International Journal on Software Tools for Technology Transfer (2023) 25:355–374
https://doi.org/10.1007/s10009-023-00701-6

EXPLANATION PARADIGMS LEVERAGING ANALYTIC INTUITION

Special Section: Introducing Explanation Paradigms Leveraging Analytic Intuition

The power of typed affine decision structures: a case study

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

Accepted: 19 February 2023 / Published online: 21 April 2023
© The Author(s) 2023

Abstract

TADS are a novel, concise white-box representation of neural networks. In this paper, we apply TADS to the problem of neural network verification, using them to generate either proofs or concise error characterizations for desirable neural network properties. In a case study, we consider the robustness of neural networks to adversarial attacks, i.e., small changes to an input that drastically change a neural networks perception, and show that TADS can be used to provide precise diagnostics on how and where robustness errors a occur. We achieve these results by introducing *Precondition Projection*, a technique that yields a TADS describing network behavior precisely on a given subset of its input space, and combining it with PCA, a traditional, well-understood dimensionality reduction technique. We show that PCA is easily compatible with TADS. All analyses can be implemented in a straightforward fashion using the rich algebraic properties of TADS, demonstrating the utility of the TADS framework for neural network explainability and verification. While TADS do not yet scale as efficiently as state-of-the-art neural network verifiers, we show that, using PCA-based simplifications, they can still scale to medium-sized problems and yield concise explanations for potential errors that can be used for other purposes such as debugging a network or generating new training samples.

Keywords (Rectifier) neural networks · (Piece-wise) affine functions · Decision trees · Explainability · Verification · Robustness · Principal component analysis · Diagnostics · Digit recognition · MNIST

1 Introduction

In recent years, neural networks have been a driving force behind many of the most exciting success stories in machine learning. From image recognition [24] and speech recognition [10] to playing complex games on a superhuman level [51], neural networks have achieved results that were almost unthinkable even a decade ago.

However, while the size, performance and scope of neural networks steadily increases, their opaqueness remains an equally important and essentially unsolved problem [2]. Frequently denoted as “blackbox”-models, the decisions of neural networks are to this day hard to explain and, likewise, their properties hard to verify.

In this paper, we are concerned with Typed Affine Decision Structures (TADS) [45], a novel decision-tree-like datastructure that represents piece-wise affine functions. TADS are specifically designed to act as interpretable white-box models that can precisely represent any piece-wise linear neural network in an understandable fashion.

While TADS are structurally well suited for global model explanation and verification of neural networks, the full explanation of even medium-sized neural networks is well out of scope. It is well known that the semantic complexity of a neural network, with respect to many different measures of complexity, grows exponentially in its size. As a consequence, any precise global explanation of such a model incurs exponential scaling issues [7, 18, 39].

In this paper, we are interested in applying TADS to verifying local properties of neural networks, most notably robustness properties [12, 35, 40, 49]. Robustness properties encode that, at certain points that a user desires, a neural network’s classification is invariant to small changes of its

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

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

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

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

1 Dortmund, Germany

input. For example, in image recognition, if one knows that an image represents a certain object, a single flip of a pixel should not drastically change the networks *correct* classification of said image. Robustness properties are the most commonly considered properties in neural network verification and make up the majority of current benchmarks in the VNNComp verification competition [6].

To apply TADS, which are in principle global model explanations, to local properties, we will introduce precondition projection, a transformation of TADS that restricts their domain to a certain region of interest. Further, we show how the algebraic properties of TADS can be used to directly model the classification behavior of a neural network. This is important as neural networks, although often used as classifiers (assigning one of finitely many classes to an input), are fundamentally regression models (assigning real values to their input). By modeling the argmax function directly on a TADS level, this gap can be bridged in an elegant fashion.

Finally, we will present a case study in which we apply TADS to robustness analysis and present its advantages. At present, TADS do not yet scale well to larger problems. We will introduce an approach that uses the well-understood dimensionality reduction technique PCA to prove an underapproximation to the robustness property of interest. This approach mitigates the scaling issues incurred by TADSs, but lacks reliable guarantees on robustness. Thus, we introduce another approach that directly trains neural networks to operate on inputs that are simplified by PCA. This method is of similar computational complexity than the underapproximation approach, but yields neural networks for which TADS can give reliable robustness guarantees, while incurring only a small loss in neural network accuracy.

Lastly, we will show on a concrete example how a TADS-based robustness proof looks like and what additional information it yields beyond already existing verification tools. We will show how this information can be used to characterize precisely and completely the entire set of inputs that violate a given property, and how it can be used to find "closest" adversarial examples, if they exist.

2 Preliminaries

2.1 Linear algebra and notation

The following notations of linear algebra are based on the book [5]. The real vector space $(\mathbb{R}^n, +, \cdot)$ with $n > 0$ is an algebraic structure with the operations

$$+ : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \qquad \text{vector addition}$$

$$\cdot : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n \qquad \text{scalar multiplication}$$

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 its i-th component, we write x_i (in contrast, $\vec{x_i}$ denotes the i-th *vector* in some enumeration). 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.

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

To indicate the number of rows and columns, one says $\boldsymbol{W}$ has *type* $n \times m$ commonly 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

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

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

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

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

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

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 $\vec{v}, \vec{w} \in \mathbb{R}^n$

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

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

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

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*.

2.2 Affine functions

Definition 1 (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 $b \in \mathbb{R}^m$. 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 $\Phi^{n\to m}$ with *type* (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

$$\circ\colon \Phi^{r\to m} \times \Phi^{n\to r} \to \Phi^{n\to m}.$$

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 [5]:

Theorem 1 (Algebraic Properties) *Denoting, as usual, scalar multiplication with $\cdot$ and function composition with $\circ$, we have:*

- $(\Phi^{n\to m}, +, \cdot)$ *is a vector space and*
- $(\Phi^{n\to n}, \circ, \mathrm{id})$ *is 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 2 (Properties of Typed Operations) $(\Phi, +_t, \cdot, \circ_t)$ *is a typed algebra, i.e., an algebraic structure that is closed under well-typed operations.*

2.3 Piece-wise affine functions

Piece-wise affine functions (PAFs) are studied extensively in tropical geometry (introductory book [36], in context of machine learning [38]), are used in interpolation (given their strong connection to splines and Riemann integrals), and are more and more analyzed with respect to their connection to neural networks [4, 13, 19, 28–30, 39, 43, 44, 46, 48, 55–57] (specifically to PLNNs, see Definition 7). PAFs are usually defined over a polyhedral partitioning of the pre-image space [9, 22, 41]. Polyhedra arise by intersecting halfspaces:

Definition 2 (Hyperplanes and Halfspace) Let $\vec{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Then the set

$$p = \{\, \vec{x} \in \mathbb{R}^n \mid \langle \vec{w}, \vec{x} \rangle + b = 0 \,\}$$

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, respectively, are defined as

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

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

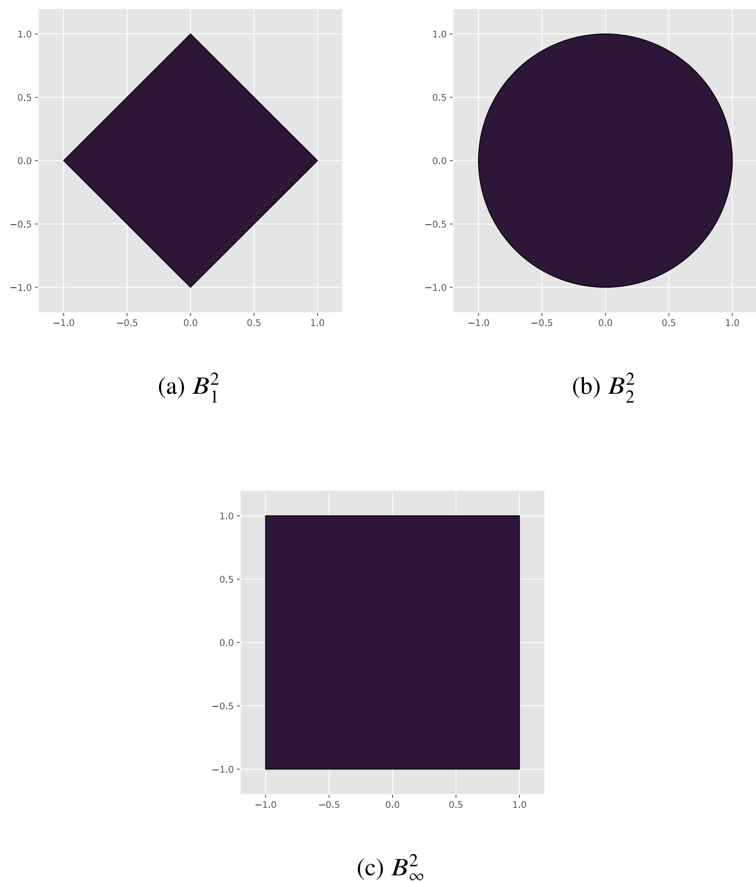
Definition 3 (Polyhedron) A polyhedron $Q \subseteq \mathbb{R}^n$ is the intersection of k halfspaces for some natural number k.

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

Definition 4 (Piece-wise Affine Function) A function $\psi :$ $\mathbb{R}^n \to \mathbb{R}^m$ 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 $\mathbb{R}^n$ and $\alpha_1, \ldots, \alpha_k$ are affine functions. We call $\alpha_i = \boldsymbol{W}_i\vec{x} + \vec{b}_i$ with $1 \le i \le k$ the function associated with polyhedron Q_i.



(a) B_1^2 (b) B_2^2 (c) B_∞^2

Fig. 1 The two-dimensional m-balls B_m^2 for $m = 1$ (top left), $m = 2$ (top right) and $m = \infty$ (bottom)

2.3.1 Norms and distances

Throughout this work, we will often be concerned with the behavior of neural networks and how it changes when a point is slightly altered. Thus, we will often be concerned with different *neighborhoods* of points. These are formalized in mathematics using metric spaces and normed spaces [37]. For our purposes, however, a special type of normed spaces defined by so-called l-norms is sufficient:

Definition 5 (L-Norms) For $m \in \mathbb{N}$, the l_m-norm is the function $\|\cdot\|_m : \mathbb{R}^n \to \mathbb{R}$ defined by:

$$\|\vec{x}\|_m := \sqrt[m]{\sum_{i=1}^{n} |x_i|^m}$$

Important l-norms are the l_1 norm

$$\|\vec{x}\|_1 := \sum_{i=1}^{n} |x_i|$$

and the l_2 norm or euclidean norm[1]

$$\|\vec{x}\|_2{}^2 := \sqrt{\sum_{i=1}^{n} x_i^2} \, .$$

Another important norm is the so-called l_∞ norm. While not technically an l-norm according to the definition above, it arises naturally as the limit of the l-norms as m approaches infinity and is defined by:

$$\|\vec{x}\|_\infty := \max_{i \in \{1,\ldots,n\}} |x_i|$$

With these definitions we can now formalize the neighborhood of a point.

Definition 6 (Unit Ball) For a given l_m-norm $\|\cdot\|_m$ we define the corresponding m-unit ball as

$$B_m^n := \{ \vec{x} \in \mathbb{R}^n \mid \|\vec{x}\|_m \leq 1 \} \tag{1}$$

The unit ball is a closed, convex subset of $\mathbb{R}^n$ centered at the origin. It generalizes the notion of a disk with radius 1 to both higher dimensions ($n > 2$) and non-euclidean spaces ($m \neq 2$). The relevant unit balls for this paper are illustrated in Fig. 1. Every generic m-ball of $\mathbb{R}^n$ can be constructed using translation

$$\vec{y} + B_m^n = \{ \vec{x} \in \mathbb{R}^n \mid \|\vec{x} - \vec{y}\|_m \leq 1 \}$$

[1]For real vector spaces $\mathbb{R}^n$ the euclidean norm is the canonical norm as $\|\vec{x}\|_2 = \langle \vec{x}, \vec{x} \rangle$.

and scaling

$$r B_m^n = \{ \vec{x} \in \mathbb{R}^n \mid \|\vec{x}\|_m \leq r \}$$

of unit m-balls. In the latter case r is called the radius. If it is clear from the context, we omit the dimensionality of the m-ball.

2.4 Neural networks

The following brief introduction to neural networks is based on [20], but in its presentation adapted to better fit the context of this work.

Neural networks are perhaps todays most important machine learning models that are most succinctly characterized by their layered structure. There exist numerous neural network architectures that one might consider. For this work, we focus on the very general class of fully connected neural networks and define neural networks as follows:

Definition 7 (Piece-wise Linear Neural Networks) A piece-wise linear neural network ν with l layers is a machine learning model consisting of an alternating sequence of l *affine preactivation functions* α_i and $l-1$ *ReLU activation functions* ϕ:

$$\nu = \alpha_{l+1} \,;\, \phi \,;\, \alpha_l \,;\, \cdots \,;\, \phi \,;\, \alpha_1$$

For the PLNN to be syntactically correct, the affine functions must be compatible, i.e., the output dimension of each preactivation must match the input dimension of the following.

In accordance to standard neural network terminology, we call the combination of a preactivation with its activation $\phi \circ \alpha_i$ the i-th layer of the neural network.

In the following paragraphs preactivations and activations are properly introduced. After that, the semantics of a PLNN can be defined in terms of its components. Lastly, a common complexity measure of PLNN is presented.

Preactivations In traditional applications, the concrete affine functions α_k of a PLNN ν, as defined in Definition 7, would result from a training process, usually using gradient descent based optimization techniques [20], where the PLNN is trained to accurately predict desired outputs on a given dataset.

Activations The activation function ϕ is an architectural design choice made a-priori by the user. The primary purpose of the activation function is to introduce *non-linearity* into the neural network, which can drastically increase the amount of functions that can be approximated. For the purposes of this paper, we exclusively use the rectified linear unit (ReLU) function.

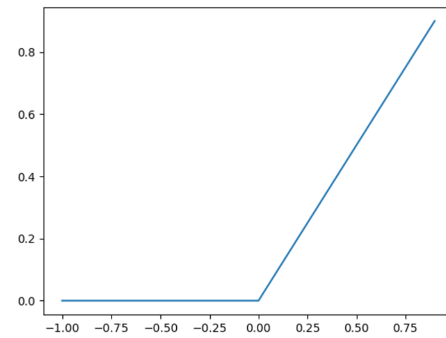


Fig. 2 The function plot of the ReLU function

ReLU The ReLU function ϕ (c.f., Fig. 2) has proven to be a successful activation function in practical applications, combining convenient properties of linear functions with a sufficient degree of non-linearity. It is prominently recommended as the default choice of activation function for fully connected neural networks [20]. Furthermore, due to the simple structure of the ReLU function, neural networks with ReLU activations lend themselves well to formal analysis and are typically considered in verification tasks [6, 31].

Definition 8 (Rectified Linear Unit) The one-dimensional ReLU function $\phi_1 : \mathbb{R} \to \mathbb{R}$ is defined as the positive part of its argument:

$$\phi_1(x) = \max(0, x)$$

The k-dimensional ReLU function $\phi_k : \mathbb{R}^k \to \mathbb{R}^k$ is the elementwise application of the one-dimensional ReLU function:

$$\phi_k\big((x_1, x_2, \ldots, x_k)^\top\big) = \big(\phi_1(x_1), \phi_1(x_2), \ldots, \phi_1(x_k)\big)^\top$$

As the ReLU activation function is the only activation function we consider, we will use ϕ for the remainder of this paper exclusively to refer to the ReLU function and omit the explicit mention of the dimensionality when it is clear from context.

Definition 9 (PLNN Semantics) The semantics of a piece-wise linear neural network ν is a piece-wise affine function $[\![\nu]\!]_{\mathcal{N}} : \mathbb{R}^n \to \mathbb{R}^m$ given by the sequential evaluation of its layers:

$$[\![\nu]\!]_{\mathcal{N}} = \alpha_{l+1} \circ \phi \circ \alpha_l \circ \cdots \circ \phi \circ \alpha_1$$

For evaluation, a vector $\vec{x} \in \mathbb{R}^n$ is passed layer by layer through the PLNN. The data-flow is unidirectional and using the above notation from right to left.

Note that, given the close relationship between a PLNN's syntax and semantics, many works in deep learning choose to not clearly separate the syntax and semantics of PLNN's.

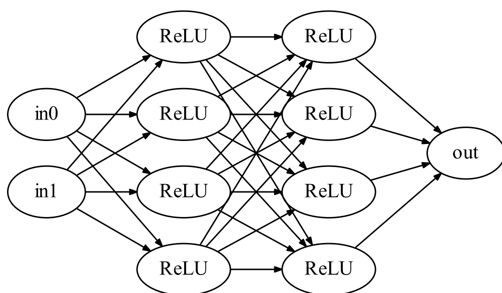


Fig. 3 A simple PLNN with two hidden layers and ReLU activations

A transition between the two definitions can be easily achieved by replacing ';' with '∘' in Definition 7.

Traditionally, neural networks are visualized as computation graphs, where the nodes are the eponymous "neurons". There, each affine function $\alpha_i : \mathbb{R}^n \to \mathbb{R}^m$ is visualized as a bipartite graph connecting n input neurons to m output neurons. An example of such graph is given in Fig. 3.

From the representation used in Definition 7, the number of neurons of a neural network can be computed through the preactivations as follows: Let $\nu = \alpha_{l+1} \,;\, \cdots \,;\, \alpha_1$ with $\alpha_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$ then the total number of neurons of ν is given by

$$\sum_{i=1}^{l+1} n_{i+1}$$

The number of neurons is a natural measure of "size" in a neural network, and it is well known that the semantic complexity of functions – measured in the number of linear regions that are needed to characterize them – that a neural network can represent increases exponentially in its number of neurons [7, 18, 39].

2.4.1 Neural networks classifiers

As defined in Definition 7, PLNNs are fundamentally representations of continuous functions $\mathbb{R}^n \to \mathbb{R}^m$. However, they are frequently employed in classification tasks where the co-domain is instead a discrete set of classes $\{1, \ldots, c\}$. To bridge this gap, one typically proceeds by training a neural network $\nu : \mathbb{R}^n \to \mathbb{R}^c$ and associating each component y_i of its output $\vec{y} = \nu(\vec{x})$ with the i-th class. Then, the class with the largest y_i is chosen for classification.

This is formalized by the argmax function.

Definition 10 (Argmax) The k-dimensional argmax function

$$\arg\max_k : \mathbb{R}^k \to \{1, \ldots, k\}$$

is defined as

$$\arg\max(x_1, \ldots, x_k) = j$$

iff j is the smallest index for which $x_j \geq x_i$ holds for all $1 \leq i \leq k$.

Again, when it is clear from context, we omit the index denoting the dimensionality and simply write arg max.

As described before, the argmax function can be used to convert PLNNs into classifiers. This naturally leads us to define PLNN classifiers.

Definition 11 (PLNN Classifiers) For a PLNN ν with $[\![\nu]\!] : \mathbb{R}^n \to \mathbb{R}^m$, the corresponding *PLNN classifier* $\nu_c : \mathbb{R}^n \to \{1, \ldots, m\}$ is defined as

$$\nu_c = \arg\max \circ [\![\nu]\!]_{\mathcal{N}}$$

2.5 Typed affine decision structures

Central to our explanation approach is a decision-tree-like data structure that we call *Typed Affine Decision Structure* (TADS). Based on the transformation process presented in [45], it is possible to transform a PLNN ν into a *semantically equivalent* TADS $\theta(\nu)$. The transformation is based on the common syntactical representation of PLNNs and is compositional in the layers.

PLNNs explanation and verification is challenging because of the complex data flow of PLNNs [45].

Data structure Skipping implementation details, TADS can be introduced intuitively using decision trees. In a decision tree, one distinguishes two types of nodes:

1. Inner nodes have decision predicates. For every possible evaluation of that predicate, the node has exactly one successor.
2. Leaves are elements from a given universe that one wants to distinguish.

For TADS, specifically, leaves are from the universe of affine functions and decision predicates are affine inequalities.[2] An example of a TADS can be found in Fig. 4. TADS structurally resemble decision trees that are widely considered explainable machine learning models, i.e., they can, by virtue of their structure, be understood by a human [26].

Based on this introduction using decision trees one can straightforwardly define TADS.

Definition 12 (TADS) A TADS $t = (N, \to, \zeta)$ is a decision DAG[3] $(N, \to, \zeta)$ with root ζ whose nodes N have the following two types:

[2] Also called inhomogeneous inequalities.

[3] A decision DAG is a decision tree where isomorphic subtrees have been merged, see [23].

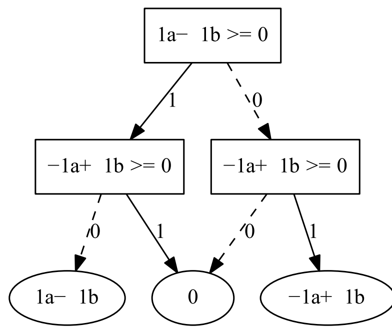


Fig. 4 A simple TADS, implementing the piece-wise affine function $|x_1 - x_2|$

1. Inner nodes are called decisions or predicates. They consist of an affine inequality and two successors, one if the predicate is true and one if not.
2. Leaves are also called terminals. They are affine functions and have no successors.

To be syntactically correct, all nodes (i.e., all inequalities and affine functions) must accept input vectors with a fixed number of entries. This is called the *input dimension* of the TADS. Similarly, all terminals must map input vectors into a common output space. The dimensionality of this output space is called the *output dimension*.[4] For given input dimension n and output dimension m we define the set of all TADS as $\Theta^{n\to m}$.

TADS are sequentially evaluated like a decision tree.

Definition 13 (TADS Evaluation) The semantic function of TADS[5]

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

is inductively defined as

$$[\![p]\!]_\Theta(\vec{x}) := [\![p']\!]_\Theta(\vec{x}) \qquad \text{if } p \xrightarrow{l} p' \wedge [\![p]\!](\vec{x}) = l$$
$$[\![\alpha]\!]_\Theta(\vec{x}) := \alpha(\vec{x}) \qquad \text{if } \alpha \not\to$$

for a TADS $t = (N, \to, \zeta)$, with $p, p', \alpha \in \nu$. For convenience we introduce the shorthand $[\![t]\!]_\Theta := [\![\zeta]\!]_\Theta$.

Semantically, both PLNNs and TADS represent piece-wise affine functions. Moreover, PLNNs can be transformed into TADS:

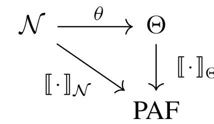
[4]Note that the input and output vectors always result from the real vector spaces $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. Therefore, the number of entries is well-defined an equal to the dimensionality of the full vector space of inputs and outputs.

[5]In this definition we use *currying,* as known in, e.g., functional programming.

Lemma 2 (Trinity: PLNNs, TADS, and PAFs) *There exists a semantics preserving transformation*

$$\theta \colon \mathcal{N} \to \Theta$$

from PLNNs to TADS, such that the following diagram commutes:



TADS are computationally transparent and semantically equivalent to PLNNs.

Algebraic properties Much like ADDs and BDDs, TADS inherit the algebraic properties of their leaf algebra. For TADS, the leave algebra—affine functions—forms a vector space. Using *lifting* one can directly implement the vector space operations on TADS [45].

Lemma 3 (Lifting) *Lifting addition* $(+)$ *and scalar multiplication* $(\cdot)$ *from affine functions to TADS gives semantically equivalent operators to their PAF counterparts, i.e., for all TADS* $t_1, t_2 \in \Theta$

$$[\![t_1 + t_2]\!]_\Theta = [\![t_1]\!]_\Theta + [\![t_2]\!]_\Theta$$
$$[\![s \cdot t]\!]_\Theta = s \cdot [\![t]\!]_\Theta$$

By the lifting theorem of [45] the algebraic properties are preserved and thus:

Theorem 3 (TADS vector space) *TADS* $(\Theta, +, \cdot)$ *form a vector space.*

It is well known that piece-wise affine functions are closed under composition. Even though this operator can not be directly lifted, it can be easily implemented on TADS [45].

Theorem 4 (TADS Composition) *TADS composition*

$$\bowtie \colon \Theta^{n\to m} \times \Theta^{m\to r} \to \Theta^{n\to r}$$

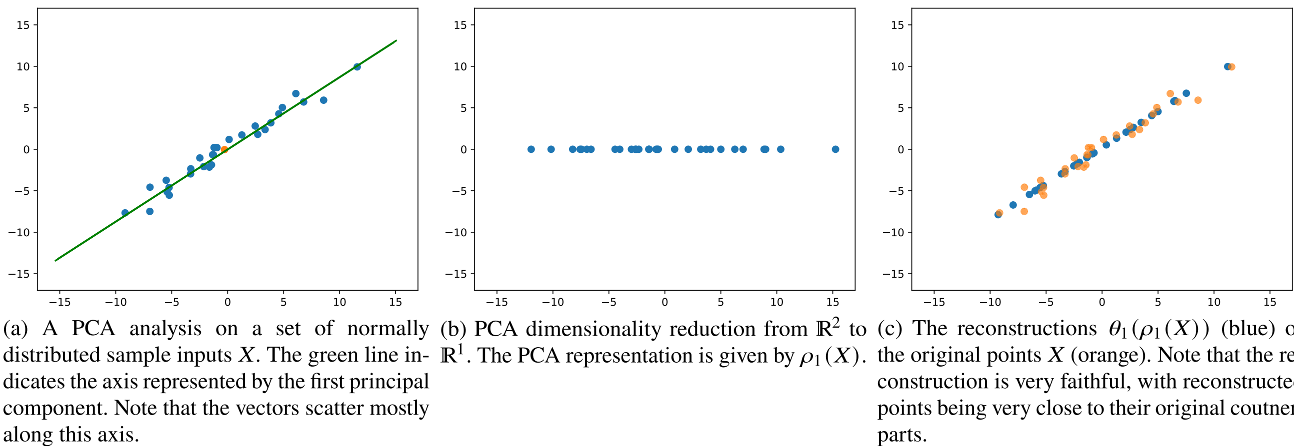
is defined such that for all $t_1 \in \Theta^{n\to m}$, $t_2 \in \Theta^{m\to r}$:

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

It follows straightforwardly that:

Theorem 5 (TADS Monoid) *TADS* $(\Theta, \bowtie)$ *forms a monoid.*

The composition operator $\bowtie$ is especially important in the context of neural networks, as neural networks are inherently compositions of piece-wise affine functions.



(a) A PCA analysis on a set of normally distributed sample inputs X. The green line indicates the axis represented by the first principal component. Note that the vectors scatter mostly along this axis.

(b) PCA dimensionality reduction from $\mathbb{R}^2$ to $\mathbb{R}^1$. The PCA representation is given by $\rho_1(X)$.

(c) The reconstructions $\theta_1(\rho_1(X))$ (blue) of the original points X (orange). Note that the reconstruction is very faithful, with reconstructed points being very close to their original coutnerparts.

Fig. 5 Example for PCA dimensionality reduction. A set of points X (shown in blue) follows a multinormal distribution that scatters more along one axis. This axis is very closely resembled by the first principal component (shown in green). Through orthogonal projection one can reduce the dimensionality (b), and the reconstruction (c) is very close to the original (Color figure online)

2.6 Principal component analysis

Principal Component Analysis (PCA) is one of the most popular techniques for dimensionality reduction and feature extraction [1, 8, 54]. At a high level, it seeks to find, for a given dataset $D \subset \mathbb{R}^n$, a linear subspace $V \subset \mathbb{R}^n$ with dimension $\dim(V) \ll n$ that can be used to encode D with as little reconstruction loss as possible.

Such an encoding is useful for machine learning algorithms as it can drastically reduce the input dimension. Large input dimensions can be very problematic in machine learning and entail numerous potential problems, altogether known as the *curse of dimensionality* [50].

The fundamental objects of PCA are the eponymous principal components that are defined as follows:

Definition 14 (Principal Components) For a given dataset $D = \{\vec{d}_1, \ldots, \vec{d}_j\} \subset \mathbb{R}^n$ with $j \geq n$ and zero mean $\sum_{\vec{d} \in D} d = \vec{0}$, there exist n *principal components* $\vec{p}_1, \ldots, \vec{p}_n \in \mathbb{R}^n$ which are characterized as iterative solutions to the following optimization problem: The i-th principal component $\vec{p}_i$ maximizes the variance of the data when it is projected onto $\vec{p}_i$:

$$\sum_{d \in D} \langle \vec{p}_i, \vec{d} \rangle^2 \rightarrow \max$$

under the constraint that p_i has unit length

$$\|\vec{p}_i\|_2 = 1$$

and is orthogonal to all previous principal components

$$\forall h < i : \langle \vec{p}_i, \vec{p}_h \rangle = 0$$

Note that every dataset D with non-zero mean, i.e., $\sum_{\vec{d} \in D} \frac{\vec{d}}{|D|} = \vec{\mu}$ with $\vec{\mu} \neq 0$, can be made to obey the restriction $\vec{\mu} = \vec{0}$ by performing the following transformation on each datapoint: $\vec{d}_i' = \vec{d}_i - \vec{\mu}$. By definition, the principal components are pair-wise orthogonal and normed and therefore linearly independent. Thus, they form a basis of $\mathbb{R}^n$. It follows that there is a natural, unique representation based on the principal components $\rho(\vec{x}) = (r_1, \ldots, r_k)^\top$ such that:

$$\vec{x} = \sum_{i=1}^{n} r_i \vec{p}_i$$

In particular, in the case of PCA, the r_i can be computed as

$$r_i = \langle \vec{p}_i, \vec{x} \rangle \ .$$

With this, PCA can naturally be used as a dimensionality reduction tool.

Definition 15 (PCA Dimensionality Reduction) Let $0 < k < n$. For some $\vec{x} \in \mathbb{R}^n$ with $\rho(\vec{x}) = (r_1, \ldots, r_n)$, the k-*dimensional PCA representation* is given by cutting off the PCA representation after the k-th element:

$$\rho_k(\vec{x}) = (r_1, \ldots, r_k)^\top$$

Consequently, the k-*dimensional PCA reconstruction* to $\vec{x}$ is given as:

$$\vec{x} \approx \theta_k(\rho_k(\vec{x})) = \sum_{i=1}^{k} r_i \vec{p}_i$$

As ρ_k is a projection for $k < n$, it loses information. Therefore, the PCA reconstruction after dimensionality reduction is approximative, as visualized in Fig. 6.

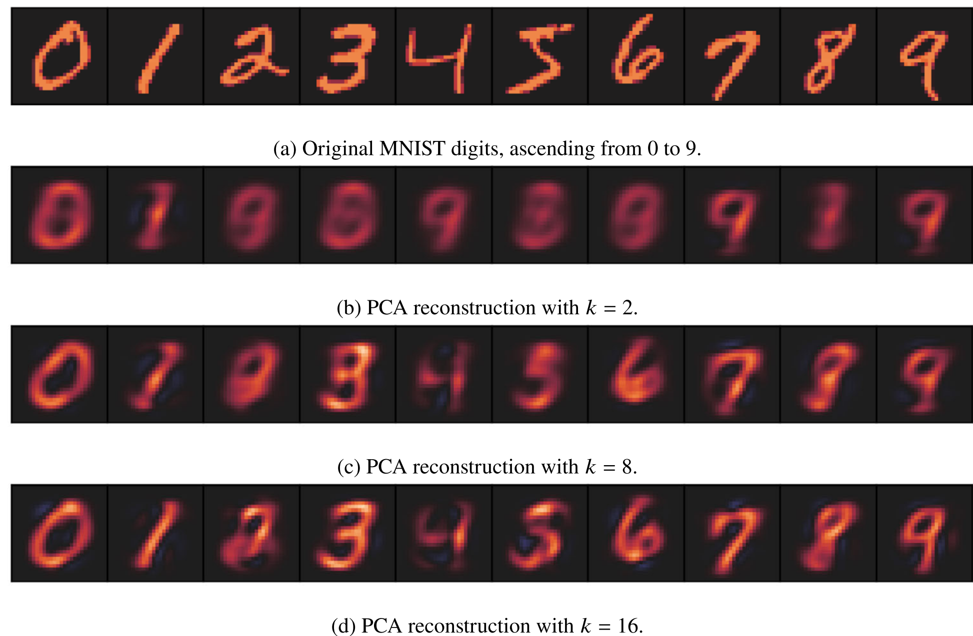


Fig. 6 Reconstruction of 784 pixel MNIST digits with various number of principal components, showcasing how a PCA reconstruction can faithfully reconstruct an original input based on very little information ($k = 16$ principal components vs. 784 pixel). Original MNIST digits in first row, following rows show reconstruction with $j = 2, 8, 16$ principal components. Vectors are visualized using a perceptually uniform diverging color palette (Seaborn's "icefire")

In essence, the composition of PCA encoding and reconstruction forms a function that is close to the identity function *on the dataset and its surrounding points* while reducing the number of dimensions needed to express the data. The success of PCA is heavily dependent on the dataset being mainly distributed along a linear subspace of $\mathbb{R}^n$ and its generalization performance requires that new data follow the same distribution as the training data. However, if these assumptions hold, it is a very good approximation, as indicated by the following defining property of the principal components:

Lemma 4 *The principal components are exactly those vectors that make the reconstruction error minimal over D among all linear, orthogonal encoders and decoders using k dimensions* [1]. *I.e., for all orthogonal, linear functions* $e : \mathbb{R}^n \to \mathbb{R}^k$, $d : \mathbb{R}^k \to \mathbb{R}^n$, *the term*

$$\sum_{\vec{x} \in D} \|\vec{x} - d(e(\vec{x}))\|_2^2$$

is minimal if $e = \rho_k$, $d = \theta_k$.

PCA is attractive for multiple reasons. First, PCA representations and approximations are linear functions which makes them easy to work with. Second, PCA supports reductions to k for any $0 < k \leq n$, which makes PCA very flexible. Lastly, but perhaps most importantly, PCA is a well-understood and well-proven method in practice and can elegantly enable strong performance in even relatively simple machine learning models. An example of a PCA encoding and reconstruction is shown in Figs. 5a to 5c.

PCA allows the compression of high-dimensional inputs into lower-dimensional representations such that a given dataset is compressed with as little information loss as is possible using orthogonal, linear compression.

3 Problem setting: robustness on MNIST

3.1 Introduction to MNIST

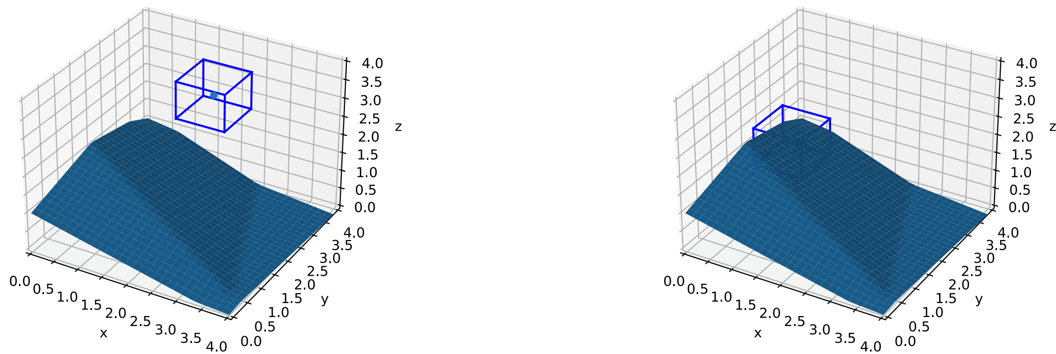
In the remainder of the paper we consider the problem of digit recognition using the MNIST dataset [16]. The MNIST dataset provides a traditional baseline-problem scenario for machine learning. While simpler than modern, large-scale machine learning tasks, MNIST requires PLNNs of relevant size for satisfactory classification and stands to this day as an introductory problem in verification benchmarks [6].

The MNIST dataset consists of 70.000 gray-scale images of hand-written digits, each labeled with the digit they represent to a human observer. The dataset is split into 60.000 examples for training and 10.000 examples for testing. Images consist of 28×28 pixels and are represented as vectors $\vec{x} \in \mathbb{R}^{28 \cdot 28}$ with each component $\vec{x}_i$ representing the gray-scale value of the i-th pixel on a scale from 0 to 1. Thus, each sample has the form

$$(\vec{x}, l)$$

with $\vec{x} \in [0, 1]^{28 \cdot 28}$ and $l \in \{0, \ldots, 9\}$. The task is to find a PLNN classifier that represents a function

$$\nu_c : \mathbb{R}^{28*28} \to \{0, \ldots, 9\}$$



(a) We consider robustness around the point $\vec{x} = (2, 3, 3)^\top$. The neighborhood defined by the ∞-ball (blue cube) lies on the same side of the classification boundary, thus the point is robust.

(b) Using the same PLNN, robustness is considered for a point close to the decision boundary. The ∞-ball intersects the boundary, thus the network is not robust for this point.

Fig. 7 Illustration of two robustness scenarios using its geometric interpretation. In (a) robustness is achieved while in (b) robustness is violated as the ∞-ball around $\vec{x}$ intersects with the decision boundary (Color figure online)

assigning to each image the digit it is supposed to represent. At a baseline, ν_c should classify most training examples correctly and should perform acceptably well on the test data.

A challenge for *classification problems* like this is to control so-called *adversarial examples,* as discussed in the following.

3.2 Robustness to adversarial examples

In essence, robustness is the absence of adversarial examples, which are perhaps the most well-known manifestations of chaotic behavior of neural networks and have received wide attention in research [21, 33, 49]. We work with the following definition of adversarial examples:

Definition 16 (Adversarial Example) Let $\nu_c : \mathbb{R}^n \to \{1, \ldots, m\}$ be a PLNN classifier. Further, let $\vec{x} \in \mathbb{R}^n$ be a given point of interest that is correctly classified by ν_c. Then, $\vec{y} \in \mathbb{R}^n$ is an ϵ-adversarial example to $\vec{x}$ iff

$$\|\vec{y} - \vec{x}\|_\infty \leq \epsilon \;\wedge\; \nu_c(\vec{y}) \neq \nu_c(\vec{x})$$

If ν_c admits no ϵ-adversarial examples for a given input $\vec{x}$, then it is called ϵ*-robust* around $\vec{x}$.

Intuitively, an adversarial example is a slight perturbation of an input that, although minor, changes the neural networks prediction. Note that in image recognition problems such as MNIST, the restriction $\|\vec{y} - \vec{x}\|_\infty \leq \epsilon$ encodes that between $\vec{x}$ and $\vec{y}$, each pixel can only differ by at most ϵ.

In practice, adversarial examples can be almost imperceptible to a human [35, 49] while arbitrarily altering previously correct decisions, sometimes yielding outlandish classification results, which may enable outside attacks on neural network systems. Thus, it is critical that neural networks cannot be adversarially attacked at points where the desired semantics is clear.[6]

3.3 Verifying robustness

Generally, PLNN verification is the task proving a property for the result of a PLNN where the input is restricted to a given domain [3, 11]. Formally, let $\nu_c : \mathbb{R}^n \to \mathbb{R}^m$ be a PLNN, $S \subset \mathbb{R}^n$ a restriction of the input domain, and $P : \mathbb{R}^m \to \{0, 1\}$ a predicate. Then PLNN verification is the task of proving or refuting with a counterexample the formula

$$\forall \vec{x} \in S : P(\nu_c(\vec{x})) \; . \tag{2}$$

For the case of verifying ϵ-robustness around $\vec{x}$ for ν_c, we can formulate (2) specifically as (cf. Definition 16)

$$\forall \vec{y} \in \vec{x} + \epsilon B_\infty : \nu_c(\vec{y}) = \nu_c(\vec{x}) \; .$$

Corresponding state-of-the-art verification tools use different methods like [11]:

- Satisfiability Modulo Theories
- Mixed Integer Programming
- Branch and Bound

For more information, see Sect. 7 on related work.

[6] Of course, neural networks necessarily have regions where the prediction flips from one class to another. Ideally, these flips should only occur in regions where inputs are non-sensical and would not intuitively be assigned to any class by a human. Therefore, robustness is usually considered only at some select sample inputs where semantics is clear.

4 Extending TADS to cover robustness properties

TADS are characterized by:

1. *Global explanations*, i.e., they explain the behavior of a PLNN over the entire space of possible inputs. Robustness properties however concern only the relatively small neighborhood $x + \epsilon B_\infty$ of a point $\vec{x}$.
2. *Regression behavior*, they represent a continuous function. With respect to robustness, we are however interested in the behavior of the associated *PLNN classifier*.

The following two subsections will show that TADS are nevertheless well suited to deal with robustness properties.

4.1 Precondition projection on TADS

When studying adversarial examples, one may use the strict preconditions (given as infinity balls ϵB_∞) to reduce the work load. Given the strong connection between affine functions and (convex) polytopes, it is a straightforward procedure to apply polyhedral preconditions—such as infinity balls as particularly required for robustness properties—on TADS. Please note that stronger preconditions result in less work.

Given a TADS t representing a piece-wise affine function $f : \mathbb{R}^n \to \mathbb{R}^m$ we are interested in the behavior of f on a given (small) polyhedron $S \subset \mathbb{R}^n$. In other words, we are interested in the function $f\big|_S : S \to \mathbb{R}^m$ which is given by:

$$f\big|_S(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if } x \in S \\ \bot & \text{otherwise} \end{cases}$$

Technically, this is implemented by encoding the polytope S as a TADS using affine inequalities:

$$t\big|_S := \left(\text{id}\big|_S\right) \bowtie t$$

By explicitly eliminating paths that lead to $\bot$ (see [45]), the resulting TADS is significantly reduced in size.

4.2 Argmax on TADS for classification

Neural networks are frequently used for classification, as outlined in Sect. 2.4.1. As described there, the neural network classifier associated with a given neural network ν can naturally be modeled as

$$\nu_c = \arg\max \circ [\![\nu]\!]$$

Interpreting a neural networks behavior in this way drastically changes its nature, and if one seeks to analyze a neural network that is meant to be used as a neural network classifier, it is important that one analyzes it with respect to its classification behavior.

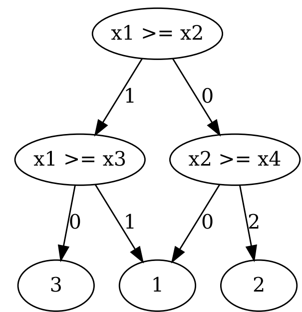


Fig. 8 The TADS t_a, representing the argmax function with 3 variables

We know how to construct a t_ν for any PLNN ν. On the other hand, it is also easy to see how a TADS t_a can be constructed for arg max: Intuitively, such a TADS need only to perform a linear search for the maximum of $\vec{x} = (x_1, \ldots, x_n)$ from x_1 to x_n. Figure 8 illustrates this for the three-dimensional argmax in [7] This TADS first compares x_1 and x_2 in the first layer, then compares their maximum with x_3 to attain the result. The extension to higher dimensions is straightforward.

Taken together, it is straightforward to construct the classification TADS t_{ν_c} using TADS composition as follows:

$$t_{\nu_c} = t_\nu \bowtie t_a$$

The semantical correctness of this construction follows directly from the correctness of the TADS composition, i.e.:

$$[\![t_{\nu_c}]\!]_\Theta = \arg\max \circ [\![\nu]\!]$$

5 Verifying robustness on MNIST using TADS

The following subsections of this Section present four approaches to robustness verification via TADS and illustrate them using the MNIST data set:

– A straightforward approach where the considered PLNN is directly transformed into a TADS (cf. Fig. 9a). This approach typically does not scale due to the typical exponential explosion of the TADS transformation.
– An approximative approach based on PCA-based dimensionality reduction that scales, provides a good heuristics to search for adversarial examples, but is insufficient to prove robustness (cf. Fig. 9b). In this case, the TADS-based analysis only covers the subspace that can be 'reached' from the initial, low-dimensional PCA-based vectors space via decoding and adequate basis transformation, as indicated by the blue part. Thus, this approach

[7]Note that this TADS deviates slightly from the representation of TADS we use for the rest of this paper, notably with respect to the way linear inequalities are represented. This is purely done to enhance readability.

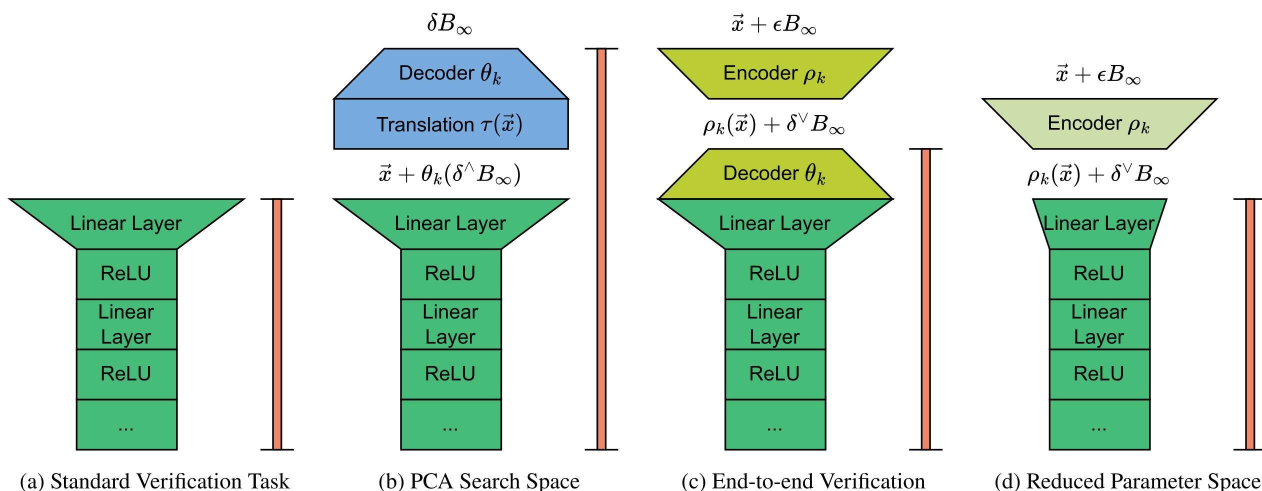


Fig. 9 Overview of the different approaches to verifying robustness with PCA encoding. Legend: (green) entity for usage in the real world, (blue) components only used during verification, (orange sidebar) components used for TADS construction, (mint green) components used during training and actively trained, (pale green) parts included in training but not actively learned, (olive green) parts that are not included in the training process (Color figure online)

cannot guarantee that the analysis of the TADS is sufficient to reveal all adversarial examples of the original PLNN.

– A transformational approach based on PCA-based dimensionality reduction, where the PLNN is extended by a preprocessing step, defined by PCA-based auto-encoding, i.e., the composition of a PCA-based dimensionality reduction followed by a linear function that embeds (decodes) the low-dimensional space into the original space (cf., Fig. 9c). Here we can show that analyses of the partial extension that start with the decoding are sufficient to obtain robustness results for the extended PLNN that is defined for the 784-dimensional space of MNIST.
– A modification of the third approach, where the linear function defined by the composition of the decoder and the initialization layer of the original net is replaced by a linear layer to provide a network architecture with the same number of layers but with a strongly reduced input dimension (cf. Fig. 9d). The PLNN considered for verification is now given as the result of a learning process using the same sample set as in the other cases, but starting with a PCA-based reduction step. Technically, the subsequent TADS-based robustness analysis proceeds exactly in the same way as before guaranteeing that the robustness result proven for the dark green part can again be lifted to the overall net.

We will show that the third and fourth approaches allow us to prove full robustness in a computationally efficient manner. However, they come at the price of modifying the PLNN.

In our eyes, this is no disadvantage as long as the modified PLNN is still sufficiently accurate; Neural networks are themselves only results of a heuristic training process and have no intrinsic merit beyond their predictive accuracy. In fact, the results shown in Fig. 12 indicate that predictive accuracy can still be achieved after a significant reduction in dimensionality, drastically easing formal verification.

5.1 Full verification with TADS

At their baseline, TADS are so called *model explanations* [25] of PLNNs, i.e., for any classification PLNN $\nu_c\colon \mathbb{R}^n \to \mathcal{C}$, a corresponding TADS can be generated that represents the same function as ν_c in an easily comprehensible and analyzable manner. Of course, the global behavior of neural networks is usually too large to be represented with a TADS. However, in the case of robustness verification, we are only interested in the behavior of ν_c in the neighborhood around some point of interest $\vec{x}$, formalized by an infinity ball (see Definition 6). Recall from Definition 16 that ϵ-robustness for a point $\vec{x}$ is formalized by the property

$$\forall \vec{y} : \|\vec{y} - \vec{x}\|_\infty \leq \epsilon \implies \nu_c(\vec{x}) = \nu_c(\vec{y})$$

Equivalently, this problem can also be stated as

$$|\nu_c(\vec{x} + \epsilon B^n_\infty)| = 1$$

that is, the neighborhood of $\vec{x}$ defined by the infinity ball ϵB_∞ of dimension n with radius ϵ is classified consistently as one class. This property can be verified using the following theorem:

Theorem 6 (TADS Verification) *Let ν_c be a PLNN classifier, $\vec{x}$ a point of interest, and t a TADS satisfying $[\![t]\!]_\Theta = [\![\nu_c]\!]_\mathcal{N}$.*

Then ν_c is ϵ-robust around $\vec{x}$ iff

$$t\big|_{\epsilon B^n_\infty}$$

contains only feasible paths to the class $\nu_c(\vec{x})$.

The correctness of this theorem follows directly from the correctness results regarding TADS that were established in Sect. 4.

The approach to directly verify the original network sketched at the very left of Fig. 9 only works for quite small MNIST networks. Core reason for this scaling problem is the dimensionality of MNIST: With 784-dimensional inputs, the volume of the ϵ-ball around $\vec{x}$ is proportional to ϵ^{784}, which grows quite quickly leading to intractably large TADS.

> The complexity of robustness verification increases exponentially with the number of input dimensions. Reducing dimensionality is therefore key to mitigating scaling issues and proving robustness for larger ϵ.

5.2 PCA guided validation

To improve scalability of TADS-based verification, one might consider approximative robustness instead. More concretely, instead of searching for adversarial examples in the full ball $\epsilon B_\infty(\vec{x})$, we will present an approach that restricts the search to a lower dimensional subset $S \subset \epsilon B_\infty(\vec{x})$.

This will yield an underapproximation to robustness: If an adversarial example is found in S, it also exists in $B_\infty(\vec{x})$ and robustness is violated. However, the absence of adversarial examples in S does not imply the absence of adversarial examples in $B_\infty(\vec{x})$. Key for the construction of the lower-dimensional manifold S is *principal component analysis* (PCA) as introduced in Sect. 2.6.

Applying PCA to MNIST results in a list of $n = 784$ principal components

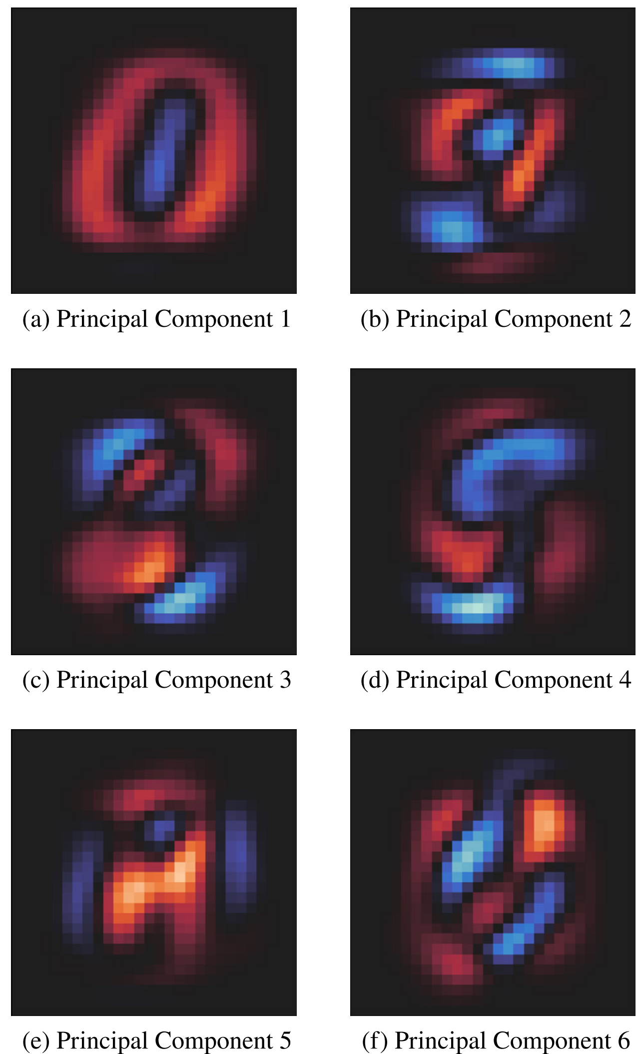
$$\vec{p}_1, \ldots, \vec{p}_n$$

ordered by decreasing variance along the respective axis.

The first six of which are visualized in Fig. 10. Recall from Definition 15, that the principal components of $\vec{p}_i$ are precisely those along which a given dataset scatters most. They are therefore natural candidates to explore in a heuristic search for adversarial examples.

Let $\vec{x}$ be some point for which we seek to find adversarial examples. Then, we can define the k-dimensional PCA space around $\vec{x}$ as follows:

$$U_k(\vec{x}) = \left\{ \vec{x} + \sum_{i=1}^{k} w_i \vec{p}_i \;\middle|\; w_i \in \mathbb{R} \right\}$$



(a) Principal Component 1 (b) Principal Component 2 (c) Principal Component 3 (d) Principal Component 4 (e) Principal Component 5 (f) Principal Component 6

Fig. 10 The first 6 principal components of the MNIST dataset. Note that in the context of MNIST, images are just 784-dimensional vectors, and we therefore represent the PCA vectors as images. Vectors are visualized using a perceptually uniform diverging color palette (Seaborn's "icefire"). Negative values are shown in blue, positives in red. Higher values are expressed with higher color intensity

This space contains all vectors that are reachable from $\vec{x}$ along the principal components or, equivalently, the image of the PCA decoding function θ_k.

This allows us to define a search space for adversarial examples:

$$S := U_k(\vec{x}) \cap (\vec{x} + \epsilon B^n_\infty) \tag{3}$$

Observe that S is by definition a subset of B^n_∞ of dimensionality $k \ll n$ and that, as both U_k and ϵB^n_∞ are defined by linear equations, it can be conveniently expressed as a TADS precondition.

Restricting the search for adversarial examples to S via a PCA-based transformations that adequately decodes the vectors of some k-dimensional ball δB^k_∞, as sketched in

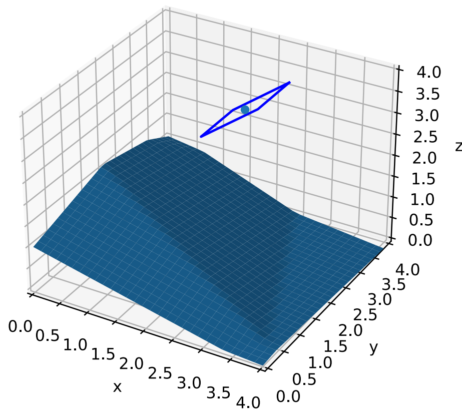


Fig. 11 The same general robustness scenario that is depicted in Fig. 7a, except that now only approximative robustness is considered

Fig. 9b, drastically reduces the computational load.[8] However, this reduction comes at a price, which is usually quite high (cf., Fig. 11): Independently of the choice of δ, it can never prove the absence of adversarial examples.

> Dimensionality reduction mitigates scalability issues, but can only be considered as a heuristics for finding adversarial examples.

5.3 Built-in PCA verification

Fundamentally, neural networks are heuristical models that seek *only* to achieve high performance, which is typically defined as the accuracy of their predictions. If a neural network is only as useful as its predictive accuracy, then any change that is made to the neural network that does not drastically alter its predictive accuracy is acceptable. This opens up a new angle to neural network verification that is unlike traditional program verification: Rather than trying to verify a given network as it is, one may well alter the network as long as this does not impair the prediction quality too much. In fact, we consider such a step (often) necessary, as classifiers defined by high-dimensional neural networks will often not be robust, but small alterations may well be.

Figure 9c sketches how the idea of PCA can be used to achieve such an alteration: The point is that each input is channelled through the low-dimensional PCA space, which, similar to the situation in the previous section, is simple enough to support verification. However we will see that, in contrast to the previous section, the special character of PCA encoding allows us to infer robustness result from the robustness results for the PCA space. More concretely, after verifying the robustness of

$$\nu_c \circ \theta_k$$

we establish a robustness result for the full modified net

$$\nu_r := \nu_c \circ \theta_k \circ \rho_k$$

The success of this method very much depends on the accuracy of ν_r, which itself strongly depends on the chosen k. We will discuss this issue in the Sect. 5.4.

In the remainder of this section, we show how to infer robustness result for the full n-dimensional vector space from robustness results for $\nu_c \circ \theta_k$. Key observation to prove this property is that PCA preserves neighborhoods:

Lemma 5 (PCA Preserves Neighborhoods) *Let ρ_k be the PCA transformation of the first k principal components. For an input $\vec{x}$ and an ϵ-neighbor $\vec{y}$ with $\|\vec{y}-\vec{x}\|_\infty \leq \epsilon$ one can estimate their distance in the image of ρ_k as*

$$\|\rho_k(\vec{y}) - \rho_k(\vec{x})\|_\infty \leq \epsilon \max_i \|\vec{p}_i\|_1$$

Proof

$$\begin{aligned}
& \|\rho_k(\vec{y}) - \rho_k(\vec{x})\|_\infty \\
= & \|\rho_k(\vec{y} - \vec{x})\|_\infty && \text{linearity} \\
= & \max_{i=1}^k |\langle \vec{p}_i, \vec{y} - \vec{x}\rangle| && \text{def. } \|\cdot\|_\infty \\
= & \max_{i=1}^k \left| \sum_{j=1}^k (\vec{p}_i)_j \cdot (\vec{y}_j - \vec{x}_j) \right| && \text{def } \langle\cdot,\cdot\rangle \\
\leq & \max_{i=1}^k \sum_{j=1}^k |(\vec{p}_i)_j| \cdot |\vec{y}_j - \vec{x}_j| && \triangle \text{ for } |\cdot| \\
\leq & \epsilon \ \max_{i=1}^k \sum_{j=1}^k |(\vec{p}_i)_j| && \text{assumption} \\
= & \epsilon \ \max_{i=1}^k \|\vec{p}_i\|_1 && \text{def. } \|\cdot\|_1
\end{aligned}$$

One can see that the bound is tight by setting

$$\vec{y} = \vec{x} + \epsilon \operatorname{sgn}(\vec{p}_i)$$

where $\operatorname{sgn}(\vec{p}_i)$ is the sign function applied component wise to $\vec{p}_i$. For that $\vec{y}$ equality holds for all steps. □

As all p_i have unit length, it is possible to derive an upper bound for $\|\vec{p}_i\|_1$ for every PCA. It is obtained when at least one principal component $\vec{p}_i$ (with some $1 \leq i \leq k$) equals

$$\vec{p}_i = \left(\pm\frac{1}{\sqrt{n}}, \ldots, \pm\frac{1}{\sqrt{n}}\right)^\top \in \mathbb{R}^n$$

In that case, the norm is $\|\vec{p}_i\|_1 = \sqrt{n}$, leading to the following proposition.

[8] Considerations about a good choice of δ are postponed to the next section, where we determine a δ that is suffices to prove robustness with he method indicated in Fig. 9c.

Corollary 1 *For every $k \leq n$ and every set of principal components $\vec{p}_1, \ldots, \vec{p}_n$ the PCA representation ρ_k satisfies*

$$\|\vec{y} - \vec{x}\|_\infty \leq \epsilon \implies \|\rho_k(\vec{x}) - \rho_k(\vec{y})\|_\infty \leq \epsilon\sqrt{n}$$

This suffices to prove the announced robustness result:

Theorem 7 (Robustness) *Let $\nu \colon \mathbb{R}^n \to \mathbb{R}^m$ by a PLNN. Then, let*

$$\nu'_r := \nu \circ \theta_k$$

$$\nu_r := \nu \circ \theta_k \circ \rho_k$$

If ν'_r is δ-robust around $\rho_k(\vec{x})$ with

$$\delta = \max_i \|\vec{p}_i\|_1 \leq \epsilon\sqrt{n} \,,$$

then ν_r is ϵ-robust around $\vec{x}$.

Proof For a proof by contraposition, we show that if ν_r is not ϵ-robust, then ν'_r is not δ-robust either. Let $\vec{z} \in \mathbb{R}^n$ be an adversarial example for ν_r with $\|\vec{z} - \vec{x}\|_\infty \leq \epsilon$. By Lemma 5 it follows that $\|\rho_k(\vec{z}) - \rho(\vec{x})\|_\infty \leq \delta$. Therefore $\rho_k(\vec{z}) \in \rho_k(\vec{x}) + \delta B^k_\infty$. And since $\vec{z}$ is an adversarial, it follows as desired that

$$\nu'_r(\rho_k(\vec{z})) = \nu_r(\vec{z}) \neq \nu_r(\vec{x}) = \nu'_r(\rho_k(\vec{x})) \qquad \square$$

In other words, proving ν'_r's robustness on the k-dimensional PCA space with radius δ directly proves robustness for the entire construct ν_r with radius $\epsilon = \frac{\delta}{\sqrt{n}}$. In the case of MNIST, n is equal to 784. Therefore, proving robustness of ν'_r for some radius δ implies robustness of ν_r with radius at least

$$\delta \geq \epsilon \geq \frac{\delta}{28}$$

> Altering the classifier via PCA enables full robustness verification via low-dimensional reasoning. However, as shown in Fig. 12, this comes at the cost of accuracy, in particular for small k.

5.4 Improving accuracy

As laid out in Sect. 5.3, PCA can be used to modify a neural network in a manner that makes it much easier to verify at the cost of some predictive accuracy. Fortunately, by modifying not only the neural network itself, but also its training process, some of that lost accuracy can be regained at almost no cost. Figure 9d sketches a way how both, verification can be eased and accuracy for low k can be improved. Key to this approach is the observation that in Fig. 9c, the PCA decoder and the first linear layer are adjacent and can therefore simply evaluate to a linear function with k-dimensional input

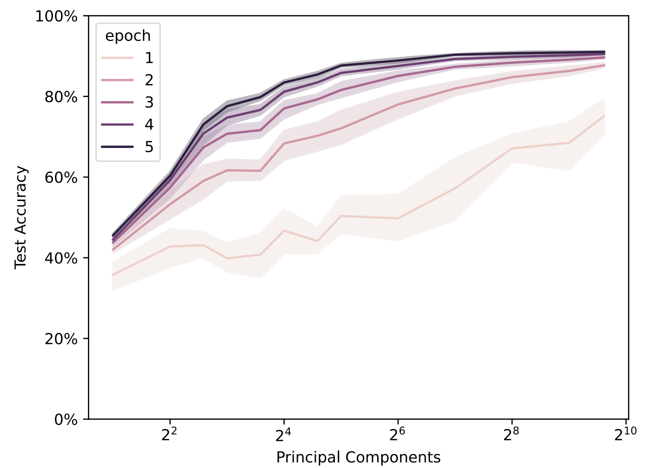


Fig. 12 A plot showing the dependence of the neural network accuracy on the number k of input dimensions allowed in the input encoding. Multiple networks were trained with different initializations for 5 epochs. Error bars illustrate 95% confidence interval. Parameters: PyTorch framework with random seeds 0, 5, 10, 15, 20, 25, 42; network with 5 layers, 10 neurons per layer, ReLU activation, kaiming normal initialization, Adam optimizer, cross-entropy loss. PCA implementation of SciPy

and an output dimension defined by the first hidden layer. Thus, rather than just modifying the original classifier via PCA auto-encoding, one can (re-) learn the entire green part through the PCA encoder. This results in a much smaller trained network ν_t which, in particular, is shielded from the 784 dimensions of MNIST by the PCA decoder. In fact, in our setup,

- the number of neurons in ν_t is essentially an order of magnitude smaller than the original net, and
- the performance of $\nu_t \circ \rho_k$ is much better for small k, as shown in Fig. 13.

> Specifically training a neural network according to PCA encoding improves accuracy, in particular, for small PCA dimensions k.

6 Experimental results

In the following, we will showcase experimental results regarding the TADS-based verification of neural networks using PCA to reduce the dimensionality of the verification problem. We will start by considering the reduction to two dimensions, allowing us to visualize the process and showcase its workings conceptually. Afterwards, we will move towards higher dimensions, examining more concrete questions of scalability.

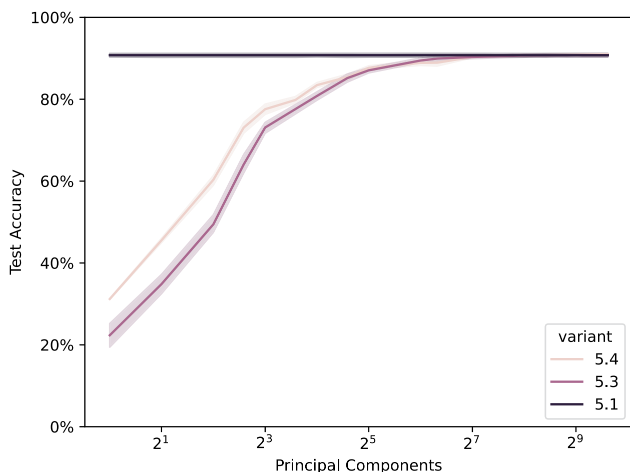


Fig. 13 Comparison of predictive accuracy on MNIST's test set for the modified PLNNs of Sect. 5.3 and Sect. 5.4. The violet line ("variant 5.1") shows the reference accuracy of an unmodified PLNN with same architecture and hyperparameters. All networks were trained as in Fig. 12, but only the accuracy after the 5-th epoch is shown

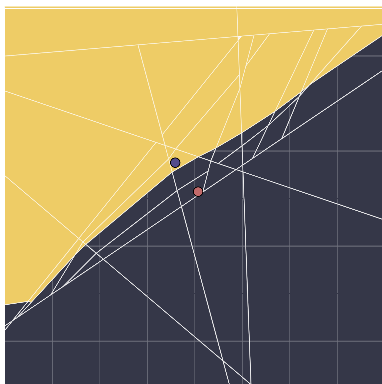


Fig. 14 A function plot representing the TADS of ν'_c in an area around $\vec{x}_9$. The yellow area contains points that are classified correctly, the black area contains points that are classified incorrectly. The blue point represents $\vec{x}_9$ and the red point represents the adversarial example $\vec{x}_5$ (Color figure online)

6.1 Conceptual showcase and visualization

For this section, we consider the neural network classifier

$$\nu_c = \arg\max \circ [\![\nu']\!]_{\mathcal{N}} \circ \rho_2$$

where ν' is a fully connected ReLU-network with 5 layers of 10 neurons each. Training is done on the MNIST training set with batches of 300 images per training step using standard settings of the ADAM optimizer [32]. This classifier uses the two-dimensional PCA representation. This allows us to plot the function represented by $[\![\nu']\!]_{\mathcal{N}} \circ \rho_2$ as done in Fig. 14.

We consider the sample $\vec{x}_9$ shown in Fig. 15. This image is classified correctly by ν_c, being assigned the label "9". However, as we will see, this classification is very unstable.

Using TADS, we can gain insight into this prediction by creating the class characterization TADS

$$t^9_{\nu_c} = t_{\nu'} \bowtie t_a \bowtie t_{x=9}$$

for ν'_c and class "9" on the infinity ball $\vec{x}_9 + 0.3 \cdot B^2_\infty$ This TADS is shown in Fig. 16 and can be interpreted as follows:

> Any input belonging to the set $\vec{x}_9 + 0.3 \cdot B^2_\infty$ that reaches the "1" terminal in the TADS depicted in Fig. 16 is classified as a "9" by ν_c (which is the desired behavior), while all others are adversarial examples.

Moreover, we can visualize the function plot corresponding to this TADS as shown in Fig. 14. Note that lines in this plot indicate decision boundaries that are implied by the non-terminal nodes in the TADS. These decision boundaries separate the regions of the piece-wise affine function encoded by the neural network. As a consequence, each polygon that is enclosed by such linear boundaries corresponds to precisely one path in the TADS $t^9_{\nu_c}$.

One can immediately observe that while ν_c classifies $\vec{x}_9$ correctly, there exists a close region of inputs that are classified incorrectly. Using the information contained in the TADS, it is trivial to obtain adversarial examples by picking any path in the TADS ending in the "0" terminal and finding a point satisfying the corresponding path condition. An example adversarial example generated in this way is shown in Fig. 15. Observe that, while being classified differently by ν_c, both images are almost identical to the human eye, which indicates that this neural network might not be entirely trustworthy even though it classified $\vec{x}_9$ correctly.

6.2 Scaling to higher dimensions

After showcasing our verification approach conceptually on a 2-dimensional problem, we now move towards higher dimensions and seek to examine how the addition of new dimensions affects scalability. To do this, we construct a neural network classifier that uses a 6-dimensional input representation instead of a 2-dimensional one (all other settings are equal)

$$\nu_c = \underbrace{\arg\max \circ [\![\nu']\!]}_{\text{classifier}} \circ \rho_6 \,.$$

This increase of dimension drastically improves the accuracy, however, at the price of an explosion in size of the corresponding TADS. All reported numbers reflect an average according to six random runs.

Accuracy The six-dimensional neural network classifier ν_c achieved roughly 74% accuracy on the test set in comparison to the 91% accuracy of the original unrestricted network, but

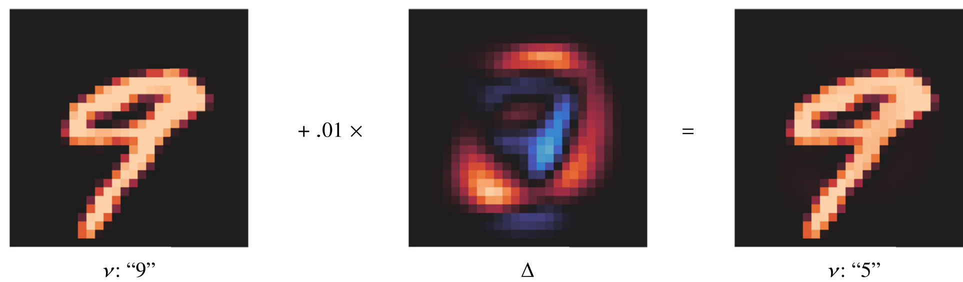


Fig. 15 MNIST sample that represents the number "9" (left) and a close adversarial example that is classified as "5" (right). The difference between the two is marginal (center). The adversarial was found in a neighboring linear region using a restricted TADS (cf., Fig. 16). Vectors are visualized using a perceptually uniform diverging color palette (Seaborn's "icefire"). Idea of representation [21]

much better than the 46% accuracy of the two-dimensional classifier (cf., Fig. 13).

We also tested different dimensions for PCA with respect to network accuracy, the results of which can be found in Fig. 12. These results show that in this case, a dimensionality reduction by an order of magnitude still allows one to achieve 90% accuracy, which is very close to the 91% accuracy of the original network.

Scalability In the two-dimensional case, we showed an example where robustness around some input could be accurately disproven with a radius of $\delta = 0.3$, which according to Lemma 5 implies

$$\epsilon = \frac{\delta}{\max_i \|p_i\|_1} \approx \frac{0.3}{20} = 0.015$$

robustness of the 785-dimensional network.[9] The corresponding TADS, describing network behavior in the space of interest, had 51 nodes and could be handled quite easily. We repeat this experiment with the input image shown in Fig. 17 and the six-dimensional neural network instead. The TADS resulting from this experiment possesses roughly 4600 nodes. This is still manageable computationally, but indicates the expected explosion in size.

7 Related work

The topic of robustness has been widely discussed in the machine learning community ever since it first gained attention in 2013 [49]. One topic of interest is research into heuristic methods that quickly and reliably find adversarial examples for modern neural networks, serving to understand how adversarial examples occur and therefore how they might be mitigated [12, 21]. As they are devised by the machine learning community, it is not surprising that these methods devised to find adversarial examples typically leverage methods from the machine learning toolbox, using gradient descent and other training heuristics to find adversarial examples.

Another natural topic with respect to robustness has been constructing neural networks that are reliably robust after training. A typical approach to this is defensive distillation [42]. Defensive distillation seeks to secure a previously trained neural network. This is achieved by using the outputs of the first neural network to train a second neural network with equivalent architecture. This process is called distilling. The additional information provided by the first neural network allows for efficient training in less training steps, reducing the need for large parameter values and therefore reducing the risk of adversarial attacks. Other approaches directly modify the training process to ensure scalability, usually by introducing additional regularization terms that are meant to steer the training process into a robust direction, often working in tandem with formal methods [27, 52, 58].

Closer to our approach are neural network verification approaches (for robustness). They can be split into two categories, approaches based on branch-and-bound tree search algorithms [15, 34] and approaches based on abstract interpretation [14].

Neural network verification—tree search Much like SAT and SMT solvers, these approaches use a branch-and-bound tree search algorithm to find a counterexample to the property of interest. A critical part of this is finding an apt ReLU configuration, i.e., which neuron activation values need to be set to 0 by the ReLU activation function and which do not. This corresponds to finding a satisfiable path in a TADS that contains a counterexample, which makes TADS based verification inherently a representative of this category.

Other examples include Reluplex [31], one of the earliest scalable neural network verifiers, and alpha-beta-crown [53], a modern method that can be regarded as current state-of-the-art [6]. Methods of this type differ mostly in the heuristics that guide their branching and bounding.

Tree search methods are accurate and leading in practice, but they tend to be more time intensive than abstract-interpretation based methods. Moreover, they are, much like

[9] The denominator of 20 was computed as the maximum of the l_1 norms of the first six principal components of MNIST.

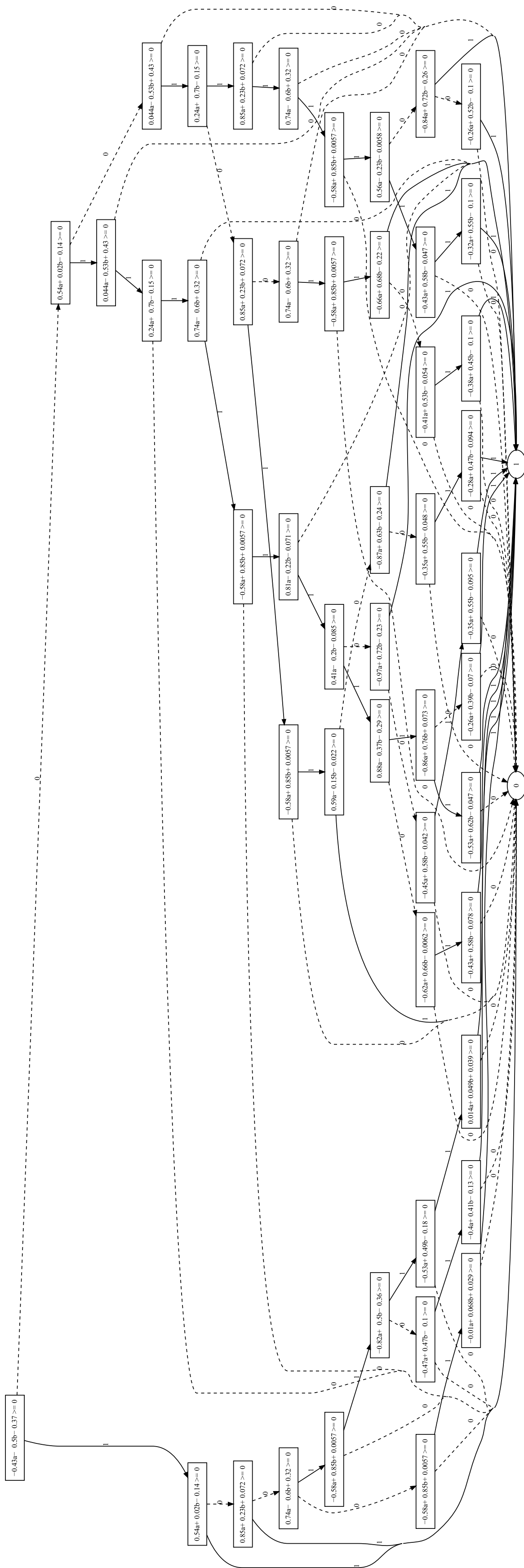


Fig. 16 A TADS representing the behavior of v' around $\vec{x}_9$. For readability, this TADS is constructed such that $\vec{x}_9$ corresponds to the vector $(0, 0)$. The terminal node "1" represents a correct classification, the node "0" an incorrect one

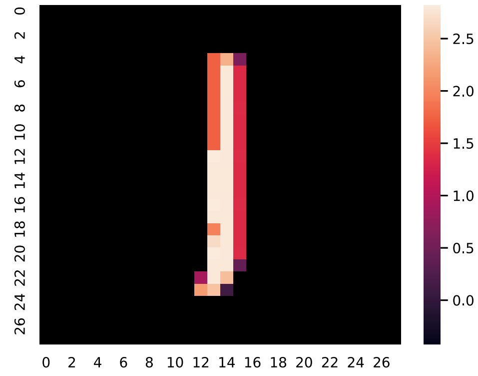


Fig. 17 An MNIST sample image $\vec{x}_1$ representing the digit "1"

TADSs, naturally restricted to piece-wise affine neural networks and cannot cover activation functions such as sigmoid or softmax.

Neural network verification—abstract interpretation These neural network verifiers define an abstract interpretation of neural networks to attain an overapproximation of the reachable states that a neural network can output on a given input region [17]. As these methods compute an overapproximation of the truly reachable states, they are safe, but not complete, i.e., they might incorrectly state that a given property is violated when it is not. On the flipside, abstract interpretation verifiers are typically computationally quite efficient and extend to neural networks that are not piece-wise affine. Examples of verifiers based on abstract interpretation include AI^2 [17] and DeepPoly [47]. Our TADS-based approach naturally also applies to abstractly interpreted neural networks.

8 Conclusion

In this paper, we have applied TADS, a whitebox representation of neural networks, to the problem of neural network robustness. To apply TADS to this problem, we have introduced precondition projection and showed how to extend the argmax function, that is typically used with neural networks in classification tasks, to generate TADS that precisely describe a neural networks classification behavior in a given area around a fixed input point. Choosing the considered robustness region as precondition, robustness becomes equivalent to the property that the entire corresponding TADS collapses to one node that then characterizes the robust classification. If this is not the case, the resulting TADS explicitly represents the set of all adversarial examples.

This unique power of TADS-based robustness verification comes at the price of an exponential complexity, which we have proposed to mitigate via PCA-based dimensionality reduction by focussing the verification on the image of a

low-dimensional PCA encoding. Three versions of this approach have been discussed:

– An approximative version that can be regarded as an elaborate search heuristics for adversarial examples,
– A transformational approach where the PLNN is extended by a preprocessing step defined by PCA-based auto-encoding, and which allows one to infer robustness of the 784-dimensional transformed network based on the analysis of the corresponding low-dimensional PCA space, and
– An approach that is based on a modified learning process, specifically tailored to the corresponding PCA-based encoding. This method leverages the machine learning toolbox to improve the accuracy of the 784-dimensional transformed network while still allowing low-dimensional robustness verification.

We believe that dimensionality reduction, as illustrated in this paper for PCA, is key to achieve neural networks that are ready for verification. The challenge is to find dimensionality reduction techniques that maintain a high level of accuracy. In our experience, the success of such techniques hinges on characteristics of the application domain. We are optimistic that this approach will widen the scope of applications where neural networks are accepted.

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

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Abdi, H., Williams, L.J.: Principal component analysis. Wiley Interdiscip. Rev.: Comput. Stat. **2**(4), 433–459 (2010)
2. Adadi, A., Berrada, M.: Peeking inside the black-box: a survey on explainable artificial intelligence (xai). IEEE Access **6**, 52138–52160 (2018)
3. Albarghouthi, A., et al.: Introduction to neural network verification. Found. Trends Program. Lang. **7**(1–2), 1–157 (2021)
4. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. Arxiv preprint (2016). arXiv:1611.01491
5. Axler, S.: Linear Algebra Done Right. Springer, Berlin (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. Bianchini, M., Scarselli, F.: On the complexity of neural network classifiers: a comparison between shallow and deep architectures. IEEE Trans. Neural Netw. Learn. Syst. **25**(8), 1553–1565 (2014)
8. Bro, R., Smilde, A.K.: Principal component analysis. Anal. Methods **6**(9), 2812–2831 (2014)
9. Brondsted, A.: An Introduction to Convex Polytopes, first edn. Springer, New York, NY (1983). https://doi.org/10.1007/978-1-4612-1148-8
10. 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)
11. Bunel, R.R., Turkaslan, I., Torr, P., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: Advances in Neural Information Processing Systems, vol. 31 (2018)
12. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP) pp. 39–57. IEEE Comput. Soc., Los Alamitos (2017)
13. 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)
14. Cousot, P., Cousot, R.: Abstract interpretation frameworks. J. Log. Comput. **2**(4), 511–547 (1992)
15. Dakin, R.J.: A tree-search algorithm for mixed integer programming problems. Comput. J. **8**(3), 250–255 (1965)
16. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Process. Mag. **29**(6), 141–142 (2012)
17. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: International Conference on Computer Aided Verification, pp. 43–65. Springer, Berlin (2020)
18. Fazlyab, M., Robey, A., Hassani, H., Morari, M., Pappas, G.: Efficient and accurate estimation of lipschitz constants for deep neural networks. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
19. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 3–18. IEEE Comput. Soc., Los Alamitos (2018)
20. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
21. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. Arxiv preprint (2014). arXiv:1412.6572
22. Gorokhovik, V.V., Zorko, O.I., Birkhoff, G.: Piecewise affine functions and polyhedral sets. Optimization **31**(3), 209–221 (1994)
23. 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.
24. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al.: Recent advances in convolutional neural networks. Pattern Recognit. **77**, 354–377 (2018)
25. Guidotti, R., Monreale, A., Pedreschi, D.: The ai black box explanation problem. ERCIM News **116**, 12–13 (2019)
26. 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), 93 (2018). https://doi.org/10.1145/3236009.
27. Han, B., Yao, Q., Yu, X., Niu, G., Xu, M., Hu, W., Tsang, I., Sugiyama, M.: Co-teaching: Robust training of deep neural networks with extremely noisy labels. In: Advances in Neural Information Processing Systems, vol. 31. (2018)

28. 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. PMLR Proceedings of Machine Learning Research, vol. 97, pp. 2596–2604. (2019). https://proceedings.mlr.press/v97/hanin19a.html
29. Hanin, B., Rolnick, D.: Deep relu networks have surprisingly few activation patterns. In: Advances in Neural Information Processing Systems, vol. 32. (2019)
30. Hinz, P.: Using activation histograms to bound the number of affine regions in ReLU feed-forward neural networks. Arxiv (2021). arXiv:2103.17174
31. 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)
32. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. Arxiv preprint (2014). arXiv:1412.6980
33. Kurakin, A., Goodfellow, I., Bengio, S., et al.: Adversarial Examples in the Physical World (2016)
34. Leofante, F., Narodytska, N., Pulina, L., Tacchella, A.: Automated verification of neural networks: advances, challenges and perspectives (2018). ArXiv preprint. arXiv:1805.09938
35. Luo, B., Liu, Y., Wei, L., Xu, Q.: Towards imperceptible and robust adversarial example attacks against neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)
36. Maclagan, D., Sturmfels, B.: Introduction to Tropical Geometry, vol. 161. Am. Math. Soc., Providence (2021)
37. Magnus, R.: Metric spaces. In: Metric Spaces, pp. 1–27. Springer, Berlin (2022)
38. Maragos, P., Charisopoulos, V., Theodosis, E.: Tropical geometry and machine learning. Proc. IEEE **109**(5), 728–755 (2021)
39. Montufar, G.F., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: Advances in Neural Information Processing Systems vol. 27 (2014)
40. Mothilal, R.K., Sharma, A., Tan, C.: Explaining machine learning classifiers through diverse counterfactual explanations. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, pp. 607–617 (2020)
41. Ovchinnikov, S.: Discrete piecewise linear functions. Eur. J. Comb. **31**(5), 1283–1294 (2010)
42. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 582–597. IEEE Comput. Soc., Los Alamitos (2016)
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. PMLR, pp. 2847–2854. (2017)
45. Schlüter, M., Nolte, G., Murtovi, A., Bernhard, S.: Towards rigorous understanding of Neural Networks via semantics-preserving transformations. Int. J. Softw. Tools Technol. Transf. (2023, in press). https://doi.org/10.1007/s10009-023-00700-7
46. Serra, T., Tjandraatmadja, C., Ramalingam, S.: Bounding and counting linear regions of deep neural networks. In: International Conference on Machine Learning. PMLR, pp. 4558–4566. (2018)
47. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. In: Proceedings of the ACM on Programming Languages. POPL vol. 3, pp. 1–30 (2019)
48. Sudjianto, A., Knauth, W., Singh, R., Yang, Z., Zhang, A.: Unwrapping the black box of deep ReLU networks: Interpretability, diagnostics, and simplification. Arxiv (2020). arXiv:2011.04041
49. 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
50. Theodoridis, S., Koutroumbas, K.: Pattern Recognition. Elsevier, Amsterdam (2006)
51. 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)
52. Wang, S., Chen, Y., Abdou, A., Jana, S.: Mixtrain: Scalable training of verifiably robust neural networks. Arxiv preprint (2018). arXiv:1811.02625
53. 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 neural network robustness verification. Adv. Neural Inf. Process. Syst. **34**, 29909–29921 (2021)
54. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemom. Intell. Lab. Syst. **2**(1–3), 37–52 (1987)
55. 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)
56. Zhang, L., Naitzat, G., Lim, L.H.: Tropical geometry of deep neural networks. In: International Conference on Machine Learning. PMLR, pp. 5824–5832 (2018)
57. Zhang, X., Wu, D.: Empirical studies on the properties of linear regions in deep neural networks. Arxiv preprint (2020). arXiv:2001.01072
58. Zheng, S., Song, Y., Leung, T., Goodfellow, I.: Improving the robustness of deep neural networks via stability training. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4480–4488 (2016)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.