International Journal on Software Tools for Technology Transfer (2023) 25:301–327
https://doi.org/10.1007/s10009-023-00700-7

EXPLANATION PARADIGMS LEVERAGING ANALYTIC INTUITION

Special Section: Introducing Explanation Paradigms Leveraging Analytic Intuition

Towards rigorous understanding of neural networks via semantics-preserving transformations

Maximilian Schlüter[1] · **Gerrit Nolte**[1] · **Alnis Murtovi**[1] · **Bernhard Steffen**[1]

Accepted: 20 May 2022 / Published online: 30 May 2023

Abstract

In this paper, we present an algebraic approach to the precise and global verification and explanation of *Rectifier Neural Networks*, a subclass of *Piece-wise Linear Neural Networks* (PLNNs), i.e., networks that semantically represent piece-wise affine functions. Key to our approach is the symbolic execution of these networks that allows the construction of semantically equivalent *Typed Affine Decision Structures* (TADS). Due to their deterministic and sequential nature, TADS can, similarly to decision trees, be considered as white-box models and therefore as precise solutions to the model and outcome explanation problem. TADS are linear algebras, which allows one to elegantly compare Rectifier Networks for equivalence or similarity, both with precise diagnostic information in case of failure, and to characterize their classification potential by precisely characterizing the set of inputs that are specifically classified, or the set of inputs where two network-based classifiers differ. All phenomena are illustrated along a detailed discussion of a minimal, illustrative example: the continuous XOR function.

Keywords (Rectifier) neural networks · Activation functions · (Piece-wise) affine functions · Linear algebra · Typed affine decision structures · Symbolic execution · Explainability · Verification · Robustness · Semantics · XOR · Diagnostics · Precision · Digit recognition

1 Introduction

Neural networks are perhaps today's most important machine learning models, with exciting results, e.g., in image recognition [50], speech recognition [9, 11], and even in highly complex games [7, 49, 60]. As the name suggests, neural networks are learned from data using efficient, but approximate training algorithms [30, 46]. At their core, neural networks are (dataflow-oriented) computation graphs [17]. They consist of many computation units, called *neurons*, that are arranged in layers such that computations in each layer can be performed in parallel, with successive layers only depending on the preceding layer. Modern neural networks, in practice, possess up to multiple billions of parameters [9] and leverage parallel hardware such as GPUs to perform computations of this scale [41]. This highly quantitative approach is responsible for exciting success stories, but also for their main weakness: Neural network behavior is often chaotic and hard to comprehend for a human. Perhaps most infamously, a neural network's prediction can change drastically under imperceptible changes to its input, so-called *adversarial examples* [16, 33, 55].

The explainability of neural networks, which are computationally considered as black-boxes due to their highly parallel and non-linear nature, is therefore one of the current core challenges in AI research [14]. The fact that neural networks are increasingly used in safety-critical systems such as self-driving cars [4] turns trustworthiness of machine learning into a must [14]. However, state-of-the-art explanation technology is more about reassuring intuition, e.g., to support cooperative work of humans with AI systems, such as in the field of medical diagnostics [57], than about precise explanation or guarantees [31]. Moreover, current approaches to Neural Network verification are still in their infancy in that they are not yet sufficiently tailored to the nature of Neural Networks to achieve the required scalability or to provide diagnostic information beyond individual witness traces in cases where the verification attempts fail (cf., [6, 28, 61] and Sect. 8 for a more detailed discussion).

✉ B. Steffen
bernhard.steffen@tu-dortmund.de

M. Schlüter
maximilian.schlueter@tu-dortmund.de

G. Nolte
gerrit.nolte@tu-dortmund.de

A. Murtovi
alnis.murtovi@tu-dortmund.de

[1] TU Dortmund University, Dortmund, Germany

In this paper, we present an algebraic approach to the verification and explanation of Rectifier Neural Networks (PLNN), a very popular subclass of neural networks that semantically represent piece-wise affine functions (PAF) [37]. Key to our approach are *Typed Affine Decision Structures* (TADS) that concisely represent PAF in a white-box fashion that is as accessible to human understanding as decision trees. TADS can nicely be derived from PLNNs via symbolic execution [13, 29], or, alternatively, compositionally along the PLNN's layering structure, and their algebraic structure allows for elegant solutions to verification and explanation tasks:

- TADS can be used for PLNNs similarly as *Algebraic Decision Diagrams* (ADDs) have been used for Random Forests in [21] to elegantly provide model and outcome explanations as well as class characterizations.
- Using the algebraic operations of TADS one can not only decide the equivalence problem, i.e., whether two PLNNs are semantically equivalent, but also whether they are ϵ-similar, i.e., never differ more than ϵ. In both cases, diagnostic information in terms of a corresponding 'difference' TADS is provided that precisely specifies where one of these properties is violated.
- TADS comprise non-continuous piece-wise linear operations, which cannot be represented by PLNNs. This is necessary to not only deal with *regression tasks*, where one aims at approximating continuous functions, but also with *classification tasks* with discrete output domains.[1] In the latter case, TADS-based class characterization allows one to precisely characterize the set of inputs that are classified as members of a given class, or the set of inputs where two (PLNN-based) classifiers differ.
- Finally, TADS can also profitably be used for the verification of preconditions and postconditions, the illustration of which is beyond the scope of this paper, but will be discussed in [40] in the setting of digit recognition.

The paper illustrates the essential features of TADS using a minimal, illustrative example: the continuous XOR function. The simplicity of XOR is ideally suited to provide an intuitive entry into the presented theory. A more comprehensive example is presented in [40], where digit recognition based on the MNIST data base is considered. In this highly dimensional setting, specific scalability measures are required to apply our TADS technology.

After specifying the details of our running example in Sect. 2, Sect. 3 sketches Algebraic Decision Structures that later on will be instantiated with Affine Functions recalled in Sect. 4 to introduce the central notion of this paper, Typed Affine Decision Structures (TADS). Semantically, TADS represent piece-wise affine functions, which marks them as a fitting representation for Rectifier Networks that represent continuous piece-wise affine functions[2] and that are discussed in Sect. 5. Our main contribution is the derivation of TADS, using both symbolic execution and compositionality along the layering structure of PLNN, as a complete and precise model explanation of PLNNs. We introduce TADS in Sect. 6 and state important algebraic properties that allow the manipulations mentioned beforehand. Subsequently, Sect. 7 illustrates the impact on verification and explanation of the algebraic properties of TADS that are also established in Sect. 6 along the running example. The paper closes after a discussion of related work in Sect. 8 with conclusions and direction to future work in Sect. 9.

2 Running example – XOR

As a running example throughout this paper, we discuss the XOR function under the perspective of a regression task and a classification task, as specified below. We chose the XOR problem for illustration for two reasons:

- The XOR problem concerns a two-dimensional function, which can be visualized as a function plot.
- While simple, the XOR problem has been a roadblock in early AI research because it cannot be solved by linear approaches [35]. Therefore, it is a minimal problem that still requires a more powerful non-linear model such as a PLNN.

For the following formalizations, let us fix some basic notation: The set of natural numbers (including zero) is denoted $\mathbb{N}$ and the set of real numbers $\mathbb{R}$. Unions and intersections of sets are defined as usual. The cartesian product of two sets is defined as

$$M \times N := \{(x, y) \mid x \in M \wedge y \in N\}.$$

A sequence of cartesian products over the same set may be abbreviated as M^n $(n \geq 0)$, where

$$M^0 = \emptyset \qquad M^1 = M \qquad M^{n+2} = M \times M^{n+1}$$

and the Kleene star operator is defined as

$$M^* := \bigcup_{n \in \mathbb{N}} M^n$$

Moreover, intervals of $\mathbb{R}$ are of the form $[a, b]$ for $a, b \in \mathbb{R}$ with $a \leq b$ and denote the set of all real values between a and b.

As a *regression* task, the XOR problem is stated as follows:

[1] As PLNNs always represent continuous functions, an additional outcome interpretation mechanism is needed to bridge the gap from continuous networks to discrete classification tasks.

[2] Rectifier Networks are often also called Piece-wise Linear Neural Networks, the reason for us to denote them as PLNNs.

Definition 1 (XOR regression)
Find a piece-wise affine function $f\colon [0,1]^2 \longrightarrow [0,1]$ that is continuous and satisfies:

$$f(0,0) \approx 0 \approx f(1,1) \quad \text{and} \quad f(1,0) \approx 1 \approx f(0,1)$$

Thus, a learning algorithm is tasked with approximating a continuous version of an XOR gate, interpolating between the four edge points for which the XOR function is defined.[3]

When posing the XOR problem as a classification task, the XOR function can be regarded as a function with *discrete* binary output 1 or 0 but with a continuous domain $\mathbb{R}^2$.

Definition 2 (XOR classification)
Find a piece-wise affine function $f\colon [0,1]^2 \longrightarrow \{0,1\}$ such that:

$$f(0,0) = 0 = f(1,1) \quad \text{and} \quad f(1,0) = 1 = f(0,1)$$

As the XOR-problem requires fixed values only at four points, there exist infinitely many solutions. This is typical for machine learning problems where only some few points are fixed and others are left for the machine learning model to freely interpolate. Different machine learning models have different principles that dictate this interpolation. For example, concerning PLNNs, the interpolation is (piece-wise) linear.

In line with the principle of Occam's razor [52], humans[4] would optimally solve the XOR-regression problem with a function as simple as:

$$f_*(x,y) = |x - y|$$

A visualisation of f_* can be found in Fig. 1. Similarly, a human would probably choose the following corresponding straightforward extension to the classification problem:

$$g_*(x,y) = \begin{cases} 1 & \text{if } f_*(x,y) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

An illustration of g_* can be found in Fig. 2. It is straightforward to check that these functions solve the XOR-regression and XOR-classification problems optimally in the sense that they match the traditional XOR function at all points where it is defined.

The continuous XOR problems will serve as running examples throughout this work: We will demonstrate different representations of piece-wise linear functions (such as f_*) and transformations between them along the development of our theory, and showcase differences between the manually constructed solutions to the regression and classification tasks and their learned counterparts in Sect. 7.

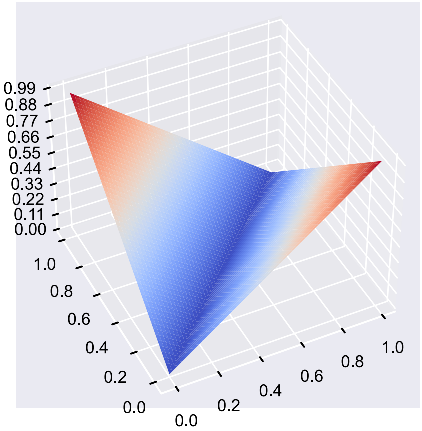


Fig. 1 A baseline solution to the XOR-regression problem given by $f_*(x,y) = |x - y|$. Note that this function is piece-wise linear, having two separate linear regions, which is minimal for the problem

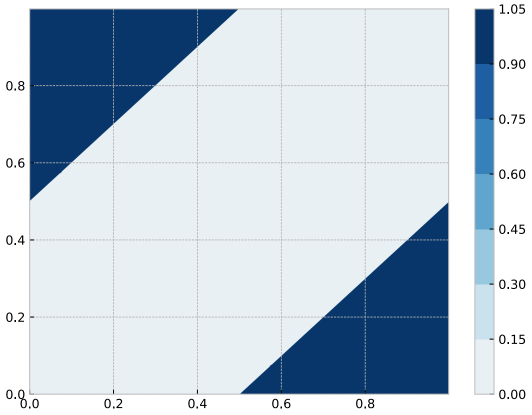


Fig. 2 A baseline solution to the XOR-classification problem given by $g_*(x,y) = 1$ iff $f_*(x,y) \geq 0.5$

3 Algebraic decision structures

In order to prepare the algebraic treatment of decision structures, we focus on decision structures whose leafs are labeled with the elements of an algebra $A = (\mathcal{A}, O)$, so-called *Algebraic Decision Structures* (ADDs). This subsumes the classical case where leafs of decision structures are elements of a set, as these are simply algebras where O is empty. In this section we summarize the definitions and theorems of [21], which are required later in this paper.[5]

Definition 3 (Algebraic structures)
An *algebraic structure*, or *algebra* for short, is a pair $(\mathcal{A}, O)$ of a carrier set $\mathcal{A}$ and a set of operations O. Operations $op \in O$ have a fixed arity and are closed under $\mathcal{A}$.

[3] The restriction to the interval [0, 1] is meant to ease the exposition.

[4] In contrast to, e.g., solutions learned by machines.

[5] Some definitions and theorems were slightly improved or adjusted from [21] for better alignment with the rest of this paper. We omit the proofs for the adjustments because they are straightforward.

In the following, the algebra is identified with its carrier set and both are written calligraphically.

Definition 4 (Algebraic Decision Tree)
An Algebraic Decision Tree (ADT) over the algebra A and the predicates $\mathcal{P}$[6] is inductively defined by the following BNF:

$$T \;::=\; \varepsilon \mid a \mid (p,T,T) \qquad \text{with } a \in \mathcal{A} \text{ and } p \in \mathcal{P}.$$

ADTs are (directed) trees where for a node (p,l,r) the root is given by p and the left and right children by l and r, respectively. ADTs of the form a are leafs with no children and ε denotes the empty tree. We can merge nodes in these ADTs, which leads to the more general Algebraic Decision Structures (ADS):

Definition 5 (Node merging)
Let t be an ADT and t_1 and t_2 be two nodes in t such that t_2 is not reachable from t_1. Then, the two-step transformation of t

- re-route the incoming edges of t_2 to t_1 and
- eliminate all unreachable nodes of t.

is called a t_2 *into* t_1 *merge*.

Node merges aggregate subtrees in a manner that does not create directed cycles. Later, we will define semantics preserving merges.

Definition 6 (Algebraic decision structure)
A rooted directed acyclic graph (DAG) that results from an ADT by a series of node merges is called an Algebraic Decision Structure (ADS). Let $\mathcal{S}_A$ denote the set of all such ADSs over an algebra A.

We can define their semantics inductively.

Definition 7 (Decision structure semantics)
For a set Σ of valuations the semantic function

$$[\![\cdot]\!]_{\mathcal{S}_A} : \mathcal{S}_A \to (\Sigma \to \mathcal{A})$$

for ADSs is inductively defined as

$$[\![a]\!]_{\mathcal{S}_A}(\sigma) := a$$

$$[\![(p,l,r)]\!]_{\mathcal{S}_A}(\sigma) := \begin{cases} [\![l]\!]_{\mathcal{S}_A}(\sigma) & \text{if } [\![p]\!](\sigma) = 1 \\ [\![r]\!]_{\mathcal{S}_A}(\sigma) & \text{if } [\![p]\!](\sigma) = 0 \end{cases}$$

ADS can be considered the universe in which we operate, typically using semantics-preserving transformations. In particular, we will frequently apply *semantic reduction* and *infeasible path reduction,* as discussed in the next two subsections. These two operations reduce the representational overhead of ADS while preserving their semantics.

3.1 Semantic reduction

Semantic functions naturally induce an equivalence relation:

Definition 8 (Semantic equivalence)
Two ADSs t_1 and t_2 are semantically equivalent iff their semantic functions coincide

$$t_1 \sim t_2 \quad \text{iff} \quad [\![t_1]\!]_{\mathcal{S}_A} = [\![t_2]\!]_{\mathcal{S}_A}$$

The following theorem states that one of two different nodes of an ADS that are semantically equivalent is redundant:

Theorem 1 (Semantic reduction)
Let t be an ADS with two nodes t_1 and t_2 that are semantically equivalent, i.e., $t_1 \sim t_2$, and such that t_2 is not reachable from t_1. Moreover, let t_3 be the t_2 into t_1 merge *of t. Then t and t_3 are semantically equivalent, i.e., $t \sim t_3$.*

Our implementation heuristically reduces the number of semantically equivalent nodes of the ADSs. However, in contrast to Algebraic Decision Diagrams [5], which are known for their normal forms, we cannot guarantee canonicity here.

3.2 Vacuity reduction

Typically, there are dependencies between different predicates in $\mathcal{P}$, which induces so-called infeasible paths in the corresponding ADSs. This can be exploited for further reducing ADSs by eliminating so-called *vacuous* predicates:

Definition 9 (Vacuity)
Let $\bar{\mathcal{P}}$ be the set of negated predicates of $\mathcal{P}$. Then we call $\pi = p_0 \cdots p_m \in (\mathcal{P} \cup \bar{\mathcal{P}})^*$ a predicate path.

- π is called a predicate path of a decision structure $t \in \mathcal{S}_A$ iff there exists a path $\pi' = p'_0 \ldots p'_m \in \mathcal{P}^*$ from the root of t to one of its other nodes such that $p_i = p'_i$ in case that π' follows the left/true branch at p_i in t and $p_i = \bar{p}'_i$ otherwise.
  We denote the last predicate $p_m \in \mathcal{P} \cup \bar{\mathcal{P}}$ of π by *final*(π).
- Given a predicate path $\pi = p_0 \cdots p_m$ the predicate *final*(π) is called *vacuous* for π iff the conjunction of the preceding predicates $p_0 \wedge \cdots \wedge p_{m-1}$ in π implies *final*(π).
- Let Π_n be the set of predicate paths of $t \in \mathcal{S}_A$ that end in a given node n. We call n vacuous in t, iff *final*(π) is vacuous for all paths $\pi \in \Pi$ and *final*(π) coincides for all π.

[6] In contrast to ADDs, we do not require an ordering over $\mathcal{P}$ and therefore cannot guarantee canonicity.

- A decision structure $t \in \mathcal{S}_A$ is called *vacuity-free* iff there exists no vacuous node.

This allows us to define the following optimization step.

Definition 10 (Vacuity reduction)
Let $t \in \mathcal{S}_A$ be a decision structure with a vacuous node n and $\mathit{final}(\pi) \in \mathcal{P} \cup \bar{\mathcal{P}}$ be the last predicate of some predicate path π ending in n. Then, re-routing all incoming edges of n to the 'true'-successor of n in case of $\mathit{final}(\pi) \in \mathcal{P}$ and to the 'false'-successor otherwise is called a *vacuity reduction* step.

ADSs, being DAGs, only have finitely many predicate paths which can be effectively analysed for vacuous predicates, as long as the individual predicates are decidable. As the elimination of vacuous predicates is a simple semantics-preserving transformation, we have:

Theorem 2 (Minimality)
Every ADS can be effectively transformed into a semantically equivalent, vacuity-free ADS that is minimal in the sense that any further reduction would change its semantics.

In the remainder of the paper, we will not explicitly discuss the effects of semantic reduction and vacuity reduction. Rather, we will concentrate on the algebraic properties of ADS that they inherit from their leaf algebra via lifting.

3.3 Lifting

It is well-known that algebraic structures $A = (\mathcal{A}, O)$ can point-wise be lifted to cartesian products and arbitrary function spaces $M \longrightarrow A$. This has successfully been exploited for Binary Decision Diagrams (BDDs) and Algebraic Decision Diagrams (ADDs) that canonically represent functions of type $\mathbb{B}^n \longrightarrow \mathbb{B}$ and $\mathbb{B}^n \longrightarrow \mathcal{A}$ respectively. In fact, the canonicity of these representations allows one to identify the BDD/ADD representations directly with their semantics, which in particular reduces the verification of semantic equivalence to checking for isomorphism.

In our case, canonicity is unrealistic for two reasons (cf., Sect. 6.1):

1. Considering predicates rather than Boolean values introduces infeasibility and thereby prohibits minimal canonical representations.
2. The ordering of predicates may lead to an exponential explosion of the representation. Please note that, in contrast to, e.g., the typical BDD setting, we do not have just a few (64, 128, 256,… or the like) input bits that specify the control of some circuit, but predicates capture the effect of the ReLU function in a history-dependent way; Predicates that result from computations in a later layer depend on predicates from earlier layers. Moreover, as predicates are continuous objects, the probability of them coinciding can be considered 0. Thus, ordering predicates would typically lead to representations that are doubly exponential in the number of neurons of a neural network.

We will see, however, that all the algebraic properties we need also hold for unordered ADSs, and that we can conveniently compute on (arbitrary) representatives of the partition defined by semantic equivalence. This way, we arrive at an exponential worst-case complexity (in the size of the argument PLNNs) both, for the algebraic operations and the decision of semantic equivalence.

Although ADSs are not canonical one can effectively apply operators on concrete representatives while preserving semantics. Every operator can be lifted inductively as follows

Definition 11 (Lifted operators)
For every operator $\square : \mathcal{A}^2 \longrightarrow \mathcal{A}$ of an algebra $A = (\mathcal{A}, O)$ we define the lifted operator $\blacksquare : {\mathcal{S}_A}^2 \longrightarrow \mathcal{S}_A$ that operates over ADS inductively as

$$a \blacksquare a' = a \square a'$$

$$a \blacksquare (p, l, r) = (p, a \blacksquare l, a \blacksquare r)$$

$$(p, l, r) \blacksquare t = (p, l \blacksquare t, r \blacksquare t)$$

where $a, a' \in \mathcal{A}$ are ADS identified with an element of the algebra, $t, l, r \in \mathcal{S}_A$ are ADS, and $p \in \mathcal{P}$ is a predicate.

Intuitively, for two ADS t_1 and t_2, this construction replaces leaves in t_1 with copies of t_2. Thus, each path of the resulting ADS t_3 expresses a conjunction of one path in t_1 and one path in t_2. The partition of the domain imposed by all paths of t_2 therefore coincides with the intersection imposed by the intersection of partitions imposed by t_1 and t_2. The required lifting of the operators to leaf nodes is straightforward (cf. Fig. 4 for illustration).

The following theorem which states the correctness of the lifted operators can straightforwardly be proved by induction:

Theorem 3 (Correctness of lifted operators)
Let $t_1, t_2 \in \mathcal{S}_A$ be two ADS over some algebra $A = (\mathcal{A}, O)$. Let $\blacksquare : {\mathcal{S}_A}^2 \longrightarrow \mathcal{S}_A$ denote the lifted version of the operator $\square \in O$. Then the following equation holds for all $\sigma \in \Sigma$:

$$[\![t_1 \blacksquare t_2]\!]_{\mathcal{S}_A}(\sigma) := [\![t_1]\!]_{\mathcal{S}_A}(\sigma) \square [\![t_2]\!]_{\mathcal{S}_A}(\sigma)$$

3.4 Abstraction

Abstraction is one of the most powerful means for achieving scalability. The following easy to prove theorem concerns the interplay of abstractions imposed by a homomorphism of the leaf algebra and their effect on some classification function.

Theorem 4 (Abstraction)
Let $A = (\mathcal{A}, O)$ and $A' = (\mathcal{A}', O')$ be two algebras, and $\alpha\colon A \rightarrow A'$ a homomorphism. Then $\alpha_S\colon \mathcal{S}_A \rightarrow \mathcal{S}_{A'}$ defined by simply applying α to all the leaves of the argument ADS completes the following commutative diagram:

$$\begin{tikzcd}
\mathcal{S}_A \arrow[r, "\alpha_S"] \arrow[d, "{[\![\cdot]\!]_{\mathcal{S}_A}}"] & \mathcal{S}_{A'} \arrow[d, "{[\![\cdot]\!]_{\mathcal{S}_{A'}}}"] \\
\Sigma \rightarrow A \arrow[r, "\alpha"] & \Sigma \rightarrow A'
\end{tikzcd}$$

We will see in Sect. 7 how elegantly abstraction can be dealt with in the TADS setting: The abstraction that transforms the XOR regression setting into a classification setting can be easily realized via the TADS composition operator.

4 Affine functions

The following notations of linear algebra are based on the book [2]. The real vector space $(\mathbb{R}^n, +, \cdot)$ with $n > 0$ is an algebraic structure with the operations

$$+\colon \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n \qquad \text{vector addition}$$
$$\cdot\colon \mathbb{R} \ \ \times \mathbb{R}^n \rightarrow \mathbb{R}^n \qquad \text{scalar multiplication}$$

which are defined as

$$(x_1, \ldots, x_n) + (y_1, \ldots, y_n) = (x_1 + y_1, \ldots, x_n + y_n)$$
$$\lambda \cdot (x_1, \ldots, x_n) = (\lambda \cdot x_1, \ldots, \lambda \cdot x_n)$$

A real vector $(x_1, \ldots, x_n)$ of $\mathbb{R}^n$ is abbreviated as $\vec{x}$. To refer to one of the components, we write $\vec{x}_i := x_i$ (note the arrow ends before the subscript in contrast to $\vec{x_i}$, which denotes the i-th vector). The dimension of a real vector space $\mathbb{R}^n$ is given as $\dim \mathbb{R}^n = n$.

A matrix $\boldsymbol{W}$ is a collection of real values arranged in a rectangular array with n rows and m columns.

$$\boldsymbol{W} = \begin{pmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,m} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \ldots & w_{n,m} \end{pmatrix}$$

To indicate the number of rows and columns, one says $\boldsymbol{W}$ has *type* $n \times m$ also notated as $\boldsymbol{W} \in \mathbb{R}^{n \times m}$.

An element at position i, j of the matrix $\boldsymbol{W}$ is denoted by $\boldsymbol{W}_{i,j} := w_{i,j}$ (where $1 \leq i \leq n$ and $1 \leq j \leq m$). A matrix $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ can be reflected along the main diagonal resulting in the transpose $\boldsymbol{W}^\top$ of shape $m \times n$ defined by the equation

$$(\boldsymbol{W}^\top)_{i,j} := \boldsymbol{W}_{j,i}$$

The i-th row of $\boldsymbol{W}$ can be regarded as a $1 \times m$ matrix given by

$$\boldsymbol{W}_{i,\cdot} := (w_{i,1}, \ldots, w_{i,m}).$$

Similarly, the j-th column of $\boldsymbol{W}$ can be regarded as a $n \times 1$ matrix defined as

$$\boldsymbol{W}_{\cdot,j} := (w_{1,j}, \ldots, w_{n,j})^\top.$$

Matrix addition is defined over matrices with the same type to be component-wise, i.e.,

$$(\boldsymbol{W} + \boldsymbol{N})_{i,j} := \boldsymbol{W}_{i,j} + \boldsymbol{N}_{i,j}$$

and scalar multiplication as

$$(\lambda \cdot \boldsymbol{W})_{i,j} := \lambda \cdot \boldsymbol{W}_{i,j}.$$

The (type-correct) multiplication of two matrices $\boldsymbol{W} \in \mathbb{R}^{n \times r}$ and $\boldsymbol{N} \in \mathbb{R}^{r \times m}$ is defined as

$$(\boldsymbol{W} \cdot \boldsymbol{N})_{i,j} := \sum_{k=1}^{r} \boldsymbol{W}_{i,k} \cdot \boldsymbol{N}_{k,j}$$

Identifying

- $n \times 1$ matrices with (column) vectors
- $1 \times m$ matrices with row vectors
- 1×1 matrices with scalars

as indicated above, makes the well-known dot product of $\vec{v}, \vec{w} \in \mathbb{R}^n$

$$\vec{v}^\top \cdot \vec{w} := \sum_{i=1}^{n} \vec{v}_i \cdot \vec{w}_i$$

just a special case of matrix multiplication. The same holds for matrix-vector multiplication that is defined for a $n \times m$ matrix $\boldsymbol{W}$ and a vector $\vec{x} \in \mathbb{R}^n$ as

$$(\boldsymbol{W}\vec{x})_i := (\boldsymbol{W}_{i,\cdot})\vec{x}$$

Matrices with the same number of rows and columns, i.e., with type $n \times n$ for some $n \in \mathbb{N}$, are said to be *square matrices*. Square matrices have a neutral element for matrix multiplication, called *identity matrix*, that is zero everywhere except for the entries on the main diagonal, which are one.

$$\boldsymbol{I}^n := \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$$

The i-th unit vector $\vec{e}_i$ is a vector where all entries are zero except the i-th, which is one.

4.1 Piece-wise affine functions

Definition 12 (Affine function)
A function $\alpha\colon \mathbb{R}^n \to \mathbb{R}^m$ is called *affine* iff it can be written as

$$\alpha(\vec{x}) = \boldsymbol{W}\vec{x} + \vec{b}$$

for some matrix $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ and vector $b \in \mathbb{R}^m$.[7]

We identify the semantics and syntax of affine functions with the pair $(\boldsymbol{W}, \vec{b})$, which can be considered as a canonical representation of affine functions. Furthermore, we denote the set of all affine functions $\mathbb{R}^n \to \mathbb{R}^m$ as $\Phi(n,m)$ and call (n,m) the *type* of $\Phi(n,m)$. The untyped version Φ is meant to refer to the set of all affine functions, independently of their type.

Lemma 1 (Operations on affine functions)
Let α_1, α_2 be two affine functions in canonical form, i.e.,

$$\alpha_1(\vec{x}) = \boldsymbol{W_1}\vec{x} + \vec{b}_1$$

$$\alpha_2(\vec{x}) = \boldsymbol{W_2}\vec{x} + \vec{b}_2$$

Assuming matching types, the operations + (addition), $\cdot$ (scalar multiplication), and $\circ$ (function application) can be calculated on the representation as

$$(s \cdot \alpha_1)(\vec{x}) = (s \cdot \boldsymbol{W_1})\,\vec{x} + (s \cdot \vec{b}_1)$$

$$(\alpha_1 + \alpha_2)(\vec{x}) = (\boldsymbol{W_1} + \boldsymbol{W_2})\,\vec{x} + (\vec{b}_1 + \vec{b}_2)$$

$$(\alpha_2 \circ \alpha_1)(\vec{x}) = (\boldsymbol{W_2}\boldsymbol{W_1})\,\vec{x} + (\boldsymbol{W_2}\vec{b}_1 + \vec{b}_2)$$

resulting again in an affine function in canonical representation.

It is well-known that the type resulting from function composition evolves as follows

$$\circ\colon\ \Phi(r,m) \times \Phi(n,r) \longrightarrow \Phi(n,m).$$

The type of the operation is important for the closure axiom, the basis for most algebraic structures. This leads to the following well-known theorem [2]:

Theorem 5 (Algebraic properties)
Denoting, as usual, scalar multiplication with $\cdot$ and function composition with $\circ$, we have:

- $(\Phi(n,m), +, \cdot)$ *forms a vector space and*
- $(\Phi(n,n), \circ)$ *forms a monoid.*

This theorem can straightforwardly be lifted to untyped Φ by simply restricting all operations to the cases where they are well-typed, i.e., where addition is restricted to functions of the same type ($+_t$), and function composition to situation where the output type of the first function matches the input type of the second ($\circ_t$):

Theorem 6 (Properties of typed operations)
$(\Phi, +_t, \cdot, \circ_t)$ *forms a typed algebra, i.e, an algebraic structure that is closed under well-typed operations.*

Piece-wise affine functions are usually defined over a polyhedral partitioning of the pre-image space [8, 20, 42].

Definition 13 (Halfspace)
Let $\vec{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Then the set

$$p = \{\vec{x} \in \mathbb{R}^n \mid \vec{w}^\top \cdot \vec{x} + b = 0\}$$

is called a *hyperplane* of $\mathbb{R}^n$. A hyperplane partitions $\mathbb{R}^d$ into two convex subspaces, called *halfspaces*. The positive and negative halfspaces of p are defined as

$$p^+ := \{\vec{x} \in \mathbb{R}^n \mid \vec{w}^\top \cdot \vec{x} + b > 0\}$$

$$p^- := \{\vec{x} \in \mathbb{R}^n \mid \vec{w}^\top \cdot \vec{x} + b < 0\}$$

Definition 14 (Polyhedron)
A polyhedron $Q \subseteq \mathbb{R}^n$ is the intersection of k halfspaces for some natural number k.

$$Q = \bigcap_{i=1}^{k} \{\vec{x} \in \mathbb{R}^n \mid \vec{w}_i^{\ \top} \cdot \vec{x} + b_i \le 0\}$$

Definition 15 (Piece-wise affine function)
A function $\psi\colon \mathbb{R}^n \to \mathbb{R}^m$ is called *piece-wise affine* if it can be written as

$$\psi(\vec{x}) = \alpha_i(\vec{x}) \ \text{ for } \ \vec{x} \in Q_i$$

where $Q = \{Q_1, \ldots, Q_k\}$ is a set of polyhedra that partitions the space of $\vec{x}$ and $\alpha_1, \ldots, \alpha_k$ are some affine functions. We call $\alpha_i = \boldsymbol{W}_i\vec{x} + \vec{b}_i$ $(1 \le i \le k)$ the function associated with polyhedron Q_i.

The proof of the following property is straightforward:

Proposition 1 (Continuity)
A piece-wise affine function is continuous iff at each border between two connected polyhedra the affine functions associated with either polygon agree.

[7] Note that in a traditional neural network application, the $\boldsymbol{W}$ and $\vec{b}$ occurring in a network are the result of some training procedure. In this work, we assume that they are always known and fixed.

4.2 The activation function ReLU

In this paper, we focus on neural network architectures that use the ReLU activation function:

Definition 16 (ReLU)
The Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}^k : \mathbb{R}^k \to \mathbb{R}^k_+$$

is a projection of $\mathbb{R}^k$ onto the space of positive vectors $\mathbb{R}^k_+$ defined by replacing each component x_i of a vector $\vec{x}$ by $\max\{0, x_i\}$:

$$\left(\mathrm{ReLU}^k(\vec{x})\right)_j := \max\{0, x_j\}$$

If the input dimension is clear, we omit the index and just write ReLU.

The term $\max\{0, x_i\}$ is continuous and piece-wise linear. As ReLU operates independently on all dimensions of its input, it is itself piece-wise linear.

From a practical perspective, ReLU is one of the best understood activation functions, and the corresponding rectifier networks are one of the most popular modern neural network architectures [15].

For ease of notation in later sections, we use the fact that ReLU operates on each component of a vector independently, and can therefore be decomposed into

$$\mathrm{ReLU}^k = \phi^k_k \circ \phi^k_{k-1} \circ \cdots \circ \phi^k_1$$

where $\phi^k_i : \mathbb{R}^k \to \mathbb{R}^k$ is the *partial ReLU function* defined by setting the i-th component of a vector $\vec{x}$ to 0 iff $x_i < 0$. More formally,

$$\left(\phi^k_i(\vec{x})\right)_j := \begin{cases} x_j & \text{if } i \neq j \\ \max\{0, x_j\} & \text{if } i = j. \end{cases}$$

5 Piece-wise linear neural network

Piece-wise linear neural networks are specific representations of continuous piece-wise affine functions. Calling them piece-wise linear is formally incorrect (the term piece-wise affine would be correct), but established. For the ease of exposition, we restrict the following development to the case where all activation functions are partial ReLU functions. This suffices to capture the entire class of Rectifier Networks, which themselves can represent all piece-wise affine functions [26]. We adopt the popular naming in the following definition:

Definition 17 (Rectifier neural networks)
The syntax for *Rectifier Neural Networks*, or here synonymously used, *Piece-wise Linear Neural Networks* (PLNNs), is defined by the following BNF

$$\mathcal{N} ::= \varepsilon \mid \alpha \,;\mathcal{N} \mid \phi \,;\mathcal{N}$$

where the meta variables α and ϕ stand for affine functions and partial ReLU functions, respectively. Writing PLNNs as $N = f_0 \,; \cdots ; f_l$ where $f \in \{\alpha, \phi\}$ we denote the set of all PLNNs with $\mathrm{dom}(f_0) = \mathbb{R}^n$ and $\mathrm{codom}(f_l) = \mathbb{R}^m$ as $\mathcal{N}(n, m)$ and the set of all PLNNs as

$$\mathcal{N} = \bigcup_{n,m \in \mathbb{N}} \mathcal{N}(n, m)$$

This definition of a PLNN slightly flexibilizes the classical definition as it does not require the strict alternation of affine functions and activation functions and uses partial ReLU functions instead of ReLU. We will exploit this flexibility to directly have the right granularity for defining according operational semantics (cf., Sect. 5.2).

Example 1 (Representing XOR as PLNN)
As stated in Sect. 2, our baseline solution to the XOR regression model is defined by the function $|x - y|$. We can represent this function as a PLNN N_*. It consists of two affine functions

$$\alpha_1 = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \qquad \alpha_2 = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

and two partial ReLUs applied in this order:

$$N_* := \alpha_1 \,; \phi^2_1 \,; \phi^2_2 \,; \alpha_2$$

Note that typically N_* would be defined as

$$N'_* = \alpha_1 \,; \mathrm{ReLU} \,; \alpha_2$$

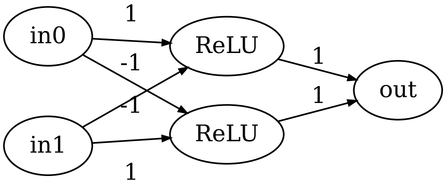
in the context of machine learning. However, as both definitions impose the same semantics

$$[\![N_*]\!]_{\mathrm{DS}} = [\![N'_*]\!]_{\mathrm{DS}}$$

we defined it directly using the notational conventions of this paper. This construction uses the observation that

$$\mathrm{ReLU}(x - y) = \begin{cases} x - y & \text{if } x > y \\ 0 & \text{otherwise.} \end{cases}$$

The following figure shows the (instantiated) corresponding network architecture:



This example shows that PLNNs can encode our baseline solution to the XOR problem. However, it is important to note that PLNNs are not usually manually defined, but rather trained to approximate a function using approximative learning algorithms [17], see also Sect. 7.

5.1 Semantics of PLNNs

PLNNs come with a very natural denotational semantics:

Definition 18 (Denotational semantics)
The denotational semantics

$$[\![\cdot]\!]_{\mathrm{DS}} : \mathcal{N}(n,m) \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R}^m)$$

of PLNNs is inductively defined as the composition of all functions in a PLNN:

$$[\![\varepsilon]\!]_{\mathrm{DS}} = \mathrm{id} \quad \text{and} \quad [\![f\,;N]\!]_{\mathrm{DS}} = [\![N]\!]_{\mathrm{DS}} \circ f$$

where $f \in \{\alpha, \phi\}$.

Remark
In this definition we overload f to represent both its corresponding syntactic artifact (e.g., a matrix) and its semantic artifact (e.g. the corresponding affine function or partial ReLU function, respectively). In the remainder of the paper it should always be clear from the context which interpretation we refer to.

PLNNs can also be evaluated in an operational manner based on derivation rules which closely resemble their process of computation. For that we define the defect matrix $\boldsymbol{E}_i^k$ $(1 \le i \le k)$ as the identity matrix $\boldsymbol{I}^k$ where the i-th entry on the main diagonal, i.e., element (i,i), is set to zero.

Definition 19 (Operational semantics)
The operational semantics of PLNNs is defined via the following three rules that operate on configurations $\langle N, \vec{x}\rangle$ consisting of the remainder N of the PLNN to process and the current intermediate result vector $\vec{x}$.

$$\text{Affine}\;\frac{\alpha \text{ is affine}}{\langle \alpha\,; N, \vec{x}\rangle \overset{\mathrm{tt}}{\rightharpoonup} \langle N, \alpha(\vec{x})\rangle}$$

$$\text{ReLU 1}\;\frac{x_i \ge 0}{\langle \phi_i^k\,; N, \vec{x}\rangle \overset{1}{\rightharpoonup} \langle N, \vec{x}\rangle}$$

$$\text{ReLU 2}\;\frac{x_i < 0}{\langle \phi_i^k\,; N, \vec{x}\rangle \overset{0}{\rightharpoonup} \langle N, \boldsymbol{E}_i^k \vec{x}\rangle}$$

The labels (the symbols above the arrow) provide a history of which rule was applied. It is easy to see that the rule to be applied next is always uniquely determined by the first component (the PLNN) which guarantees that the operational semantics is deterministic. In fact, for each input there exists a unique computation path. Thus, the following definition is well-defined:

Definition 20 (Semantic functional $[\![\cdot]\!]_{\mathrm{OS}}$)
The semantic functional for the operational semantics

$$[\![\cdot]\!]_{\mathrm{OS}} : \mathcal{N}(n,m) \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R}^m)$$

is defined as

$$[\![N]\!]_{\mathrm{OS}}(\vec{x}) = \vec{y} \;\text{ iff }\; \langle N, \vec{x}\rangle \rightharpoonup^{*} \langle \varepsilon, \vec{y}\rangle$$

Note that these rules stand in close correspondence to the denotational semantics of PLNNs with each rule describing the evaluation of one of its constituent functions. In fact, we have:

Theorem 7 (Correctness of $[\![\cdot]\!]_{\mathrm{OS}}$)
For any $N \in \mathcal{N}$ we have $[\![N]\!]_{\mathrm{DS}} = [\![N]\!]_{\mathrm{OS}}$.

Proof sketch
The proof follows straightforwardly by induction on the number of layers of a PLNN. It suffices to show that the affine rule corresponds to the application of the affine function α and that the executions of the adequate rules ReLU 1 and ReLU 2 correctly cover the partial ReLU activation functions. □

Thus, the operational semantics $[\![\cdot]\!]_{\mathrm{OS}}$ provides a constructive, local, and correct semantic interpretation of PLNNs.

Example 2 (Semantics of N_*)
We consider the baseline solution to the XOR regression model defined by the function $|x-y|$. The network N_* implements this function as a PLNN. We calculate $[\![N_*]\!]_{\mathrm{OS}}$ by applying the SOS rules to the initial configuration $\langle N_*, (1,0)^{\mathsf{T}}\rangle$.

$$\begin{aligned} & \langle \alpha_1\,; \phi_1^2\,; \phi_2^2\,; \alpha_2, (1,0)^{\mathsf{T}}\rangle \\ \overset{\mathrm{tt}}{\rightharpoonup}\; & \langle \phi_1^2\,; \phi_2^2\,; \alpha_2, (1,-1)^{\mathsf{T}}\rangle \\ \overset{1}{\rightharpoonup}\; & \langle \phi_2^2\,; \alpha_2, (1,-1)^{\mathsf{T}}\rangle \end{aligned}$$

$$\begin{aligned} &\overset{0}{\rightharpoonup} \quad \langle \alpha_2, (1,0)^{\mathsf{T}} \rangle \\ &\overset{\mathrm{tt}}{\rightharpoonup} \quad \langle \varepsilon, 1 \rangle \end{aligned}$$

This is the correct outcome. Note that the SOS rules correspond to an iterative processing of each component function (i.e., layer) of the neural network, much like function composition.

Next, we will naturally adapt the presented rules to symbolic execution, which by itself provides the first outcome explanation of PLNNs.

5.2 Symbolic execution of PLNNs

Symbolic execution aims at characterizing program states in terms of symbolic input values and corresponding path conditions. In particular, it reveals how program states depend on the initial values during execution. PLNNs are ideally suited for symbolic execution as they are acyclic computation graphs and contain only affine computations.

- Affine functions are closed under composition. This allows one to aggregate (partially evaluate) the entire symbolic computation history corresponding to some symbolic execution path in terms of a single affine function, and to express all paths conditions as affine inequalities, also expressed in terms of the initial values.
- PLNNs possess finite, acyclic computation graphs, which conceptually allow for precise execution without need for abstractions.

In Sect. 6, we will see that this results in a directed, acyclic, side-effect-free computation graph whose leaves are affine function in $\Phi(n,m)$ that express the PLNN's effect on inputs belonging to the polyhedron specified by the path condition.

We define the required symbolic execution via derivation rules that transform configurations of the form $\langle N, \alpha, \pi \rangle$, where

- $N \in \mathcal{N}(r,m)$,
- $\alpha \colon \mathbb{R}^n \longrightarrow \mathbb{R}^r$ with representation $\alpha(\vec{x}) = \boldsymbol{W}\vec{x} + \vec{b}$,
- and π is a path condition

throughout the transformation. The dimensions of n and m are bound by the initial PLNN while r is the dimension of some hidden layer. The following definition operates on the concrete representations of N, α, and π. In the case of the last two, the representation is expected to be canonical and therefore syntax and semantics can be identified.[8] Operations are expected to be applied directly to the representation. Thus, the effect of a concrete execution path of $[\![\cdot]\!]_{\mathrm{OS}}$ is *aggregated* (instead of simply recorded) into the components α and π, while N is destructed further and further until all layers have been considered.

Definition 21 (Symbolic execution of PLNNs)

$$\langle \alpha' \,;\, N, \alpha, \pi \rangle \overset{\mathrm{tt}}{\rightharpoonup}_{\mathrm{SOS}} \langle N, \alpha' \circ \alpha, \pi \rangle$$

$$\langle \phi_i^k \,;\, N, \alpha, \pi \rangle \overset{1}{\rightharpoonup}_{\mathrm{SOS}} \langle N, \alpha, \pi' \wedge \pi \rangle$$

$$\langle \phi_i^k \,;\, N, \alpha, \pi \rangle \overset{0}{\rightharpoonup}_{\mathrm{SOS}} \langle N, \boldsymbol{E}_i^k \circ \alpha, \neg\pi' \wedge \pi \rangle$$

where $\pi' = \alpha(x)_i \geq 0$.

For a sequence

$$c_0 \overset{a_1}{\rightharpoonup}_{\mathrm{SOS}} c_1 \,\cdots\, c_{n-1} \overset{a_n}{\rightharpoonup}_{\mathrm{SOS}} c_n$$

of derivations we write $c_0 \overset{a_1 \cdots a_n}{\rightharpoonup}_{\mathrm{SOS}} c_n$. Further, we denote with $(\rightharpoonup_{\mathrm{SOS}})^k$ the application of $\rightharpoonup_{\mathrm{SOS}}$ k times, and we write $\rightharpoonup_{\mathrm{SOS}}^*$ if k is of no interest. The following properties follow by straightforward induction on the length of the derivation sequences:

Proposition 2 (Derivation sequences)
The following properties hold for all derivations of $\rightharpoonup_{\mathrm{SOS}}$*:*

1. $\langle N, \mathrm{id}, \mathrm{tt} \rangle \overset{w}{\rightharpoonup}_{\mathrm{SOS}} \langle \varepsilon, \alpha, \pi \rangle \iff$
   $\langle N \,;\, N', \mathrm{id}, \mathrm{tt} \rangle \overset{w}{\rightharpoonup}_{\mathrm{SOS}} \langle N', \alpha, \pi \rangle$,
2. $\langle N, \mathrm{id}, \mathrm{tt} \rangle \overset{w}{\rightharpoonup}_{\mathrm{SOS}} \langle \varepsilon, \alpha, \pi \rangle \implies$
   $\langle N, \alpha', \pi' \rangle \overset{w}{\rightharpoonup}_{\mathrm{SOS}} \langle \varepsilon, \alpha \circ \alpha', \pi \wedge \pi' \rangle$,
3. $\langle N, \alpha, \pi \rangle \overset{w}{\rightharpoonup}_{\mathrm{SOS}} \langle N', \alpha', \pi' \rangle$ *is unique in* w.

Intuitively, the first identity states that derivations with the same prefix in the first component result in the same configuration after the prefix was completely processed. The second states the effect of other starting values in the initial configuration. Note that this relation does not hold in the reversed case, as $\circ$ and $\wedge$ are not injective and the configuration is only determined up to isomorphism. The last identity corresponds to the result that $\rightharpoonup$ of Definition 19 is uniquely determined. As the symbolic rules are more general, the result is restricted to the case where the word w is known.

Moreover, the path conditions induce a partition of the input space $\mathbb{R}^n$.

Lemma 2 (Partition of π)
For an arbitrary but fixed $N \in \mathcal{N}$ define the set of all derivations with depth k as

$$V_k(N) := \{ c \mid \langle N, \mathrm{id}, \mathrm{tt} \rangle \overset{w}{\rightharpoonup}_{\mathrm{SOS}} c \;\wedge\; |w| = k \}$$

[8] By using canonical representations it is impossible to trace the history of operations. One effectively cannot distinguish between isomorphic objects.

Define the set of all path conditions of the same V_k as Π_k, then

- *each $\pi \in \Pi_k$ defines a polyhedron for $k > 0$*
- *the polyhedra of Π_k are a partition of $\mathbb{R}^n$.*

Proof sketch
Induction over derivation sequences of N. □

Specifically, for each input vector $\vec{x} \in \mathbb{R}^n$, there exists exactly one sequence of derivations $\langle N, \mathrm{id}, \mathrm{tt}\rangle \overset{w}{\rightharpoonup}_{\mathrm{SOS}} \langle \varepsilon, \alpha, \pi\rangle$ such that $\pi(\vec{x})$ holds. Therefore, the following is well-defined:

Definition 22 (Semantic functional $\llbracket \cdot \rrbracket_{\mathrm{SOS}}$)
The semantic functional for the symbolic operational semantics

$$\llbracket \cdot \rrbracket_{\mathrm{SOS}} : \mathcal{N}(n,m) \to (\mathbb{R}^n \to \mathbb{R}^m)$$

is defined as $\llbracket N \rrbracket_{\mathrm{SOS}}(\vec{x}) = \vec{y}$ iff

$$\langle N, \mathrm{id}, \mathrm{tt}\rangle \rightharpoonup^{*}_{\mathrm{SOS}} \langle \varepsilon, \alpha, \pi\rangle \quad \wedge \quad \pi(\vec{x}) = 1 \quad \wedge \quad \alpha(\vec{x}) = \vec{y}$$

Also the symbolic operational semantics is fully aligned with the denotational semantics:

Theorem 8 (Correctness of $\llbracket \cdot \rrbracket_{\mathrm{SOS}}$)
For any $N \in \mathcal{N}$ we have: $\llbracket N \rrbracket_{\mathrm{DS}} = \llbracket N \rrbracket_{\mathrm{SOS}}$.

Proof sketch
According to Theorem 8, it suffices to show the semantic equivalence with $\llbracket \cdot \rrbracket_{\mathrm{OS}}$. As both the concrete and the symbolic operational semantics define unique computation paths for each input vector, the proof follows straightforwardly by an inductive proof that establishes the desired equivalence as an invariant when simultaneously following these paths. More concretely, we can prove

$$\forall \vec{x} \in \mathbb{R}^n : \llbracket N \rrbracket_{\mathrm{OS}}(\vec{x}) = \llbracket N \rrbracket_{\mathrm{SOS}}(\vec{x})$$

using the following induction hypothesis

$$\begin{aligned} &\langle N_0, \vec{x_0}\rangle \overset{w}{\rightharpoonup}_{\mathrm{OS}} \langle N_k, \vec{x_k}\rangle \iff \\ &\langle N_0, \mathrm{id}, \mathrm{tt}\rangle \overset{w}{\rightharpoonup}_{\mathrm{SOS}} \langle N_k, \alpha_k, \pi_k\rangle \wedge \alpha_k(\vec{x_0}) = \vec{x_k} \wedge \pi_k(\vec{x_0}) \end{aligned}$$

by a simple analysis of the following three cases:

1. $N_{k+1} = \alpha' \,;\, N_k$
2. $N_{k+1} = \phi_i^k \,;\, N_k \;\wedge\; \vec{x}_i \geq 0$
3. $N_{k+1} = \phi_i^k \,;\, N_k \;\wedge\; \vec{x}_i < 0$ □

The symbolic operational semantics of PLNNs is sufficient to derive local explanations and decision boundaries similar to the ones presented in [12, 18]. In the following, we will show how symbolic operational semantics can be used to define semantically equivalent Typed Affine Decisions Structures (TADS), which themselves are specific Algebraic Decision Structures (ADS), as defined in the next section. TADS collect all the local explanations in an efficient query structure such that we arrive at model explanations.

Example 3 (XOR-regression)
As a brief example for the symbolic execution of PLNNs, we will calculate $\llbracket N_* \rrbracket_{\mathrm{SOS}}$ by applying the symbolic SOS rules to the initial configuration $\langle N_*, \mathrm{id}, \mathrm{tt}\rangle$. Symbolic interpretation is not deterministic for the partial ReLU functions. We therefore chose the execution path that corresponds to the former example $\vec{x} = (1,0)^\intercal$, i.e., with the label sequence $w = (\mathrm{tt}, 1, 0, \mathrm{tt})$, for illustration:

$$\begin{aligned} & \langle \alpha_1 \,;\, \phi_1^2 \,;\, \phi_2^2 \,;\, \alpha_2, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mathrm{tt}\rangle \\ \overset{\mathrm{tt}}{\rightharpoonup}_{\mathrm{SOS}} \quad & \langle \phi_1^2 \,;\, \phi_2^2 \,;\, \alpha_2, \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \mathrm{tt}\rangle \\ \overset{1}{\rightharpoonup}_{\mathrm{SOS}} \quad & \langle \phi_2^2 \,;\, \alpha_2, \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, x_1 - x_2 \geq 0\rangle \\ \overset{0}{\rightharpoonup}_{\mathrm{SOS}} \quad & \langle \alpha_2, \begin{pmatrix} 1 & -1 \\ 0 & 0 \end{pmatrix}, x_1 - x_2 > 0\rangle \\ \overset{\mathrm{tt}}{\rightharpoonup}_{\mathrm{SOS}} \quad & \langle \varepsilon, \begin{pmatrix} 1 & -1 \end{pmatrix}, x_1 - x_2 > 0\rangle \end{aligned}$$

Note that the path conditions and the affine functions have been simplified in every step. The affine functions are given in their canonical representation $\boldsymbol{W}\vec{x} + \vec{b}$ (as $\vec{b}$ is zero in all steps it is omitted). For the path conditions we have not fixed a representation, instead they are simplified to aid readability. The most important simplifications are

$$\begin{aligned} \left(\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \vec{x} \right)_1 \geq 0 &\iff x_1 - x_2 \geq 0 \\ \neg(-x_1 + x_2 \geq 0) \wedge x_1 - x_2 \geq 0 &\iff x_1 - x_2 > 0 \end{aligned}$$

6 Typed affine decision structures

Consider the transition system $(V, \rightharpoonup_{\mathrm{SOS}})$ that represents the symbolic operational semantics $\llbracket \cdot \rrbracket_{\mathrm{SOS}}$ of some $N \in \mathcal{N}(n,m)$ where

$$V = \{ c \mid \langle N, \mathrm{id}, \mathrm{tt}\rangle \rightharpoonup^{*}_{\mathrm{SOS}} c \}$$

is the set of configurations which are reachable from $\langle N, \mathrm{id}, \mathrm{tt}\rangle$ and let (recall Definitions 6 and 12)

$$\tau : V \to \mathcal{S}_\Phi$$

denote the following inductively defined transformation that closely follows the symbolic SOS rules:

- $\tau(\langle \varepsilon, \alpha, \cdot \rangle) \ := \ \alpha$
- $\tau(\langle \alpha' \, ; N, \alpha, \cdot \rangle) \ := \ (\mathrm{tt}, \tau(\langle N, \alpha' \circ \alpha, \cdot \rangle), \varepsilon)$
- $\tau(\langle \phi_i^k \, ; N, \alpha, \cdot \rangle)$
  $:= \ (\alpha(x)_i \geq 0, \tau(\langle N, \alpha, \cdot \rangle), \tau(\langle N, \boldsymbol{E}_i^k \circ \alpha, \cdot \rangle))$

where " $\cdot$ " should be considered a don't care entry. Identifying N with its computation tree, which is specified by its set of configurations that are reachable from $\langle N, \mathrm{id}, \mathrm{tt} \rangle$,[9] τ can be regarded as an injective relabeling of this tree, which results in the structure of an ADT:

Theorem 9 (TADT)
Let $N \in \mathcal{N}(n,m)$. Then $\tau(N)$ is an ADT over $\Phi(n,m)$ whose predicates are all of the form of affine inequalities.

Proof sketch
The proof follows by straightforward induction along the isomorphic structure of the two trees. The following invariants hold for all steps of the transformation

$$\tau(c) = (p, \tau(c_t), \cdot\,) \iff c \xrightarrow{1}_{\mathrm{SOS}} c_t$$

$$\tau(c) = (p, \cdot\,, \tau(c_f)) \iff c \xrightarrow{0}_{\mathrm{SOS}} c_f$$

where we abbreviate $c = \langle N, \alpha, \pi \rangle$, $c_t = \langle N', \alpha', p \wedge \pi \rangle$, and $c_f = \langle N', \alpha', \neg p \wedge \pi \rangle$. □

We call the structures resulting from τ-transformation *Typed Affine Decision Trees* (TADT). A TADT inherits the type from its underlying algebra of typed affine functions $\Phi(n,m)$ (cf. Lemma 1 and Theorem 6). Similar to ADTs, TADT can also be generalized to acyclic graph structures:

Definition 23 (Typed affine decision structure)
An ADS over the algebra $(\Phi(n,m), +_t, \cdot, \circ_t)$ where all predicates are linear inequalities in $\mathbb{R}^n$ is called *Typed Affine Decision Structure* of type $n \times m$.

The set of all such decision structures is denoted by $\Theta(n,m)$, and the set of all typed affine decision structures of any type with:

$$\Theta = \bigcup_{n,m \in \mathbb{N}^+} \Theta(n,m)$$

TADS are special kinds of ADS. Thus, they inherit the ADS semantics (cf. Definition 7) when specializing Σ to $\mathbb{R}^n$ and σ to $\vec{x}$. The fact that the semantics of leafs is given by affine functions that are also applied to $\vec{x}$ is not important for the resulting specialized definition which reads:

Definition 24 (Semantics of TADS)
The semantic function

$$[\![\, \cdot \,]\!]_\Theta \colon \Theta(n,m) \to (\mathbb{R}^n \to \Phi(n,m))$$

for TADS is inductively defined as

$$[\![\alpha]\!]_\Theta(\vec{x}) := \alpha(\vec{x})$$

$$[\![(p,l,r)]\!]_\Theta(\vec{x}) := \begin{cases} [\![l]\!]_\Theta(\vec{x}) & \text{if } [\![p]\!](\vec{x}) = 1 \\ [\![r]\!]_\Theta(\vec{x}) & \text{if } [\![p]\!](\vec{x}) = 0 \end{cases}$$

Every PLNN N defines an ADT t_N over Φ. We can therefore apply the results of Sect. 3. In particular, we can apply the various reduction techniques, which transform t_N into the more general form of an ADS, or more precisely, of a TADS.

Optimizations in terms of semantic reduction and infeasible path elimination do not alter the semantics of a (T)ADS. In other words

$$\Theta(N) \ = \ \{t \mid [\![t]\!]_\Theta = [\![(\tau(N)]\!]_\Theta\}$$

is closed under semantic reduction and infeasible path elimination. Moreover, we have:

Theorem 10 (Correctness of $[\![t]\!]_\Theta$)
Let $N \in \mathcal{N}$ and $t \in \Theta(N)$. Then we have:

$$[\![N]\!]_{\mathrm{DS}} = [\![t]\!]_\Theta$$

In the following, we sometimes abuse notation and also write $\tau(N)$ for other members of $t \in \Theta(N)$ when the concrete structure of the TADS does not matter. This concerns, in particular, Sect. 7 where we always apply semantic reduction and infeasible path elimination to reduce size.

Following [23]:

> In the state of the art a small set of existing interpretable models is recognized: decision tree, rules, linear models [...]. These models are considered easily understandable and interpretable for humans. ([23])

We have:

Corollary 1 (Model explanation)
TADS provide precise solutions to the model explanation *problem, and therefore also to the* outcome explanation *problem.*

Please note that outcome explanation is easily derived from model explanation simply by following the respective evaluation path.

Example 4 (XOR-TADS)
As an example, the resulting TADS of the symbolic execution ADS of N_* is shown in Fig. 3.

[9] Please note that the transition labels tt, 1, and 0 are redundant.

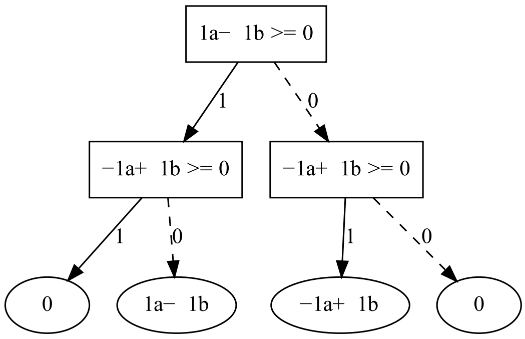


Fig. 3 The TADS $\tau(N_*)$

6.1 The TADS linear algebra

According to Theorem 6, $(\Phi, +_t, \cdot_t)$ forms a typed algebra. Moreover, due to the canonical representation of affine functions, Φ also supports the equality relation $=$. Applying Theorem 3, all these operations can be lifted to obtain the following corresponding operations on TADS:

1. $\oplus\colon \Theta(n,m) \times \Theta(n,m) \to \Theta(n,m)$
2. $\ominus\colon \Theta(n,m) \times \Theta(n,m) \to \Theta(n,m)$
3. $\odot\colon \mathbb{R} \times \Theta(n,m) \to \Theta(n,m)$
4. $⊜\colon \Theta(n,m) \times \Theta(n,m) \to \Theta(n,1)$

These operations lift, in the order that they are given, (1) addition, (2) subtraction,[10] (3) scalar multiplication, and (4) equality. The resulting TADS has size $\mathcal{O}(|t_1| \cdot |t_2|)$ where $|t_i|$ is the number of nodes of considered TADS t_i. An example of addition is given in Fig. 4.

The operations $\oplus$ and $\odot$ are characteristic for vector spaces. Indeed, TADS form a (function) vector space (cf., Theorems 5 and 6):

Theorem 11 (The TADS linear algebra)
$(\Theta, \oplus_t, \odot_t)$ *forms a typed linear algebra.*

We will exploit this theorem in Sect. 7.

However, when lifting these two operators over affine predicates, a second interpretation occurs naturally: that of *piece-wise affine functions*. Both interpretations are compatible, as stated in the following lemma.

Theorem 12 (Two consistent views on TADS)
Let $\psi_1, \psi_2 \colon \mathbb{R}^n \to \mathbb{R}^m$ be two piece-wise affine functions and $\vec{x} \in \mathbb{R}^n$ be a real vector. Define α_1 as the affine function of ψ_1 that is associated with the region for $\vec{x}$ and α_2 for ψ_2, respectively and denote with $\Box$ a generic operation over (piece-wise) affine functions. Then, if for all such $\vec{x}$

$$\psi_1(\vec{x}) \Box \psi_2(\vec{x}) = \alpha_1(\vec{x}) \Box \alpha_2(\vec{x})$$

holds both interpretations agree for $\Box$.

One can easily show that this is indeed the case for $\Box \in \{\oplus, \ominus, \odot\}$. However, there is a slight difference in the interpretations. The first *lifts* affine functions over affine predicates and the latter *associates* affine functions with affine predicates. This distinction can, for example, be seen in the signature of the respective semantics:

$$[\![t]\!]_{S_A} : \mathbb{R}^n \to (\mathbb{R}^n \to \mathbb{R}^m)$$
$$[\![t]\!]_{\Theta} \;: \mathbb{R}^n \to \;\mathbb{R}^m$$

For TADS to be equivalent to piece-wise affine functions, the semantics have to be adapted to $[\![\cdot]\!]_{\Theta}$, which slightly differs from $[\![\cdot]\!]_{S_A}$ in that the leafs are also evaluated under the input.

Considering Lemma 2, one can easily see that every path in a TADS defines a polyhedron and that the set of all paths partitions $\mathbb{R}^n$. As all terminals of TADS are affine functions, it is straightforward to prove that for every TADS t the semantics $[\![t]\!]_{\Theta}$ is a piece-wise affine function.

The complexity of piece-wise affine functions is commonly defined as the smallest number of classes (so-called regions) that are needed to partition the input space [24, 37, 43], and which we call *region complexity*. This complexity measure can easily be adopted for TADS using the above reasoning, as it is simply the number of all paths from the root to its terminals. In other words, TADS are linear-size representations of PAF with respect to their region complexity, which implies:

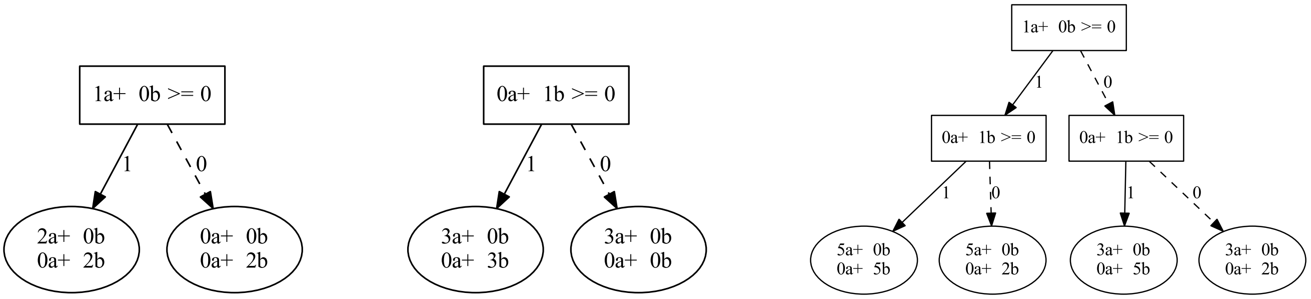
Theorem 13 (Complexity of operations)
The operations $\oplus, \ominus, ⊜$ are of quadratic and $\odot$ of linear time region complexity.

Proof sketch
Via structural induction along Definition 11 it is easy to establish that each node of the tree underlying t_1 is processed at most once, while the nodes of the tree underlying t_2 may be processed at most once for each leaf of t_1. The theorem follows from the fact that the number of nodes in a binary tree is at most twice the number of its paths. □

Interesting is the expression $t_1 \ominus t_2$ which evaluates to the constant function 0 iff t_1 and t_2 are semantically equivalent. Thus, we have the following:

Corollary 2 (Complexity of $\equiv$)
Deciding semantic equivalence between two TADS has quadratic time region complexity.

[10] Subtraction is usually not stated explicitly as it can be defined using addition and scalar multiplication.



(a) (T)ADS $2 \odot \phi_1^2$

(b) (T)ADS $3 \odot \phi_2^2$

(c) Sum of (a) and (b) $2 \odot \phi_1^2 \oplus 3 \odot \phi_2^2$

Fig. 4 Example for (T)ADS addition. The (T)ADS in (**a**) and (**b**) are based on partial ReLUs and (**c**) is the sum of both. The input vector is given as $\vec{x} = (a, b)$ with $a, b \in \mathbb{R}$

Please note that this way of deciding semantic equivalence does not only provide a Yes/No answer, but, in case of failure, also precise diagnostic information: For $t_2 - t_1$ we have (see Fig. 11)

- positive parts mark regions where t_2 is bigger
- zero marks regions where both TADS agree
- negative parts mark regions where t_1 is bigger

This is particularly interesting when combined with a threshold ε (see Fig. 13).

6.2 The TADS typed monoid

As shown in previous sections, TADS are a comprehensible and efficient representation of piece-wise affine functions. In the following, we will go even further and show that TADS also directly support all common operations on piece-wise affine functions.

Piece-wise affine functions form a typed monoid under function composition, i.e., the composition of two piece-wise affine functions is again piece-wise affine, assuming that domain and co-domain adequately match. This property is highly useful both for the design of neural networks (which are themselves fundamentally compositions of multiple, simple piece-wise affine functions) and neural network analysis, as will be shown in Sect. 7.

Consider the following result, which follows as a consequence of the previous correctness theorems and the compositionality of $[\![\cdot]\!]_{\mathrm{DS}}$:

Corollary 3 (Compositionality)
Let $N_0, N_1, N_2 \in \mathcal{N}$ with $N_0 = N_1 \,;\, N_2$ and $t_i \in \Theta(N_i)$. Then we have:

$$\begin{aligned}[\![N_0]\!]_{\mathrm{DS}} &= [\![N_1 \,;\, N_2]\!]_{\mathrm{DS}} \\ &= [\![N_2]\!]_{\mathrm{DS}} \circ [\![N_1]\!]_{\mathrm{DS}} \\ &= [\![t_2]\!]_{\Theta} \circ [\![t_1]\!]_{\Theta} = [\![t_0]\!]_{\Theta}\end{aligned}$$

Obviously, there is a gap in the result that poses the question: "Is it possible to define composition operator that directly works on TADS?" Just composing the affine functions at the leafs, which would be sufficient to, e.g., for $\oplus$, is insufficient because of the side effect of the first TADS. Thus, we end up with the following composition operator that handles this side effect in a way that is typical for structured operational semantics:

Definition 25 (TADS composition)
The composition operator $\bowtie$ of TADS with type

$$\bowtie \colon \Theta(n,r) \times \Theta(r,m) \longrightarrow \Theta(n,m)$$

is inductively defined as

$$\begin{aligned}\alpha \bowtie \alpha' &= \alpha' \circ \alpha \\ \alpha \bowtie (p,l,r) &= (p \circ \alpha, \alpha \bowtie l, \alpha \bowtie r) \\ (p,l,r) \bowtie t &= (p, l \bowtie t, r \bowtie t)\end{aligned}$$

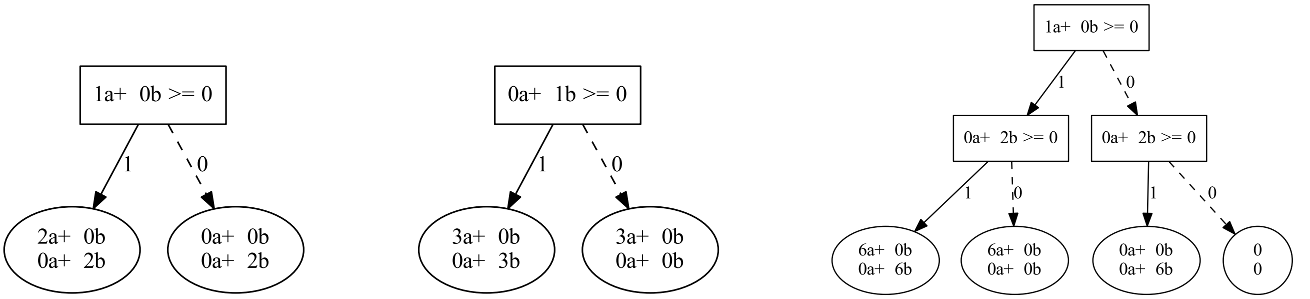
where $\alpha, \alpha' \in \Phi$ are TADS identified with their affine function, $t, l, r \in \Theta$ are TADS, and $p \in \mathcal{P}$ is a predicate. Here $p \circ \alpha$ with $p = \alpha'(x)_i \geq 0$ is defined as

$$(\alpha' \circ \alpha)(x)_i \geq 0$$

Notice that this definition is similar to the lifted operators of Definition 11. However, TADS composition is not side-effect free as can be seen by the modification of the predicate in the second case. This is due to the fact that the first TADS distorts the input vector space of the second TADS. Again, let us formalize the correctness of this operation.

Theorem 14 (TADS composition)
Let $t_1 \in \Theta(n,r)$ and $t_2 \in \Theta(r,m)$. Then we have:

$$[\![t_1 \bowtie t_2]\!]_{\Theta} = [\![t_2]\!]_{\Theta} \circ [\![t_1]\!]_{\Theta}$$



(a) TADS $2 \odot \phi_1^2$

(b) TADS $3 \odot \phi_2^2$

(c) Composition of (a) and (b) $2 \odot \phi_1^2 \bowtie 3 \odot \phi_2^2$

Fig. 5 Example for TADS composition. The TADS (**a**) and (**b**) are based on partial ReLUs. TADS (**c**) is the composition of (**a**) and (**b**). Note the difference between *lifting* (Fig. 4) and *composing* in the inner nodes. The input vector is given as $\vec{x} = (a, b)$ with $a, b \in \mathbb{R}$

Proof sketch
Structural induction along the second component and in the inductive step induction along the first component. □

An example of a composition can be found in Fig. 5.
This directly yields:

Corollary 4 (The TADS typed monoid)
$(\Theta, \bowtie)$ *forms a typed monoid, i.e., an algebraic structure that is closed under type-correct composition and that has typed neutral elements* ε*.*

On this structure τ *is a homomorphism between the monoids* $(\Theta, \bowtie)$ *and* $(N, ;)$*, i.e., the following diagram commutes*

$$\begin{tikzcd}
N^2 \arrow[r, ";"] \arrow[d, "\tau"] & N \arrow[d, "\tau"] \\
\Theta^2 \arrow[r, "\bowtie"] & \Theta
\end{tikzcd}$$

Due to their similarity to the lifted operators, it is easy to show that composing to TADS results in a third TADS that has size complexity equal to product of its inputs and whose complexity with respect to the measure of affine regions is quadratic in its inputs. Following the same line of reasoning as for Theorem 13 yields:

Theorem 15 (Complexity of composition)
TADS compositions $\bowtie$ *has quadratic time region complexity.*

One may argue that semantic equivalence between two TADS is of limited practical value, in particular, as in most applications of neural networks, small errors are, to a certain degree accepted. In contrast, ϵ-similarity, i.e., whether two TADS differ more than ϵ for some small threshold $\epsilon \in \mathbb{R}$, can be regarded as a practically very relevant notion, in particular, to study robustness properties. The corresponding property required for TADS leaves is easily defined:

$$l_\epsilon(x, y) := (|x - y| - \epsilon)\mathbb{I}(|x - y| \geq \epsilon)$$

Intuitively, this function yields 0 if the difference of x and y is less than ϵ and the absolute value (minus epsilon) of their difference otherwise. l_ϵ can easily be realized using only standard algebraic operations and ReLU applications, which are already defined:

$$l_\epsilon = \mathrm{ReLU}(x - y - \epsilon) + \mathrm{ReLU}(y - x - \epsilon)$$

Just lifting this function to the TADS level

$$t_4 = \mathrm{ReLU}(t_1 \ominus t_2 \ominus \epsilon) \oplus \mathrm{ReLU}(t_2 \ominus t_1 \ominus \epsilon)$$

(where $\mathrm{ReLU}(t) = t \bowtie \tau(\mathrm{ReLU})$) is sufficient to decide ϵ-similarity. Thus, we have:

Corollary 5 (Deciding ϵ-similarity)
ϵ-similarity has quadratic time region complexity.

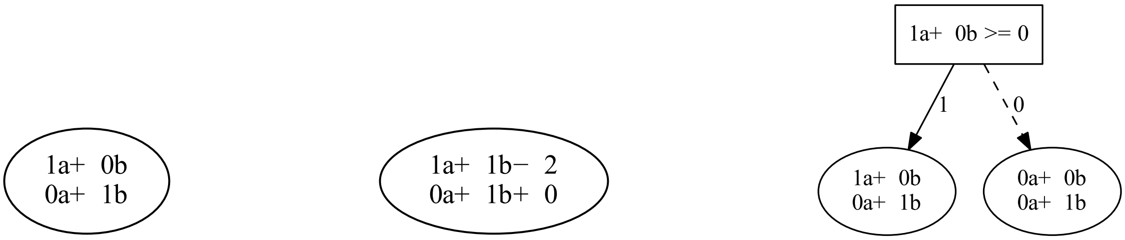
Please note that, again, this way of deciding ϵ-similarity does not only provide a Yes/No answer, but, in case of failure, also precise diagnostic information. All this will be showcased in Sect. 7.

In the remainder of this section, we elaborate on the compositionality that is imposed by $\bowtie$:

Corollary 6 (Layer-wise transformation)
By Theorem 14, we can transform a PLNN layer-wise into a TADS.

$$\begin{aligned}
[\![N]\!]_{\mathrm{DS}} &= [\![\alpha_1 ; \ldots ; \alpha_n]\!]_{\mathrm{DS}} \\
&= [\![\tau(\alpha_n)]\!]_\Theta \circ \cdots \circ [\![\tau(\alpha_1)]\!]_\Theta \\
&= [\![\tau(\alpha_1) \bowtie \cdots \bowtie \tau(\alpha_n)]\!]_\Theta \\
&= [\![\tau(N)]\!]_\Theta
\end{aligned}$$

As a consequence, the transformation function τ can also be inductively defined using the following three atomic TADS



(a) identity ε

(b) affine function with bias

(c) ReLU ϕ_1^2

Fig. 6 A few examples for atomic TADS. The input vector is given as $\vec{x} = (a, b)$ with $a, b \in \mathbb{R}$

- identity ε
- affine functions $\alpha \colon \mathbb{R}^n \to \mathbb{R}^m$ where $n, m \in \mathbb{N}$
- single ReLUs ϕ_i^n where $n, i \in \mathbb{N}$, $i \leq n$

which are illustrated in Fig. 6.

Corollary 7 (Inductive definition of τ)
The transformation of a network to a TADS can be defined inductively as

$$\tau'(\varepsilon) = \varepsilon$$

$$\tau'(\alpha\, ; N) = \alpha \bowtie \tau'(N)$$

$$\tau'(\phi_i^k\, ; N) = \left(\vec{e}_i^{\,\top} \cdot \vec{x} \geq 0, \boldsymbol{I}^k, \boldsymbol{E}_i^k\right) \bowtie \tau'(N)$$

such that

$$\tau'(N) = \tau(N).$$

This consistency of viewpoints and operational handling indicates that the TADS setup is natural, and that it supports to approach PLNN analysis and explanation from various perspectives.

7 TADS at work

In this section, we continue the discussion of the XOR function as a minimal example to showcase the power of TADS for:

- **Model Explanation.** For a given PLNN, describe precisely its behavior in a comprehensible manner. This allows for a semantic comparison of PLNNs comprising (approximative) semantic equivalence with precise diagnostic information in case of violation.
- **Class Characterization.** PLNNs are frequently extended by the so-called argmax function to be used as classifiers. TADS-based class characterization allows one to precisely characterize the set of inputs that are specifically classified, or the set of inputs where two (PLNN-based) classifier differ.
- **Verification.** Verification is beyond the scope of this section, but will be discussed in [40] in the setting of digit recognition.

In the remainder of this section, we focus on the impact of Model Explanation and Class Characterization. Two properties of TADS are important here:

Compositionality. Due to the compositional nature of TADS, any TADS that represents a given PLNN can be modified and extended by *output interpretation* mechanisms. This mirrors a very important use case of neural networks; while neural networks are fundamentally functions $\mathbb{R}^n \to \mathbb{R}^m$, they are often used for discrete problems, which requires a different interpretation of their output.

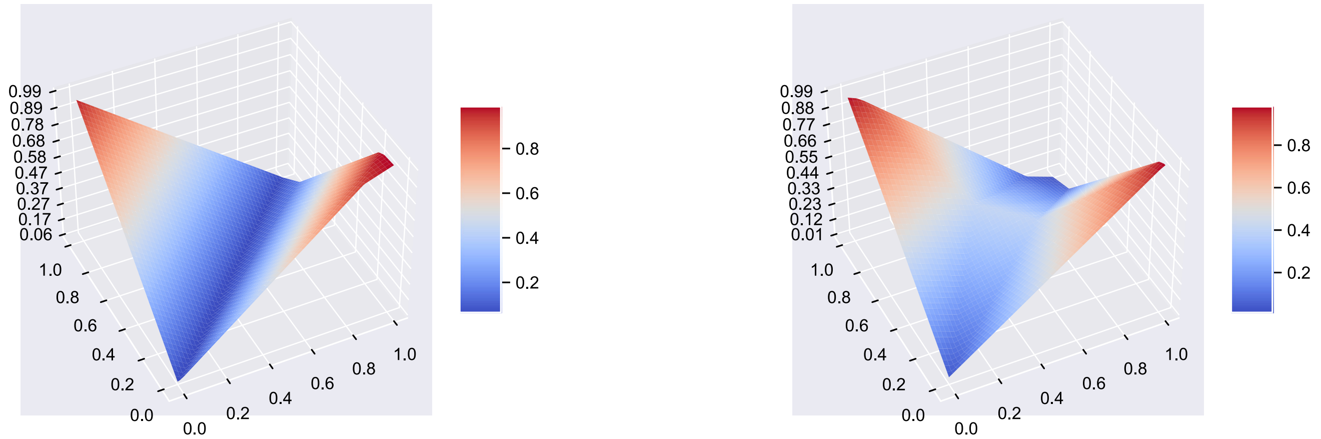
Precision. As the TADS transformation of a PLNN is semantics-preserving, all results are precise.

Based on these properties, it is possible to solve all the aforementioned problems elegantly by simple algebraic transformations of TADS.

7.1 Model explanation and algebraic implications

To start, we train a small neural network to solve the continuous XOR problem. The resulting network, N_1, represents a continuous function $f_1 = [\![N_1]\!]_{\mathrm{DS}}$ (see, Fig. 7a). N_1 solves the XOR problem relatively well, with all corners being within a distance of < 0.1 to the desired values of 1 and 0 respectively.

The architecture of N_1 is shown in Fig. 9. Note that this architecture is much bigger than the architecture for N_* (cf., Sect. 5). This is needed as the training procedure is approximate and does not reach a global optimum. On all substantially smaller networks, we failed to train a network that was close to the specifications of XOR.



(a) Graph of N_1

(b) Graph of N_2

Fig. 7 Function graphs corresponding to the PLNNs N_1 and N_2. Observe that both PLNNs fulfill the conditions of the XOR problem very closely

7.1.1 Model explanation

First, we consider full model explanation of N_1.[11] We can attain a precise and complete characterization of f_1 by creating the corresponding TADS $t_1 = \tau(N_1)$, as shown in Fig. 8a. This TADS describes precisely and completely the behavior of f_1 in a whitebox manner.

Similarly to the function plot shown in Fig. 7a, the TADS gives a comprehensible view of f_1. In contrast to the mere function plot, the TADS of Fig. 8a is a solid basis for further systematic analyses and extends to more than two dimensions.

Model Explanation as illustrated here is the basic use case of a TADS as a white-box model for PLNNs, however, the true power of TADS becomes apparent when used for high-level analyses using algebraic operations on TADS.

7.1.2 Algebraic implications

As mentioned in the last section, the training process of neural networks is approximate and can lead to many different solutions. A very natural question to ask is: “How differently do two neural networks solve the same problem?”. This question can be answered using algebraic operations on TADS.

Consider N_2, a PLNN that has also been trained with the network architecture shown in Fig. 9, but with a different setting of hyperparameters.[12] Its represented (semantic) function $f_2 = [\![N_2]\!]_{\text{DS}}$ is depicted in Fig. 7b and the corresponding TADS $t_2 = \tau(N_2)$ in Fig. 8b.

As TADS form a linear algebra, one can easily mirror the computation $f_2 - f_1$ by $t_2 - t_1$ on TADS level. The result is identical because the transformation process is precise, i.e.,

$$[\![N_2]\!]_{\text{DS}} - [\![N_1]\!]_{\text{DS}} = f_2 - f_1 = [\![t_2 - t_1]\!]_{\ominus}$$

The resulting difference TADS t_3 is shown in Fig. 10 and the corresponding function graph in Fig. 11.

t_3 is ideal to study the semantic difference between PLNN N_1 andN_2. Most interestingly, as can be visually seen in Fig. 11, the largest differences between both networks occur in the middle of the function domain, i.e., in the region most distant from the edges where the XOR problem is clearly defined. This matches the intuition that points further away from known points are more uncertain under neural network training.

Further, observe that the difference of both networks yields a TADS of roughly double the size. This moderate increase in size indicates the similarity of N_1 and N_2, as linear regions of the difference TADS t_3 result from the intersection of the regions for f_1 and f_2 which could, in the worst case, grow quadratically.

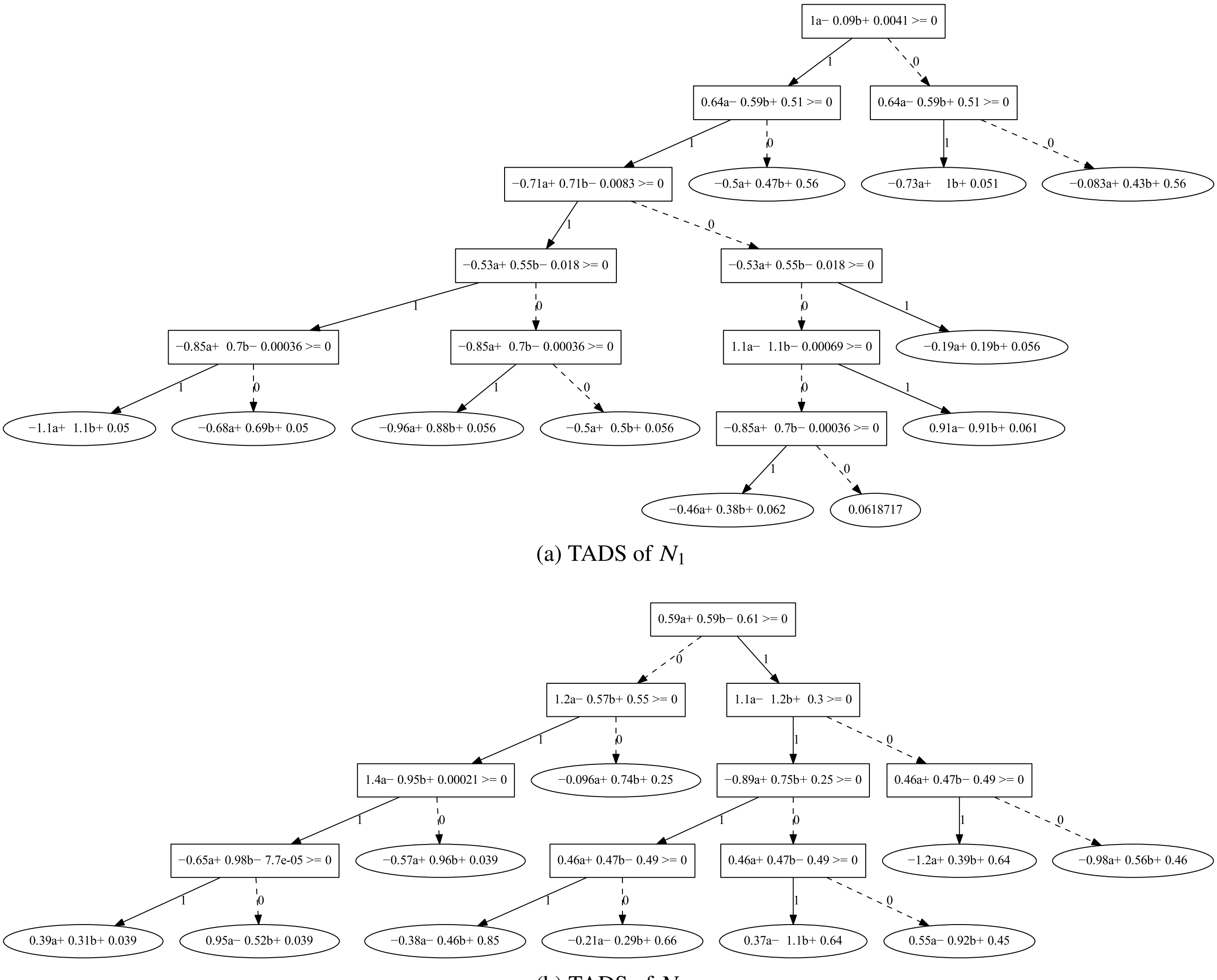
As mentioned above (cf., Corollary 5), it is also possible to analyse ϵ-similarity via algebraic operations to, in this case, obtain the TADS shown in Fig. 12, which is much smaller than the full difference TADS (cf., Fig. 10). The piece-wise affine function of this TADS is visualised in Fig. 13.

There are 8 regions in which the difference values exceed 0.3, all close to the center of $[0, 1]^2$. This, again matches the intuition that the volatility of solution grows with the distance to the defined values.

This result is interesting as it shows that, while the two neural networks that we trained differ, they do not differ more than 0.3 except for a small region in the center of the input

[11] Of course, in this two-dimensional case, a function plot akin to Fig. 7a might seem sufficient, but this is not feasible in anything beyond two-dimensional problems.

[12] The discussion of the learning process is beyond the scope of this paper.



(a) TADS of N_1

(b) TADS of N_2

Fig. 8 TADS corresponding to the PLNNs N_1 and N_2. Note that both TADS are a full characterization of the semantic functions f_1 and f_2, respectively

space. Similar constructions can be used to analyze robustness of neural networks. Robustness of neural networks is of large interest to neural network research [10] and the application of TADS to this problem is discussed in more detail in [40].

7.2 Classification

Applications of neural networks are traditionally split into *regression tasks* and *classification tasks*. In regression tasks, one seeks to approximate a function with continuous values, whereas classification tasks have discrete outputs. As learned, piece-wise linear functions are inherently continuous, classification tasks require an additional step that *interprets* the continuous output of a neural network as one of multiple discrete classes. Note that this is a change of mindset, with the same neural network being interpreted differently depending on the context.

In our context, one might be interested in a model that classifies each input point $\vec{x} \in \mathbb{R}^2$ as either 1 or 0 instead of returning a real-value.

7.2.1 Class characterization

A standard method for classification tasks is the interpretation of neural network outputs as a probability distribution over classes [17]. In our XOR example, it is natural to interpret $f_1(\vec{x})$ as the probability of $\vec{x}$ belonging to class 1 and $1 - f_1(\vec{x})$ as the probability of $\vec{x}$ belonging to class 0.

At evaluation time, one might naturally choose the class with the highest probability. Thus, N_1's output is set to 1 if it is greater than 0.5 and 0 otherwise, which is, actually, in

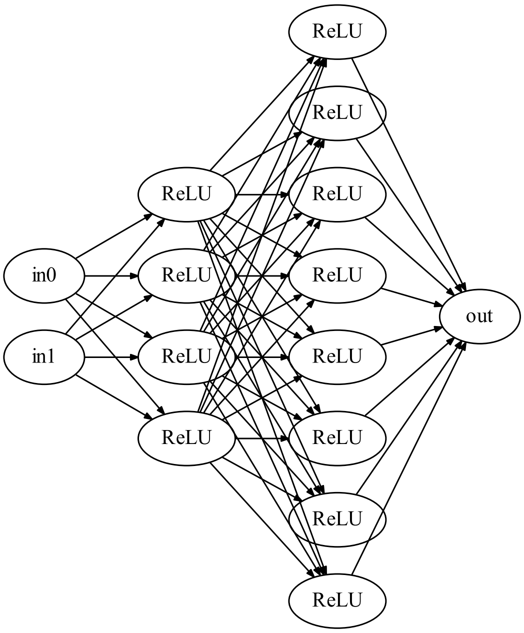


Fig. 9 The architecture of the networks N_1 and N_2. Weights and biases are omitted for brevity

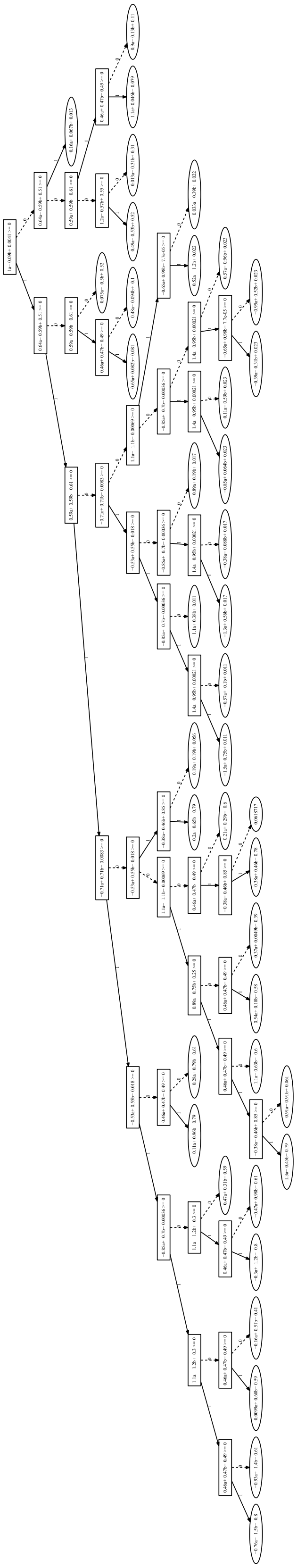


Fig. 10 The TADS $t_2 - t_1$ describing the difference of f_2 and f_1. This TADS corresponds to the function plot of Fig. 11

line with the definition of g_*. Applying

$$\mathbb{I}(x \geq 0.5) = \begin{cases} 1 & \text{if } x \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

to the continuous learned function f_1 therefore results in a suitable classifier for this problem:

$$g = \mathbb{I}(x \geq 0.5) \circ f_1$$

Note that $\mathbb{I}(x \geq 0.5)$ is not continuous and therefore cannot be represented by a PLNN.[13]

To construct the TADS, we use the compositionality of TADS. We manually construct the simple TADS $\tau(\mathbb{I}(x \geq 0.5))$, as shown in Fig. 15 and compose it with the TADS t_1 of f_1. The resulting TADS

$$t_1^c = \tau(\mathbb{I}(x \geq 0.5)) \bowtie t_1$$

is shown in Fig. 14a. Note that this TADS is reminiscent of a binary decision diagram with just two final nodes. Figure 16a and shows precisely which inputs are interpreted as 1 and which as 0. As we only have two classes here, this classification can be considered as what is called *class characterization* in [21] for both classes 0 and 1. Please note that class characterizations allows one to change the perspective

[13] This is a general observation that holds for all discrete valued classification tasks. Most notably, the argmax function, a standard method for n-ary classification, also cannot be represented by a PLNN and must be handled on the side of TADS. More discussions on the role of argmax in classification can be found in [40].

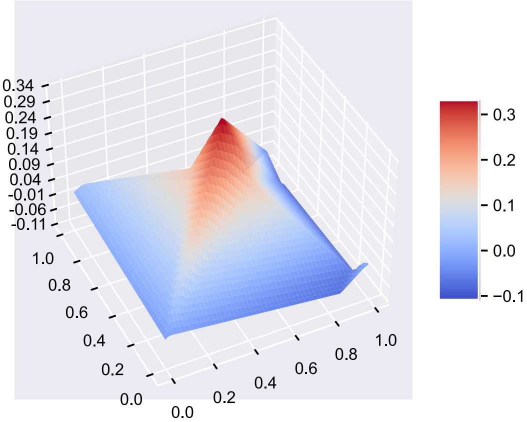


Fig. 11 The function plot describing the difference between the two networks N_1 and N_2

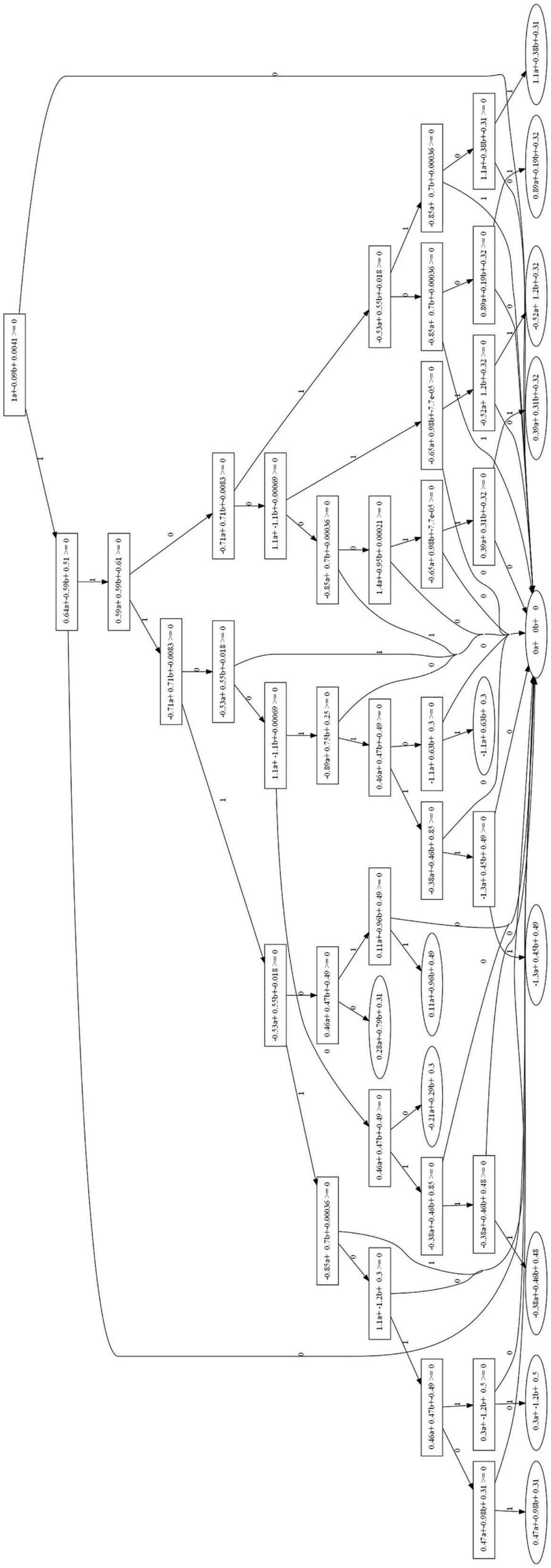


Fig. 12 A TADS describing the difference between f_2 and f_1 iff it exceeds $\epsilon = 0.3$

from a classification task to the task of finding adequate candidates of specific profile, here given as a corresponding class.

This shows that, given an output interpretation that maps the continuous network outputs to discrete classes, it is possible to transform neural networks, fundamentally blackbox representations of real valued functions, into semantically equivalent decision diagrams, fundamentally whitebox representations of discrete valued functions.

7.2.2 Comparison of classifiers

After having constructed TADS that characterize the classification behavior of neural networks, we can also characterize the *difference* in classification behavior of two neural networks. We can simply do so by using the lifted equality relation to the TADS level and compute the TADS:

$$t_1^c \circledequal t_2^c$$

The resulting TADS is shown in Fig. 17a and the corresponding function graph in Fig. 16c. This plot describes precisely the areas where both functions differ and where they coincide.

Notably, it shows that, while the absolute difference of f_1 and f_2 is highest in the center of the interval $[0,1]^2$, the networks agree in that area with respect to classification. Indeed, it appears that the largest difference with respect to classification occurs in the diagonals separating the classes 1 and 0. This is not too surprising, as it is at the borderline between classes were small changes may affect the classification result.

Using an encoding of boolean values as 1 and 0 respectively, we can also compute the difference of t_1^c and t_2^c

$$t_1^c \ominus t_2^c$$

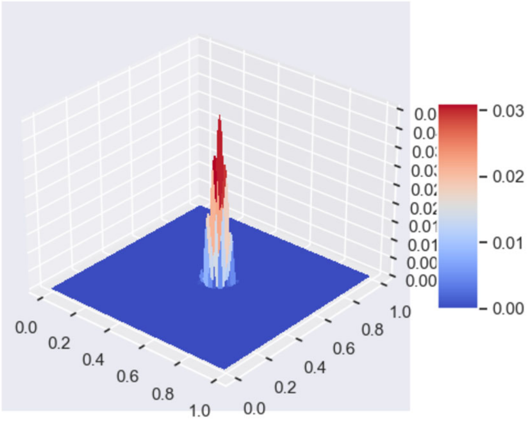


Fig. 13 The function graph describing the difference between f_2 and f_1 iff it exceeds $\epsilon = 0.3$

This TADS not only describes where t_1^c and t_2^c disagree, but also how they disagree. The corresponding TADS is shown in Fig. 17b.

This shows the utility of TADS for output interpretation. While the absolute difference between two networks is a suitable measure of difference for *regression* tasks, the difference of the classification functions is suitable for *classification*. Playing with this difference, e.g., by modifying the classification function, is a powerful analytical instrument. E.g., in settings with many classes, separately analyzing the class characterizations of the individual classes typically leads to much smaller and easier to comprehend TADS and may therefore be a good means to increase scalability.

In machine learning, one often compares learned classifiers to groundtruth solutions by sampling from the groundtruth solution and checking whether the neural network matches the groundtruth predictions. TADS enable a straightforward and precise way of evaluating a neural network in instances where one has access to the groundtruth model. E.g., the TADS of Fig. 18 precisely specifies where N_1 differs from the baseline solution $\mathbb{I}(|x - y| \geq 0.5)$.

8 Related work

The presented TADS-based approach towards understanding of neural networks is explicitly meant to bridge between the various existing initiative that aim in the same direction, but typically with quite different means. In this section, we review the state of the art under three perspectives:

- The intent, explainability, as approached in the neural networks community.
- Applied concepts, e.g., symbolic execution that aim at (locally) precise results.
- Applied background, in particular concerning piece-wise affine functions.

Whereas the first perspective (Sect. 8.1) is conceptually distant, both in its applied technologies as well as in its achievements, the mindset of second perspective (Sect. 8.2) is similar in aims and means, but, except for our previous work, restricts its attention to a locally precise analysis close to some (partly symbolic) input. The third perspective just concerns the mathematical background (Sect. 8.3). We are not aware of any previous work that systematically applies algebraic reasoning to achieve precise explanation and verification results about neural networks.

8.1 Machine learning explainability

In recent years, explainable machine learning (XML) as a subfield of machine learning has seen a surge of activity. In line with existing machine learning research, XML focuses on approaches that scale efficiently at the cost of precision and comprehensibility.

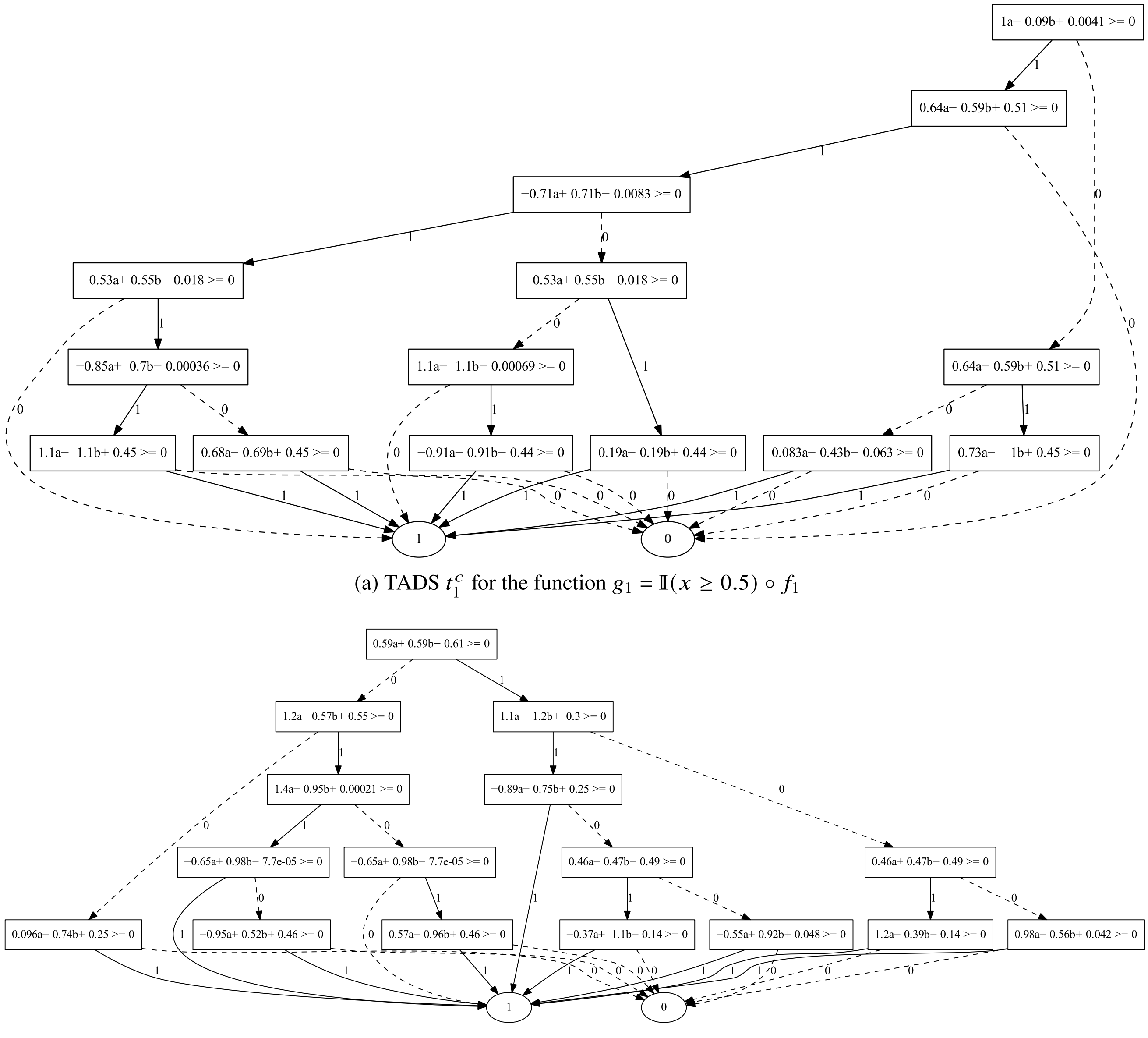
Due to vast amount of work in this direction, we can only provide sketch of the field here, which from our perspective is characterized by is use of ‘traditional’ deep learning technologies such as gradient based optimization and its focus on directly investigating the neural networks themselves in an approximative fashion and without explicit link to some semantic model.

A typical example of a gradient-based method is activation maximization [34, 47], which seeks to find, for one class C and network N, the input $\vec{x}$ for which $[\![\vec{x}]\!]$ is maximal for class C. Being based on gradient based optimization, this approach is clearly approximate.

Other examples of approaches working on the neural network level are frequently found in attribution methods. Attribution methods focus on attributing a prediction $[\![N]\!](\vec{x}) = y$, to parts of the input that one deems responsible for this prediction. In general, this question is unclear and subjective. As a consequence, there exist multiple different methods that attribute the prediction differently to the original input. Examples include gradient based saliency maps [38, 51], layer-wise relevance propagation (LRP) [3] and deep Taylor decomposition [36]. As attribution is natural to answer for linear models, these methods focus on linearly approximating the model (gradient based saliency maps) or parts of the model (LRP and deep Taylor decomposition). The latter methods depend strongly on the neural network architecture and not on its semantics.

These methods are useful to gain rough intuition, but they do not offer any guarantees or reliable results. This is a direct consequence of most of these methods working with classical machine learning tools such as backpropagation and numerical optimization, which are fast but approximative [17].

The class of XML methods that is most closely related to TADS are local proxy models like LIME [45] and SHAP [32]. Both methods consider one fixed input $\vec{x}$ and



(a) TADS t_1^c for the function $g_1 = \mathbb{I}(x \geq 0.5) \circ f_1$

(b) TADS t_2^c for the function $g_2 = \mathbb{I}(x \geq 0.5) \circ f_2$

Fig. 14 Classification TADS that indicates where the PLNNs N_1 and N_2 output a value greater than 0.5

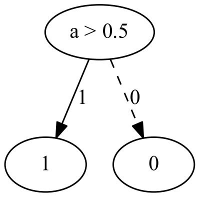
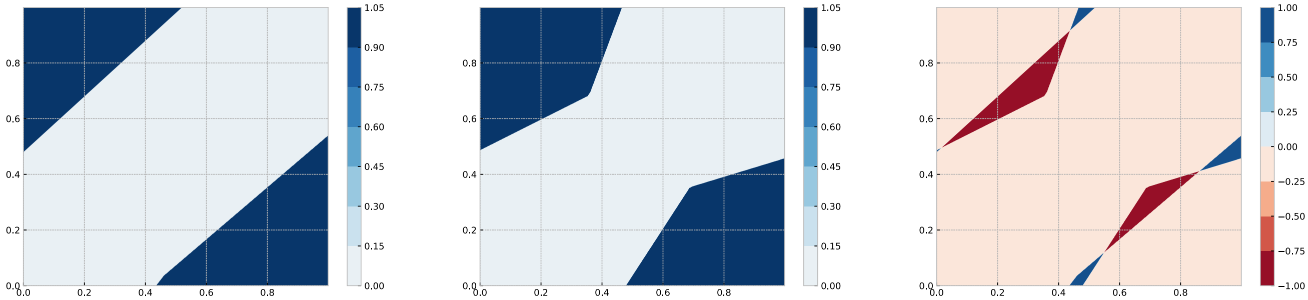


Fig. 15 A TADS describing the function $\mathbb{I}(x \geq 0.5)$ that is used to transform the output of neural networks into discrete values

treat the model as a blackbox. They observe the model's behavior on multiple perturbations of the form $\vec{x} + p$ with p being sampled randomly. Then, they use simple machine learning models such as a linear classifier or a decision tree to describe the model's behavior on the perturbations $\vec{x} + p$ they observed.

These methods are similar in their intent to the TADS approach, as they use conceptually simple models to represent the blackbox behavior of a neural network. However, both LIME and SHAP are imprecise. They both sample only a few points $\vec{x} + p$ in the neighborhood of $\vec{x}$ and might miss important properties of the neural network model. Further, both methods use a machine learning classifier to represent these points. These classifiers are usually linear models (or a comparably simple model) and cannot capture the full behavior of the network, which leads to potentially large and uncontrolled errors.



(a) $g_1 = \mathbb{I}(x \geq 0.5) \circ f_1$ (cf. Figure 14a) (b) $g_2 = \mathbb{I}(x \geq 0.5) \circ f_2$ (cf. Figure 14b) (c) $[\![t_1^c \ominus t_2^c]\!]_\ominus$ (cf. Figure 17b)

Fig. 16 Contour plots of the classification functions (**a**) g_1 and (**b**) g_2. The difference between both classifiers is visualized in (**c**)

We are not aware of XML methods that provide guarantees strong enough to justify a responsible use in safety critical applications.

8.2 Conceptually related work

Symbolic execution of neural networks More closely related to TADS are approaches to explainability based on symbolic execution of neural networks.

The idea of explainability via symbolic (or rather concolic execution) of neural networks was already explored in the works of [19]. In their work, the authors translate a given PLNN into an imperative program and concolically execute one given input $\vec{x} \in \mathbb{R}^n$. This corresponds to exploring the one path of a TADS corresponding to $\vec{x}$. This yields the path condition and the affine transformation that are responsible for the prediction $\mathcal{N}(\vec{x})$. The authors further use these explanations to find adversarial examples and find parts of an input that they deem important for a given classification. The results of this work are promising, but (very) local, as they restrict themselves to one linear region of the input.

The authors of [12] propose a method that closely mirrors the method of [18]. In essence, both methods are almost identical, but differ in their conceptual derivation of the method. The authors of [12] also consider sets of predictions and work out what features act as discriminators in many of these predictions.

Moving from the idea of explanation, the authors of [54] consider concolic testing instead. Similar to the work of [18], they execute singular inputs concolically. They use the results from concolic execution to heuristically derive new inputs that cover large areas of the input space.

TADS improve on these approaches in two ways. First, TADS offer a global viewpoint on neural network semantics, independent of a sample set. Second, TADS support algebraic operations on a conceptual level to derive globally precise explanation and verification results. As illustrated in Sect. 7, algebraic operations nicely serve as a toolbox to derive tailored and precise analyses.

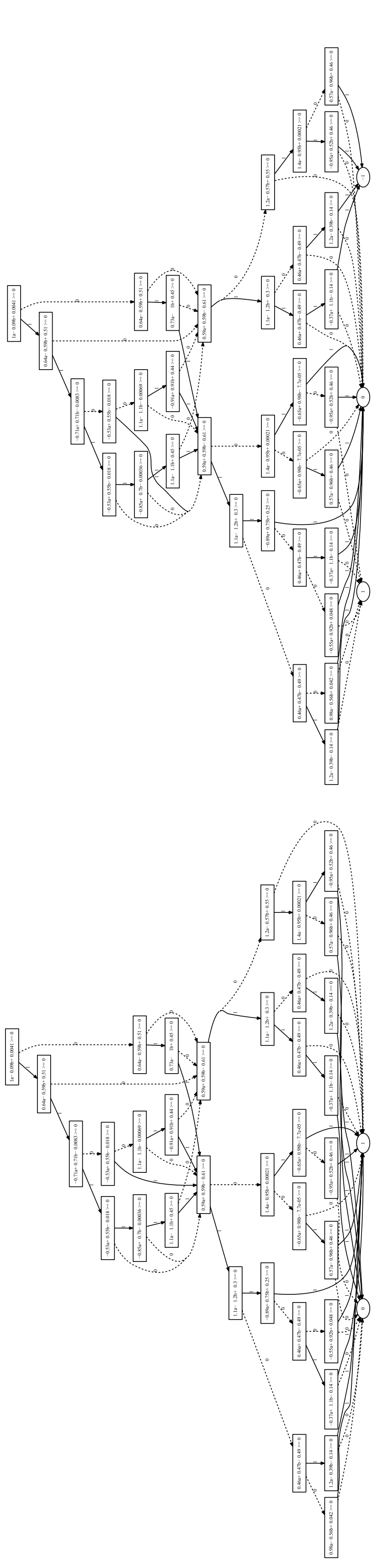
Neural network verification Neural network verification aims to verify properties of neural networks, usually piece-wise linear neural networks using techniques from SMT-solving and abstract interpretation extended by domain specific techniques [28, 59, 61]. Verification approaches are usually precise, or at least provide a counterexample if a property is shown false. Modern solvers can scale quite well, but are still far from being able to tackle practically relevant application [6].

Verification approaches are related to TADS as they also provide tools for the precise analysis of piece-wise linear neural networks. However, while SMT-based verification approaches currently scale better than TADS, they focus only on binary answers to a verification problem. They are not able to provide full diagnostics and descriptions of where and how an error occurs. However, please note, SMT-based approaches should not be considered as an alternative but rather as a provider of technologies that can also be applied at the TADS level. In fact, we use SMT solving, e.g., to eliminate infeasible paths in TADS.

Precise explainability of random forests This work is conceptually closely related to and builds upon the work of [21, 39]. There, ADDs are used and extended to derive explainable global models for random forests. Similar to the approach in this paper, these models are derived through a sequence of semantics-preserving transformations and later on refined by performing algebraic operations on the white-box representation of random forests. In fact, considering the underlying mindset, the work in this paper can be regarded as an extension of our work on random forest to neural networks. However, the much higher complexity of PLNN requires substantial generalization, which to our surprise did not clutter the theory, but rather added to its elegance.

8.3 Technologically related work

Linear regions of neural networks Vast amounts of research have been conducted regarding the number and shape



(a) TADS t_1^c ⊜ t_2^c

(b) TADS $t_1^c \ominus t_2^c$

Fig. 17 TADS representing the difference of the classification behavior of N_1 and N_2. The TADS in (**a**) expresses the equality of the classifications while (**b**) expresses the difference. Thus, by mapping the nodes −1 and 1 to 0 and 0 to 1 one can transform (**b**) to (**a**). Note, however, that despite their syntactical similarities, they represent different concepts

of linear regions in a given PLNN [1, 12, 24, 25, 27, 37, 43, 44, 48, 53, 62, 63]. Linear regions are of huge interest to neural network research as they give a natural characterization of the expressive power of neural network classes. This research is beneficial to the understanding of TADS as it can be used to bound the size of TADS and understand where and when explosions and size occur. On the other hand, TADS give a precise and minimal representation of the linear regions belonging to given neural network and can be used to facilitate experiments in this field, e.g., to find a linear region containing a negative example for a given property that could not be verified [28].

Structures for polyhedral sets At their core, TADS are efficient representations of multiple polyhedral regions within high-dimensional spaces. Similar problems occur in other divisions of computer science, most notably computer graphics.

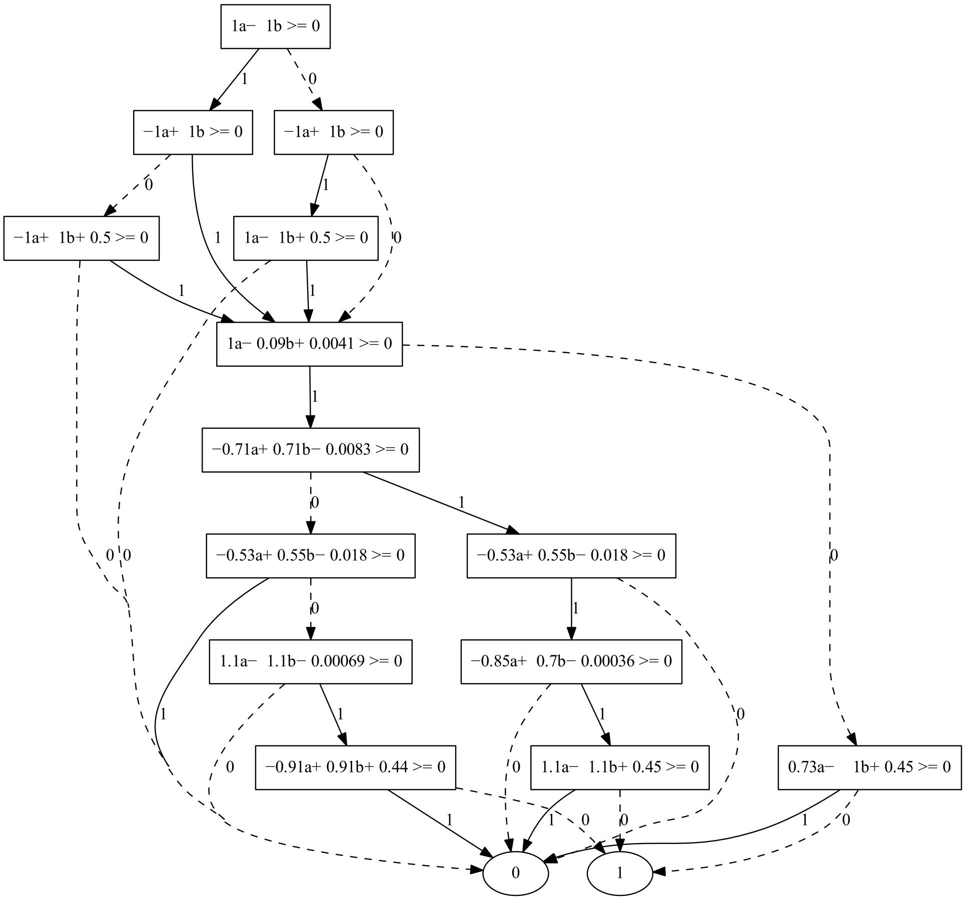
TADS are closely related to Binary Space Partition Trees (BSP-trees) [56] and comparable structures [58]. These structures are built to represent a partition of a real-dimensional space into polygons, much like TADS do. TADS extend these structures with optimizations from ADDs to account for domain-specific properties of piece-wise linear functions that are not present in the general case of polygonal partitions.

9 Conclusion and future work

We have presented an algebraic approach to the precise and global explanation of Rectifier Neural Networks (PLNNs), one of the most popular kinds of Neural Networks. Key to our approach is the symbolic execution of these networks that allows the construction of semantically equivalent *Typed Affine Decision Structures* (TADS). Due to their deterministic and sequential nature, TADS can be considered as white-box models and therefore as precise solutions to the model explanation problem, which directly imposes also solutions to the outcome explanation, and class characterization problems [21, 22]. Moreover, as linear algebras, TADS support operations that allows one to elegantly compare Rectifier Networks for equivalence or ϵ-similarity, both with precise diagnostic information in case of failure, and to characterize their classification potential by precisely characterizing the set of inputs that are specifically classified, or the set of inputs where two Network-based classifiers differ. These are steps towards a more rigorous understanding of Neural Networks that is required when applying them in safety-critical domains without the possibility of human interference, such as self-driving cars.

This elegant situation at the semantic TADS level is in contrast with today's practical reality where people directly work

Fig. 18 The TADS describing the difference between the baseline solution $\mathbb{I}(|x-y| \geq 0.5)$ and the solution learned by N_1



with learned PLNNs that are in particular characterized by their hidden layers that often comprise millions sometimes even billions of parameters. The reason for this complex structure is learning efficiency, a property paid for with semantic intractability: There is essentially no way to control the impact of minor changes of a parameter or input values, and even the mere evaluation for a sample input exceed the capacity of a human's mind by far. This is why PLNNs are considered as black-box models.

The reason why TADS have not yet been studied may be due to their size: they may be exponentially larger than a corresponding PLNN. The reason for this expansion is the transformation of the incomprehensible hidden layers structure into a large decision structure, which conceptually is as easy to comprehend as a decision tree and a linear classifier. In this sense, our transformation into TADS can be regarded as trade of size for transparency, turning the verification and explanation problem into a scalability issue. There are at least three promising angles for attacking the scalability problem:

1. Learned PLNN have a high amount of noise resulting from the underlying learning process that works by locally optimizing individual parameters of the hidden layers. Noise reduction may have a major impact on size. Detecting noise is clearly a semantic task and can therefore profit from TADS-based semantic analyses.
2. PLNN are accepted to be approximative. Thus, controlled modifications with minor semantic impact are easily tolerated. TADS provide the means to control the effect of modifications and thereby to keep modifications in the tolerable range.
3. Modern neural network architectures are typically compositions of multiple sub-networks that are intended to support the learning of different subtasks. However, this structure at the representational layer gets semantically blurred during joint the learning process, which, e.g., prohibits compositional approaches as known from formal methods. The semantic transparency of TADS may provide means to reinforce the intended compositional structure also at the semantical level in order to support compositional reasoning and incremental construction.

Of course, there seems to be a hen/egg problem here. If we can construct the TADS, we are able to reduce it in order to achieve scalability. On the other hand, we need scalability first to construct the TADS. This is a well-known problem in the formal methods world, and despite a wealth of heuristics and domain-specific technologies, the answer is *compositionality* and *incremental construction*. This is exactly in line with the observation reported in the third item above: We need to learn how to use divide and conquer techniques for PLNN in a semantics-aware fashion. TADS are designed to support this quest by providing both a leading mindset and a tool-supported technology.

Funding Note Open Access funding enabled and organized by Projekt DEAL.

References

1. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. arXiv preprint (2016). arXiv:1611.01491
2. Axler, S.: Linear Algebra Done Right. Springer, Berlin (1997)
3. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLoS ONE **10**(7), e0130140 (2015)
4. Badue, C., Guidolini, R., Carneiro, R.V., Azevedo, P., Cardoso, V.B., Forechi, A., Jesus, L., Berriel, R., Paixao, T.M., Mutz, F., et al.: Self-driving cars: a survey. Expert Syst. Appl. **165**, 113816 (2021)
5. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebric decision diagrams and their applications. Form. Methods Syst. Des. **10**(2), 171–206 (1997)
6. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (VNN-COMP 2021): summary and results. arXiv preprint (2021). arXiv:2109.00498
7. Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al.: Dota 2 with large scale deep reinforcement learning. arXiv preprint (2019). arXiv:1912.06680
8. Brondsted, A.: An Introduction to Convex Polytopes, first edn. Springer, New York (1983). https://doi.org/10.1007/978-1-4612-1148-8
9. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Adv. Neural Inf. Process. Syst. **33**, 1877–1901 (2020)
10. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. IEEE (2017)
11. Chiu, C.C., Sainath, T.N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R.J., Rao, K., Gonina, E., et al.: State-of-the-art speech recognition with sequence-to-sequence models. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4774–4778. IEEE (2018)
12. Chu, L., Hu, X., Hu, J., Wang, L., Pei, J.: Exact and consistent interpretation for piecewise linear neural networks: a closed form solution. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1244–1253 (2018)
13. Clarke, L.A.: A system to generate test data and symbolically execute programs. IEEE Trans. Softw. Eng. **3**, 215–222 (1976)
14. Doran, D., Schulz, S., Besold, T.R.: What does explainable AI really mean? A new conceptualization of perspectives. arXiv preprint (2017). arXiv:1710.00794
15. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, pp. 315–323 (2011)
16. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint (2014). arXiv:1412.6572
17. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
18. Gopinath, D., Wang, K., Zhang, M., Pasareanu, C.S., Khurshid, S.: Symbolic execution for deep neural networks. arXiv preprint (2018). arXiv:1807.10439
19. Gopinath, D., Pasareanu, C.S., Wang, K., Zhang, M., Khurshid, S.: Symbolic execution for attribution and attack synthesis in neural networks. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 282–283. IEEE (2019)
20. Gorokhovik, V.V., Zorko, O.I., Birkhoff, G.: Piecewise affine functions and polyhedral sets. Optimization **31**(3), 209–221 (1994)
21. Gossen, F., Steffen, B.: Algebraic aggregation of random forests: towards explainability and rapid evaluation. Int. J. Softw. Tools Technol. Transf. (2021). https://doi.org/10.1007/s10009-021-00635-x
22. Gossen, F., Margaria, T., Steffen, B.: Formal methods boost experimental performance for explainable AI. IT Prof. **23**(6), 8–12 (2021)
23. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv. **51**(5), 1–42 (2018). https://doi.org/10.1145/3236009
24. Hanin, B., Rolnick, D.: Complexity of linear regions in deep networks. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 2596–2604 (2019), PMLR. https://proceedings.mlr.press/v97/hanin19a.html
25. Hanin, B., Rolnick, D.: Deep ReLU networks have surprisingly few activation patterns. Advances in Neural Information Processing Systems, vol. 32 (2019)
26. He, J., Li, L., Xu, J., Zheng, C.: ReLU deep neural networks and linear finite elements. arXiv preprint (2018). arXiv:1807.03973
27. Hinz, P.: Using activation histograms to bound the number of affine regions in ReLU feed-forward neural networks (2021). arXiv:2103.17174 [abs]
28. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: International Conference on Computer Aided Verification, pp. 97–117. Springer, Berlin (2017)
29. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976)
30. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint (2014). arXiv:1412.6980
31. Linardatos, P., Papastefanopoulos, V., Kotsiantis, S.: Explainable AI: a review of machine learning interpretability methods. Entropy **23**(1), 18 (2021)
32. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. Advances in Neural Information Processing Systems, vol. 30 (2017)

33. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint (2017). arXiv:1706.06083
34. Mahendran, A., Vedaldi, A.: Visualizing deep convolutional neural networks using natural pre-images. Int. J. Comput. Vis. **120**(3), 233–255 (2016)
35. Minsky, M., Papert, S.: Perceptrons (1969)
36. Montavon, G., Lapuschkin, S., Binder, A., Samek, W., Müller, K.R.: Explaining nonlinear classification decisions with deep Taylor decomposition. Pattern Recognit. **65**, 211–222 (2017)
37. Montufar, G.F., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. Advances in Neural Information Processing Systems, vol. 27 (2014)
38. Mundhenk, T.N., Chen, B.Y., Friedland, G.: Efficient saliency maps for explainable AI. arXiv preprint (2019). arXiv:1911.11293
39. Murtovi, A., Nolte, G., Schlüter, M., Bernhard, S.: Forest Gump: a tool for verification and explanation. Int. J. Softw. Tools. Technol. Transf. (2023, in this issue). https://doi.org/10.1007/s10009-023-00702-5
40. Nolte, G., Schlüter, M., Murtovi, A., Bernhard, S.: The power of Typed Affine Decision Structures: a case study. Int. J. Softw. Tools. Technol. Transf. (2023, in this issue). https://doi.org/10.1007/s10009-023-00701-6
41. Oh, K.S., Jung, K.: GPU implementation of neural networks. Pattern Recognit. **37**(6), 1311–1314 (2004)
42. Ovchinnikov, S.: Discrete piecewise linear functions. Eur. J. Comb. **31**(5), 1283–1294 (2010). https://doi.org/10.1016/j.ejc.2009.11.005
43. Pascanu, R., Montufar, G., Bengio, Y.: On the number of response regions of deep feed forward networks with piece-wise linear activations. arXiv preprint (2013). arXiv:1312.6098
44. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Sohl-Dickstein, J.: On the expressive power of deep neural networks. In: International Conference on Machine Learning, pp. 2847–2854. PMLR (2017)
45. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should i trust you?" Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144 (2016)
46. Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint (2016). arXiv:1609.04747
47. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-CAM: visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 618–626 (2017)
48. Serra, T., Tjandraatmadja, C., Ramalingam, S.: Bounding and counting linear regions of deep neural networks. In: International Conference on Machine Learning, pp. 4558–4566. PMLR (2018)
49. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
50. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint (2014). arXiv:1409.1556
51. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: visualising image classification models and saliency maps. arXiv preprint (2013). arXiv:1312.6034
52. Sober, E.: Ockham's Razors. Cambridge University Press, Cambridge (2015)
53. Sudjianto, A., Knauth, W., Singh, R., Yang, Z., Zhang, A.: Unwrapping the black box of deep ReLU networks: interpretability, diagnostics, and simplification (2020). arXiv:2011.04041 [abs]
54. Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., Kroening, D.: Concolic testing for deep neural networks. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 109–119 (2018)
55. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint (2013). arXiv:1312.6199
56. Thibault, W.C., Naylor, B.F.: Set operations on polyhedra using binary space partitioning trees. In: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 153–162 (1987)
57. Tjoa, E., Guan, C.: A survey on explainable artificial intelligence (XAI): toward medical XAI. IEEE Trans. Neural Netw. Learn. Syst. **32**(11), 4793–4813 (2020)
58. Tøndel, P., Johansen, T.A., Bemporad, A.: Evaluation of piecewise affine control via binary search tree. Automatica **39**(5), 945–950 (2003). https://doi.org/10.1016/S0005-1098(02)00308-4
59. Tran, H.D., Manzanas Lopez, D., Musau, P., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Star-based reachability analysis of deep neural networks. In: International Symposium on Formal Methods, pp. 670–686. Springer, Berlin (2019)
60. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
61. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-crown: efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. arXiv preprint (2021). arXiv:2103.06624
62. Woo, S., Lee, C.L.: Decision boundary formation of deep convolution networks with ReLU. In: 2018 IEEE 16th Intl. Conf. on Dependable, Autonomic and Secure Computing, 16th Intl. Conf. on Pervasive Intelligence and Computing, 4th Intl. Conf. on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp. 885–888. IEEE (2018)
63. Zhang, X., Wu, D.: Empirical studies on the properties of linear regions in deep neural networks. arXiv preprint (2020). arXiv:2001.01072