Automated Design of Linear Bounding Functions for Sigmoidal Nonlinearities in Neural Networks

Matthias König[1]*, Xiyue Zhang[2] (✉), Holger H. Hoos[1,3], Marta Kwiatkowska[2], and Jan N. van Rijn[1]

[1] Leiden University, The Netherlands
{h.m.t.konig,j.n.van.rijn}@liacs.leidenuniv.nl
[2] University of Oxford, United Kingdom
{xiyue.zhang,marta.kwiatkowska}@cs.ox.ac.uk
[3] RWTH Aachen University, Germany
hh@aim.rwth-aachen.de

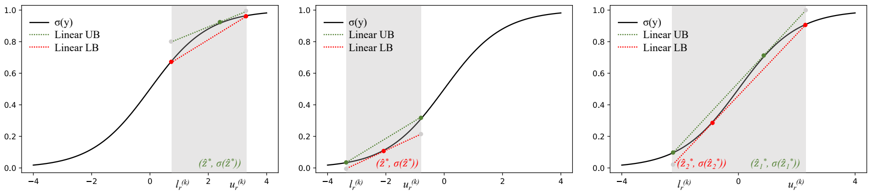
Abstract. The ubiquity of deep learning algorithms in various applications has amplified the need for assuring their robustness against small input perturbations such as those occurring in adversarial attacks. Existing *complete* verification techniques offer provable guarantees for all robustness queries but struggle to scale beyond small neural networks. To overcome this computational intractability, *incomplete* verification methods often rely on convex relaxation to over-approximate the nonlinearities in neural networks. Progress in tighter approximations has been achieved for piecewise linear functions. However, robustness verification of neural networks for general activation functions (*e.g.*, Sigmoid, Tanh) remains under-explored and poses new challenges. Typically, these networks are verified using convex relaxation techniques, which involve computing linear upper and lower bounds of the nonlinear activation functions. In this work, we propose a novel parameter search method to improve the quality of these linear approximations. Specifically, we show that using a simple search method, carefully adapted to the given verification problem through state-of-the-art algorithm configuration techniques, improves the average global lower bound by 25% on average over the current state of the art on several commonly used local robustness verification benchmarks.

Keywords: Neural Network Verification · Automated Algorithm Configuration · Convex Relaxation

1 Introduction

Over the last decade, deep learning algorithms have gained increasing significance as essential tools across diverse application domains and usage scenarios. Their applications range from manoeuvre advisory systems in unmanned aircraft to face recognition in mobile phones (see, *e.g.*, [18]). Simultaneously, it is now

* Work done while at University of Oxford.



(a) $\mathcal{S}^+$ (b) $\mathcal{S}^-$ (c) $\mathcal{S}^\pm$

Fig. 1: Linear bounding rules for different cases of the Sigmoid activation function. The x-axis shows the pre-activation bounds, while the y-axis indicates the output of the activation function.

widely acknowledged that neural networks are susceptible to adversarial attacks, where a given input is manipulated to cause misclassification [31]. Remarkably, in image recognition tasks, the perturbation required can be so subtle that it remains virtually imperceptible to the human eye.

Numerous methods have been proposed to evaluate the robustness of neural networks against adversarial attacks. Early methods involve empirical attacks [12, 22, 7]; however, these approaches do not provide a comprehensive assessment of neural network robustness, as a defence mechanism against one type of attack might still be vulnerable to another, potentially novel, class of attacks. Consequently, there have been efforts to develop approaches that formally verify the robustness of neural networks against adversarial attacks [19, 11, 37, 6, 33, 3, 13]. These formal verification methods enable a principled assessment of neural network robustness, providing provable guarantees on desirable properties, typically in the form of a pair of input-output specifications. However, *complete* verification of neural networks, where the verifier is theoretically guaranteed to provide a definite answer to the property under verification, is a challenging NP-complete problem [19]. Solving this problem usually requires computationally expensive methods, *e.g.*, SMT solvers [19] or mixed integer linear programming systems [34], limiting scalability and efficiency.

The aforementioned computational complexity is mainly due to the nonlinear activation functions in a neural network, which results in the neural network verification problem becoming a non-convex optimisation problem. In light of this, *incomplete* methods have been proposed that exploit *convex relaxation* techniques, which approximate the nonlinearities using linear symbolic bounds, to provide sound and efficient verification [1, 30, 39, 11, 28]; completeness can be achieved through branch and bound (see, *e.g.* [5]). Most formal verification methods are limited to ReLU-based networks (see [23]), which satisfy the piecewise linear property. However, convex relaxation techniques are applicable to *more general* commonly-used activation functions, such as Sigmoid or Tanh, by approximating them in terms of piecewise linear functions. This enables an extension of formal verification methods based on convex relaxation to general activation functions, though these remain under-explored.

An important application of neural network verification tools is *robustness certification*, which computes guarantees that the prediction of the network is stable (invariant) around a given input point. CROWN [39] is the first generic framework leveraging adaptive linear bounds to efficiently certify robustness for general activation functions. Another series of techniques, *e.g.*, DeepZ [28], DeepPoly [30] employs abstract interpretation and abstract transformers for commonly-used activation functions, such as Sigmoid and Tanh. CROWN computes linear upper and lower bounds of nonlinear functions, while DeepZ and DeepPoly incorporate convex relaxation into the abstract transformer.

Convex relaxation methods typically over-approximate the activation functions in all nonlinear neurons in the neural network, which inevitably introduces imprecision. Consequently, the corresponding verification algorithms may fail to prove the robustness of a neural network when the original network satisfies the specification and thus weaken certification guarantees. A globally, *i.e.*, network-wise, tighter over-approximation is crucial for strong certification guarantees. However, for Sigmoidal activations, it remains unclear how to design or configure the linear approximation of nonlinear neurons to achieve globally tight output bounds, which are directly used to determine the robustness of neural networks with respect to a specific property under investigation.

Motivated by the need to reduce imprecision of the bounding functions (as an over-approximation of the activation function), in this work we introduce an automated and systematic method to compute tighter bounding functions to improve certification guarantees. These bounding functions are defined by the tangent point, where they touch the activation function. To this end, we propose a novel parameter search method for identifying the tangent points to find better-suited linear bounding functions by considering different cases of Sigmoidal activation functions. To tackle the infinite search space (of tangent points) for the bounding functions, our approach leverages state-of-the-art algorithm configuration techniques.

Concretely, we use automated algorithm configuration techniques to find optimal hyper-parameters of the search method used for obtaining the tangent points of the linear bounding functions, which has previously been done using binary search [39]. These hyper-parameters control the initial tangent point per neuron as well as the rate at which these initial points are updated (we will refer to the latter as a *multiplier* for the remainder of this work) until a feasible bounding function has been obtained. Notice that these hyper-parameters are set for the entire network, *i.e.*, all neurons share the same starting point and multiplier; however, they eventually result in different tangent points, as the search process only ends once a feasible bounding function has been found.

Moreover, we show that, by using our proposed method, we improve the average lower bound on the network output by 25% on average across several verification benchmarks, and can certify robustness for instances that were previously unsolved.

2 Related Work

In the following, we give some background on using convex relaxation for neural network robustness verification and on automated algorithm configuration.

2.1 Convex Relaxation for Neural Network Verification

We use $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ to denote a neural network trained to make predictions on an m-class classification problem. Let $\mathbf{W}^{(i)}$ denote the weight matrix and $\mathbf{b}^{(i)}$ denote the bias for the i-th layer. We use $\hat{z}^{(i)}$ to denote the pre-activation neuron values, and $z^{(i)}$ to denote the post-activation ones, such that we have $\hat{z}^{(i)} = \mathbf{W}^{(i)} z^{(i-1)} + \mathbf{b}^{(i)}$ (*i.e.*, the post-activation values of layer $i-1$ are linearly weighted to become the pre-activation values of layer i). We use σ to denote the activation functions in intermediate layers, such that $z = \sigma(\hat{z})$. In this work, we focus on neural networks with sigmoidal activation functions, including Sigmoid $\sigma(x) = 1/(1 + e^{-x})$ and Tanh $\sigma(x) = (e^x - e^{-x})/(e^x + e^{-x})$. We use $f(x)$ to denote the neural network output for all classes and $f_j(x)$ to denote the output associated with class j (with $1 \leq j \leq m$). The final decision is computed by taking the label with the greatest output value, *i.e.*, $\arg\max_j f_j(x)$.

Neural network verification. The robustness to adversarial perturbations is one of the most important properties of neural networks, requiring that the predictions of a neural network should be preserved for a local input region $\mathcal{C}$, typically defined as a l_p norm ball with a radius of ϵ around the original input x_0, in the following way:

$$\forall x \in \mathcal{C}, \arg\max_j f_j(x) = \arg\max_j f_j(x_0). \tag{1}$$

For an input instance x_0 with ground-truth label y_0, verifying the robustness property can then be transformed into proving that $\forall x \in \mathcal{C}, f_{y_0}(x) - f_j(x) \geq 0$ for all $j \neq y_0$ where $j \in [1, m]$.

Given a verification problem, for example, to check whether the output constraint $f_{y_0}(x) - f_j(x) \geq 0$ is satisfied, we can append an additional layer at the end of the neural network, such that the output property can be merged into the new neural network function $g : \mathbb{R}^n \rightarrow \mathbb{R}$ where $g(x) = f_{y_0}(x) - f_j(x)$. While this should formally be done for each possible network output $j \neq y_0$, to simplify the notation we use a single g.

In this way, the verification problem can be formulated canonically as follows, i.e., to prove or falsify:

$$\forall x \in \mathcal{C}, g(x) \geq 0 \tag{2}$$

One way to verify the property is to solve the optimisation problem $\min_{x \in \mathcal{C}} g(x)$. However, due to the nonlinearity of the activation function σ, the neural network verification problem is NP-complete [19]. To address such intractability, state-of-the-art verification algorithms leverage convex relaxation to transform the verification problem into a convex optimisation problem.

Definition 1 (Convex relaxation of activation functions). *For a nonlinear activation function $\sigma(\hat{z})$ with pre-activation bounds $\hat{z} \in [l, u]$, convex relaxation relaxes the non-convex equality constraint $z = \sigma(\hat{z})$ to two convex inequality constraints by computing the linear lower and upper bounding functions $h_L(\hat{z}) = \alpha_L \hat{z} + \beta_L$ and $h_U(\hat{z}) = \alpha_U \hat{z} + \beta_U$, such that $h_L(\hat{z}) \leq \sigma(\hat{z}) \leq h_U(\hat{z})$.*

In Figure 1, the pre-activation bounds are indicated by the grey-shaded area, h_L is illustrated by the red dotted line, whereas h_U is indicated by the green dotted line. Notice that the parameters $\alpha_L, \beta_L, \alpha_U, \beta_U$ of the linear functions depend on the pre-activation bounds l and u. With such convex inequality constraints, we can propagate the relaxations through layers using an efficient back-substitution procedure to compute the linear lower bounds (denoted by $\underline{g}$) and upper bounds (denoted by $\overline{g}$) for the neural network g.

Definition 2 (Convex relaxation of neural networks). *A convex relaxation of the neural network $g : \mathbb{R}^n \to \mathbb{R}$ over an input region $\mathcal{C}$, are two linear functions $\underline{g}$ and $\overline{g}$ such that $\underline{g}(x) \leq g(x) \leq \overline{g}(x)$ for all $x \in \mathcal{C}$.*

Using convex relaxation, the verification problem is then reduced to the following convex optimisation problem:

$$g^* = \min_{x \in \mathcal{C}} \underline{g}(x) \tag{3}$$

The robustness property is proved if the optimal solution $g^* \geq 0$, as we have $\forall x \in \mathcal{C}$, $g(x) \geq \underline{g}(x)$. Meanwhile, convex relaxation of the nonlinear constraints inevitably introduces approximation. As a consequence, the computed minimum might fail to satisfy $\min_{x \in \mathcal{C}} \underline{g}(x) \geq 0$ even in cases in which the network is robust, *i.e.*, $\min_{x \in \mathcal{C}} g(x) \geq 0$. Instead, the true verification remains unknown. Therefore, improving the quality of the approximation is crucial to reducing false failures in robustness verification, and thus strengthening certification guarantees, which is the aim of our contribution.

2.2 Automated Algorithm Configuration

In general, the algorithm configuration problem can be described as follows: Given an algorithm A (also referred to as the *target algorithm*) with parameter configuration space Θ, a set of problem instances Π, and a cost metric $c : \Theta \times \Pi \to \mathbb{R}$, find a configuration $\theta^* \in \Theta$ that minimises cost c across the instances in Π:

$$\theta^* \in \arg\min_{\theta \in \Theta} \sum_{\pi \in \Pi} c(\theta, \pi) \tag{4}$$

The general workflow of the algorithm configuration procedure starts with selecting a configuration $\theta \in \Theta$ and an instance $\pi \in \Pi$. Next, the configurator initialises a run of algorithm A with configuration θ on instance π, and measures the resulting cost $c(\theta, \pi)$. The configurator uses this information about the target algorithm's performance to find a configuration that performs well on the training instances. This is enabled by a surrogate model, which provides a posterior probability distribution that characterises the potential cost for $c(\theta, \pi)$ at a configuration θ. At every time point t, we have a new observation of the cost c_t at a new configuration point θ_t. The posterior distribution is updated based on the augmented observation set $S_t = S_{t-1} \cup \{(\theta_t, c_t)\}$, and the next configuration point is selected by maximising the acquisition function in the following form:

$$\theta_{t+1} = \arg\max_{\theta \in \Theta} a_t(\theta, S_t) \tag{5}$$

where a_t denotes the acquisition function. A typical choice for the acquisition functions is the expected improvement [16].

Once the configuration budget (e.g., time budget or number of trials) is exhausted, the procedure returns the current incumbent θ^*, which represents the best configuration found so far. Finally, when running the target algorithm with configuration θ^*, it should result in lower cost (such as average running time) or improved solution quality across the benchmark set.

Automated algorithm configuration has already been shown to work effectively in the context of formal neural network verification [21], but also in other, related domains, such as SAT solving [14, 17], scheduling [8], mixed-integer programming [15, 24], evolutionary algorithms [2], answer set solving [10], AI planning [35], and machine learning [32, 9].

3 Method

We consider the task of local robustness verification with Sigmoidal activation functions. To this end, we focus on convex relaxation based perturbation analysis as employed in the CROWN framework [39], and use automated algorithm configuration techniques to improve the linear bounds of the nonlinear activation functions. Notice that, by default, CROWN employs binary search to obtain the points at which the linear bounding functions are tangent to the activation function [39].

In this study, on the other hand, we used SMAC [16] to guide the search for suitable tangent points. SMAC is a widely known, freely available, state-of-the-art configurator based on sequential model-based optimisation (also known as Bayesian optimisation). The main idea of SMAC is to construct and iteratively update a statistical model of target algorithm performance to guide the search for promising configurations; SMAC uses a Random Forest regressor [4].

3.1 Configuration Objective

The objective of the configuration procedure is to maximise the minimum of the global lower bound g^* as defined in Equation (3) for each instance under verification. Notice that the optimisation procedure that solves Equation (3) is performed separately for each instance, *i.e.*, $|\Pi| = 1$. Recall that an instance is verified to be robust when $g^* > 0$, as outlined in Section 2.1.

By design, SMAC solves a minimisation problem; see Equation (4). Since we are interested in maximising the lower bound of the network output, we apply appropriate sign changes and define the cost metric c as the negative of the global lower bound.

3.2 Configuration Space

The CROWN framework [39] proposed a general certification solution for neural networks with nonlinear activation functions, which relies on convex relaxation to compute the output bounds. Since Sigmoidal activation functions (Sigmoid/Tanh/Arctan) share the same features, that is, convex on the negative side ($x < 0$) and concave on the other side ($x > 0$), the authors of [39] leveraged this feature and proposed a general method to compute the parameters (*i.e.*, the tangent points) of linear upper and lower functions h_U, h_L.

Based on the curvature of Sigmoidal activation functions, each nonlinear neuron of the i-th layer (denoted as $[n_i]$) is categorised into one of the three cases: $\mathcal{S}^+$, $\mathcal{S}^-$, and $\mathcal{S}^\pm$ where $\mathcal{S}^+ = \{j \in [n_i] \mid 0 \leq l_j^{(i)} \leq u_j^{(i)}\}$, $\mathcal{S}^- = \{j \in [n_i] \mid l_j^{(i)} \leq u_j^{(i)} \leq 0\}$, and $\mathcal{S}^\pm = \{j \in [n_i] \mid l_j^{(i)} \leq 0 \leq u_j^{(i)}\}$. Intuitively, $\mathcal{S}^+$ represents the case in which the pre-activation bounds are both positive, $\mathcal{S}^-$ represents the case in which they are both negative and $\mathcal{S}^\pm$ represents the case in which l is negative and u is positive. Different bounding rules are then proposed for these three cases; these are illustrated in Figure 1.

Bounding rules for $\boldsymbol{S^+}$ domain For any node $j \in \mathcal{S}^+$, the activation function $\sigma(\hat{z}_j)$ is concave; hence, a tangent line of $\sigma(\hat{z}_j)$ (represented in green in Figure 1) at any tangent point $\hat{z}_j^* \in [l_j^{(i)}, u_j^{(i)}]$ is a valid upper bounding function $h_{U,j}^{(i)}$, and the linear function passing the two endpoints, $(l_j^{(i)}, \sigma(l_j^{(i)}))$ and $(u_j^{(i)}, \sigma(u_j^{(i)}))$, serves as a valid lower bounding function $h_{L,j}^{(i)}$.

To select good values of $\hat{z}_j^*$, we propose a search method whose behaviour depends on two hyper-parameters, which are optimised by SMAC:

- starting point s at which each tangent point $\hat{z}_j^*$ is initialised;
- multiplier ψ to change the value of $\hat{z}_j^*$ if the linear bound is found to be invalid.

This search method is illustrated in Algorithm 1. The algorithm initiates and maintains the best configuration $\theta^* = (s^*, \psi^*)$ found so far, which leads to a tighter bounding function (defined by the tangent points $\hat{z}_j^*$ for each node j) and, thus, a tighter global lower bound g^*. To find optimal settings for the hyper-parameters s and ψ, SMAC samples values of s and ψ from a pre-defined configuration space Θ. Configurations θ, as introduced in Equation (4), are sampled from $s \in [0.01, 2]$ and $\psi \in [1.01, 3]$. Note that these ranges are based on empirical observations; values outside this range typically result in extremely loose bounds. In principle, other ranges could be used to sample values of s and ψ, respectively, as long as $s > 0$ and $\psi > 1$. Also note that the search method

Algorithm 1 Our proposed tangent point search method for sigmoidal functions. It details only the search method in the S^+ domain (for space reasons). The hyper-parameter s determines the initial tangent point and the hyper-parameter ψ determines the rate by which s is changed. The values of s and the update rate ψ are not exposed as hyper-parameters in vanilla CROWN. In our case, these are optimised by SMAC, where SMAC uses a fixed budget of $n_{\texttt{max}}$ evaluation calls. Notice that in vanilla CROWN, binary search is performed once initial bounds have been obtained to move tangent points closer to 0; this step is not performed in our method.

```
procedure OptimiseSearchParameters($x$, $s$, $\psi$)
    Initialize cost $c_0 = \inf$ ▷ first iteration of the optimisation loop can overwrite it
    Initialise observation set $S_0 = \{\emptyset\}$
    for $n = 1, \cdots, n_{\texttt{max}}$ do
        Select starting point and multiplier rate based on observation set
        $(s_n, \psi_n) \leftarrow \arg\max_{(s,\psi)\in\Theta} a_n((s,\psi), S_{n-1})$
        for all node $j$ in network $f$ do
            if $j \in S^+$ then
                Initialise upper bound tangent point for current node $\hat{z}_j^* \leftarrow s_n$
                while Tangent point $\hat{z}_j^*$ leads to invalid upper bound $h_{U,j}^{(i)}$ do
                    Update $\hat{z}_j^*$: $\hat{z}_j^* \leftarrow \hat{z}_j^* \cdot \psi_n$
                end while
            else if $j \in S^-$ then
                Determine tangent point for lower bounding function, in a similar fashion as above
            else                                                        ▷ In this case, $j \in S^{\pm}$;
                Determine tangent point for both lower bounding and upper bounding function, in a similar fashion as above
            end if
        end for
        Evaluate cost $c_n = -1 \cdot g^*(x)$ ▷ Linear bounding layer-by-layer via convex relaxation as per Def. 2 and Eq. 3
        Augment observation set $S_n = S_{n-1} \cup \big\{\big((s_n, \psi_n), c_n\big)\big\}$
        Train surrogate model using $S_n$
        if $c_n < c^*$ then
            $s^* \leftarrow s_n$, $\psi^* \leftarrow \psi_n$, $c^* \leftarrow c_n$
        end if
    end for
    return $c^*$, $(s^*, \psi^*)$
end procedure
```

will be the same for the entire network; however, it will result in different tangent points per node.

Based on a given hyperparameter configuration, we can identify the tangent points $\hat{z}_j^*$ for each node j, and compute the global lower bound g^* as well as the cost value $c = -1 \cdot g^*(x)$. At each iteration, we can acquire the parameter configuration based on the updated surrogate model (trained with the new observation set S_n). With the selected candidate configuration, we perform a sanity check to ensure the validity of the linear upper bounding function incurred by the tangent points $\hat{z}_j^*$ at each node. An upper bound is considered invalid if, at any given point $\hat{z}_j^*$, the value of $h_{U,j}^{(i)}(\hat{z}_j^*)$ is smaller than the value of the non-linear sigmoidal $\sigma(\hat{z}_j^*)$. The global lower bound g_n (also the cost value c_n) is then updated with the new upper bounding function. The loop terminates and returns the best-achieved cost value c^* as well as the corresponding configuration (s^*, ψ^*) when the maximum iteration limit $n_{\texttt{max}}$ is reached. Notice that SMAC has several additional options that can slightly deviate from the description [16].

While Algorithm 1 elaborates on tangent point search for the upper bounding function, it can be similarly configured to search for lower bounding functions with two key modifications. Firstly, the parameters are initialised to different value ranges. Secondly, the validity evaluation is to check the bounding function will always return a lower value than the sigmoidal function. We introduce the details for S^- and $S^\pm$ domains in the following.

Bounding rules for S^- domain Symmetrically, for any node $j \in \mathcal{S}^-$, $h_{U,j}^{(i)}$ is defined as the linear function passing the two endpoints and $h_{L,j}^{(i)}$ can be a tangent line of $\sigma(\hat{z}_j)$ at any point $\hat{z}_j^* \in [l_j^{(i)}, u_j^{(i)}]$. In this case, we employ a similar search method, except that a configuration θ^* is now sampled from $s \in [-0.01, -2]$ and $\psi \in [-1.01, -3]$. Again, other ranges could be used to sample values of s and ψ, respectively, as long as $s < 0$ and $\psi < -1$. Furthermore, a bound is considered invalid if, at any given point $\hat{z}_j^*$, the value of $h_{L,j}^{(i)}(\hat{z}_j^*)$ is larger than the value of $\sigma(\hat{z}_j^*)$.

Bounding rules for $S^\pm$ domain Lastly, for any node $j \in \mathcal{S}^\pm$, $h_{U,j}^{(i)}$ is a tangent line passing $(l_j^{(i)}, \sigma(l_j^{(i)}))$ and a tangent point $(\hat{z}_1^*, \sigma(\hat{z}_1^*))$, where $\hat{z}_1^* \geq 0$, and $h_{L,j}^{(i)}$ is a tangent line that passes $(u_j^{(i)}, \sigma(u_j^{(i)}))$ and a tangent point $(\hat{z}_2^*, \sigma(\hat{z}_2^*))$, where $\hat{z}_2^* \leq 0$. Again, we employ the same search method to obtain $\hat{z}_j^*$ for each bound. Moreover, for any $h_{U,j}^{(i)}$, a configuration θ^* is sampled from $s \in [0.01, 2]$ and $\psi \in [1.01, 3]$, while for any $h_{L,j}^{(i)}$, a configuration θ^* is sampled from $s \in [-0.01, -2]$ and $\psi \in [-1.01, -3]$. Notice that we can use similar rules for bounding Tanh functions, which share the Sigmoidal shape but range from -1 to 1 instead of 0 to 1.

4 Setup for Empirical Evaluation

We will empirically investigate the effectiveness of the proposed search procedure. We consider two types of neural network architecture: convolutional neural networks (CNNs) and fully connected neural networks (FNNs). We evaluate the effectiveness of our approach on the ERAN benchmark [25, 26, 30, 27, 28]. Following the naming conventions, we refer to these networks as ConvMed and

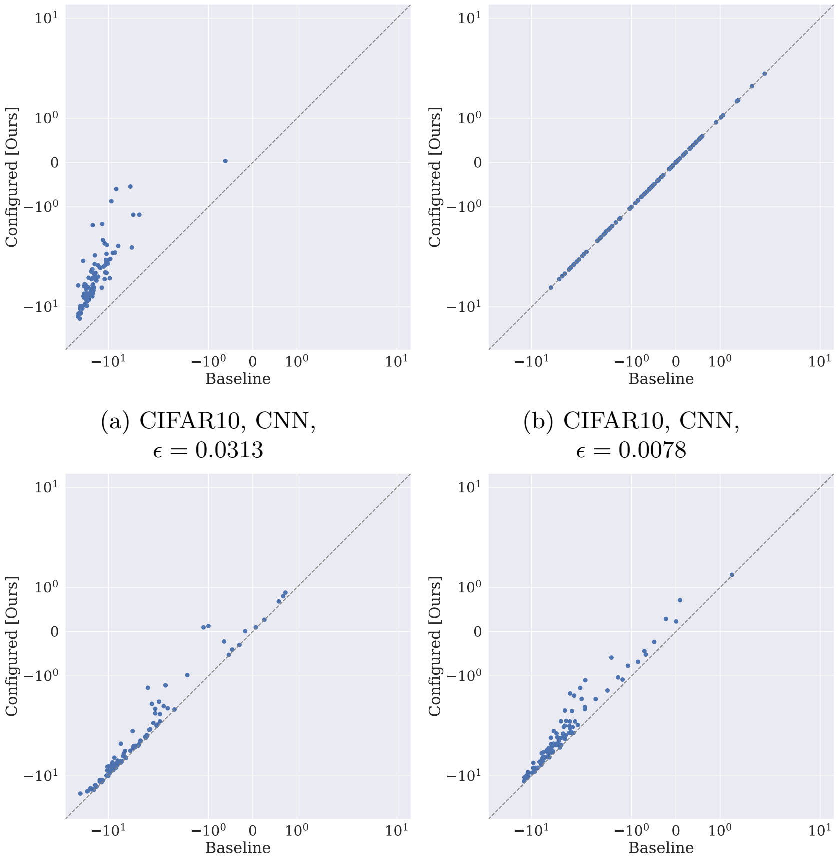
Table 1: Experimental results obtained from our automated bound configuration method, compared to the vanilla CROWN algorithm, which relies on binary search to obtain the tangent points (with α-optimisation for Sigmoid networks). Boldfaced values indicate superior performance.

				Avg. Global Lower Bound g*			**# Certified Instances**	
Dataset	**Network**	**Activation**	**Epsilon**	Baseline	Configured	Improvement	Baseline	Configured
CIFAR10	ConvMed	Sigmoid	0.0313	-15.611	**-5.506**	184%	0	**1**
CIFAR10	ConvMed	Sigmoid	0.0157	-1.727	**-1.633**	6%	13	13
CIFAR10	ConvMed	Sigmoid	0.0078	-0.709	**-0.702**	1%	29	**31**
CIFAR10	ConvMed	Sigmoid	0.0039	-0.440	-0.440	-	39	39
CIFAR10	FNN_6x500	Sigmoid	0.0313	-18.448	**-16.433**	12%	0	0
CIFAR10	FNN_6x500	Sigmoid	0.0157	-17.819	**-16.234**	10%	0	0
CIFAR10	FNN_6x500	Sigmoid	0.0078	-15.075	**-13.478**	12%	0	0
CIFAR10	FNN_6x500	Sigmoid	0.0039	-6.731	**-5.915**	14%	5	**8**
MNIST	ConvMed	Sigmoid	0.3	-5.153	**-4.175**	23%	3	**4**
MNIST	ConvMed	Sigmoid	0.12	-9.484	**-8.563**	10%	0	0
MNIST	ConvMed	Sigmoid	0.06	-0.110	**-0.070**	57%	54	54
MNIST	ConvMed	Sigmoid	0.03	2.753	2.756	-	92	92
CIFAR10	ConvMed	Tanh	0.0313	-65.849	**-55.565**	19%	0	0
CIFAR10	ConvMed	Tanh	0.0157	-44.504	**-33.723**	32%	0	0
CIFAR10	ConvMed	Tanh	0.0078	-10.835	**-9.348**	16%	2	2
CIFAR10	ConvMed	Tanh	0.0039	-2.299	**-2.259**	2%	15	**16**

FNN_6x500. These networks are adversarially trained on the CIFAR-10 dataset with $\epsilon = 0.0313$. In addition, to investigate whether our findings hold for additional datasets, we consider a CNN with similar architecture that is adversarially trained on the MNIST dataset with $\epsilon = 0.3$. Furthermore, to demonstrate the generality of our method to other Sigmoidal activation functions, we perform the evaluation on a neural network trained on CIFAR-10 with similar architecture, adversarially trained with $\epsilon = 0.0313$, but using Tanh activation functions. Further details about the considered networks can be found in the ERAN repository.

We verify each network for local robustness with respect to the first 100 instances in the test set of the MNIST and CIFAR-10 datasets, respectively, initially with perturbation radii equivalent to the values used during training. We also evaluate our method on a broader range of perturbation radii; specifically, we considered $\epsilon \in \{0.0157, 0.0078, 0.0039\}$ for CIFAR-10 and $\epsilon \in \{0.12, 0.06, 0.03\}$ for MNIST. Notice that these values of ϵ are in line with commonly chosen values from the verification literature [20, 36, 29].

For verification, we employ CROWN with default settings as provided by the authors. For Sigmoid-based networks, CROWN also performs α-optimisation [38]; however, this is not implemented for Tanh activation functions, where we run CROWN without further optimisation. As we are purely interested in the global lower bound on the network output, we skip the PGD attack used for upper bound computation. For the configuration procedure, we set the number of evaluation calls performed by SMAC to 150, *i.e.*, the configuration process terminates after 150 trials.



(a) CIFAR10, CNN, $\epsilon = 0.0313$

(b) CIFAR10, CNN, $\epsilon = 0.0078$

(c) CIFAR10, FNN, $\epsilon = 0.0039$

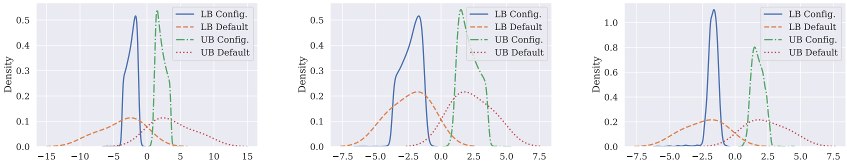
(d) MNIST, CNN, $\epsilon = 0.3$

Fig. 2: Experimental results obtained for Sigmoid-based networks. Each dot represents a problem instance and the global lower bound, *i.e.*, the value of g^*, for that instance achieved by the baseline approach (x-axis) *vs* our method (y-axis).

All experiments are performed on a cluster of machines equipped with NVIDIA GeForce GTX 1080 Ti GPUs with 11 GB video memory.

5 Experimental Results and Discussion

Table 1 shows the average global lower bound over the given test instances achieved by the vanilla CROWN algorithm, which represents our baseline, and those achieved by CROWN in combination with our configured bounding method. In addition, we report the absolute number of instances for which robustness certification could be obtained by each approach.



(a) Layer 1

(b) Layer 2

(c) Layer 3

Fig. 3: Probability density functions of the values of $\hat{z}_j^*$ obtained by our method as well as vanilla CROWN for lower and upper bounding functions per activation layer. Remember that $\hat{z}_j^*$ determines the tangent point of the bounding function of a given node j.

5.1 Sigmoid-based Networks

We first report the results obtained for the Sigmoid-based CNN trained on the CIFAR-10 dataset. These can be found in Table 1. When certifying this network with $\epsilon = 0.03$, we obtained an improvement in the average global lower bound of 184% (-15.611 *vs* -5.506). This is also visualised in Figure 2a. As shown there, the global lower bound is improved by almost an order of magnitude for some of the instances. Furthermore, using our configured bounds, we could certify robustness for an instance that could not be solved by the baseline approach.

For $\epsilon = 0.0157$ and $\epsilon = 0.0078$, we achieved improvements of 6% and 1%, respectively. Interestingly, although the latter is a rather small improvement in the average global lower bound, our method could verify two additional instances, which the baseline method was unable to solve. These results are visualised in Figure 2b. Although instances generally lie very close to the equality line, our method could improve on instances with lower bounds very close to 0, for which even a very small increase can lead to certified robustness.

Lastly, when $\epsilon = 0.0039$, we achieved a similar performance as the baseline method. This indicates that the effectiveness of configured bounds decreases as the perturbation radius becomes very small.

Next, we investigate whether our approach extends to fully connected neural networks; experimental results are shown in Table 1. In general, bounds obtained for this network type are looser than those obtained for the CNN, irrespective of the perturbation radius. Nonetheless, our method achieved improvements in the average global lower bound between 10 and 14 per cent across all considered perturbation radii.

For FNNs, we achieved the greatest improvement when the perturbation radius is smallest, *i.e.* $\epsilon = 0.0039$. In this scenario, we were able to certify robustness for 3 additional instances, which were previously unsolved; see also Figure 2c for a visualisation of these results.

Table 1 also shows the results from our experiments on the CNN trained on the MNIST dataset. When verifying this network with $\epsilon = 0.3$, we achieved an improvement in the average global lower bound of 23%. Furthermore, using our configured bounds, we could again certify robustness for an instance that could not be solved previously. This is also visualised in Figure 2d.

For $\epsilon = 0.12$ and $\epsilon = 0.06$, we achieved improvements in the average global lower bound of 10% and 57%, respectively. Lastly, when $\epsilon = 0.03$, our method did not improve over the baseline. This again shows that, for CNNs, configuring the bounds is less effective if the perturbation radius is minimal.

5.2 Tanh-based Networks

Next, we report the results obtained for the Tanh-based CNN trained on the CIFAR-10 dataset; these are also presented in Table 1. Notably, we achieved consistent improvement for any given value of ϵ. Furthermore, we found that the global lower bounds are generally much lower than those obtained for the Sigmoid-based CNN, although verifying the same properties. Nevertheless, when $\epsilon = 0.0039$, our method enables the verification of an additional instance, which was previously unsolved. Overall, our results demonstrate the strength of our approach, and its potential to improve verification performance on networks with non-piecewise linear activation functions in general.

5.3 Distribution of Tangent Points

To gain a better understanding of the difference between bounding functions obtained by our method and vanilla CROWN, we perform an empirical analysis of tangent point distributions obtained for the linear bounding functions of the Sigmoid-based CNN network trained on CIFAR-10 when verified for local robustness with $\epsilon = 0.0313$. Notice that this benchmark shows the greatest improvement in the average global lower bound.

Figure 3 shows probability density functions of tangent points for lower and upper bounding functions per activation layer of the considered network. Notably, they show that the distribution of tangent points found by the configured bounding algorithm is much more centred around a specific value than those obtained by the baseline approach. Furthermore, the difference between the empirical distribution functions of tangent points obtained by our search method and the baseline approach was determined as statistically significant by means of a Kolmogorov–Smirnov test with a standard significance threshold of 0.05. Moreover, these empirical observations hint towards the existence of an optimal region for bounding parameters of Sigmoidal activation functions, which might be difficult to identify using the baseline approach without automated configuration of the tangent point search method.

6 Conclusions and Future Work

In this work, we have shown that automated algorithm configuration can provide a systematic and effective way for designing bounding functions of nonlinearities beyond the commonly studied piecewise linear activation functions (*e.g.*, ReLU).

Specifically, our new method achieved consistent improvements in average global lower bound across several robustness verification benchmarks and perturbation radii.

At the same time, we see several fruitful avenues for future work. First of all, the proposed search method for obtaining the tangent points is controlled by only two hyper-parameters. A more sophisticated method, allowing for a more fine-grained configuration of the search method, could improve performance further. In addition, one could configure the hyper-parameters of the search method as well as those of the α-optimisation method, *e.g.*, the learning rate of the projected gradient algorithm, jointly. Overall, we see this study as a promising step towards the automated design of versatile and efficient neural network verification algorithms.

Acknowledgment This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation program under GA No. 952215. It further received funding from the ERC under the European Union's Horizon 2020 research and innovation program (FUN2MODEL, grant agreement No. 834115) and ELSA: European Lighthouse on Secure and Safe AI project (grant agreement No. 101070617 under UK guarantee).

References

1. Bak, S., Tran, H.D., Hobbs, K., Johnson, T.T.: Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In: Proceedings of the 32nd International Conference on Computer Aided Verification (CAV 2020). pp. 66–96 (2020)
2. Bezerra, L.C., López-Ibánez, M., Stützle, T.: Automatic Component-Wise Design of Multiobjective Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation **20**(3), 403–417 (2015)
3. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient Verification of ReLU-based Neural Networks via Dependency Analysis. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20). pp. 3291–3299 (2020)
4. Breiman, L.: Random Forests. Machine Learning **45**(1), 5–32 (2001)
5. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: Branch and Bound for Piecewise Linear Neural Network Verification. Journal of Machine Learning Research **21**, 42:1–42:39 (2020)
6. Bunel, R., Turkaslan, I., Torr, P., Kohli, P., Mudigonda, P.K.: A Unified View of Piecewise Linear Neural Network Verification. In: Advances in Neural Information Processing Systems 31 (NeurIPS 2018). pp. 1–10 (2018)
7. Carlini, N., Wagner, D.: Towards Evaluating the Robustness of Neural Networks. In: Proceedings of the 38th IEEE Symposium on Security and Privacy (IEEE S&P 2017). pp. 39–57 (2017)
8. Chiarandini, M., Fawcett, C., Hoos, H.H.: A Modular Multiphase Heuristic Solver for Post Enrolment Course Timetabling. In: Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008) (2008)

9. Feurer, M., Springenberg, J.T., Hutter, F.: Initializing Bayesian Hyperparameter Optimization via Meta-Learning. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI–15). pp. 1128–1135 (2015)
10. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M.T., Ziller, S.: A Portfolio Solver for Answer Set Programming: Preliminary Report. In: Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR2019). pp. 1–6 (2011)
11. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In: Proceedings of the 39th IEEE Symposium on Security and Privacy (IEEE S&P 2018). pp. 3–18 (2018)
12. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and Harnessing Adversarial Examples. In: Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). pp. 1–11 (2015)
13. Henriksen, P., Lomuscio, A.: Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search. In: Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020). pp. 2513–2520 (2020)
14. Hutter, F., Babic, D., Hoos, H.H., Hu, A.J.: Boosting Verification by Automatic Tuning of Decision Procedures. In: Proceedings of Formal Methods in Computer Aided Design (FMCAD'07). pp. 27–34 (2007)
15. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated Configuration of Mixed Integer Programming Solvers. In: Proceedings of the 7th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming (CPAIOR 2010). pp. 186–202 (2010)
16. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential Model-Based Optimization for General Algorithm Configuration. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION 5). pp. 507–523 (2011)
17. Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H., Leyton-Brown, K.: the Configurable SAT Solver Challenge (CSSC). Artificial Intelligence **243**, 1–25 (2017)
18. Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. Journal of Guidance, Control, and Dynamics **42**(3), 598–608 (2019)
19. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017). pp. 97–117 (2017)
20. König, M., Bosman, A.W., Hoos, H.H., van Rijn, J.N.: Critically assessing the state of the art in neural network verification. Journal of Machine Learning Research **25**(12), 1–53 (2024)
21. König, M., Hoos, H.H., Rijn, J.N.v.: Speeding up neural network robustness verification via algorithm configuration and an optimised mixed integer linear programming solver portfolio. Machine Learning **111**(12), 4565–4584 (2022)
22. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533 (2016)
23. Li, L., Xie, T., Li, B.: Sok: Certified robustness for deep neural networks. In: Proceedings of the 44th IEEE Symposium on Security and Privacy (SP2023). pp. 1289–1310 (2023)
24. Lopez-Ibanez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. European Journal of Operational Research **235**(3), 569–582 (2014)

25. Müller, C., Serre, F., Singh, G., Püschel, M., Vechev, M.: Scaling Polyhedral Neural Network Verification on gpus. In: Proceedings of Machine Learning and Systems 3 (MLSys 2021). pp. 1–14 (2021)
26. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the Single Neuron Convex Barrier for Neural Network Certification. In: Advances in Neural Information Processing Systems 32 (NeurIPS 2019). pp. 1–12 (2019)
27. Singh, G., Gehr, T.: Boosting Robustness Certification of Neural networks. In: Proceedings of the 7th International Conference on Learning Representations (ICLR 2019). pp. 1–12 (2019)
28. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and Effective Robustness Certification. In: Advances in Neural Information Processing Systems 31 (NeurIPS 2018). pp. 1–12 (2018)
29. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. In: Proceedings of the 3rd ACM on Programming Languages (POPL 2019). pp. 1–30 (2019)
30. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An Abstract Domain for Certifying Neural Networks. In: Proceedings of the 46th ACM SIGPLAN Symposium on Principles of Programming Languages (ACMPOPL 2019). pp. 1–30 (2019)
31. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014). pp. 1–10 (2014)
32. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2013). pp. 847–855 (2013)
33. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating Robustness of Neural Networks with Mixed Integer Programming. In: Proceedings of the 7th International Conference on Learning Representations (ICLR 2019). pp. 1–21 (2019)
34. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. 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=HyGIdiRqtm
35. Vallati, M., Fawcett, C., Gerevini, A.E., Hoos, H., Saetti, A.: Automatic Generation of Efficient Domain-Specific Planners from Generic Parametrized Planners. In: Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS). pp. 184–192 (2013)
36. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Advances in Neural Information Processing Systems 34 (NeurIPS 2021). pp. 29909–29921 (2021)
37. Xiang, W., Tran, H.D., Johnson, T.T.: Output Reachable Set Estimation and Verification for Multilayer Neural Networks. IEEE Transactions on Neural Networks and Learning Systems **29**(11), 5777–5783 (2018)
38. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: Proceedings of the 9th International Conference on Learning Representations (ICLR 2021) (2021)
39. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient Neural Network Robustness Certification with General Activation Functions. Advances in Neural Information Processing Systems 31 (NeurIPS 2018) **31**, 4944–4953 (2018)