Formal Methods in System Design (2024) 64:178–199
https://doi.org/10.1007/s10703-024-00457-y

RESEARCH

The hexatope and octatope abstract domains for neural network verification

Stanley Bak[1] · Taylor Dohmen[2] · K. Subramani[3] · Ashutosh Trivedi[2] · Alvaro Velasquez[2] · Piotr Wojciechowski[3]

Received: 7 December 2023 / Accepted: 16 May 2024 / Published online: 17 June 2024

Abstract

Efficient verification algorithms for neural networks often depend on various abstract domains such as *intervals*, *zonotopes*, and *linear star sets*. The choice of the abstract domain presents an expressiveness vs. scalability trade-off: simpler domains are less precise but yield faster algorithms. This paper investigates the *hexatope* and *octatope* abstract domains in the context of neural net verification. Hexatopes are affine transformations of higher-dimensional hexagons, defined by difference constraint systems, and octatopes are affine transformations of higher-dimensional octagons, defined by unit-two-variable-per-inequality constraint systems. These domains generalize the idea of zonotopes which can be viewed as affine transformations of hypercubes. On the other hand, they can be considered as a restriction of linear star sets, which are affine transformations of arbitrary $\mathcal{H}$ -Polytopes. This distinction places hexatopes and octatopes firmly between zonotopes and linear star sets in their expressive power, but what about the efficiency of decision procedures? An important analysis problem for neural networks is the *exact range computation* problem that asks to compute the exact set of possible outputs given a set of possible inputs. For this, three computational procedures are needed: (1) optimization of a linear cost function; (2) affine mapping; and (3) over-approximating the intersection with a half-space. While zonotopes allow an efficient solution for these approaches, star sets solves these procedures via linear programming. We show that these operations are faster for hexatopes and octatopes than they are for the more expressive linear star sets by reducing the linear optimization problem over these domains to the minimum cost network flow, which can be solved in strongly polynomial time using the Out-of-Kilter algorithm. Evaluating exact range computation on several ACAS Xu neural network benchmarks, we find that hexatopes and octatopes show promise as a practical abstract domain for neural network verification.

Keywords Neural network verification · Abstract domains · Zonotopes · Star sets · Difference constraint system · Unit-two-variables-per- inequality (UTVPI) constaint system

Extended author information available on the last page of the article

1 Introduction

The success of deep feed-forward neural networks (DNN) in computer vision and speech recognition has prompted applications in critical infrastructure. These applications range from using pre-trained perception and speech-recognition modules in safety-critical logic (self-driving cars and medical decision making) to learning controllers from reinforcement signals [38] to learning succinct representations of formally verified controllers (ACAS Xu). The increasing prevalence of DNNs in safety-, privacy-, and social- critical systems motivates the focus of the formal methods community [2, 4, 7, 41] in developing verification technology to meet the challenge of improving trust in DNNs. Robustness [12, 25], safety [24, 30], and fairness [8, 11] are among the key verification problems over neural networks.

Abstract interpretation [3, 15] is a well-established framework for program verification that formalizes the exploration of the program semantics at the granularity provided by the underlying domain. For example, intervals [15] form an abstract domain facilitating analysis in which sets of states are represented as hyperrectangles. Other abstract domains such as difference constraints, octagons (unit-two-variables-per-inequality or UTVPI), and polyhedral (linear constraints) have been successfully deployed for the verification of DNNs. However, the multi-layer architecture of DNNs, when combined with linear function composition followed by a non-linear activation function at each layer, results in the repeated intersection of abstract spaces with linear inequalities. For this reason, abstract domains that do not permit an efficient *affine mapping* suffer in exploring the layered state space of the DNNs.

Zonotopes [36] solve this problem by representing an abstract set as an affine mapping of an interval generator set. For zonotopes, the key operations for DNN verification, such as nonemptiness, optimization, and over-approximation, can be performed via efficient, enumerative procedures. Linear star sets [17, 42] generalize zonotopes by representing the generator set using the polyhedral domains. This generalization, while improving the expressiveness, leads to the decision procedures depend upon solving linear programs, which tends to be the performance bottleneck in the overall algorithm. While linear programming is known to be solvable in polynomial time, via a number of celebrated interior-point algorithms [28], there is no known strongly polynomial algorithm. Dantzig's simplex algorithm is a popular algorithm to solve LP and works well in practice, but for general LPs, the time complexity of the simplex algorithm is not polynomial [29], and subexponential lower bounds hold even for randomized pivoting rules [18].

For some subclasses of linear programming problems, more efficient solutions exist. In particular, when the constraints are restricted to difference constraints ($x_i - x_j \leq c$) or UTVPI constraints ($\pm x_i \pm x_j \leq c$), then the duals of the corresponding LPs can be reduced to *minimum cost flow* (MCF) problems [1], for which there exist strongly polynomial time algorithms [21]. The Out-of-Kilter algorithm is one popular algorithm for solving minimum cost flow that also produces a solution to the dual [1]. It runs in time $O((m^2 + m \cdot n \cdot \log n) \cdot U)$ on a network with m arcs and n nodes and maximum supply/demand U. Alternatively, the network simplex algorithm is a specialized version of the simplex to solve minimum cost flow problems. Unlike standard simplex, network simplex runs in polynomial time [35]. Given its relative efficiency, it is natural to ask:

> *Is it possible to replace linear programming with min-cost flow in neural network verification?*

This question motivates the investigation of sub-classes of star sets that are more general than zonotopes, but enable efficient decision procedures based on MCF problems. For this purpose, we introduce *octatopes*: sets that can be defined as affine maps of UTVPI constrained sets (octagons [34]). Since octatopes are a special class of star sets, the affine transformation remains efficient. We also study *hexatopes* as the images of difference constrained sets (hexagons [34] or zones [10]). A key contribution of this paper is that the key operations required for verification using octatopes and hexatopes can be performed efficiently using algorithms for MCF problems.

Given that the MCF problem can be solved efficiently via Out-of-Kilter algorithm and network simplex (touted [9] to be 200–300 times faster than simplex), this benefit will translate to the efficiency of octatopes/hexatopes for LP-intensive applications like reachability analysis of neural networks. While the current state-of-the-art implementations of the algorithms for the MCF problem are not as advanced as those for LP, we believe that this will change in light of the proposed application. We implement the octatope and hexatope abstract domains and show their effectiveness on several ACAS Xu networks [26], a popular benchmark for neural network verification. An extended abstract of this paper was presented at the 25th International Symposium on Formal Methods [5].

Related work A growing body of research exists on different methods to verify neural networks [32], including recent tool competitions [6]. Algorithms can be categorized into search, optimization, and reachability solutions. In the space of search procedures, the seminal Reluplex method proposes an extension of the simplex algorithm used for linear programming to handle ReLU networks [26]. This method has been widely adopted and extended by, for example, posing verification as a constraint satisfaction problem [27]. This can then be solved using off-the-shelf Satisfiability Modulo Theory (SMT) solvers like z3 [16]. The use of SMT enables reasoning over different activation functions and topologies.

Interval arithmetic is another popular approach often used to estimate the range of output values given a range of inputs while tracking the input and output ranges of individual activation functions [45]. This can be computed by using linear programming to derive lower and upper bounds for a given node in the network. The work of [22] combines this with symbolic interval propagation and gradient descent to find counter-examples to the over-approximations established by the linear programming solutions. More sophisticated node splitting strategies that account for downstream effects on successor nodes can also be used as part of the symbolic interval propagation phase [23]. Per-neuron split constraints can also further improve efficiency [46].

Optimization solutions to the verification based on ILP have been explored. This is a natural formulation for the verification of neural networks due to the use of affine transformations and the fact that piecewise linear activation functions can be encoded using a set of binary linear constraints [2]. The work in [39] extends similar ideas by estimating the maximum disturbance that is permitted at the input and proposing presolve procedures to speed up the solution.

Although solutions based on SMT-solving and mathematical programming are often complete, they require the entire network to be encoded within the corresponding constraints, thereby limiting scalability. In contrast to these search and optimization solutions, the use of reachability analysis for verification of neural networks has been shown to scale to larger instances at the cost of completeness. Examples of this include the use of zonotope and star set abstract domains. The former can be efficiently employed to compute conservative over-approximations of output bounds of nodes in a

network [20], whereas linear programming can be employed for the latter to find tight bounds at the cost of scalability [44]. The work proposed herein seeks to advance the state of verification methods based on reachability analysis by providing tighter over-approximations than zonotopes and more efficient computations than star sets.

2 Preliminaries

Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{Q}$ denote the set of rational numbers. We write $\mathbb{R}^{m\times n}$ for the set of all $m\times n$ dimensional matrices of reals.

For a matrix $M \in \mathbb{R}^{m\times n}$, we write $M(i,\cdot) \in \mathbb{R}^{1\times n}$ and $M(\cdot,j) \in \mathbb{R}^{m\times 1}$ for the i^{th} row vector and j^{th} column vector, respectively, of M, for $1 \le i \le m$ and $1 \le j \le m$. Similarly, we write $M(i,j)$ for the matrix element at row i and column j. By default, a vector is a column vector and we associate a set of matrices $\mathbb{R}^{m\times 1}$ with the set of vectors $\mathbb{R}^m$. For a matrix $M \in \mathbb{R}^{m\times n}$ we write $M^{\mathsf{T}} \in \mathbb{R}^{n\times m}$ for its transpose matrix. For a row vector $\boldsymbol{v} \in \mathbb{R}^{1\times n}$, we write $\boldsymbol{v}^{\mathsf{T}} \in \mathbb{R}^n$ for the corresponding (transposed) vector. We write $\mathbf{1}_n$ for the all-ones vector of size n and I for the identity matrix of some fixed dimension (often clear from context). For a (column) vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ we write $\boldsymbol{v}_i$ for its i^{th} element. For a vector $\boldsymbol{v} \in \mathbb{R}^m$ and scalar $a \in \mathbb{R}$, we write $a \cdot \boldsymbol{v}$ for the vector $(a \cdot \boldsymbol{v}_1, \ldots, a \cdot \boldsymbol{v}_m)$. For two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$, we write $\boldsymbol{u} \cdot \boldsymbol{v}$ for their dot product, i.e., $\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{m} \boldsymbol{u}_i \cdot \boldsymbol{v}_i$. For two matrices $M \in \mathbb{R}^{m\times n}$ and $N \in \mathbb{R}^{n\times p}$, their product $MN \in \mathbb{R}^{m\times p}$ is defined as $MN(i,j) = M(i,\cdot)^{\mathsf{T}} \cdot N(\cdot,j)$.

We call a function $f : \mathbb{R}^n \to \mathbb{R}^m$ *linear* if $f(\boldsymbol{u}) + f(\boldsymbol{v}) = f(\boldsymbol{u} + \boldsymbol{v})$ and $f(a \cdot \boldsymbol{v}) = a \cdot f(\boldsymbol{v})$ for all scalars $a \in \mathbb{R}$ and vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$. A linear function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be represented as a matrix $A \in M^{m\times n}$ such that $f(v) = A\boldsymbol{v}$ for every $\boldsymbol{v} \in \mathbb{R}^n$. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is affine if it is a sum of a linear function and a constant, i.e., $f(\boldsymbol{v}) = A\boldsymbol{v} + \boldsymbol{b}$ for some $A \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$.

Definition 1 *(Linear Constraint System (LCS))* A linear constraint over a vector $\boldsymbol{x} \in \mathbb{R}^n$ is a constraint of the form

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \le b,$$

where $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$ and $b \in \mathbb{R}$. A linear constraint system (LCS) is a conjunction of linear constraints $A\boldsymbol{x} \le \boldsymbol{b}$ where $A \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$.

3 Verification of neural networks

We primarily work with feedforward neural networks (NNs) with rectified linear unit (ReLU) activation functions. We focus on networks with k fully-connected layers, also called multi-layer perceptrons. A ReLU is a commonly used activation function $\sigma : \mathbb{R} \to \mathbb{R}$ defined as $\sigma(x) = \max\{x, 0\}$. We can extend this function from scalars to vectors as $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ in a straightforward fashion by applying ReLU component-wise. This setup is the most typical situation considered for neural network verification tools [6], although extensions have been made to other layer types [40, 43] and activation functions [37].

Formally, a *neural network* (NN) is a function $f : \mathbb{R}^i \to \mathbb{R}^o$, where i is the input dimension and o is the output dimension. Each layer of the network is also a function $f_k : \mathbb{R}^{i_k} \to \mathbb{R}^{o_k}$ with its own input dimension i_k and output dimension o_k, defined as the composition $\sigma_k \circ \alpha_k$ in which $\alpha_k : \mathbb{R}^{i_k} \to \mathbb{R}^{o_k}$ is an affine mapping that is followed by the application of a ReLU $\sigma_k : \mathbb{R}^{o_k} \to \mathbb{R}^{o_k}$ of appropriate dimension. For the network to be well formed, it is required that the input dimension of the first layer is i, the output dimension of the final layer is o, and all intermediate layers have input dimensions equal to the output dimension of the preceding layers and output dimension equal to the input dimension of the following layer. For a network with n layers, the function f is defined by the composition $f_n \circ f_{n-1} \circ \ldots \circ f_2 \circ f_1$. Figure 1 shows a feedforward neural network with three layers.

Given the importance of neural networks in critical infrastructure, establishing formal guarantees on their performance is of paramount interest. The key verification problems concerning neural networks include robustness [12, 25], safety [24, 30], and fairness [8, 11]. Given a neural network and an input, the *robustness problem* [12, 25] asks whether small perturbations around the input keep the output close to the original value. The safety problem [24, 30] asks whether for every input from a given set, the neural network outputs do not go outside a given set. Finally, the fairness problem [8, 11] for neural networks is typically posed by assuming a partition over the input features within protected and unprotected features, and the verification question involves checking whether the output is robust while keeping the protected features constant. All of these problems require computing the range of the output for a given range of the input. Since our focus is on developing verification support, we frame the problem in a broader context as the range computation and empty intersection problems.

Definition 2 *(Exact Range Computation Problem)* Given a neural network implementing the function $f : \mathbb{R}^i \to \mathbb{R}^o$ and an input set $\mathcal{I} \subseteq \mathbb{R}^i$, the exact range computation problem is to compute the image $\mathsf{Range}(f, \mathcal{I}) = \{f(\boldsymbol{x}) \ : \ \boldsymbol{x} \in \mathcal{I}\}$.

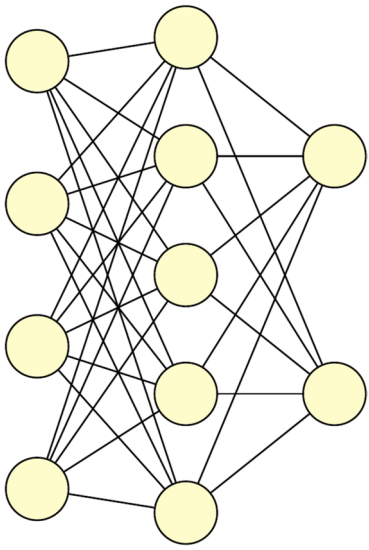

Fig. 1 A three layer neural network with an input layer with four neurons, an internal layer with five neurons, and an output layer with 2 neurons. This network represents a function $f : \mathbb{R}^4 \to \mathbb{R}^2$.

Definition 3 *(Neural Network Verification Problem)* Given an input set $\mathcal{I} \subseteq \mathbb{R}^i$, an unsafe set $\mathcal{U} \subseteq \mathbb{R}^o$, and a neural network that computes $f : \mathbb{R}^i \to \mathbb{R}^o$, the neural network verification problem asks whether

$$\mathsf{Range}(f, \mathcal{I}) \cap \mathcal{U} = \emptyset.$$

As is typical with the state-of-practice in neural network verification, we restrict the input sets and unsafe sets to be defined by systems of linear constraints $\mathcal{I} = \left\{\boldsymbol{x} \in \mathbb{R}^i : A_i \boldsymbol{x} \leq b_i\right\}$ and $\mathcal{U} = \left\{\boldsymbol{x} \in \mathbb{R}^o : A_u \boldsymbol{x} \leq b_u\right\}$. The popular ACAS Xu neural network verification benchmarks [26] satisfy this assumption, and will be used in our experimental evaluation.

3.1 Abstraction based methods

We now describe a general approach to the neural network verification problem that is based on interpreting sets in the domains and codomains of NNs (and their component layers) with respect to some *abstract domain* which, in our case, comprises a particular class of geometric objects. Suppose, for the time being, that an appropriate abstract domain has already been chosen. Given an instance of the neural network verification problem, the abstraction based paradigm proceeds roughly as follows.

Algorithm 3.1 Abstract Neural Network Verification

1: **input:** (I) $f_k = \sigma_k \circ \alpha_k$, for each layer k in the given NN, (II) input set $\mathcal{I}$, (III) unsafe set $\mathcal{U}$.
2: Identify elements $\mathcal{A}, \mathcal{B}$ of the abstract domain such that $\mathcal{I} \subseteq \mathcal{A}$ and $\mathcal{U} \subseteq \mathcal{B}$.
3: **for** each layer k **do**
4: Compute the image $f_k(\mathcal{A})$.
5: Find an element $\mathcal{A}'$ of the abstract domain such that $f_k(\mathcal{A}) \subseteq \mathcal{A}'$.
6: Set $\mathcal{A} \leftarrow \mathcal{A}'$.
7: **return** $\mathcal{A} \cap \mathcal{B} = \emptyset$.

From the above algorithm, we distill three operations that are fundamental to the overall approach:

1. **affine transformation** is required to compute $\alpha_k(\mathcal{A})$,
2. **half-space intersection** is necessary for computing $\sigma_k(\alpha_k(\mathcal{A}))$, and
3. **emptiness determination** is essential for deciding the final return value.

For the purposes of abstract neural network verification, any reasonable abstract domain should facilitate algorithmic execution of these operations. Besides the feasibility of carrying out the critical operations, additional properties of an abstract domain that are helpful in this context may be inferred from Algorithm 3.1. If a given abstract domain is *closed under affine transformation*, then the instruction on line 5 may be omitted. If the abstract domain is *closed under intersection*, the emptiness check to determine the return value is simplified. We proceed by examining two particular geometric abstract domains that are useful in the context of abstract neural network verification.

3.2 Abstract domains: zonotopes and linear star sets

A relatively simple abstract domain is the family of *zonotopes* [3, 20] and it is defined as an affine image of a hypercube.

Definition 4 *(Zonotope)* An n-dimensional zonotope is the image of a p-dimensional hypercube under an affine transformation $\mathbb{R}^p \to \mathbb{R}^n$. It is given as a pair $Z = \langle \boldsymbol{c}, G \rangle$ comprising a center $\boldsymbol{c} \in \mathbb{R}^n$ and a collection of generator vectors $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_p \in \mathbb{R}^n$ forming a matrix $G = \begin{bmatrix} \boldsymbol{g}_1 & \cdots & \boldsymbol{g}_p \end{bmatrix} \in \mathbb{R}^{n \times p}$.

The semantics of Z are defined as

$$[\![Z]\!] = \left\{ G\boldsymbol{x} + \boldsymbol{c} \ : \ -\mathbf{1}_p \leq \boldsymbol{x} \leq \mathbf{1}_p \right\}.$$

A more expressive abstract domain is the family of *linear star sets* [33, 42] that can be considered as affine image of a polytope.

Definition 5 *(Linear Star Set)* Linear star sets generalize zonotopes by letting the kernel be defined by an LCS; a linear star set is the image of a p-dimensional polytope under an affine transformation $\mathbb{R}^p \to \mathbb{R}^n$. Formally, an n-dimensional star set S is specified as a tuple $\langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ including a matrix A and vector $\boldsymbol{b}$ representing the polytope $A\boldsymbol{x} \leq b$, a center $\boldsymbol{c} \in \mathbb{R}^n$, and a collection of generator vectors $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_p \in \mathbb{R}^n$ that form a matrix $G = \begin{bmatrix} \boldsymbol{g}_1 & \cdots & \boldsymbol{g}_p \end{bmatrix} \in \mathbb{R}^{n \times p}$. The semantics of S are defined as

$$[\![S]\!] = \{ G\boldsymbol{x} + \boldsymbol{c} \ : \ A\boldsymbol{x} \leq \boldsymbol{b} \}.$$

Some examples of zonotopes and linear star sets are shown in Fig. 2. Both a zonotope and a linear star set may be viewed as an n-dimensional image of a polytope—which we refer to as the *kernel*—under affine transformation. For zonotopes, the kernel

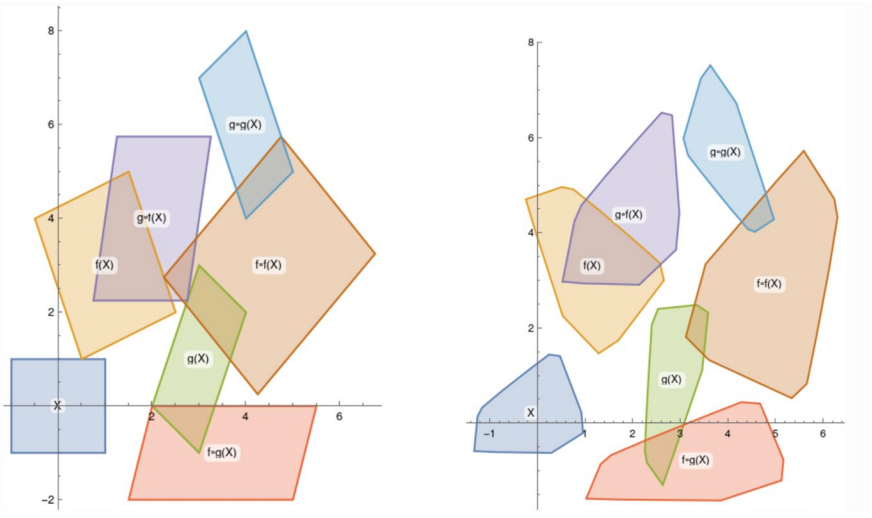

Fig. 2 (**Left.**) A hypercube (X), two zonotopes $f(X)$ and $g(X)$, and their various compositions ($f \circ g, g \circ f, f \circ f$, and $g \circ g$) as zonotopes. (**Right.**) A polytope (X), two linear star sets $f(X)$ and $g(X)$, and their various compositions as linear star sets.

is a hypercube, while for linear star sets the kernel is a set defined by a generic polytope. Thus, the kernel of a linear star set may be specified as a linear constraint system. The kernel of a zonotope can be encoded by an *interval constraint system*, which is a system of linear constraints of the form $a_i \leq x_i \leq b_i$ where $a_i, b_i \in \mathbb{R}$. In the present work, we examine geometric abstract domains that fall between zonotopes and linear star sets in terms of precision and complexity. Such domains shall be characterized by the subclasses of linear constraint systems that specify their kernels.

3.3 Fundamental Theorems of linear star sets

We now state formally the results [42] about linear star sets that are fundamental to our approach towards neural network verification. Each of the following theorems is key to ensuring that the operations of affine transformation, half-space intersection, and emptiness checking are available and feasible for linear star sets. Since each abstract domain considered in this paper forms a subclass of linear star sets, these serve as a template for the stronger results we subsequently establish for hexatopes and octatopes.

Theorem 1 *Linear star sets are closed under affine transformation.*

Theorem 2 *The problems of linear optimization and emptiness checking over linear star sets can be solved in polynomial time by reduction to linear programming.*

Theorem 3 *The intersection of a linear star set* $S = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ *and half space* $\{\boldsymbol{y} \mid M\boldsymbol{y} \leq \boldsymbol{d}\}$ *is another linear star set* $S' = \langle \boldsymbol{c}, G, A', \boldsymbol{b'} \rangle$ *where* $A'\boldsymbol{x} \leq \boldsymbol{b}'$ *comprises the conjunction of constraints*

$$A\boldsymbol{x} \leq \boldsymbol{b} \qquad \text{and} \qquad MG\boldsymbol{x} \leq \boldsymbol{d} - M\boldsymbol{c}.$$

3.4 Illustration of NN verification with zonotopes

Now that we have concretely defined some abstract domains, the process of Algorithm 3.1 can be explicitly visualized (in the two dimensional case). Figure 3 depicts the first iteration of the algorithm using zonotopes as the abstract domain:

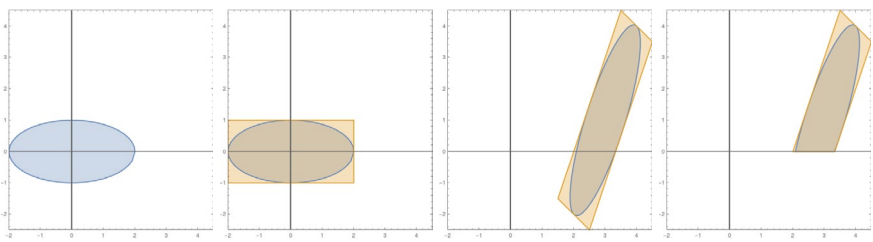

Fig. 3 An illustration of the first step of Algorithm 3.1 using the zonotope abstract domain.

- Figure 3a displays the initial set, which is an elliptical region in the plane in this case;
- Figure 3b overlays the initial set with a box;
- Figure 3c shows an affine image of the initial set and its approximating box;
- Figure 3d plots the result of applying a ReLU after the affine mapping.

For other abstract domains, the graphical illustration remains unchanged, except the approximations are stored as those corresponding domains.

3.5 Organization

In the rest of the paper, we extend the notion of zonotopes to define octatopes and hexatopes and develop a series of results, analogous to Theorems 1 to 3, that provide the theoretical framework for the application of these abstract domains to the verification of neural networks.

4 Hexatopes

In this section, we introduce hexatopes that generalize zonotopes, with the kernel being specified using a difference constraint system. We will develop analogs of Theorems 1 to 3 for hexatopes.

Definition 6 *(Difference Constraint System)* A difference constraint is a linear constraint of the form

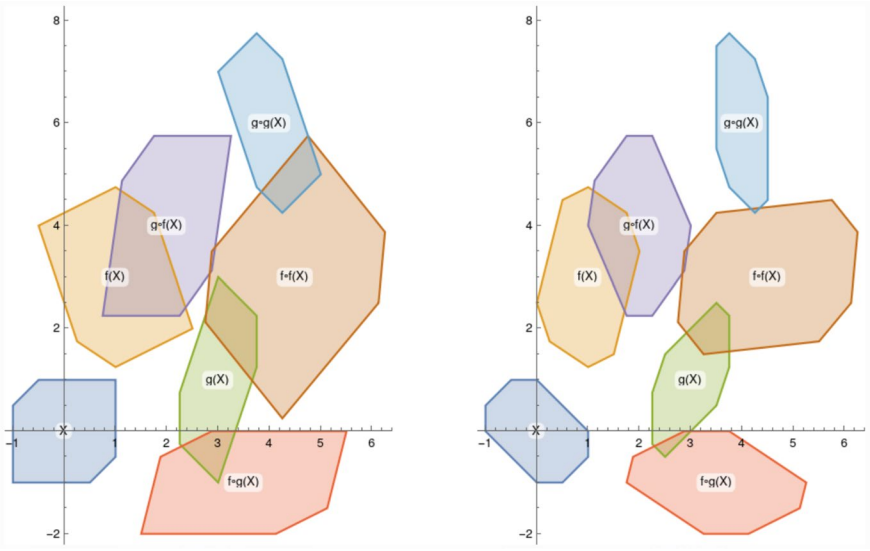

Fig. 4 (**Left.**) A difference constraint system (X), two hexatopes $f(X)$ and $g(X)$, and their various compositions. (**Right.**) A UTVPI system (X), two octatopes $f(X)$ and $g(X)$, and their various compositions.

$$x_i - x_j \leq b, \qquad \text{where} \qquad b \in \mathbb{R}.$$

A difference constraint system (DCS) is a conjunction of difference constraints.

Definition 7 *(Hexatope)* A hexatope $H = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ is a special type of linear star set, having a kernel $A\boldsymbol{x} \leq \boldsymbol{b}$ defined by a difference constraint system.

Some examples of hexatopes are shown in Fig. 4. Our first result mirrors Theorem 1 and establishes closure under affine mappings for hexatopes.

Theorem 4 *Hexatopes are closed under affine transformation.*

Proof From Theorem 1 it follows for any hexatope $H = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ and any affine mapping $f(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{d}$, that $\{W\boldsymbol{x} + \boldsymbol{d} \ : \ \boldsymbol{x} \in [\![H]\!]\}$ is linear star set $H' = \left\langle \boldsymbol{c'}, G', A, \boldsymbol{b} \right\rangle$ where

$$\boldsymbol{c'} = W\boldsymbol{c} + \boldsymbol{d} \qquad \text{and} \qquad G' = \left[W\boldsymbol{g}_1 \ \cdots \ W\boldsymbol{g}_p \right].$$

Since the transformation does not change the kernel, H' is indeed a hexatope. □

4.1 Linear optimization via minimum cost flow

In this subsection, we show that for any linear optimization problem over a difference constraint system, the dual problem can be reduced to the *minimum cost flow problem* [1]. The minimum cost flow problem is a canonical problem in flow networks (also known as transportation networks), where a network is represented as a directed graph with vertices (nodes) as *junctions* and edges (arcs) as *channels* between those junctions. One assumes that some quantity of interest-such as water, oil, population, or traffic-flows through the network respecting certain capacity constraints of the channels and we are interested in optimizing the total cost associated the flow, given as the cost per unit amount of flow in a channel. Formally, we define a flow network and the corresponding minimum-cost flow problem as follows.

Definition 8 A *flow network* $\boldsymbol{G} = (V, E, c, a, d)$ is a directed graph $G = (V, E)$ with a capacity $c : E \to \mathbb{R}_{\geq 0}$ and a cost $a : E \to \mathbb{R}$ associated with every edge (arc) and a demand

$d : V \to \mathbb{R}$ associated with every vertex (node). We assume that $\sum_{v \in V} d(v) = 0$. The *minimum cost flow* (MCF) problem can be stated as follows:

$$\begin{aligned} \text{Minimize} \quad & \sum_{(u,v)\in E} f(u,v) \cdot a(u,v) \\ \text{subject to} \quad & \sum_{u\in V} f(u,v) - \sum_{u\in V} f(v,u) = d(v) \qquad \text{for all } v \in V, \\ 0 \le f(u,v) \quad & \le c(u,v) \text{for all } (u,v) \in E. \end{aligned}$$

It is well known that there exist strongly polynomial time algorithms [21] for the MCF problem. One example is the Out-of-Kilter algorithm (Algorithm 4.1) that we now review.

Algorithm 4.1 OUT-OF-KILTER($\boldsymbol{G} = (G = (V, E), c, a, d)$)

```
1: Initialize the potential as π ← 0.
2: Let f be a flow in G.
3: Construct the residual network G_f.
4: Compute the kilter number k(u, v) of each edge (u, v) in G_f.
5: while (G_f contains an edge with positive kilter number) do
6:     Select an edge (u, v) in G_f with positive kilter number.
7:     Let the weight of each edge (u, v) in G_f be max{0, c^π(u, v)}.
8:     For w ∈ V \ {u, v}, let l(w) be the weight of the least weight path from v to w.
9:     Let P be a shortest path from v to u.
10:    For each node w, set π(w) ← π(w) − l(w).
11:    if (c^π(u, v) < 0) then
12:        Q ← P ∪ {(u, v)}.
13:        δ ← min_{(u,v)∈Q} r(u, v).
14:        Augment δ units of flow along Q.
15:        Update f and G_f.
16: return f.
```

Out-of-Kilter algorithm A pseudocode for the Out-of-Kilter algorithm is given as Algorithm 4.1. It starts with a possibly infeasible flow and iteratively modifies this flow in a way that decreases the infeasibility of the solution and moves it closer to optimality. Each step of the algorithm consists of solving a shortest path problem and augmenting the flow along the shortest path. It operates on the residual network $\boldsymbol{G}_f$ corresponding to the current flow f. This residual network is constructed as follows.

Feasible Edges:	If $f(u,v) < c(u,v)$, we add the edge (u, v) with a residual capacity of $r(u,v) = c(u,v) - f(u,v)$ and cost $a(u, v)$. If $f(u,v) > 0$, we add the edge (v, u) with a residual capacity of $r(v,u) = f(u,v)$ and cost $-a(u,v)$.
Lower-Infeasible Edges:	If $f(u,v) < 0$, we add edge (u, v) with residual capacity $r(u,v) = -f(u,v)$ and cost $a(u,v)$.
Upper-Infeasible Edges:	If $f(u,v) > c(u,v)$, we add the edge (v, u) with a residual capacity of $r(v,u) = f(u,v) - c(u,v)$ and cost $-a(u,v)$.

For each vertex v in the residual network, the algorithm maintains a potential $\pi(v)$ and for each edge (u, v) with cost $a(u, v)$, it maintains the reduced cost $a^\pi(u,v) = c(u,v) - \pi(u) + \pi(v)$. Additionally, for each edge in the residual network, it maintains a kilter number $k(u, v)$ which is 0 if $c^\pi(u,v) \ge 0$ and is the residual capacity

$r(u, v)$ if $c^{\pi}(u, v) < 0$. This kilter number represents the change in flow required so that each edge satisfies its optimality condition.

Note that the node potentials π and reduced costs c^{π} corresponding to the optimal flow f are the optimal solution of the dual problem [1]. The Out-of-Kilter algorithm runs in time $O((m^2 + m \cdot n \cdot \log n) \cdot D)$ on a network with m edges and n vertices and maximum demand D.

Theorem 5 *The linear optimization problem over hexatopes can be solved in strongly polynomial time via reduction to the minimum cost flow problem.*

Proof Consider an n-dimensional hexatope $H = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ which is the image of a p-dimensional DCS-defined set. In order to optimize a linear function f over $[\![H]\!]$, it suffices to optimize the composition of functions $f \circ h$ where $h(\boldsymbol{x}) = G\boldsymbol{x} + \boldsymbol{c}$ over the difference constrained set $A\boldsymbol{x} \leq \boldsymbol{b}$. Suppose that $f(\boldsymbol{x}) = \boldsymbol{f}\boldsymbol{x} = \sum_k f_k x_k$, then the composite objective function can be written as $f \circ h(\boldsymbol{x}) = \boldsymbol{f}G\boldsymbol{x} + \boldsymbol{f}\boldsymbol{c}$. Then, omitting the constant term $\boldsymbol{f}\boldsymbol{c}$ which can be reincorporated in the end, we have that $f \circ h(\boldsymbol{x}) = \sum_k w_k x_k$ for some appropriate coefficients w_k.

Following chapter 24, Sect. 4 of [14], we construct a directed graph, called the constraint graph, to represent the difference constraint system determining the kernel of the given hexatope. For each variable x_i, there is a vertex v_i in the constraint graph. Additionally, there is one extra vertex v_0. Set the demand of each vertex as $d(v_i) = w_i$, for all $i > 0$, and let $d(v_0) = -\sum_k w_k$. For every constraint of the form $x_i - x_j \leq b$ in the DCS, there is an edge (v_j, v_i) in the constraint graph with cost b and infinite capacity. For every vertex v_i, with $i > 0$, there is also an edge (v_0, v_i) with cost 0 and infinite capacity.

The MCF problem instance constructed in this manner is equivalent to the dual of the given linear optimization problem instance over the given DCS. Since the Out-of-Kilter algorithm also solves the dual to the minimum cost flow problem [1], running it on the dual of the DCS optimization problem will also solve the DCS optimization problem itself. For a DCS with m constraints, this process takes $O((m^2 + m \cdot p \cdot \log p) \cdot C)$ time where C is the largest absolute value of any coefficient in the objective function. □

5 Octatopes

In this section, we introduce octatopes that generalize zonotopes, with the kernel being specified using a UTVPI constraint system. We will develop analogs of Theorems 1 to 3 for octatopes.

A UTVPI constraint $a_i \cdot x_i + a_j \cdot x_j \leq b$ is said to be an absolute constraint if $a_i = 0$ or $a_j = 0$. An absolute constraint can be converted into constraints of the form: $a_i \cdot x_i + a_j \cdot x_j \leq b$, where both a_i and a_j are non-zero. Note that a UTVPI constraint $a_i \cdot x_i + a_j \cdot x_j \leq b$ is a difference constraint if $a_i = -a_j$. The constant that bounds a UTVPI constraint is called the defining constant. For instance, the defining constant for the constraint $x_1 - x_2 \leq 9$ is 9.

Definition 9 *(Octatope)* An octatope is a special kind of linear star set $\langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$, having a kernel $A\boldsymbol{x} \leq \boldsymbol{b}$ defined by a UTVPI constraint system.

Some examples of octatopes are shown in Fig. 4 (right). Following the same argument used to prove Theorem 1, we establish closure of octatopes under affine mappings.

Theorem 6 *Octatopes are closed under affine transformation.*

By Theorem 2, linear optimization over linear star sets can be done in polynomial time. Our next result shows that linear optimization over octatopes and hexatopes can be done in *strongly* polynomial time.

Theorem 7 *The linear optimization problem for octatopes can be solved in strongly polynomial time via reduction to the linear optimization problem for hexatopes.*

Proof Following techniques of [31, 34], we convert a UCS **U** into a DCS **D**. The first part of the conversion creates the variables x_i^+ and x_i^- in **D** for each variable x_i in **U**. Then, each constraint in **U** is converted as follows:

1. Each constraint of the form $x_i + x_j \leq b$ becomes two constraints
$$x_i^+ - x_j^- \leq b \qquad \text{and} \qquad -x_i^- + x_j^+ \leq b.$$
2. Each constraint of the form $x_i - x_j \leq b$ becomes two constraints
$$x_i^+ - x_j^+ \leq b \qquad \text{and} \qquad -x_i^- + x_j^- \leq b.$$
3. Each constraint of the form $-x_i + x_j \leq b$ becomes two constraints
$$x_i^- - x_j^- \leq b \qquad \text{and} \qquad -x_i^+ + x_j^+ \leq b.$$
4. Each constraint of the form $-x_i - x_j \leq b$ becomes two constraints
$$x_i^- - x_j^+ \leq b \qquad \text{and} \qquad -x_i^+ + x_j^- \leq b.$$
5. Each constraint of the form $x_i \leq b$ becomes a constraint
$$x_i^+ - x_i^- \leq 2 \cdot b.$$
6. Each constraint of the form $-x_i \leq b$ becomes constraint
$$x_i^- - x_i^+ \leq 2 \cdot b.$$

Observe that that $x_i = \frac{1}{2}(x_i^+ - x_i^-)$ satisfies the original UCS. Thus, we can consider this as the problem maximizing the objective function over variables $\frac{1}{2}(x_i^+ - x_i^-)$ of the DCS **D**. □

5.1 Emptiness checking

We also consider the feasibility problem for octatopes. That is, the problem of deciding whether an octatope is empty.

Theorem 8 *The emptiness of an octatope can be decided in $O(p \cdot m)$ time and $O(p + m)$ space where p is the number of generator vectors and m is the number of* UTVPI *constraints defining its kernel.*

Proof It is easy to see that an octatope is empty if and only if the UTVPI constraints of its kernel are unsatisfiable as linear mappings over polytopes that are monotone with respect to set inclusion. The complexity then follows from results on checking the feasibility of UTVPI constraint systems [31]. □

5.2 Intersection with half-spaces

It follows from Theorem 3 that the intersection of an octatope $O = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ and half space $\{\boldsymbol{y} \mid M\boldsymbol{y} \leq \boldsymbol{d}\}$ is a star set $O' = \langle \boldsymbol{c}, G, A', \boldsymbol{b'} \rangle$ where the constraints $A'\boldsymbol{x} \leq \boldsymbol{b'}$ are the conjunction of UCS constraints $A\boldsymbol{x} \leq \boldsymbol{b}$ and the hyperplane $MG\boldsymbol{x} \leq \boldsymbol{d} - M\boldsymbol{c}$. In the rest of this section, we show how an over-approximation of this intersection can be represented as UCS constraints. The treatment for hexatopes is similar, and hence omitted.

Algorithm 5.1 UTVPIBoundingBox($\mathbf{U}, l$)

Input: UCS $\mathbf{U}$ and constraint l
Output: A UTVPI bounding box $\mathbf{U}'$

$\mathbf{U}' \leftarrow \emptyset$
for all pairs of variables x_i, x_j in $\mathbf{U}$ **do**
  Let $u_{ij}^{+-} = \max_{\mathbf{U} \cup \{l\}} x_i - x_j$ and add constraint $x_i - x_j \leq u_{ij}^{+-}$ to $\mathbf{U}'$
  Let $u_{ij}^{-+} = \max_{\mathbf{U} \cup \{l\}} x_j - x_i$ and add constraint $x_j - x_i \leq u_{ij}^{-+}$ to $\mathbf{U}'$
  Let $u_{ij}^{++} = \max_{\mathbf{U} \cup \{l\}} x_i + x_j$ and add constraint $x_i + x_j \leq u_{ij}^{++}$ to $\mathbf{U}'$
  Let $u_{ij}^{--} = \max_{\mathbf{U} \cup \{l\}} -x_i - x_j$ and add $-x_i - x_j \leq u_{ij}^{--}$ to $\mathbf{U}'$
  Let $u_i^{+} = \max_{\mathbf{U} \cup \{l\}} x_i$ and add constraint $x_i \leq u_i^{+}$ to $\mathbf{U}'$
  Let $u_i^{-} = \max_{\mathbf{U} \cup \{l\}} -x_i$ and add constraint $-x_i \leq u_i^{-}$ to $\mathbf{U}'$
return $\mathbf{U}'$.

We formalize this problem as the UTVPI *bounding box problem*.

Definition 10 (UTVPI *Bounding Box)* Given a UCS $\mathbf{U}$ and an arbitrary linear constraint l, a UTVPI *bounding box* is a UCS $\mathbf{U}'$, such that every solution to $\mathbf{U} \cup \{l\}$ is a solution to $\mathbf{U}'$. For a given UCS $\mathbf{U}$ and constraint l, a *tightest* UTVPI bounding box is a bounding box of $\mathbf{U} \cup \{l\}$ that is contained within every other bounding box of $\mathbf{U} \cup \{l\}$.

Thus, a UTVPI bounding box of a UCS $\mathbf{U}$ and constraint l is a UCS that overestimates the solution space of $\mathbf{U} \cup \{l\}$. A tightest bounding box is a UCS that overestimates the solution space the least. Each of the linear programs used to construct $\mathbf{U}'$ can be solved (with L bits of precision) in $O(n^{2.38} \cdot L)$ time [13]. Since finding the UTVPI bounding box requires solving $O(n^2)$ linear programs, the UTVPI bounding box can be found in $O(n^{4.38} \cdot L)$ time.

Theorem 9 *Let $\mathbf{U}$ be a UCS and let l be an arbitrary linear constraint. The UCS $\mathbf{U}'$, constructed by Algorithm 3, is a* UTVPI *bounding box of $\mathbf{U} \cup \{l\}$.*

Proof Let $\mathbf{x}^*$ be a solution to $\mathbf{U} \cup \{l\}$. Let $a_i \cdot x_i + a_j \cdot x_j \leq u_{ij}$ be an arbitrary constraint in $\mathbf{U}'$. By construction of $\mathbf{U}'$, we have $u_{ij} = \max_{\mathbf{U}\cup\{l\}} a_i \cdot x_i + a_j \cdot x_j$.

Since $\mathbf{x}^*$ is a solution of $\mathbf{U} \cup \{l\}$, $a_i \cdot x_i^* + a_j \cdot x_j^* \leq u_{ij}$. This means that $\mathbf{x}^*$ satisfies the constraint $a_i \cdot x_i + a_j \cdot x_j \leq u_{ij}$. Since the constraint $a_i \cdot x_i + a_j \cdot x_j \leq u_{ij}$ was chosen arbitrarily, $\mathbf{x}^*$ is a solution to $\mathbf{U}'$. Note that $\mathbf{x}^*$ was an arbitrary solution to $\mathbf{U} \cup \{l\}$. Thus, every solution to $\mathbf{U} \cup \{l\}$ is a solution to $\mathbf{U}'$. Consequently, $\mathbf{U}'$ is a UTVPI bounding box of $\mathbf{U} \cup \{l\}$. □

We now show that $\mathbf{U}'$ is a tightest UTVPI bounding box of $\mathbf{U} \cup \{l\}$. Note that $\mathbf{U} \cup \{l\}$ must have a tightest bounding box. Consider two bounding boxes $\mathbf{U}_1$ and $\mathbf{U}_2$ of $\mathbf{U} \cup \{l\}$. Let $\mathbf{U}^*$, be the UCS formed by combining the constraints in $\mathbf{U}_1$ and $\mathbf{U}_2$. Note that $\mathbf{U}^*$ is also a bounding box of $\mathbf{U} \cup \{l\}$. Additionally, every solution to $\mathbf{U}^*$ is a solution to both $\mathbf{U}_1$ and $\mathbf{U}_2$. Thus, if $\mathbf{U} \cup \{l\}$ has two incomparable bounding boxes, then a new bounding box can be constructed that is tighter than both.

Theorem 10 *Let* $\mathbf{U}$ *be a UCS and let* l *be a linear constraint. The UCS* $\mathbf{U}'$*, produced by Algorithm* 3*, is a tightest* UTVPI *bounding box of* $\mathbf{U} \cup \{l\}$.

Proof Assume for the sake of contradiction, that $\mathbf{U}'$ is not a tightest UTVPI bounding box of $\mathbf{U} \cup \{l\}$. Thus, there exist a UTVPI bounding box $\mathbf{U}''$ and a point $\mathbf{x}^*$ such that $\mathbf{x}^*$ is a solution to $\mathbf{U}'$, but not a solution to $\mathbf{U}''$. This means that there is a UTVPI constraint $a_i \cdot x_i + a_j \cdot x_j \leq b$ in $\mathbf{U}''$ that is violated by $\mathbf{x}^*$.

Let $u_{ij} = \max_{\mathbf{U}\cup\{l\}} a_i \cdot x_i + a_j \cdot x_j$. Since $\mathbf{U}''$ is a UTVPI bounding box of $\mathbf{U} \cup \{l\}$, every solution to $\mathbf{U} \cup \{l\}$ is a solution to $\mathbf{U}''$. Thus, every solution to $\mathbf{U} \cup \{l\}$ satisfies the constraint $a_i \cdot x_i + a_j \cdot x_j \leq b$. This means that

$$\max_{\mathbf{U}\cup\{l\}} a_i \cdot x_i + a_j \cdot x_j$$

is bounded from above by b. Thus, u_{ij} exists and $u_{ij} \leq b$.

By the construction of $\mathbf{U}'$, the constraint $a_i \cdot x_i + a_j \cdot x_j \leq u_{ij}$ is in $\mathbf{U}'$. However, $\mathbf{x}^*$ is a solution to $\mathbf{U}'$ such that $a_i \cdot x_i^* + a_j \cdot x_j^* > b \geq u_{ij}$. This is a contradiction. Thus, $\mathbf{U}'$ must be a tightest UTVPI bounding box of $\mathbf{U} \cup \{l\}$. □

5.3 Range computation for neural nets with prefilters

The exact range computation problem from Definition 2 can be solved using linear star sets (see Algorithms 1 and 2 in earlier work for a full review [7]).

The neural network function f as defined in Sect. 3 is a piece-wise affine function of the inputs. The range computation proceeds using geometric set operations. The initial set of states is represented as a linear star set and propagated through each layer of the network. To go from the output of one layer to the vector of intermediate values at the next layer, an affine transformation operation is performed on the set. The effect of the ReLU activation in a layer is handled iteratively for each neuron. The set of states is potentially split along the neuron input constraint $y_i = 0$, into a negative region and a positive region, using a half-space intersection operation. The negative region is then projected to zero to match the semantics of a ReLU. The two sets are then considered independently for the remaining neurons in the layer, as well as the rest of the layers in the network. For a given

input set, not all neurons require splitting the set in two, since the input constraints may restrict inputs to be strictly positive or negative. To check this, before splitting we first optimize over the set in the direction of the intermediate value $x_j^{(i)}$ corresponding to a specific neuron j in layer i. If splitting occurs, the two sets are treated independently and propagated through the remaining neurons in the layer, possibly requiring further splitting in the remaining parts of the network.

After applying a number of optimizations, the bottleneck of exact range computation with star sets is the use of LP solving to compute the input bounds for each neuron [7]. To improve analysis speed, rather than speeding up LP solving—which is a well-studied problem where further progress is likely to be difficult—we instead seek methods that can reduce the number of LPs needed.

In earlier work, zonotope abstract domains have been considered for this task. Rather than just propagating star sets through a network, we also propagate a zonotope overapproximation that we use in a *prefiltering* step. Recall that before splitting we first need to optimize over the set in the direction of the intermediate value $x_j^{(i)}$. Before optimizing over the star set using LP, we first optimize over the zonotope abstraction prefilter. If the zonotope abstraction can prove that that the inputs are strictly positive or negative, than we are guaranteed the exact result from the LP will be strictly positive or negative as well (as the zonotope is an overapproximation of the star set). This allows us to avoid LP, as optimization over zonotopes can be done efficiently using a simple loop.

The reason zonotope analysis is not exact is that zonotopes do not support general halfspace intersections when sets must be split. Instead, two approaches have been considered. The easiest option is to ignore intersections, which is fast but can cause significant overapproximation error in the abstraction [19, 43]. Alternatively, we can perform *domain contraction*, which is to search for zonotopes that more tightly overapproximate the intersection. Different approaches for domain contraction are possible, ranging from reasoning methods over individual constraints to more accurate approaches that use LP solving on the star set in the generator coefficient space [7]. Although the LP approach uses the expensive operation we are trying to reduce, it can result in an overall reduction of LPs, as the neuron input bounds can be computed more accurately.

This work proposes using octatope abstract domains as a prefilter. As described earlier, optimization over octatopes can be done more efficiently than general LP solving. The greater expressiveness of octatopes compared with zonotopes means that we can hope to further reduce the number of LPs needed with the star set when computing a neuron's input bounds for splitting. We evaluate this impact in our experiments. In terms of handling intersections when splitting sets, octatopes (like zonotopes) cannot exactly support

Table 1 Number of LP calls to find neuron input bounds for different abstract domain prefilters on various ACASXu properties and networks

Prop	Net	Star-Only	Zono-NC	Zono-C	Hex	Oct	Minimum
3	1–6	91,762	11,152	3382	2635	2571	1886
3	2–7	77,896	9365	2921	2240	2198	1626
3	3–5	80,988	8990	2711	2131	2092	1710
3	5–2	54,758	15,523	7762	6820	6704	3779
4	1–4	53,036	7736	2597	2389	2330	1926
4	2–7	38,748	3851	1249	888	861	753
4	5–9	68,750	8814	2952	2286	2151	1591

any general half-space intersection operation. This means that a domain contraction step may be necessary to ensure tight overapproximation.

5.4 Experimental results

We next evaluate the potential savings in LP computation to computing neuron input bounds during exact range computation for neural networks. Our evaluation is performed on several benchmarks from the ACAS Xu benchmark suite [26], specifically focusing on property 3 and 4 where earlier work has shown exact range computation is tractable [7]. We generally report number of LPs for different operations rather than runtime, as the runtime is influenced by other factors such code optimizations and the choice of LP solver.

First, we examine the number of LPs needed to perform neuron input range computation, for different choices of prefilter abstract domain. The LP calls to find the neuron input ranges is the bottleneck of the overall range computation algorithm, so its reduction is of particular importance. The results are in Table 1. The Star-Only approach uses only LP solving with no prefiler, and therefore has the highest number of LPs. The next column, Zonotope-NC corresponds to the case where zonotope prefilters are used, but no domain contraction is performed (halfspace intersections are ignored). This has a significant reduction on the number of LP calls, for example in the first row with property 3 and network 1–6, where the number of LP calls is reduced from 91K to 11K. Using domain contraction with zonotopes, Zono-C, further reduces this to around 3.3K. The more precise domains with hexatopes and octatopes can further reduce this to around 2.6K and 2.5K, respectively. The minimum column is computed by seeing how many bounds computations could not be eliminated as they correspond to cases where the input to a neuron truly can be either positive or negative. Even a perfect prefilter could not eliminate these LPs, as prefilters only eliminate cases where splitting is impossible. Other approaches could be considered to remove these LPs, such as tracking specific witness input points that can prove a neuron can have both positive and negative inputs, which we may consider in future work. Overall, the proposed octatope abstract domain has the potential to reduce the number of unnecessary LPs significantly in exact range computation.

When using the new abstract domain, however, there is a trade-off where extra operations are needed to perform domain contraction as well as to optimize within the abstract domains. We used a witness-tracking approach [4], where for each constraint a witness point was included that was in the star set and on the boundary of the constraint. When new intersections are performed, each witness point is checked to see if it is now excluded from the set. When points are excluded, new witness points get generated by solving an LP

Table 2 Number of LP calls for the domain contraction step for different abstract domain prefilters for various ACAS Xu properties and networks

Prop	Net	Star-Only	Zono-NC	Zono-C	Hex	Oct
3	1–6	0	0	12,765	38,400	115,200
3	2–7	0	0	12,280	36,840	110,520
3	3–5	0	0	10,407	31,230	93,690
3	5–2	0	0	21,249	63,750	191,250
4	1–4	0	0	11,828	35,493	106,476
4	2–7	0	0	5533	16,620	49,860
4	5–9	0	0	9906	29,730	89,190

in the direction of the constraint, which may tighten the constraint. This results in the tight abstract domains, but can be expensive when many constraints are possible. For hexatopes and octatopes, the number of possible constraints is quadratic in the number of variables (ACAS Xu has 5 input variables).

Table 2 shows the number of LPs needed for each example when performing domain contraction. Star-Only and Zono-NC do not perform domain contraction, and so have 0 LPs for this operation. As expected, the more complex the abstract domain, the more operations are needed. This is due to the contraction method performed, where the number of possible LPs needed at a domain contraction step increases as the number of possible constraints increases.

In terms of the performance of network simplex for optimizing within the octatope domain, the engineering aspect of the problem also requires further development. When computing the range of network 2–7 with the input set from property 4, the UTVPI constraints were optimized 38,748 times. When using the commercial LP solver Gurobi on these constraints, each call took on average of 0.17ms. Formulating the min-cost flow problem and calling the `network_simplex` implementation from the `networkx` python library, however, used about 1.9 ms per call, about 11x slower. Further, while Gurobi always obtained a result, numerical issues caused network simplex to fail about 0.65% of the time.

While octatopes effectively reduce the bottleneck step of input bounds computation, further improvements must be made to octatope domain contraction algorithms as well as to implementation optimizations of min-cost flow solvers, before an overall speedup can be achieved. Nonetheless, it is an encouraging result for DNN verification as we view this as an encouraging result to improving the performance of exact range computation of neural networks— developing more efficient domain contraction algorithms and improving min-cost flow implementations is likely easier than coming up with new ways to speed up LP solving.

Hexatopes versus octatopes: expressiveness and scalability From their definitions, it is clear that hexatopes are less expressive than octatopes, as every octatope is also a hexatope, but not the other way around. On the other hand, our experimental results (Table 1) show that for similar problems, the octatope abstract domain requires fewer LP calls compared to hexatopes. For example, in the first row with property 3 and network 1–6, the number of LP calls is reduced from 91K to around 2.6K and 2.5K for hexatopes and octatopes, respectively. However, note that the algorithms for octatopes work on UTVPI constraints, while hexatopes work with difference constraints. Since the algorithms for UTVPI constraints roughly double the size of the constraint graph, optimization over octatopes may take more time than for hexatopes on problems of similar size. However, in the absence of an efficient UTVPI solver implementation, an exact comparison of scalability between hexatopes and octatopes is difficult.

6 Conclusion

The advent of deep neural networks and their inevitable widespread adoption necessitates tools by which we can reason about their robustness. The verification community has made great strides on this front in recent years through the development of neural network verification solutions based on search, optimization, and reachability. While search and optimization can often be used to yield sound and complete solutions, such techniques pay the

cost of scalability. Methods based on reachability analysis, on the other, can often scale better at the cost of completeness. These methods typically employ an abstract domain representation of the input–output behavior of nodes in the neural network for a given set of inputs. These abstract domains range from zonotopes to star sets that differ in their trade-off between scalability and precision.

We proposed octatopes as a new abstract domain which corresponds to affine transformations of unit two-variable per inequality (UTVPI) constraints. Octatopes provide tighter abstractions than zonotopes while optimization can be formulated as a min-cost flow problem that is theoretically more efficient than linear programming. Our experiments using octatope abstract domains for exact range computation of neural networks confirmed their accuracy, as we were able to reduce the bottleneck step of using LP to compute each neuron's input bounds. Given that the minimum-cost flow (MCF) problem can be efficiently solved via the Out-of-Kilter algorithm and network simplex algorithms, this benefit is expected to extend to the efficiency of octatopes/hexatopes for neural network verification problems. Unfortunately, the current state-of-the-art implementations of the algorithms for the MCF problem are not as optimized as those for LP, preventing this work from experimentally demonstrating scalability gains in wall-clock time. In our experiments, we found that it was faster to use the highly-optimized commercial LP solver Gurobi instead of the theoretically faster min-cost flow formulation. A key takeaway from this research is the need for optimal implementations of UTVPI solvers and their potential applications in improving the state-of-the-art in neural network verification. Other future research directions include examining ways to improve domain contraction, as well as investigating other application areas of octatopes such as neural network verification with over-approximations, software analysis, and hybrid systems reachability.

Acknowledgement This material is based upon work supported by the Air Force Office of Scientific Research and the Office of Naval Research under award numbers FA9550-19-1-0288, FA9550-21-1-0121, FA9550-22-1-0450, FA9550-22-1-0029, N00014-22-1-2156, and FA9550-23-1-0066. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or the United States Navy. This research was supported in part by the Air Force Research Laboratory Information Directorate through the Air Force Office of Scientific Research Summer Faculty Fellowship Program, contract numbers FA8750-15-3-6003, FA9550-15-0001, and FA9550-20-F-0005. This work is also supported by the National Science Foundation (NSF) grant CCF-2009022 and by NSF CAREER awards CCF-2146563 and CNF-2237229.

Author contributions All authors contributed equally.

Data availability Data is available upon request.

Declarations

Conflict of interest We declare that we have no Conflict of interest.

Ethical approval This manuscript was created by following all the ethical guidelines. In particular, an extended abstract of this work was published in FM 2023. This has been indicated in the abstract as a footnote.

References

1. Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows: theory, algorithms, and applications. Prentice Hall, Upper Saddle River

2. Akintunde M, Lomuscio A, Maganti L, Pirovano E (2018). Reachability analysis for neural agent-environment systems. In: 16th international conference on principles of knowledge representation and reasoning
3. Albarghouthi A (2021) Introduction to neural network verification. http://verifieddeeplearning.com
4. Bak S (2021) nnenum: verification of ReLU neural networks with optimized abstraction refinement. In: NASA formal methods symposium, pp 19–36. Springer
5. Bak S, Dohmen T, Subramani K, Trivedi A, Velasquez A, Wojciechowski P (2023) The octatope abstract domain for verification of neural networks. In: Chechik M, Katoen J-P, Leucker M (eds), Formal methods—25th international symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings, volume 14000 of Lecture Notes in Computer Science, pp 454–472. Springer
6. Bak S, Liu C, Johnson T (2021) The second international verification of neural networks competition (VNN-comp 2021): summary and results. arXiv:2109.00498
7. Bak S, Tran H-D, Hobbs K, Johnson TT (2020) Improved geometric path enumeration for verifying Relu neural networks. In: Proceedings of the 32nd international conference on computer aided verification. Springer
8. Baluta T, Shen S, Shinde S, Meel KS, Saxena P (2019) Quantitative verification of neural networks and its security applications. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, pp 1249–1264
9. Bazaraa MS, Jarvis JJ, Sherali HD (2008) Linear programming and network flows. Wiley, New York
10. Behrmann G, David A, Larsen KG, Håkansson J, Pettersson P, Yi W, Hendriks M (2006) UPPAAL 4.0. In: 3rd international conference on the quantitative evaluation of systems (QEST 2006), 11-14 September 2006, Riverside, California, USA, pp 125–126. IEEE Computer Society
11. Biswas S, Rajan H (2023) Fairify: fairness verification of neural networks. In: 2023 IEEE/ACM 45th international conference on software engineering (ICSE), pp 1546–1558. IEEE
12. Casadio M, Komendantskaya E, Daggitt ML, Kokke W, Katz G, Amir G, Refaeli I (2022) Neural network robustness as a verification property: a principled case study. In: International conference on computer aided verification, pp 219–231. Springer
13. Cohen MB, Lee YT, Song Z (2021) Solving linear programs in the current matrix multiplication time. J ACM 68(1):3:1-3:39
14. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. MIT Press, Cambridge
15. Cousot P, Cousot R (1977) Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on principles of programming languages, POPL '77, pp 238–252, New York, NY, USA. Association for Computing Machinery
16. De Moura L, Bjørner N (2008). Z3: an efficient SMT solver. In: International conference on tools and algorithms for the construction and analysis of systems, pp 337–340. Springer
17. Duggirala PS, Viswanathan M (2016). Parsimonious, simulation based verification of linear systems. In: International conference on computer aided verification, pp 477–494. Springer
18. Friedmann O, Hansen TD, Zwick U (2011) Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In: Symposium on theory of computing, STOC'11, pp 283–292, New York, NY, USA. ACM
19. Gehr T, Mirman M, Drachsler-Cohen D, Tsankov P, Chaudhuri S, Vechev M (2018). Ai2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE symposium on security and privacy (SP), pp 3–18. IEEE
20. Ghorbal K, Goubault E, Putot S (2009). The zonotope abstract domain Taylor1+. In: International conference on computer aided verification, pp 627–633. Springer
21. Goldberg AV, Tarjan RE (1989) Finding minimum-cost circulations by canceling negative cycles. J ACM 36(4):873–886
22. Henriksen P, Lomuscio A (2020). Efficient neural network verification via adaptive refinement and adversarial search. In: ECAI 2020, pp 2513–2520. IOS Press
23. Henriksen P, Lomuscio A (2021). Deepsplit: an efficient splitting method for neural network verification via indirect effect analysis. In: Proceedings of the 30th international joint conference on artificial intelligence (IJCAI21), To appear
24. Huang X, Kroening D, Ruan W, Sharp J, Sun Y, Thamo E, Wu M, Yi X (2020) A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability. Comput Sci Rev 37:100270
25. Huang X, Kwiatkowska M, Wang S, Wu M (2017) Safety verification of deep neural networks. In: Computer aided verification: 29th international conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30, pp 3–29. Springer

26. Katz G, Barrett C, Dill DL, Julian K, Kochenderfer MJ (2017) Reluplex: an efficient SMT solver for verifying deep neural networks. In: International conference on computer aided verification, pp 97–117. Springer
27. Katz G, Huang DA, Ibeling D, Julian K, Lazarus C, Lim R, Shah P, Thakoor S, Wu H, Al Z, Dill DL, Kochenderfer MJ, Barrett C (2019) The marabou framework for verification and analysis of deep neural networks. In: Dillig I, Serdar T (eds) Comput Aided Verif. Springer, Cham, pp 443–452
28. Khachiyan LG (1979), A polynomial time algorithm for linear programming. Doklady Akademii Nauk SSSR, 244(5), 1093–1096, English translation in Soviet Math. Dokl. 20:191–194
29. Klee F, Minty GJ (1972) How good is the simplex algorithm? Inequalities III:159–175
30. Kuutti S, Bowden R, Jin Y, Barber P, Fallah S (2020) A survey of deep learning applications to autonomous vehicle control. IEEE Trans Intell Transp Syst 22(2):712–733
31. Lahiri SK, Musuvathi M (2005) An efficient decision procedure for UTVPI constraints. In: Gramlich B (ed) Frontiers of combining systems. Springer, Berlin, pp 168–183
32. Liu C, Arnon T, Lazarus C, Barrett C, Kochenderfer MJ (2019). Algorithms for verifying deep neural networks. arXiv:1903.06758
33. Manzanas Lopez D, Johnson T, Tran H-D, Bak S, Chen X, Hobbs KL (2021) Verification of neural network compression of ACAS Xu lookup tables with star set reachability. In: AIAA Scitech 2021 Forum, p 0995
34. Miné A (2006) The octagon abstract domain. Higher-order Symb Comput 19(1):31–100
35. Orlin JB (1996) A polynomial time primal network simplex algorithm for minimum cost flows. In: Proceedings of the 7th annual ACM-SIAM symposium on discrete algorithms, SODA '96, 474-481, USA. Society for Industrial and Applied Mathematics
36. Singh G, Gehr T, Mirman M, Püschel M, Vechev MT (2018) Fast and effective robustness certification. NeurIPS 1(4):6
37. Singh G, Gehr T, Püschel M, Vechev M (2019) An abstract domain for certifying neural networks. Proc ACM Program Lang 3(POPL):1–30
38. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction, 2nd edn. MIT Press, Cambridge
39. Tjeng V, Xiao KY, Tedrake R (2018) Evaluating robustness of neural networks with mixed integer programming. In: International conference on learning representations
40. Tran H-D, Bak S, Xiang W, Johnson TT (2020). Verification of deep convolutional neural networks using imagestars. In: International conference on computer aided verification, pp 18–42. Springer
41. Tran H-D, Cai F, Diego ML, Musau P, Johnson TT, Koutsoukos X (2019) Safety verification of cyber-physical systems with reinforcement learning control. ACM Trans Embed Comput Syst (TECS) 18(5s):1–22
42. Tran H-D, Manzanas Lopez D, Musau P, Yang X, Nguyen LV, Xiang W, Johnson TT (2019) Star-based reachability analysis of deep neural networks. In: ter Beek MH, McIver A, Oliveira JN (eds) Formal methods—the next 30 years. Springer, Cham, pp 670–686
43. Tran H-D, Pal N, Musau P, Lopez DM, Hamilton N, Yang X, Bak S, Johnson TT (2021) Robustness verification of semantic segmentation neural networks using relaxed reachability. In: International conference on computer aided verification, pp 263–286. Springer
44. Tran H-D, Yang X, Lopez DM, Musau P, Nguyen LV, Xiang W, Bak S, Johnson TT (2020) NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: International conference on computer aided verification, pp 3–17. Springer
45. Wang S, Pei K, Whitehouse J, Yang J, Jana S (2018) Efficient formal safety analysis of neural networks. In: Advances in neural information processing systems, vol 31
46. Wang S, Zhang H, Xu K, Lin X, Jana S, Hsieh C-J, Kolter JZ (2021) Beta-crown: efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. arXiv:2103.06624

Authors and Affiliations

Stanley Bak[1] · Taylor Dohmen[2] · K. Subramani[3] · Ashutosh Trivedi[2] · Alvaro Velasquez[2] · Piotr Wojciechowski[3]

✉ K. Subramani
k.subramani@mail.wvu.edu

Stanley Bak
stanley.bak@stonybrook.edu

Taylor Dohmen
taylor.dohmen@colorado.edu

Ashutosh Trivedi
ashutosh.trivedi@colorado.edu

Alvaro Velasquez
alvaro.velasquez@colorado.edu

Piotr Wojciechowski
pwojciec@mail.wvu.edu

[1] Stony Brook University, Stony Brook, USA

[2] University of Colorado Boulder, Boulder, USA

[3] West Virginia University, Morgantown, USA