq 0$, the half-space $f^\Lambda_k\circ f(\mathbf{x}) \geq 1$ and the pair of half-spaces $f^\Lambda_k\circ f(\mathbf{x}) \geq 0$ and $f^\Lambda_k\circ f(\mathbf{x}) \leq 1$, where f is a tuple of linear functions such that $f(\mathbf{x}) = L_{\Lambda-1}\circ\cdots\circ L_1(\mathbf{x})$, for $\mathbf{x}\in\Omega$ (lines 9, 11 and 13);
- And rewriting Ξ_k by adding the pairs $\langle \kappa_0^{|L_0|}, \Omega_{\leq}\rangle$, $\langle f^\Lambda_k\circ f, \Omega_{\leqslant}\rangle$ and $\langle \kappa_1^{|L_0|}, \Omega_{\geq}\rangle$ to it (lines 10, 12 and 14).

Let Ω be any region appearing in the output of the base algorithm; it is built in $\Lambda+1$ steps in a way that, in each step, new inequalities are added to a polyhedron (identified with a set of inequalities) until it becomes Ω. The first step adds the inequalities that determine $[0,1]^{|L_0|}$ (Algorithm 1, line 3). The next $\Lambda-1$ steps, where the produced polyhedra are named $\Omega^i_{\bowtie}$ (Algorithm 2, line 3), are associated to layers $L_1,\ldots,L_{\Lambda-1}$ of N. The final step, where the produced region Ω is named either as $\Omega_{\leq}$, $\Omega_{\geq}$ or $\Omega_{\leqslant}$ (Algorithm 2, line 9, 11 or 13), is associated to layer L_Λ.

Algorithm 1 NN2PWL: puts neural networks in the closed regional format

Input: A ReLU–TId neural network N for which $\mathscr{L}_N = \{L_0,\ldots,L_\Lambda\}$.
Output: A set Ξ_N representing rational McNaughton functions computed by the output nodes of N.

1: $\Xi_1 := \varnothing, \ldots, \Xi_{|L_\Lambda|} := \varnothing$;
2: $\Xi_N := \langle \Xi_1,\ldots,\Xi_{|L_\Lambda|}\rangle$;
3: $\Omega_{[0,1]} := \{x_1 \geq 0, x_1 \leq 1, \ldots, x_{|L_0|} \geq 0, x_{|L_0|} \leq 1\}$;
4: $\pi := \langle \pi_1^{|L_0|},\ldots,\pi_{|L_0|}^{|L_0|}\rangle$;
5: NN2PWL-R$(\Xi_N \parallel L_1, \Omega_{[0,1]}, \pi)$;
6: **return** Ξ_N;

Lemma 1 *Let a ReLU–TId neural network N, for which $\mathscr{L}_N = \{L_0,\ldots,L_\Lambda\}$, be given as input to Algorithm 1. Then, Algorithm 1 terminates and outputs a tuple $\Xi_N = \langle \Xi_1,\ldots,\Xi_{|L_\Lambda|}\rangle$ where, for $k\in\{1,\ldots,|L_\Lambda|\}$, we have:*

- $\bigcup_{\Omega,\, for\ \langle p,\Omega\rangle\in\Xi_k} \Omega = [0,1]^{|L_0|}$;
- $\Omega'^{\circ} \cap \Omega''^{\circ} = \varnothing$, *for distinct* $\langle p',\Omega'\rangle, \langle p'',\Omega''\rangle \in \Xi_k$. □

PROOF Algorithm 1 always terminates since all of its loops, which are originated from Algorithm 2 calls, range over some finite set and all recursive calls in Algorithm 2 increments the index of the input layer L_i, which will eventually reach the last layer L_Λ and break the recursion by falsifying the conditional statement in line 1 of Algorithm 2. Let Ξ_k be an entry in Ξ_N; all inequalities added to regions in Ξ_k determine half-spaces in $[0,1]^{|L_0|}$. Indeed, this is the case in the first step of the construction of regions (Algorithm 1, line 3). This is also the case for the remaining half-spaces, whose corresponding inequalities are recursively added in lines 3, 9, 11 and 13 of Algorithm 2 and depend on its input tuple of linear functions f, which, in turn, are inductively defined over $[0,1]^{|L_0|}$: first by π (Algorithm 1, line

Algorithm 2 NN2PWL-R: recursive routine called by NN2PWL

Input: A tuple $\Xi_N = \langle \Xi_1, \ldots, \Xi_{|L_\Lambda|} \rangle$, a ReLU-TId neural network N, for which $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, with a distinguished layer $L_i \neq L_0$, a set of inequalities Ω and a tuple of linear functions $f = \langle f_1, \ldots, f_{|L_{i-1}|} \rangle$.

1: **if** $L_i \neq L_\Lambda$ **then**
2: **for** $\bowtie \in \{\leq, \geq\}^{|L_i|}$ **do**
3: $\Omega^i_{\bowtie} := \Omega \cup \{f^i_j \circ f(\mathbf{x}) \bowtie_j 0 \mid j = 1, \ldots, |L_i|\}$;
4: $f^i_{\bowtie} := \langle \chi_{|L_0|}(\bowtie_1) \cdot (f^i_1 \circ f), \ldots, \chi_{|L_0|}(\bowtie_{|L_i|}) \cdot (f^i_{|L_i|} \circ f) \rangle$;
5: NN2PWL-R$(\Xi_N \parallel L_{i+1}, \Omega^i_{\bowtie}, f^i_{\bowtie})$;
6: **end for**
7: **else**
8: **for** $k = 1, \ldots, |L_\Lambda|$ **do**
9: $\Omega_{\leq} := \Omega \cup \{f^\Lambda_k \circ f(\mathbf{x}) \leq 0\}$;
10: $\Xi_k := \Xi_k \cup \{\langle \kappa_0^{|L_0|}, \Omega_{\leq} \rangle\}$;
11: $\Omega_{\lesseqgtr} := \Omega \cup \{f^\Lambda_k \circ f(\mathbf{x}) \geq 0,\ f^\Lambda_k \circ f(\mathbf{x}) \leq 1\}$;
12: $\Xi_k := \Xi_k \cup \{\langle f^\Lambda_k \circ f, \Omega_{\lesseqgtr} \rangle\}$;
13: $\Omega_{\geq} := \Omega \cup \{f^\Lambda_k \circ f(\mathbf{x}) \geq 1\}$;
14: $\Xi_k := \Xi_k \cup \{\langle \kappa_1^{|L_0|}, \Omega_{\geq} \rangle\}$;
15: **end for**
16: **end if**

4) in the first call of NN2PWL-R (Algorithm 1, line 5); then, by $f^i_{\bowtie}$, for $i \in \{2, \ldots, \Lambda\}$ (Algorithm 2, line 4) in the following $\Lambda - 1$ calls of NN2PWL-R (Algorithm 2, line 5). Let $\mathbf{x} \in [0,1]^{|L_0|}$; note that among the possibilities for inequalities to be added in each step of the construction of regions, there is certainly one that is satisfied by $\mathbf{x}$. Thus, with the suitable configuration of symbols, there is a region Ω of Ξ_k built such that $\mathbf{x} \in \Omega$. Now, in the construction of two regions Ω' and Ω'' of Ξ_k, with $\Omega' \neq \Omega''$, there is some step where the added inequalities differ for Ω' and Ω'' for the first time. Such differing inequalities guarantee that $\Omega'^{\circ} \cap \Omega''^{\circ} = \varnothing$, whether they appear in an intermediate step or the final one. ■

Lemma 2 *Let a ReLU–TId neural network N, for which $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, be given as input to Algorithm 1 and let Ξ_k be an entry in the outputted tuple Ξ_N for which $\langle p, \Omega \rangle \in \Xi_k$. If $\mathbf{x} \in \Omega$, then $N(\mathbf{x})_k = p(\mathbf{x})$.* □

PROOF Let $\mathbf{x} \in \Omega$. Note that $\Omega = \Omega_{[0,1]} \cap \Omega^1_{\bowtie} \cap \cdots \cap \Omega^{\Lambda-1}_{\bowtie} \cap \Omega_{\bowtie}$ and $\Omega_{[0,1]} \supseteq \Omega^1_{\bowtie} \supseteq \cdots \supseteq \Omega^{\Lambda-1}_{\bowtie} \supseteq \Omega_{\bowtie}$, where $\Omega^i_{\bowtie}$ is such that $\bowtie \in \{\leq, \geq\}^{|L_i|}$ and $\Omega_{\bowtie} \in \{\Omega_{\leq}, \Omega_{\lesseqgtr}, \Omega_{\geq}\}$. Then, as $\mathbf{x} \in \Omega_{[0,1]}$, $\mathbf{x} \in [0,1]^{|L_0|}$. Also, as $\mathbf{x} \in \Omega^i_{\bowtie}$, for $i \in \{1, \ldots, \Lambda - 1\}$, the tuples of functions $f^i_{\bowtie}$ defined in line 4 of Algorithm 2, given as arguments in the recursive call of NN2PWL-R, are such that $f^i_{\bowtie}(\mathbf{x}) = L_i \circ \cdots \circ L_1(\mathbf{x})$. Indeed, as $\mathbf{x} \in \Omega^1_{\bowtie}$, $\mathbf{x}$ satisfies the inequalities

$$f^1_1(\mathbf{x}) = f^1_1 \circ \pi(\mathbf{x}) \bowtie_1 0, \quad \ldots, \quad f^1_{|L_1|}(\mathbf{x}) = f^1_{|L_1|} \circ \pi(\mathbf{x}) \bowtie_{|L_1|} 0.$$

Then, we have that

$$f^1_{\bowtie}(\mathbf{x}) = \langle \chi(\bowtie_1) \cdot f^1_1 \circ \pi(\mathbf{x}), \ldots, \chi(\bowtie_{|L_1|}) \cdot f^1_{|L_1|} \circ \pi(\mathbf{x}) \rangle = \langle \mathrm{ReLU}(f^1_1(\mathbf{x})), \ldots, \mathrm{ReLU}(f^1_{|L_1|}(\mathbf{x})) \rangle = L_1(\mathbf{x}).$$

Now, let us assume that $f^i_{\bowtie}(\mathbf{x}) = L_i \circ \cdots \circ L_1(\mathbf{x})$, for $\mathbf{x} \in \Omega$. As, in particular, $\mathbf{x} \in \Omega^{i+1}_{\bowtie}$, $\mathbf{x}$ satisfies the inequalities

$$f^{i+1}_1 \circ f^i_{\bowtie}(\mathbf{x}) = f^{i+1}_1 \circ L_i \circ \cdots \circ L_1(\mathbf{x}) \bowtie_1 0, \quad \ldots, \quad f^{i+1}_{|L_{i+1}|} \circ f^i_{\bowtie}(\mathbf{x}) = f^{i+1}_{|L_{i+1}|} \circ L_i \circ \cdots \circ L_1(\mathbf{x}) \bowtie_{|L_{i+1}|} 0,$$

it follows that

$$\begin{aligned} f^{i+1}_{\bowtie}(\mathbf{x}) &= \langle \chi(\bowtie_1)\cdot f^{i+1}_1 \circ L_i \circ \cdots \circ L_1(\mathbf{x}), \quad \ldots, \quad \chi(\bowtie_{|L_1|})\cdot f^{i+1}_{|L_1|} \circ L_i \circ \cdots \circ L_1(\mathbf{x})\rangle \\ &= \langle \mathrm{ReLU}(f^{i+1}_1 \circ L_i \circ \cdots \circ L_1(\mathbf{x})), \quad \ldots, \quad \mathrm{ReLU}(f^{i+1}_{|L_1|} \circ L_i \circ \cdots \circ L_1(\mathbf{x}))\rangle \\ &= L_{i+1} \circ \cdots \circ L_1(\mathbf{x}). \end{aligned}$$

Finally, in case $\Omega_{\bowtie} = \Omega_{\leqslant}$ (Algorithm 2, line 11), as, in particular, $\mathbf{x} \in \Omega_{\leqslant}$, we have that

$$0 \leq f^{\Lambda}_k \circ L_{\Lambda-1} \circ \cdots \circ L_1(\mathbf{x}) \leq 1.$$

Therefore,

$$N(\mathbf{x})_k = \mathrm{TId}(f^{\Lambda}_k \circ L_{\Lambda-1} \circ \cdots \circ L_1(\mathbf{x})) = f^{\Lambda}_k \circ L_{\Lambda-1} \circ \cdots \circ L_1(\mathbf{x}) = p(\mathbf{x}).$$

The other cases where Ω is either $\Omega_{<}$ or $\Omega_{>}$ are similar. ■

Theorem 1 (Correctness) *Let a ReLU–TId neural network N, for which $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, be given as input to Algorithm 1 and let $\Xi_N = \langle \Xi_1, \ldots, \Xi_n \rangle$ be its output. Then, each entry Ξ_k in Ξ_N codifies a rational McNaughton function in the pre-closed regional format which is exactly the function computed by N through the path to its k-th output node.* □

PROOF By construction and Lemma 1, regions in Ξ_k comply to the properties of pre-closed regional format. Lemma 2 establishes that evaluation via Ξ_k is the same as via the k-th output node. Since the function computed via the k-th output node is a composition of continuous functions—both linear functions associated to nodes of N and activation functions—, it is a continuous function. Therefore, entries in the tuple Ξ_N codify continuous functions which are rational McNaughton functions. ■

Corollary 1 *ReLU–TId neural networks are ν-rational McNaughton neural networks, where $\nu = |L_\Lambda|$.* □

3.1 Decreasing the execution time of the base algorithm

The base algorithm just introduced has the downside to be exponential in the number of nodes of a given neural network. For a neural network N with $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, we have seen that it computes $3|L_\Lambda| \times 2^{|L_1|+\cdots+|L_{\Lambda-1}|}$ regions. However, many of such regions may be the empty set, which makes the outputs of the base algorithm examples of degenerate codification in pre-closed regional format.

Example 3 In a configuration of symbols where we associate $\geq$ to n^1_1 and $\leq$ to n^1_2 in the neural network E in Example 1, the corresponding inequalities $\frac{4}{3}x_1 - x - 2 \geq 0$ and $x_1 - x_2 + \frac{1}{2} \leq 0$ together, related to the first layer L_1, determine the empty set. □

In the step-by-step construction of a region $\Omega = \varnothing$ by the base algorithm, there is some step from the second when equations are added to the current polyhedron turning it into the empty set. In view of that, the first addition proposed for decreasing the execution time of the base algorithm consists in:

- Conditioning the call of NN2PWL-R in line 5 of Algorithm 2 by placing it within the scope of an if-statement that verifies whether $\Omega^i_{\bowtie} \neq \varnothing$;
- Conditioning the addition of new pairs $\langle p, \Omega_{\bowtie} \rangle$, for all $\bowtie \in \{\leq, \leqslant, \geq\}$, to tuple Ξ_N in lines 10, 12 and 14 of Algorithm 2 by placing these commands within the scope of if-statements that verify whether $\Omega_{\leq} \neq \varnothing$, $\Omega_{\leqslant} \neq \varnothing$ and $\Omega_{\geq} \neq \varnothing$.

Verifying whether $\Omega_{\bowtie} \neq \varnothing$ might significantly decrease the running time of the translation algorithm in practice. Indeed, each true statement $\Omega_{\bowtie} \neq \varnothing$ occurring in the the i-th step of the construction of regions, for $i \in \{1, \ldots, \Lambda - 1\}$, avoids a call of NN2PWL-R that, in the pure base algorithm, would yield

the computation of $3|L_\Lambda| \times 2^{|L_{i+1}|+\cdots+|L_{\Lambda-1}|}$ pairs $\langle p, \Omega \rangle$. On the other hand, verifying whether $\Omega_{\leq} \neq \varnothing$, $\Omega_{\lneq} \neq \varnothing$ or $\Omega_{\geq} \neq \varnothing$ in the last step of the construction of regions only prevents the algorithm to add pairs with empty regions to the regional format codification, which, nevertheless, makes the final representative tuple Ξ_N smaller.

A possible way to verify whether a polyhedron Ω given as in (3) is nonempty is by applying the known polynomial techniques used to verify whether a linear optimization program constrained by Ω is feasible [2].

For another method for easing the execution time of the base algorithm, observe that, for a layer L_i, for $i \in \{1, \ldots, \Lambda - 1\}$, each of the hyperplanes $f_j^i(\mathbf{x}) = 0$ related to nodes n_j^i of L_i, for $j \in \{1, \ldots, |L_i|\}$, divides the euclidean space $\mathbb{R}^{|L_0|}$ in two half-spaces determined by the inequalities $f_j^i(\mathbf{x}) \geq 0$ and $f_j^i(\mathbf{x}) \leq 0$. Each of these inequalities is added to half of the $2^{|L_i|}$ polyhedra generated in the first for-loop of Algorithm 2 (lines 2 to 6); these are the polyhedra generated in the i-th step of the construction of regions. Now, note that if the hyperplane $f_j^i(\mathbf{x}) = 0$ does not intercept the interior of the unit cube $[0,1]^{|L_0|}$, half of the new generated polyhedra are certainly empty. For instance, the hyperplane $x_1 + x_2 - 2 = 0$ does not intercept $[0,1]^{|L_0|}$. Thus, although the half-space given by $x_1 + x_2 - 2 \leq 0$ contains the entire unit cube $[0,1]^{|L_0|}$, the half-space given by $x_1 + x_2 - 2 \geq 0$ does not intersect it; so, if $x_1 + x_2 - 2 \geq 0$ is added to a polyhedron in some step of the construction of regions by the base algorithm, the regions generated from such polyhedron will be the empty set.

Thus, for the step related to layer L_i, for $i \in \{1, \ldots, \Lambda\}$, in the construction of regions, the proposed method consists in building a set $\mathbf{I} \subseteq \{\leq, \geq\}^{|L_i|}$ to be iterated instead of the set $\{\leq, \geq\}^{|L_i|}$ in the for-loop beginning in line 2 of Algorithm 2, so avoiding the generation of empty polyhedra. For that, we compute $\mathbf{I} = \mathbf{I}_1 \times \cdots \times \mathbf{I}_{|L_i|}$ where, for $j \in \{1, \ldots, |L_i|\}$,

$$\mathbf{I}_j = \begin{cases} \{\geq, \leq\}, & \text{if } f_j^i(\mathbf{x}) = 0 \text{ intercepts the interior of } [0,1]^{|L_0|} \\ \{\leq\}, & \text{if } f_j^i(\mathbf{x}) \leq 0 \text{ contains the entire } [0,1]^{|L_0|} \\ \{\geq\}, & \text{if } f_j^i(\mathbf{x}) \geq 0 \text{ contains the entire } [0,1]^{|L_0|} \end{cases}$$

Determining which is the case for each $\mathbf{I}_j$ may be done by solving both of the following maximization and minimization linear programs, which are known to be solvable in polynomial time [2]:

$$\begin{array}{ll} \max / \min & f_j^i(\mathbf{x}) \\ \text{subject to} & [0,1]^{|L_0|} \end{array}$$

Let M and m respectively be the maximum and the minimum optimum values of the linear programs above. Then: if $M \geq 0$ and $m \leq 0$ or if $M \leq 0$ and $m \geq 0$, $f_j^i(\mathbf{x}) = 0$ intercepts $[0,1]^{|L_0|}$; if $M \geq 0$ and $m \geq 0$, $f_j^i(\mathbf{x}) \geq 0$ contains $[0,1]^{|L_0|}$; and if $M \leq 0$ and $m \leq 0$, $f_j^i(\mathbf{x}) \leq 0$ contains $[0,1]^{|L_0|}$. The overall execution time of the translation algorithm, even with an additional routine for building $\mathbf{I}$, might be significantly smaller than the time for the original base algorithm. In fact, let $J \subseteq \{1, \ldots, |L_i|\}$ be the set of indexes such that $\mathbf{I}_j \neq \{\leq, \geq\}$ if, and only if, $j \in J$; then, the for-loop beginning in line 2 of Algorithm 2 has $2^{|L_i|-|J|}$ iterations instead of $2^{|L_i|}$.

Combining both of the methods described in this section with the base translation algorithm makes it compute exactly the same pairs $\langle p, \Omega \rangle$ that it would compute without such methods with the exception of the ones for which $\Omega = \varnothing$. Therefore, we are able to establish the following result.

Theorem 2 *Replacing the routine* NN2PWL-R *for a version of it that includes the methods proposed in this section maintains the correctness of Algorithm 1 established in Theorem 1.* □

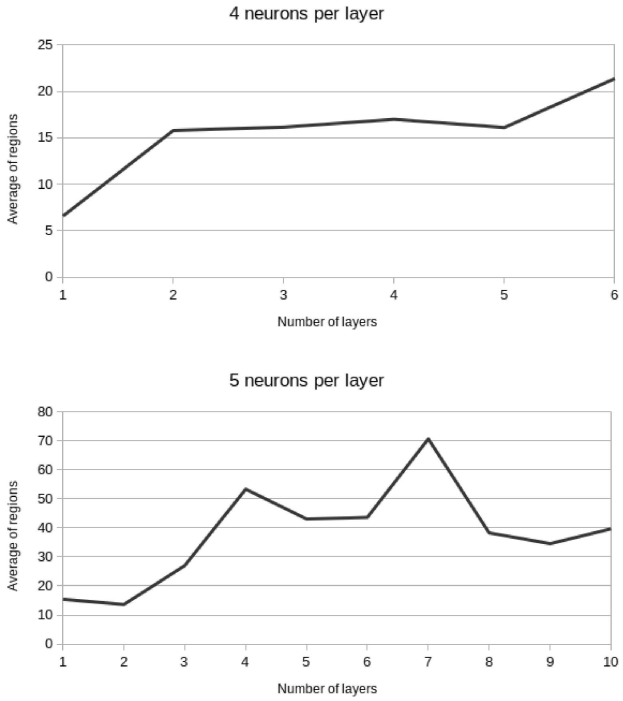


Figure 5: Experiments increasing the number of layers

4 Experiments and Results

We perform experiments for measuring the *complexity* of pre-closed regional format encodings of randomly generated ReLU–TId neural networks by counting the number of nonempty regions in them. All weights of the neural networks have the form $i+d$, where both i and d are uniformly generated from $\{-1,0,1\}$ and $[0,1)$, respectively.

For each encoding in pre-closed regional format, we also evaluate its *degree of satisfiability* of the lattice property by counting the number of pairs of regions $\langle \Omega_i, \Omega_j \rangle$ for which there is no linear piece p_k such that p_i is above p_k over Ω_i and p_k is above p_j over Ω_j, that is the number of pairs $\langle \Omega_i, \Omega_j \rangle$ that falsifies lattice property. If the counting is 0, such an encoding completely satisfies lattice property; the higher the count the further from satisfying lattice property the encoding is.

Implementations of NN2PWL, including the methods for decreasing its execution time, and a neural network generator were developed for the experiments; the source code is publicly available.[2]

In the first batch of experiments, for a fixed value h, ReLU–TId neural networks with h input neurons, h neurons in each hidden layer and one output neuron are generated. Such random generation is done in such a way that the neural networks are partitioned in L classes, each containing n neural networks with l hidden layers, for $l \in \{1,\ldots,L\}$. We ran such experiment for two parameter setups: $h = 4$, $L = 6$, $n = 50$ and $h = 5$, $L = 10$, $n = 25$. Figure 5 depicts the average number of regions extracted from the neural networks in each class of l hidden layers.

In order to analyze whether the results of the previous experiments depend on the distribution of

[2] `http://github.com/spreto/reluka`

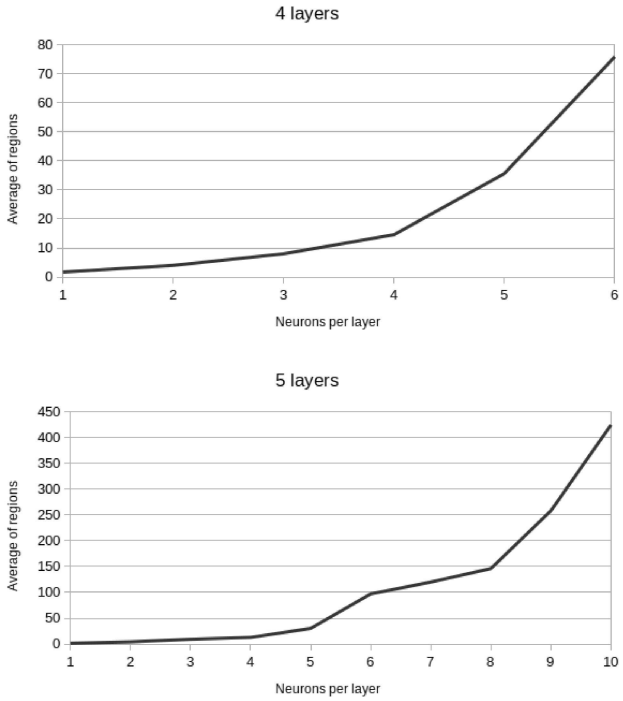


Figure 6: Experiments increasing the number of neurons per layer

neurons per layer, in the second batch of experiments, ReLU–TId neural networks with a fixed number l of hidden layers and one output neuron are generated. Now, the randomly generated neural networks are partitioned in M classes, each containing n neural networks with m input neurons and m neurons in each of their hidden layers, for $m \in \{1, \ldots, M\}$. We ran such experiment for two parameter setups: $l = 4$, $M = 6$, $n = 50$ and $l = 5$, $M = 10$, $n = 25$. Figure 6 depicts the average number of regions extracted from the neural networks in each class of m neurons in the input layer and per hidden layer.

Note that the first experiment in both batches of experiments are related: for $l = m$, neural networks in a class with l layers (first experiment, first batch) have the same number of neurons than the neural networks in a class with m nodes per layer (first experiment, second batch). The same relation may be seen between the second experiments of each batch.

In all experiments, we may see that the average number of regions increases as long as the number of neurons increases. However, while such variation in the number of regions is smooth for varying the number of layers, a sharp variation may be perceived for varying the number of neurons per layer. A neural network with 5 hidden layers and 10 neurons in each of them (50 neurons in all hidden layers) achieved the maximum number of 1852 regions among all neural networks generated. For comparison, among the neural networks with 50 neurons distributed in 10 hidden layers (5 neurons per hidden layer), the maximum number of regions achieved is 228. And among all the neural networks with more than 5 hidden layers, but only 5 neurons in each of them, the maximum number of regions achieved is 446 (in a neural network with 7 hidden layers).

Regarding lattice property, among all 1100 ReLU–TId neural networks that were generated in all experiments, only one failed to satisfy it. Such neural network has 5 neurons in each of its 5 hidden layers (25 neurons in all hidden layers) and its pre-closed regional format encoding has 91 regions and

fails to fulfill lattice property for 36 pairs of regions $\langle \Omega_i, \Omega_j \rangle$.

5 Conclusions

We have proposed an algorithm for translating ReLU–TId neural networks into the pre-closed regional format, which is a more interpretable representation than the traditional graph one. We also proposed methods for decreasing the computation time of the base algorithm and proved that ReLU–TId neural networks are ν-rational McNaughton neural networks.

Empirically, we measured the complexity of pre-closed regional format encodings of randomly generated ReLU–TId neural networks by counting the number of nonempty regions in such encodings. We could verify a bigger increase in the number of regions in the encodings with wider, but fewer, layers than in the encodings with more, but thinner, layers. The fast increase of curves in Figure 6, related to the variation in the size of a fixed number of layers, points to the high complexity of regional representation. Therefore, the reported results foresee scaling issues in the regional representation of real-world neural networks, which often are larger than those generated in our investigation.

We have also investigated the degree of satisfiability of the lattice property by the neural networks generated in our experiments. The results empirically indicate that the outputs of NN2PWL lacking lattice property are a very rare event. Only one of the neural networks generated do not fulfill such a property.

For the future, approximate and less complex regional representations might be pursued. A possible path is to establish the reasonability of allowing encodings not satisfying lattice property as approximations of neural networks. From an exact perspective, one might investigate efficient procedures for turning a rational McNaughton function encoding in pre-closed regional format into closed regional format.

Funding

This work was carried out at the Center for Artificial Intelligence (C4AI-USP), with support by the São Paulo Research Foundation (FAPESP) [grant #2019/07665-4] and by the IBM Corporation. This study was financed in part by the São Paulo Research Foundation (FAPESP) [grants #2021/03117-2 to S.P., #2015/21880-4 and #2014/12236-1 to M.F.]; and the National Council for Scientific and Technological Development (CNPq) [grant PQ 303609/2018-4 to M.F.].

References

[1] Brendon G. Anderson, Samuel Pfrommer & Somayeh Sojoudi (2023): *Tight Certified Robustness via Min-Max Representations of ReLU Neural Networks*. In: *2023 62nd IEEE Conference on Decision and Control (CDC)*, pp. 6348–6355, doi:10.1109/CDC49753.2023.10383700.

[2] Dimitris Bertsimas & John N. Tsitsiklis (1997): *Introduction to linear optimization*. Athena scientific series in optimization and neural computation, Athena Scientific.

[3] Davide Castelvecchi (2016): *Can we open the black box of AI?* Nature 538(7623), pp. 20–23, doi:10.1038/538020a.

[4] Roberto L.O. Cignoli, Itala M.L. D'Ottaviano & Daniele Mundici (2000): *Algebraic Foundations of Many-Valued Reasoning*. Trends in Logic, Springer Netherlands, doi:10.1007/978-94-015-9480-6.

[5] Marcelo Finger (2020): *Logic in Times of Big Data*. In J. Acacio de Barros & Décio Krause, editors: *A True Polymath: A Tribute to Francisco Antonio Doria*, College Publications, pp. 184–198.

[6] Marcelo Finger & Sandro Preto (2020): *Probably Partially True: Satisfiability for Łukasiewicz Infinitely-Valued Probabilistic Logic and Related Topics*. Journal of Automated Reasoning 64(7), pp. 1269–1286, doi:10.1007/s10817-020-09558-9.

[7] Brunella Gerla (2001): *Rational Łukasiewicz Logic and DMV-algebras*. Neural Network World 11(6), pp. 579–594, doi:10.48550/arXiv.1211.5485

[8] Ian Goodfellow, Yoshua Bengio & Aaron Courville (2016): *Deep Learning*. MIT Press.

[9] R. McNaughton (1951): *A Theorem About Infinite-Valued Sentential Logic*. Journal of Symbolic Logic 16, pp. 1–13, doi:10.2307/2268660.

[10] Daniele Mundici (1994): *A constructive proof of McNaughton's theorem in infinite-valued logic*. The Journal of Symbolic Logic 59(2), pp. 596–602, doi:10.2307/2275410.

[11] Sandro Preto & Marcelo Finger (2020): *An Efficient Algorithm for Representing Piecewise Linear Functions into Logic*. Electronic Notes in Theoretical Computer Science 351, pp. 167–186, doi:10.1016/j.entcs.2020.08.009. Proceedings of LSFA 2020, the 15th International Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2020).

[12] Sandro Preto & Marcelo Finger (2022): *Efficient representation of piecewise linear functions into Łukasiewicz logic modulo satisfiability*. Mathematical Structures in Computer Science 32(9), pp. 1119–1144, doi:10.1017/S096012952200010X.

[13] Sandro Preto & Marcelo Finger (2023): *Effective Reasoning over Neural Networks Using Łukasiewicz Logic*. In Pascal Hitzler, Md Kamruzzaman Sarker & Aaron Eberhart, editors: Compendium of Neurosymbolic Artificial Intelligence, chapter 28, Frontiers in Artificial Intelligence and Applications 369, IOS Press, pp. 609–630, doi:10.3233/FAIA230160.

[14] Sandro Preto & Marcelo Finger (2023): *Proving properties of binary classification neural networks via Łukasiewicz logic*. Logic Journal of the IGPL 31(5), pp. 805–821, doi:10.1093/jigpal/jzac050.

[15] Haakon Robinson, Adil Rasheed & Omer San (2019): *Dissecting deep neural networks*. arXiv preprint *arXiv:1910.03879*, doi:10.48550/arXiv.1910.03879