Compression and Approximation of Neural Networks

Hugo Mouton

Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), 31055 Toulouse, FRANCE

Abstract

In this paper, we further the study of the approximation and the compression of neural networks and specifically ReLU-based neural networks. To that end, we first explain what tropical algebra and tropical geometry actually is and in what way these mathematical tools help us in the representation of ReLU-based neural networks. We also present some algorithms of compression based either on Euclidean division in the max-plus semiring either on Hausdorff distance. Afterwards, we introduce the beginning of an extension of the *LazySets* package programmed in Julia allowing the computation of tropical objects and the propagation of polyhedra through a neural network.

Index Terms - Tropical algebra, tropical polynomials, tropical geometry, neural network minimization, computer representation

1 Introduction

Neural networks are used today in a wide range of domains such as speech recognition, computer vision, control of autonomous systems and many others. Nevertheless, our current theoretical understanding of these networks and their efficiency remains incomplete.

One central question is how to verify that neural networks are correct with respect to some criteria. A classic neural network problem is that of *range estimation*, i.e bounding the values of output neurons given the range of values of the input neurons. This is essential to ensure that a neural network is correct and safely usable. In the aeronautical sector, a prototypical application is the one of the verifications of the ACAS Xu - a collision avoidance system for autonomous aircrafts - thoroughly presented in [1].

Another central question is how to compress and simplify a neural network as much as possible, in order to limit its size and the resources required to run it. Interest in pruning or compressing neural networks has also significantly grown in recent years with the appearance of multiple methods allowing one to drastically reduce the size of a neural network without compromising its performance.

Recent works show the connections between tropical geometry and feedforward ReLU-based neural networks, i.e neural newtorks with ReLU activation functions : $x \rightarrow \max(x, 0)$. Tropical geometry has been subject to exponential growth in the recent decade in the study of these neural networks insofar as ReLU functions are tropically linear. We can thus see a ReLU neural network as a tropical rational function ([2]). In the same way reachability analysis on these networks can be done by using tropical polyhedra.

This kind of study requires powerful algorithms to track the evolution of the space of possible values within such a network. The ensemblistic approach can be used to track this evolution, but the sets chosen must be complex enough to lose as little information as possible. The algorithm presented in [3] is one of these high-performance algorithms, based on the use of zonotopes, but for which complexity explodes as a function of the number of neurons, and accuracy decreases in the case of several layers. This calls for the use of compression and approximation methods.

In the following section we introduce basic tropical algebra and tropical geometry which are relevant to us to represent neural networks as tropical objects. We present some approaches of compression in sections 3 and 4. Lastly in section 5 we bring in a computer representation of such objects by extending an existing package that will be presented further in the article. This representation will be used to write algorithms to answer the problem of *range estimation*.

2 A tour of tropical mathematics

Generally speaking, tropical algebra is a tropical equivalent of classical algebra. In this section, we introduce some notations and the general principles of tropical algebra. For further study, we refer the reader to [4].

2.1 Preliminaries

Tropical algebra usually refers to the study of *max-plus semiring* $(\mathbb{R}_{\max}, \oplus, \otimes)$ which is the set $\mathbb{R}_{\max} = \mathbb{R} \cup \{+\infty\}$ equipped with the two operations :

$$x \oplus y = \max(x, y)$$
$$x \otimes y = x + y$$

where $+$ stands for the classical addition operator.

Definition 1 (Semiring)**.** A set $(\mathcal{T}, \oplus, \otimes)$ is a semiring if the following requirements are satisfied :

- $(\mathcal{T}, \oplus)$ is a commutative monoid equipped with a zero element denoted $\mathbb{0}$,
- $(\mathcal{T}, \otimes)$, is a monoid provided with a unit element $\mathbb{1}$,
- The operator $\otimes$ is distributive over $\oplus$,
- $\mathbb{0}$ is an absorbing element for the multiplication.

In the case of the *max-plus semiring*, $\mathbb{0} = -\infty$ and $\mathbb{1} = 0$. Please note that the equivalent with the $\min$ operator, the *min-plus semiring*, exists too.

2.2 Notations

The set $\mathbb{R}^d_{\max}$ denotes the d-fold of $\mathbb{R}_{\max}$. Its element will be denoted by bold symbols like classical vectors. For instance, $\mathbf{x} = (x_1, ..., x_d)$. $\mathbf{0}$ and $\mathbf{1}$ will denote the vectors whose coordinates are all equal to $\mathbb{0}$ and $\mathbb{1}$ respectively.

The two operations $\oplus$ and $\otimes$ are extended to vectors in the sense that :

$$\mathbf{x} \oplus \mathbf{y} = (x_1 \oplus y_1, ..., x_d \oplus y_d)$$
$$\lambda \otimes \mathbf{x} = (\lambda \otimes x_1, ..., \lambda \otimes x_d)$$

From the tropical multiplication comes the tropical power defined as :

$$x^{\otimes a} = x \oplus ... \oplus x = a \cdot x$$

where $\cdot$ stands for the classical product operator.

2.3 Tropical polyhedra

Tropical polyhedra are the equivalent or polyhedra in classical geometry. They are used in reachability analysis of ReLU-based neural networks.

Definition 2 (Tropical halfspace)**.** A *tropical halfspace* is a set of elements $\mathbf{x} \in \mathbb{R}^d_{\max}$ verifying an inequality of the form :

$$\bigoplus_{i \in [\![1,d]\!]} a_i \otimes x_i \leqslant \bigoplus_{i \in [\![1,d]\!]} c_i \otimes x_i$$

where $\mathbf{a},\ \mathbf{c} \in \mathbb{R}^d_{\max}$.

Definition 3 (Tropical affine halfspace)**.** A *tropical affine halfspace* is a set of elements $\mathbf{x} \in \mathbb{R}^d_{\max}$ verifying an inequality of the form :

$$\left(\bigoplus_{i \in [\![1,d]\!]} a_i \otimes x_i\right) \oplus b \leqslant \left(\bigoplus_{i \in [\![1,d]\!]} c_i \otimes x_i\right) \oplus d$$

where $\mathbf{a},\ \mathbf{c} \in \mathbb{R}^d_{\max}$ and $b,\ d \in \mathbb{R}_{\max}$.

These two definitions allow us to describe tropical polyhedra and tropical cones which are a certain type of polyhedra.

Definition 4 (Tropical polyhedron and cone)**.** A *tropical polyhedron* of $\mathbb{R}^d_{\max}$ is defined as a finite intersection of *tropical affine halfspaces* of $\mathbb{R}^d_{\max}$.
A *tropical cone* of $\mathbb{R}^d_{\max}$ is defined as a finite intersection of *tropical halfspaces* of $\mathbb{R}^d_{\max}$. Thus it is a tropical polyhedra with both affine components b and d equal to $\mathbb{0} = -\infty$.

2.4 Tropical polynomials

In tropical algebra, tropical equivalents of polynomials as well as rational functions and rational mappings can also be defined.

Definition 5 (Polynomials)**.** A tropical polynomial p of degree n is a function defined as :

$$\begin{aligned} p(\mathbf{x}) &= \bigoplus_{i=1}^{n} \mathbf{x}^{\otimes \mathbf{a}_i} \otimes b_i \\ &= \bigoplus_{i=1}^{n} \mathbf{a}_i^T \mathbf{x} \otimes b_i \\ &= \max_{i=1}^{n} (\sum_{j=1}^{d} a_{ij} x_j + b_i) \end{aligned}$$

where $(\mathbf{a}_i)_{i \in [\![1,d]\!]}$ are vectors of $\mathbb{R}^d_{\max}$ and $(b_i)_{i \in [\![1,d]\!]}$ are elements of $\mathbb{R}_{\max}$. These polynomials are also reffered to as *signomials* and constitutes the semiring $\mathbb{R}_{\max}[\mathbf{x}]$.

Definition 6 (Rational function)**.** A tropical rational function is a tropical quotient (or a standard difference) of two tropical polynomials f and g :

$$f(\mathbf{x}) - g(\mathbf{x}) = f(\mathbf{x}) \oslash g(\mathbf{x})$$

Definition 7 (Polynomial and rational mapping)**.**

$$\begin{aligned} F : \mathbb{R}^d_{\max} &\to \mathbb{R}^p_{\max} \\ \mathbf{x} = (x_1, ..., x_d) &\mapsto (f_1(\mathbf{x}), ..., f_p(\mathbf{x})) \end{aligned}$$

is called a tropical polynomial map if each f_i for $i \in [\![1,p]\!]$ is a tropical polynomial, and a tropical rational map if each f_i is a tropical rational function.

Such functions can be used to represent neural networks. For everything that follows, we consider the ReLU activated neural network given by Figure 1.

It consists of an *input layer* $\mathbf{x} = (x_1, ..., x_d)$, a *hidden layer* $\mathbf{f} = (f_1, ..., f_n)$ with ReLU activations functions, and an *output layer* $\mathbf{y} = (y_1, ..., y_m)$. These two layers are defined by the matrices of weights $A = (a_{ij})_{i \in [\![1,n]\!], j \in [\![1,d]\!]}$ and $C = (c_{ki})_{i \in [\![1,n]\!], k \in [\![1,m]\!]}$ respectively. The hidden layer also consists of a bias matrix $B = (b_i)_{i \in [\![1,n]\!]}$.

The i-th node of the hidden layer f is thus computed as :

$$f_i(\mathbf{x}) = \max(\mathbf{a}_i \mathbf{x} + b_i, 0) \tag{1}$$

And so, f_i is a tropical polynomial composed of two terms. It follows that $ENewt(f_i)$ is a linear segment in $\mathbb{R}^{d+1}_{\max}$.

Therefore the components of the ouput layer are computed as :

$$\begin{aligned} v_k(\mathbf{x}) &= \sum_{i=1}^{n} c_{ki} f_i(\mathbf{x}) && (2) \\ &= \sum_{c_{ki}>0} |c_{ki}| f_i(\mathbf{x}) - \sum_{c_{ki}<0} |c_{ki}| f_i(\mathbf{x}) && (3) \\ &= p_k(\mathbf{x}) - q_k(\mathbf{x}) && (4) \end{aligned}$$

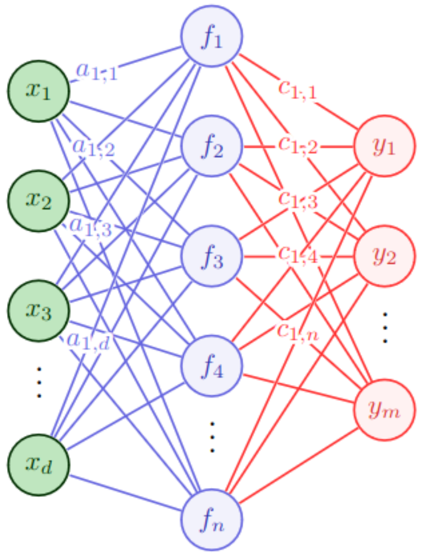


FIGURE 1 – Neural network with one hidden ReLU layer. The first linear layer has weights $\{\mathbf{a}_i^T\}$ with bias $\{b_i\}$ (included in the nodes f_i). The second layer has weights c_{ki}. $\forall i \in [\![1, n]\!], k \in [\![1, m]\!]$

As functions p_k and q_k are both linear positive combinations of (f_i) they are tropical polynomials, and so (v_k) are tropical rational functions. We will not go into further detail on the equivalence between tropical rational mappings and feedforward ReLU-based neural networks. We refer the reader to [2] for more information. For the purpose of our work, we will keep in mind the theorem resulting from the cited article :

Theorem 1 (Zhang et al., 2018)**.** *A ReLU activated deep neural network* $F : \mathbb{R}^d \to \mathbb{R}^p$ *is equivalent to a tropical rational mapping.*

2.5 Geometrical representation of polynomials

Analogues of many notions in classical geometry are numerous ([2] [4]). Polytopes and zonotopes are among the most studied and the most used as geometric tools for linear programming and optimization in fields like the one of neural networks. For instance, in [2], it has been shown that linear regions of neural networks correspond to vertices of certain polytopes. In this last subsection, we explain how tropical polynomials can be seen as tropical geometrical objects.

Consider a tropical polynomial $p(\mathbf{x}) = \max_{i=1}^{n}(\mathbf{a}_i^T\mathbf{x} + b_i)$. The *Newton Polytope* and the *Extended Newton Polytope* allow a geometrical description of this latter.

Definition 8 (Newton Polytope)**.** Let $p(\mathbf{x}) = \max_{i=1}^{k}(\mathbf{a}_i^T\mathbf{x} + b_i), \mathbf{x} \in \mathbb{R}^d$ be a tropical polynomial. The Newton polytope of p is defined as : $Newt(p) = co(\mathbf{a}_i, i \in [\![1, k]\!])$ where $co(S)$ denotess the convex hull of a given set S.

Definition 9 (Extended Newton Polytope)**.** With the same polynomial as above, the extended Newton polytope of p is defined as : $ENewt(p) = co((\mathbf{a}_i, b_i), i \in [\![1, k]\!])$.

As an example, let $p(x) = \max(3x, 2x + 1.5, x + 1, 0)$, we therefore have : $Newt(p) = co(0, 1, 2, 3) = [0, 3]$ and also $ENewt(p) = co((0, 0), (1, 1), (2, 1.5), (3, 0))$. These two polytopes are represented in Figure 2.

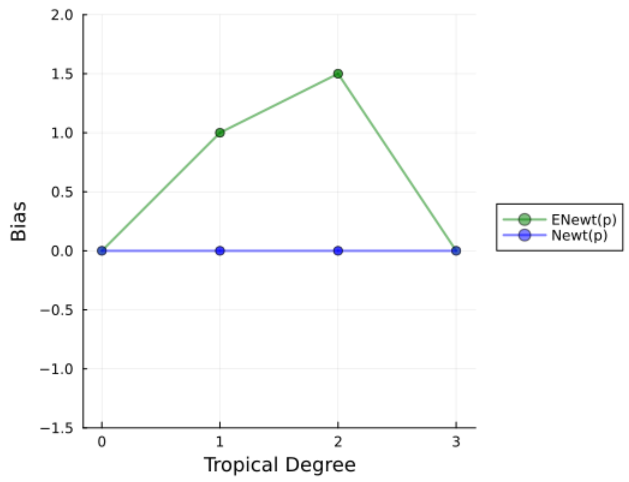


FIGURE 2 – $Newt(p)$ and $ENewt(p)$ with $p(x) = \max(3x, 2x + 1.5, x + 1, 0)$.

Such a representation of tropical polynomials makes the expression of the mathematical structure of ReLU activated neural networks easier and enables the development of neural network compression methods which will be presented in the next sections.

3 Euclidean Division

This topic has been largely introduced by (Smyrnis et al., 2020). First, we recall the algorithm developped in [5]. Next, we use these results to propose a potential solution to the neural network approximation problem.

3.1 Maxpolynomial division algorithm

Let $p(\mathbf{x}) = \max_{i=1}^{n}(\mathbf{a}_i^T\mathbf{x} + b_i)$ and $d(\mathbf{x}) = \max_{i=1}^{m}(\mathbf{c}_i^T\mathbf{x} + d_i)$ be two tropical polynomial functions. The following algorithm outputs two tropical polynomial functions $q(\mathbf{x})$ and $r(\mathbf{x})$ (which can be interpreted as the *quotient* and the *remainder* of the division of $p(\mathbf{x})$ by $d(\mathbf{x})$) for which :

$$p(\mathbf{x}) \geqslant \max(q(\mathbf{x}) + d(\mathbf{x}), r(\mathbf{x})), \forall \mathbf{x} \in \mathbb{R}^d \tag{5}$$

The maxpolynomial division algorithm ([5]). The algorithm dividing p by d is given by :

1. Let $C \subseteq \mathbb{Z}^d$ be the set of slopes $\mathbf{c}$ with which we can shift $Newt(d)$, so that $Newt(\mathbf{c}^T\mathbf{x} + d(\mathbf{x})) \subseteq Newt(p(\mathbf{x}))$.
2. For all $\mathbf{c} \in C$, let's define $q_c \in \mathbb{R}$ as the largest value q for which $ENewt(p)$ is higher than $ENewt(q + \mathbf{c}^T\mathbf{x} + d(\mathbf{x}))$, i.e $p(\mathbf{x}) \geqslant q + \mathbf{c}^T\mathbf{x} + d(\mathbf{x})$.

3. Output $q(\mathbf{x}) = \max_{c \in C}(q_c + \mathbf{c}^T\mathbf{x})$ and $r(\mathbf{x}) = \max_{t \in T}(t)$, where $T = \{\mathbf{a}_j^T\mathbf{x} + b_j \mid \nexists \mathbf{c},\ \mathbf{a}_j \in Newt(\mathbf{c}^T\mathbf{x} + d(\mathbf{x}))\}$.

Besides, we have the following theorem :

Theorem 2. *The resulting couple (q, r) is maximal in the sense that for any other couple $(\tilde{q}, \tilde{r})$ for which 5 holds and containing terms of the same degree than q and r respectively, we have :*

$$\max(q(\boldsymbol{x}) + d(\boldsymbol{x}), r(\boldsymbol{x})) \geqslant \max(\tilde{q}(\boldsymbol{x}) + d(\boldsymbol{x}), \tilde{r}(\boldsymbol{x})) \qquad (6)$$

To illustrate how the algorithm works, two examples are described and depicted below.

Example 1. (Figure 3) For this example and the next one we will work in $\mathbb{R}^2$ and let $p(x) = \max(3x, 2x+1.5, x+1, 0), x \in \mathbb{R}$. Let $d(x) = \max(x+1, 0)$.

1. The set of valid slopes is $C = 0, 1, 2$.
2. For $c = 1$, we see that $ENewt(d(x) + c^T x)$ can be translated upwards by up to $q_1 = 0.5$. In this case, the right vertex of $ENewt(d(x) + c^T x + q_c)$ coincides with the third vertex of $ENewt(p)$.
3. By now, $q(x) = \max(2x - 1, x + 0.5, 0)$ and there is no remainder because $T = \varnothing$. Indeed, there is no tropical monomial $(a_i x + b_i)$ of p such as $a_i \notin \bigcup_{c \in C} Newt(d(x) + c^T x)$.

In this example, there is no gap between $ENewt(q + d)$ and $ENewt(p)$ and we could show by developing that $p(x) = q(x) + d(x)$. Furthermore, if we modify the first term of p so that now $p(x) = \max(3.5x, 2x + 1.5, x + 1, 0)$, 3e shows that there is a non-zero remainder $r(x) = max(3.5x) = 3.5x$ as $T = \{3.5x\}$.

Example 2. (Figure 4) Now, $d(x) = \max(x, 0)$ and $p(x) = \max(3x, 2x + 1.5, x + 1, 0)$. The method and what is derived from the results do not change except that now there is a gap between $ENewt(q + d)$ and $ENewt(p)$, and $p(x) > q(x) + d(x)$.

3.2 Application to approximation of neural networks

As it has been shown in [2] and using theorem 1 that a neural network can be represented as a tropical rational mapping. Consider the case of the output layer composed of only one node, let P and D be the two tropical polynomials such as the neural network is equivalent to the following rational mapping :

$$\mathbf{x} \to \frac{P(\mathbf{x})}{D(\mathbf{x})}, \mathbf{x} \in \mathbb{R}^d_{\max}$$

The tropical quotient of two polynomials corresponds to the classic subtraction between these two polynomials. The objective is to find two tropical polynomials Q and $\tilde{Q}$ such as :

$$-\tilde{Q} \geqslant \frac{P}{D} \geqslant Q$$

The left-hand side of the equation may be obtained by an adaptation of the above algorithm to get an over-approximation, there will not be any further development for this part because of the following. As for the right-hand side, we can try to use the algorithm by authorizing real translations, i.e $C \subseteq \mathbb{R}^d$. Looking at last example in the previous section, we can hope there will be no remainder.

However, we can easily find a counter-example. Let us consider the following system :

$$\begin{aligned} y_1 &= max(x_1 - x_2 - 1, 0) \\ y_2 &= max(x_1 + x_2 + 1, 0) \\ c &= y_2 - y_1 - 1 \end{aligned}$$

This example is adapted from [3]. To approximate c we use the maxpolynomial division algorithm dividing y_2 by y_1 which is illustrated by Figure 5. The fact that we can do real translations is not very useful after all, because we still get a remainder.

The Figure 5 shows that there is no translation c such that $Newt(c^T\mathbf{x} + y_1) \subseteq Newt(y_2)$. So there will be a remainder but no correctly defined quotient q such as :

$$y_2 - y_1 \geqslant q$$

This attempt of such an approximation has not been explored further.

4 Compression using Hausdorff distance

In this section, we present a way to compress neural networks. First, we give another representation of neural networks which is mainly based on the results of the Subsection 2.3. Then, we introduce the notion of Hausdorff distance and how it is used in the compression. Finally, two algorithms will be presented together with a description of their potential implementation.

4.1 Geometrical Representation

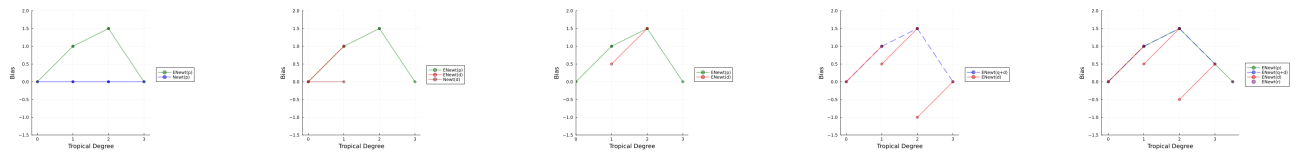
For what follows, let's introduce the notion of *Zonotopes* :

Definition 10 (Zonotopes)**.** *Zonotopes* are defined as the Minkowski sum of a finite set of line segments.

Definition 11 (Minkowski sum)**.** The Minkowski sum of two sets A and B is defined as

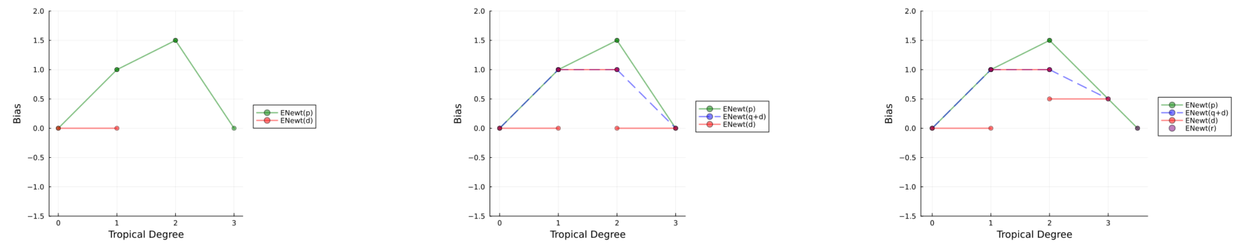
$$A \pm B = \{\mathbf{a} + \mathbf{b} | \ \mathbf{a} \in A,\ \mathbf{b} \in B\}$$

We already know from Subsection 2.3 and the Theorem 1 that neural networks as the one depicted by the Figure 1 are equivalent to tropical rational mappings. So let's go back to the notations we have used before by using the neural network of the Figure 1 as example and by denoting by p_k and q_k the two tropical polynomials such as the k-th output of the neural network is $v_k = p_k - q_k$.



(a) (b) (c) (d) (e)

FIGURE 3 – (a) : $Newt(p)$ and $ENewt(p)$ with $p(x) = \max(3x, 2x+1.5, x+1, 0)$ and $d(x) = \max(x+1, 0)$. (b) : $ENewt(p)$, $ENewt(d)$ and $Newt(d)$. (c),(d) : Translation of $ENewt(d)$. (d) : $ENewt(q+d)$. (e) : Same plots but with a non-zero remainder using $p(x) = \max(3.5x, 2x+1.5, x+1, 0)$.



(a) (b) (c)

FIGURE 4 – (a) : $Newt(p)$ and $ENewt(d)$ with $p(x) = \max(3x, 2x+1.5, x+1, 0)$ and $d(x) = \max(x, 0)$. (b) : $ENewt(p)$, $ENewt(d)$ and $ENewt(q+d)$. (c) : Same plots but with a non-zero remainder using $p(x) = \max(3.5x, 2x+1.5, x+1, 0)$.

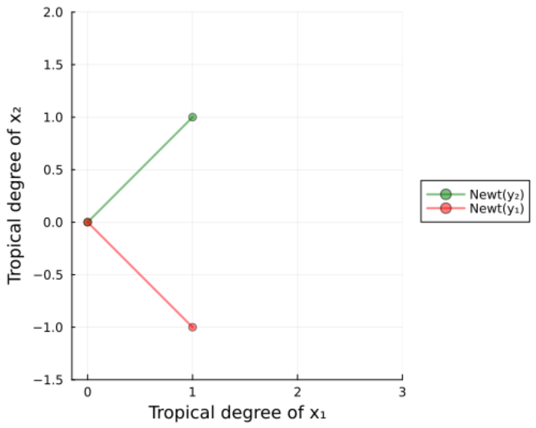


FIGURE 5 – Extended Polytopes of Newton of y_1 and y_2

p_k and q_k are both positive combinations of terms of the form :

$$f_i = \max(\mathbf{a}_i\mathbf{x} + b_i, 0),\ \mathbf{x} \in \mathbb{R}^d_{\max}$$

whose *Extended Newton Polytopes* are linear segment in $\mathbb{R}^{d+1}_{\max}$. As a result, p_k and q_k are written as linear combinations of tropical polynomials that correspond to linear segments. Their *Extended Newton Polytope*, P_k and Q_k respectively, are therefore zonotopes. In view of how v_k is written, we choose to refer to P_k as the *positivie* polytope and to Q_k as the negative one.

These two polytopes are thus generated by the line segments $(ENewt(f_i))_{i\in[\![1,n]\!]}$ and each vertex of these zonotopes can be computed as the sum of vertices of these line segments, either $\mathbf{0}$ or $c_{ki}(\mathbf{a}_i^T, b_i)$.

4.2 Approximation with Hausdorff distance

The computation of the upper bound on the compression error as well as the algorithms presented in the next section have been recently developed. We therefore refer the reader to the complete article of Misiakos et al. (2022) [6].

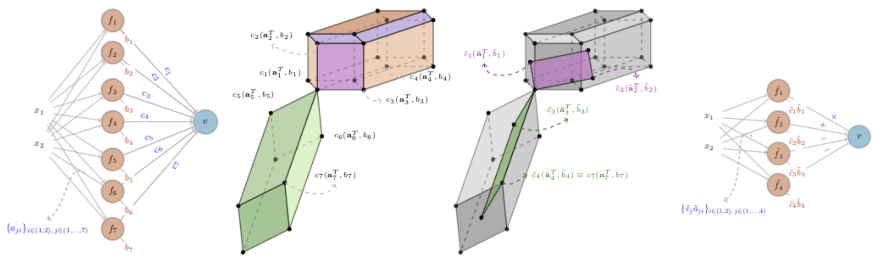
The extended Newton polytope provides a geometrical representation of a tropical polynomial. In addition, it may be used to compute the values that the polynomial attains.

Property 3 (Charisopoulos & Maragos, 2018)**.** *Let f be a tropical polynomial in d variables. Let $UF(ENewt(f))$ be the points in the upper face of $ENewt(p)$, where upward direction is taken with respect to the last dimension of $\mathbb{R}^{d+1}$. Then for each $i \in [\![1, n]\!]$ there exists a linear region of f on which $f(\boldsymbol{x}) = \boldsymbol{a}_i^T\boldsymbol{x} + b_i$ if and only if $(\boldsymbol{a}_i^T, b_i)$ is a vertex of $UF(ENewt(f))$.*

Intuitively, the knowledge of the upper face of the extended Newton polytope of a given tropical polynomial determines the behavior of the polynomial on its whole space of definition.

The Property 3 indicates that a polynomial's values are determined at each point of the input space by the vertices of the upper face of its extended Newton polytope. Therefore, it is expected that two tropical polynomials with approximatively the same extended Newton polytope should attain similar values. This serves as intuition for the following. We use the Hausdorff distance as our metric to compare two extended Newton polytopes.

Definition 12 (Hausdorff distance)**.** Let $\mathbf{u} \in \mathbb{R}^d$ and a finite set $V \subset \mathbb{R}^d$. The distance of $\mathbf{u}$ from V is denoted as dist$(\mathbf{u},\ V)$ =



(a) Original network. (b) Original zonotopes. (c) Resulting zonotopes. (d) Compressed network.

FIGURE 6 – Illustration of Zonotope K-means execution. The original zonotope P is generated by $c_i(\mathbf{a}_i^T, b_i)$ for $i = 1, ..., 4$ and the negative zonotope Q generated by the remaining ones $i = 5, 6, 7$. The approximation $\tilde{P}$ of P is generated by $\tilde{c}_i(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$ for $i = 1, 2$ where the first generator is the K-means center representing the generators 1 and 2 of P and the second is the representative center of 3 and 4. Similarly, the approximation $\tilde{Q}$ of Q is generated by $\tilde{c}_i(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$ for $i = 3, 4$ that stand as representative centers for $\{5, 6\}$ and 7 respectively.

dist(V, $\mathbf{u}$) and computed as $\min_{\mathbf{v} \in V} ||\mathbf{u}, \mathbf{v}||$ where $|| \cdot ||$ is the Euclidean norm.

The Hausdorff distance between two finite subsets U, V of $\mathbb{R}^d$ is defined as

$$\mathcal{H}(U, V) = \max\{\max_{\mathbf{v} \in V} \text{dist}(\mathbf{v}, U), \max_{\mathbf{u} \in U} \text{dist}(V, \mathbf{u})\}$$

Thus, let P and $\tilde{P}$ be two polytopes with their associated vertex sets denotes by V_P and $V_{\tilde{P}}$ respectively. The Hausdorff distance of two polytopes is so the Hausdorff distance of their associated vertex sets :

$$\mathcal{H}(P, \tilde{P}) = \mathcal{H}(V_P, V_{\tilde{P}})$$

It is clear that the Hausdorff distance is a metric of how close two polytopes are to each other which becomes zero when the two polytopes coincide. Using this metric, the following bound on the error of tropical polynomial approximation is derived :

Property 4. *Let p and $\tilde{p}$ be two tropical polynomials and let P and $\tilde{P}$ their extended Newton polytope respectively. Then,*

$$\max_{\boldsymbol{x} \in \mathcal{B}} |p(\boldsymbol{x}) - \tilde{p}(\boldsymbol{x})| \leqslant \rho \cdot \mathcal{H}(P, \tilde{P})$$

where $\mathcal{B} = \{\boldsymbol{x} \in \mathbb{R}^d : ||\boldsymbol{x}|| \leqslant r\}$ is the hypersphere of radius r, and $\rho = \sqrt{r^2 + 1}$.

This proposition enables us to handle the case of neural networks with the same architecture as the one described in the Figure 1. By repeatedly applying Property 4 to each tropical polynomial corresponding to the network, we get :

Theorem 5 (Misiakos et al., 2022 [6]). *Let v and $\tilde{v}$ be two neural networks with the same architecture as the one in Figure 1. With $\tilde{P}_k$ and $\tilde{Q}_k$, we denote the positive and negative zonotopes of $\tilde{v}$. The following bound holds :*

$$\max_{\boldsymbol{x} \in \mathcal{B}} ||v(\boldsymbol{x}) - \tilde{v}(\boldsymbol{x})|| \leqslant \rho \cdot \left(\sum_{k=1}^{m} \mathcal{H}(P_k, \tilde{P}_k) + \mathcal{H}(Q_k, \tilde{Q}_k) \right)$$

4.3 Compression Algorithms

The knowledge of the upper bound on the approximation error enables us to derive compression algorithms for ReLU activated neural networks. For now, we continue to use the neural network architecture of Figure 1 and we suppose that we want to compress it by reducing the number of neurons in the hidden layer from n to K. Let us denote the compressed neural network by $\tilde{v} = (\tilde{v}_1, ..., \tilde{v}_m)$ where each output may be written as $\tilde{v}_k = \tilde{p}_k - \tilde{q}_k$ and the zonotopes of $\tilde{p}_k$ and $\tilde{q}_k$ are generated by $\tilde{c}_{ki}(\tilde{a}_i^T, \tilde{b}_i)$, $\forall i$. These generators need to be chosen in such a way that $\tilde{v}_k(\mathbf{x}) \approx v_k(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{B}$. From Theorem 5, it is equivalent to finding generators such as the zonotopes P_k, $\tilde{P}_k$, Q_k and $\tilde{Q}_k$ make $\mathcal{H}(P_k, \tilde{P}_k)$ and $\mathcal{H}(Q_k, \tilde{Q}_k)$ as small as possible for all k.

In the following algorithms, generators of compressed zonotopes are derived from those of the original zonotopes. In our case, K-means algorithm has be chosen to manipulate the $c_{ki}(a_i^T, b_i)$ easily and obtain compressed generators not to far from them.

4.3.1 Zonotope K-means

The first compression algorithm can only be used in the case of neural networks with a single output neuron as the one depicted on the Figure 6. It reduces the number of neurons in the hidden layer from n to K. We keep our usual notations by denoting by c_i the weights of the second layer.

4.3.2 Neural Path K-means

The problem of the above algorithm is that it can only be used in single output neural networks. Indeed, in a multi-layers neural network, we cannot think about an iterative use of this algorithm layer by layer. To overcome this obstacle, the idea is to work with every weights connected to each neuron. Thus the algorithm applies K-means to the vectors $(\mathbf{a}_i^T, b_i, c_{1i}, ..., c_{mi})$. The procedure is described in the Algorithm 5.

However, we didn't have enough time to write a first implementation of these algorithms. The next section details what we were missing.

```
Algorithm 1: Zonotope K-means Compression Algorithm
  1. Split original generators into two sets
     $\{c_i(\mathbf{a}_i^T, b_i) \mid c_i > 0\}$ and $\{c_i(\mathbf{a}_i^T, b_i) \mid c_i < 0\}$.
  2. Apply K-means algorithm for $\frac{K}{2}$ centers for both sets and
     obtain $\{\tilde{c}_i(\tilde{\mathbf{a}}_i^T, \tilde{b}_i) \mid \tilde{c}_i > 0\}$ and $\{\tilde{c}_i(\tilde{\mathbf{a}}_i^T, \tilde{b}_i) \mid \tilde{c}_i < 0\}$ as
     output.
  3. Construct the final weights. For the first linear layer, the
     weights and the bias which correspond to the $i$-th neuron
     become the vector $\tilde{c}_i(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$.
  4. The weights of the second linear layer are set to $1$ or $-1$
     depending on whether the associated hidden neuron
     generates the positive or negative zonotope.
```

```
Algorithm 2: Neural Path K-means Compression Algorithm
  1. Apply K-means for K centers to the vectors
     $(\mathbf{a}_i^T, b_i, C_{:,i}^T),\ i \in [\![1, n]\!]$ and get the centers
     $(\tilde{\mathbf{a}}_i^T, \tilde{b}_i, \tilde{C}_{:,i}^T),\ i \in [\![1, K]\!]$.
  2. Construct the final weights. For the first linear layer,
     weights and bias becomes $(\tilde{\mathbf{a}}_i^T, \tilde{b}_i)$, while for the second
     layer, weights becomes $\tilde{C}_{:,i}$.
```

5 Computer representation

The Julia programming language is a dynamic language, powerful for scientific and numerical computing, with performance comparable to traditional statically-typed languages. Numerous packages are available and well developed to perform neural networks computations. *LazySets* is one of them, it allows to make calculations on convex sets and it is widely used in neural networks reachability studies. However it is not suitable for tropical polyhedra which are not convex in the classical sense. The development of such a package would allow a simplified study of ReLU neural networks and the implementation of algorithms as presented in [3].

The objective of this section is to propose and present the beginnings of an implementation of such polyhedra. We will first expose how these polyhedra will be represented and then we explain algorithms used to manipulate such sets.

5.1 Data Structure

We already know from Section 2 that a tropical polyhedra is defined as a finte intersection of tropical affine halfspaces. Thus it is a set of the form :

$$\{\mathbf{x} \in \mathbb{R}_{\max}^d \mid A \otimes \mathbf{x} \oplus \mathbf{b} \leqslant C \otimes \mathbf{x} \oplus \mathbf{d}\}$$

where $A,\ C \in \mathbb{R}_{\max}^{r\times d}$, $\mathbf{b},\ \mathbf{d} \in \mathbb{R}_{\max}^r$ and $r \geqslant 0$ is the number of constraints.

Remark. The matrix product is perform in the same way as the classical one by replacing the classical operators by their tropical equivalent.

It therefore makes sense to choose a data structure based on four matrices or should we say vectors of vectors for algorithmic convenience.

```
struct TropicalPolyhedron{T<:Real}
    A::Vector{Vector{T}}
    B::Vector{T}
    C::Vector{Vector{T}}
    D::Vector{T}
end
```

As the class is generic, we can work with polyhedra defined by both integer and real constraints. We will not dwell on auxiliary functions (constructors, getters, etc.).

5.2 Is a polyhedron empty ?

The first thing we need to know when handling tropical polyhedra is whether or not they are empty. A lot of work has already been done on this subject in [7] and we will just remind what is essential for our implementation. For further details, we refer the reader to the latter reference.

5.2.1 Equivalence with mean payoff games

Let $\mathcal{P}$ be a tropical polyhedron defined by real constraints :

$$A \otimes \mathbf{x} \oplus \mathbf{b} \leqslant C \otimes \mathbf{x} \oplus \mathbf{d}$$

We can associate $\mathcal{P}$ and a mean payoff games between two players *Max* and *Min*, the first playing on *row nodes* $i \in [\![1, r]\!]$ and the second on *column nodes* $j \in [\![1, d+1]\!]$. The digraph contains an arc from :

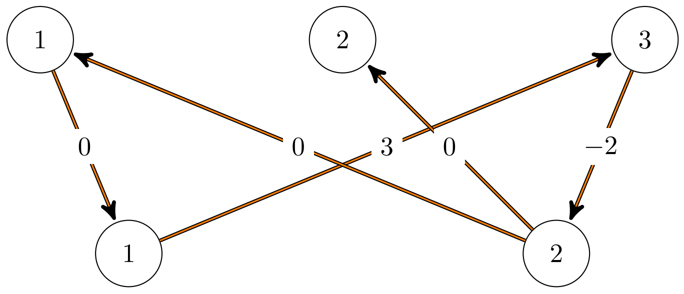
1. i to $j \leqslant d$ with weight C_{ij} when $C_{ij} \neq -\infty$.
2. $j \leqslant d$ to i with weight $-A_{ij}$ when $A_{ij} \neq -\infty$
3. i to $j = d+1$ with weight d_i when $d_i \neq -\infty$
4. $j = d+1$ to i with weight $-b_i$ when $b_i \neq -\infty$

When the equivalent graph is drawn, *row nodes* are at the bottom, *column nodes* are at the top.

Example Let $\mathcal{P}$ be a tropical polyhedron defined by the following system :

$$x \leqslant 3$$
$$2 \leqslant x + y$$

The equivalent graph is :



5.2.2 Empty criterion

Considering infinite runs of the game, let v be the value of the game associated with the graph when it starts at column $d+1$.

Property 6 (Allamigeon et al., 2014 [7]). *The tropical polyhedron $\mathcal{P}$ is non-empty if and only if $v > 0$ for any run.*

We have to consider all possible runs, since there may be several outgoing arcs from a node. For instance on the above example, there are two outgoing arcs with the same weight from the second row node.

In addition, there is necessarily at least one outgoing node for row nodes, but not necessarily for column nodes. In this case, the asymptotic gain will be considered positive.

Obviously, we cannot consider infinite runs on a computer, so we will try to obtain a cycle. Once this cycle has been obtained, all we need to do is look at the sign of the asymptotic gain of the game. The algorithm is described below :

Algorithm 3: Emptiness test for a polyhedron

```
1. Generate the graph of the equivalent mean payoff game.
2. Start a first game from column node $d+1$.
3. At each node of the game, if several outgoing arcs are
   possible, recursively start as many games as there are
   outgoing arcs. If there are no outgoing nodes, consider
   the asymptotic gain to be positive for the current run.
4. If for any game the asymptotic gain is negative, stop the
   algorithm, the polyhedron is empty.
5. Otherwise, the polyhedron is non-empty.
```

Example Let us take the example given above. We start from column node 3. We can only go to row node 2, at which there are two possible outgoing nodes, column nodes 1 and 2. First we continue the game towards column node 2, where we are blocked, so the gain is positive for this first game. We now return to the row node 2, this time heading for the column node 1. We leave the reader the pleasure of continuing the game, but for the rest there is only one possible outgoing arc at each node leading us back to column node 3. If we add up the weights of the edges taken, we obtain a gain $v = 1$ for this game, which is therefore positive. The polyhedron defined above is therefore non-empty, which is indeed the case given its definition.

5.3 Redundancy of constraints

Another important feature is the ability to check whether a constraint is redundant with respect to a polyhedron. This is particularly useful when dealing with the intersections of tropical polyhedra. The objective is to have a decision procedure for implications of the form :

$$A \otimes \mathbf{x} \oplus \mathbf{b} \leqslant C \otimes \mathbf{x} \oplus \mathbf{d} \implies \mathbf{e} \otimes \mathbf{x} \oplus g \leqslant \mathbf{f} \otimes \mathbf{x} \oplus h$$

where $\mathbf{e},\ \mathbf{f} \in \mathbb{R}^d_{\max}$ and $g,\ h \in \mathbb{R}_{\max}$.

Work on this decision procedure has already been produced and explained in [7]. We will exploit the Propostion 19 which we will adapt to our case.

Property 7. *Let $\mathcal{Q}$ be the tropical polyhedron defined by the system $A \otimes \boldsymbol{x} \oplus \boldsymbol{b} \leqslant C \otimes \boldsymbol{x} \oplus \boldsymbol{d}$ and the following constraints :*

$$\begin{aligned} f_i x_i &\leqslant (\boldsymbol{e} - \varepsilon) \otimes \boldsymbol{x} \oplus (g - \varepsilon) \text{ if } f_i \in \mathbb{R}_{\max} \\ x_i &\leqslant -\infty \text{ if } f_i = +\infty \\ h &\leqslant (\boldsymbol{e} - \varepsilon) \otimes \boldsymbol{x} \oplus (g - \varepsilon) \text{ if } h \in \mathbb{R}_{\max} \end{aligned}$$

for all $i \in [\![1, n]\!]$. Then, the above implication holds if and only if $\mathcal{Q}$ is empty.

The epsilon must be chosen small enough for the algorithm to work properly. However, we have not had the time to find a criterion for this factor to ensure proper operation. In the current implementation, it is fixed to 1.10^{-9} and the results seem conclusive for constraints with real coefficients. In the case of integer coefficients, we would have to take $\varepsilon = 1$.

Starting from this property and using the results of the previous section, we can write the following algorithm to conclude whether or not a constraint is redundant with respect to a certain tropical polyhedron.

Algorithm 4: Redundancy of a constraint

```
Input : a polyhedron $\mathcal{P}$, a constraint $\mathcal{C}$
1. Make an independent copy $\mathcal{Q}$ of $\mathcal{P}$.
2. Based on the coefficients of $\mathcal{C}$, add the new constraints
   defined in Property 7 to $\mathcal{Q}$.
3. Use the Algorithm 3 on $\mathcal{Q}$.
```

5.4 Intersection of tropical polyhedra

Since we can now know whether a constraint is redundant with respect to a certain polyhedron, we can write the intersection of two tropical polyhedra. Indeed, each polyhedron is defined by a system of constraints, so the intersection of two polyhedra corresponds to the union of the associated systems of constraints.

Thanks to the chosen data structure, it is easy enough to compute the intersection of two given polyhedra. The use of Algorithm 4 saves memory space by eliminating redundant constraints.

Algorithm 5: Intersection of polyhedra

```
Input : two polyhedra $\mathcal{P}$ and $\mathcal{Q}$
1. Make an independent copy $\mathcal{Z}$ of $\mathcal{P}$ (not mandatory, it
   allows $\mathcal{P}$ to remain unchanged).
2. For each constraint of $\mathcal{Q}$, use the Algorithm 4. If it is not
   redundant, add the constraint to $\mathcal{Z}$.
```

5.5 Calculation of the enclosing hypercube

In this subsection, we present the first work done on the computational calculation of the encompassing hypercube for a certain polyhedron. This calculation is quite recurrent in reachability studies of ReLU networks.

Property 2 of article [3] gives us a way of calculating the enclosing "zone". We will not go into details here, but will adapt this property to the calculation of the enclosing hypercube.

Property 8 (Enclosing hypercube)**.** *Let M be the matrix of the g generators for a certain tropical polyhedron $\mathcal{P}$ stripped out of rows consisting only of $-\infty$ entries, and M/M the residuated matrix which entries are $(M/M)_{ij} = \min_{1\leqslant k\leqslant g} a_{ik} - a_{jk}$. Then the smallest enclosing hypercube $\mathcal{H}$ is given by the system :*

$$x_i - x_j \geqslant (M/M)_{ij} \ \ \forall i, j \in [\![1, d]\!]$$

At present, generators have only been calculated for tropical cones using the algorithm developed and widely explained in [8]. Further work would enable us to calculate the generators of tropical polyhedra, and thus their enclosing hypercube. Reachability algorithms could then be written more easily.

5.6 Experiment results

We will not comment the results of these early implementations. The first experiments seem conclusive, confirming that this data structure and the associated algorithms work well. However, no larger-scale tests have yet been carried out on larger neural networks. As explained in the previous section, more in-depth work would enable us to implement complete algorithms for reachability studies as presented in [3]. Software such as *PolyMake* could then be used to bring a graphical interface to these studies.

The GitHub gives access to the current code and the various test files. How the various functions work and how to use them are described in the README.

References

[1] Kyle Julian, Mykel J.Kochenderfer, and Michael P.Owen. Deep neural networks compression for aircraft collision avoidance systems. *AIAA Journal of Guidance, Control, and Dynamics*, 2018.

[2] Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. *CoRR*, abs/1805.07091, 2018.

[3] Eric Goubault, Sébastien Palumby, Sylvie Putot, Louis Rustenholz, and Sriram Sankaranarayanan. Static analysis of relu neural networks with tropical polyhedra. In *Static Analysis - 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings*, pages 166–190, 2021.

[4] Xavier Allamigeon. Introduction to tropical convexity. *Static analysis of memory manipulations by abstract interpretation*, pages 25–42, 2009.

[5] Georgios Smyrnis, Petros Maragos, and George Retsinas. Maxpolynomial division with application to neural network simplification. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4192–4196, 2020.

[6] Panagiotis Misiakos, Georgios Smyrnis, George Retsinas, and Petros Maragos. Neural network approximation based on hausdorff distance of tropical zonotopes. In *International Conference on Learning Representations*, 2022.

[7] Xavier Allamigeon, Uli Fahrenberg, Stéphane Gaubert, Ricardo D.Katz, and Axel Legay. Tropical fourier-motzkin elimination with an application to real-time verification. pages 15–23, 2014.

[8] Xavier Allamigeon, Stéphane Gaubert, and Éric Goubault. The tropical double description method. *Symposium on Theoretical Aspects of Computer Science*, 2010.