Open- and Closed-Loop Neural Network Verification Using Polynomial Zonotopes

Niklas Kochdumper[1(✉)], Christian Schilling[2], Matthias Althoff[3], and Stanley Bak[1]

[1] Stony Brook University, Stony Brook, NY, USA
{niklas.kochdumper,stanley.bak}@stonybrook.edu
[2] Aalborg University, Aalborg, Denmark
christianms@cs.aau.dk
[3] Technical University of Munich, Garching, Germany
althoff@tum.de

Abstract. We present a novel approach to efficiently compute tight non-convex enclosures of the image through neural networks with ReLU, sigmoid, or hyperbolic tangent activation functions. In particular, we abstract the input-output relation of each neuron by a polynomial approximation, which is evaluated in a set-based manner using polynomial zonotopes. While our approach can also can be beneficial for open-loop neural network verification, our main application is reachability analysis of neural network controlled systems, where polynomial zonotopes are able to capture the non-convexity caused by the neural network as well as the system dynamics. This results in a superior performance compared to other methods, as we demonstrate on various benchmarks.

Keywords: Neural network verification · Neural network controlled systems · Reachability analysis · Polynomial zonotopes · Formal verification

1 Introduction

While previously artificial intelligence was mainly used for soft applications such as movie recommendations [9], facial recognition [23], or chess computers [11], it is now also increasingly applied in safety-critical applications, such as autonomous driving [32], human-robot collaboration [27], or power system control [5]. In contrast to soft applications, where failures usually only have minor consequences, failures in safety-critical applications in the worst case result in loss of human lives. Consequently, in order to prevent those failures, there is an urgent need for efficient methods that can verify that the neural networks used for artificial intelligence function correctly. Verification problems involving neural networks can be grouped into two main categories:

– **Open-loop verification:** Here the task is to check if the output of the neural network for a given input set satisfies certain properties. With this setup one can for example prove that a neural network used for image classification is robust against a certain amount of noise on the image.

K. Y. Rozier and S. Chaudhuri (Eds.): NFM 2023, LNCS 13903, pp. 16–36, 2023.
https://doi.org/10.1007/978-3-031-33170-1_2

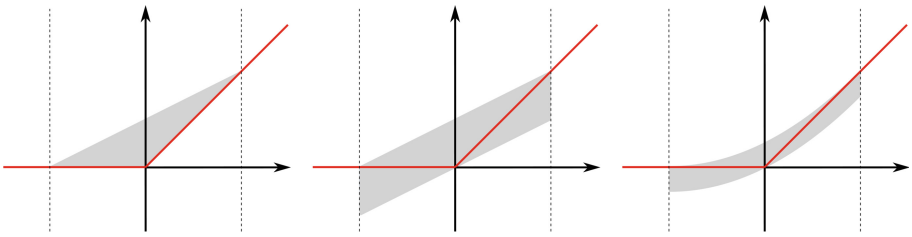

Fig. 1. Triangle relaxation (left), zonotope abstraction (middle), and polynomial zonotope abstraction (right) of the ReLU activation function.

– **Closed-loop verification:** In this case the neural network is used as a controller for a dynamical system, e.g., to steer the system to a given goal set while avoiding unsafe regions. The safety of the controlled system can be verified using reachability analysis.

For both of the above verification problems, the most challenging step is to compute a tight enclosure of the image through the neural network for a given input set. Due to the high expressiveness of neural networks, their images usually have complex shapes, so that convex enclosures are often too conservative for verification. In this work, we show how to overcome this limitation with our novel approach for computing tight non-convex enclosures of images through neural networks using polynomial zonotopes.

1.1 State of the Art

We first summarize the state of the art for open-loop neural network verification followed by reachability analysis for neural network controlled systems. Many different set representations have been proposed for computing enclosures of the image through a neural network, including intervals [43], polytopes [38], zonotopes [34], star sets [40], and Taylor models [21]. For neural networks with ReLU activation functions, it is possible to compute the exact image. This can be either achieved by recursively partitioning the input set into piecewise affine regions [42], or by propagating the initial set through the network using polytopes [38,48] or star sets [40], where the set is split at all neurons that are both active or inactive. In either case the exact image is in the worst case given as a union of 2^v convex sets, with v being the number of neurons in the network. To avoid this high computational complexity for exact image computation, most approaches compute a tight enclosure instead using an abstraction of the neural network. For ReLU activation functions one commonly used abstraction is the triangle relaxation [15] (see Fig. 1), which can be conveniently integrated into set propagation using star sets [40]. Another possibility is to abstract the input-output relation by a zonotope (see Fig. 1), which is possible for ReLU, sigmoid, and hyperbolic tangent activation functions [34]. One can also apply Taylor model arithmetic [26] to compute the image through networks with sigmoid and hyperbolic tangent activation [21], which corresponds to an abstraction of the input-output relation by a Taylor series expansion. In order to better

capture dependencies between different neurons, some approaches also abstract the input-output relation of multiple neurons at once [28,36].

While computation of the exact image is infeasible for large networks, the enclosures obtained by abstractions are often too conservative for verification. To obtain complete verifiers, many approaches therefore use branch and bound strategies [7] that split the input set and/or single neurons until the specification can either be proven or a counterexample is found. For computational reasons branch and bound strategies are usually combined with approaches that are able to compute rough interval bounds for the neural network output very fast. Those bounds can for example be obtained using symbolic intervals [43] that store linear constraints on the variables in addition to the interval bounds to preserve dependencies. The DeepPoly approach [35] uses a similar concept, but applies a back-substitution scheme to obtain tighter bounds. With the FastLin method [45] linear bounds for the overall network can be computed from linear bounds for the single neurons. The CROWN approach [49] extends this concept to linear bounds with different slopes as well as quadratic bounds. Several additional improvements for the CROWN approach have been proposed, including slope optimization using gradient descent [47] and efficient ReLU splitting [44]. Instead of explicitly computing the image, many approaches also aim to verify the specification directly using SMT solvers [22,30], mixed-integer linear programming [8,37], semidefinite programming [31], and convex optimization [24].

For reachability analysis of neural network controlled systems one has to compute the set of control inputs in each control cycle, which is the image of the current reachable set through the neural network controller. Early approaches compute the image for ReLU networks exactly using polytopes [46] or star sets [39]. Since in this case the number of coexisting sets grows rapidly over time, these approaches have to unite sets using convex hulls [46] or interval enclosures [39], which often results in large over-approximations. If template polyhedra are used as a set representation, reachability analysis for neural network controlled systems with discrete-time plants reduces to the task of computing the maximum output along the template directions [12], which can be done efficiently. Neural network controllers with sigmoid and hyperbolic tangent activation functions can be converted to an equivalent hybrid automaton [20], which can be combined with the dynamics of the plant using the automaton product. However, since each neuron is represented by an additional state, the resulting hybrid automaton is very high-dimensional, which makes reachability analysis challenging. Some approaches approximate the overall network with a polynomial function [14,18] using polynomial regression based on samples [14] and Bernstein polynomials [18]. Yet another class of methods [10,21,33,41] employs abstractions of the input-output relation for the neurons to compute the set of control inputs using intervals [10], star sets [41], Taylor models [21], and a combination of zonotopes and Taylor models [33]. Common tools for reachability analysis of neural network controlled systems are JuliaReach [6], NNV [41], POLAR [19], ReachNN* [16], RINO [17], Sherlock [13], Verisig [20], and Verisig 2.0 [21], where JuliaReach uses zonotopes for neural network abstraction [33], NVV supports multiple set representations, ReachNN* applies the Bernstein polynomial

method [18], POLAR approximates single neurons by Bernstein polynomials [19], RINO computes interval inner- and outer-approximations [17], Sherlock uses the polynomial regression approach [14], Verisig performs the conversion to a hybrid automaton [20], and Verisig 2.0 uses the Taylor model based neural network abstraction method [21].

1.2 Overview

In this work, we present a novel approach for computing tight non-convex enclosures of images through neural networks with ReLU, sigmoid, or hyperbolic tangent activation functions. The high-level idea is to approximate the input-output relation of each neuron by a polynomial function, which results in the abstraction visualized in Fig. 1. Since polynomial zonotopes are closed under polynomial maps, the image through this function can be computed exactly, yielding a tight enclosure of the image through the overall neural network. The remainder of this paper is structured as follows: After introducing some preliminaries in Sect. 2, we present our approach for computing tight enclosures of images through neural networks in Sect. 3. Next, we show how to utilize this result for reachability analysis of neural network controlled systems in Sect. 4. Afterwards, in Sect. 5, we introduce some special operations on polynomial zonotopes that we require for image and reachable set computation, before we finally demonstrate the performance of our approach on numerical examples in Sect. 6.

1.3 Notation

Sets are denoted by calligraphic letters, matrices by uppercase letters, and vectors by lowercase letters. Given a vector $b \in \mathbb{R}^n$, $b_{(i)}$ refers to the i-th entry. Given a matrix $A \in \mathbb{R}^{o \times n}$, $A_{(i,\cdot)}$ represents the i-th matrix row, $A_{(\cdot,j)}$ the j-th column, and $A_{(i,j)}$ the j-th entry of matrix row i. The concatenation of two matrices C and D is denoted by $[C\ D]$, and $I_n \in \mathbb{R}^{n \times n}$ is the identity matrix. The symbols $\mathbf{0}$ and $\mathbf{1}$ represent matrices of zeros and ones of proper dimension, the empty matrix is denoted by $[\,]$, and $\text{diag}(a)$ returns a diagonal matrix with $a \in \mathbb{R}^n$ on the diagonal. Given a function $f(x)$ defined as $f : \mathbb{R} \to \mathbb{R}$, $f'(x)$ and $f''(x)$ denote the first and second derivative with respect to x. The left multiplication of a matrix $A \in \mathbb{R}^{o \times n}$ with a set $\mathcal{S} \subset \mathbb{R}^n$ is defined as $A\,\mathcal{S} := \{A\,s \mid s \in \mathcal{S}\}$, the Minkowski addition of two sets $\mathcal{S}_1 \subset \mathbb{R}^n$ and $\mathcal{S}_2 \subset \mathbb{R}^n$ is defined as $\mathcal{S}_1 \oplus \mathcal{S}_2 := \{s_1 + s_2 \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\}$, and the Cartesian product of two sets $\mathcal{S}_1 \subset \mathbb{R}^n$ and $\mathcal{S}_2 \subset \mathbb{R}^o$ is defined as $\mathcal{S}_1 \times \mathcal{S}_2 := \left\{[s_1^T\ s_2^T]^T \mid s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\right\}$. We further introduce an n-dimensional interval as $\mathcal{I} := [l, u],\ \forall i\, l_{(i)} \leq u_{(i)},\ l, u \in \mathbb{R}^n$.

2 Preliminaries

Let us first introduce some preliminaries required throughout the paper. While the concepts presented in this work can equally be applied to more complex network architectures, we focus on feed-forward neural networks for simplicity:

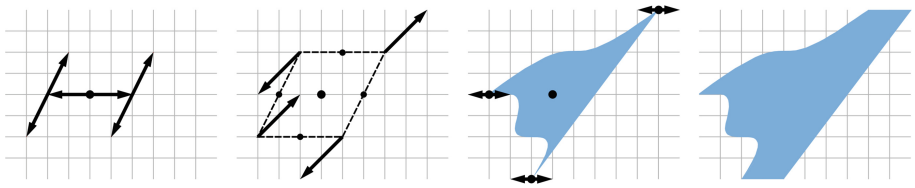

Fig. 2. Step-by-step construction of the polynomial zonotope from Example 1.

Definition 1. *(Feed-forward neural network) A feed-forward neural network with κ hidden layers consists of weight matrices $W_i \in \mathbb{R}^{v_i \times v_{i-1}}$ and bias vectors $b_i \in \mathbb{R}^{v_i}$ with $i \in \{1, \dots, \kappa+1\}$ and v_i denoting the number of neurons in layer i. The output $y \in \mathbb{R}^{v_{\kappa+1}}$ of the neural network for the input $x \in \mathbb{R}^{v_0}$ is*

$$y := y_{\kappa+1} \quad \text{with} \quad y_0 = x, \quad y_{i(j)} = \mu\Big(\sum_{k=1}^{v_{i-1}} W_{i(j,k)}\, y_{i-1(k)} + b_{i(j)} \Big), \quad i = 1, \dots, \kappa+1,$$

where $\mu : \mathbb{R} \to \mathbb{R}$ is the activation function.

In this paper we consider ReLU activations $\mu(x) = \max(0, x)$, sigmoid activations $\mu(x) = \sigma(x) = 1/(1+e^{-x})$, and hyperbolic tangent activations $\mu(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. Moreover, neural networks often do not apply activation functions on the output neurons, which corresponds to using the identity map $\mu(x) = x$ for the last layer. The image $\mathcal{Y}$ through a neural network is defined as the set of outputs for a given set of inputs $\mathcal{X}_0$, which is according to Def. 1 given as

$$\mathcal{Y} = \left\{ y_{\kappa+1} \;\middle|\; y_0 \in \mathcal{X}_0,\ \forall i \in \{1, \dots, \kappa+1\} :\ y_{i(j)} = \mu\Big(\sum_{k=1}^{v_{i-1}} W_{i(j,k)}\, y_{i-1(k)} + b_{i(j)} \Big) \right\}.$$

We present a novel approach for tightly enclosing the image through a neural network by a polynomial zonotope [2], where we use the sparse representation of polynomial zonotopes [25][1]:

Definition 2. *(Polynomial zonotope) Given a constant offset $c \in \mathbb{R}^n$, a generator matrix of dependent generators $G \in \mathbb{R}^{n \times h}$, a generator matrix of independent generators $G_I \in \mathbb{R}^{n \times q}$, and an exponent matrix $E \in \mathbb{N}_0^{p \times h}$, a polynomial zonotope $\mathcal{PZ} \subset \mathbb{R}^n$ is defined as*

$$\mathcal{PZ} := \left\{ c + \sum_{i=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) G_{(\cdot,i)} + \sum_{j=1}^{q} \beta_j G_{I(\cdot,j)} \;\middle|\; \alpha_k, \beta_j \in [-1, 1] \right\}.$$

The scalars α_k are called dependent factors since a change in their value affects multiplication with multiple generators. Analogously, the scalars β_j are called independent factors because they only affect the multiplication with one generator. For a concise notation we use the shorthand $\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ}$.

[1] In contrast to [25, Def. 1], we explicitly do not integrate the constant offset c in G. Moreover, we omit the identifier vector used in [25] for simplicity.

Let us demonstrate polynomial zonotopes by an example:

Example 1. The polynomial zonotope

$$\mathcal{PZ} = \left\langle \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \begin{bmatrix} 2 & 1 & 2 \\ 0 & 2 & 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \end{bmatrix} \right\rangle_{PZ}$$

defines the set

$$\mathcal{PZ} = \left\{ \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} \alpha_1 + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \alpha_2 + \begin{bmatrix} 2 \\ 2 \end{bmatrix} \alpha_1^3 \alpha_2 + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \beta_1 \;\middle|\; \alpha_1, \alpha_2, \beta_1 \in [-1, 1] \right\}.$$

The construction of this polynomial zonotope is visualized in Fig. 2.

3 Image Enclosure

We now present our novel approach for computing tight non-convex enclosures of images through neural networks. The general concept is to approximate the input-output relation of each neuron by a polynomial function, the image through which can be computed exactly since polynomial zonotopes are closed under polynomial maps. For simplicity, we focus on quadratic approximations here, but the extension to polynomials of higher degree is straightforward.

The overall procedure for computing the image is summarized in Algorithm 1, where the computation proceeds layer by layer. For each neuron in the current layer i we first calculate the corresponding input set in Line 5. Next, in Line 6, we compute a lower and an upper bound for the input to the neuron. Using these bounds we then calculate a quadratic approximation for the neuron's input-output relation in Line 7. This approximation is evaluated in a set-based manner in Line 8. The resulting polynomial zonotope $\langle c_q, G_q, G_{I,q}, E_q \rangle_{PZ}$ forms the j-th dimension of the set $\mathcal{PZ}$ representing the output of the whole layer (see Line 9 and Line 12). To obtain a formally correct enclosure, we have to account for the error made by the approximation. We therefore compute the difference between the activation function and the quadratic approximation in Line 10 and add the result to the output set in Line 12. By repeating this procedure for all layers, we finally obtain a tight enclosure of the image through the neural network. A demonstrating example for Algorithm 1 is shown in Fig. 3.

For ReLU activations the quadratic approximation only needs to be calculated if $l < 0 \wedge u > 0$ since we can use the exact input-output relations $g(x) = x$ and $g(x) = 0$ if $l \geq 0$ or $u \leq 0$ holds. Due to the evaluation of the quadratic map defined by $g(x)$, the representation size of the polynomial zonotope $\mathcal{PZ}$ increases in each layer. For deep neural networks it is therefore advisable to repeatedly reduce the representation size after each layer using order reduction [25, Prop. 16]. Moreover, one can also apply the `compact` operation described in [25, Prop. 2] after each layer to remove potential redundancies from $\mathcal{PZ}$. Next, we explain the approximation of the input-output relation as well as the computation of the approximation error in detail.

Algorithm 1. Enclosure of the image through a neural network

Require: Neural network with weight matrices W_i and bias vectors b_i, initial set $\mathcal{X}_0$.
Ensure: Tight enclosure $\mathcal{PZ} \supseteq \mathcal{Y}$ of the image $\mathcal{Y}$.

1: $\mathcal{PZ} \leftarrow \mathcal{X}_0$
2: **for** $i \leftarrow 1$ to $\kappa + 1$ **do** (loop over all layers)
3: $c \leftarrow \mathbf{0},\ G \leftarrow \mathbf{0},\ G_I \leftarrow \mathbf{0},\ \underline{d} \leftarrow \mathbf{0},\ \overline{d} \leftarrow \mathbf{0}$
4: **for** $j \leftarrow 1$ to v_i **do** (loop over all neurons in the layer)
5: $\mathcal{PZ}_j \leftarrow W_{i(j,\cdot)}\mathcal{PZ} \oplus b_{i(j)}$ (map with weight matrix and bias using (5))
6: $l, u \leftarrow$ lower and upper bound for $\mathcal{PZ}_j$ according to Prop. 1
7: $g(x) = a_1\, x^2 + a_2\, x + a_3 \leftarrow$ quad. approx. on $[l, u]$ according to Sect. 3.1
8: $\langle c_q, G_q, G_{I,q}, E_q \rangle_{PZ} \leftarrow$ image of $\mathcal{PZ}_j$ through $g(x)$ according to Prop. 2
9: $c_{(j)} \leftarrow c_q,\ G_{(j,\cdot)} \leftarrow G_q,\ G_{I(j,\cdot)} \leftarrow G_{I,q},\ E \leftarrow E_q$ (add to output set)
10: $\underline{d}_{(j)}, \overline{d}_{(j)} \leftarrow$ difference between $g(x)$ and activation function acc. to Sect. 3.2
11: **end for**
12: $\mathcal{PZ} \leftarrow \langle c, G, G_I, E \rangle_{PZ} \oplus [\underline{d}, \overline{d}]$ (add approximation error using (6))
13: **end for**

3.1 Activation Function Approximation

The centerpiece of our algorithm for computing the image of a neural network is the approximation of the input-output relation defined by the activation function $\mu(x)$ with a quadratic expression $g(x) = a_1\, x^2 + a_2\, x + a_3$ (see Line 7 of Algorithm 1). In this section we present multiple possibilities to obtain good approximations.

Polynomial Regression

For polynomial regression we uniformly select N samples x_i from the interval $[l, u]$ and then determine the polynomial coefficients a_1, a_2, a_3 by minimizing the average squared distance between the activation function and the quadratic approximation:

$$\min_{a_1,a_2,a_3} \frac{1}{N} \sum_{i=1}^{N} \big(\mu(x_i) - a_1\, x_i^2 - a_2\, x_i - a_3\big)^2. \tag{1}$$

It is well known that the optimal solution to (1) is

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = A^{\dagger} b \quad \text{with} \quad A = \begin{bmatrix} x_1^2 & x_1 & 1 \\ \vdots & \vdots & \vdots \\ x_N^2 & x_N & 1 \end{bmatrix}, \quad b = \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_N) \end{bmatrix},$$

where $A^{\dagger} = (A^T A)^{-1} A^T$ is the Moore-Penrose inverse of matrix A. For the numerical experiments in this paper we use $N = 10$ samples.

Closed-Form Expression

For ReLU activations a closed-form expression for a quadratic approximation can be obtained by enforcing the conditions $g(l) = 0$, $g'(l) = 0$, and $g(u) = u$. The

solution to the corresponding equation system $a_1\,l^2+a_2\,l+a_3=0$, $2a_1l+a_2=0$, $a_1\,u^2+a_2\,u+a_3=u$ is

$$a_1=\frac{u}{(u-l)^2},\quad a_2=\frac{-2lu}{(u-l)^2},\quad a_3=\frac{u^2(2l-u)}{(u-l)^2}+u,$$

which results in the enclosure visualized in Fig. 1. This closed-form expression is very precise if the interval $[l,u]$ is close to being symmetric with respect to the origin ($|l|\approx|u|$), but becomes less accurate if one bound is significantly larger than the other ($|u|\gg|l|$ or $|l|\gg|u|$).

Taylor Series Expansion

For sigmoid and hyperbolic tangent activation functions a quadratic fit can be obtained using a second-order Taylor series expansion of the activation function $\mu(x)$ at the expansion point $x^*=0.5(l+u)$:

$$\mu(x)\approx\mu(x^*)+\mu'(x^*)(x-x^*)+0.5\,\mu''(x^*)(x-x^*)^2=$$
$$\underbrace{0.5\,\mu''(x^*)}_{a_1}x^2+\big(\underbrace{\mu'(x^*)-\mu''(x^*)\,x^*}_{a_2}\big)x+\underbrace{\mu(x^*)-\mu'(x^*)x^*+0.5\,\mu''(x^*)\,x^{*2}}_{a_3},$$

where the derivatives for sigmoid activations are $\mu'(x)=\sigma(x)(1-\sigma(x))$ and $\mu''(x)=\sigma(x)(1-\sigma(x))(1-2\sigma(x))$, and the derivatives for hyperbolic tangent activations are $\mu'(x)=1-\tanh(x)^2$ and $\mu''(x)=-2\tanh(x)(1-\tanh(x)^2)$. The Taylor series expansion method is identical to the concept used in [21].

Linear Approximation

Since a linear function represents a special case of a quadratic function, Algorithm 1 can also be used in combination with linear approximations. Such approximations are provided by the zonotope abstraction in [34]. Since closed-form expressions for the bounds $\underline{d}$ and $\overline{d}$ of the approximation error are already specified in [34], we can omit the error bound computation described in Sect. 3.2 in this case. For ReLU activations we obtain according to [34, Theorem 3.1]

$$a_1=0,\quad a_2=\frac{u}{u-l},\quad a_3=\frac{-u\,l}{2(u-l)},\quad \underline{d}=\frac{-u\,l}{2(u-l)},\quad \overline{d}=\frac{u\,l}{2(u-l)},$$

which results in the zonotope enclosure visualized in Fig. 1. For sigmoid and hyperbolic tangent activations we obtain according to [34, Theorem 3.2]

$$a_1=0,\quad a_2=\min(\mu'(l),\mu'(u)),\quad a_3=0.5(\mu(u)+\mu(l)-a_2(u+l)),$$
$$\underline{d}=0.5(\mu(u)-\mu(l)-a_2(u-l)),\quad \overline{d}=-0.5(\mu(u)-\mu(l)-a_2(u-l)),$$

where the derivatives of the sigmoid function and the hyperbolic tangent are specified in the paragraph above.

We observed from experiments that for ReLU activations the closed-form expression usually results in a tighter enclosure of the image than polynomial regression. For sigmoid and hyperbolic tangent activations, on the other hand,

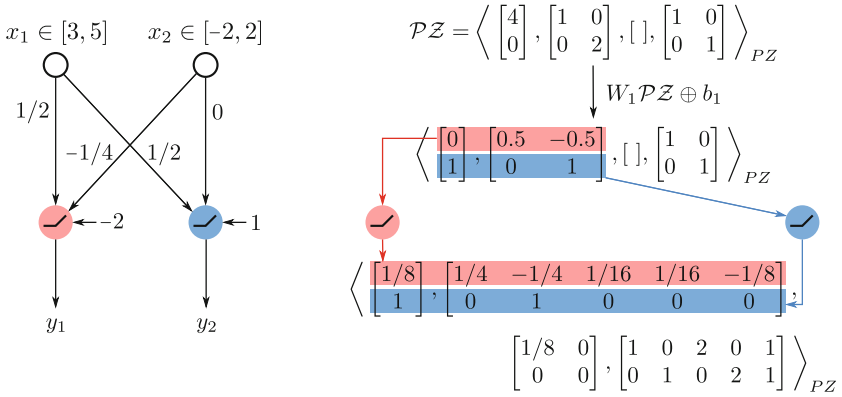

Fig. 3. Exemplary neural network with ReLU activations (left) and the corresponding image enclosure computed with polynomial zonotopes (right), where we use the approximation $g(x) = 0.25\,x^2 + 0.5\,x + 0.25$ for the red neuron and the approximation $g(x) = x$ for the blue neuron. (Color figure online)

polynomial regression usually performs better than the Taylor series expansion. It is also possible to combine multiple of the methods described above by executing them in parallel and selecting the one that results in the smallest approximation error $[\underline{d}, \overline{d}]$. Since the linear approximation does not increase the number of generators, it represents an alternative to order reduction when dealing with deep neural networks. Here, the development of a method to decide automatically for which layers to use a linear and for which a quadratic approximation is a promising direction for future research.

3.2 Bounding the Approximation Error

To obtain a sound enclosure we need to compute the difference between the activation function $\mu(x)$ and the quadratic approximation $g(x) = a_1\,x^2 + a_2\,x + a_3$ from Sec. 3.1 on the interval $[l, u]$. In particular, this corresponds to determining

$$\underline{d} = \min_{x \in [l,u]} \underbrace{\mu(x) - a_1\,x^2 - a_2\,x - a_3}_{d(x)} \quad \text{and} \quad \overline{d} = \max_{x \in [l,u]} \underbrace{\mu(x) - a_1\,x^2 - a_2\,x - a_3}_{d(x)}.$$

Depending on the type of activation function, we use different methods for this.

Rectified Linear Unit (ReLU)

For ReLU activation functions we split the interval $[l, u]$ into the two intervals $[l, 0]$ and $[0, u]$ on which the activation function is constant and linear, respectively. On the interval $[l, 0]$ we have $d(x) = -a_1\,x^2 - a_2\,x - a_3$, and on the interval $[0, u]$ we have $d(x) = -a_1\,x^2 + (1 - a_2)\,x - a_3$. In both cases $d(x)$ is a quadratic function whose maximum and minimum values are either located on the interval boundary or at the point x^* where the derivative of $d(x)$ is equal

to zero. The lower bound on $[l, 0]$ is therefore given as $\underline{d} = \min(d(l), d(x^*), d(0))$ if $x^* \in [l, 0]$ and $\underline{d} = \min(d(l), d(0))$ if $x^* \notin [l, 0]$, where $x^* = -0.5\, a_2/a_1$. The upper bound as well as the bounds for $[0, u]$ are computed in a similar manner. Finally, the overall bounds are obtained by taking the minimum and maximum of the bounds for the intervals $[l, 0]$ and $[0, u]$.

Sigmoid and Hyperbolic Tangent

Here our high-level idea is to sample the function $d(x)$ at points x_i with distance Δx distributed uniformly over the interval $[l, u]$. From rough bounds for the derivative $d'(x)$ we can then deduce how much the function value between two sample points changes at most, which yields tight bounds $\overline{d}_b \geq \overline{d}$ and $\underline{d}_b \leq \underline{d}$. In particular, we want to choose the sampling rate Δx such that the bounds $\overline{d}_b, \underline{d}_b$ comply to a user-defined precision $\delta > 0$:

$$\overline{d} + \delta \geq \overline{d}_b \geq \overline{d} \quad \text{and} \quad \underline{d} - \delta \leq \underline{d}_b \leq \underline{d}. \tag{2}$$

We observe that for both, sigmoid and hyperbolic tangent, the derivative is globally bounded by $\mu'(x) \in [0, \overline{\mu}]$, where $\overline{\mu} = 0.25$ for the sigmoid and $\overline{\mu} = 1$ for the hyperbolic tangent. In addition, it holds that the derivative of the quadratic approximation $g(x) = a_1\, x^2 + a_2\, x + a_3$ is bounded by $g'(x) \in [\underline{g}, \overline{g}]$ on the interval $[l, u]$, where $\underline{g} = \min(2a_1 l + a_2, 2a_1 u + a_2)$ and $\overline{g} = \max(2a_1 l + a_2, 2a_1 u + a_2)$. As a consequence, the derivative of the difference $d(x) = \mu(x) - g(x)$ is bounded by $d'(x) \in [-\overline{g}, \overline{\mu} - \underline{g}]$. The value of $d(x)$ can therefore at most change by $\pm \Delta d$ between two samples x_i and x_{i+1}, where $\Delta d = \Delta x \max(|-\overline{g}|, |\overline{\mu} - \overline{g}|)$. To satisfy (2) we require $\Delta d \leq \delta$, so that we have to choose the sampling rate as $\Delta x \leq \delta / \max(|-\overline{g}|, |\overline{\mu} - \overline{g}|)$. Finally, the bounds are computed by taking the maximum and minimum of all samples: $\overline{d}_b = \max_i d(x_i) + \delta$ and $\underline{d}_b = \min_i d(x_i) - \delta$. For our experiments we use a precision of $\delta = 0.001$.

4 Neural Network Controlled Systems

Reachable sets for neural network controlled systems can be computed efficiently by combining our novel image enclosure approach for neural networks with a reachability algorithm for nonlinear systems. We consider general nonlinear systems

$$\dot{x}(t) = f\big(x(t), u_c(x(t), t), w(t)\big), \tag{3}$$

where $x \in \mathbb{R}^n$ is the system state, $u_c : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$ is a control law, $w(t) \in \mathcal{W} \subset \mathbb{R}^r$ is a vector of uncertain disturbances, and $f : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r \to \mathbb{R}^n$ is a Lipschitz continuous function. For neural network controlled systems the control law $u_c(x(t), t)$ is given by a neural network. Since neural network controllers are usually realized as digital controllers, we consider the sampled-data case where the control input is only updated at discrete times $t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots, t_F$ and kept constant in between. Here, t_0 is the initial time, t_F is the final time, and Δt is the sampling rate. Without loss of generality, we assume from now on that $t_0 = 0$ and t_F is a multiple of Δt. The reachable set is defined as follows:

Algorithm 2. Reachable set for a neural network controlled system

Require: Nonlinear system $\dot{x}(t) = f(x(t), u_c(x(t), t), w(t))$, neural network controller $u_c(x(t), t)$, initial set $\mathcal{X}_0$, disturbance set $\mathcal{W}$, final time t_F, sampling rate Δt.
Ensure: Tight enclosure $\mathcal{R} \supseteq \mathcal{R}([0, t_F])$ of the reachable set $\mathcal{R}([0, t_F])$.

1: $t_0 \leftarrow 0$, $\mathcal{R}(t_0) \leftarrow \mathcal{X}_0$
2: **for** $i \leftarrow 0$ to $t_F/\Delta t - 1$ **do** (loop over all control cycles)
3: $\mathcal{Y} \leftarrow$ image of $\mathcal{R}(t_i)$ through the neural network controller using Algorithm 1
4: $\widehat{\mathcal{R}}(t_i) \leftarrow \mathcal{R}(t_i) \times \mathcal{Y}$ (combine reachable set and input set using (7))
5: $t_{i+1} \leftarrow t_i + \Delta t$, $\tau_i \leftarrow [t_i, t_{i+1}]$ (update time)
6: $\widehat{\mathcal{R}}(t_{i+1}), \widehat{\mathcal{R}}(\tau_i) \leftarrow$ reachable set for extended system in (4) starting from $\widehat{\mathcal{R}}(t_i)$
7: $\mathcal{R}(t_{i+1}) \leftarrow [I_n \ \mathbf{0}]\, \widehat{\mathcal{R}}(t_{i+1})$, $\mathcal{R}(\tau_i) \leftarrow [I_n \ \mathbf{0}]\, \widehat{\mathcal{R}}(\tau_i)$ (projection using (5))
8: **end for**
9: $\mathcal{R} \leftarrow \bigcup_{i=0}^{t_F/\Delta t - 1} \mathcal{R}(\tau_i)$ (reachable set for the whole time horizon)

Definition 3. *(Reachable set) Let $\xi(t, x_0, u_c(\cdot), w(\cdot))$ denote the solution to (3) for initial state $x_0 = x(0)$, control law $u_c(\cdot)$, and the disturbance trajectory $w(\cdot)$. The reachable set for an initial set $\mathcal{X}_0 \subset \mathbb{R}^n$ and a disturbance set $\mathcal{W} \subset \mathbb{R}^r$ is*

$$\mathcal{R}(t) := \left\{ \xi(t, x_0, u_c(\cdot), w(\cdot)) \;\middle|\; x_0 \in \mathcal{X}_0, \forall t^* \in [0, t] : \; w(t^*) \in \mathcal{W} \right\}.$$

Since the exact reachable set cannot be computed for general nonlinear systems, we compute a tight enclosure instead. We exploit that the control input is piecewise constant, so that the reachable set for each control cycle can be computed using the extended system

$$\begin{bmatrix} \dot{x}(t) \\ \dot{u}(t) \end{bmatrix} = \begin{bmatrix} f(x(t), u(t), w(t)) \\ \mathbf{0} \end{bmatrix} \tag{4}$$

together with the initial set $\mathcal{X}_0 \times \mathcal{Y}$, where $\mathcal{Y}$ is the image of $\mathcal{X}_0$ through the neural network controller. The overall algorithm is specified in Algorithm 2. Its high-level concept is to loop over all control cycles, where in each cycle we first compute the image of the current reachable set through the neural network controller in Line 3. Next, the image is combined with the reachable set using the Cartesian product in Line 4. This yields the initial set for the extended system in (4), for which we compute the reachable set $\widehat{\mathcal{R}}(t_{i+1})$ at time t_{i+1} as well as the reachable set $\widehat{\mathcal{R}}(\tau_i)$ for the time interval τ_i in Line 6. While it is possible to use arbitrary reachability algorithms for nonlinear systems, we apply the conservative polynomialization algorithm [2] since it performs especially well in combination with polynomial zonotopes. Finally, in Line 7, we project the reachable set back to the original system dimensions.

5 Operations on Polynomial Zonotopes

Algorithm 1 and Algorithm 2 both require some special operations on polynomial zonotopes, the implementation of which we present now. Given a polynomial

zonotope $\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ} \subset \mathbb{R}^n$, a matrix $A \in \mathbb{R}^{o \times n}$, a vector $b \in \mathbb{R}^o$, and an interval $\mathcal{I} = [l, u] \subset \mathbb{R}^n$, the affine map and the Minkowski sum with an interval are given as

$$A\,\mathcal{PZ} \oplus b = \langle Ac + b, AG, AG_I, E \rangle_{PZ} \tag{5}$$

$$\mathcal{PZ} \oplus \mathcal{I} = \langle c + 0.5(u + l), G, [G_I \; 0.5\,\mathrm{diag}(u - l)], E \rangle_{PZ}, \tag{6}$$

which follows directly from [25, Prop. 8], [25, Prop. 9], and [1, Prop. 2.1]. For the Cartesian product used in Line 4 of Algorithm 2 we can exploit the special structure of the sets to calculate the Cartesian product of two polynomial zonotopes $\mathcal{PZ}_1 = \langle c_1, G_1, G_{I,1}, E_1 \rangle_{PZ} \subset \mathbb{R}^n$ and $\mathcal{PZ}_2 = \langle c_2, [G_2 \; \widehat{G}_2], [G_{I,2} \; \widehat{G}_{I,2}], [E_1 \; E_2] \rangle_{PZ} \subset \mathbb{R}^o$ as

$$\mathcal{PZ}_1 \times \mathcal{PZ}_2 = \left\langle \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \begin{bmatrix} G_1 & \mathbf{0} \\ G_2 & \widehat{G}_2 \end{bmatrix}, \begin{bmatrix} G_{I,1} & \mathbf{0} \\ G_{I,2} & \widehat{G}_{I,2} \end{bmatrix}, [E_1 \; E_2] \right\rangle_{PZ}. \tag{7}$$

In contrast to [25, Prop. 11], this implementation of the Cartesian product explicitly preserves dependencies between the two sets, which is possible since both polynomial zonotopes have identical dependent factors. Computing the exact bounds of a polynomial zonotope in Line 6 of Algorithm 1 would be computationally infeasible, especially since this has to be done for each neuron in the network. We therefore compute a tight enclosure of the bounds instead, which can be done very efficiently:

Proposition 1. *(Interval enclosure) Given a polynomial zonotope* $\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ} \subset \mathbb{R}^n$*, an enclosing interval can be computed as*

$$\mathcal{I} = [c + g_1 - g_2 - g_3 - g_4, c + g_1 + g_2 + g_3 + g_4] \supseteq \mathcal{PZ}$$

with

$$g_1 = 0.5 \sum_{i \in \mathcal{H}} G_{(\cdot,i)}, \;\; g_2 = 0.5 \sum_{i \in \mathcal{H}} |G_{(\cdot,i)}|, \;\; g_3 = \sum_{i \in \mathcal{K}} |G_{(\cdot,i)}|, \;\; g_4 = \sum_{i=1}^{q} |G_{I(\cdot,i)}|$$

$$\mathcal{H} = \left\{ i \;\middle|\; \prod_{j=1}^{p} \left(1 - E_{(j,i)} \bmod 2\right) = 1 \right\}, \quad \mathcal{K} = \{1, \ldots, h\} \setminus \mathcal{H},$$

where $x \bmod y$*,* $x, y \in \mathbb{N}_0$ *is the modulo operation and* $\setminus$ *denotes the set difference.*

Proof 1. We first enclose the polynomial zonotope by a zonotope $\mathcal{Z} \supseteq \mathcal{PZ}$ according to [25, Prop. 5], and then compute an interval enclosure $\mathcal{I} \supseteq \mathcal{Z}$ of this zonotope according to [1, Prop. 2.2]. □

The core operation for Algorithm 1 is the computation of the image through a quadratic function. While it is possible to obtain the exact image by introducing new dependent factors, we compute a tight enclosure for computational reasons:

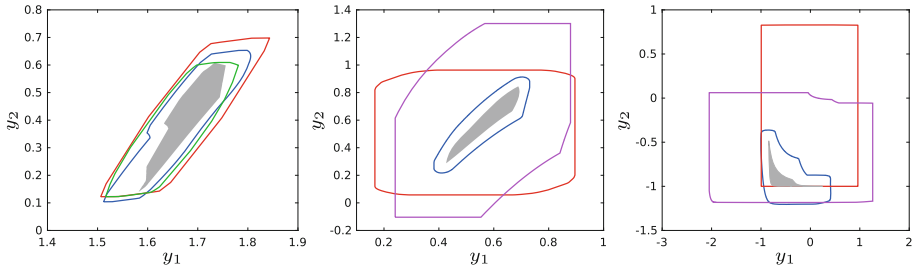

Fig. 4. Image enclosures computed with zonotopes (red), star sets (green), Taylor models (purple), and polynomial zonotopes (blue) for randomly generated neural networks with ReLU activations (left), sigmoid activations (middle), and hyperbolic tangent activations (right). The exact image is shown in gray. (Color figure online)

Proposition 2. *(Image quadratic function) Given a polynomial zonotope* $\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ} \subset \mathbb{R}$ *and a quadratic function* $g(x) = a_1\, x^2 + a_2\, x + a_3$ *with* $a_1, a_2, a_3, x \in \mathbb{R}$*, the image of* $\mathcal{PZ}$ *through* $g(x)$ *can be tightly enclosed by*

$$\{g(x) \mid x \in \mathcal{PZ}\} \subseteq \langle c_q, G_q, G_{I,q}, E_q \rangle_{PZ}$$

with

$$\begin{aligned} c_q &= a_1 c^2 + a_2 c + a_3 + 0.5\, a_1 \sum_{i=1}^{q} G_{I(\cdot,i)}^2, \quad G_q = \begin{bmatrix} (2a_1 c + a_2) G & a_1 \widehat{G} \end{bmatrix}, \\ E_q &= \begin{bmatrix} E & \widehat{E} \end{bmatrix}, \quad G_{I,q} = \begin{bmatrix} (2a_1 c + a_2) G_I & 2a_1 \overline{G} & a_1 \check{G} \end{bmatrix}, \end{aligned} \tag{8}$$

where

$$\begin{aligned} \widehat{G} &= \begin{bmatrix} G^2 & 2\,\widehat{G}_1 & \dots & 2\,\widehat{G}_{h-1} \end{bmatrix}, \quad \widehat{E} = \begin{bmatrix} 2\,E & \widehat{E}_1 & \dots & \widehat{E}_{h-1} \end{bmatrix}, \\ \widehat{G}_i &= \begin{bmatrix} G_{(i)} G_{(i+1)} & \dots & G_{(i)} G_{(h)} \end{bmatrix}, \quad i = 1, \dots, h-1, \\ \widehat{E}_i &= \begin{bmatrix} E_{(\cdot,i)} + E_{(\cdot,i+1)} & \dots & E_{(\cdot,i)} + E_{(\cdot,h)} \end{bmatrix}, \quad i = 1, \dots, h-1, \\ \overline{G} &= \begin{bmatrix} G_{(1)} G_I & \dots & G_{(h)} G_I \end{bmatrix}, \quad \check{G} = \begin{bmatrix} 0.5\, G_I^2 & 2\,\check{G}_1 & \dots & 2\,\check{G}_{q-1} \end{bmatrix}, \\ \check{G}_i &= \begin{bmatrix} G_{I(i)} G_{I(i+1)} & \dots & G_{I(i)} G_{I(q)} \end{bmatrix}, \quad i = 1, \dots, q-1, \end{aligned} \tag{9}$$

and the squares in G^2 *as well as* G_I^2 *are interpreted elementwise.*

Proof 2. The proof is provided in Appendix A.

6 Numerical Examples

We now demonstrate the performance of our approach for image computation, open-loop neural network verification, and reachability analysis of neural network controlled systems. If not stated otherwise, computations are carried out in

MATLAB on a 2.9GHz quad-core i7 processor with 32GB memory. We integrated our implementation into CORA [3] and published a repeatability package[2].

Image Enclosure

First, we demonstrate how our approach captures the non-convexity of the image through a neural network. For visualization purposes we use the deliberately simple example of randomly generated neural networks with two inputs, two outputs, and one hidden layer consisting of 50 neurons. The initial set is $\mathcal{X}_0 = [-1, 1] \times [-1, 1]$. We compare our polynomial-zonotope-based approach with the zonotope abstraction in [34], the star set approach in [40] using the triangle relaxation, and the Taylor model abstraction in [21]. While our approach and the zonotope abstraction are applicable to all types of activation functions, the star set approach is restricted to ReLU activations and the Taylor model abstraction is limited to sigmoid and hyperbolic tangent activations. The resulting image enclosures are visualized in Fig. 4. While using zonotopes or star sets only yields a convex over-approximation, polynomial zonotopes are able to capture the non-convexity of the image and therefore provide a tighter enclosure. While Taylor models also capture the non-convexity of the image to some extent they are less precise than polynomial zonotopes, which can be explained as follows: 1) The zonotopic remainder of polynomial zonotopes prevents the rapid remainder growth observed for Taylor models, and 2) the quadratic approximation obtained with polynomial regression used for polynomial zonotopes is usually more precise than the Taylor series expansion used for Taylor models.

Open-Loop Neural Network Verification

For open-loop neural network verification the task is to verify that the image of the neural network satisfies certain specifications that are typically given by linear inequality constraints. We examine the ACAS Xu benchmark from the 2021 and 2022 VNN competition [4,29] originally proposed in [22, Sec. 5], which features neural networks that provide turn advisories for an aircraft to avoid collisions. All networks consist of 6 hidden layers with 50 ReLU neurons per layer. For a fair comparison we performed the evaluation on the same machine that was used for the VNN competition. To compute the image through the neural networks with polynomial zonotopes, we apply a quadratic approximation obtained by polynomial regression for the first two layers, and a linear approximation in the remaining layers. Moreover, we recursively split the initial set to obtain a complete verifier. The comparison with the other tools that participated in the VNN competition shown in Table 1 demonstrates that for some verification problems polynomial zonotopes are about as fast as the best tool in the competition.

Neural Network Controlled Systems

The main application of our approach is reachability analysis of neural network controlled systems, for which we now compare the performance to other state-of-the-art tools. For a fair comparison we focus on published results for which the

[2] https://codeocean.com/capsule/8237552/tree/v1.

Table 1. Computation times[a] in seconds for different verification tools on a small but representative excerpt of network-specification combinations of the ACAS Xu benchmark. The symbol - indicates that the tool failed to verify the specification.

Net.	Spec.	Cgdtest	CROWN	Debona	ERAN	Marabou	MN-BaB	nnenum	nnv	NV.jl	oval	RPM	venus2	VeriNet	Poly. zono.
1.9	1	0.37	1.37	111	3.91	0.66	48.7	0.41	-	1.44	0.71	-	0.53	0.55	**0.31**
2.3	4	-	0.95	1.78	1.91	0.57	12.2	**0.06**	-	-	0.97	-	0.46	0.17	0.16
3.5	3	0.41	0.37	1.15	1.85	0.61	6.17	**0.05**	-	-	0.58	34.1	0.42	0.25	0.32
4.5	4	-	0.35	0.20	1.82	0.61	5.57	**0.08**	0.24	-	0.48	-	0.42	0.21	0.16
5.6	3	0.38	0.63	2.27	1.82	0.66	6.51	**0.08**	-	-	0.52	40.6	0.48	0.37	0.43

[a] Times taken from https://github.com/stanleybak/vnncomp2021_results and https://github.com/ChristopherBrix/vnncomp2022_results.

Table 2. Computation times[b] in seconds for reachability analysis of neural network controlled systems considering different tools and benchmarks. The dimension, the number of hidden layers, and the number of neurons in each layer is specified in parenthesis for each benchmark, where $a = 100$, $b = 5$ for ReLU activation functions, and $a = 20$, $b = 3$ otherwise. The symbol - indicates that the tool failed to verify the specification.

	ReLU			**sigmoid**					**hyp. tangent**				
	Sherlock	JuliaReach	Poly. zono.	Verisig	Verisig 2.0	ReachNN*	POLAR	Poly. zono.	Verisig	Verisig 2.0	ReachNN*	POLAR	Poly. zono.
B1 $(2, 2, 20)$				-	49	69	23	**2**	-	48	-	25	**8**
B2 $(2, 2, 20)$				12	8	32	10	**1**	-	-	-	**3**	-
B3 $(2, 2, 20)$				98	47	130	37	**3**	98	43	128	38	**3**
B4 $(3, 2, 20)$				24	12	20	4	**1**	23	11	20	4	**1**
B5 $(3, 3, 100)$				196	1063	31	25	**2**	-	168	-	31	**2**
TORA $(4, 3, a)$	30	2040	**13**	136	83	13402		**1**	134	70	2524		**1**
ACC $(6, b, 20)$	4	**1**	2						-	1512	-	312	**2**
Unicycle $(3, 1, 500)$	526	93	**3**										
Airplane $(12, 3, 100)$	-	29	**7**										
Sin. Pend. $(2, 2, 25)$	1	1	**1**										

[b] Computation times taken from [33, Table 1] for Sherlock and JuliaReach, from [21, Table 2] for Verisig, Verisig 2.0, and ReachNN*, and from [19, Tab. 1] for POLAR.

authors of the tools tuned the algorithm settings by themselves. In particular, we examine the benchmarks from [33] featuring ReLU neural network controllers, and the benchmarks from [21] containing sigmoid and hyperbolic tangent neural network controllers. The goal for all benchmarks is to verify that the system reaches a goal set or avoids an unsafe region. As the computation times shown

in Table 2 demonstrate, our polynomial-zonotope-based approach is for all but two benchmarks significantly faster than all other state-of-the-art tools, mainly since it avoids all major bottlenecks observed for the other tools: The polynomial approximations of the overall network used by Sherlock and ReachNN* are often imprecise, JuliaReach loses dependencies when enclosing Taylor models by zonotopes, Verisig is quite slow since the nonlinear system used to represent the neural network is high-dimensional, and Verisig 2.0 and POLAR suffer from the rapid remainder growth observed for Taylor models.

7 Conclusion

We introduced a novel approach for computing tight non-convex enclosures of images through neural networks with ReLU, sigmoid, and hyperbolic tangent activation functions. Since we represent sets with polynomial zonotopes, all required calculations can be realized using simple matrix operations only, which makes our algorithm very efficient. While our proposed approach can also be applied to open-loop neural network verification, its main application is reachability analysis of neural network controlled systems. There, polynomial zonotopes enable the preservation of dependencies between the reachable set and the set of control inputs, which results in very tight enclosures of the reachable set. As we demonstrated on various numerical examples, our polynomial-zonotope-based approach consequently outperforms all other state-of-the-art methods for reachability analysis of neural network controlled systems.

Acknowledgements. We gratefully acknowledge the financial support from the project justITSELF funded by the European Research Council (ERC) under grant agreement No 817629, from DIREC - Digital Research Centre Denmark, and from the Villum Investigator Grant S4OS. In addition, 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, FA9550-23-1-0066 and N00014-22-1-2156. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force or the United States Navy.

Appendix A

We now provide the proof for Prop. 2. According to Def. 2, the one-dimensional polynomial zonotope $\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ}$ is defined as

$$\begin{aligned} \mathcal{PZ} &= \left\{ c + \underbrace{\sum_{i=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) G_{(i)}}_{d(\alpha)} + \underbrace{\sum_{j=1}^{q} \beta_j G_{I(j)}}_{z(\beta)} \;\middle|\; \alpha_k, \beta_j \in [-1, 1] \right\} \\ &= \left\{ c + d(\alpha) + z(\beta) \;\middle|\; \alpha, \beta \in [-\mathbf{1}, \mathbf{1}] \right\}, \end{aligned} \tag{10}$$

where $\alpha = [\alpha_1 \ \dots \ \alpha_p]^T$ and $\beta = [\beta_1 \ \dots \ \beta_q]^T$. To compute the image through the quadratic function $g(x)$ we require the expressions $d(\alpha)^2$, $d(\alpha)z(\beta)$, and $z(\beta)^2$, which we derive first. For $d(\alpha)^2$ we obtain

$$\begin{aligned} d(\alpha)^2 &= \left(\sum_{i=1}^{h}\left(\prod_{k=1}^{p}\alpha_k^{E_{(k,i)}}\right)G_{(i)}\right)\left(\sum_{j=1}^{h}\left(\prod_{k=1}^{p}\alpha_k^{E_{(k,j)}}\right)G_{(j)}\right) \\ &= \sum_{i=1}^{h}\sum_{j=1}^{h}\left(\prod_{k=1}^{p}\alpha_k^{E_{(k,i)}+E_{(k,j)}}\right)G_{(i)}G_{(j)} \\ &= \sum_{i=1}^{h}\left(\prod_{k=1}^{p}\alpha_k^{2E_{(k,i)}}\right)G_{(i)}^2 + \sum_{i=1}^{h-1}\sum_{j=i+1}^{h}\left(\prod_{k=1}^{p}\underbrace{\alpha_k^{E_{(k,i)}+E_{(k,j)}}}_{\alpha_k^{\widehat{E}_{i(k,j)}}}\right)2\underbrace{G_{(i)}G_{(j)}}_{\widehat{G}_{i(j)}} \\ &\overset{(9)}{=} \sum_{i=1}^{h(h+1)/2}\left(\prod_{k=1}^{p}\alpha_k^{\widehat{E}_{(k,i)}}\right)\widehat{G}_{(i)}, \end{aligned} \tag{11}$$

for $d(\alpha)z(\beta)$ we obtain

$$\begin{aligned} d(\alpha)z(\beta) &= \left(\sum_{i=1}^{h}\left(\prod_{k=1}^{p}\alpha_k^{E_{(k,i)}}\right)G_{(i)}\right)\left(\sum_{j=1}^{q}\beta_j G_{I(j)}\right) \\ &= \sum_{i=1}^{h}\sum_{j=1}^{q}\underbrace{\left(\beta_j\prod_{k=1}^{p}\alpha_k^{E_{(k,i)}}\right)}_{\beta_{q+(i-1)h+j}}G_{(i)}G_{I(j)} \overset{(9)}{=} \sum_{i=1}^{hq}\beta_{q+i}\,\overline{G}_{(i)}, \end{aligned} \tag{12}$$

and for $z(\beta)^2$ we obtain

$$\begin{aligned} z(\beta)^2 &= \left(\sum_{i=1}^{q}\beta_i G_{I(i)}\right)\left(\sum_{j=1}^{q}\beta_j G_{I(j)}\right) = \sum_{i=1}^{q}\sum_{j=1}^{q}\beta_i\beta_j\, G_{I(i)}G_{I(j)} \\ &= \sum_{i=1}^{q}\beta_i^2 G_{I(i)}^2 + \sum_{i=1}^{q-1}\sum_{j=i+1}^{q}\beta_i\beta_j\, 2\, G_{I(i)}G_{I(j)} \\ &= 0.5\sum_{i=1}^{q}G_{I(i)}^2 + \sum_{i=1}^{q}\underbrace{(2\beta_i^2-1)}_{\beta_{(h+1)q+i}}0.5\,G_{I(i)}^2 + \sum_{i=1}^{q-1}\sum_{j=i+1}^{q}\underbrace{\beta_i\beta_j}_{\beta_{a(i,j)}}\,2\,\underbrace{G_{I(i)}G_{I(j)}}_{\check{G}_{i(j)}} \\ &\overset{(9)}{=} 0.5\sum_{i=1}^{q}G_{I(i)}^2 + \sum_{i=1}^{q(q+1)/2}\beta_{(h+1)q+i}\,\check{G}_{(i)}, \end{aligned} \tag{13}$$

where the function $a(i,j)$ maps indices i, j to a new index:

$$a(i,j) = (h+2)q + j - i + \sum_{k=1}^{i-1} q - k.$$

In (12) and (13), we substituted the expressions $\beta_j \prod_{k=1}^{p} \alpha_k^{E_{(k,i)}}$, $2\beta_i^2 - 1$, and $\beta_i \beta_j$ containing polynomial terms of the independent factors β by new independent factors, which results in an enclosure due to the loss of dependency. The substitution is possible since

$$\beta_j \prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \in [-1,1], \quad 2\beta_i^2 - 1 \in [-1,1], \quad \text{and} \quad \beta_i \beta_j \in [-1,1].$$

Finally, we obtain for the image

$$\{g(x) \mid x \in \mathcal{PZ}\} = \{a_1 x^2 + a_2 x + a_3 \mid x \in \mathcal{PZ}\}$$

$$\overset{(10)}{=} \{a_1(c + d(\alpha) + z(\beta))^2 + a_2(c + d(\alpha) + z(\beta)) + a_3 \mid \alpha, \beta \in [-\mathbf{1}, \mathbf{1}]\}$$

$$= \{a_1 c^2 + a_2 c + a_3 + (2a_1 c + a_2) d(\alpha) + a_1 d(\alpha)^2 + (2a_1 c + a_2) z(\beta) + 2a_1 d(\alpha) z(\beta) + a_1 z(\beta)^2 \mid \alpha, \beta \in [-\mathbf{1}, \mathbf{1}]\}$$

$$\overset{(11),(12),(13)}{\subseteq} \left\langle a_1 c^2 + a_2 c + a_3 + 0.5\, a_1 \sum_{i=1}^{q} G_I^2, \left[(2a_1 c + a_2) G \;\; a_1 \widehat{G}\right], \left[(2a_1 c + a_2) G_I \;\; 2a_1 \overline{G} \;\; a_1 \check{G}\right], \left[E \;\; \widehat{E}\right] \right\rangle_{PZ} \overset{(8)}{=} \langle c_q, G_q, G_{I,q}, E_q \rangle_{PZ},$$

which concludes the proof.

References

1. Althoff, M.: Reachability analysis and its application to the safety assessment of autonomous cars. Ph.D. thesis, Technical University of Munich (2010)
2. Althoff, M.: Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In: Proceedings of the International Conference on Hybrid Systems: Computation and Control, pp. 173–182 (2013)
3. Althoff, M.: An introduction to CORA 2015. In: Proceedings of the International Workshop on Applied Verification for Continuous and Hybrid Systems, pp. 120–151 (2015)
4. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. arXiv:2109.00498 (2021)
5. Beaufays, F., Abdel-Magid, Y., Widrow, B.: Application of neural networks to load-frequency control in power systems. Neural Netw. **7**(1), 183–194 (1994)
6. Bogomolov, S., et al.: JuliaReach: a toolbox for set-based reachability. In: Proceedings of the International Conference on Hybrid Systems: Computation and Control, pp. 39–44 (2019)
7. Bunel, R., et al.: Branch and bound for piecewise linear neural network verification. J. Mach. Learn. Res. **21**(42) (2020)
8. Cheng, C.H., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: Proceedings of the International Symposium on Automated Technology for Verification and Analysis, pp. 251–268 (2017)

9. Christakou, C., Vrettos, S., Stafylopatis, A.: A hybrid movie recommender system based on neural networks. Int. J. Artif. Intell. Tools **16**(5), 771–792 (2007)
10. Clavière, A., et al.: Safety verification of neural network controlled systems. In: Proceedings of the International Conference on Dependable Systems and Networks, pp. 47–54 (2021)
11. David, O.E., Netanyahu, N.S., Wolf, L.: DeepChess: end-to-end deep neural network for automatic learning in chess. In: Proceedings of the International Conference on Artificial Neural Networks, pp. 88–96 (2016)
12. Dutta, S., et al.: Learning and verification of feedback control systems using feedforward neural networks. In: Proceedings of the International Conference on Analysis and Design of Hybrid Systems, pp. 151–156 (2018)
13. Dutta, S., et al.: Sherlock-a tool for verification of neural network feedback systems. In: Proceedings of the International Conference on Hybrid Systems: Computation and Control, pp. 262–263 (2019)
14. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Proceedings of the International Conference on Hybrid Systems: Computation and Control, pp. 157–168 (2019)
15. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Proceedings of the International Symposium on Automated Technology for Verification and Analysis, pp. 269–286 (2017)
16. Fan, J., Huang, et al.: ReachNN*: a tool for reachability analysis of neural-network controlled systems. In: Proceedings of the International Symposium on Automated Technology for Verification and Analysis, pp. 537–542 (2020)
17. Goubault, E., Putot, S.: RINO: robust inner and outer approximated reachability of neural networks controlled systems. In: Proceedings of the International Conference on Computer Aided Verification, pp. 511–523 (2022)
18. Huang, C., et al.: ReachNN: reachability analysis of neural-network controlled systems. Trans. Embed. Comput. Syst. **18**(5s) (2019)
19. Huang, C., et al.: POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems. In: Proceedings of the International Symposium on Automated Technology for Verification and Analysis, pp. 414–430 (2022)
20. Ivanov, R., et al.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Proceedings of the International Conference on Hybrid Systems: Computation and Control, pp. 169–178 (2019)
21. Ivanov, R., et al.: Verisig 2.0: verification of neural network controllers using Taylor model preconditioning. In: Proceedings of the International Conference on Computer Aided Verification, pp. 249–262 (2021)
22. Katz, G., et al.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Proceedings of the International Conference on Computer Aided Verification, pp. 97–117 (2017)
23. Khan, S., et al.: Facial recognition using convolutional neural networks and implementation on smart glasses. In: Proceedings of the International Conference on Information Science and Communication Technology (2019). Article 19
24. Khedr, H., Ferlez, J., Shoukry, Y.: PEREGRiNN: penalized-relaxation greedy neural network verifier. In: Proceedings of the International Conference on Computer Aided Verification, pp. 287–300 (2021)
25. Kochdumper, N., Althoff, M.: Sparse polynomial zonotopes: a novel set representation for reachability analysis. Trans. Autom. Control **66**(9), 4043–4058 (2021)
26. Makino, K., Berz, M.: Taylor models and other validated functional inclusion methods. Int. J. Pure and Appl. Math. **4**(4), 379–456 (2003)

27. Mukherjee, D., et al.: A survey of robot learning strategies for human-robot collaboration in industrial settings. Robot. Comput.-Integr. Manuf. **73** (2022)
28. Müller, M.N., et al.: PRIMA: precise and general neural network certification via multi-neuron convex relaxations. Proceedings on Programming Languages 1 (2022). Article 43
29. Müller, M.N., et al.: The third international verification of neural networks competition (VNN-COMP 2022): summary and results. arXiv preprint arXiv:2212.10376 (2022)
30. Pulina, L., Tacchella, A.: Challenging SMT solvers to verify neural networks. AI Commun. **25**(2), 117–135 (2012)
31. Raghunathan, A., Steinhardt, J., Liang, P.: Semidefinite relaxations for certifying robustness to adversarial examples. In: Proceedings of the International Conference on Neural Information Processing Systems, pp. 10900–10910 (2018)
32. Riedmiller, M., Montemerlo, M., Dahlkamp, H.: Learning to drive a real car in 20 minutes. In: Proceedings of the International Conference on Frontiers in the Convergence of Bioscience and Information Technologies, pp. 645–650 (2007)
33. Schilling, C., Forets, M., Guadalupe, S.: Verification of neural-network control systems by integrating Taylor models and zonotopes. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 8169–8177 (2022)
34. Singh, G., et al.: Fast and effective robustness certification. In: Proceedings of the International Conference on Advances in Neural Information Processing Systems (2018)
35. Singh, G., et al.: An abstract domain for certifying neural networks. Proce. Prog. Lang. **3** (2019). Article 41
36. Singh, G., et al.: Beyond the single neuron convex barrier for neural network certification. In: Proceedings of the International Conference on Advances in Neural Information Processing Systems (2019)
37. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: Proceedings of the International Conference on Learning Representations (2019)
38. Tran, H.D., et al.: Parallelizable reachability analysis algorithms for feed-forward neural networks. In: Proceedings of the International Conference on Formal Methods in Software Engineering, pp. 51–60 (2019)
39. Tran, H.D., et al.: Safety verification of cyber-physical systems with reinforcement learning control. Trans. Embedded Comput. Syst. 18(5s) (2019). Article 105
40. Tran, H.D., et al.: Star-based reachability analysis of deep neural networks. In: Proceedings of the International Symposium on Formal Methods, pp. 670–686 (2019)
41. Tran, H.D., et al.: NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Proc. of the Int. Conf. on Computer Aided Verification. pp. 3–17 (2020)
42. Vincent, J.A., Schwager, M.: reachable polyhedral marching (RPM): a safety verification algorithm for robotic systems with deep neural network components. In: Proceedings of the International Conference on Robotics and Automation, pp. 9029–9035 (2021)
43. Wang, S., et al.: Formal security analysis of neural networks using symbolic intervals. In: Proceedings of the USENIX Security Symposium, pp. 1599–1614 (2018)
44. Wang, S., et al.: Beta-CROWN: efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Proceedings of the International Conference on Neural Information Processing Systems (2021)

45. Weng, L., et al.: Towards fast computation of certified robustness for ReLU networks. In: Proceedings of the International Conference on Machine Learning, pp. 5276–5285 (2018)
46. Xiang, W., et al.: Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. In: Proceedings of the American Control Conference, pp. 1574–1579 (2018)
47. Xu, K., et al.: Fast and complete: enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: Proceedings of the International Conference on Learning Representations (2021)
48. Yang, X., et al.: Reachability analysis of deep ReLU neural networks using facet-vertex incidence. In: Proceedings of the International Conference on Hybrid Systems: Computation and Control (2021). Article 18
49. Zhang, H., et al.: Efficient neural network robustness certification with general activation functions. In: Proceedings of the International Conference on Neural Information Processing Systems, pp. 4944–4953 (2018)