Certified Error Analysis of Homomorphically Encrypted Neural Networks

Philipp Kern[1], Edoardo Manino[2]*, and Carsten Sinz[1,3]

[1] Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany
philipp.kern@kit.edu, carsten.sinz@kit.edu
[2] The University of Manchester, Manchester M13 9PL, United Kingdom
edoardo.manino@manchester.ac.uk
[3] Karlsruhe University of Applied Sciences, 76133 Karlsruhe, Germany
carsten.sinz@h-ka.de

Abstract. Fully-Homomorphic Encryption (FHE) has been touted as the ultimate solution for preserving user data privacy in Machine Learning as a Service (MLaaS) applications. Under this scheme, the server never sees the user data in clear, but executes the ML model on encrypted data instead. Unfortunately, efficient FHE schemes only support addition and multiplication operations, which cannot exactly represent common activation functions in neural networks. Substituting activation functions with polynomial approximations may cause significant output deviations. In this paper, we propose ZonoPoly, an efficient algorithm to compute guaranteed bounds on the maximum output deviation of FHE neural networks. We implement our algorithm in the VeryDiff framework by extending its zonotope-based reachability analysis primitives to support high-degree polynomials. Experimental results show that ZonoPoly produces approximately 3× tighter bounds than existing methods in most cases.

Keywords: Fully-homomorphic encryption · Privacy-preserving machine learning · Neural network verification · Equivalence checking.

1 Introduction

The past decade has seen an increasing use of cloud computing services to run large machine learning models, a product often referred to as Machine Learning as a Service (MLaaS) [37]. On the one hand, MLaaS has the advantage of lifting the high computational requirements of machine learning away from edge devices and providing on-demand infrastructure for large-scale use cases. On the other hand, MLaaS introduces privacy risks, as the user data is transferred to a third-party server for processing.

While there are many ways to mitigate the privacy risk in MLaaS, including anonymising the data prior to transmission, the most principled solution is Fully-Homomorphic Encryption (FHE). This family of cryptographic primitives allows

* Corresponding author

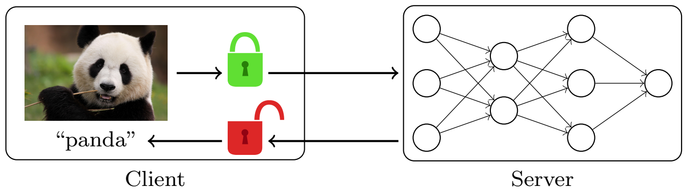


Fig. 1: In FHE-enabled MLaaS, the server operates on encrypted data.

a third party – the server – to execute a computer program on encrypted data, without ever accessing the plaintext. After that, the still-encrypted output can be sent back to the user, who decrypts it and recovers the plaintext output (see Figure 1). With such FHE scheme, a MLaaS provider could offer fully-confidential execution of arbitrary ML models.

In this light, the introduction of modern FHE schemes has rapidly sparked interest in the feasibility of running large ML models and neural networks with it [22]. In addition to the considerable computational overhead [12], a key challenge with FHE is that it imposes strict limitations on the operations that can be performed on encrypted data [35]. For the most ML-friendly FHE schemes like CKKS [11], these restrictions can go as far as allowing only the execution of additions and multiplications, but not the non-linear activation functions of neural networks like ReLU. Consequently, activation functions must be replaced with approximating high-degree polynomials, often also requiring a retraining of the network.

After a number of early attempts at training and fine-tuning neural networks with quadratic and low-degree polynomial activations, the field has realised that they degrade the accuracy of the model too much. As a result, the state of the art solution is replacing each activation function in a neural network with a high-degree polynomial approximation. Since higher-degree polynomials increase FHE's computational cost, an acceptable compromise between accuracy and execution time needs to be found via repeated experimentation.

Attempts at computing verified error bounds between the original and modified network exist [28], but have not yet become mainstream.

In this paper, we present ZonoPoly, an efficient algorithm to compute verified error bounds between a neural network and its polynomial counterpart. With it, we demonstrate that it is possible to derive tighter bounds on the error of FHE polynomial approximations of neural networks, improving upon existing approaches. We believe that our algorithm represents an important step towards integrating neural network verification methods into the design of FHE-enabled neural networks for ML-as-a-Service (MLaaS) applications.

More in detail, we make the following contributions to the state of the art:

– We propose ZonoPoly, a zonotope-based method capable of both synthesizing neural networks with polynomial activation functions and computing

verified bounds on the output difference between the original and polynomial approximated networks.
- We implement our ZonoPoly algorithm in the VeryDiff framework, thus extending its reachability analysis engine to support polynomial transformations.
- We compare ZonoPoly with LiGAR [28], the only existing formal method baseline in this domain, and show that we can derive 3.37× tighter bounds on average.

2 Background

For the sake of simplicity, we will assume that our ML model is a feedforward neural network $\mathcal{N} : \mathbb{R}^m \to \mathbb{R}^n$ with L fully-connected layers h_l, such that $\mathcal{N} = h_L \circ h_{L-1} \circ \cdots \circ h_1$. Each layer h_l has the following form:

$$h_l(\boldsymbol{x}_l) = \sigma\big(W_l \boldsymbol{x}_l + \boldsymbol{b}_l\big) \ , \tag{1}$$

where W_l is a matrix, $\boldsymbol{b}_l$ a column vector, and $\sigma(x) = \max(x, 0)$ the element-wise ReLU activation function. Our algorithm is – in principle – extensible to more sophisticated neural architectures, but we leave that effort to future work (see Section 5).

2.1 Fully-Homomorphic Encryption (FHE) Schemes

We define a homomorphic encryption scheme as follows.

Definition 1 (Homomorphic Encryption). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme, m_1, m_2 two plaintext messages, and g a function. Further assume that $m_3 = g(m_1, m_2)$ is the result of applying function g to the plaintext messages m_1,m_2, and $c_3 \equiv g(c_1, c_2)$ is the result of applying function g on the corresponding cyphertexts $c_1 \equiv \mathsf{Enc}(m_1)$ and $c_2 \equiv \mathsf{Enc}(m_2)$. Then $\mathcal{E}$ is homomorphic with respect to function g if we have $\mathsf{Dec}(c_3) = m_3$.*

Early attempts at creating a homomorphic scheme are limited to one operation g alone [3]. For example, the scheme in [32] supports only homomorphic addition $g(m_1, m_2) = m_1 + m_2$, whereas RSA encryption [38] supports only multiplication $g(m_1, m_2) = m_1 m_2$. The first scheme that supports both addition and multiplication, while also allowing the computation of programs of arbitrary depth, is by Gentry [20]. After that, the field has focused on leveraging more efficient computational problems, i.e. Learning With Errors (LWE) [7] and Ring Learning With Error (RLWE) [8], to reduce the computational overhead of FHE.

Introduction of *approximate* FHE schemes like CKKS [11] has enabled practical execution of machine learning algorithms. Instead of representing individual binary values, the CKKS scheme and its variants are able to manipulate vectors of real numbers $m = (x_1, x_2, \ldots, x_n)$, which makes them more efficient for machine learning purposes [35]. The downside is that the results are not *exact* [15]: for each homomorphic operation g, we have $\mathsf{Dec}(g(c_1, c_2)) = g(m_1, m_2) + \eta$, where the error term η depends on the user's encryption key and the value of the plaintext messages m_1, m_2. For the purpose of MLaaS, the error η can be arbitrarily reduced by changing the parameters of the FHE scheme, at the cost of increasing the computational overhead [1,6]. As such, we assume the intrinsic encryption error η is negligible and focus on the error introduced by approximating activation functions by polynomials (see Section 2.2) in the remainder of the paper.

2.2 Design of FHE Neural Networks

One of the main challenges in the execution of FHE neural networks is dealing with activation functions, as they cannot be represented as a computational circuit of additions and multiplications [35]. Instead, early attempts try to replace all activations with approximating low-degree polynomials, such as quadratic activations [10,22,29]. Unfortunately, any polynomial activation of degree $d \geq 2$ has unbounded derivatives, which cause gradient instability during training and fine-tuning. Furthermore, reusing the original weights is not viable as the approximation error at each activation accumulates, yielding low overall predictive accuracy [19].

A more recent approach consists of keeping the trained model unmodified and building close approximations $p_d(x) \approx \sigma(x)$ of the activation functions with high-degree polynomials ($d \approx 100$ or more) [26,27]. In order to improve numerical stability, some authors prefer to build approximations in Chebyshev basis [39] or employ custom hierarchical factorisations [25]. Still, these approximations minimize the error for a specific input range $x \in [l, u]$, while becoming increasingly worse outside of it (see Figure 2). To our knowledge, the existing literature does not explore the problem of correctly estimating the ranges $[l, u]$. Instead, most works seem to rely on symmetric ranges $[-c, +c]$ and either fix their width a priori (e.g. $c = 3$) [10,30] or use some unspecified empirical approach to estimate its value (up to $c = 50$ for a CIFAR-10 model) [27,39].

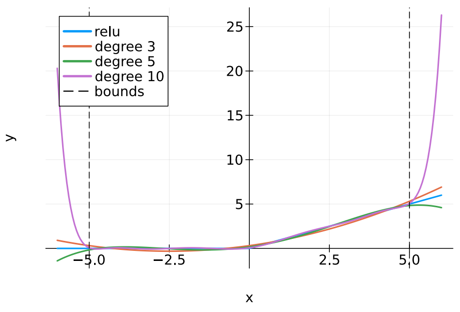


Fig. 2: Polynomial approximations quickly deviate to $\pm\infty$ outside of the approximation interval.

Certified Design The only existing attempt at designing a FHE neural network with formal guarantees is the LiGAR algorithm [28]. Its authors propose to abstract the polynomial approximation error by injecting an additional symbolic input after every activation function. Then, they compute a certified bound on the activation ranges $[l, u]$, and estimate the numerical stability of the network by computing its Lipschitz constant (for a definition see, e.g., [28]). With it, they can optimize the polynomial degree of each activation to achieve a desired output error bound.

Still, LiGAR employs relatively inefficient neural network verification techniques, including parallel linear bounds for reachability analysis [45] and Lipschitz bounds for equivalence checking [40]. Furthermore, it simplifies the optimisation problem by replacing the objective function with an asymptotic approximation of its value [44]. Together, these factors cause LiGAR to yield very conservative estimates, as we show in Section 4.3.

2.3 Equivalence Checking of ReLU Neural Networks

In this paper, we cast the problem of analysing the difference between polynomial FHE networks and original ReLU networks as an *equivalence checking* problem. Specifically, we are interested in the following definition of equivalence:

Definition 2 (ε-Equivalence). *The neural networks $\mathcal{N}_1$ and $\mathcal{N}_2$ are equivalent in the domain $\mathcal{D}$ according to norm $\|\cdot\|_p$, if for any input $x \in \mathcal{D}$ we have $\|\mathcal{N}_1(x) - \mathcal{N}_2(x)\|_p \leq \varepsilon$.*

In general, checking the equivalence of two ReLU networks $\mathcal{N}_1$ and $\mathcal{N}_2$ is a coNP-complete problem [42]. As such, exact verification techniques, such as mixed integer programming [23] and SMT solving [17], struggle to scale to reasonably-sized networks. Popular alternatives use incomplete verification techniques that over-approximate the set of outputs that are reachable from the input domain $\mathcal{D}$ [33,34,43], as detailed in Section 2.4.

2.4 Zonotope-Based Reachability Analysis

In this paper, we use *zonotopes* [21,41,43] to represent reachable sets:

Definition 3 (Zonotope). *A zonotope $\mathcal{Z} = (\boldsymbol{c}, G)$ with n generators and dimension m is an affine transformation of the hypercube $[-1, 1]^n$ described by*

$$\mathcal{Z} = \{x \in \mathbb{R}^m \mid G\boldsymbol{\epsilon} + \boldsymbol{c} = x, \boldsymbol{\epsilon} \in [-1, 1]^n\} \ , \tag{2}$$

where $G \in \mathbb{R}^{m \times n}$ and $\boldsymbol{c} \in \mathbb{R}^m$ are the generator matrix and center of the zonotope, respectively. The columns of G are called generators of the zonotope.

Affine Layers. A zonotope $\mathcal{Z} = (\boldsymbol{c}, G)$ can be propagated through an affine transformation $\boldsymbol{x} \mapsto A\boldsymbol{x} + \boldsymbol{b}$ without introducing any over-approximation. The resulting zonotope $\hat{\mathcal{Z}}$ can be computed as

$$\hat{\mathcal{Z}} = A\,\mathcal{Z} + \boldsymbol{b} = (A\boldsymbol{c} + \boldsymbol{b}, AG) \ . \tag{3}$$

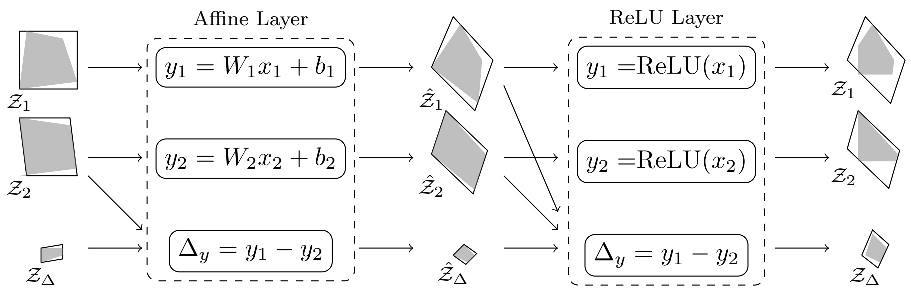


Fig. 3: Zonotope propagation (black) over-approximates the reachable sets (gray) at every layer. Differential verification keeps track of the explicit difference $\mathcal{Z}_\Delta$.

ReLU Layers. A zonotope $\mathcal{Z} = (\boldsymbol{c}, G)$ cannot be propagated through a non-linear function like ReLU without over-approximation [41]. More specifically, assume that $\sigma : \mathbb{R} \to \mathbb{R}$ is a non-linear activation function. Also, assume that σ is bounded in the interval domains $I_i = [l_i, u_i]$, $1 \le i \le m$, such that $\forall x_i \in I_i \,:\, |\alpha_i x_i + \beta_i - \sigma(x_i)| \le \gamma_i$ for some $\alpha_i, \beta_i \in \mathbb{R}$ and error bounds $\gamma_i \in \mathbb{R}$. The resulting over-approximated zonotope $\hat{\mathcal{Z}}$ becomes:

$$\hat{\mathcal{Z}} = \boldsymbol{\alpha} \odot \mathcal{Z} + \boldsymbol{\beta} + \boldsymbol{\gamma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \in [-1,1]^m \tag{4}$$
$$= \left(\boldsymbol{\alpha} \odot \boldsymbol{c} + \boldsymbol{\beta}, \left[\boldsymbol{\alpha} \odot G \quad \mathrm{diag}(\boldsymbol{\gamma})\right]\right) \ , \tag{5}$$

where $\odot$ represents element-wise multiplication and $\mathrm{diag}(\cdot)$ turns a vector into a diagonal matrix. The fresh variables $\boldsymbol{\epsilon}$ in Eq. (4) allow each dimension to independently vary within its error bounds [14]. To capture this concept in the zonotope formalism, the error bounds $\boldsymbol{\gamma}$ are appended as a diagonal matrix to the end of the scaled original generator matrix G.

2.5 Differential Verification

Computing the output sets of $\mathcal{N}_1$ and $\mathcal{N}_2$ may not be enough to prove their equivalence, since the over-approximation grows due to the contribution of $\boldsymbol{\gamma}$ at every ReLU layer (see Equation (4)). Instead, both the pioneering work of Paulsen [33,34] and the VeryDiff tool [43] show that we should keep track of a third reachability set over the difference function between $\mathcal{N}_1$ and $\mathcal{N}_2$ (see Figure 3). When $\mathcal{N}_1$ and $\mathcal{N}_2$ have similar architecture and weights, this *differential verification* approach allows us to greatly reduce the over-approximation.

Affine Layers. Given affine transformations $\boldsymbol{y}_i = W_i\boldsymbol{x}_i + \boldsymbol{b}_i$ for $i \in \{1,2\}$ and $\boldsymbol{\Delta}_x = \boldsymbol{x}_1 - \boldsymbol{x}_2$, the difference function $\boldsymbol{\Delta}_y = \boldsymbol{y}_1 - \boldsymbol{y}_2$ can be written as

$$\begin{aligned}\boldsymbol{\Delta}_y &= \boldsymbol{y}_1 - \boldsymbol{y}_2 && (6)\\ &= W_1\boldsymbol{x}_1 + \boldsymbol{b}_1 - (W_2\boldsymbol{x}_2 + \boldsymbol{b}_2) && (7)\\ &= W_1\left((\boldsymbol{x}_1 - \boldsymbol{x}_2) + \boldsymbol{x}_2\right) + \boldsymbol{b}_1 - (W_2\boldsymbol{x}_2 + \boldsymbol{b}_2) && (8)\\ &= W_1\left(\boldsymbol{\Delta}_x + \boldsymbol{x}_2\right) + \boldsymbol{b}_1 - (W_2\boldsymbol{x}_2 + \boldsymbol{b}_2) && (9)\\ &= (W_1 - W_2)\boldsymbol{x}_2 + W_1\boldsymbol{\Delta}_x + (\boldsymbol{b}_1 - \boldsymbol{b}_2) && (10)\end{aligned}$$

which depends only on $\boldsymbol{x}_2$ and $\boldsymbol{\Delta}_x = \boldsymbol{x}_1 - \boldsymbol{x}_2$ as inputs.

Proposition 1 (Affine Differential Zonotope [43]). *Given zonotopes* $\mathcal{Z}_1$ *with* $\boldsymbol{x}_1 \in \mathcal{Z}_1$ *(written in the following as* $\mathcal{Z}_1 \ni \boldsymbol{x}_1$*),* $\mathcal{Z}_\Delta \ni \boldsymbol{x}_1 - \boldsymbol{x}_2, \mathcal{Z}_2 \ni \boldsymbol{x}_2$ *and affine transformations* $\boldsymbol{y}_i = W_i\boldsymbol{x}_i + \boldsymbol{b}_i$ *for* $i \in \{1,2\}$*, the zonotopes*

$$\begin{aligned}\hat{\mathcal{Z}}_1 &= W_1\,\mathcal{Z}_1 + \boldsymbol{b}_1 && (11)\\ \hat{\mathcal{Z}}_\Delta &= (W_1 - W_2)\,\mathcal{Z}_2 + W_1\,\mathcal{Z}_\Delta + (\boldsymbol{b}_1 - \boldsymbol{b}_2) && (12)\\ \hat{\mathcal{Z}}_2 &= W_2\,\mathcal{Z}_2 + \boldsymbol{b}_2 && (13)\end{aligned}$$

over-approximate $\boldsymbol{y}_1, \boldsymbol{y}_1 - \boldsymbol{y}_2$ *and* $\boldsymbol{y}_2$.

ReLU Layers. Similarly to the propagation through the affine layer, the ReLU-difference function $\Delta_y^R = \mathrm{ReLU}(x_1) - \mathrm{ReLU}(x_2)$ can again be expressed as

$$\begin{aligned}\Delta_y^R &= \mathrm{ReLU}(x_1) - \mathrm{ReLU}(x_2) && (14)\\ &= \mathrm{ReLU}\left((x_1 - x_2) + x_2\right) - \mathrm{ReLU}(x_2) && (15)\\ &= \mathrm{ReLU}(\Delta_x + x_2) - \mathrm{ReLU}(x_2) && (16)\\ &= \mathrm{ReLU}(x_1) - \mathrm{ReLU}\left(x_1 - (x_1 - x_2)\right) && (17)\\ &= \mathrm{ReLU}(x_1) - \mathrm{ReLU}(x_1 - \Delta_x) && (18)\end{aligned}$$

which depends on either x_1 and Δ_x (Equation (18)) or x_2 and Δ_x (Equation (16)).

Proposition 2 (ReLU **Differential Zonotope [43]).** *Given zonotopes* $\mathcal{Z}_1 \ni \boldsymbol{x}_1, \mathcal{Z}_\Delta \ni \boldsymbol{x}_1 - \boldsymbol{x}_2, \mathcal{Z}_2 \ni \boldsymbol{x}_2$ *and* $\boldsymbol{y}_1 = \mathrm{ReLU}(\boldsymbol{x}_1), \boldsymbol{y}_2 = \mathrm{ReLU}(\boldsymbol{x}_2)$*, the zonotopes*

$$\begin{aligned}\hat{\mathcal{Z}}_1 &= \boldsymbol{\lambda}_1 \odot \mathcal{Z}_1 + \boldsymbol{\xi}_1 + \boldsymbol{\xi}_1\boldsymbol{\epsilon}_1 && (19)\\ \hat{\mathcal{Z}}_\Delta &= \boldsymbol{a}_1 \odot \mathcal{Z}_1 + \boldsymbol{a}_2 \odot \mathcal{Z}_2 + \boldsymbol{a}_\Delta \odot \mathcal{Z}_\Delta + \boldsymbol{b} + \boldsymbol{c}\boldsymbol{\epsilon}_\Delta && (20)\\ \hat{\mathcal{Z}}_2 &= \boldsymbol{\lambda}_2 \odot \mathcal{Z}_2 + \boldsymbol{\xi}_2 + \boldsymbol{\xi}_2\boldsymbol{\epsilon}_2 && (21)\end{aligned}$$

over-approximate $\boldsymbol{y}_1, \boldsymbol{y}_1 - \boldsymbol{y}_2$ *and* $\boldsymbol{y}_2$*. Where* $\boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \boldsymbol{\epsilon}_\Delta$ *are new independent generators,*

$$\boldsymbol{\lambda}_i = \frac{\boldsymbol{u}_i}{\boldsymbol{u}_i - \boldsymbol{l}_i}, \quad \boldsymbol{\xi}_i = \frac{-\boldsymbol{l}_i\boldsymbol{\lambda}_i}{2} \qquad (22)$$

for $\boldsymbol{l}_i \le \mathcal{Z}_i \le \boldsymbol{u}_i$ *element-wise and the values of* a_1, a_2, a_Δ, b, c *are set according to Table 1.*

Table 1: Case distinction for differential ReLU propagation. Different ReLU phases are denoted by $-$ (negative), $+$ (positive) and $\sim$ (unstable). The values for λ_i, ξ_i are defined in (22). $\alpha_i = \frac{-l_i}{u_i - l_i}, \mu_i = \frac{1}{2}\lambda_i u_i$ and $\lambda_\Delta = \text{clamp}\left(\frac{u_\Delta}{u_\Delta - l_\Delta}, 0, 1\right)$, $\mu_\Delta = \frac{1}{2}\max(-l_\Delta, u_\Delta)$, $\nu_\Delta = \lambda_\Delta \max(0, -l_\Delta)$

ReLU		**Factor**				
σ_1	σ_2	a_1	a_2	a_Δ	b	c
$-$	$-$	0	0	0	0	0
$-$	$+$	0	-1	0	0	0
$+$	$-$	1	0	0	0	0
$+$	$+$	0	0	1	0	0
$\sim$	$-$	λ_1	0	0	ξ_1	0
$-$	$\sim$	0	$-\lambda_2$	0	$-\xi_2$	0
$\sim$	$+$	$-\alpha_1$	0	1	μ_1	μ_1
$+$	$\sim$	0	α_2	1	$-\mu_2$	μ_2
$\sim$	$\sim$	0	0	λ_Δ	$\nu_\Delta - \mu_\Delta$	μ_Δ

3 Methodology

In this section, we present ZonoPoly, an algorithm to synthesise polynomial approximation networks with certified error bound. In contrast to previous work [28], ZonoPoly constructs concrete polynomials instead of abstracting them away. As such, ZonoPoly produces tighter error estimates than [28] at the expense of increased computational cost (see Section 4.3). At the same time, manipulating high degree polynomials can lead to numerical instability: we show how to mitigate such risk in Section 3.3.

At high level, the process of running ZonoPoly can be divided in two stages. First, we construct a polynomial network using verified bounds, as described in Section 3.1. We output not only the polynomial degrees (as in [28]), but the concrete coefficients of the polynomial activation functions as well as the bounds used for approximation. Second, we compute a certified bound on the output difference between the original network $\mathcal{N}_1$ and the corresponding polynomial network $\mathcal{N}_2$. To do so, we augment the zonotope-based differential verification approach of VeryDiff with a novel relaxation for $p(x) - \text{ReLU}(x - \Delta)$, as described in Section 3.2.

3.1 Verified Construction of Polynomial Networks

In order to synthesize networks with polynomial activation functions, we need estimates over the pre-activation ranges $[l, u]$ of each neuron (see Section 2.2). Then, we can compute the polynomial approximation $p(x) \approx \text{ReLU}(x)$ that minimises the absolute error over $[l, u]$, by using the Remez algorithm [36]. Since the approximation error quickly diverges to $\pm\infty$ outside of $[l, u]$ (see Figure 2), we need verified bounds over the ranges $[l, u]$ that contain all possible values attainable by concrete execution of the network.

Algorithm 1 Construction of network with polynomial approximations as activation functions using verified bounds

```
1: function CONSTRUCT_NETWORK($\langle W_l, \boldsymbol{b}_l \rangle_{l=1}^{L}$, $\boldsymbol{l}_0$, $\boldsymbol{u}_0$, $d$)
Require: $\langle W_l, \boldsymbol{b}_l \rangle_{l=1}^{L}$ neural network parameters, $\boldsymbol{l}_0, \boldsymbol{u}_0$ concrete bounds on the network inputs, $d$ degree of the polynomial approximations
Ensure: Pre-activation bounds $\langle \boldsymbol{l}_l \rangle_{l=1}^{L-1}$, $\langle \boldsymbol{u}_l \rangle_{l=1}^{L-1}$, list of vectors of polynomials $\langle \boldsymbol{p}_l \rangle_{l=1}^{L-1}$
2:     $\mathcal{Z} \leftarrow$ zonotope($\boldsymbol{l}_0, \boldsymbol{u}_0$)
3:     for $l \leftarrow 1, \dots, L-1$ do
4:         $\mathcal{Z} \leftarrow W_l\, \mathcal{Z} + \boldsymbol{b}_l$
5:         $\boldsymbol{l}_l, \boldsymbol{u}_l \leftarrow$ bounds($\mathcal{Z}$)
6:         $\boldsymbol{p}_l, \_ =$ remez(ReLU, $\boldsymbol{l}_l, \boldsymbol{u}_l, d$)        ▷ element-wise for each neuron
7:         $\mathcal{Z} \leftarrow \boldsymbol{p}(\mathcal{Z})$        ▷ Proposition 3
8:     end for
9: end function
```

Furthermore, the bounds $[l, u]$ on the ranges of neurons at layer $l \in [2, L]$ depend on the error introduced by all approximations $p(x)$ at previous layers $j \in [1, l-1]$. Therefore, it is not sufficient to compute the bounds by running reachability analysis on the original network with ReLU activations. Instead, we have to track the approximation error we introduce at each layer l and propagate it correctly to the next layers. We formally describe the resulting procedure in Algorithm 1.

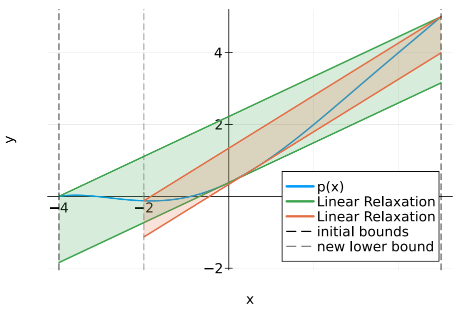


Fig. 4: Two zonotope relaxations of a polynomial activation $p(x)$. Note how the zonotope relaxation for a smaller input domain (orange) might not be fully contained in the one for a larger input domain (green).

There, we use the zonotope propagation rules for affine layers presented in Equation (3) (Line 4). The resulting zonotope gives us an easy way to compute bounds on the pre-activation ranges (Line 5). With them, we can compute polynomial approximations of the activation functions (Line 6). Finally, we need to propagate the zonotope through the polynomial activations (Line 7). For this purpose, we propose a way to over-approximate each univariate polynomial activation $p(x)$ with linear parallel bounds (see Figure 4). More formally:

Proposition 3 (Linear Relaxation of Polynomials). *Given $\boldsymbol{y} = \boldsymbol{p}(\boldsymbol{x})$ for a vector of univariate polynomials $\boldsymbol{p}(\boldsymbol{x}) = (p_1(x_1), \dots, p_n(x_n))$ and a zonotope $\mathcal{Z} \ni \boldsymbol{x}$, we obtain a linear relaxation (a polynomial of degree 1) using*

$$\hat{\boldsymbol{p}}, \boldsymbol{\gamma} = \text{remez}(\boldsymbol{p}, \boldsymbol{l}, \boldsymbol{u}, 1)\ , \tag{23}$$

where $\hat{\boldsymbol{p}}(\boldsymbol{x}) = \boldsymbol{\alpha} \odot \boldsymbol{x} + \boldsymbol{\beta}$ and $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$ are obtained from $\mathcal{Z}$ and $\boldsymbol{\alpha} \odot \boldsymbol{x} + \boldsymbol{\beta} - \boldsymbol{\gamma} \leq \boldsymbol{p}(\boldsymbol{x}) \leq \boldsymbol{\alpha} \odot \boldsymbol{x} + \boldsymbol{\beta} + \boldsymbol{\gamma}$ for all $\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]$. The new zonotope $\hat{\mathcal{Z}} \ni \boldsymbol{y}$ is then given *by*

$$\hat{\mathcal{Z}} = \boldsymbol{\alpha} \odot \mathcal{Z} + \boldsymbol{\beta} + \boldsymbol{\gamma} \odot \boldsymbol{\epsilon} \ , \tag{24}$$

where $\boldsymbol{\epsilon} \in [-1,1]^n$ *are new error symbols.*

3.2 Differential Verification of Polynomial and ReLU Networks

In Section 2.5, we reviewed how existing work models the difference between two ReLU activation functions. Here, we instead consider the difference $\varDelta_y = p(x_1) - \mathrm{ReLU}(x_2)$ between a polynomial and a ReLU activation. We can still take inspiration from Equations (14)–(18) and write the following:

$$\varDelta_y = p(x_1) - \mathrm{ReLU}(x_2) \tag{25}$$

$$= p(\varDelta_x + x_2) - \mathrm{ReLU}(x_2) \tag{26}$$

$$= p(x_1) - \mathrm{ReLU}(x_1 - \varDelta_x) \ , \tag{27}$$

While Equation (26) requires reasoning over a multivariate polynomial of possibly high degree, Equation (27) is much easier to handle. Indeed, we can analyse the two linear segments of $\mathrm{ReLU}(x_1 - \varDelta_x)$ separately, thus splitting Equation (27) into two cases:

$$p(x_1) - \mathrm{ReLU}(x_1 - \varDelta_x) = \begin{cases} p(x_1) & x_1 \leq \varDelta_x \\ (p(x_1) - x_1) + \varDelta_x & x_1 > \varDelta_x \end{cases} \tag{28}$$

The advantage of Equation (28) is that we only need to reason about the *univariate* polynomials $p(x_1)$ and $p(x_1) - x_1$, while treating the addition of $\varDelta_x$ independently. Furthermore, we are left with only three possible activation patterns, rather than the nine required when comparing two ReLU networks (see Table 1). More specifically, let the pre-activation ranges be $x_1 \in [l_1, u_1], x_2 \in [l_2, u_2]$ and $\varDelta_x \in [l_\varDelta, u_\varDelta]$. Then, we can distinguish between always inactive, always active, and unstable ReLU as follows.

Inactive ReLU If $u_2 \leq 0$, then $\mathrm{ReLU}(x_2)$ is always inactive. In this case, the difference function simplifies to $p(x_1) - \mathrm{ReLU}(x_2) = p(x_1)$. As such, the difference zonotope is identical to the zonotope of the polynomial network, which we compute according to Proposition 3.

Active ReLU If $l_2 \geq 0$, then $\mathrm{ReLU}(x_2)$ is always active. In this case, the difference function simplifies to $p(x_1) - \mathrm{ReLU}(x_2) = (p(x_1) - x_1) + \varDelta_x$. Thus, we can first compute a linear over-approximation of the univariate polynomial $p(x_1) - x_1$ and then add the linear term $\varDelta_x$. In particular, let:

$$\hat{p}, \gamma = \mathrm{remez}(p(x_1) - x_1, l_1, u_1, 1) \ , \tag{29}$$

where $\hat{p}(x_1) = \alpha x_1 + \beta$ and $\alpha x_1 + \beta - \gamma \leq p(x_1) - x_1 \leq \alpha x_1 + \beta + \gamma$ for $x_1 \in [l_1, u_1]$. Then, the final linear relaxation is:

$$\alpha x_1 + \varDelta_x + \beta - \gamma \leq (p(x_1) - x_1) + \varDelta_x \leq \alpha x_1 + \varDelta_x + \beta + \gamma \ . \tag{30}$$

Unstable ReLU If $l_2 < 0 < u_2$, then we do not know whether $\mathrm{ReLU}(x_2)$ is active or inactive. In this case, we need to find a linear relaxation $a_1 x_1 + a_\Delta \Delta_x + b \pm c$ that satisfies:

$$a_1 x_1 + a_\Delta \Delta_x + b - c \leq p(x_1) - \mathrm{ReLU}(x_1 - \Delta_x) \leq a_1 x_1 + a_\Delta \Delta_x + b + c \quad (31)$$

In ZonoPoly, we construct a valid over-approximation by guessing the values of a_1, a_Δ and optimising for the values of b, c. Specifically, we take the values of a_1, a_Δ from Table 1, which are the slopes VeryDiff uses for comparing ReLU activations [43]. Since we have $p(x) \approx \mathrm{ReLU}(x)$ for all neurons, the values of a_1, a_Δ are not far from optimal.

Relaxation Error. Given the values of a_1, a_Δ, the relaxation error is:

$$g(x_1, \Delta_x) = (p(x_1) - \mathrm{ReLU}(x_1 - \Delta_x)) - (a_1 x_1 + a_\Delta \Delta_x) \quad (32)$$

and we can set:

$$b = \frac{\max g + \min g}{2}, \quad c = \frac{\max g - \min g}{2} \quad (33)$$

For simplicity, we only describe how to compute the maximum relaxation error. The minimum can be computed in a similar fashion.

Error Maximisation. Since Equation 32 contains a ReLU activation, we split the domain of the maximisation problem into two regions: $x_1 \leq \Delta_x$ (inactive) and $x_1 \geq \Delta_x$ (active). More formally, we have $\max g(x_1, \Delta_x) = \max\{\mu_\leq, \mu_\geq\}$, where:

$$\mu_\leq = \max\ p(x_1) - a_1 x_1 - a_\Delta \Delta_x, \qquad s.t.\ \ x_1 \leq \Delta_x \quad (34)$$

$$\mu_\geq = \max\ p(x_1) - (1 + a_1) x_1 - (a_\Delta - 1)\Delta_x, \qquad s.t.\ \ x_1 \geq \Delta_x \quad (35)$$

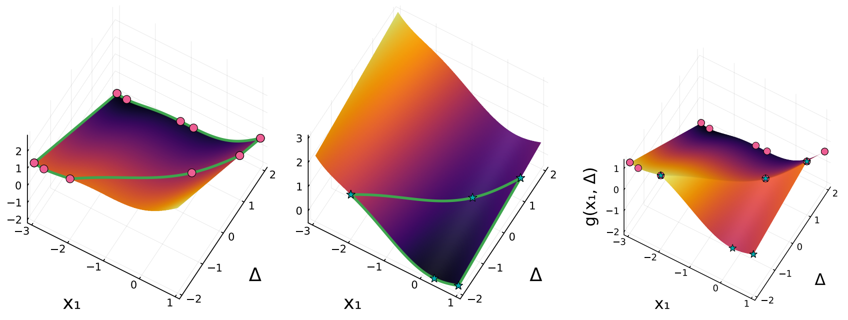
with $x_1 \in [l_1, u_1]$ and $\Delta_x \in [l_\Delta, u_\Delta]$. Note that both objective functions can be expressed as $p(x_1) + \alpha x_1 + \beta \Delta_x$ for suitable choices of α and β. Thus, solving the above optimization problems requires finding the maximum of a polynomial on a compact bounded set.

Critical Points. Since $g(x_1, \Delta_x) = p(x_1) + \alpha x_1 + \beta \Delta_x$ is a linear function in Δ_x, the maximum is always located at one of the three boundaries $\Delta_x = l_\Delta$, $\Delta_x = u_\Delta$, or $\Delta_x = x_1$. For the first two boundaries, define the critical points of the univariate polynomial $g(x_1, 0)$ as follows:

$$\mathcal{C}_{\Delta_x = 0} = \left\{ x : \frac{d}{dx}\Big(p(x) + \alpha x\Big) = 0 \right\} \cup \Big\{ l_1, u_1 \Big\} \quad (36)$$

For the latter boundary $\Delta_x = x_1$, define the critical points of another univariate polynomial $g(x_1, x_1)$ as:

$$\mathcal{C}_{\Delta_x = x_1} = \left\{ x : \frac{d}{dx}\Big(p(x) + \alpha x + \beta x\Big) = 0 \right\} \cup \Big\{ l_1, u_1, l_\Delta, u_\Delta \Big\} \quad (37)$$



(a) inactive case ($x_1 \leq \Delta$) (b) active case ($x_1 \geq \Delta$) (c) overall

Fig. 5: Critical points for $g(x_1, \Delta_x) = (p(x_1) - \text{ReLU}(x_1 - \Delta_x)) - (-{}^1\!/_7 x_1 + {}^8\!/_{10}\Delta_x)$. The two subproblems are shown on the left with the boundary of their domain shown as green lines. The overall relaxation error function with its critical points computed via the subproblems is shown on the right.

Then, we can build our final set of critical points, which is guaranteed to contain the solution to the optimisation problems in Equations 34 and 35:

$$\hat{\mathcal{C}} = \Big\{(x_1, \Delta_x) : x_1 \in \mathcal{C}_{\Delta_x=0}, \Delta_x \in \{l_\Delta, u_\Delta\}\Big\} \cup \Big\{(x_1, x_1) : x_1 \in \mathcal{C}_{\Delta_x=x_1}\Big\} \quad (38)$$

Since $\hat{\mathcal{C}}$ is finite, ZonoPoly simply searches over its feasible subset.

3.3 Numerical Stability

Polynomial Basis The methods we introduce in Sections 3.1 and 3.2 require the manipulation of polynomials of arbitrary degree and rely on common primitives such as evaluation, interpolation, and root finding. It is well known that their implementation in monomial basis can lead to numerical instability [44]. We show an example of this in Figure 6.

To avoid these numerical difficulties, we follow the recommendations of [44] and represent all polynomials in Chebyshev basis with inputs normalized to $[-1, 1]$. Each polynomial is stored as a triple $(\boldsymbol{c}, l, u)$ of its Chebyshev coefficients $\boldsymbol{c}$ and original input range $x \in [l, u]$. With it, we can implement the following:

Evaluation and Interpolation. For the former, we use Clenshaw recurrence [13]. For the latter, we implement the version of the Remez algorithm described in [31], which was originally developed for the Chebfun system [16].

Root Finding. Given the Chebyshev coefficients, finding the roots of the polynomial reduces to finding the eigenvalues of the corresponding colleague matrix [44]. The correctness of the critical points in Equations 36 and 37 depends on the soundness of the underlying eigenvalue solver. In our Julia implementation, we use the built-in `eigvals` command [5].

Zonotope Monotonicity Zonotope relaxations of activation functions are not monotonic in the size of the input domain (see Figure 4). As a consequence, running differential verification for a smaller input domain $\mathcal{D}' \subseteq \mathcal{D}$ may lead to non-overlapping input range bounds $[l', u'] \not\subseteq [l, u]$ for some neurons in the network. If the new bounds $[l', u']$ exceed the domain where the polynomial activation is well behaved (see Figure 2), numerical issues arise. We mitigate this risk by storing the *verified* bounds $[l, u]$ that we compute via Algorithm 1, and tightening any range estimate $[l', u']$ that exceeds them during differential verification.

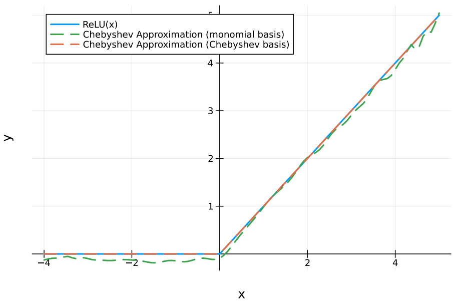


Fig. 6: Chebyshev approximation of degree 100 of $\mathrm{ReLU}(x)$ for $x \in [-4, 5]$. Determining monomial coefficients by solving a linear system and standard evaluation in monomial form vs. Chebyshev coefficients and evaluation using Clenshaw recurrence.

4 Experiments

We integrated the ZonoPoly algorithms for synthesis and differential verification of polynomially approximated networks in the VeryDiff [43] tool implemented in Julia [5]. The code of our implementation[4] and experiments[5] is available on GitHub. All experiments were run on a 4-core Intel Xeon E5-2670 CPU and 128 GB of RAM using single-threaded execution.

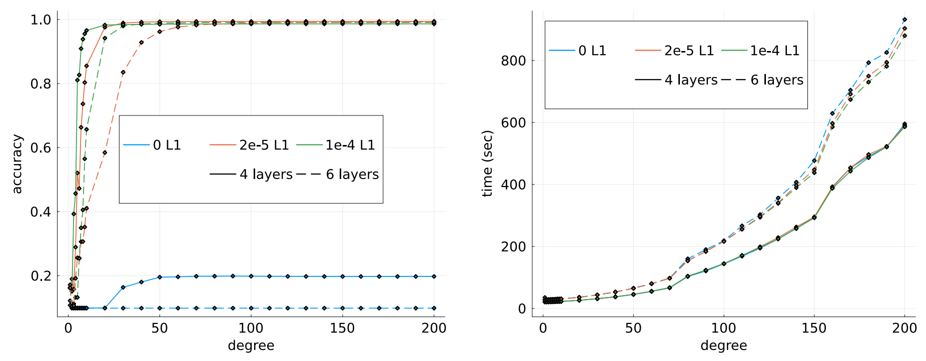
In our evaluation, we examine the accuracy of the polynomial networks constructed by ZonoPoly and compare the tightness of their differential bounds with the state-of-the-art tool LiGAR [28]. To ensure a fair comparison, we modify the latter to construct neural networks with uniform (rather than heterogeneous) polynomial degree. Furthermore, we introduce a LiGAR+Cheby variant that outputs concrete polynomials in Chebyshev basis, rather than just asymptotic bounds over minimax polynomials.

4.1 Benchmarks

We evaluate our approach on neural networks trained for privacy critical applications like credit scoring and human activity recognition as well as standard image classification:

MNIST. [24] The machine learning task in MNIST is to correctly classify the handwritten digit shown on 28×28 pixel grayscale images. Each pixel can attain

[4] https://github.com/samysweb/VeryDiff/tree/PolynomialEquivalence
[5] https://github.com/phK3/VeryDiffPolyExperiments



(a) Accuracy of polynomial networks (b) Synthesis runtime

Fig. 7: Accuracy of polynomial networks generated by ZonoPoly for different polynomial degrees and base ReLU networks and runtime of ZonoPoly for synthesis.

the values from $[0, 1]$ leading to an input set of $[0, 1]^{784}$. We use two base networks of size 4×256 (4 hidden layers of 256 neurons each) and 6×256 taken from the annual competition VNN-COMP[6][9]. For each of the networks, we train two networks of the same architecture, but with additional L_1-regularization with weights 2×10^{-5} and 10^{-4}.

HAR. [2] Given 561 statistical aggregates computed from smartphone gyroscopic data, the goal is to correctly predict human activity out of 6 candidates (standing, sitting, lying, walking, walking downstairs and walking upstairs). Each input dimension was normalized to $[-1, 1]$. We evaluate on the 1×500 network already used in the evaluations of ReluDiff[33] and NeuroDiff[34]. We also train two networks ourselves with the same structure and L_1-penalties with weights 2×10^{-5} and 10^{-4}.

HELOC. The *home equity line of credit* dataset was part of the FICO explainable machine learning challenge [18]. Given 23 features, the goal is to classify whether or not a person is credit-worthy. We normalized all features to $[0, 1]$ and trained two neural networks – one with standard training and one with L_1-penalty of 2×10^{-5}. Despite their small size (only two hidden layers of 64 and 32 neurons respectively) the networks achieve comparable accuracy to the models submitted to the FICO challenge [4].

4.2 Synthesis of Polynomially Approximated Neural Networks

Predictive Accuracy To examine the quality of polynomial networks generated by ZonoPoly, we constructed an initial zonotope containing the *whole*

[6] https://github.com/stanleybak/vnncomp2021

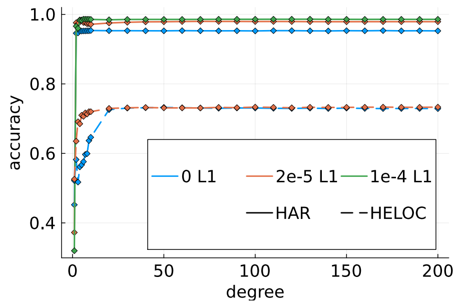


Fig. 8: Accuracy of polynomial networks generated by ZonoPoly for different base ReLU networks and polynomial degrees for HAR and HELOC.

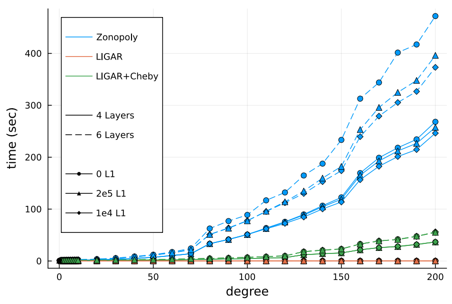


Fig. 9: Runtime for computation of verified bounds on polynomially approximated MNIST networks

input set (e.g. $[0,1]^{784}$ for the MNIST networks) and propagated that zonotope through the network to generate verified bounds and polynomial approximations according to Algorithm 1 for polynomial degrees of $1, 2, \dots, 10, 20, 30, \dots, 200$. Subsequently, we *statistically* evaluated the accuracy of the generated networks, as well as the maximum difference $\max_i \|\hat{\mathcal{N}}(x_i) - \mathcal{N}(x_i)\|_\infty$ between the polynomially approximated and the original network over the associated dataset. Results for *verified* bounds on the difference are reported in Section 4.3.

At least in the MNIST case, the most important factor for achieving good accuracy and low statistical difference from the original network is whether standard training or training with L_1 penalty was used. As shown in Figure 7a, ZonoPoly was able to generate polynomial networks that match the accuracy of the ReLU networks trained with some form of L_1 penalty for polynomials of degree 80 in the worst case. For the two networks that only used standard training, however, the accuracy of the generated polynomial networks never rises or gets stuck on a plateau even for high degrees – despite small improvements in the sampled maximum error shown in the left of Figure 10. Shallower networks also achieved better accuracy for smaller polynomial degrees than deeper networks.

For the one-layer HAR networks and small HELOC networks, ZonoPoly is also able to generate polynomial networks of matching accuracy for the ReLU networks obtained via standard training. However, the networks trained with L_1 penalty achieve better accuracy (Figure 8) and lower sampled error for smaller polynomial degrees (deferred to Appendix A.2).

Overall, the quality of the synthesized polynomial networks seems to be good whenever small bounds can be obtained via zonotope propagation. This in turn allows the polynomial approximation to be fitted to a small approximation interval and thus less error is incurred. Training using L_1 regularization was already shown to improve verification performance [46] by encouraging a smaller range of attainable values at the neurons.

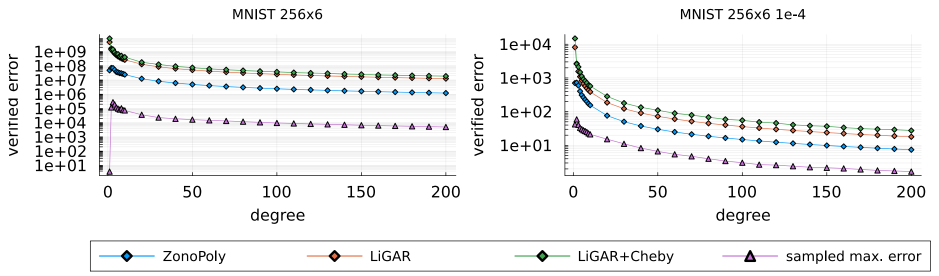


Fig. 10: Verified error bounds for polynomially approximated networks of different degree for different ReLU base networks for MNIST

Synthesis Time The synthesis time for a polynomially approximated network is primarily influenced by the number of neurons in the network and the degree of the polynomial approximation. This trend is illustrated in the MNIST case (Figure 7b), where the runtime increases with the polynomial degree. Networks with the same number of layers—corresponding to the same number of neurons in this case—exhibit overlapping runtime curves. Runtime curves for the other datasets are qualitatively similar and are deferred to Appendix A.1.

The dominant factor contributing to ZonoPoly's synthesis time is the computation of the approximation error between the current polynomial $p(x)$ and the ReLU function $\mathrm{ReLU}(x)$ over the interval $x \in [l, u]$ during each step of the Remez algorithm. This error must be evaluated for each neuron and becomes increasingly costly for higher-degree polynomials.

4.3 Verified Error Bounds

Bound Tightness We now compare the verified bounds on the error $\|\hat{\mathcal{N}}(x) - \mathcal{N}(x)\|_\infty$ between the polynomially approximated and the original ReLU networks computed by ZonoPoly to the verified bounds produced by LiGAR and LiGAR+Cheby. For ZonoPoly, we use the networks generated in the previous section and let both LiGAR versions compute their own polynomial networks. Here, we only log verification time, not generation time.

For each benchmark, we computed verified error bounds over the *whole input set* (e.g. $[0, 1]^{784}$ for the MNIST networks). Even for this large input set, ZonoPoly was able to produce useful bounds especially for the HELOC and HAR benchmarks and (to a lesser extent) also for the MNIST networks that were trained using L_1 regularization.

The change of the verified error for all tools as well as the sampled error over the MNIST training dataset is illustrated in Figure 10 on the left for the 256×6 MNIST network trained using standard training and the network with the same architecture trained with an L_1 penalty of 10^{-4} on the right. Graphs for other benchmarks follow a similar pattern and are deferred to Appendix A.2.

ZonoPoly was able to compute tighter bounds than LiGAR and LiGAR+Cheby on all benchmarks. For higher polynomial degrees, the relative improve-

Table 2: Verified bounds for the error $|\hat{\mathcal{N}}(x) - \mathcal{N}(x)|$ betweem polynomial networks and different ReLU base networks

base network		Error at degree 200			relative improvement	
Structure	L_1 weight	LiGAR	LiGAR+Cheby	ZonoPoly	$\frac{\text{LiGAR}}{\text{ZonoPoly}}$	$\frac{\text{LiGAR+Cheby}}{\text{ZonoPoly}}$
MNIST						
256x4	0	14240.51	21342.21	3472.10	4.10	6.15
256x4	2×10^{-5}	29.00	41.14	10.39	2.79	3.96
256x4	10^{-4}	5.55	8.06	2.15	2.58	3.74
256x6	0	1.25×10^7	1.87×10^7	1.21×10^6	10.35	15.49
256x6	2×10^{-5}	131.68	187.80	42.16	3.12	4.45
256x6	10^{-4}	17.94	27.54	7.42	2.42	3.71
HELOC						
$64-32$	0	0.54	0.80	0.31	1.73	2.57
$64-32$	2×10^{-5}	0.19	0.29	0.11	1.73	2.69
HAR						
500×1	0	0.66	0.82	0.27	2.43	3.02
500×1	2×10^{-5}	0.65	0.73	0.21	3.03	3.43
500×1	10^{-4}	0.63	0.72	0.23	2.79	3.18

ment of the bounds seems to converge to a constant factor for each base network (parallel lines in the logarithmic plot in Figure 10), which is shown alongside the absolute verified error for degree 200 in Table 2. The relative improvement for smaller degrees is slightly larger.

Verification Time However, the improvements in the verified bounds do not come for free. While LiGAR works on an efficient abstraction of the polynomial activations, ZonoPoly has to consider their actual coefficients and compute roots and linear approximations to these possibly high degree polynomials. Therefore – as illustrated in Figure 9 for MNIST – verification time grows much quicker with the degree of the polynomials compared to LiGAR whose runtime is constant in that regard.

It is also expected that ZonoPoly is slower than LiGAR+Cheby since this augmentation of LiGAR only has to fit the Chebyshev approximation and compute its error once for each neuron whereas ZonoPoly has to compute the error between the linear relaxation and the polynomial for every iteration of the Remez algorithm. The number of Remez iterations required can also be dependent on the width of the approximation interval of a polynomial. This may also explain, why differential verification for the MNIST network trained with standard training takes more time than verification of the L_1 trained networks of the same structure.

4.4 Verified Error for Samples

Neural network verifiers are commonly evaluated in the context of adversarial robustness for small ϵ-balls around points in a training dataset. We also perform

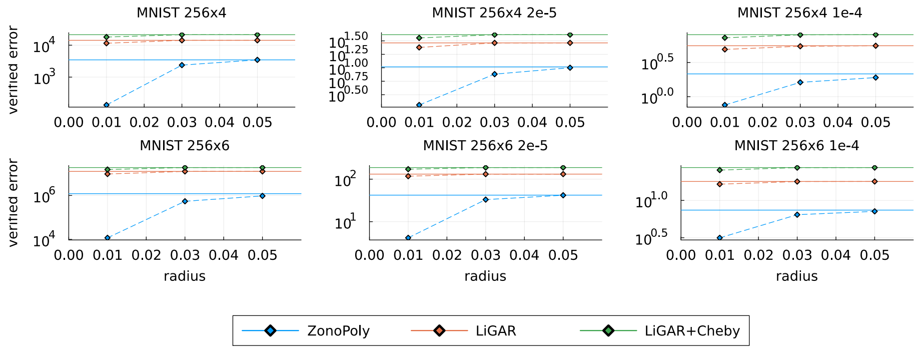


Fig. 11: Verified error bounds for ϵ-balls of different radii around the first 10 samples of the MNIST training dataset. The solid lines represent the globally-valid bounds (for the total input set of $[0, 1]^{784}$) for each tool.

a small comparison of the difference bounds computed by ZonoPoly against LiGAR and LiGAR+Cheby for ϵ-balls of different radii around samples of MNIST images. Due to ZonoPoly's runtime, we only evaluated on the first 10 samples of the MNIST training dataset. While equivalence is only checked for the ϵ-balls around the samples, we consider polynomial networks that were generated using verified bounds for the whole MNIST input set.

Figure 11, shows the bounds computed by each tool for the whole MNIST input set as solid horizontal lines, while the dashed lines represent the maximum verified difference among the ϵ-balls around the 10 center images. The bounds computed by ZonoPoly for the whole MNIST input set are already tighter than the verified bounds for LiGAR and LiGAR+Cheby for radius 0.01. With smaller radii, ZonoPoly also improves its bounds more than both other tools.

5 Conclusions

In this paper, we tackle the problem of designing polynomial activations for FHE neural networks. Our ZonoPoly algorithm employs a zonotope-based differential verification approach to propagate the approximation error through the network. As a result, ZonoPoly can construct a stable polynomial network with verified neuron ranges and output error. Our evaluation shows that ZonoPoly produces tighter bounds than existing certified techniques such as LiGAR.

Acknowledgments. The authors thank Samuel Teuber for technical support with VeryDiff. Edoardo Manino is partially funded by the EPSRC grant EP/T026995/1 entitled "EnnCore: End-to-End Conceptual Guarding of Neural Architectures" under *Security for all in an AI enabled society.* Philipp Kern's work was supported by funding from the pilot program Core-Informatics of the Helmholtz Association (HGF).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption standard. Cryptology ePrint Archive, Paper 2019/939 (2019), https://eprint.iacr.org/2019/939
2. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: A public domain dataset for human activity recognition using smartphones. In: 21st European Symposium on Artificial Neural Networks, ESANN 2013, Bruges, Belgium, April 24-26, 2013 (2013), https://www.esann.org/sites/default/files/proceedings/legacy/es2013-84.pdf
3. Armknecht, F., Boyd, C., Carr, C., Gjøsteen, K., Jäschke, A., Reuter, C.A., Strand, M.: A guide to fully homomorphic encryption. Cryptology ePrint Archive, Paper 2015/1192 (2015), https://eprint.iacr.org/2015/1192
4. Arya, V., Bellamy, R.K.E., Chen, P., Dhurandhar, A., Hind, M., Hoffman, S.C., Houde, S., Liao, Q.V., Luss, R., Mojsilovic, A., Mourad, S., Pedemonte, P., Raghavendra, R., Richards, J.T., Sattigeri, P., Shanmugam, K., Singh, M., Varshney, K.R., Wei, D., Zhang, Y.: AI explainability 360: Impact and design. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022. pp. 12651–12657. AAAI Press (2022). https://doi.org/10.1609/AAAI.V36I11.21540, https://doi.org/10.1609/aaai.v36i11.21540
5. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. SIAM review **59**(1), 65–98 (2017), https://doi.org/10.1137/141000671
6. Bossuat, J.P., Cammarota, R., Chillotti, I., Curtis, B.R., Dai, W., Gong, H., Hales, E., Kim, D., Kumara, B., Lee, C., Lu, X., Maple, C., Pedrouzo-Ulloa, A., Player, R., Polyakov, Y., Lopez, L.A.R., Song, Y., Yhee, D.: Security guidelines for implementing homomorphic encryption. Cryptology ePrint Archive, Paper 2024/463 (2024), https://eprint.iacr.org/2024/463
7. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011. pp. 97–106. IEEE Computer Society (2011). https://doi.org/10.1109/FOCS.2011.12, https://doi.org/10.1109/FOCS.2011.12
8. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6841, pp. 505–524. Springer (2011). https://doi.org/10.1007/978-3-642-22792-9_29, https://doi.org/10.1007/978-3-642-22792-9_29
9. Brix, C., Müller, M.N., Bak, S., Johnson, T.T., Liu, C.: First three years of the international verification of neural networks competition (VNN-COMP). Int. J. Softw. Tools Technol. Transf. **25**(3), 329–339 (2023)
10. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Paper 2017/035 (2017), https://eprint.iacr.org/2017/035

11. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 409–437. Springer International Publishing, Cham (2017)
12. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Dolev, S., Margalit, O., Pinkas, B., Schwarzmann, A. (eds.) Cyber Security Cryptography and Machine Learning. pp. 1–19. Springer International Publishing, Cham (2021)
13. Clenshaw, C.W.: A note on the summation of chebyshev series. Mathematics of Computation **9**, 118–120 (1955). https://doi.org/10.1090/S0025-5718-1955-0071856-0, https://doi.org/10.1090/S0025-5718-1955-0071856-0
14. Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. SIBGRAPI'93, Recife, PE (Brazil), October (1993)
15. Costache, A., Curtis, B.R., Hales, E., Murphy, S., Ogilvie, T., Player, R.: On the precision loss in approximate homomorphic encryption. In: Carlet, C., Mandal, K., Rijmen, V. (eds.) Selected Areas in Cryptography – SAC 2023. pp. 325–345. Springer Nature Switzerland, Cham (2024)
16. Driscoll, T.A., Hale, N., Trefethen, L.N.: Chebfun Guide. Pafnuty Publications (2014), http://www.chebfun.org/docs/guide/
17. Eleftheriadis, C., Kekatos, N., Katsaros, P., Tripakis, S.: On neural network equivalence checking using SMT solvers. In: Bogomolov, S., Parker, D. (eds.) Formal Modeling and Analysis of Timed Systems - 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13-15, 2022, Proceedings. LNCS, vol. 13465, pp. 237–257. Springer (2022). https://doi.org/10.1007/978-3-031-15839-1_14
18. FICO: Fico explainable machine learning challenge (2018), https://community.fico.com/s/explainable-machinelearning-challenge
19. Garimella, K., Jha, N.K., Reagen, B.: Sisyphus: A cautionary tale of using low-degree polynomial activations in privacy-preserving deep learning (2021), https://arxiv.org/abs/2107.12342
20. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
21. Ghorbal, K., Goubault, E., Putot, S.: The zonotope abstract domain taylor1+. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5643, pp. 627–633. Springer (2009). https://doi.org/10.1007/978-3-642-02658-4_47, https://doi.org/10.1007/978-3-642-02658-4_47
22. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 201–210. PMLR, New York, New York, USA (20–22 Jun 2016), https://proceedings.mlr.press/v48/gilad-bachrach16.html
23. Kleine Büning, M., Kern, P., Sinz, C.: Verifying equivalence properties of neural networks with ReLU activation functions. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings. LNCS, vol. 12333, pp. 868–884. Springer (2020). https://doi.org/10.1007/978-3-030-58475-7_50

24. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791
25. Lee, E., Lee, J.W., No, J.S., Kim, Y.S.: Minimax approximation of sign function by composite polynomial for homomorphic comparison. IEEE Transactions on Dependable and Secure Computing **19**(6), 3711–3727 (2022). https://doi.org/10.1109/TDSC.2021.3105111
26. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., No, J.S.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IEEE Access **10**, 30039–30054 (2022). https://doi.org/10.1109/ACCESS.2022.3159694
27. Lee, J., Lee, E., Lee, J.W., Kim, Y., Kim, Y.S., No, J.S.: Precise approximation of convolutional neural networks for homomorphically encrypted data. IEEE Access **11**, 62062–62076 (2023). https://doi.org/10.1109/ACCESS.2023.3287564
28. Manino, E., Magri, B., Mustafa, M., Cordeiro, L.: Certified private inference on neural networks via Lipschitz-guided abstraction refinement. In: Narodytska, N., Amir, G., Katz, G., Isac, O. (eds.) Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems. Kalpa Publications in Computing, vol. 16, pp. 35–46. EasyChair (2023). https://doi.org/10.29007/59w3, /publications/paper/bVbP
29. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference service for neural networks. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 2505–2522. USENIX Association (Aug 2020), https://www.usenix.org/conference/usenixsecurity20/presentation/mishra
30. Obla, S., Gong, X., Aloufi, A., Hu, P., Takabi, D.: Effective activation functions for homomorphic evaluation of deep neural networks. IEEE Access **8**, 153098–153112 (2020). https://doi.org/10.1109/ACCESS.2020.3017436
31. Pachón, R., Trefethen, L.N.: Barycentric-remez algorithms for best polynomial approximation in the chebfun system. BIT Numerical Mathematics **49**, 721–741 (2009). https://doi.org/10.1007/s10543-009-0240-1, https://doi.org/10.1007/s10543-009-0240-1
32. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 223–238. Springer Berlin Heidelberg (1999)
33. Paulsen, B., Wang, J., Wang, C.: ReluDiff: differential verification of deep neural networks. In: Rothermel, G., Bae, D. (eds.) ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020. pp. 714–726. ACM (2020). https://doi.org/10.1145/3377811.3380337
34. Paulsen, B., Wang, J., Wang, J., Wang, C.: NeuroDiff: scalable differential verification of neural networks using fine-grained approximation. In: 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020. pp. 784–796. IEEE (2020). https://doi.org/10.1145/3324884.3416560
35. Pulido-Gaytan, B., Tchernykh, A., Cortés-Mendoza, J.M., Babenko, M., Radchenko, G., Avetisyan, A., Drozdov, A.Y.: Privacy-preserving neural networks with homomorphic encryption: C hallenges and opportunities. Peer-to-Peer Networking and Applications **14**(3), 1666–1691 (2021)
36. Remes, E.: Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. CR Acad. Sci. Paris **198**, 2063–2065 (1934)

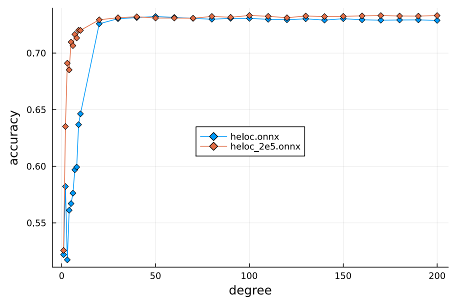
37. Ribeiro, M., Grolinger, K., Capretz, M.A.: Mlaas: Machine learning as a service. In: 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). pp. 896–902 (2015). https://doi.org/10.1109/ICMLA.2015.152
38. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978). https://doi.org/10.1145/359340.359342, https://doi.org/10.1145/359340.359342
39. Rovida, L., Leporati, A.: Encrypted image classification with low memory footprint using fully homomorphic encryption. International journal of neural systems **34**(5), 2450025 (2024). https://doi.org/10.1142/S0129065724500254
40. Shi, Z., Wang, Y., Zhang, H., Kolter, J.Z., Hsieh, C.J.: Efficiently computing local lipschitz constants of neural networks via bound propagation. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 2350–2364. Curran Associates, Inc. (2022), https://proceedings.neurips.cc/paper_files/paper/2022/file/0ff54b4ec4f70b3ae12c8621ca8a49f4-Paper-Conference.pdf
41. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 10825–10836 (2018), https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html
42. Teuber, S., Büning, M.K., Kern, P., Sinz, C.: Geometric path enumeration for equivalence verification of neural networks. In: 33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021. pp. 200–208. IEEE (2021). https://doi.org/10.1109/ICTAI52525.2021.00035
43. Teuber, S., Kern, P., Janzen, M., Beckert, B.: Revisiting differential verification: Equivalence verification with confidence. CoRR **abs/2410.20207** (2024). https://doi.org/10.48550/ARXIV.2410.20207, https://doi.org/10.48550/arXiv.2410.20207
44. Trefethen, L.N.: Approximation theory and approximation practice, extended edition. SIAM (2019)
45. Weng, L., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Daniel, L., Boning, D., Dhillon, I.: Towards fast computation of certified robustness for ReLU networks. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 5276–5285. PMLR (10–15 Jul 2018), https://proceedings.mlr.press/v80/weng18a.html
46. Xiao, K.Y., Tjeng, V., Shafiullah, N.M.M., Madry, A.: Training for faster adversarial robustness verification via inducing relu stability. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=BJfIVjAcKm

A Further Results

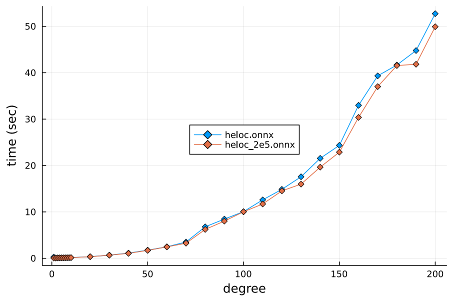
We omitted detailed figures for some benchmarks from the evaluation section, since the results are qualitatively similar to results of other benchmarks.

For completeness, we print the omitted figures for the accuracy of generated polynomial networks in Section A.1 and for the verified difference between generated and original networks in Section A.2.

A.1 Synthesis of Polynomially Approximated Networks

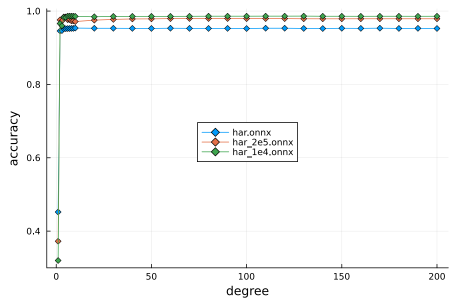


(a) Accuracy for polynomially approximated HELOC networks for polynomials of different degrees and different base ReLU networks.

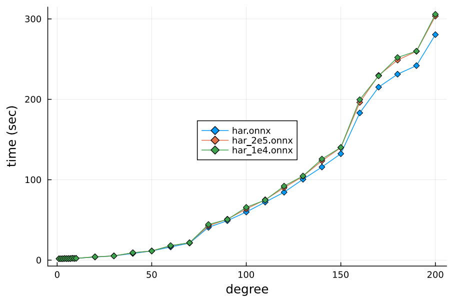


(b) Runtime of ZonoPoly to synthesize polynomially approximated networks for different degrees and base ReLU networks

Fig. 12: Accuracy of runtime of ZonoPoly for different synthesized polynomially approximated networks for HELOC.



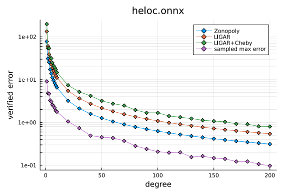
(a) Accuracy for polynomially approximated HAR networks for polynomials of different degrees and different base ReLU networks.



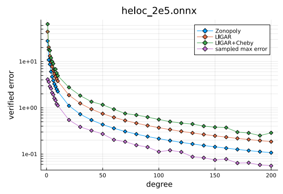
(b) Runtime of ZonoPoly to synthesize polynomially approximated networks for different degrees and base ReLU networks

Fig. 13: Accuracy of runtime of ZonoPoly for different synthesized polynomially approximated networks for HAR.

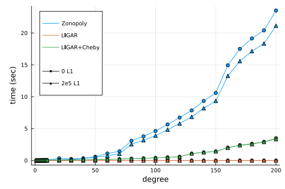
A.2 Verified Error Bounds



(a) no L_1 penalty

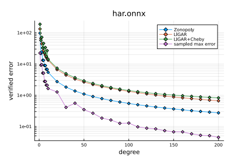


(b) 2×10^5 L_1 penalty

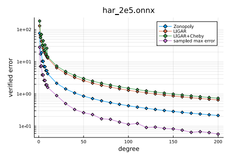


(c) Verification time

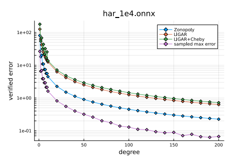
Fig. 14: Verified error bounds for networks trained on the HELOC dataset and runtime for their computation.



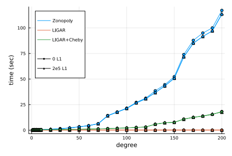
(a) no L_1 penalty



(b) 2×10^{-5} L_1 penalty



(c) 10^{-4} L_1 penalty



(d) Verification time

Fig. 15: Verified error bounds for networks trained on the HAR dataset and runtime for their computation.

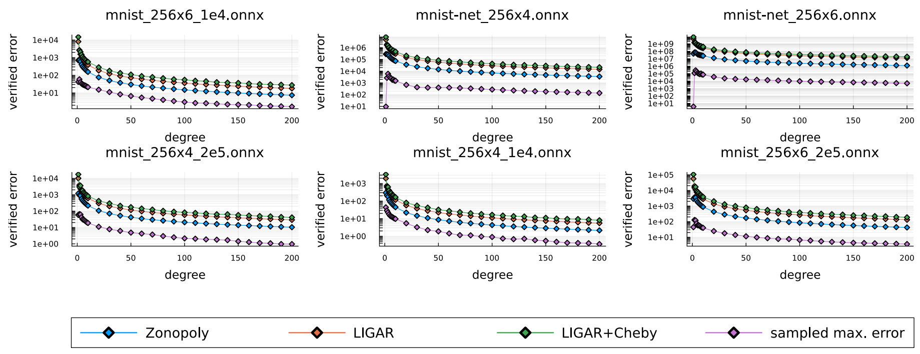


Fig. 16: Verified error bounds for polynomially approximated networks of different degree for different ReLU base networks for MNIST

A somewhat surprising effect which can be seen in Figure 16 is that for the MNIST networks, the output error of the degree-1 (linear) approximations is often smaller than for quadratic polynomials. This is because the zonotopes we use during synthesis of the polynomial networks, can exactly represent linear transformations, resulting in tight bounds for neuron inputs. In contrast, higher-degree approximations introduce overapproximation error, which causes wider input intervals and poorer polynomial fits. While this difference is minor in most networks, it is significant for the MNIST networks from VNN-COMP [9], where the true output ranges are very narrow. As less overapproximation is introduced for linear approximations, they lead to smaller output ranges and thus to smaller absolute output errors. While the output ranges are small for the linear networks, the outputs themselves are not necessarily useful. As shown in Figure 7a, the classification accuracy is still not high.