Logical Approximation and Formal Verification of Neural Networks

João Lobo[1], Marcelo Finger[1], Sandro Preto[2,1]
[1]Institute of Mathematics and Statistics
University of São Paulo – São Paulo – Brazil
[2]Center for Mathematics, Computing and Cognition
Federal University of ABC – Santo André – Brazil
{joaolobo,mfinger}@ime.usp.br

Abstract. *Explainability and formal verification of neural networks may be crucial when using these models to perform critical tasks. Pursuing explainability properties, we present a method for approximating neural networks by piecewise linear functions, which is a step to achieve a logical representation of the network. We also explain how such logical representations may be applied in the formal verification of some properties of neural networks. Furthermore, we present the results of an empirical experiment where the methods introduced are used in a case study.*

1. Introduction

Explainability of neural networks models has become of great interest to society, as the need to explain the reasoning behind the decision-making of such models has arisen greatly. For instance, for neural networks that perform critical tasks such as issuing aircraft collision avoidance alerts.

A path towards explainability of neural networks that compute piecewise linear functions is to represent them in a more amenable format, given by pairs comprehending a domain region and a linear function that realizes the network computation in such region. Further, from such a regional representation, a logical representation may be given, for instance, by a pair of formulas of Łukasiewicz infinitely-valued logic ($Ł_\infty$). A polynomial procedure that builds this logical representation is introduced in [Preto and Finger 2020, Preto and Finger 2022].

However, neural networks do not necessarily compute piecewise linear functions. And neural networks that compute such functions may have an exponential regional format representation in the size of their traditional graph representations. The main goal of this work is to propose a method for approximating neural networks by piecewise linear functions in regional format, allowing their approximate logical representation in $Ł_\infty$.

Another path to reliability in critical tasks for neural networks is to formally verify properties of interest concerning them. In [Preto and Finger 2023a, Preto and Finger 2023b], some of these properties are codified in the language of $Ł_\infty$ departing from the network representation in such logical system. Then, the properties are verified through decisions about satisfiability and logical consequence of their codification. This work aims to present these concepts through examples and to perform an empirical experiment of formal verification using the logical approximation of a neural network.

2. Preliminaries: Łukasiewicz Logic and McNaughton Functions

As our final goal is to represent neural networks in Łukasiewicz infinitely-valued logic [Cignoli et al. 2000], let us start by introducing such system. The basic language $\mathcal{L}$ of Ł_∞ 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 $v : \mathcal{L} \to [0, 1]$, such that, for $\varphi, \psi \in \mathcal{L}$, $v(\varphi \oplus \psi) = \min(1, v(\varphi) + v(\psi))$ and $v(\neg\varphi) = 1 - v(\varphi)$. From disjunction and negation, we derive the following operators:

Implication: $\varphi \to \psi =_{\text{def}} \neg\varphi \oplus \psi$	$v(\varphi \to \psi) = \min(1, 1 - v(\varphi) + v(\psi))$
Maximum: $\varphi \vee \psi =_{\text{def}} \neg(\neg\varphi \oplus \psi) \oplus \psi$	$v(\varphi \vee \psi) = \max(v(\varphi), v(\psi))$
Minimum: $\varphi \wedge \psi =_{\text{def}} \neg(\neg\varphi \vee \neg\psi)$	$v(\varphi \wedge \psi) = \min(v(\varphi), v(\psi))$
Bi-implication: $\varphi \leftrightarrow \psi =_{\text{def}} (\varphi \to \psi) \wedge (\psi \to \varphi)$	$v(\varphi \leftrightarrow \psi) = 1 - \lvert v(\varphi) - v(\psi)\rvert$

A formula φ is *satisfiable* if there exists a valuation v such that $v(\varphi) = 1$. A formula φ is said to be a *(semantic) consequence* of a set of formulas Φ if each valuation that satisfies Φ also satisfies φ; in this case we write $\Phi \models \varphi$. We also call an expression as $\Phi \models \varphi$ an *instance of logical consequence* and say that such expression is *valid* if it represents an actual consequence.

McNaughton functions are continuous piecewise linear functions whose domain is $[0, 1]^n$, for some $n \in \mathbb{N}$, and range is in $[0, 1]$; also, their linear pieces only have integer coefficients. McNaughton functions may be represented in Ł_∞ in such a way that there is a one-to-one correspondence between McNaughton functions and the elements of the Lindenbaum algebra of Ł_∞-formulas [McNaughton 1951, Mundici 1994]. Unfortunately, McNaughton functions cannot approximate continuous functions in general with any desired precision. For that, one may rely on *rational McNaughton functions*, which are as McNaughton functions, but allowing rational coefficients in their linear pieces. In this work, we focus on neural networks that may be approximated by rational McNaughton functions, that is networks with n input values in $[0, 1]^n$ and one output value in $[0, 1]$.

Although Ł_∞ cannot express rational McNaughton functions in the traditional way, such a logical system may implicitly represent them using the technique of *representation modulo satisfiability* [Finger and Preto 2020, Preto and Finger 2020, Preto and Finger 2022], which we introduce in the following.

Let us denote the Ł_∞*-semantics*, that is the set of all valuations, by $\mathbf{Val}$. 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_∞*-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 $v \in \mathbf{Val}_{\Phi_f}$.

For a formula φ and $n \in \mathbb{N}^*$, we denote the disjunction $\varphi \oplus \cdots \oplus \varphi$, n times, by $n\varphi$.

Let Ω° denote the topological interior of set Ω. There is a polynomial algorithm for the representation in Ł$_\infty$-MODSAT having as input a rational McNaughton function given in *closed regional format*: a collection of pairs $\langle p_i, \Omega_i \rangle$ of linear pieces p_i and regions Ω_i such that the regions are polyhedra that satisfy:

- $\bigcup_i \Omega_i = [0,1]^n$;
- $\Omega_i^\circ \cap \Omega_j^\circ = \emptyset$, for $i \neq j$; 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 – i.e., $p_i(\mathbf{x}) \geq p_k(\mathbf{x})$, for $\mathbf{x} \in \Omega_i$ – and linear piece p_k is above linear piece p_j over region Ω_j.

If there is no guarantee that such a collection of pairs satisfy the lattice property, we say that it is in *pre-closed regional format*.

3. Piecewise Linear Approximation of Neural Networks

We present an algorithm for approximating a neural network by a rational McNaughton function in pre-closed regional format. Piecewise linear regression is a technique used to try and find a collection of simpler linear functions that are able to approximate a continuous function, which is known to be possible due to the Stone-Weierstrass Approximation Theorem [Weierstrass 1885].

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function determined by a neural network and $D_f \subset \mathbb{R}^n$ its domain. The goal is to find an estimator $\hat{f}$ of f such that

$$\hat{f}(x) = \begin{cases} p_1(x), & \text{if } x \in S_1 \\ \vdots & \\ p_k(x), & \text{if } x \in S_k \end{cases}$$

where $p_i : \mathbb{R}^n \to \mathbb{R}$ is an element of a collection of linear functions $P = \{p_1, ..., p_k\}$ and S_i is an element of a collection $\mathbf{S} = \{S_1, ..., S_k\}$ of subsets of $\mathbb{R}^n$ such that $\bigcup_{i=0}^{k} S_i = D_f$.

3.1. An Algorithm for Piecewise Regression

Dynamic programming algorithms to solve piecewise regression problems are introduced in [Acharya et al. 2016]. The algorithm described in this work is a variation of those and breaks segments based on an input cost C instead of specifying how many pieces the regression should have.

For this we consider the following formulation: let (X, y) be a dataset of points sampled from an arbitrary function $f : \mathbb{R}^n \to \mathbb{R}$, where X is a matrix of points sampled from f and y is a vector with the corresponding output values of f. For an estimator $\hat{f}$ of f, the square error generated by the estimator, for m input-ouput pairs $(x_i, y_i)_{i=1}^m$, with $i, j, m \in \mathbb{N}$, from the dataset is $\sum_{i=j}^{m} (y_i - \hat{f}(x_i))^2$. The goal of the algorithm is to find an estimator $\hat{f}$, which should be a piecewise linear function with k linear pieces, such that it minimizes the square error between the approximation and the observed data.

Let $OPT(j)$ be the minimum possible error considering only pairs $\{(x_p, y_p)\}_{p \in J}$ from (X, y) in the interval of indices $J = \{1, \ldots, j\}$. Let, also, $err(i, j)$, where $i < j$, be the least squared error generated by a segment fitted through the points lying in the interval of indices $I = \{i, i+1, \ldots, j\}$. The optimal segment error for pairs with indices in J can

be formulated as the recursive relationship: $OPT(j) = min_{i \in I}\{err(i,j) + OPT[i-1]\}$. After constructing OPT, we can backtrack and find the solution which has the minimum overall squared error to determine the linear pieces. We propose Algorithm 1, heavily based on the one presented by Acharya et al., that constructs such table OPT.

Algorithm 1 Piecewise Regression by Dynamic Programming

Input
- X Matrix of points sampled from f where each row correponds to a point in $\mathbb{R}^n$
- y Vector containing the outputs of the f evaluated on each point of X
- C Cost for creating a segment

Output
- Cost for creating the optimal piecewise linear function

$OPT[0] \leftarrow 0$
for $j \in \{1, ..., N\}$ **do**
  for $i \in \{1, ..., j\}$ **do**
    $err(i,j) \leftarrow$ least square error for indices in the interval $\{i, i+1, ..., j\}$
  end for
end for
for $j \in \{1, ..., N\}$ **do**
  $OPT[j] = min_{i<j}(err(i,j) + OPT[i-1] + C)$
end for
return $OPT[n]$

The intuition behind how this works is that each interval of points is treated as a sub-problem. If we have a point x_j that is the last point of the optimal segment that starts at x_i, where $i < j$, we can compute the cost, $OPT(j)$, of fitting a segment through $\{x_i, ..., x_j\}$ if we know the cost $OPT(j-1)$. Also, by varying the value of C, we can determine how many segments the algorithm should create because the final cost value will contain C times the number of segments.

Before obtaining an approximation for the function, we must address two issues:

- The order in which the points are presented to Algorithm 1 matters and it generates different linear pieces depending on how the data is presented;
- The resulting approximation is not guaranteed to be continuous.

3.2. Order Matters

Taking a closer look at the algorithm, it is imperative that exists a partial order underlying the data. Imagine that, for the one-dimensional case, we have a set of points ordered by the x-axis. Suppose there are two optimal segments that divide this dataset: p_1, which is a regression for points $\{x_1, ..., x_j\}$; and p_2, which is one for points $\{x_{j+1}, ..., x_n\}$. If we purposefully create a dataset where we present $\{x_1, ..., x_{j+1}, x_j\}$ to the algorithm, it will try to construct the optimal segment of x_j using the cost for x_{j+1}, which lies further in the x-axis and that will generate a completely different approximation from before. Because of that, we need to establish ways to sort the data before trying to apply the algorithm to it.

For the one-dimensional case, just ordering the points based on the x-axis is sufficient. For greater dimensions, we experimented with two arrangements. The first one was described in [Acharya et al. 2016]: sort the data by a determined coordinate; in this case, by the first coordinate x_1 of every point. We call this type of ordering $\mathbf{X1}$ from now on. The other arrangement, which we call $\mathbf{C}$, sorts points by values $c = x_1 - x_2$, created based on prior knowledge of the function for experimentation purposes only.

3.3. Simplex Covering: Forcing Continuity

Another issue to be addressed is that, because the algorithm is based on a discrete set of input points, all intervals of domain points between two created segments do not have an approximate linear function assigned to them. Suppose that, in the approximating procedure, we get two consecutive linear pieces p_i and p_k. Let p_i be the linear approximation of points in the interval delimited by x_i and x_j and p_k be the same for points x_k and x_l, where $i < j < k < l$, for $i, j, k, l \in \mathbb{N}$. We can see that for at least one $j < d < k$, x_d is unmapped in our approximation, in other words it has a discontinuity. That is an issue because the methods we intend to use require our approximation to be continuous.

To account for these points, we may partition the unmapped interval into simplices and associate a linear piece to each of them in a way that makes the whole function continuous. This is particularly useful for us because the algorithm described, coupled with simplex covering, induces a partitioning of the domain that yields a collection of rectangles and triangles, thus guaranteeing that we produce a function in *pre-closed regional format*, at least. Unfortunately, simplex partitioning grows exponentially in complexity as the dimension increases. In the two-dimensional case, which is the focus of our experiments, we need only two simplices (triangles) to cover an unmapped region of the domain, but, for instance, in $[0, 1]^7$, we would need at least 1175 simplices [Hughes and Anderson 1996].

4. Modeling Properties in Łukasiewicz Logic

We focus on properties of a binary classification neural network N with input values in $[0, 1]^n$ and only one output value in $[0, 1]$, which is interpreted as the probability of the input belonging to a class. Thus, for input values $\mathbf{x} \in [0, 1]^n$, if $N(\mathbf{x}) \geq \frac{1}{2}$, $\mathbf{x}$ belongs to the class in question and, if $N(\mathbf{x}) < \frac{1}{2}$, $\mathbf{x}$ does not belong.

Let $\langle \varphi, \Phi \rangle$ be an approximate representation of N in Ł_∞-MODSAT, that is a representation of a piecewise linear approximation of the function computed by N. We follow [Preto and Finger 2023b] and show that properties of *reachability* and *robustness* may be modeled in the language of Ł_∞.

The *reachability* of a given state can be thought as the problem of determining if N reaches a specific probability $\pi = \frac{a}{b} \in \mathbb{Q} \cap [0, 1]$ for some input $\mathbf{x} \in [0, 1]^n$. This is the case if, and only if, formula

$$\left(\bigwedge \Phi\right) \wedge \varphi_{\frac{1}{b}} \wedge aZ_{\frac{1}{d}} \leftrightarrow \varphi$$

is satisfiable [Preto and Finger 2023b, Theorem 1].

For *robustness*, we verify if N maintains its prediction of belonging to a class with respect to a perturbation limit $\varepsilon = \frac{\alpha}{\beta} \in \mathbb{Q}$ and a probability $\pi = \frac{a}{b} \in [0, 1] \cap \mathbb{Q}$.

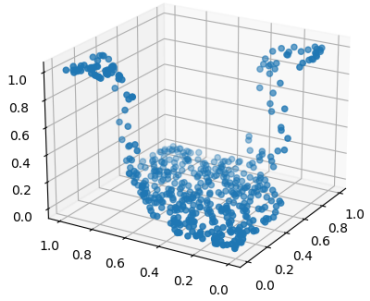

Figure 1. Sampled points of the case study XOR neural network.

That is, wether $N(\mathbf{x}+\mathbf{p}) \geq 0.5$, for all $\mathbf{x} \in [0,1]^n$ and $p = \langle p_1, p_2, \ldots, p_n \rangle \in \mathbb{R}^n$, such that $N(\mathbf{x}) \geq \pi, |p_i| \leq \varepsilon$, for $i = 1, 2, ..., n$, and $\mathbf{x}+\mathbf{p} \in [0,1]^n$. For that, let $\langle \varphi', \Phi' \rangle$ be defined from $\langle \varphi, \Phi \rangle$ replacing each occurrence of a propositional variable X by propositional variable X'. For each propositional variable $X_i \in \{X_1, ..., X_n\}$, let P_i be a new propositional variable. Then, N is robust according to ε and π if, and only if,

$$\begin{aligned}\Phi, \Phi', \varphi_{\frac{1}{\beta}}, \varphi_{\frac{1}{b}}, \varphi_{\frac{1}{2}}, P_1 \to \alpha Z_{\frac{1}{\beta}}, ..., P_n \to \alpha Z_{\frac{1}{\beta}}, aZ_{\frac{1}{b}} \to \varphi,\\ (X_1' \leftrightarrow X_1 \oplus P_1) \vee (X_1' \leftrightarrow \neg(X_1 \to P_1)),\\ \ldots, (X_n' \leftrightarrow X_n \oplus P_n) \vee (X_n' \leftrightarrow \neg(X_n \to P_n)) \vDash Z_{\frac{1}{2}} \to \varphi',\end{aligned}$$

is a valid logical consequence [Preto and Finger 2023b, Theorem 3].

5. Case Study: Approximating and Verifying a Trained Network

A neural network N for solving the XOR problem was trained in order to conduct experiments. The aim is for the network to learn function $f : [0,1]^2 \to [0,1]$, given by

$$f(x_1, x_2) = \begin{cases} 0, & x_1 = x_2 \\ 1, & x_1 \neq x_2 \end{cases}.$$

We can look at N as a binary classification network computing the probability of input $\mathbf{x} \in [0,1]^2$ to produce output 1.

We implemented Algorithm 1 and a module to perform the discussed simplex covering in C++ [1], leveraging the linear algebra library Eigen,[2] to produce a continuous piecewise linear approximation of an input function $f : [0,1]^2 \to [0,1]$. We also implemented a component to translate such output approximation to the input format of the software `pwl2limodsat`,[3] that computes representations in Ł_∞-MODSAT.

For the approximation, 250 random points were sampled from the network and ordered with the mentioned sorting methods; Algorithm 1 was fed with such ordered points (Figure 1). Then, after the simplex segmentation of the parts of the domain that remained unmapped, the approximation was translated into the input format taken by

[1] `https://github.com/juaolobo/linear-piecewise-neuralnets`
[2] `https://eigen.tuxfamily.org/`
[3] `https://github.com/spreto/pwl2limodsat`

`pwl2limodsat`, which produced its representation in Ł$_\infty$-MODSAT. The representation was used to construct encodings of reachability and robustness properties, which were decided by an SMT-based Ł$_\infty$-solver [Ansótegui et al. 2012].

The results obtained for the sorting method **X1** were unsatisfactory. The algorithm generated the maximum amount of linear pieces, taking the minimum amount of neighboring points to create a plane, and doing this for all points. This yielded too many linear pieces, making it very inefficient for the Ł$_\infty$-solver to deal with and deeming the verification of the function untractable.

On the other hand, the pieces generated with method **C** are what one would expect to be the best pieces when looking at the graph of the function. We obtained 3 pieces generated by the algorithm plus 4 simplices used to assert the continuity of the approximation. The representation was checked to be in closed regional format by `pwl2limodsat` methods. Results of the performed verifications are in Table 1.

As expected, given the nature of the XOR problem, the network accesses every single value between 0 and 1. As for robustness, given that the network is in state $\pi = \frac{3}{4}$, we perturbed it by adding a small number to each coordinate of the input. In this particular scenario we can see that the network is fairly robust, because it continues to reach $N(\mathbf{x} + \mathbf{p}) \geq 0.5$ even for a perturbation of 0.25 in every coordinate of the input.

Reachability		*Robustness*	
Parameters	Result	Parameters	Result
$\pi = 0.1$	✓	$\pi = 0.75, \varepsilon = 0.01$	✓
$\pi = 0.2$	✓	$\pi = 0.75, \varepsilon = 0.1$	✓
$\pi = 0.3$	✓	$\pi = 0.75, \varepsilon = 0.2$	✓
$\pi = 0.4$	✓	$\pi = 0.75, \varepsilon = 0.25$	✓
$\pi = 0.5$	✓	$\pi = 0.75, \varepsilon = 0.3$	✗
$\pi = 0.6$	✓	$\pi = 0.75, \varepsilon = 0.35$	✗
$\pi = 0.7$	✓	$\pi = 0.75, \varepsilon = 0.4$	✗
$\pi = 0.8$	✓	$\pi = 0.75, \varepsilon = 0.5$	✗
$\pi = 0.9$	✓		

Table 1. Verification of an approximation of the network, utilizing sorting method C on the input points.

6. Conclusions and Future Work

The results obtained in our experiments were in accordance to expectation and to what we knew beforehand about the function. That entails that if one is able to circumvent the issues presented, this is a valid method to create a piecewise linear approximation of a neural network.

For future works that intend to use such method of approximation, specially when dealing with networks in a higher dimension, one must find an efficient way to assert the continuity of the approximated function and to find the optimal way of ordering the input data before executing the algorithm. Moreover, one can investigate efficient exact methods for representing neural networks which already compute piecewise linear functions.

Funding

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) [Finance Code 001]; the São Paulo Research Foundation (FAPESP) [grants #2015/21880-4 and #2014/12236-1 to M.F., #2021/03117-2 to S.P.]; and the National Council for Scientific and Technological Development (CNPq) [grant PQ 303609/2018-4 to M.F.]. 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.

References

Acharya, J., Diakonikolas, I., Li, J., and Schmidt, L. (2016). Fast algorithms for segmented regression. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2878–2886. JMLR.org.

Ansótegui, C., Bofill, M., Manyà, F., and Villaret, M. (2012). Building automated theorem provers for infinitely-valued logics with satisfiability modulo theory solvers. In *2012 IEEE 42nd International Symposium on Multiple-Valued Logic*, pages 25–30.

Cignoli, R., D'Ottaviano, I., and Mundici, D. (2000). *Algebraic Foundations of Many-Valued Reasoning*. Trends in Logic. Springer Netherlands.

Finger, M. and Preto, S. (2020). Probably partially true: Satisfiability for Łukasiewicz infinitely-valued probabilistic logic and related topics. *Journal of Automated Reasoning*, 64(7):1269–1286.

Hughes, R. B. and Anderson, M. R. (1996). Simplexity of the cube. *Discret. Math.*, 158:99–150.

McNaughton, R. (1951). A theorem about infinite-valued sentential logic. *Journal of Symbolic Logic*, 16:1–13.

Mundici, D. (1994). A constructive proof of McNaughton's theorem in infinite-valued logic. *The Journal of Symbolic Logic*, 59(2):596–602.

Preto, S. and Finger, M. (2020). An efficient algorithm for representing piecewise linear functions into logic. *Electronic Notes in Theoretical Computer Science*, 351:167–186. Proceedings of LSFA 2020, the 15th International Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2020).

Preto, S. and Finger, M. (2022). Efficient representation of piecewise linear functions into Łukasiewicz logic modulo satisfiability. *Mathematical Structures in Computer Science*, 32(9):1119–1144.

Preto, S. and Finger, M. (2023a). Effective reasoning over neural networks using Łukasiewicz logic. In Hitzler, P., Kamruzzaman Sarker, M., and Eberhart, A., editors, *Compendium of Neurosymbolic Artificial Intelligence*, volume 369 of *Frontiers in Artificial Intelligence and Applications*, chapter 28, pages 609–630. IOS Press.

Preto, S. and Finger, M. (2023b). Proving properties of binary classification neural networks via Łukasiewicz logic. *Logic Journal of the IGPL*, 31(5):805–821.

Weierstrass, K. (1885). Uber die analytische darstellbarkeit sogenannter willkrlicher functionen einer reellen vernderlichen.