Formally Verifying Analog Neural Networks With Device Mismatch Variations

Yasmine Abu-Haeyeh[†1], Thomas Bartelsmeier[†2], Tobias Ladner[†3],
Matthias Althoff[3], Lars Hedrich[1], Markus Olbrich[2]
[1] Goethe University Frankfurt, Germany
{abu-haeyeh,hedrich}@em.uni-frankfurt.de
[2] Leibniz University Hannover, Germany
{thomas.bartelsmeier,markus.olbrich}@ims.uni-hannover.de
[3] Technical University of Munich, Germany
{tobias.ladner,althoff}@tum.de

***Abstract*—Training and running inference of large neural networks comes with excessive cost and power consumption. Thus, realizing these networks as analog circuits is an energy- and area-efficient alternative. However, analog neural networks suffer from inherent deviations within their circuits, requiring extensive testing for their correct behavior under these deviations. Unfortunately, tests based on Monte Carlo simulations are extremely time- and resource-intensive. We present an alternative approach to proving the correctness of the neural network using formal neural network verification techniques and developing a modeling methodology for these analog neural circuits. Our experimental results compare two methods based on reachability analysis showing their effectiveness by reducing the test time from days to milliseconds. Thus, they offer a faster, more scalable solution for verifying the correctness of analog neural circuits.**

***Index Terms*—Analog neural networks, formal verification, set-based computing, energy-efficient computing, AI hardware.**

I. Introduction

Artificial neural networks can solve complex problems in a wide range of applications [1]. However, conventional neural networks under von Neumann architectures suffer from excessive computational costs and power consumption [2]. In contrast, analog-based inference systems provide an energy- and area-efficient hardware solution and can effectively execute neural networks without requiring the transmission of data from memories to computing devices [3].

However, these analog circuits are vulnerable to perturbations, such as process variations and device mismatch variations, especially when operating in sub-threshold regions [4]. Such vulnerabilities have also been discussed for conventionally implemented neural networks in the context of adversarial attacks [5]. Thus, the correct behavior under such perturbations has to be tested. For analog neural networks considered in this work, this is mainly done by extensive and time-consuming Monte Carlo simulations [6].

A. Related Work

Analog neural networks have been developed for several decades [3, 7]. We focus on works dealing with the perturbations discussed above: Perturbation-aware design and modeling of analog neural networks is a progressing research field [6, 8, 9, 10]. For example, a MOS transistor model can capture the effect of device mismatch on an analog neuromorphic chip [11], where the model parameters are extracted from Monte Carlo simulations at the circuit level. However, it has not been tested with a full network on real applications. Calibrating an analog neural network processor against process and mismatch variations using multi-resolution weights can increase the inference robustness against such perturbations [12], but the physical behavior of the analog processor is not considered in this work.

Bounding perturbations in neural networks is also discussed in the research field of formal neural network verification [13, 14]. In this field, one always considers a conventionally implemented neural network, and thus only perturbations to the input are usually considered. The exact output of neural networks can then be computed using complete verifiers [15, 16]; however, it has been shown that this problem is NP-hard for ReLU neural networks [16]. Thus, many verifiers enclose the output of the network by relaxing the problem, where both optimization-based approaches [17, 18, 19, 20] and approaches based on reachability analysis [21, 22, 23, 24] are used. These approaches also often use branch-and-bound strategies to deal with the complexity of the problem [25, 26, 27, 28].

B. Contributions

This work presents a formal verification method for analog neural networks. To summarize, our contributions are:

- A detailed description of our analog circuit design combining analog neurons with digitally programmable weights and biases.
- A system-level verification approach to model the non-deterministic behavior of analog neural networks. In particular, device mismatch variations are considered.
- We compare two approaches to formally enclose these variations due to device mismatch, eliminating time-consuming testing using Monte Carlo simulations of the networks: The first approach offers a fast and scalable computation of this enclosure, whereas the second approach obtains tighter enclosures tailored to ReLU activation at the cost of scalability.
- The proposed model and verification approaches are showcased on real-world tasks, including image recognition on the MNIST dataset.

[†] Equal contribution, sorted alphabetically.

II. Preliminaries

A. Notation

We denote scalars and vectors by lowercase letters, matrices by uppercase letters, and sets by calligraphic letters. The i-th element of a vector $v \in \mathbb{R}^n$ is written as $v_{(i)}$. The element in the i-th row and j-th column of a matrix $A \in \mathbb{R}^{n \times m}$ is written as $A_{(i,j)}$, the entire i-th row and j-th column are written as $A_{(i,\cdot)}$ and $A_{(\cdot,j)}$, respectively. The concatenation of A with a matrix $B \in \mathbb{R}^{n \times o}$ is denoted by $[A\ B] \in \mathbb{R}^{n \times (m+o)}$. We denote the element-wise multiplication of two vectors by $\odot$. The symbol $\mathbf{1}$ refers to the matrix with all ones of proper dimensions. Given $n \in \mathbb{N}$, we use the shorthand notation $[n] = \{1, \dots, n\}$. Let $\mathcal{S} \subset \mathbb{R}^n$ be a set and $f \colon \mathbb{R}^n \to \mathbb{R}^m$ be a function, then $f(\mathcal{S}) = \{f(x) \mid x \in \mathcal{S}\}$. An interval with bounds $a, b \in \mathbb{R}^n$ is denoted by $[a, b]$, where $a \leq b$ holds element-wise.

B. Neural Networks

We consider feed-forward neural networks with ReLU activation in this work:

Definition 1 (ReLU Neural Network [29, Sec. 5.1]). *Given an input $x \in \mathbb{R}^{n_0}$ and an output $y \in \mathbb{R}^{n_\kappa}$, a neural network $y = \Phi(x)$ with $\kappa \in \mathbb{N}$ layers can be formulated as*

$$\begin{aligned} h_0 &= x, \\ h_k &= L_k(h_{k-1}) = \mathrm{ReLU}(W_k h_{k-1} + b_k), \quad k \in [\kappa], \\ y &= h_\kappa, \end{aligned}$$

with $L_k \colon \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$, $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$, and $b_k \in \mathbb{R}^{n_k}$.

C. Set-Based Computing

We use continuous sets to verify neural networks. In particular, we use zonotopes and affine arithmetic decision diagrams (AADD) throughout this work. Let us start by formally introducing zonotopes:

Definition 2 (Zonotope [30, Def. 1]). *Given a center vector $c \in \mathbb{R}^n$ and a generator matrix $G \in \mathbb{R}^{n \times p}$, a zonotope is defined as*

$$\mathcal{Z} = \langle c, G \rangle_Z = \left\{ c + \sum_{i=1}^{p} \beta_i G_{(\cdot,i)} \;\middle|\; \beta_i \in [-1, 1] \right\}.$$

Let us also define the required operations on zonotopes: Given $\mathcal{Z} = \langle c, G \rangle_Z \subset \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$, the affine map is computed by [31, Eq. 2.1]

$$A\,\mathcal{Z} + b = \{Ax + b \mid x \in \mathcal{Z}\} = \langle A\,c + b, A\,G \rangle_Z \,. \tag{1}$$

The Minkowski sum of a zonotope and an interval $\mathcal{I} = [l, u]$ is computed by [31, Prop. 2.1 and Eq. 2.1]:

$$\begin{aligned} \mathcal{Z} \oplus \mathcal{I} &= \{x_1 + x_2 \mid x_1 \in \mathcal{Z},\, x_2 \in \mathcal{I}\} \\ &= \langle c + 1/2(u + l), [G\ \mathrm{diag}\,(1/2(u - l))] \rangle_Z \,. \end{aligned} \tag{2}$$

The enclosing interval $[l, u] \supseteq \mathcal{Z}$ is computed by [31, Prop. 2.2]

$$\begin{aligned} l &= c - \Delta g, \\ u &= c + \Delta g, \end{aligned} \quad \text{with } \Delta g = \sum_{i=1}^{p} |G_{(\cdot,i)}|. \tag{3}$$

Let us also formally introduce the affine arithmetic decision diagrams (AADD) [32] representing a set as the union of constrained affine forms, which are effectively (constrained) zonotopes with $n = 1$ with shared noise symbols ϵ [33].

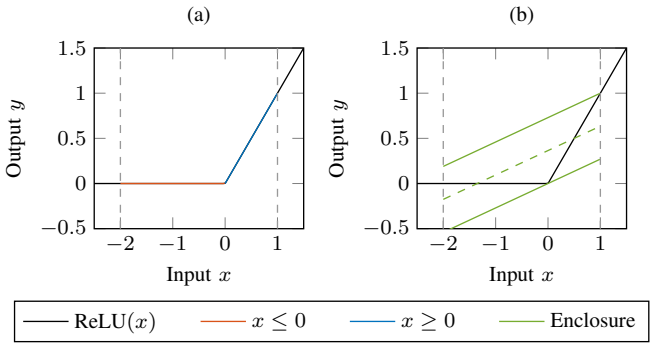

Fig. 1: Enclosure of a ReLU neuron: Splitting into piecewise linear parts (a) and using zonotopes (b).

Definition 3 (AADD [32]). *An AADD is a binary decision tree specified by the tuple $(\mathcal{N}_I, \mathcal{N}_L, v_r, \mathcal{E}_T, \mathcal{E}_F)$ with internal nodes $\mathcal{N}_I$, leaf nodes $\mathcal{N}_L$, root node $v_r \in \mathcal{N}_I \cup \mathcal{N}_L$, true-edges $\mathcal{E}_T$, false-edges $\mathcal{E}_F$, and it holds:*

- *Each internal node $v_i \in \mathcal{N}_I$ is associated with an affine form $\mathcal{AF}_i$, which defines two constraints in the ϵ-space of $\mathcal{AF}_i$ such that $\mathcal{AF}_{i,t} = \{x \in \mathcal{AF}_i \mid x \geq 0\}$ and $\mathcal{AF}_{i,f} = \mathcal{AF}_i \setminus \mathcal{AF}_{i,t}$. An internal node has two leaving edges $e_t \in \mathcal{E}_T$ and $e_f \in \mathcal{E}_F$ that lead to child nodes $v_t, v_f \in \mathcal{N}_I \cup \mathcal{N}_L$, where each constraint is further considered, respectively.*
- *Each leaf node $v_l \in \mathcal{N}_L$ contains an affine form $\mathcal{AF}_l$ that corresponds to a subset of the ϵ-hypercube, cut by the conjunction of constraints on the path from v_r to v_l.*

We refer to [32, 33] for the exact operations on AADDs. Intuitively, as they are based on constrained affine forms, most operations are analogous to the ones on zonotopes (1)-(3). The only difference is its ability to impose constraints on the noise symbols ϵ to model the conditions $x \geq 0$ and $x < 0$, respectively.

D. Formal Verification of Neural Networks

Uncertainties require us to evaluate a neural network (Def. 1) set-based. In particular, for an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$, the exact output sets of each layer are given by

$$\mathcal{H}_0^* = \mathcal{X}, \quad \mathcal{H}_k^* = L_k\left(\mathcal{H}_{k-1}^*\right), \quad \mathcal{Y}^* = \mathcal{H}_\kappa^*, \quad k \in [\kappa]. \tag{4}$$

Unfortunately, these exact sets are often not obtainable in practice as the problem is NP-hard [16]. Thus, we enclose the output of each layer:

Proposition 1 (Neural Network Enclosure [22, Sec. 3]). *Given an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$ to a neural network Φ, we obtain an enclosure of the output set $\mathcal{Y} = \texttt{enclose}(\Phi, \mathcal{X}) \supseteq \mathcal{Y}^*$ by iteratively enclosing the output of each layer:*

$$\mathcal{H}_k = \texttt{enclose}\left(L_k, \mathcal{H}_{k-1}\right) \supseteq \mathcal{H}_k^*, \quad k \in [\kappa],$$

with $\mathcal{H}_0 = \mathcal{X}$ and $\mathcal{Y} = \mathcal{H}_\kappa$.

The linear part of a neural network (Def. 1) can be computed exactly without additional outer approximations with zonotopes using (1). However, the ReLU activation needs to be enclosed if the input set to a layer intersects with both linear parts (Fig. 1): Given the bounds of our input set, we can find a polynomial approximating the activation function and bound the approximation error (Fig. 1b), where the error is added

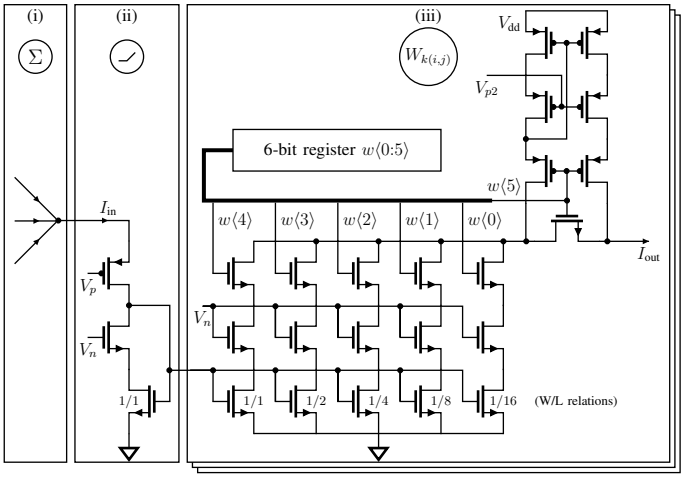

Fig. 2: Circuit schematic of the elements of the analog neuron circuit: (i) Summation block based on Kirchhoff's law, (ii) ReLU activation function, and (iii) weight cells. The programming bits are stored in a 6-bit register. The ReLU and weight blocks establish a binary weighted ($w\langle 0{:}4\rangle$) current mirror. The bit $w\langle 5\rangle$ is used to realize negative weights by inverting the output current.

to the set using (2). In contrast, the exact output set can be computed for networks with ReLU activation by splitting the input set at 0 and considering each subset individually in subsequent layers (Fig. 1a). The final output set is then obtained by splitting the input set at each neuron while propagating them through the entire network and computing the union of the output sets. We maintain all of these subsets using an AADD (Def. 3), where the leaf nodes correspond to subsets in the output space of the network. In the worst case, this approach based on AADDs suffers from an exponential number of subsets in the number of ReLU neurons of the network [16].

III. Analog Neural Network Implementation

In this section, we present our neural network inference circuit based on a sub-threshold 130nm, 1.2V CMOS design [34]. Our design combines analog neurons with digital programmable weights and biases. All currents within the circuit (input, output, and internal) are in the nA-range, making it extremely energy-efficient.

A. Circuit Composition

The analog neuron is the main building block modeling a combined ReLU activation with an appended linear layer. An overview of the neuron circuit schematic is shown in Fig. 2. It consists of three components: (i) a summation point and (ii) a ReLU cell with (iii) a combined weight current mirror. First, all weighted input currents and a bias current are summed utilizing the basic Kirchhoff's circuit laws [35]. Secondly, a ReLU activation is applied by using the first part of a current mirror in the subthreshold region to cut off negative currents. We need two cascode transistors here to implement a low-voltage cascode current mirror and ensure a constant voltage at the summation nodes. Thirdly, the second part of the current mirror scales the output current using a 5-bit weight $w\langle 0{:}4\rangle$ (see W/L relations in Fig. 2 (iii)). The sixth bit $w\langle 5\rangle$ is used for the sign by inverting the current with another current mirror. A bias circuit not shown in Fig. 2 maps an 8-bit quantized programmable bias into an analog current. The circuit receives

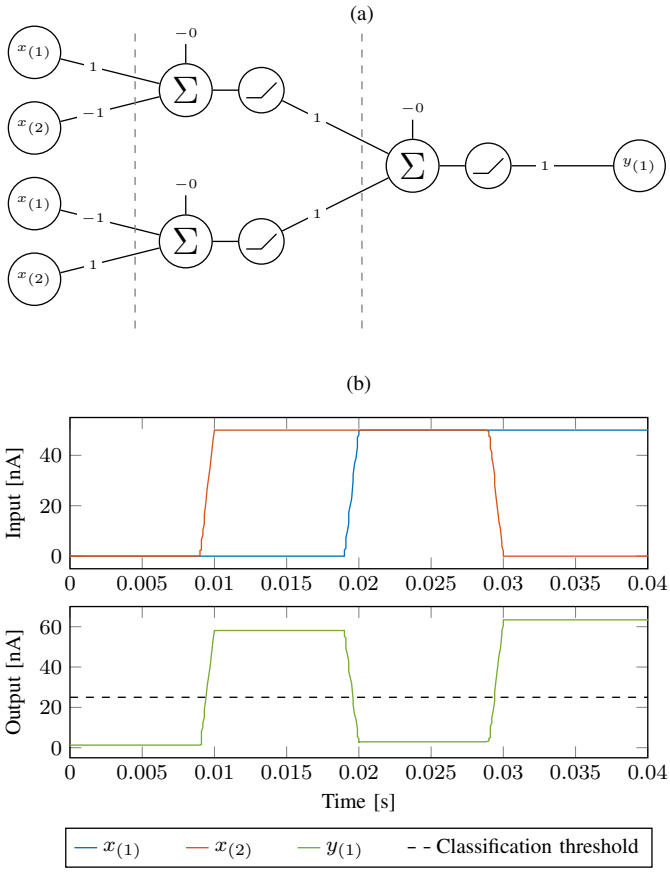

Fig. 3: Running example computing the XOR function: (a) Neural network and (b) example in- and output. The main blocks of (a) are shown in Fig. 2 and all weights and biases are rounded to their nearest integer.

input currents in the range of $[-200\text{nA}, 200\text{nA}]$ and output currents in the range of $[0\text{nA}, 300\text{nA}]$.

B. Transferring TensorFlow Models

The network is trained off-chip using TensorFlow [36]. The weights of the trained network are quantized to the six available bits, where 50nA in the inference circuit corresponds to a value of 1 in TensorFlow [34]. Please note that the circuit consists of a combined ReLU and linear layer (Fig. 2). This requires us to make adaptations to the input and the output of a TensorFlow network of the form as defined in Def. 1: At the input of the network, a slightly different circuit is used to scale the input signal by the weight value without applying the ReLU function to maintain the neural network functionality for negative input signals. At the output of the network, an additional layer with a weight of 1 is required to apply the final ReLU activation.

A simple example is depicted in Fig. 3a to demonstrate the presented inference circuit. This neural network was trained to mimic an XOR function. We indicate with dashed lines how this two-layer TensorFlow network is parsed into a three-layered circuit: In the first layer, the input currents $x_{(1)}$ and $x_{(2)}$ are scaled by the weight W_1. In the second layer, all weighted currents and the bias current corresponding to b_1 are summed together, the ReLU activation is applied, and the output currents are scaled according to W_2. This is done analogously for the final layer, where a weight of $W_3 = 1$ is used at the end. The input and output of the network are shown in Fig. 3b, where the

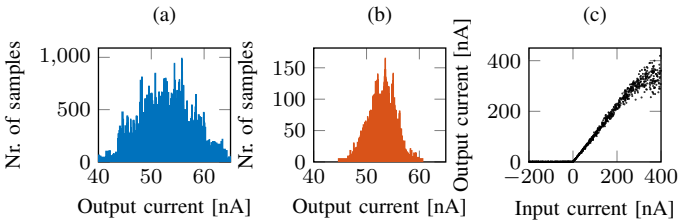

Fig. 4: Monte Carlo analysis distribution of the neuron netlist to mismatch variations: (a) mismatch in the input neuron, (b) mismatch in hidden and output neurons, and (c) output of the neuron impacted by mismatch.

input signal changes every 10ms, and the output current quickly expresses a correct classification using a threshold of 25nA.

IV. Formally Verifying Analog Neural Networks

To capture the effect of device mismatch variations on the performance of each layer in the circuit, we formally verify the output of the circuit under these variations.

A. Uncertainties in Analog Neural Networks

Based on the knowledge acquired from the behavior of the analog neuron netlist explained in Sec. III, we develop a mathematical model for the output of the neural network considering these uncertainties: Monte Carlo simulations (Fig. 4) show (a) a high device mismatch uncertainty at the input neuron due to the adapted circuit design with more transistors, but the network exhibits (b) a lower uncertainty for hidden and output neurons. Additionally, please note that (c) this uncertainty scales linearly with the output of the neuron up to the saturation point at 300nA.

Based on these observations and the circuit design (Fig. 2), we model the non-deterministic output $\widetilde{y}$ of an analog neural network for an input x as:

$$\begin{aligned}
\widetilde{h}_0 &= x,\\
\widetilde{h}'_1 &= L'_1(\widetilde{h}_0) = W_1\widetilde{h}_0 + b_1,\\
\widetilde{h}_1 &= \widetilde{h}'_1 \odot (\mathbf{1} + \alpha\epsilon_1),\\
\widetilde{h}'_{k'} &= L'_{k'}(\widetilde{h}_{k'-1}) = W_{k'}\mathrm{ReLU}(\widetilde{h}_{k'-1}) + b_{k'},\\
\widetilde{h}_{k'} &= \widetilde{h}'_{k'} \odot (\mathbf{1} + \beta\epsilon_{k'}), \qquad k' \in \{2,\dots,\kappa\},\\
\widetilde{y} &= \widetilde{h}_\kappa,
\end{aligned} \tag{5}$$

where the noise symbol $\epsilon_k \in [-1,1]^{n_k}$, $k \in [\kappa]$, reflects the model uncertainties of each neuron of that layer [37], the parameter $\alpha \in \mathbb{R}$ reflects the high sensitivity of the first layer ($k = 1$) to the device mismatch, and the parameter $\beta < \alpha \in \mathbb{R}$ reflects the lower sensitivity of each subsequent layer ($k > 1$). Please compare the computation of $\widetilde{y}$ to the nominal output y in Def. 1.

As the Monte Carlo simulations approximately follow a normal distribution (Fig. 4a and 4b), we can estimate the standard deviation

$$\sigma = \sqrt{\left(\textstyle\sum_{i=1}^{n_s}(\widetilde{y}_i - \overline{y})\right) / (n_s - 1)} \tag{6}$$

given $n_s \in \mathbb{N}$ Monte Carlo sample outputs $\widetilde{y}_i$, $i \in [n_s]$, with mean $\overline{y}$. This gives us the ability to formally enclose the outputs using a third user-defined parameter $\gamma \in \mathbb{R}$ specifying the range $[y - \gamma,\, y + \gamma]$ we want to enclose, where e.g., setting $\gamma = \sigma$ encloses $\sim$68% of the uncertainty and $\gamma = 3\sigma$ encloses $\sim$99%.

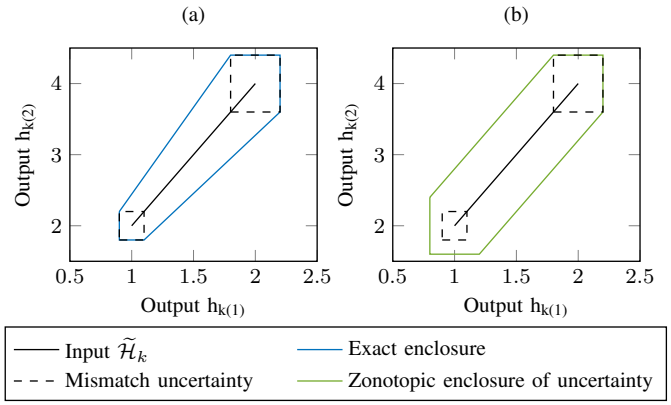

Fig. 5: Enclosure of mismatch: (a) Exact and (b) using zonotopes.

Moreover, we say that the model is conformant if γ is chosen large enough such that all measurements are contained. Given γ, the internal parameters α and β are then minimized using the generalized reduced gradient method [38].

B. Bounding the Uncertainties

Given our developed model, we can present the required adaptations of Prop. 1 to enclose the mismatch uncertainty given in (5) using set-based computing. We present two approaches here based on zonotopes (Def. 2) and AADDs (Def. 3), respectively:

1) Enclosure using zonotopes: For an input set $\widetilde{\mathcal{H}}_{k-1} \subset \mathbb{R}^{n_{k-1}}$, we compute $\widetilde{\mathcal{H}}'_k := \texttt{enclose}(L'_k, \widetilde{\mathcal{H}}_{k-1})$ using Prop. 1. To also bound the added uncertainty in (5), let us first consider the case $k = 1$: We need to compute $\widetilde{\mathcal{H}}'_k \odot (\mathbf{1} + \alpha\epsilon_k)$ to obtain the output of the entire layer $\widetilde{\mathcal{H}}_k$. Unfortunately, this operation cannot be computed exactly using zonotopes. Thus, we bound the error by the worst-case uncertainty:

$$\begin{aligned}
\widetilde{\mathcal{H}}'_k \odot (1 + \alpha\epsilon_k) &= \widetilde{\mathcal{H}}'_k \oplus \widetilde{\mathcal{H}}'_k \alpha\epsilon_k\\
&\overset{(3)}{\subseteq} \widetilde{\mathcal{H}}'_k \oplus [l_k,\, u_k]\alpha\epsilon_k\\
&\overset{\epsilon_k \in [-\mathbf{1},\mathbf{1}]}{=} \widetilde{\mathcal{H}}'_k \oplus [-\alpha u_k,\, \alpha u_k] \overset{(2)}{=:} \widetilde{\mathcal{H}}_k.
\end{aligned} \tag{7}$$

A comparison of this enclosure of the mismatch uncertainty using zonotopes with the exact enclosure is shown in Fig. 5. The same holds for $k > 1$ by replacing α with β (5). Please note that this enclosure and the enclosure of the ReLU layer induce outer approximations such that we can guarantee that at least all points within the desired threshold γ are enclosed.

2) Enclosure using AADDs: Using our second approach, each neuron of a network is modeled as an AADD (Def. 3) with shared noise symbols across all neurons in the network. For a layer $k \in [\kappa]$ and neuron $i \in [n_k]$, this corresponds to the i-th dimension of $\widetilde{\mathcal{H}}_k$. While enclosing each layer k (Prop. 1), affine transformations in linear layers can be computed using (1). However, we are missing the implementation of the ReLU activation. As ReLUs are piece-wise linear, we can split the input set at 0 by adding a single constraint in the shared ϵ-space of the AADDs for each case. Thus, the AADD is split into true and false subtrees, which are maintained via the parent node. Depending on the constraints in the AADD before the operation is applied, the resulting AADD may be simplified to mitigate the path explosion problem to an extent.

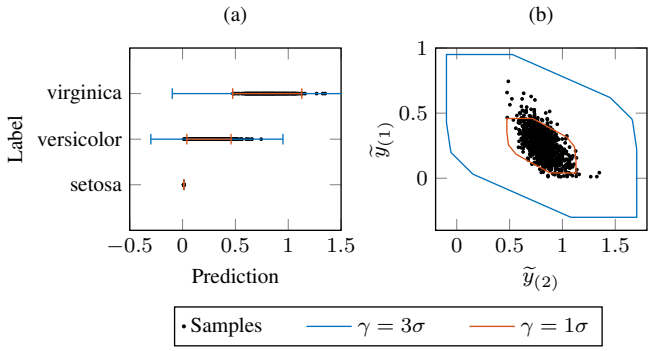

Fig. 6: Zonotopic enclosure of Iris example: (a) Projection of all dimensions and (b) comparison of two most significant dimensions, where the samples are computed using Monte Carlo simulations.

TABLE I: Time comparison and average enclosure of Monte Carlo simulations of considered approaches.

Dataset	Approach	Time [s]	Enclosure of simulations $\gamma = 1\sigma$	Enclosure of simulations $\gamma = 3\sigma$
Iris	Cadence	7,935	-	-
	Zonotope	0.265	88.66%	99.99%
	AADD	0.181	87.83%	99.78%
MNIST	Cadence	162,815	-	-
	Zonotope	0.763	93.71%	99.99%
	AADD	0.444	93.69%	99.94%

To bound the added uncertainty in (5), we multiply the output set $\widetilde{\mathcal{H}}'_{k(i)}$ of each neuron with a new AADD consisting only of a leaf node. For $k = 1$ and $i \in [n_k]$, this is given by

$$\mathcal{AF}^*_{k,i} = \left\{1 + \alpha\epsilon^*_{k,i} \mid \epsilon^*_{k,i} \in [-1, 1]\right\}. \tag{8}$$

Analogous is done for $k > 1$ using β. Notably, we use a different noise symbol $\epsilon^*_{k,i}$ for each neuron such that the uncertainties from different neurons are uncorrelated. Since the AADD uses affine arithmetic for computations, the nonlinear multiplication $\mathcal{AF}^*_{k,i} \cdot \widetilde{\mathcal{H}}'_{k(i)}$ is not exact and is bounded by the worst-case uncertainty analogous to (7) (Fig. 5).

V. Experimental Results

In this section, we compare the effectiveness of our approaches to Monte Carlo simulations. The Monte Carlo simulations were run in Cadence Spectre [39] on a machine equipped with two Intel Xeon E5-2683 v4 CPUs (16 cores each) operating at default settings and 256 GB of RAM. In contrast, both of our approaches were run on a home computer with an AMD Ryzen 7 5800X3D CPU operating at default settings and 32 GB of RAM. The first approach based on zonotopes is written in MATLAB using the toolbox CORA [40], and the second approach based on AADDs is written in Kotlin using the jAADD library [41]. Our approaches are evaluated on the Iris [42] and MNIST [43] datasets.

A. Iris

The Iris dataset [42] is a classification task with three classes: Setosa, Versicolour, and Virginica. We trained a ReLU neural network in TensorFlow with Adam optimizer with 4 inputs, a fully connected layer with 8 hidden neurons, and an output layer with 3 neurons representing the three classes. The network achieves an accuracy of 92% on the test set consisting of 75 input patterns.

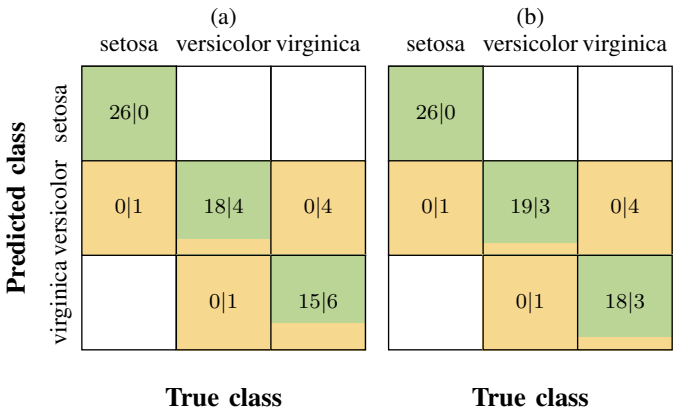

Fig. 7: Confusion matrix for Iris using the (a) zonotope approach and (b) AADD approach: Number of predictions with verifiably correct outputs (green). Wrong predictions and predictions with too much variance are shown in yellow.

We evaluated our approaches to predict the output of the inference circuit under mismatch variations and compared them with the Monte Carlo simulations using $\gamma = 1\sigma$ and $\gamma = 3\sigma$. In Fig. 6, we show the results using the first approach based on zonotopes on a single input pattern. The projection of all dimensions is presented in Fig. 6a, where the bounds of the computed output zonotope are shown as intervals for each class. In Fig. 6b, we present the projected enclosure of the two most significant dimensions, which matches our expectations for the chosen γ value. In particular, $\gamma = 3\sigma$ leads to an enclosure of all 1,000 Monte Carlo samples for this input pattern, whereas choosing $\gamma = 1\sigma$ encloses 91% of the samples. Please note that the projections in Fig. 6 do not provide the full picture of the higher-dimensional output and, thus, its enclosures.

The results on all 75 input patterns are shown in Tab. I and Fig. 7: Fig. 7 shows two confusion matrices for both approaches, respectively, where we color entries of the main diagonal in green if the prediction is verifiably correct under the mismatch variations and the considered threshold γ. The prediction is verifiably correct if the lower bound of the dimension corresponding to the true class is larger than the upper bound of all other classes. Thus, in Fig. 6a, the prediction is verifiably correct for $\gamma = \sigma$ as the orange intervals do not overlap, whereas the predictions can not be verified for $\gamma = 3\sigma$ as the blue intervals overlap. Whenever these intervals overlap, or the prediction is wrong entirely, we color the respective cell in Fig. 7 in yellow. The outputs can be verified for most input patterns using both approaches, where the AADD method is able to verify a few more input patterns (Fig. 7b).

We additionally show the required time to compute the respective enclosures in Tab. I: While the computation of the Monte Carlo simulations takes hours for Iris and even days for MNIST on a server running the Cadence Spectre simulator, the enclosures with either approach can be computed in under a second on a home computer. This shows the effectiveness of our approaches. However, it is important to note that our proposed modeling approach requires running Monte Carlo simulations once on a very small network with two connected neuron circuits to extract the α and β parameters based on the chosen threshold γ. This process on this simple

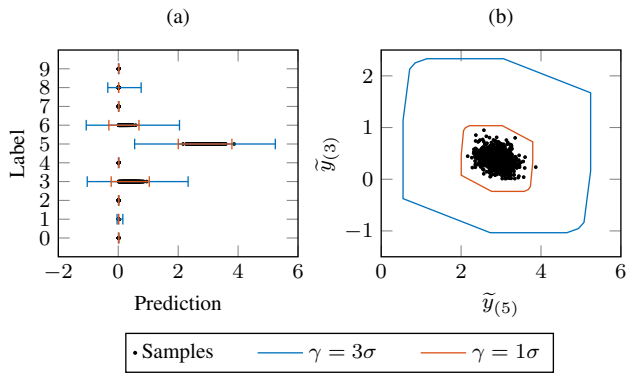

Fig. 8: Zonotopic enclosure of MNIST example: (a) Projection of all dimensions and (b) comparison of two most significant dimensions.

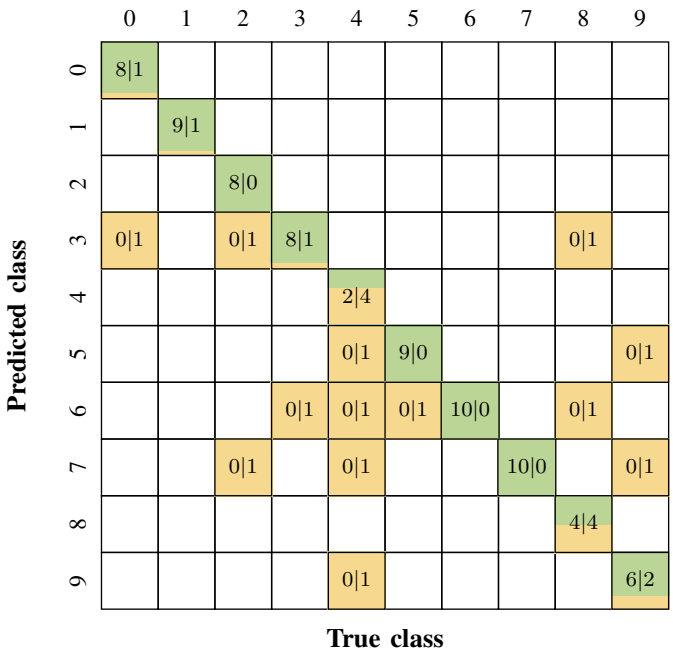

Fig. 9: Confusion matrix for MNIST: Number of predictions with verifiably correct outputs (green). Wrong predictions and predictions with too much variance are shown in yellow.

network requires 75.3s and the extracted parameters can be reused on any network architecture based on dense layers and ReLU activations. The developed model is then automatically generated to be utilized for formal verification, which provides a more efficient verification approach for deeper networks.

B. MNIST

For the MNIST dataset [43] consisting of handwritten digits, we trained a ReLU neural network consisting of fully connected layers with 10 hidden neurons and 10 outputs representing each digit class. We use a reduced input size of 14×14 to reduce the inference circuit complexity and power consumption while maintaining a fairly good inference accuracy of 87% over 100 input patterns.

We again refer to Tab. I for the overall comparison of the enclosures using our approaches as well as the heavily reduced verification time, which shows the scalability of our approaches compared to running Monte Carlo simulations as these take days for this network. Additionally, we show the bounding

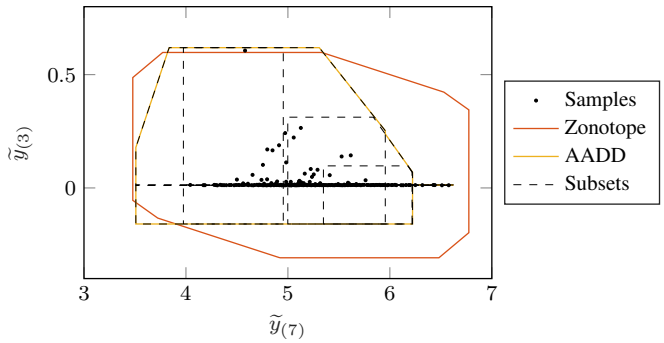

Fig. 10: Comparison of enclosures using zonotopes (orange) and AADD for the MNIST example: The computed subsets of AADD are shown in dashed lines and its union in yellow.

boxes for each dimension as well as the projected enclosure of the two most significant dimensions in Fig. 8. The confusion matrix for the approach using zonotopes is shown in Fig. 9 with very similar results using AADD. Moreover, we provide a detailed comparison of our two approaches in Fig. 10: Please recall that the output set of the AADD is computed by the union of the computed subsets in the leaf nodes of the AADD. These subsets are shown in black dashed lines, and the union is visualized in yellow. Due to the recursive splitting of the AADD tree, it can obtain tighter results than the single output set computed using zonotopes (orange). However, this is only possible for relatively small networks as, in the worst case, the number of leaf nodes is exponential in the number of neurons of the network [16]. For the analog neural networks considered in this work, the method is still feasible and better verification results are obtained than with the zonotopes (Fig. 7 and 10).

VI. Conclusion

In this paper, we present an efficient formal verification approach for analog neural networks with ReLU activation. We have introduced two approaches based on set-based computing to model the uncertainty in analog inference circuits due to mismatch variations: Both approaches formally enclose the output of each layer and also provably enclose the uncertainty up to a desired threshold. By choosing the threshold large enough so that all measurements are enclosed, the conformance of the model can be shown. Whereas traditional testing of the analog circuits can take days based on resource-intensive Monte Carlo simulations, our approach shrinks the testing time to under a second on a home computer. This is demonstrated on two datasets and two different network architectures. Both approaches realize a fast enclosure of the output of analog neural networks under mismatch variations. As some input patterns can not be verified, this indicates that the circuit has to be improved to guarantee the correct output. As an improvement in the circuit design can again be verified quickly, our approach also heavily reduces the design cycle of analog neural circuits. Future work will address extending our proposed approach to different network architectures including convolution layers and different activation functions.

Acknowledgment

The authors gratefully acknowledge financial support by the project FAI (No. 286525601) funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG).

REFERENCES

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, 2018.

[2] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09, 2020, pp. 13 693–13 696.

[3] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, 2010.

[4] P. G. Drennan and C. C. McAndrew, "Understanding mosfet mismatch for analog design," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, pp. 450–456, 2003.

[5] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

[6] Z. Yan, X. S. Hu, and Y. Shi, "Computing-in-memory neural network accelerators for safety-critical systems: Can small device variations be disastrous?" in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[7] T. Mohaidat and K. Khalil, "A survey on neural network hardware accelerators," *IEEE Transactions on Artificial Intelligence*, 2024.

[8] N. Ye, L. Cao, L. Yang, Z. Zhang, Z. Fang, Q. Gu, and G.-Z. Yang, "Improving the robustness of analog deep neural networks through a Bayes-optimized noise injection approach," *Communications Engineering*, vol. 2, no. 25, 2023.

[9] M. J. Rasch, C. Mackin, M. Le Gallo, A. Chen, A. Fasoli, F. Odermatt, N. Li, S. R. Nandakumar, P. Narayanan, H. Tsai *et al.*, "Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators," *Nature communications*, vol. 14, no. 1, p. 5282, 2023.

[10] A. S. Rekhi, B. Zimmer, N. Nedovic, N. Liu, R. Venkatesan, M. Wang, B. Khailany, W. J. Dally, and C. T. Gray, "Analog/mixed-signal hardware error modeling for deep learning inference," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[11] B. V. Benjamin, R. L. Smith, and K. A. Boahen, "An analytical mos device model with mismatch and temperature variation for subthreshold circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 6, pp. 1826–1830, 2023.

[12] K. Jia, Z. Liu, Q. Wei, F. Qiao, X. Liu, Y. Yang, H. Fan, and H. Yang, "Calibrating process variation at system level with in-situ low-precision transfer learning for analog neural network processors," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[13] C. Brix, S. Bak, C. Liu, and T. T. Johnson, "The fourth international verification of neural networks competition (VNN-COMP 2023): Summary and results," in *arXiv preprint arXiv:2312.16760*, 2023.

[14] D. Manzanas Lopez, M. Althoff, M. Forets, T. T. Johnson, T. Ladner, and C. Schilling, "ARCH-COMP23 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants," in *EPiC Series in Computing*, 2023.

[15] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*, 2017, pp. 3–29.

[16] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*, 2017, pp. 97–117.

[17] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[18] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, and C. Barret, "The Marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer Aided Verification*, 2019, pp. 443–452.

[19] P. Henriksen and A. Lomuscio, "Efficient neural network verification via adaptive refinement and adversarial search," in *European Conference on Artificial Intelligence*, 2020, vol. 325, pp. 2513–2520.

[20] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," in *Proceedings of the ACM on Programming Languages*, vol. 3, 2019, pp. 1–30.

[21] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI2: Safety and robustness certification of neural networks with abstract interpretation," in *IEEE Symposium on Security and Privacy*, 2018, pp. 3–18.

[22] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[23] D. M. Lopez, S. W. Choi, H.-D. Tran, and T. T. Johnson, "NNV 2.0: The neural network verification tool," in *International Conference on Computer Aided Verification*, 2023, pp. 397–412.

[24] T. Ladner and M. Althoff, "Automatic abstraction refinement in neural network verification using sensitivity analysis," in *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, 2023, pp. 1–13.

[25] R. Bunel, I. Turkaslan, P. Torr, M. P. Kumar, J. Lu, and P. Kohli, "Branch and bound for piecewise linear neural network verification," in *Journal of Machine Learning Research*, vol. 21, 2020, pp. 1–39.

[26] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification," in *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[27] C. Ferrari, M. N. Mueller, N. Jovanović, and M. Vechev, "Complete verification via multi-neuron relaxation guided branch-and-bound," in *International Conference on Learning Representations*, 2022.

[28] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, 2018, pp. 5777–5783.

[29] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, 2006, vol. 4.

[30] A. Girard, "Reachability of uncertain linear systems using zonotopes," in *International Workshop on Hybrid Systems: Computation and Control*, 2005, pp. 291–305.

[31] M. Althoff, "Reachability analysis and its application to the safety assessment of autonomous cars," Ph.D. dissertation, Technical University of Munich, 2010.

[32] C. Zivkovic, C. Grimm, M. Olbrich, O. Scharf, and E. Barke, "Hierarchical verification of ams systems with affine arithmetic decision diagrams," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1785–1798, 2018.

[33] J. K. Scott, D. M. Raimondo, G. R. Marseglia, and R. D. Braatz, "Constrained zonotopes: A new tool for set-based estimation and fault detection," *Automatica*, vol. 69, pp. 126–136, 2016.

[34] F. Aul, N. Katsaouni, L. Krischker, S. Schmalhofer, M. H. Schulz, and L. Hedrich, "Schematic generation of programmable analog neural networks for signal proccessing," in *International Conference on SMACD and 16th Conference on PRIME*, 2021, pp. 1–4.

[35] T. P. Xiao, C. H. Bennett, B. Feinberg, S. Agarwal, and M. J. Marinella, "Analog architectures for neural network acceleration based on non-volatile memory," *Applied Physics Reviews*, vol. 7, no. 3, 2020.

[36] M. Abadi, A. Agarwal, P. Barham, and et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[37] J. Stolfi and L. H. de Figueiredo, "An introduction to affine arithmetic," *Trends in Computational and Applied Mathematics*, vol. 4, no. 3, pp. 297–312, 2003.

[38] L. S. Lasdon, R. L. Fox, and M. W. Ratner, "Nonlinear optimization using the generalized reduced gradient method," *Revue française d'automatique, informatique, recherche opérationnelle. Recherche opérationnelle*, vol. 8, no. V3, pp. 73–103, 1974.

[39] I. Cadence Design Systems, "Cadence virtuoso spectre simulator datasheet," www.cadence.com, 2024.

[40] M. Althoff, "An introduction to CORA 2015," in *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015, pp. 120–151.

[41] C. Grimm and C. Zivkovic, "jaadd," https://github.com/tukcps/jAADD, 2017, accessed: 2024-09-10.

[42] R. A. Fisher, "Iris," UCI Machine Learning Repository, 1936.

[43] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," http://yann.lecun.com/exdb/mnist, 2010.