Liu WW, Song F, Zhang THR *et al.* Verifying ReLU neural networks from a model checking perspective. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 35(6): 1365–1381 Nov. 2020. DOI 10.1007/s11390-020-0546-7

Verifying ReLU Neural Networks from a Model Checking Perspective

Wan-Wei Liu[1], *Senior Member, CCF*, Fu Song[2,3], *Senior Member, CCF*, Tang-Hao-Ran Zhang[1], and Ji Wang[1,4], *Distinguished Member, CCF*

[1] *College of Computer Science, National University of Defense Technology, Changsha 410073, China*
[2] *School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China*
[3] *Shanghai Engineering Research Center of Intelligent Vision and Imaging, Shanghai 201210, China*
[4] *State Key Laboratory for High Performance Computing, Changsha 410073, China*

E-mail: wwliu@nudt.edu.cn; songfu@shanghaitech.edu.cn; {zhangthr, Jiwang}@nudt.edu.cn

Received April 12, 2020; revised October 6, 2020.

Abstract Neural networks, as an important computing model, have a wide application in artificial intelligence (AI) domain. From the perspective of computer science, such a computing model requires a formal description of its behaviors, particularly the relation between input and output. In addition, such specifications ought to be verified automatically. ReLU (rectified linear unit) neural networks are intensively used in practice. In this paper, we present ReLU Temporal Logic (ReTL), whose semantics is defined with respect to ReLU neural networks, which could specify value-related properties about the network. We show that the model checking algorithm for the $\Sigma_2 \cup \Pi_2$ fragment of ReTL, which can express properties such as output reachability, is decidable in EXPSPACE. We have also implemented our algorithm with a prototype tool, and experimental results demonstrate the feasibility of the presented model checking approach.

Keywords model checking, rectified linear unit neural (ReLU) network, temporal logic

1 Introduction

Neural network (NN, for short) [1], is considered to be an extremely important computing model in artificial intelligence domain. It has attracted more and more attention in recent years. It acts as a classifier, which takes an input from the feature space, and tells the corresponding class to which the input belongs.

Though such kind of computing models has been developed for decades, and numerous variants have been presented (e.g., recurrent neural networks [2] and convolutional neural networks [3]), for the most cases, neural networks are working as black-boxes. This makes the behavior of an NN extremely difficult to predict, and therefore one can barely give a rigorous description of such a network.

A neural network mimics what the human's brain works in reality, and its structure is designed in a bionic way — it consists of a series of perceptrons (or neurons), and each perceptron has several inputs (dendrites) and one output (axon). The approach to modeling a perceptron is to consider it as a function from inputs to the output, and such a function is composed of a linear part and a nonlinear part. The linear part is a weighted summation of the inputs, whereas the nonlinear part is in general an activation function. Then, the perceptrons form a network via interconnection. In general, to simplify the computing model, we organize these perceptrons into layers, and usually a connection only happens between two adjacent layers.

One can employ various activation functions when designing NNs, for example, the sigmoid function, the tanh function. Among these, networks using ReLU activation function [4] are so attractive to mathematicians and computer scientists. The reason might be that this kind of functions is most likely to be linear — we call it semi-linear, or piecewise linear [5]. Therefore, one can obtain a linear map if we restrict the domain to a re-

Regular Paper
Special Section on Software Systems 2020
This work is supported by the National Natural Science Foundation of China under Grant No. 61872371, the Open Fund from the State Key Laboratory of High Performance Computing of China (HPCL) under Grant No. 202001-07, and the Natural Key Research and Development Program of China under Grant No. 2018YFB0204301.

gion which is small enough, and this embodies the idea that any continuous mapping can be approximated via some piecewise linear ones.

The piecewise linear mechanism also provides a chance to perform exact verification against this kind of neural networks. Actually, as a mature technique, model checking [6,7] has been widely accepted. Inputs of a model checker consist of a model and a specification. The model is the description (might be abstracted) of the system to be verified, whereas the specification is a property we need to verify upon the model. For most cases, model checking is a completely automatic approach, because the model is merely a semantic structure of the specification, and model checking is to judge if the specification is satisfied by the model, which is required to be decidable. Another attractive feature of model checking is: when the specification is violated, the algorithm also yields a counterexample which explains the reason of violation, and this co-product is always considered to be important and valuable in locating bugs of design.

Actually, for an NN centered machine learning algorithm, the task of model checking can be classified into the following two levels.

Object Level. We take a (trained) neural network as the model, and the specification describes properties of the given NN.

Algorithm Level. We can take the learning algorithm itself as the model — after all, such an algorithm can be described as a transition system. Therefore, in this case, a specification is used to designate some property of the algorithm.

There is no wonder that giving an algorithm-level model checking framework is much more difficult than giving an object-level one. Even, for the object level, this task is never a trivial one. To the best of our knowledge, only a few of this kind of frameworks have been proposed for model checking simple non-temporal properties (e.g., the recent work on verifying neural networks with SMT (satisfiability modulo theory) solver [8]).

For classical model checking, one typically uses temporal logics as specification languages. Most temporal logics are merely built up from boolean variables together with temporal connectives. However, a boolean variable is too coarse for describing the data processing in the neural networks. On the other hand, involving too powerful mechanism in algebra tends to lead to the risk of undecidability or a prohibitively high complexity in decision.

As an attempt, by embedding terms about data processing into LTL (linear-time temporal logic) [9], we in this paper present a temporal logic which can describe value-related properties in ReLU networks. We call it ReLU temporal logic (ReTL, for short), whose semantics is defined with respect to ReLU-NNs.

Why do we say that specification languages are important? First of all, as an interface to persons conducting verification, a logic is much more intuitive and succinct, whereas using equality/inequality is much more complex and error-prone. In addition, as a standard mathematical description mechanism in computer science, logical formulas yield a connection to many problems, such as model checking, synthesis [10], and game-solving.

As a common sense in computer science, logics richer than Tarski arithmetic are considered to be complex in doing decision which is doubly exponential [11,12]. Therefore, a special fragment of ReTL is tailored being the specification language for doing ModelChecking. Indeed, since we use quantifiers (i.e., ∀ and ∃) in ReTL, a hierarchy of formulas exists according to the nesting and alternation of quantifiers. We follow the convention using Σ_i and Π_i to denote the set of formulas having i-alternations of ∃-∀ and ∀-∃ quantifiers in their normal form, respectively. Among these, $\Sigma_2 \cup \Pi_2$ is an important fragment of ReTL formulas. Existing work on the formal verification of neural networks concerns on four types of properties/problems:

1) local robustness, which expresses that all inputs in a given region are classified as the same class [13];

2) reachability property, which aims to compute the set of outputs for a given set of inputs [14];

3) interval property, which aims to compute a tight bound of outputs for a given set of inputs [15,16];

4) Lipschitzian property, which monitors the changes of the output with respect to small changes of the inputs [17].

The first three properties can be directly described by some $\Sigma_2 \cup \Pi_2$ formulas, whereas Lipschitzian property is describable only when we choose L_1-norm or L_∞-norm (cf. Section 5 for further explanation). In this paper, we show that the ModelChecking problem of $\Sigma_2 \cup \Pi_2$ fragment of ReTL is decidable in EXPSPACE — in this paper, complexity is measured w.r.t. the number of hidden neurons.

The main approach to achieving this result is to boil it down to the parameter emptiness (PE) decision problem for linear inequality/equality systems. Another technique we use to do model checking is ReLU-elimination, which converts a ReLU-system to a set of standard linear inequality systems. In this paper, we mainly use the Moore-Penrose inverse to solve linear equality/inequality systems.

The rest of this paper is organized as follows. In Section 2, we briefly revisit some basic notions and notations on linear algebra and ReLU networks. Since the general theory of linear equality/inequality systems is important to our work, we provide some conceptions and theorems that are used in Section 3. In Section 4, we introduce the ReLU-elimination technique, and show that we can deal with the so-called PE problem. ReTL is introduced in Section 5. In Section 6, we show how to perform ModelChecking against $\Sigma_2 \cup \Pi_2$ formulas, and this section also provides an analysis of complexity. We implement a prototype toolkit, and experimental results are given in Section 7. Related work about this paper is exhibited in Section 8, and we conclude this paper in Section 9.

2 Preliminaries

2.1 Vectors and Matrices

In this paper, we use $\mathbb{R}$ and $\mathbb{C}$ to denote the set of real numbers and the set of complex numbers, respectively. We use $\boldsymbol{M}$, $\boldsymbol{N}$ (possibly with subscript), etc. to range over matrices, and use $\boldsymbol{u}$, $\boldsymbol{v}$, etc. to denote vectors.

For an m-by-n matrix $\boldsymbol{M} = (a_{i,j})_{m\times n}$, we denote by $\mathit{row}(\boldsymbol{M})$ (resp. $\mathit{col}(\boldsymbol{M})$) the number of rows (resp. columns) of $\boldsymbol{M}$. We thus have $\mathit{row}(\boldsymbol{M}) = m$ and $\mathit{col}(\boldsymbol{M}) = n$ in this case. We use $\boldsymbol{M}^{\mathrm{T}}$ to denote the transposition of $\boldsymbol{M}$, and we denote by $\boldsymbol{M}^{\mathrm{H}}$ the transposed conjugate of $\boldsymbol{M}$.

If both $\boldsymbol{a} = (a_i)_n$ and $\boldsymbol{b} = (b_i)_n$ are real vectors, we denote by $\boldsymbol{a} \sim \boldsymbol{b}$ if $a_i \sim b_i$ for every $i \leqslant n$, here $\sim$ may be one of $>, <, \geqslant, \leqslant$ or $=$.

Given two vector sets $\mathcal{V}_1, \mathcal{V}_2 \subseteq \mathbb{C}^n$, we denote by $\mathcal{V}_1 + \mathcal{V}_2$ the set $\{\boldsymbol{v}_1 + \boldsymbol{v}_2 \mid \boldsymbol{v}_1 \in \mathcal{V}_1, \boldsymbol{v}_2 \in \mathcal{V}_2\}$. Particularly, we directly write $\{\boldsymbol{v}\} + \mathcal{V}$ as $\boldsymbol{v} + \mathcal{V}$.

We follow the convention using $\boldsymbol{I}_n$ to denote the identity matrix with shape n by n, and use $\boldsymbol{0}_n$ and $\boldsymbol{1}_n$ to denote the vectors with length n, whose elements are all 0 and 1, respectively. We usually drop off n from the subscript in the case that the shape of the matrix (or, vector) is clear from context.

There is no wonder that each matrix determines a linear mapping over vector space, namely, for matrix $\boldsymbol{M} = (a_{i,j})_{m\times n} \in \mathbb{C}^{m\times n}$ and vector $\boldsymbol{v} = (b_i)_n \in \mathbb{C}^n$, we definitely obtain vector $\boldsymbol{M} \cdot \boldsymbol{v} \in \mathbb{C}^m$. Then, $\boldsymbol{M}$ is injective (resp. surjective) if and only if $\mathrm{rank}(\boldsymbol{M}) = \mathit{col}(\boldsymbol{M})$ (resp. $\mathrm{rank}(\boldsymbol{M}) = \mathit{row}(\boldsymbol{M})$).

Matrix $\boldsymbol{M} \in \mathbb{C}^{n\times n}$ is said to be normal if $\boldsymbol{M}^{\mathrm{H}}\boldsymbol{M} = \boldsymbol{M}\boldsymbol{M}^{\mathrm{H}}$. Thus, if $\boldsymbol{M}$ is a Hermitian matrix (i.e., $\boldsymbol{M} = \boldsymbol{M}^{\mathrm{H}}$), then $\boldsymbol{M}$ is automatically normal. In addition, every symmetric real matrix is normal. Here is an important property of normal matrices.

Theorem 1 (Spectral Decomposition). *Let $\boldsymbol{M} \in \mathbb{C}^{n\times n}$ be a normal matrix, and then we have*

$$\boldsymbol{M} = \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^{\mathrm{H}}, \tag{1}$$

where each $\boldsymbol{v}_i$ is an eigenvector of $\boldsymbol{M}$ w.r.t. the eigenvalue λ_i, and the set $\{\boldsymbol{v}_i\}$ forms an orthonormal base of $\mathbb{C}^n$, i.e., $\boldsymbol{v}_i^{\mathrm{H}} \cdot \boldsymbol{v}_j = \delta_{i,j}$.

Recall that the value of Kronecker delta $\delta_{i,j}$ equals 1 if $i = j$, and 0 whenever $i \neq j$. Righthand of (1) is said to be the spectral decomposition of $\boldsymbol{M}$. For this reason, we sightly take an abuse of notations and let

$$\boldsymbol{M}^{-1} = \sum_{\lambda_i \neq 0} \frac{1}{\lambda_i} \boldsymbol{v}_i \boldsymbol{v}_i^{\mathrm{H}}, \tag{2}$$

even if the normal matrix $\boldsymbol{M}$ is singular.

For a normal matrix $\boldsymbol{M}$, let $\mathrm{supp}(\boldsymbol{M})$ be the linear space spanned by all its eigenvectors that do not correspond to eigenvalue 0. Namely,

$$\mathrm{supp}(\boldsymbol{M}) = \mathrm{span}\{\sum_i c_i \boldsymbol{v}_i \mid c_i \in \mathbb{C}, \lambda_i \neq 0\},$$

if the spectral decomposition of $\boldsymbol{M}$ is $\sum_i \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^{\mathrm{H}}$.

A U-matrix is a normal matrix such that each of its eigenvalue λ fulfills $|\lambda|^2 = 1$. Hence, for a U-matrix $\boldsymbol{M}$, we have $\boldsymbol{M}^{\mathrm{H}}\boldsymbol{M} = \boldsymbol{M}\boldsymbol{M}^{\mathrm{H}} = \boldsymbol{I}$. In addition, the set of all the row (or column) vectors of a U-matrix must be orthonormal.

Theorem 2 (Singularity Value Decomposition, SVD). *Every matrix $\boldsymbol{M}$ can be decomposed into $\boldsymbol{UDV}$, where both $\boldsymbol{U}$ and $\boldsymbol{V}$ are U-matrices, and $\boldsymbol{D}$ is a diagonal matrix (may be not a square one).*

For a real symmetric matrix, all its eigenvalues are real numbers, and it can always have a spectral decomposition that involves only real eigenvectors.

For this reason, in the following we mainly concern real matrices and vectors; thus the number field is confined to $\mathbb{R}$, rather than $\mathbb{C}$. In this setting, $\boldsymbol{M}^{\mathrm{H}}$ and $\boldsymbol{M}^{\mathrm{T}}$ coincide.

2.2 Non-Linear Mappings

In contrast, a non-linear mapping $\boldsymbol{f} = (f_i)_n$ never changes the shape of a vector, i.e., $\boldsymbol{f}((x_i)_n) = (f_i(x_i))_n$, and each f_i is a function from $\mathbb{R}$ to $\mathbb{R}$. Some typical non-linear mappings are listed as follows:

• sigmoid: where $f_i(x) = \exp(x)/(1+\exp(x))$,

• tanh: if we let $f_i(x) = \dfrac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$,

• relu: in which $f_i(x) = \max\{0, x\}$.

More generally, we also allow multiple non-linear mappings from $\mathbb{R}^n$ to $\mathbb{R}^n$ being of $\boldsymbol{F} = (F_i)_n$, where each F_i is an n-ary function from $\mathbb{R}^n$ to $\mathbb{R}$. Hence, $\boldsymbol{F}(\boldsymbol{v}) = \Big(F_i(v_1,\dots,v_n)\Big)_n$ if $\boldsymbol{v} = (v_i)_n$. For the particular case,

$$F_i(v_1,\dots,v_n) = \frac{\exp(v_i)}{\sum_{j=1}^n \exp(v_j)}.$$

$\boldsymbol{F}$ is called softmax.

For two matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ such that $row(\boldsymbol{M}) = row(\boldsymbol{N})$, we denote $(\boldsymbol{M} \vdots \boldsymbol{N})$ as the juxtaposition of $\boldsymbol{M}$ and $\boldsymbol{N}$.

For a linear or nonlinear mapping $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$, we define:

• $\mathrm{Ran}(\boldsymbol{f}) = \{\boldsymbol{f}(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathbb{R}^n\}$, and

• $\mathrm{Ker}(\boldsymbol{f}) = \{\boldsymbol{v} \in \mathbb{R}^n \mid \boldsymbol{f}(\boldsymbol{v}) = \boldsymbol{0}\}$,

which are called the range and the kernel of $\boldsymbol{f}$, respectively.

Theorem 3. *For any (real) matrix $\boldsymbol{M}$, as a linear mapping, we have*

$$\mathrm{Ran}(\boldsymbol{M})^{\perp} = \mathrm{Ker}(\boldsymbol{M}^{\mathrm{T}}),$$

where $\mathrm{Ran}(\boldsymbol{M})^{\perp}$ is the orthogonal space of $\mathrm{Ran}(\boldsymbol{M})$.

2.3 ReLU Neural Network

For a neural network consisting of ℓ-layers of perceptrons (neuron cells), it could be described via a set of matrices $\{\boldsymbol{M}_i\}_{i=0}^{\ell-1}$ and vectors $\{\boldsymbol{b}_i\}_{i=0}^{\ell-1}$, where:

• $\boldsymbol{M}_i$ depicts the weights of connections between the i-th and the $(i+1)$-th layers of perceptrons,

• $\boldsymbol{b}_i$ refers to the biases associated with the perceptrons of the i-th layer ($i \geqslant 0$).

Since we are concerned about fully-connected perceptrons in this paper, we require that $row(\boldsymbol{M}_i) = row(\boldsymbol{b}_i) = col(\boldsymbol{M}_{i+1})$ for every $i \geqslant 0$. Indeed, $row(\boldsymbol{M}_i)$ designates the number of perceptrons of the i-th layer.

For a given input vector $\boldsymbol{v}_0 \in \mathbb{R}^{col(\boldsymbol{M}_0)}$, a ReLU network performs in the following way.

• It produces a series of intermediate vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_\ell$, where

$$\boldsymbol{v}_{t+1} = \begin{cases} \mathrm{relu}(\boldsymbol{M}_t\boldsymbol{v}_t + \boldsymbol{b}_t), & \text{if } t < \ell - 1,\\ \boldsymbol{M}_t\boldsymbol{v}_t + \boldsymbol{b}_t, & \text{if } t = \ell - 1.\end{cases}$$

• Let $\mathrm{softmax}(\boldsymbol{v}_\ell) = [c_j]_N$, and then it takes $\operatorname{argmax}_j(\{c_j \mid j \leqslant N\})$ as the final output, i.e., the index of the maximum element in $\mathrm{softmax}(\boldsymbol{v}_\ell)$.

In the case that the last step is not well-defined, namely, there is more than one maximum element in $\mathrm{softmax}(\boldsymbol{v}_\ell)$, we call that $\boldsymbol{v}_0$ admits an adversarial example.

3 Theory of Generic Linear Inequality

Although the theory of linear equality/inequality system is well-founded already, in this paper, in order to establish the decision procedure for the ModelChecking problem of our proposed logic, we need a special technique to give an explicit representation of the solution space. To make the paper self-contained, we need to revisit the related issues in this section.

Let $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ be a real matrix, and let

$$\boldsymbol{M}^{+} = (\boldsymbol{M}^{\mathrm{T}}\boldsymbol{M})^{-1}\boldsymbol{M}^{\mathrm{T}}, \tag{3}$$

be the Moore-Penrose inverse (M-P inverse, for short) of $\boldsymbol{M}$. Remind again that here $\boldsymbol{M}^{-1}$ is the general inverse operation (cf. (2)).

Theorem 4 (Penrose[18]). *$\boldsymbol{X} = \boldsymbol{M}^{+}$ is the unique solution of the following equation system:*

$$\begin{array}{ll} \boldsymbol{MXM} = \boldsymbol{M}, & \boldsymbol{XMX} = \boldsymbol{X},\\ (\boldsymbol{MX})^{\mathrm{T}} = \boldsymbol{MX}, & (\boldsymbol{XM})^{\mathrm{T}} = \boldsymbol{XM}.\end{array} \tag{4}$$

Proof. The fact that $\boldsymbol{M}^{+}$ fulfills the system can be examined immediately. Supposing we have another solution $\boldsymbol{M}'$ that fulfills (4) as well, then:

$$\begin{aligned}
&\boldsymbol{M}'\\
=\;& \boldsymbol{M}'\boldsymbol{M}\boldsymbol{M}' = \boldsymbol{M}'\boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{M}\boldsymbol{M}'\\
=\;& (\boldsymbol{M}'\boldsymbol{M})^{\mathrm{T}}(\boldsymbol{M}^{+}\boldsymbol{M})^{\mathrm{T}}\boldsymbol{M}' = (\boldsymbol{M}^{+}\boldsymbol{M}\boldsymbol{M}'\boldsymbol{M})^{\mathrm{T}}\boldsymbol{M}'\\
=\;& (\boldsymbol{M}^{+}\boldsymbol{M})^{\mathrm{T}}\boldsymbol{M}' = \boldsymbol{M}^{+}\boldsymbol{M}\boldsymbol{M}' = \boldsymbol{M}^{+}(\boldsymbol{M}\boldsymbol{M}')^{\mathrm{T}}\\
=\;& \boldsymbol{M}^{+}(\boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{M}\boldsymbol{M}')^{\mathrm{T}} = \boldsymbol{M}^{+}(\boldsymbol{M}\boldsymbol{M}')^{\mathrm{T}}(\boldsymbol{M}\boldsymbol{M}^{+})^{\mathrm{T}}\\
=\;& \boldsymbol{M}^{+}\boldsymbol{M}\boldsymbol{M}'\boldsymbol{M}\boldsymbol{M}^{+} = \boldsymbol{M}^{+}\boldsymbol{M}\boldsymbol{M}^{+} = \boldsymbol{M}^{+}. \qquad \square
\end{aligned}$$

Theorem 5. *Both $\boldsymbol{MM}^{+}$ and $\boldsymbol{M}^{+}\boldsymbol{M}$ are projectors, i.e., an eigenvalue of $\boldsymbol{MM}^{+}$ (resp. $\boldsymbol{M}^{+}\boldsymbol{M}$) can only be 0 or 1; hence the spectral decomposition of $\boldsymbol{MM}^{+}$ (resp. $\boldsymbol{M}^{+}\boldsymbol{M}$) must be of the form $\sum_i \boldsymbol{v}_i\boldsymbol{v}_i^{\mathrm{H}}$.*

Proof. Supposing that $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ and $\mathrm{rank}(\boldsymbol{M}) = r$, let $\boldsymbol{UDV}$ be the SVD decomposition of $\boldsymbol{M}$, where

$\boldsymbol{D} = \begin{pmatrix} \boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}$ in which $\boldsymbol{C}$ is a non-singular diagonal matrix. Then, let

$$\boldsymbol{M}' = \boldsymbol{V}^{\mathrm{H}} \begin{pmatrix} \boldsymbol{C}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \boldsymbol{U}^{\mathrm{H}},$$

and one can check that $\boldsymbol{X} = \boldsymbol{M}'$ fulfills all requirements in (4). Then, we immediately have that $\boldsymbol{M}^{+} = \boldsymbol{M}'$ according to Theorem 4, and the conclusion can be thus ensured. □

The proof of Theorem 5 provides an alternative way to calculate $\boldsymbol{M}^{+}$. Therefore, if $\boldsymbol{M}$ is a non-singular square matrix, we immediately have $\boldsymbol{M}^{+} = \boldsymbol{M}^{-1}$. In this sense, M-P inverse generalizes standard inverse matrix.

Theorem 6. *For any real matrix $\boldsymbol{M}$, we have*

$$\mathrm{Ran}(\boldsymbol{I} - \boldsymbol{M}^{+}\boldsymbol{M}) = \mathrm{Ker}(\boldsymbol{M}^{+}\boldsymbol{M}) = \mathrm{Ran}(\boldsymbol{M}^{\mathrm{T}})^{\perp}, \quad (5)$$

and hence $\mathrm{Ran}(\boldsymbol{M}) = \mathrm{Ran}(\boldsymbol{M}\boldsymbol{M}^{+})$.

Theorem 7 [19]. *For each $\boldsymbol{b}$, we have that $\boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{b}$ is the asymptotical vector of $\boldsymbol{b}$ within* $\mathrm{Ran}(\boldsymbol{M})$. *In other words, we have*

$$\|\boldsymbol{M}\boldsymbol{x} - \boldsymbol{b}\| \geqslant \|\boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{b} - \boldsymbol{b}\|,$$

for every $\boldsymbol{x}$.

Proof. We first observe that $\boldsymbol{b} = \boldsymbol{b}_1 + \boldsymbol{b}_2$ where $\boldsymbol{b}_1 = \boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{b}$ and $\boldsymbol{b}_2 = (\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{b}$. Therefore, for every $\boldsymbol{x}$, we have

$$\begin{aligned} & \boldsymbol{M}\boldsymbol{x} - \boldsymbol{b} \\ = {} & \boldsymbol{M}\boldsymbol{x} - \boldsymbol{b}_1 - \boldsymbol{b}_2 \\ = {} & \boldsymbol{M}(\boldsymbol{x} - \boldsymbol{M}^{+}\boldsymbol{b}) - (\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{b}. \end{aligned}$$

Remind that $\boldsymbol{M}(\boldsymbol{x} - \boldsymbol{M}^{+}\boldsymbol{b}) \in \mathrm{Ran}(\boldsymbol{M})$ and $(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{b} \in \mathrm{Ran}(\boldsymbol{M})^{\perp}$ according to Theorem 6; hence

$$\begin{aligned} & \|\boldsymbol{M}\boldsymbol{x} - \boldsymbol{b}\|^2 \\ = {} & \|\boldsymbol{M}(\boldsymbol{x} - \boldsymbol{M}^{+}\boldsymbol{b}) - (\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{b}\|^2 \\ = {} & \|\boldsymbol{M}(\boldsymbol{x} - \boldsymbol{M}^{+}\boldsymbol{b})\|^2 + \|(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{b}\|^2 \\ \geqslant {} & \|(\boldsymbol{I} - \boldsymbol{M}\boldsymbol{M}^{+})\boldsymbol{b}\|^2 \\ = {} & \|\boldsymbol{M}\boldsymbol{M}^{+}\boldsymbol{b} - \boldsymbol{b}\|^2. \end{aligned}$$ □

According to the proof of the above theorem, we can see that $\boldsymbol{x} = \boldsymbol{M}^{+}\boldsymbol{b}$ is a special solution of the asymptotical equation[1] $\boldsymbol{M}\boldsymbol{x} \doteq \boldsymbol{b}$, and hence its general solution turns out to be

$$\boldsymbol{M}^{+}\boldsymbol{b} + \mathrm{Ker}(\boldsymbol{M}) = \boldsymbol{M}^{+}\boldsymbol{b} + \mathrm{supp}(\boldsymbol{I} - \boldsymbol{M}^{+}\boldsymbol{M}),$$

because $\mathrm{Ker}(\boldsymbol{M})$ is precisely $\mathrm{supp}(\boldsymbol{I} - \boldsymbol{M}^{+}\boldsymbol{M})$.

Therefore, the exact solution of $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{v}$ turns out to be $\boldsymbol{x} = \boldsymbol{M}^{+}\boldsymbol{b} + \mathrm{Ker}(\boldsymbol{M})$ if $\boldsymbol{b} \in \mathrm{Ran}(\boldsymbol{M})$.

Subsequently, we try to find the pre-image of a set $\mathcal{V}$ (not necessarily a linear space) of $\boldsymbol{M}$, denoted as $\boldsymbol{M}^{-1}[\mathcal{V}]$, which is the set $\{\boldsymbol{x} \mid \boldsymbol{M}\boldsymbol{x} \in \mathcal{V}\}$.

According to the previous analysis, if $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{y} \in \mathcal{V}$, then we must have $\boldsymbol{y} \in \mathrm{Ran}(\boldsymbol{M}) \cap \mathcal{V}$. We thus have

$$\boldsymbol{M}^{-1}[\mathcal{V}] = \{\boldsymbol{M}^{+}\boldsymbol{b} \mid \boldsymbol{b} \in \mathcal{V} \cap \mathrm{Ran}(\boldsymbol{M})\} + \mathrm{Ker}(\boldsymbol{M}). \quad (6)$$

Let us now deal with the inequality $\boldsymbol{M}\boldsymbol{x} \geqslant \boldsymbol{b}$, and for this we have the following theorem.

Theorem 8. *$\boldsymbol{M}\boldsymbol{x} \geqslant \boldsymbol{b}$ if and only if $\boldsymbol{x} \in \{\boldsymbol{M}^{+}\boldsymbol{v} \mid \boldsymbol{v} \geqslant \boldsymbol{b}, \boldsymbol{v} \in \mathrm{Ran}(\boldsymbol{M})\} + \mathrm{Ker}(\boldsymbol{M})$.*

Proof. Because $\boldsymbol{M}\boldsymbol{x} \geqslant \boldsymbol{b}$ if and only if $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{v}$ for some $\boldsymbol{v} \in \mathrm{Ran}(\boldsymbol{M})$ having $\boldsymbol{v} \geqslant \boldsymbol{b}$, the conclusion can be obtained from the aforementioned conclusion about linear equation. □

Note that the conclusion of Theorem 8 can be further generalized into any other comparison operators, even if it mixes[2] $>$, $\geqslant$, or $=$. For example, in the system

$$\begin{cases} 2x_1 + 3x_2 > 2, \\ 3x_1 + x_3 = 1, \\ x_2 - 2x_3 \geqslant 5, \end{cases}$$

by denoting

$$\boldsymbol{M} = \begin{pmatrix} 2 & 3 & 0 \\ 3 & 0 & 1 \\ 0 & 1 & -2 \end{pmatrix}, \ \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} 2 \\ 1 \\ 5 \end{pmatrix}, \text{ and } \sim = \begin{pmatrix} > \\ = \\ \geqslant \end{pmatrix},$$

the above system can be written as $\boldsymbol{M}\boldsymbol{x} \sim \boldsymbol{b}$.

Theorem 9. *$\boldsymbol{M}\boldsymbol{x} \sim \boldsymbol{b}$ if and only if $\boldsymbol{x} \in \{\boldsymbol{M}^{+}\boldsymbol{v} \mid \boldsymbol{v} \sim \boldsymbol{b}, \boldsymbol{v} \in \mathrm{Ran}(\boldsymbol{M})\} + \mathrm{Ker}(\boldsymbol{M})$.*

A closely related notion to inequality system is finitely representable set (FRS, for short). Given a finite vector set $\{\boldsymbol{v}_i\}_{i=1}^{n}$, then FRS consists of vectors being of $\sum_{i=1} c_i\boldsymbol{v}_i$, possibly with additional constraints on the coefficients c_is. Here $\{\boldsymbol{v}_i\}_{i=1}^{n}$ is called the generating vector set. We now list some typical finitely representable sets:

[1] The goal of an asymptotical equation $\boldsymbol{M}\boldsymbol{x} \doteq \boldsymbol{b}$ is to find some $\boldsymbol{x}$ making $\|\boldsymbol{M}\boldsymbol{x} - \boldsymbol{b}\|$ minimal.

[2] For other two relations $<$ and $\leqslant$, we can negate elements in the corresponding row in both $\boldsymbol{M}$ and $\boldsymbol{b}$, and then change them into $>$ and $\geqslant$, respectively.

1) *Linear Space*: if no additional constraint is imposed to the coefficients;

2) *Cone* (*Hull*): if we require that each $c_i \geqslant 0$;

3) *Convex Hull*: if we require that each $0 \leqslant c_i \leqslant 1$, and $\sum_{i=1}^{n} c_i = 1$.

4) *Polyhedral Cone*: if the coefficients fulfill a linear inequality system.

Let $\boldsymbol{V} = \left(\boldsymbol{v}_1 \vdots \cdots \vdots \boldsymbol{v}_n\right)$ and $\boldsymbol{c} = (c_1, \ldots, c_n)^{\mathrm{T}}$, and then a finitely representable set can be characterized by a linear inequality system about $\boldsymbol{V}$ and $\boldsymbol{c}$.

Note that FRSs are closed under intersection, because the simultaneous system of given FRSs just captures their intersection. According to Farkas' lemma[20], we immediately have the following theorem.

Theorem 10. *The problem if the intersection of several given FRSs is empty can be decided in polynomial time.*

Also note that linear transformation (plus some shifting) preserves finite representability. The reason is: for an FRS $\mathcal{V}$ with generating vector set $\{\boldsymbol{v}_i\}_{i=1}^{n}$, the set $\{\boldsymbol{M}\boldsymbol{v} + \boldsymbol{b} \mid \boldsymbol{v} \in \mathcal{V}\}$ is also an FRS with generating vector set $\{\boldsymbol{M}\boldsymbol{v}_i\}_{i=1}^{n} \cup \{\boldsymbol{b}\}$, where $\boldsymbol{M}\boldsymbol{b}_i$ is imposed the same coefficient constraint as that to $\boldsymbol{v}_i$, and the coefficient of $\boldsymbol{b}$ is set to 1.

4 PE-Decision and ReLU-Elimination

We, in this section, introduce two key techniques about ReLU-network model checking, namely PE-decision and ReLU-elimination.

4.1 Decision Procedure for Emptiness of Parameterized Systems

In this subsection, we deal with the following problem, called parameterized emptiness (PE, for short): for a given linear inequality system $\boldsymbol{F}(\boldsymbol{x}; \boldsymbol{y})$, in which $\boldsymbol{x}$ is taken as unknown variable, whereas $\boldsymbol{y}$ acts as parameter, the goal is to find the condition of $\boldsymbol{y}$ making the solution space of $\boldsymbol{x}$ empty.

For example, the PE problem for the trivial case $\boldsymbol{0x} \geqslant \boldsymbol{y}$ is just

$$\boldsymbol{y} \in \{\boldsymbol{v} \neq \boldsymbol{0} \mid \boldsymbol{v} \geqslant \boldsymbol{0}\},$$

because in this case the inequality has no solution of $\boldsymbol{x}$ (remind that the above set is not equal to $\{\boldsymbol{y} \mid \boldsymbol{y} > \boldsymbol{0}\}$).

Since all operations are linear, it is clear that $\boldsymbol{F}(\boldsymbol{x}; \boldsymbol{y})$ can always be equivalently transformed into the normal form

$$\boldsymbol{M}\boldsymbol{x} \sim \boldsymbol{N}\boldsymbol{y} + \boldsymbol{b}, \tag{7}$$

where $\sim$ is a vector of relations, each element of which is among $\{\geqslant, >, =\}$.

First of all, let $\boldsymbol{z} = \boldsymbol{N}\boldsymbol{y} + \boldsymbol{b}$, then (7) boils down to another PE problem $\boldsymbol{M}\boldsymbol{x} \sim \boldsymbol{z}$.

According to Theorem 9, the solution space is empty if and only if

$$\mathrm{Ran}(\boldsymbol{M}) \cap \{\boldsymbol{v} \mid \boldsymbol{v} \sim \boldsymbol{z}\} = \emptyset.$$

Note that $\mathrm{Ran}(\boldsymbol{M})$ is a linear space and $\{\boldsymbol{v} \mid \boldsymbol{v} \sim \boldsymbol{z}\}$ is a finitely representable set (FRS, for short). Supposing $\sim = (\sim_i)_n$, let

$$\mathcal{V}_{\sim} = \left\{ \boldsymbol{v} = (v_i)_n \;\middle|\; \begin{array}{c} row(\boldsymbol{v}) = row(\sim), \\ \sim_i \in \{>, \geqslant\} \implies v_i \geqslant 0 \end{array} \right\},$$

then from a geometric view[3], we can conclude that the solution of the PE problem $\boldsymbol{M}\boldsymbol{x} \sim \boldsymbol{z}$ is just

$$\boldsymbol{z} \in \bigcup_{\substack{\boldsymbol{v} \in \mathrm{Ran}(\boldsymbol{M})^{\perp} \cap \mathcal{V}_{\sim} \\ \alpha \gtrsim 0}} \left(\alpha \boldsymbol{v} + \{\boldsymbol{v}\}^{\perp} \right), \tag{8}$$

where

$$\gtrsim \text{ is } \begin{cases} \geqslant, & \text{if } \sim \text{ involves at least one ``}>\text{'' as element}, \\ >, & \text{otherwise}. \end{cases}$$

From (8), one can see that such $\boldsymbol{z}$ exists if and only if $\mathrm{Ran}(\boldsymbol{M})^{\perp} \cap \mathcal{V}_{\sim} \neq \{\boldsymbol{0}\}$. Indeed, $\mathrm{Ran}(\boldsymbol{M})^{\perp} \cap \mathcal{V}_{\sim}$ itself forms a cone hull, and hence one can give its finite representation.

Let us denote by $\mathcal{V}_{\boldsymbol{Z}}$ the right side of (8), then the solution of the PE problem of (7) is

$$\begin{aligned} &\boldsymbol{N}^{-1}[\mathcal{V}_{\boldsymbol{Z}} - \boldsymbol{b}] \\ &= \{\boldsymbol{N}^{+}\boldsymbol{v} \mid \boldsymbol{v} \in \mathrm{Ran}(\boldsymbol{N}) \cap (\mathcal{V}_{\boldsymbol{z}} - \boldsymbol{b}) + \mathrm{Ker}(\boldsymbol{N})\}. \end{aligned}$$

As a consequence, this set is also finitely representable.

Nevertheless, we here give an alternative characterization for the solution of PE-problem, which is used to deal with the batch manner — namely, we are given a series of PE-problems $\{\mathcal{T}_i(\boldsymbol{x}, \boldsymbol{y})\}_i$, to determine if the intersection of all solution spaces is empty. For this, we have the following theorem.

Theorem 11. *For the PE-problem given by (7), the complementary space*[4] *of the solution is a polyhedral cone.*

Proof. For the PE-problem about $\boldsymbol{x}$ and $\boldsymbol{y}$ given as

$$\boldsymbol{M}\boldsymbol{x} \sim \boldsymbol{N}\boldsymbol{y} + \boldsymbol{b},$$

[3] Alternatively, one can also get the same conclusion by using Farkas' lemma.

[4] Here, we take $\mathbb{R}^{row(\boldsymbol{y})}$ as the universal set.

according to Theorem 9, the complementary space of the solution can be characterized as $\boldsymbol{N}^+[\mathcal{V}]$, where

$$\mathcal{V} = \{\boldsymbol{v} \mid \boldsymbol{v} \in \mathrm{Ran}(\boldsymbol{N}), \exists \boldsymbol{v}' \in \mathrm{Ran}(\boldsymbol{M}) - \boldsymbol{b} \text{ s.t. } \boldsymbol{v} \sim \boldsymbol{v}'\},$$

and then it suffices to show $\mathcal{V}$ is a polyhedral cone.

Supposing that

$$\mathrm{Ran}(\boldsymbol{N}) = \mathrm{span}\{\boldsymbol{n}_1, \ldots, \boldsymbol{n}_k\},$$

and

$$\mathrm{Ran}(\boldsymbol{M}) = \mathrm{span}\{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_t\},$$

we can introduce a set of coefficients $c_1, \ldots, c_k, c'_1, \ldots, c'_t \in \mathbb{R}$ fulfilling the constraints

$$\sum_i c_i \boldsymbol{n}_i \sim \sum_j c'_j \boldsymbol{m}_j - \boldsymbol{b},$$

and $\mathcal{V}$ is just an FRS generated by the vector set $\{\boldsymbol{l}_i\}_{i=1}^{k+t}$ with the coefficients $\{c_1, \ldots, c_k, c'_1, \ldots, c'_t\}$, where

$$\boldsymbol{l}_i = \begin{cases} \boldsymbol{n}_i, & \text{if } i \leqslant k, \\ \boldsymbol{0}, & \text{otherwise,} \end{cases}$$

and thus we claim that $\mathcal{V}$ is a polyhedral cone. □

4.2 ReLU System and ReLU-Elimination

A ReLU system is a set of equalities and/or inequalities, and each of them may include not only standard operators but also the ReLU operation. Here gives an example of such a system about $\boldsymbol{x}$:

$$\begin{cases} \boldsymbol{M}_1\boldsymbol{x} + \boldsymbol{b}_1 = \boldsymbol{b}_2, \\ \mathrm{relu}(\boldsymbol{M}_2\boldsymbol{x} + \boldsymbol{b}_2) \geqslant \mathrm{relu}(\boldsymbol{M}_3\boldsymbol{x} + \boldsymbol{b}_3), \\ \mathrm{relu}(\boldsymbol{M}_4\boldsymbol{x}) \leqslant \boldsymbol{b}_4, \end{cases}$$

where $\boldsymbol{M}_i$ and $\boldsymbol{b}_i$ for $1 \leqslant i \leqslant 4$ are given matrices and vectors.

We have discussed how to handle ReLU-free formulas in Subsection 4.1. Therefore, in what follows we mainly deal with formulas involving ReLU operators. In this paper, we call such technique ReLU-elimination.

The first step of ReLU-elimination is to replace each ReLU-expression with a series of equalities and/or inequalities. The details are listed as follows.

• Supposing that we have a sub-expression $\mathrm{relu}(\boldsymbol{t})$ with $\mathit{row}(\boldsymbol{t}) = n$, let us denote by $[n]$ the set $\{1, 2, \ldots, n\}$.

• For each subset $U \subseteq [n]$, let $\boldsymbol{E}_U = \mathrm{diag}(d_1, \ldots, d_n)$ where

$$d_i = \begin{cases} 1, & \text{if } i \in U, \\ 0, & \text{otherwise.} \end{cases}$$

• Subsequently, we impose two additional inequalities

$$\boldsymbol{E}_U\boldsymbol{t} \geqslant \boldsymbol{0} \quad \text{and} \quad (\boldsymbol{I} - \boldsymbol{E}_U)\boldsymbol{t} \leqslant \boldsymbol{0}\ , \tag{9}$$

and replace $\mathrm{relu}(\boldsymbol{t})$ with $\boldsymbol{E}_U\boldsymbol{t}$ in the original expression.

Intuitively, for a pre-assumed subset U of $[n]$, projectors $\boldsymbol{E}_U$ and $\boldsymbol{I} - \boldsymbol{E}_U$ extract the non-negative part and the non-positive part of $\boldsymbol{t}$, respectively. In the situation that (9) holds, $\mathrm{relu}(\boldsymbol{t})$ is precisely $\boldsymbol{E}_U\boldsymbol{t}$.

With such a transformation, we can eliminate all ReLU operators, and thus obtain a set of ReLU-free expressions again. That is, for each sub-expression $\mathrm{relu}(\boldsymbol{t})$ and each such subset U, we need to initialize a specific system. Therefore, as the cost, the number of systems is exponential in $\sum_i \mathit{row}(\boldsymbol{t}_i)$ wherein $\mathrm{relu}(\boldsymbol{t}_i)$ occurring as a sub-expression.

Remark. In [21], the authors introduced an alternative way to do ReLU-elimination, which requires a large constant matrix, and each element of this matrix is approximate to ∞. Indeed, our approach is specially tailored for PE-decision.

5 Temporal Logic: ReTL

Since all preparations on algebra are done, we here introduce a logic which is able to specify temporal properties w.r.t. ReLU networks. In this paper, we call this kind of temporal logic ReTL.

First of all, terms (or expressions) of ReTL are inductively built up on vector constants (such as $\boldsymbol{v}$, $\boldsymbol{v}_1$, $\boldsymbol{v}_2$), and vector variables (ranging over $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{x}_1$, ...), and they can be described with the following abstract grammar:

$$\boldsymbol{t} ::= \boldsymbol{v} \mid \boldsymbol{x} \mid \bigcirc^k \boldsymbol{t} \mid \boldsymbol{t}[i] \mid \boldsymbol{M}\boldsymbol{t} \mid \boldsymbol{t} + \boldsymbol{t},$$

where $i, k \in \mathbb{N}$, and $\boldsymbol{M}$ is some constant matrix compatible with $\boldsymbol{t}$. Intuitively, $\boldsymbol{t}[i]$ refers to the i-th element of $\boldsymbol{t}$, which is a scalar. In addition, we may directly write $\bigcirc^0\boldsymbol{t}$ and $\bigcirc^1\boldsymbol{t}$ as $\boldsymbol{t}$ and $\bigcirc\boldsymbol{t}$, respectively.

Formulas of ReTL are given as below.

• Both $\bot$ and $\top$ are ReTL formulas.

• If $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are two terms, then $\boldsymbol{t}_1 \sim \boldsymbol{t}_2$ is a formula if $\mathit{row}(\boldsymbol{t})_1 = \mathit{row}(\boldsymbol{t}_2)$, where $\sim \in \{>, \geqslant, =, \neq, <, \leqslant\}$.

• If φ and ψ are ReTL formulas, then so do $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$ and $\neg\varphi$.

• Supposing that φ and ψ are ReTL formulas, then both $\mathsf{X}\varphi$ and $\varphi\mathsf{U}\psi$ are ReTL formulas.

• If φ is a ReTL formula, $\boldsymbol{x}$ is a vector variable, then both $\exists \boldsymbol{x} \in \mathbb{R}^n.\varphi$ and $\forall \boldsymbol{x} \in \mathbb{R}^n.\varphi$ are ReTL formulas.

As in standard LTL, we introduce the following derived operators for the sake of brevity:

$$\mathsf{F}\varphi := \top \mathsf{U} \varphi, \qquad \text{and} \qquad \mathsf{G}\varphi := \neg \mathsf{F} \neg \varphi.$$

A formula φ is closed, if it involves no free variable, i.e., all the occurrences of all variables in φ are quantified by either $\forall$ or $\exists$.

Recall that a ReLU network (having ℓ layers) $\mathcal{N}$ can be characterized via a series of matrix-vector alternations

$$\boldsymbol{M}_0, \boldsymbol{b}_0, \boldsymbol{M}_1, \boldsymbol{b}_1, \ldots, \boldsymbol{M}_{\ell-1}, \boldsymbol{b}_{\ell-1},$$

fulfilling the requirement that $row(\boldsymbol{M}_i) = row(\boldsymbol{b}_i) = col(\boldsymbol{M}_{i+1})$. It works as we have described in Subsection 2.3.

Semantics of ReTL formulas is defined w.r.t. a ReLU network $\mathcal{N}$, a position $i < \ell$, and an evaluation $\mathbb{V}$ which assigns each vector variable to a vector of proper shape. We inductively define the semantics of a term $\boldsymbol{t}$, denoted as $[\![\boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i}$, as follows.

• For a vector constant $\boldsymbol{v}$, we have $[\![\boldsymbol{v}]\!]^{\mathbb{V}}_{\mathcal{N},i} = \boldsymbol{v}$.

• $[\![\boldsymbol{x}]\!]^{\mathbb{V}}_{\mathcal{N},i} = \mathbb{V}(\boldsymbol{x})$, for any vector variable $\boldsymbol{x}$.

• To compute $[\![\bigcirc^k \boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i}$, we need to distinguish several cases:

1) if $k = 0$, then $[\![\bigcirc^k \boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i} = [\![\boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i}$,

2) if $k > 0$ and $i + k < \ell$, then $[\![\bigcirc^k \boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i}$ is $\text{relu}(\boldsymbol{M}_{i+k-1}[\![\bigcirc^{k-1} \boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i} + \boldsymbol{b}_{i+k-1})$,

3) if $k > 0$ and $i + k = \ell$, then $[\![\bigcirc^k \boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i}$ is $\boldsymbol{M}_{\ell-1}[\![\bigcirc^{k-1} \boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i} + \boldsymbol{b}_{\ell-1}$, and

4) if $i + k > \ell$, this term is undefined.

Note that we discard the softmax operation for the second case, to preserve piecewise-linearity of all operations. This would not essentially affect any comparison, e.g., $\text{softmax}(\boldsymbol{t})[i] \sim \text{softmax}(\boldsymbol{t})[j]$ iff $\boldsymbol{t}[i] \sim \boldsymbol{t}[j]$.

• $[\![\boldsymbol{t}[k]]\!]^{\mathbb{V}}_{\mathcal{N},i}$ is just the k-th element of $[\![\boldsymbol{t}]\!]^{\mathbb{V}}_{\mathcal{N},i}$, and the value becomes "undefined" if k is greater than the length[5] of the corresponding vector.

• $[\![\boldsymbol{M}\boldsymbol{t}]\!]_{\mathcal{N},i} = \boldsymbol{M}[\![\boldsymbol{t}]\!]_{\mathcal{N},i}$.

• $[\![\boldsymbol{t}_1 + \boldsymbol{t}_2]\!]_{\mathcal{N},i} = [\![\boldsymbol{t}_1]\!]_{\mathcal{N},i} + [\![\boldsymbol{t}_2]\!]_{\mathcal{N},i}$.

Subsequently, we can define the satisfaction relation $\models$ as follows.

• $\mathcal{N}, i \models_{\mathbb{V}} \top$ and $\mathcal{N}, i \not\models_{\mathbb{V}} \bot$ always hold.

• $\mathcal{N}, i \models_{\mathbb{V}} \boldsymbol{t}_1 \sim \boldsymbol{t}_2$ iff $[\![\boldsymbol{t}_1]\!]^{\mathbb{V}}_{\mathcal{N},i} \sim [\![\boldsymbol{t}_2]\!]^{\mathbb{V}}_{\mathcal{N},i}$.

• Semantics of boolean operators ($\neg$, $\wedge$, $\vee$) are defined as usual, e.g., $\mathcal{N}, i \models_{\mathbb{V}} \neg\varphi$ iff $\mathcal{N}, i \not\models_{\mathbb{V}} \varphi$.

• $\mathcal{N}, i \models_{\mathbb{V}} \mathsf{X}\varphi$ iff $i \leqslant \ell$ and $\mathcal{N}, i+1 \models_{\mathbb{V}} \varphi$.

• $\mathcal{N}, i \models_{\mathbb{V}} \varphi \mathsf{U} \psi$ iff there is some $j \geqslant i$ such that $\mathcal{N}, j \models_{\mathbb{V}} \psi$ and $\mathcal{N}, k \models_{\mathbb{V}} \varphi$ for every $i \leqslant k < j$.

• $\mathcal{N}, i \models_{\mathbb{V}} \exists x \in \mathbb{R}^n.\varphi$ iff there is some $\boldsymbol{v} \in \mathbb{R}^n$ making $\mathcal{N}, i \models_{\mathbb{V}[\boldsymbol{v}/\boldsymbol{x}]} \varphi$.

• $\mathcal{N}, i \models_{\mathbb{V}} \forall x \in \mathbb{R}^n.\varphi$ iff for every $\boldsymbol{v} \in \mathbb{R}^n$ we have $\mathcal{N}, i \models_{\mathbb{V}[\boldsymbol{v}/\boldsymbol{x}]} \varphi$.

In the above, $\mathbb{V}[\boldsymbol{v}/\boldsymbol{x}]$ is also an evaluation, which is almost identical to $\mathbb{V}$, except for $\mathbb{V}[\boldsymbol{v}/\boldsymbol{x}](\boldsymbol{x}) = \boldsymbol{v}$.

For the sake of brevity, we omit the index i from the left-side of satisfaction notation when $i = 0$. In addition, we drop off the evaluation from the subscript when the formula is closed, because the satisfaction of a closed formula is irrelative to the variable evaluation.

In Section 7, we provide some examples describing properties we are concerned about using ReTL formulas.

Bewaring the use of until operator (U), one sometimes runs a risk of type mis-match. For example, for the formula

$$\exists \boldsymbol{x} \in \mathbb{R}^n.(\boldsymbol{x} > \bigcirc \boldsymbol{v}) \mathsf{U} (\boldsymbol{x} > \boldsymbol{1}),$$

the length of $\bigcirc \boldsymbol{v}$ is sensitive to the location, and such vector comparison might be nonsense whenever the width of the network (in other words, the number of perceptrons of current layer) changes.

Nevertheless, we have the operator U in the logic for several reasons.

• First of all, we want this logic subsumes LTL in syntax, and until is the soul operator of a majority of various temporal logics.

• Second, as we will see later, when doing model checking upon a simple neural network (i.e., the network does not contain recurrent loops), U can be rewritten with X if there is no type conflict.

• Last but not least, as a more ambitious intention, this operator may be needed when we generalize our model into recurrent neural networks.

6 Model Checking $\Sigma_2 \cup \Pi_2$ Fragment of ReTL

6.1 Framework

Given a ReLU network $\mathcal{N}$, and a closed ReTL formula φ, the ModelChecking problem is just to check whether $\mathcal{N} \models \varphi$ holds.

Since we allow quantifiers, algebraic operations as well as comparisons, this logic definitely subsumes Tarski arithmetic[11]. As a result, the model checking

[5]Note that a scalar is always considered as a vector of length 1 in such logic.

problem of ReTL is at least as hard as the SATISFIABILITY problem of Tarski arithmetic.

Recall that the SATISFIABILITY of Tarski arithmetic is doubly exponential in the size of the formula, and thus doing model checking upon the whole logic leads to a prohibitively high computation cost. To circumvent this, we need to confine ReTL to a particular sub-logic, and we first need to give a classification of ReTL.

For a ReTL formula φ, we can equivalently write it into the negative normal form:

$$\exists\!\!\!\forall_1 \boldsymbol{x}_1 \in \mathbb{R}^{d_1}.\exists\!\!\!\forall_2 \boldsymbol{x}_2 \in \mathbb{R}^{d_2}.\cdots\exists\!\!\!\forall_n \boldsymbol{x}_n \in \mathbb{R}^{d_n}.\psi,$$

where each $\exists\!\!\!\forall_i$ is either $\forall$ or $\exists$, and ψ is a quantifier-free formula. Depending on the nesting of quantifiers, we define a hierarchy of ReTL formulas.

- $\Delta = \Sigma_0^m = \Pi_0^m$ for all m, and this set consists of all quantifier free formulas.
- If $\phi \in \Sigma_k^m$, then $\exists \boldsymbol{x} \in \mathbb{R}^n.\phi \in \Sigma_k^{m+1}$ and $\forall \boldsymbol{x} \in \mathbb{R}^n.\phi \in \Pi_{k+1}^1$.
- If $\phi \in \Pi_k^m$, then $\exists \boldsymbol{x} \in \mathbb{R}^n.\phi \in \Sigma_{k+1}^1$ and $\forall \boldsymbol{x} \in \mathbb{R}^n.\phi \in \Pi_k^{m+1}$.

To simplify the notation, let

$$\Sigma_n = \bigcup_{\substack{m\leqslant n\\ k\in\mathbb{N}}} \Sigma_m^k \cup \bigcup_{\substack{m<n\\ k\in\mathbb{N}}} \Pi_m^k \cup \Delta,$$

$$\Pi_n = \bigcup_{\substack{m\leqslant n\\ k\in\mathbb{N}}} \Pi_m^k \cup \bigcup_{\substack{m<n\\ k\in\mathbb{N}}} \Sigma_m^k \cup \Delta,$$

and in this paper, we are concerned about the set $\Sigma_2 \cup \Pi_2$. Indeed, quantifier-nesting is the essential cause of difficulty in logic, and formulas become difficult to understand if the alternation depth of quantifiers is greater than 2. In addition, this set is powerful enough to express properties making sense w.r.t. ReLU networks.

As an example, the following Π_1^1 formula:

$$\forall \boldsymbol{x} \in \mathbb{R}^{col(\boldsymbol{M}_0)}.\ \boldsymbol{x}[2] > 5 \rightarrow \bigwedge_{k=2}^{row(\boldsymbol{b}_{\ell-1})} (\bigcirc^{\ell}\boldsymbol{x}[1] > \bigcirc^{\ell}\boldsymbol{x}[k]),$$

just declares that if the second element (feature) of a sample is greater than 5, then this sample must belong to the first class.

In [22], the authors summarized four types of crucial properties considered when verifying (deep) ReLU neural networks, namely, the local robustness, reachability property, interval property and Lipschizian property. As we have described in Section 1, the first three kinds of properties can be directly described with some $\Sigma_2 \cup \Pi_2$ formulas. However, the definition of Lipschizian property requires p-norms, which is in general non-linear except for the case of $p = 1$ or $p = \infty$ (in Section 7, we will give some examples).

To perform model checking against $\Sigma_2 \cup \Pi_2$ formulas, we first need a translator $\mathscr{T}_{\mathcal{N},i}$, which converts the quantifier-free part into a boolean combination of pure algebraic formulas — that means, the translator eliminates the $\bigcirc$ and/or index operator, and temporal operators in the terms and formulas.

For the sake of brevity, we directly write $\mathscr{T}_{\mathcal{N},i}$ as $\mathscr{T}_i$ since $\mathcal{N}$ is given in the context. Inductively, for the terms:

- $\mathscr{T}_i(\boldsymbol{t}) = \boldsymbol{t}$ if $\boldsymbol{t}$ is a vector constant or a vector variable.
- For $\mathscr{T}_i(\bigcirc^k\boldsymbol{t})$, we require that $k + i \leqslant \ell$, and:
  i) if $k = 0$, then $\mathscr{T}_i(\bigcirc^k\boldsymbol{t}) = \boldsymbol{t}$;
  ii) If $k > 0$ and $k + i < \ell$, then $\mathscr{T}_i(\bigcirc^k\boldsymbol{t})$ is $\mathrm{relu}(\boldsymbol{M}_{k+i-1}\mathscr{T}_i(\bigcirc^{k-1}\boldsymbol{t}) + \boldsymbol{b}_{k+i-1})$;
  iii) when $k > 0$ and $k + i = \ell$, then $\mathscr{T}_i(\bigcirc^k\boldsymbol{t})$ is just $\boldsymbol{M}_{\ell-1}\mathscr{T}_i(\bigcirc^{k-1}\boldsymbol{t}) + \boldsymbol{b}_{\ell-1}$.
- $\mathscr{T}_i(\boldsymbol{t}[j]) = \boldsymbol{e}_j^{\mathrm{T}}\mathscr{T}_i(\boldsymbol{t})$, where $\boldsymbol{e}_j$ is the vector with $row(\boldsymbol{e}_j) = row(\boldsymbol{t})$, its j-th element is 1 and other elements are all 0.
- $\mathscr{T}_i(\boldsymbol{M}\boldsymbol{t}) = \boldsymbol{M}\mathscr{T}_i(\boldsymbol{t})$.
- $\mathscr{T}_i(\boldsymbol{t}_1 + \boldsymbol{t}_2) = \mathscr{T}_i(\boldsymbol{t}_1) + \mathscr{T}_i(\boldsymbol{t}_2)$.

For the quantifier-free formulas, we have:

- $\mathscr{T}_i(\top) = \top$, and $\mathscr{T}_i(\bot) = \bot$;
- $\mathscr{T}_i(\boldsymbol{t}_1 \sim \boldsymbol{t}_2) = \mathscr{T}_i(\boldsymbol{t}_1) \sim \mathscr{T}_i(\boldsymbol{t}_2)$;
- $\mathscr{T}_i(\neg\varphi) = \neg\mathscr{T}_i(\varphi)$ and $\mathscr{T}_i(\varphi_1 \vee \varphi_2) = \mathscr{T}_i(\varphi_1) \vee \mathscr{T}_i(\varphi_2)$;
- $\mathscr{T}_i(\mathsf{X}\varphi) = \mathscr{T}_{i+1}(\varphi)$;
- $\mathscr{T}_i(\varphi_1\mathsf{U}\varphi_2) = \bigvee_{k=i}^{\ell+1}\left(\mathscr{T}_k(\varphi_2) \wedge \bigwedge_{j=i}^{k-1}\mathscr{T}_j(\varphi_1)\right)$.

Actually, the translator encodes the information of the model (i.e., the ReLU network), and the ReTL formula finally turns into the form:

$$\exists\!\!\!\forall_1 \boldsymbol{x}_1 \in \mathbb{R}^{d_1}.\exists\!\!\!\forall_2 \boldsymbol{x}_2 \in \mathbb{R}^{d_2}.\cdots\exists\!\!\!\forall_n \boldsymbol{x}_n \in \mathbb{R}^{d_n}.\mathscr{T}_0(\psi).$$

Now, given a Π_2 formula φ, w.l.o.g., we just suppose its normal form is

$$\forall \boldsymbol{y}_1 \in \mathbb{R}^{d_1}.\cdots\forall \boldsymbol{y}_n \in \mathbb{R}^{d_n}.\exists \boldsymbol{x}_1 \in \mathbb{R}^{k_1}.\cdots\exists \boldsymbol{x}_m \in \mathbb{R}^{k_m}.\psi,$$

where $\psi = \psi_1 \vee \ldots \vee \psi_t$, and each quantifier free formula ψ_i is in positive normal form and is $\vee$-free. Bewaring that in this system we have vectorized relation such as $\neq$, $\ngeqslant$, $\ngtr$. Thanks to the fact that we are

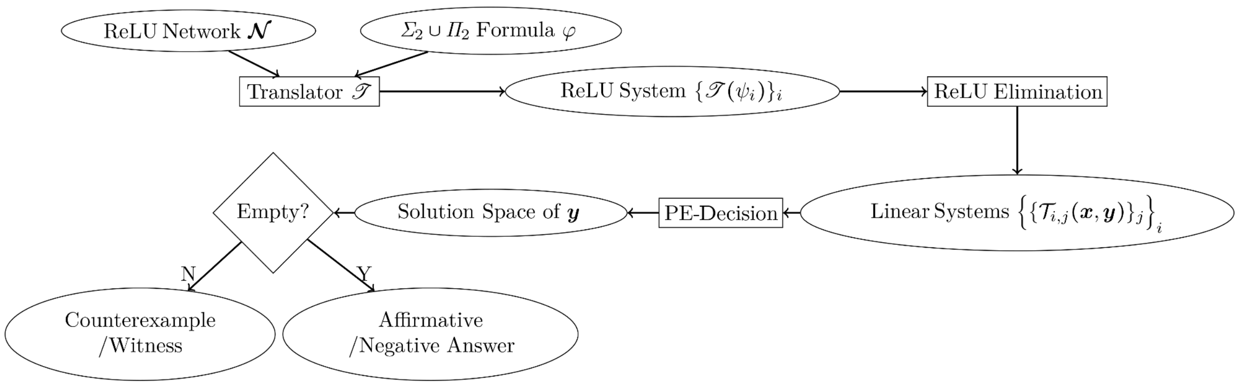
ReLU Network $\boldsymbol{\mathcal{N}}$
$\Sigma_2 \cup \Pi_2$ Formula φ
Translator $\mathscr{T}$
ReLU System $\{\mathscr{T}(\psi_i)\}_i$
ReLU Elimination
Linear Systems $\left\{\{\mathcal{T}_{i,j}(\boldsymbol{x},\boldsymbol{y})\}_j\right\}_i$
PE-Decision
Solution Space of $\boldsymbol{y}$
Empty?
N
Y
Counterexample /Witness
Affirmative /Negative Answer

Fig.1. Framework for $\Sigma_2 \cup \Pi_2$ formula model checking.

aware of the width of each layer, we can decompose it into scale version in the previous step. For example, if the width of the second level is k, then when interpreting the formula $\boldsymbol{v} \neq \boldsymbol{t}$, we can first write it as $\bigvee_{i=1}^{k}((\boldsymbol{v}[i] > \boldsymbol{t}[i]) \vee (\boldsymbol{v}[i] < \boldsymbol{t}[i]))$ before imposing $\mathscr{T}_1$ on it. Also, we can eliminate $<$ and $\leqslant$ in the formula (e.g., $\boldsymbol{x} \leqslant \boldsymbol{v}$ is transformed into $-\boldsymbol{x} \geqslant -\boldsymbol{v}$).

Once the formula is translated into the canonical form, we can perform the verification via the following steps (cf. Fig.1).

• First, we introduce two new vector variables $\boldsymbol{y} \in \mathbb{R}^{\sum_{i=1}^{n} d_i}$ and $\boldsymbol{x} \in \mathbb{R}^{\sum_{j=1}^{m} k_j}$, and each $\boldsymbol{x}_j$ and $\boldsymbol{y}_i$ can be represented by $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. For example, we have

$$\boldsymbol{y}_i = [[0]_{(\sum_{j<i} d_j)\times d_i} \vdots \boldsymbol{I}_{d_i} \vdots [0]_{(\sum_{j>i} d_j)\times d_i}] \cdot \boldsymbol{y}.$$

Intuitively, $\boldsymbol{x}$ and $\boldsymbol{y}$ are concatenations of all $\boldsymbol{x}_j$s and all $\boldsymbol{y}_i$s, respectively. By denoting $\tilde{\psi}_i$ obtained from ψ_i via doing such replacement, φ can be equivalently written as

$$\forall \boldsymbol{y} \in \mathbb{R}^{\sum_{i=1}^{n} d_i}.\exists \boldsymbol{x} \in \mathbb{R}^{\sum_{j=1}^{m} k_j}.(\tilde{\psi}_i \vee \ldots \vee \tilde{\psi}_t).$$

• Then, each $\mathscr{T}_0(\tilde{\psi}_i)$ corresponds to a ReLU-system. After doing ReLU-elimination with the technique we introduced in Subsection 4.2, each $\mathscr{T}_0(\tilde{\psi}_i)$ derives finitely many systems of equality/inequalities $\{\mathcal{T}_{i,j}\}_j$.

• Observe that $\mathcal{N} \not\models \varphi$ iff there exists some vector $\boldsymbol{y}$ in $\mathbb{R}^{\sum_{i=1}^{n} d_i}$ making no proper $\boldsymbol{x} \in \mathbb{R}^{\sum_{j=1}^{m} k_j}$ fulfill $\mathcal{T}_{i,j}$ for every i and every j. Therefore, this is precisely the question "whether the common part of PE problems of all $\mathcal{T}_{i,j}(\boldsymbol{x},\boldsymbol{y})$ is non-empty".

• Since the solution of each $\mathcal{T}_{i,j}(\boldsymbol{x},\boldsymbol{y})$ is an FRS, as we discussed in Subsection 4.1, the emptiness judging of the intersection is definitely decidable. In addition, each element belonging to this intersection yields a counterexample of this formula.

Also note that when φ is a Σ_2 formula, then $\neg\varphi$ is Π_2, and we have $\mathcal{N} \models \varphi$ iff $\mathcal{N} \not\models \neg\varphi$ by definition. Actually, in this situation, a counterexample of $\neg\varphi$ is in general called a witness for φ.

6.2 Analysis of Complexity

We now give an analysis on the complexity of ModelChecking against $\Sigma_2 \cup \Pi_2$ formulas. Recalling that in Subsection 6.1, the algorithm framework is given as followings.

1) For the first step, the cost of combining $\mathcal{N}$ and φ together is polynomial in the size of the both inputs. Particularly, we can gain a linear translation if φ is U-free.

2) Subsequently, we encounter an exponential blow-up when doing ReLU-elimination. As we have discussed in Subsection 4.2, each linear system corresponds to a subset of the neural cells.

3) Then, we need to conduct the PE-decision towards the resulting linear system set. This step results in a polynomial space cost in the number of linear systems. We will give a further explanation below.

4) Performing emptiness judgment requires polynomial times w.r.t. the input, as we have pointed in Subsection 4.2.

As a result, we can conclude that the ModelChecking problem against $\Sigma_2 \cup \Pi_2$ ReTL formulas can be solved in EXPSPACE. For this, we need give a detailed analysis of the third step.

Let $\mathcal{V}_{i,j}$ be the solution of $\mathcal{T}_{i,j}(\boldsymbol{x},\boldsymbol{y})$. Then, according to Theorem 11, we see that the complementary space of each $\mathcal{V}_{i,j}$ (namely, $\mathbb{R}^{row(\boldsymbol{y})} - \mathcal{V}_{i,j}$, for the sake of simplicity, we just denote it $\overline{\mathcal{V}_{i,j}}$) is a polyhedral cone.

Remind that we intend to test whether $\bigcap_{i,j} \mathcal{V}_{i,j} = \emptyset$. Since $\mathcal{V}_{i,j} = \overline{\overline{\mathcal{V}_{i,j}}}$ and each $\overline{\mathcal{V}_{i,j}}$ is a polyhedral cone, it must be an FRS associated with a set of linear inequality constraints. For convenience, we assume that each FRS is generated via a vector set containing a complete base of $\mathbb{R}^{row(\boldsymbol{y})}$—this always can be achieved because we can introduce extra 0 coefficients in the constraints. Since a vector belongs to $\mathcal{V}_{i,j}$ iff its coefficients violet some constraints for $\overline{\mathcal{V}_{i,j}}$, then, for each polyhedral cone in the set $\{\overline{\mathcal{V}_{i,j}}\}_{i,j}$, we can pick one constraint and then dualize (or negate) it, and we obtain a linear (inequality) system consisting of all these (negated) constraints, together with the intersection-condition (see the below example).

Due to the arbitrariness of the condition selection, we may obtain different systems. However, if $\bigcap_{i,j}\mathcal{V}_{i,j}=\emptyset$, then the solution of each such system must be $\emptyset$. In other words, we may stop the verification procedure once some non-empty solution space is detected, and this provides us the chance for an early-stop.

Just consider the following illustrative example. Supposing that we have three spaces $\mathcal{V}_i$ $(i=1,2,3)$ —for the sake of brevity, we here do not use double-index for a space. We suppose that each $\overline{\mathcal{V}_i}$ is a polyhedral cone generated by the vector set $\{\boldsymbol{v}_{i,j}\}_{j=1}^{3}$ and coefficients $\{c_{i,j}\}_{j=1}^{3}$. In addition, we suppose that the constraints for $\overline{\mathcal{V}_i}$ are given as (10)–(12), respectively.

$$\begin{cases} 3c_{1,1}+2c_{1,2}\geqslant 1, \\ 2c_{1,2}+c_{1,3}>4, \end{cases} \tag{10}$$

$$c_{2,1}+2c_{2,2}+4c_{3,2}=3, \tag{11}$$

$$\begin{cases} c_{3,1}+2c_{3,2}\neq 1, \\ c_{3,1}+c_{3,2}\geqslant 2, \\ 2c_{3,2}+c_{3,3}=0. \end{cases} \tag{12}$$

First, we establish the intersection-condition, expressed as

$$\begin{cases} \sum_{j=1}^{3} c_{1,j}\boldsymbol{v}_{1,j}=\sum_{j=1}^{3} c_{2,j}\boldsymbol{v}_{2,j}, \\ \sum_{j=1}^{3} c_{1,j}\boldsymbol{v}_{1,j}=\sum_{j=3}^{3} c_{3,j}\boldsymbol{v}_{3,j}, \end{cases} \tag{13}$$

and one can easily transform it into the scalar form. Subsequently, from each of (10)–(12), we need to choose one constraint and negate it, and to form a new system. Remind that (11) can be equivalently formulated as[⑥]

$$\begin{cases} c_{2,1}+2c_{2,2}+4c_{3,2}\geqslant 3, \\ c_{2,1}+2c_{2,2}+4c_{3,2}\leqslant 3, \end{cases}$$

so does the last constraint in (12). Then, if we select the first constraint in each system, we can establish the following sub system via negating each of them:

$$\begin{cases} 3c_{1,1}+2c_{1,2}<1, \\ c_{2,1}+2c_{2,2}+4c_{3,2}<3, \\ c_{3,1}+2c_{3,2}=1. \end{cases} \tag{14}$$

Then, we obtain a system of inequality (13)∪(14). Definitely, for the subsystem (14), we have $2\times(1+1)\times(2+2)=16$ possible choices, and we can immediately report a counterexample (or witness) if any of them has a non-empty solution when jointing with (13).

This explains why we can perform model checking within exponential space. At one moment, we just need to storage and handle one inequality system, and enumerate the next one followed by the lexicographical order if the solution is empty. For example, in the previous demo, the subsystem (14) just corresponds to the index $(1,1,1)$. It is clear that both the sizes of the system and an index are exponential in the number of constraints, which is polynomial in the size of the network.

Note that the above analysis is for the whole $\varSigma_2\cup\varPi_2$ fragment. Indeed, the complexity of ModelChecking goes lower for formulas within smaller hierarchies. First of all, for a $\varDelta$-formula ψ, the model checking problem is trivial, and one just needs to do a simple computing and judge if $\mathscr{T}_0(\psi)$ holds. Hence, it can be done within polynomial time. In addition, for the $\varSigma_1\cup\varPi_1$ fragment, this problem is PSPACE-complete. The reason is: w.l.o.g., for a $\varSigma_1$ formula, one just needs to store and deal with an instanced linear system each time (though the number of such systems is exponential), and the model checking procedure stops once one such system is detected to have a feasible solution.

7 Experimental Demonstration

We have implemented a prototype toolkit on ReTL ModelChecking. As an attempt on neural network model checking, for the time being, we do not intend to compete with other verification approaches, such as SMT-based verification[8, 15]. Rather, we are mainly concerned about the effectiveness of the proposed approach, because the $\varSigma_2\cup\varPi_2$ fragment is beyond the expressiveness of other existing work.

In this section, we also exemplify how ReTL formulas are used to designate properties in ReLU network verification. For the sake of convenience, we fix six randomly-generated models (i.o.w., ReLU networks) for the first four experiments. Each model is a two-layered

⑥ As a matter of personal taste, one can equally write the second inequality into the normal form $-c_{2,1}-2c_{2,2}-4c_{3,2}\geqslant -3$.

NN; hence it has four parameters $\boldsymbol{M}_0$, $\boldsymbol{b}_0$, $\boldsymbol{M}_1$, $\boldsymbol{b}_1$. The first three models (INT_1 – INT_3) have integer parameters, whereas other three networks (FLT_1 – FLT_3) are built up with float parameters. With these, one can make sure about the correctness of the results. Detailed parameters are given in Appendix.

The first experiment is to justify the feasibility of the following linear system:

$$\begin{cases} x_1 + x_2 + x_3 = 25, \\ 5x_1 + 3x_2 + 2x_3 = 0, \\ x_2 - x_3 = 6, \end{cases}$$

which can be captured via the Σ_1 ReTL formula:

$$\exists \boldsymbol{x} \in \mathbb{R}^3 . \boldsymbol{M}\boldsymbol{x} = (25, 0, 6)^{\mathrm{T}},$$

where $\boldsymbol{M} = \begin{pmatrix} 1 & 1 & 1 \\ 5 & 3 & 2 \\ 0 & 1 & -1 \end{pmatrix}$. Table 1 lists some experimental results of this specification upon six randomly generated networks. In this table, the "Result" column contains the information about if the specification is satisfied ("T") or not ("F"). It can be seen that this system has a unique solution $(-131/5, 143/5, 113/5)^{\mathrm{T}}$. Then, on any ReLU network, we ought to obtain an affirmative answer accompanied with a witness (i.e., the solution). We are also concerned about the time and total memory used in doing verification, which are given in the third and the fourth columns, respectively.

Table 1. ModelChecking Results on Feasibility of Some Given Systems

Model (NN)	Result	Time (s)	Memory (MB)
INT_1	T	0.716	100.9
INT_2	T	0.719	101.2
INT_3	T	0.720	99.2
FLT_1	T	0.823	99.6
FLT_2	T	0.818	101.6
FLT_3	T	0.813	101.6

The second experiment is related to adversarial examples. We take

$$\exists \boldsymbol{x} \in \mathbb{R}^2 . \, (\boldsymbol{x}[1] < 5) \wedge \left(\bigcirc^2 \boldsymbol{x}[1] = \bigcirc^2 \boldsymbol{x}[2] \right),$$

as the specification. Intuitively, it asserts that there is some adversarial example whose first feature is less than 5. Experimental results are given in Table 2. In this table, the last column provides the corresponding witness whenever the specification is satisfied. If no such witness exists (or counterexample), we use a dash sign (–) to denote it.

Next, we would perform some verification about the interval property and the local robustness property.

When taking a ReLU NN as a non-linear mapping, for a given interval $\mathcal{I} \subseteq [0,1]^{col(\boldsymbol{M}_0)}$, the obligation is to detect the range of the output. Let us consider the following Π_1 formula:

$$\forall \boldsymbol{x} \in \mathbb{R}^2 . (\boldsymbol{x} \in [0,1]^2 \rightarrow \bigcirc^2 \boldsymbol{x}[1] < 20),$$

where $\boldsymbol{x} \in [0,1]^2$ is the abbreviation of

$$(0 \leqslant \boldsymbol{x}[1] \leqslant 1) \wedge (0 \leqslant \boldsymbol{x}[2] \leqslant 1).$$

Experimental results of interval property are given in Table 3. Same as before, a counterexample is provided whenever the corresponding property is not satisfied.

Table 2. ModelChecking Results on Adversarial Examples

Model (NN)	Result	Time (s)	Memory (MB)	Witness
INT_1	T	0.824	101.6	$(-0.351\,351\,351\,325\,961, -1.729\,729\,735\,531\,42)^{\mathrm{T}}$
INT_2	T	0.823	102.0	$(0.097\,826\,087\,133\,160\,7, 0.619\,565\,217\,438\,888)^{\mathrm{T}}$
INT_2	F	1.126	103.3	–
FLT_1	T	1.124	103.1	$(-2\,476\,534\,218.440\,25, -60\,809\,664.070\,371\,6)^{\mathrm{T}}$
FLT_2	T	2.055	102.1	$(0.683\,433\,694\,692\,267, 1.342\,290\,326\,952\,93)^{\mathrm{T}}$
FLT_3	T	0.918	102.5	$(-0.250\,215\,405\,774\,543, 0.366\,012\,022\,437\,047)^{\mathrm{T}}$

Table 3. ModelChecking Results on Interval Property

Model (NN)	Result	Time (s)	Memory (MB)	Counterexample
INT_1	T	1.217	102.3	–
INT_2	T	1.234	102.5	–
INT_3	T	1.124	101.9	–
FLT_1	F	1.230	102.7	$(1.745\,688\,9 \times 10^{-10}, 9.461\,576\,43 \times 10^{-14})^{\mathrm{T}}$
FLT_2	T	1.332	103.2	–
FLT_3	F	0.924	101.4	$(3.841\,213\,6 \times 10^{-10}, 0.089\,040\,15)^{\mathrm{T}}$

Table 4. ModelChecking Results on Local Robustness

ϵ	$\boldsymbol{x}$	Result	Time (s)	Memory (MB)	Counterexample
1.000	$\boldsymbol{x}_0$	F	0.819	101.3	$(-0.080\,645\,135\,688\,914\,4, -0.999\,999\,985\,356\,191)^{\mathrm{T}}$
					$(-0.515\,151\,450\,346\,327, -0.606\,060\,402\,135\,362)^{\mathrm{T}}$
	$\boldsymbol{x}_1$	T	1.230	102.5	–
	$\boldsymbol{x}_2$	T	1.335	102.8	–
0.100	$\boldsymbol{x}_0$	F	0.921	101.7	$(0.018\,888\,888\,915\,250\,9, -0.099\,999\,999\,503\,212\,4)^{\mathrm{T}}$
					$(-0.099\,999\,999\,227\,925\,3, 0.016\,666\,667\,872\,800\,7)^{\mathrm{T}}$
	$\boldsymbol{x}_1$	T	1.324	102.9	–
	$\boldsymbol{x}_2$	T	1.333	102.9	–
0.050	$\boldsymbol{x}_0$	F	1.019	101.4	$(0.020\,555\,597\,186\,598\,3, -0.049\,999\,999\,996\,716\,5)^{\mathrm{T}}$
					$(-0.049\,999\,999\,970\,305\,2, -0.049\,999\,999\,966\,117\,3)^{\mathrm{T}}$
	$\boldsymbol{x}_1$	T	1.225	102.5	–
	$\boldsymbol{x}_2$	T	1.315	102.6	–
0.025	$\boldsymbol{x}_0$	F	0.922	101.6	$(0.021\,388\,952\,688\,033\,3, -0.049\,999\,999\,996\,716\,5)^{\mathrm{T}}$
					$(-0.049\,999\,999\,970\,305\,2, -0.499\,999\,999\,661\,173)^{\mathrm{T}}$
	$\boldsymbol{x}_1$	T	1.223	102.6	–
	$\boldsymbol{x}_2$	T	1.333	103.0	–
0.010	$\boldsymbol{x}_0$	T	1.224	102.7	–
	$\boldsymbol{x}_1$	T	1.227	102.6	–
	$\boldsymbol{x}_2$	T	1.231	102.7	–
0.001	$\boldsymbol{x}_0$	T	1.230	102.8	–
	$\boldsymbol{x}_1$	T	1.231	102.8	–
	$\boldsymbol{x}_2$	T	1.325	102.9	–

To verify local robustness, we fix the model to be INT_1, and let

$$\varphi_i = \forall \boldsymbol{y} \in \mathbb{R}^2.(\|\boldsymbol{x} - \boldsymbol{y}\|_\infty < \epsilon \to \bigcirc^2 \boldsymbol{y}[3-i] \leqslant \bigcirc^2 \boldsymbol{y}[i]),$$

for $i = 1, 2$, where $\boldsymbol{x} \in \mathbb{R}^2$ and ϵ are two parameters, and then let $\varphi = \varphi_1 \vee \varphi_2$ as the specification. Remind that $\|\boldsymbol{x} - \boldsymbol{y}\|_\infty < \epsilon$ is just abbreviation of

$$(\epsilon > (\boldsymbol{x}[1] - \boldsymbol{y}[1]) > -\epsilon) \wedge (\epsilon > (\boldsymbol{x}[2] - \boldsymbol{y}[2]) > -\epsilon).$$

In this experiment, we in turn let ϵ be 1, 0.1, 0.05, 0.025, 0.01 and 0.001, and we let $\boldsymbol{x}$ be $(0, 0)^{\mathrm{T}}$, $(410.731, 46.487)^{\mathrm{T}}$ and $(-305.796, 189.393)^{\mathrm{T}}$, respectively — the last two values are randomly sampled. For convenience, we denote by these three points $\boldsymbol{x}_0$, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively. Experimental results are shown in Table 4. Note that each counterexample consists of two vectors — the first is for φ_1 whereas the second is for φ_2.

To perform the verification against $\Sigma_2 \cup \Pi_2$ formulas, we choose the boundedness property to do the experiment. We use the Π_2 formula:

$$\forall \boldsymbol{x} \in \mathbb{R}^2.\exists c \in \mathbb{R}.\boldsymbol{x} \in [0, 1]^2 \to \bigcirc^2 \boldsymbol{x}[1] \leqslant c,$$

as the specification. Without loss of generality, we here just concern about the first dimension of the output.

We randomly generate ReLU NNs with one hidden-layer, and we make the number of neuron cells in the hidden layer vary. Both the input-layer and the output-layer are set to be 2-dimensional. Experimental results are shown in Table 5, where all outputs are affirmative. We can see that the model checker reports "time out" when there are more than seven cells in the hidden layer[7]. The cost of memory is not really exponential in the size of the model. The reason might be that our analysis just considers the worst case.

8 Related Work

We discuss existing work on formal verification of neural networks (cf. [22] for survey). We exclude studies on neural network testing (e.g., [23–26] to cite a few), which are computationally less expensive and therefore are able to work with large neural networks. However, the cost of finding the provable guarantees in testing is large, which is similar to traditional software testing against software verification.

[7] We set the time bound to be 1 hour.

Table 5. ModelChecking Results for Boundedness Verification

Number of Neuron Cells in Hidden Layer	Time (s)	Memory (MB)
3	1.02	101.80
4	6.02	103.25
5	45.89	105.10
6	353.00	117.37
7	1 547.35	154.60
8	Time out	Time out

Various verification techniques have been proposed to solve local robustness, reachability property, interval property and/or Lipschizian property. Some of them are exact while most of them are approximate. The exact techniques are achieved by reducing the verification problem into a set of constraints (with or without optimization objectives), so that they can be solved with constraint solvers [8, 15, 27], mixed integer linear programming (MILP) solvers [16, 21, 28]. For instance, in [15], Pulina and Tacchella proposed an abstraction-refinement approach based on SMT solving for checking interval properties. Katz *et al.* proposed efficient decision procedures tailored to SMT constraints obtained from the verification of neural networks [8]. Narodytska *et al.* proposed SAT-based techniques to verify properties of binarized neural networks in which both weights and activations are binary [27]. Lomuscio and Maganti proposed to reduce the reachability problem of neural networks MILP-solving [21]. Cheng *et al.* proposed a parallelized algorithm to speed up MILP-solving [28]. Further optimization was proposed by Dutta *et al.*, which integrates gradient descent based method to efficiently calculate the output range [16]. Bunel *et al.* rephrased the verification problem of boolean formulas of linear inequalities over piecewise-linear neural networks as a Branch-and-Bound optimization problem and claimed that both SMT-based and MILP-based approaches can be regarded as its special cases [29]. Recently, Tran *et al.* proposed an exact verification technique for solving reachability, by symbolically representing states via star sets and computing reachable states layer-by-layer [30].

There also exist approximate verification techniques which are either sound but incomplete, or complete but unsound. One of such techniques is abstract interpretation, a theory of sound over-approximation of the semantics of computer programs [31]. Abstract interpretation has been explored in some work to verify interval properties of neural networks [32]. Other approximate techniques include optimisation [14] and search-based methods [13, 18], interval/bound analysis [33]. Though these approximate techniques are more scalable and efficient than the exact techniques, they may lead to false negative and/or false positive cases. Singh *et al.* combined abstract interpretation and MILP-based approaches with a refinement step which aims to boost verification by bringing the best of both worlds [34]. In [35], Wan *et al.* reduced the local robustness verification problem to a series of sub-problems to boost verification, where each sub-problem considers only one class.

9 Conclusions

In this paper, we presented the temporal logic ReTL to specify classification-related properties upon ReLU networks. We showed that the ModelChecking problem of $\Sigma_2 \cup \Pi_2$ ReTL formulas is decidable in EXPSPACE. To achieve this, we presented the so-called ReLU-elimination technique and the PE-judging approach.

We employed arithmetic operators on terms and temporal connectives on formulas to define the logic ReTL. Since the structure of standard ReLU network is flat, the U operator, as we have pointed out, can be encoded with X w.r.t. a given model. However, things will be different once looping structures are involved in the network, e.g., recurrent NNs. For this reason, we still used this temporal operator in the logic.

As the main result, we showed that the $\Sigma_2 \cup \Pi_2$ fragment of ReTL is decidable for ModelChecking, and this logic is powerful enough to express some important properties for ReLU neural networks. In addition, this fragment can be (relatively) effectively verified for most cases.

We also have several future studies to do, such as algorithm optimization, the synthesis problem of ReTL, and a tighter analysis of the complexity.

References

[1] Mcculloch W S, Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943, 5: 115-133.

[2] Salehinejad H, Baarbe J, Sankar S, Barfett J, Colak E, Valaee S. Recent advances in recurrent neural networks. arXiv:1801.01078, 2018. https://arxiv.org/abs/1801.01078, May 2020.

[3] LeCun Y. Generalization and network design strategies. In *Connectionism in Perspective*, Pfeifer R, Schreter Z, Fogelman-Soulié F, Steels L (eds.), Elsevier, 1989, pp.143-155.

[4] Nair V, Hinton G E. Rectified linear units improve restricted Boltzmann machines. In *Proc. the 27th International Conference on Machine Learning*, June 2010, pp.807-814.

[5] Lei N, Luo Z, Yau S, Gu X D. Geometric understanding of deep learning. arXiv:1805.10451, 2018. https://arxiv.org/abs/1805.10451, May 2020.

[6] Clarke E M, Emerson E A. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proc. the 3rd Workshop on Logics of Programs*, May 1981, pp.52-71.

[7] Queille J, Sifakis J. Specification and verification of concurrent systems in CESAR. In *Proc. the 5th Int. Symp. Programming*, April 1982, pp.337-351.

[8] Katz G, Barrett C W, Dill D L, Julian K, Kochenderfer M J. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proc. the 29th Int. Conf. Computer Aided Verification*, July 2017, pp.97-117.

[9] Pnueli A. The temporal logic of programs. In *Proc. the 18th Annual Symp. Foundations of Computer Science*, October 1977, pp.46-57.

[10] Pnueli A, Rosner R. On the synthesis of a reactive module. In *Proc. the 16th Annual ACM Symp. Principles of Programming Languages*, January 1989, pp.179-190.

[11] Manna Z, Zarba C G. Combining decision procedures. In *Proc. the 10th Anniversary Colloquium of the Int. Institute for Software Technology of the United Nations University*, March 2002, pp.381-422.

[12] Davenport J H, Heintz J. Real quantifier elimination is doubly exponential. *J. Symbolic Computation*, 1988, 5(1/2): 29-35.

[13] Huang X, Kwiatkowska M, Wang S, Wu M. Safety verification of deep neural networks. In *Proc. the 29th Int. Conf. Computer Aided Verification*, July 2017, pp.3-29.

[14] Ruan W, Huang X, Kwiatkowska M. Reachability analysis of deep neural networks with provable guarantees. In *Proc. the 27th Int. Joint Conf. Artificial Intelligence*, July 2018, pp.2651-2659.

[15] Pulina L, Tacchella A. An abstraction-refinement approach to verification of artificial neural networks. In *Proc. the 22nd Int. Conf. Computer Aided Verification*, July 2010, pp.243-257.

[16] Dutta S, Jha S, Sankaranarayanan S, Tiwari A. Output range analysis for deep feedforward neural networks. In *Proc. the 10th Int. Symposium NASA Formal Methods*, April 2018, pp.121-138.

[17] Weng T, Zhang H, Chen H, Song Z, Hsieh C, Daniel L, Duane S B, Dhillon I S. Towards fast computation of certified robustness for ReLU networks. In *Proc. the 35th Int. Conf. Machine Learning*, July 2018, pp.5273-5282.

[18] Penrose R. A generalized inverse for matrices. *Mathematical Proc. the Cambridge*, 1955, 51: 406-413.

[19] Penrose R. On the best approximate solution of linear matrix equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 1956, 52(1): 17-19.

[20] Farkas G. über die theorie der einfachen ungleichungen. *J. die Reine und Angewandte Mathematik*, 1902, 124: 1-24. (in German)

[21] Lomuscio A, Maganti L. An approach to reachability analysis for feed-forward ReLU neural networks. arXiv:1706.07351, 2017. https://arxiv.org/abs/1706.07351, May 2020.

[22] Huang X, Kroening D, Ruan W, Sharp J, Sun Y, Thamo E, Wu M, Yi X. Safety and trustworthiness of deep neural networks: A survey. arXiv:1812.08342v4, 2019. https://arxiv.org/abs/1812.08342, April 2020.

[23] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I J, Fergus R. Intriguing properties of neural networks. In *Proc. the 2nd Int. Conf. Learning Representations*, April 2014.

[24] Lei Y, Chen S, Fan L, Song F, Liu Y. Advanced evasion attacks and mitigations on practical ML-based phishing website classifiers. arXiv:2004.06954, 2020. https://arxiv.org/abs/2004.06954, May 2020.

[25] Chen G, Chen S, Fan L, Du X, Zhao Z, Song F, Liu Y. Who is real Bob? Adversarial attacks on speaker recognition systems. arXiv:1911.01840, 2019. https://arxiv.org/abs/1911.01840, May 2020.

[26] Duan Y, Zhao Z, Bu L, Song F. Things you may not know about adversarial example: A black-box adversarial image attack. arXiv:1905.07672, 2019. https://arxiv.org/abs/1905.07672, May 2020.

[27] Narodytska N, Kasiviswanathan S P, Ryzhyk L, Sagiv M, Walsh T. Verifying properties of binarized deep neural networks. In *Proc. the 32nd AAAI Conf. Artificial Intelligence*, February 2018, pp.6615-6624.

[28] Cheng C, Nührenberg G, Ruess H. Maximum resilience of artificial neural networks. In *Proc. the 15th Int. Symp. Automated Technology for Verification and Analysis*, October 2017, pp.251-268.

[29] Bunel R, Turkaslan I, Torr P H S, Kohli P, Mudigonda P K. A unified view of piecewise linear neural network verification. In *Proc. the 32nd Annual Conf. Neural Information Processing Systems*, December 2018, pp.4795-4804.

[30] Tran H, Lopez D M, Musau P, Yang X, Nguyen L V, Xiang W, Johnson T T. Star-based reachability analysis of deep neural networks. In *Proc. the 3rd World Congress on Formal Methods*, October 2019, pp.670-686.

[31] Cousot P, Cousot R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. the 4th ACM Symp. Principles of Programming Languages*, January 1977, pp.238-252.

[32] Gehr T, Mirman M, Drachsler-Cohen D, Tsankov P, Chaudhuri S, Vechev M T. AI^2: Safety and robustness certification of neural networks with abstract interpretation. In *Proc. the 39th IEEE Symp. Security and Privacy*, May 2018, pp.3-18.

[33] Wang S, Pei K, Whitehouse J, Yang J, Jana S. Formal security analysis of neural networks using symbolic intervals. In *Proc. the 27th USENIX Security Symp.*, August 2018, pp.1599-1614.

[34] Singh G, Gehr T, Püschel M, Vechev M. T. Boosting robustness certification of neural networks. In *Proc.the 7th International Conference on Learning Representations*, May 2019.

[35] Wan W, Zhang Z, Zhu Y, Zhang M, Song F. Accelerating robustness verification of deep neural networks guided by target labels. arXiv:2007.08520, 2020. https://arxiv.org/abs/2007.08520, July 2020.

Wan-Wei Liu received his Ph.D. degree in computer science from National University of Defense Technology, Changsha, in 2009. He is an associated professor in College of Computer Science, National University of Defense Technology, Changsha. He is a senior member of CCF. His research interests include theoretical computer science (particularly in automata theory and temporal logic), formal methods (particularly in verification), and software engineering.

Fu Song received his Ph.D. degree in computer science from University Paris-Diderot, Paris, in 2013. From 2013 to 2016, he was a lecturer and an associate research professor at East China Normal University, Shanghai. Since August 2016, he is an assistant professor with ShanghaiTech University, Shanghai, and he was a recipient of EASST Best Paper Award at ETAPS 2012. His research interests include software engineering, formal methods and computer security, especially about automata, logic, model checking, and program analysis.

Tang-Hao-Ran Zhang received his Bachelor's degree in computer science from National University of Defense Technology, Changsha, in 2020. He is currently an M.Sc. student in National University of Defense Technology, Changsha. His research interests include model checking, code recommendation, and software analytics.

Ji Wang received his Ph.D. degree in computer science from National University of Defense Technology, Changsha, in 1995. He is currently a full professor in National University of Defense Technology, Changsha, and he is a distinguished member of CCF. His research interests include software engineering and formal methods.

Appendix

A. Parameters of ReLU Networks for Experiments

Parameters of INT_1:

$$\boldsymbol{M}_0 = \begin{pmatrix} 6 & -7 \\ -7 & 1 \\ 7 & 5 \\ -7 & 1 \\ 9 & -6 \end{pmatrix}, \quad \boldsymbol{b}_0 = \begin{pmatrix} -10 \\ -3 \\ 4 \\ 6 \\ 1 \end{pmatrix},$$

$$\boldsymbol{M}_1 = \begin{pmatrix} 5 & 4 & -8 & 4 & -8 \\ -7 & -3 & -4 & -1 & -5 \end{pmatrix}, \quad \boldsymbol{b}_1 = \begin{pmatrix} -10 \\ -1 \end{pmatrix}.$$

Parameters of INT_2:

$$\boldsymbol{M}_0 = \begin{pmatrix} 1 & -5 \\ -8 & -10 \\ -9 & -7 \\ 6 & -7 \\ 9 & -9 \end{pmatrix}, \quad \boldsymbol{b}_0 = \begin{pmatrix} 3 \\ -9 \\ -9 \\ 5 \\ -10 \end{pmatrix},$$

$$\boldsymbol{M}_1 = \begin{pmatrix} -9 & -3 & -2 & -3 & -2 \\ 5 & 0 & 1 & -7 & -8 \end{pmatrix}, \quad \boldsymbol{b}_1 = \begin{pmatrix} -4 \\ 1 \end{pmatrix}.$$

Parameters of INT_3:

$$\boldsymbol{M}_0 = \begin{pmatrix} 9 & -4 \\ 4 & -1 \\ -8 & 7 \\ 6 & 6 \\ -10 & -8 \end{pmatrix}, \quad \boldsymbol{b}_0 = \begin{pmatrix} -9 \\ 6 \\ 0 \\ 0 \\ -10 \end{pmatrix},$$

$$\boldsymbol{M}_1 = \begin{pmatrix} -1 & 0 & -9 & -8 & -10 \\ 0 & 7 & -7 & 9 & -5 \end{pmatrix}, \quad \boldsymbol{b}_1 = \begin{pmatrix} -8 \\ 1 \end{pmatrix}.$$

Parameters of FLT_1:

$$\boldsymbol{M}_0 = \begin{pmatrix} 212.674\,974 & -143.987\,289 \\ -375.440\,552 & 358.204\,367 \\ -132.030\,324 & 346.390\,642 \\ 49.893\,336 & -82.973\,165 \\ 27.624\,272 & -148.681\,754 \end{pmatrix}, \quad \boldsymbol{b}_0 = \begin{pmatrix} 231.989\,271 \\ 478.148\,525 \\ -222.163\,261 \\ 142.263\,284 \\ -475.462\,367 \end{pmatrix},$$

$$\boldsymbol{M}_1 = \begin{pmatrix} -61.013\,296 & 153.505\,166 & -268.398\,733 & 121.688\,212 & -167.487\,403 \\ 262.937\,216 & -307.037\,447 & -444.485\,577 & 76.950\,528 & -179.326\,151 \end{pmatrix}, \quad \boldsymbol{b}_1 = \begin{pmatrix} -106.124\,953 \\ -208.076\,675 \end{pmatrix}.$$

Parameters of FLT_2:

$$\boldsymbol{M}_0 = \begin{pmatrix} -301.002\,477 & -17.324\,118 \\ -336.113\,549 & 122.024\,888 \\ 387.605\,733 & -197.237\,036 \\ -415.715\,576 & 214.777\,717 \\ -188.914\,101 & 216.084\,912 \end{pmatrix}, \quad \boldsymbol{b}_0 = \begin{pmatrix} 228.969\,255 \\ -462.082\,536 \\ -405.481\,405 \\ -428.082\,586 \\ -160.569\,163 \end{pmatrix},$$

$$\boldsymbol{M}_1 = \begin{pmatrix} -207.466\,666 & 216.298\,495 & 415.214\,993 & 462.352\,108 & 347.718\,942 \\ 94.946\,650 & 478.985\,677 & 238.399\,533 & 369.547\,509 & -109.718\,365 \end{pmatrix}, \quad \boldsymbol{b}_1 = \begin{pmatrix} -84.330\,958 \\ 84.719\,959 \end{pmatrix}.$$

Parameters of FLT_3:

$$\boldsymbol{M}_0 = \begin{pmatrix} -94.728\,193 & 434.461\,100 \\ 492.371\,372 & -83.763\,646 \\ -461.663\,281 & 12.307\,470 \\ -234.533\,633 & 489.131\,639 \\ 259.299\,655 & 494.362\,472 \end{pmatrix}, \quad \boldsymbol{b}_0 = \begin{pmatrix} -182.720\,451 \\ 7.458\,327 \\ 284.614\,443 \\ -447.710\,596 \\ -62.244\,129 \end{pmatrix},$$

$$\boldsymbol{M}_1 = \begin{pmatrix} -371.113\,250 & 40.034\,826 & 196.479\,054 & -488.053\,642 & 391.033\,837 \\ 130.401\,299 & 9.545\,044 & 226.463\,981 & -122.851\,226 & 154.483\,804 \end{pmatrix}, \quad \boldsymbol{b}_1 = \begin{pmatrix} -229.505\,807 \\ 368.145\,768 \end{pmatrix}.$$