Efficient Neural Network Verification via Adaptive Refinement and Adversarial Search

P. Henriksen and **A. Lomuscio**[1]

Abstract. We propose a novel verification method for high-dimensional feed-forward neural networks governed by ReLU, Sigmoid and Tanh activation functions. We show that the method is complete for ReLU networks and sound for other activation functions. The technique extends symbolic interval propagation by using gradient-descent to locate counter-examples from spurious solutions generated by the associated LP problems. The approach includes a novel adaptive splitting strategy intended to refine the nodes with the greatest impact on the output of the network. The resulting implementation, called VERINET, achieved speed-ups of approximately one order of magnitude for safe cases and three orders of magnitude for unsafe cases on MNIST models against SoA complete methods. VERINET could verify networks of over 50,000 ReLU nodes trained on the CIFAR-10 data-set; these are larger than networks that could previously verified via complete methods.

1 INTRODUCTION

There is a growing trend in safety-critical applications to employ the latest advances from Artificial Intelligence (AI). In particular, application domains such as autonomous vehicles and robotics can greatly benefit from machine-learning methods, notably in vision and in automated decision making.

State-of-the art machine-learning approaches, however, suffer from well-known shortcomings including difficulties in terms of explainability and repair. Most importantly, solutions based on neural networks cannot presently be checked for correctness. Yet, if AI is to be used in mainstream safety-critical applications, it needs to be safe, particularly whenever certification is required.

Several approaches have recently been put forward to verify systems based on neural networks against their specifications. Approaches have targeted both neural networks in isolation [3, 10, 16, 24] and cyber-physical systems (CPS) where neural networks realise a particular component [1, 2, 6, 8, 27]. The verification problems analysed have largely consisted of reachability analysis for CPS and reachability and local robustness for neural networks only. The methods proposed so far differ in whether they are complete or incomplete. In complete methods all counter-examples can theoretically be found; incomplete methods provide no such guarantee, but tend to scale better at present. There is a need for both classes of methods, depending on the requirements from the application domain.

While much progress has been made in both classes of methods, several key challenges remain. Firstly, and most importantly, no method can presently scale to the size of the neural networks presently being used in key applications, such as object detection. Secondly, the most scalable methods only support networks governed by ReLU-activation functions. The present contribution introduces advances towards both these limitations.

Specifically, this paper proposes a method for the verification of neural networks; in particular we focus on networks with high input-dimensionality, as is the case in mainstream computer vision deep neural networks. We build on symbolic interval propagation paired with node splitting refinement, which is presently the best performing technique for high-dimensional inputs [24]. We extend this approach in three key aspects. Firstly, we augment this with local search based on gradient descent to identify counter-examples. Secondly, we develop a heuristic approach for adaptive node splitting based on interval propagation and error estimation. Thirdly, we overcome the limitations to ReLU networks of present interval propagation methods by extending these to Sigmoid activation functions.

The rest of the paper is organised as follows. In Section 2, we recall the basic notions on verification of neural networks and, in particular, a method for symbolic interval propagation. In Section 3, we develop the theoretical results on linear relaxations, including for Sigmoid functions, that are used in the rest of the paper. In Section 4, we present the verification algorithm and discuss various heuristics used. In Section 5, we present VERINET, the resulting implementation, and present experimental results.

Related work. We refer to [15] for a comprehensive survey of recent work on formal verification for neural networks. Formal verification algorithms can be categorised as complete or incomplete. Incomplete approaches introduce over-approximating relaxations for the network and may thus only be able to verify a subset of the inputs. In contrast, complete algorithms encode the exact network and can in principle solve any verification problem.

Complete algorithms are usually based on Mixed Integer Linear Programming (MILP), Satisfiability Modulo Theory (SMT) or Symbolic Interval Propagation (SIP). MILP-based approaches [1, 2, 22] often utilise efficient off-the-shelf solvers by encoding piecewise linear activation functions with integer constraints. In comparison, SMT-based approaches initially reason over a relaxed network and combine this with a branch-and-bound refinement to iteratively remove the overestimation. The relaxed network can be obtained by extending the Simplex algorithm to support relaxed ReLU constraints [9, 10] or by directly relaxing the activation functions [3].

Both SMT and MILP based approaches encode the whole network in the resulting problem. SIP-based approaches aim at reducing the resulting complexity by calculating symbolic bounds for the network's output nodes in a separate phase. These output-bounds are then used as constraints in an LP-solver, eliminating the need to encode the hidden nodes. At time of writing SIP-based algorithms [24, 25] are the most scalable approaches for high-dimensional input networks. In this paper we make a contribution to

[1] Department of Computing, Imperial College London, UK

this line of work in several ways. Differently from these approaches, we introduce a novel method of performing gradient-based adversarial local search to locate valid counter-examples from spurious ones. Secondly, we introduce a novel adaptive splitting strategy aimed at splitting the node with the most impact on the output bounds, rather than a hierarchical way as in present SIP-based approaches [23].

While complete methods only support piecewise linear activation functions, some sound but incomplete algorithms also support Sigmoid and Tanh activation functions. Among the most relevant to the present contribution are the abstract interpretation-based algorithms from [19, 20] and the SIP- based approach from [28]. These algorithms differ from our approach in that they do not use an iterative refinement step to improve the relaxed network and they do not produce counter-examples for unsafe cases. In contrast, the SMT-based algorithm introduced in [18] implements an iterative refinement step; however, it does not scale to larger networks.

All these elements combined allow us to obtain an implementation that generally outperforms present approaches by more than an order of magnitude on MNIST networks. In our experiments the implementation also locates all unsafe cases in less than one second, while comparable algorithms time out after one hour for several of these data-points. In common with much of the literature on the subject we focus on verification of local robustness.

2 PRELIMINARIES

In this section we briefly introduce general concepts related to verification of feed-forward neural networks, a specific method for symbolic interval propagation [24, 25], and some related notions.

A deep feed-forward neural network (FFNN) consists of an input layer, several hidden layers, and an output layer [4]. Each layer has one or more nodes; hidden nodes are endowed via an activation function $\sigma_i : \mathbb{R} \rightarrow \mathbb{R}$. The input to a node z_k^i in layer i and position k is a linear combination of outputs from previous layers; the node's output is determined by the expression $y_k^i = \sigma_i(z_k^i)$. By combining the computation on all layers an FFFN can be formally be associated with a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ representing the layer by layer computation [4].

In common with most verification algorithms for neural networks, we assume that the output layer does not have an activation function. This is not a very restrictive assumption, since most activation functions are monotonically increasing, and thus do not change the relative ordering of the output nodes. For the hidden layers, we consider the three most common activation functions, $Relu(z) = \max(0, z)$, $Sigmoid(z) = 1/(1 + e^{-z})$ and $Tanh(z) = (e^{2z} - 1)/(e^{2z} + 1)$.

In this paper we are concerned with verifying whether FFNNs are locally robust. More specifically, given a neural network, a set of inputs, and a classification c, we consider whether the network's output for classification c is larger than the output of all other classes. This is formalised in the definition below.

Definition 1 (Local robustness)**.** *Let $\langle f, \psi_{\boldsymbol{x}}, c\rangle$ be a tuple where $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is an FFNN, $\psi_{\boldsymbol{x}} = \{l_i \leq x_i \leq u_i \mid l_i, u_i \in \mathbb{R}\}_{\forall i \in \{1,...,n\}}$ is a set of constraints specifying the lower and upper bounds on the input to f and $c \in \{1, 2, ..., m\}$ is a classification. Moreover, let $f(\boldsymbol{x})_i$ denote the i^{th} output of the network. The verification of local robustness is the decision problem of determining whether it is the case that $f(\boldsymbol{x}')_c > f(\boldsymbol{x}')_t$ for all $\boldsymbol{x}'$ satisfying $\psi_{\boldsymbol{x}}$ and all classifications $t \neq c$. If this condition holds, we say that the FFNN f is safe for c with respect to the condition $\psi_{\boldsymbol{x}}$. Otherwise, f is unsafe and exhibits a counter-example $\overline{\boldsymbol{x}}$ satisfying $\psi_{\boldsymbol{x}}$, with $f(\overline{\boldsymbol{x}})_c \leq f(\overline{\boldsymbol{x}})_t$ for some classification $t \neq c$.*

Any non-linearity in an FFNNs complicates the verification task above. A method to address this involves over-approximating the network's outputs by linearly relaxing the activation functions [3, 24] and symbolic interval propagation [24, 25]. We here introduce a version of the relaxations above, based on two constraints only.

Definition 2 (Two-constraint relaxation)**.** *Let $z_l, z_u \in \mathbb{R}$ be concrete lower and upper bounds on the input of an activation function $\sigma : [z_l, z_u] \rightarrow \mathbb{R}$, where $[z_l, z_u] \subseteq \mathbb{R}$ denotes the closed real interval between z_l and z_u. A two-constraint linear relaxation is a tuple $\langle r_l, r_u\rangle$, where $r_l : [z_l, z_u] \rightarrow \mathbb{R}$ is a lower bounding linear function such that $r_l(z) \leq \sigma(z)$ and $r_u : [z_l, z_u] \rightarrow \mathbb{R}$ is an upper bounding linear function such that $r_u(z) \geq \sigma(z)$ for all $z \in [z_l, z_u]$.*

In Section 3 we derive two-constraint linear relaxations for several common activation functions. The relaxations are then used during symbolic interval propagation to propagate linear bounding equations through the network's non-linearities.

As mentioned in the related work section, the purpose of symbolic interval propagation is to calculate linear bounding equations for the network's output nodes. These bounding equations depend only on the network's input variables, thus removing the need to consider the network's hidden nodes in later phases of the verification algorithm. The method proposed in this work is based on **error-based symbolic interval propagation (ESIP)** [23]. ESIP differs from the symbolic interval propagation from [24, 25] by calculating only one linear equation for each node instead of a lower and upper bounding equation. We denote the resulting equation for a node in layer i and position k as $\boldsymbol{q}^i(\boldsymbol{x})_k$. This equation represents the node's attainable input values when all nodes in the preceding layers operate at their lower linear relaxation. The latest version of Neurify [23] implements ESIP as described here.

In the EISP method, to account for the possibility of nodes operating at their upper relaxation, an additional concrete error value is calculated for each relaxation. The errors at layer i are represented by an error-matrix $E_{in}^i \in \mathbb{R}^{m_i \times m_i'}$ where m_i is the number of nodes in layer i and m_i' is the total number of nodes in all previous layers. An element $E_{k,h}^i$ represents the maximum value that the equation $\boldsymbol{q}^i(\boldsymbol{x})_k$ would change if the equation had been propagated through the upper relaxation of node h instead of the lower one. The resulting lower symbolic bound on the input of node k in layer i is $\boldsymbol{z}_l^i(\boldsymbol{x})_k = \boldsymbol{q}^i(\boldsymbol{x})_k + \sum_{h \mid \, E_{k,h}^i < 0} E_{k,h}^i$ and the upper bound is $\boldsymbol{z}_u^i(\boldsymbol{x})_k = \boldsymbol{q}^i(\boldsymbol{x})_k + \sum_{h \mid \, E_{k,h}^i > 0} E_{k,h}^i$. So, if $z^i(\boldsymbol{x})_k$ is the node's input value as calculated in the FFNN, then $\boldsymbol{z}_l^i(\boldsymbol{x})_k \leq z^i(\boldsymbol{x})_k \leq \boldsymbol{z}_u^i(\boldsymbol{x})_k$ for all $\boldsymbol{x}$ satisfying $\psi_{\boldsymbol{x}}$.

We now show how the input equations $\boldsymbol{q}^{i+1}(\boldsymbol{x})$ and error-matrix E^{i+1} at layer $i + 1$ are calculated given $\boldsymbol{q}^i(\boldsymbol{x})$ and E^i. The first step is to calculate the lower and upper linear relaxations $r_{l,k}^i, r_{u,k}^i$ for each node k in layer i. These relaxations require a concrete lower and upper bound $z_{l,k}^i, z_{u,k}^i \in \mathbb{R}$ on the node's input. Since the input equations are linear, this can be calculated trivially as $z_{l,k}^i = \min_{\boldsymbol{x} \mid \psi_{\boldsymbol{x}}}(\boldsymbol{z}_l^i(\boldsymbol{x})_k)$ and $z_{u,k}^i = \max_{\boldsymbol{x} \mid \psi_{\boldsymbol{x}}}(\boldsymbol{z}_u^i(\boldsymbol{x})_k)$, where $\psi_{\boldsymbol{x}}$ are the input-constraints as defined in Definition 1. Indeed, this is the approach used in [23] for ReLU networks. In Section 3 we will extend the supported activation functions by introducing novel relaxations for the Sigmoid and Tanh.

The lower relaxations are then used to propagate the equations $\boldsymbol{q}^i(\boldsymbol{x})$ and error matrix E^i through the activation functions in layer i. So, if $r_{l,k}^i(z) = az + b$ then $\boldsymbol{q}^{i,out}(\boldsymbol{x})_k = a\boldsymbol{q}^i(\boldsymbol{x})_k + b$ and $\widehat{E}_{k,:}^{i,out} = aE_{k,:}^{i+1}$. The new errors introduced from only using the lower relaxation $\epsilon_k^i = max_{z \in [z_{l,k}^i, z_{u,k}^i]}(r_{u,k}^i(z) - r_{l,k}^i(z))$ are concatenated with the old errors to create the new output error matrix $E^{i,out} = [\widehat{E}^{i,out}, diag(\epsilon^i)]$. Finally, the input errors and equations to the next layer are calculated by propagating them through the affine layer. So, $\boldsymbol{q}^{i+1}(\boldsymbol{x}) = W^{i+1}\boldsymbol{q}^{i,out}(\boldsymbol{x}) + \boldsymbol{b}^{i+1}$ and $E^{i+1} = W^{i+1}E^{i,out}$ where W^{i+1} is the weight matrix and $\boldsymbol{b}^{i+1}$ is the bias.

By repeating this process for all hidden layers, we can calculate linear bounding equations for the network's output. We use the notation $\boldsymbol{y}_l(\boldsymbol{x}) = \boldsymbol{z}_l^o(\boldsymbol{x})$ and $\boldsymbol{y}_u(\boldsymbol{x}) = \boldsymbol{z}_u^o(\boldsymbol{x})$ to explicitly denote the symbolic bounds for the output layer of an FFNN with o layers.

While this is not stated in [24], it can be shown that ESIP, as implemented in [23], is sound in the sense that $\boldsymbol{y}_l(\boldsymbol{x})_k \leq f(\boldsymbol{x})_k \leq \boldsymbol{y}_u(\boldsymbol{x})_k \quad \forall \boldsymbol{x}|\psi_{\boldsymbol{x}}$ and output nodes k. It follows that a network is safe for a classification c if the lower bound for c is larger than the upper bound for all other classes, or equivalently $\boldsymbol{y}_u(\boldsymbol{x})_t - \boldsymbol{y}_l(\boldsymbol{x})_c = (\boldsymbol{q}^o(\boldsymbol{x})_t + \sum_{h|\ E^o_{t,h}>0} E^o_{t,h}) - (\boldsymbol{q}^o(\boldsymbol{x})_c + \sum_{h|\ E^o_{c,h}<0} E^o_{c,h}) < 0$. However, in [23] a slight variation is used where the individual errors are subtracted first to achieve tighter bounds. We now formalise this in a way that is in line with the presentation in the rest of this paper.

Lemma 1. *Let $\langle f, \psi_{\boldsymbol{x}}, c\rangle$ be the verification input as defined in Definition 1 and let $q^o(\boldsymbol{x}), E^o$ be the associated equations and errors at the output layer calculated by ESIP. If $\boldsymbol{y}_l(\boldsymbol{x})_{t-c} = \boldsymbol{q}^o(\boldsymbol{x})_t - \boldsymbol{q}^o(\boldsymbol{x})_c + \sum_{h|\ (E^o_{t,h} - E^o_{c,h})<0}(E^o_{t,h} - E^o_{c,h})) < 0$ for all $t \neq c$ and $\boldsymbol{x}$ satisfying $\psi_{\boldsymbol{x}}$, then f is safe for c with respect to $\psi_{\boldsymbol{x}}$ as defined in Definition 1.*

We conclude by recalling that **a linear programming (LP) problem** concerns the minimisation of a linear objective function while satisfying some linear constraints. In this paper we use satisfiability calls to efficient off-the-shelf LP-solvers in an attempt to locate counter-examples or to prove that no counter-examples exist. For more information on linear programming, we refer to [26].

In Section 4 we show how ESIP and LP can be combined with novel techniques to solve the robustness problem from Definition 1. However, first we derive the linear relaxations necessary for ESIP.

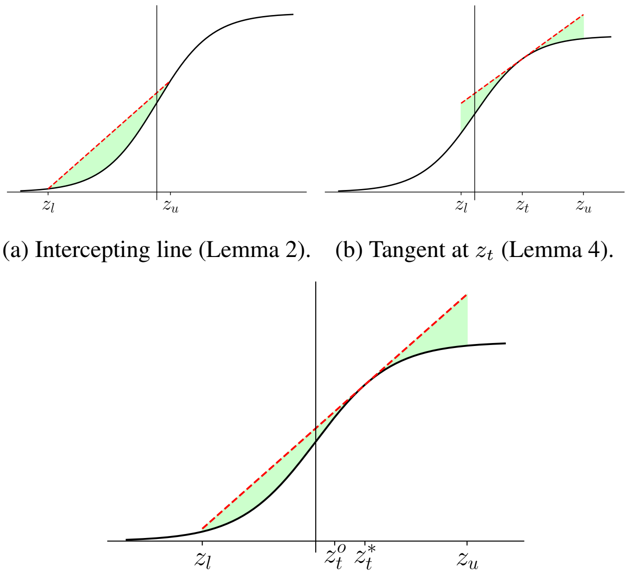
3 LINEAR RELAXATIONS

As discussed in the previous section, the ESIP method requires linear relaxation for all non-linear activation functions. One of the key contributions of this paper lies in the support of a wide class of activation functions. For ReLU-based activations we use the relaxation introduced in [24], defining the lower relaxation as $r_l(z) = \frac{z_u}{z_u - z_l}z$ and the upper relaxation as $r_u(z) = \frac{z_u}{z_u - z_l}(z - z_l)$. In the rest of this section we introduce optimal upper linear relaxations for S-shaped activation functions such as Sigmoid and Tanh; the lower relaxations are calculated similarly. Notice that we also support batch-normalisation; however, batch-normalisation acts linearly during inference and does not require relaxations. We formally introduce S-shaped functions as follows.

Definition 3. *A continuous function $\sigma : \mathbb{R} \to \mathbb{R}$ is S-shaped iff:*

1. $\sigma''(z) \begin{cases} > 0 \text{ if } z < 0 \\ < 0 \text{ if } z > 0 \\ = 0 \text{ if } z = 0 \end{cases}$
2. *$\sigma'(z) \geq 0$ for all z.*
3. *$\sigma'(z) = \sigma'(-z)$ for all z.*
4. *$\sigma''(z)$ is differentiable for all z.*

Definition 4. *An upper linear relaxation $r_u : [z_l, z_u] \to \mathbb{R}$ for $\sigma : [z_l, z_u] \to \mathbb{R}$ is optimal iff it is minimal w.r.t. $\int_{z_l}^{z_u} r_u(z) - \sigma(z)\, dz$.*



(a) Intercepting line (Lemma 2). (b) Tangent at z_t (Lemma 4).

(c) Tangent at z_t^*. (Lemma 5).

Figure 1: Optimal upper relaxations.

So, the optimal upper linear relaxation minimises the area between the relaxation and the activation function. Note that finding optimal linear relaxations is difficult due to the non-linear integral and the constraint $r_u(z) > \sigma(z)$ for all $z \in [z_l, z_u]$.

In the rest of the section we show the following three results.

1. If the line intercepting the endpoints $(z_l, \sigma(z_l))$ and $(z_u, \sigma(z_u))$ is a valid upper relaxation, then it is the optimal upper linear relaxation (see Figure 1a).
2. If the intercepting line from point 1 is not a valid upper relaxation, but the tangent to σ at $z_t = \frac{z_u^2 - z_l^2}{2(z_u - z_l)}$ is, then this tangent is the optimal upper linear relaxation (see Figure 1b).
3. If none of the above relaxations are valid, then the optimal upper linear relaxation is a tangent at $z_t^* \in [z_l, z_u]$ where z_t^* is the minimal value such that the tangent is a valid upper relaxation (see Figure 1c).

Similar to the relaxations from [28], all of the cases either use the line intercepting both endpoints or a tangent as the upper relaxation. However, the analysis here differs in that we identify when each of the options is optimal and where the optimal tangent is.

We begin with the first point above by presenting the result below, which also provides a simple test to check whether the line intercepting both endpoints is a valid upper relaxation.

Lemma 2. *Let $\sigma : [z_l, z_u] \to \mathbb{R}$ be an S-shaped activation function. If the line $r_u : [z_l, z_u] \to \mathbb{R}$ intercepting both endpoints $(z_l, \sigma(z_l))$ and $(z_u, \sigma(z_u))$ is a valid upper relaxation, then it is the optimal upper linear relaxation. Furthermore, r_u is valid iff $r_u'(z_u) \leq \sigma'(z_u)$.*

Proof sketch. If $r_u(z)$ is a valid upper relaxation, it is clearly the optimal upper linear relaxation since any other valid upper linear relaxation $l(z)$ would require $l(z) \geq r_u(z)$ for all z.

As $\sigma(z)$ is convex for $z \leq 0$, concave for $z \geq 0$ and r_u intercepts the endpoints, it is clear that if $r_u(z^*) < \sigma(z^*)$ for any $z^* \in (z_l, z_u)$ then $r_u(z) < \sigma(z)$ for all $z \in [z^*, z_u)$. This and the fact that r_u intercepts $(z_u, \sigma(z_u))$ means that $r_u'(z_u) > \sigma'(z_u)$ for all invalid upper linear relaxations. So, if $r_u'(z_u) \leq \sigma'(z_u)$ then r_u is a valid upper relaxation.

The converse follows easily. □

So, the line intercepting both endpoints is a valid and optimal upper relaxation if the derivative of σ is larger than the derivative of the line at the upper endpoint. Otherwise, the following lemma states that the optimal upper relaxation is a tangent to the activation function.

Lemma 3. *Let $\sigma : [z_l, z_u] \rightarrow \mathbb{R}$ be an S-shaped activation function. If the line intercepting both endpoints $(z_l, \sigma(z_l))$ and $(z_u, \sigma(z_u))$ is not a valid upper relaxation, then the optimal upper linear relaxation $r_u : [z_l, z_u] \rightarrow \mathbb{R}$ is a tangent to σ at a tangent point $z_t \in [z_l, z_u]$ with $z_t > 0$. Furthermore, a tangent r_u is a valid upper relaxation if $r_u(z_l) \geq \sigma(z_l)$.*

Proof sketch. It is clear that the optimal upper linear relaxation is a tangent to σ at some point $z_t \in [z_l, z_u]$; if it were not, we could always create a better relaxation by reducing the line's intercept or adjusting the derivative. The tangent point has to be at $z \geq 0$ since σ is convex for $z < 0$.

The only way the tangent r_u is not a valid upper relaxation is if it intercepts σ at some point $z^* \in [z_l, 0]$. If this is the case, then we also have $r_u(z_l) < \sigma(z_l)$ due to the convexity. This implies that the tangent r_u is a valid upper relaxation if $r_u(z_l) \geq \sigma(z_l)$. □

So, in this case, the optimal upper linear relaxation is a tangent to the S-shaped function at some point $z_t > 0$. Furthermore, a tangent is a valid upper relaxation if it is valid at the lower endpoint z_l. The following lemma provides a candidate for the optimal tangent.

Lemma 4. *Let $\sigma : [z_l, z_u] \rightarrow \mathbb{R}$ be an S-shaped activation function and assume that the line intercepting both endpoints is not a valid upper relaxation. If the tangent at $z_t = \frac{z_u^2 - z_l^2}{2(z_u - z_l)}$ is a valid upper relaxation, then it is the optimal upper linear relaxation.*

Proof sketch. Let $A : [z_l, z_u] \rightarrow \mathbb{R}$ be defined by $A(z_t) = \int_{z_l}^{z_u} r_u(z) - \sigma(z)\, dz$. Lemma 3 states that the optimal upper linear relaxation r_u is a tangent to $\sigma(z)$, so $r_u(z) = \sigma'(z_t)(z - z_t) + \sigma(z_t)$. To find z_t, we minimise $A(z_t)$:

$$\begin{aligned} A(z_t) &= \int_{z_l}^{z_u} \sigma'(z_t)(z - z_t) + \sigma(z_t)\, dz - \int_{z_l}^{z_u} \sigma(z)\, dz \\ &= \left[\frac{z^2}{2}\sigma'(z_t) - z_t z \sigma'(z_t) + z\sigma(z_t)\right]_{z_l}^{z_u} - (\Lambda(z_u) - \Lambda(z_l)) \\ A'(z_t) &= \left[\frac{z^2}{2}\sigma''(z_t) - z\sigma'(z_t) - z_t z \sigma''(z_t) + z\sigma'(z_t)\right]_{z_l}^{z_u} \\ &= \sigma''(z_t)(z_u(\frac{z_u}{2} - z_t) - z_l(\frac{z_l}{2} - z_t)) = 0 \end{aligned} \tag{1}$$

This equation has two solutions. Firstly, $\sigma''(z_t) = 0$ which only happens at $z_t = 0$ from the definition of σ. Secondly, we have:

$$z_u(\frac{z_u}{2} - z_t) - z_l(\frac{z_l}{2} - z_t) = 0 \implies z_t \quad = \frac{z_u^2 - z_l^2}{2(z_u - z_l)} \tag{2}$$

Furthermore, both of the endpoints z_l, z_u are potential candidates for the minima; however, it is easy to show that $A'(z_u) > 0$ and $A'(z_l) < 0$, which means that they are local maxima, not minima. The solution $z_t = 0$ is only a valid upper relaxation if $z_l \geq 0$ and since $z_t = z_l$ is not a minima $z_t = 0$ is clearly never a minima either. This leaves us with $z_t = \frac{z_u^2 - z_l^2}{2(z_u - z_l)}$ as the only candidate and since A is defined on a closed interval it has to be the minima. □

Note that the test from Lemma 3 can be used to check whether the tangent at $z_t = \frac{z_u^2 - z_l^2}{2(z_u - z_l)}$ is a valid upper relaxation. If this is not the case, then the optimal tangent point is given by the next result.

Lemma 5. *Let $\sigma : [z_l, z_u] \rightarrow \mathbb{R}$ be an S-shaped activation function and assume that the tangent at $z_t = \frac{z_u^2 - z_l^2}{2(z_u - z_l)}$ and the line intercepting both endpoints are not valid upper relaxations. Then the tangent at z_t^* is the optimal upper linear relaxation where $z_t^* \in [z_l, z_u]$ is the minimal value such that the tangent is a valid upper relaxation.*

Proof sketch. The proof is similar to the proof for Lemma 4 by restricting the interval to $[z_t^*, z_u]$ instead of $[z_l, z_u]$. $z_t = \frac{z_u^2 - z_l^2}{2(z_u - z_l)}$ is not a valid relaxation from the assumptions and z_u is not a minima for the same reasons as in Lemma 4. So, z_t^* is the only candidate. □

Since $z_t^* \in [z_l, z_u]$ is the minimal value such that the tangent is a valid upper relaxations, it can be shown from Lemma 3 that the tangent intercepts $(z_l, \sigma(z_l))$. So, z_t^* can be found by solving $(\sigma(z_t^*) - \sigma(z_l))/(z_t^* - z_l) = \sigma'(z_t^*)$ for z_t^*.

We can not solve this equation analytically. Instead, we use an iterative algorithm to calculate an approximation $\widehat{z_t^*} \approx z_t^*$. The algorithm is initialised with $z_0 = z_u$, and given z_{i-1} we calculate z_i by solving the equation:

$$\frac{\sigma(z_{i-1}) - \sigma(z_l)}{z_{i-1} - z_l} = \sigma'(z_i) \tag{3}$$

Solving this equation for z_i when σ is the Sigmoid results in:

$$z_i = -\log(\frac{1}{\sigma(z_i)} - 1) \qquad \sigma(z_i) = \frac{1 \pm \sqrt{1 - 4\frac{\sigma(z_{i-1}) - \sigma(z_l)}{z_{i-1} - z_l}}}{2} \tag{4}$$

And when σ is the Tanh activation function the solution is:

$$z_i = \frac{1}{2}\log\left(\frac{1 + \sigma(z_i)}{1 - \sigma(z_i)}\right) \qquad \sigma(z_i) = \pm\sqrt{1 - \frac{\sigma(z_{i-1}) - \sigma(z_l)}{z_{i-1} - z_l}} \tag{5}$$

After calculating z_i for $i \in \{0, 1, 2.....m\}$ where m is a predefined number of steps, we use $\widehat{z_t^*} = z_m$ as our approximation. The pseudo-code is provided in Algorithm 1, and in the following lemma we state two important properties of this algorithm.

Lemma 6. *Let $\sigma : [z_l, z_u] \rightarrow \mathbb{R}$ be an S-shaped activation function and assume that the tangent at $z_t = \frac{z_u^2 - z_l^2}{2(z_u - z_l)}$ and the line intercepting both endpoints are not valid upper relaxations. Furthermore, let z_i be as calculated by Algorithm 1 and z_t^* be the minimal value such that the tangent at z_t^* is a valid upper relaxation. Then z_i satisfies $z_i < z_{i-1}$ and $z_i > z_t^*$ for all i.*

Proof sketch. The derivative $\sigma'(z_i)$ is equal to the slope of the line intercepting $(z_l, \sigma(z_l))$ and $(z_{i-1}, \sigma(z_{i-1}))$ (see Equation 3). This intercepting line is not a valid upper relaxation since $z_{i-1} > z_t^*$, so it is clear from Lemma 2 that $\sigma'(z_i) > \sigma'(z_{i-1})$. Since z_i and z_{i-1} are in the concave part of σ, this implies that $z_i < z_{i-1}$.

Furthermore, as z_t^* is the minimal value such that the tangent is a valid upper relaxation it is clear that this tangent intercepts $(z_l, \sigma(z_l))$, so $\sigma'(z_t^*) = \frac{\sigma(z_t^*) - \sigma(z_l)}{z_t^* - z_l}$. Since both z_t^* and z_{i-1} are in the concave part of σ and $z_t^* < z_{i-1}$, the slope of the line intercepting $(z_l, \sigma(z_l))$ and $(z_{i-1}, \sigma(z_{i-1}))$ is clearly smaller than the slope of the line intercepting $(z_l, \sigma(z_l))$ and $(z_t^*, \sigma(z_t^*))$. Combining this, we get: $\sigma'(z_t^*) = \frac{\sigma(z_t^*) - \sigma(z_l)}{z_t^* - z_l} \geq \frac{\sigma(z_{i-1}) - \sigma(z_l)}{z_{i-1} - z_l} = \sigma'(z_i)$, where the last equality is from Equation 3. Again, since z_i and z_t^* are in the concave part of σ, this implies $z_i > z_t^*$. □

The previous lemma states that $z_i > z_t^*$, so the tangent at z_i is a valid upper relaxation for all i. Furthermore, as $z_i < z_{i-1}$, the solution improves at each iteration since z_i is closer to z_t^* than z_{i-1} was.

This concludes the last of the three cases outlined in the beginning of this section, thus defining the necessary linear relaxations for S-shaped activation functions.

Algorithm 1 Iterative approximation of z_t^*

```
$z_0, m \leftarrow z_u$, Number of iterations
for i = 1...m do
    $z_i \leftarrow update\_step(z_{i-1})$ //Given in Equation 2 & 3
end for
$\widehat{z_t^*} \leftarrow z_m$
```

4 VERIFICATION ALGORITHM

Having derived the linear relaxations for the symbolic interval propagation, we now introduce an efficient verification algorithm for the local robustness problem from Definition 1. We begin by providing a high-level overview of the algorithm, followed by a detailed account of each step. The pipeline of the procedure is presented in Figure 2.

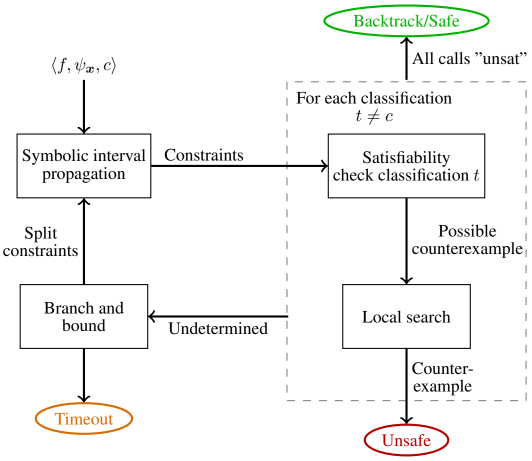


Figure 2: The verification pipeline.

The algorithm takes as input a tuple $\langle f, \psi_{\boldsymbol{x}}, c\rangle$, where $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is an FFNN, $\psi_{\boldsymbol{x}} = \{l_k \leq x_k \leq u_k | l_k, u_k \in \mathbb{R}\}_{\forall k \in \{1,\ldots,n\}}$ is a set of constraints on the input to f and $c \in \{1, 2, ..., m\}$ is a classification as defined in Definition 1. The first block of the pipeline uses error-based symbolic interval propagation to calculate linear constraints on the network's output as explained in Section 2. The output-constraints are then used in a satisfiability call to an LP-solver in order to locate an input-assignment misclassified as t for each incorrect classification $t \neq c$. If the solver determines all calls to be unsatisfiable, then the network is safe and the algorithm terminates. Otherwise, the algorithm checks whether each assignment is spurious by running them through the FFNN. For each assignment found to be spurious, the method launches a gradient-based local search to locate a valid counter-example. If a valid counter-example is found, then the network is provably unsafe and the algorithm terminates. Otherwise, the branch and bound phase is launched by splitting the input to a node and branching. The constraints from the split are added back to the LP-solver and the symbolic interval propagation, and the algorithm is repeated for each branch. The verification loop terminates when either the network is proven safe in all branches, or a valid counter-example is found, or a timeout criterion is reached.

The rest of this section covers each step of the pipeline in detail.

Symbolic interval propagation. The first phase of the verification algorithm implements error-based symbolic interval propagation (ESIP) on the FFNN f and input constraints $\psi_{\boldsymbol{x}}$ as explained in Section 2. Recall that this produces lower symbolic bounds $\boldsymbol{y}_l(\boldsymbol{x})_k$ (upper, $\boldsymbol{y}_u(\boldsymbol{x})_k$, respectively) constraining each of the network's outputs $f(\boldsymbol{x})_k$ under the given input constraints $\psi_{\boldsymbol{x}}$. Formally;

$$\boldsymbol{y}_l(\boldsymbol{x})_k \leq f(\boldsymbol{x})_k \leq \boldsymbol{y}_u(\boldsymbol{x})_k \quad \forall \boldsymbol{x}|\psi_{\boldsymbol{x}}, \quad k \in \{1, ...m\} \tag{6}$$

Lemma 1 states that the network is safe for the input-constraints $\psi_{\boldsymbol{x}}$ and classification c if the upper bounds of all classifications $t \neq c$ are smaller than the lower bound of c. The next phase aims at either proving that this condition holds, or locating a counter-example if it does not.

Satisfiability check. The second block of the pipeline takes as input the symbolic bounds calculated by ESIP, the input constraints $\psi_{\boldsymbol{x}}$ and the classification c. For each classification $t \neq c$, a satisfiability call is made to an LP-solver with the constraints $\psi_{\boldsymbol{x}}$ and the inequality $\boldsymbol{y}_l(\boldsymbol{x})_{t-c} \geq 0$ where $\boldsymbol{y}_l(\boldsymbol{x})_{t-c}$ is as defined in Lemma 1.

If the solver determines that all calls are unsatisfiable, then $\boldsymbol{y}_l(\boldsymbol{x})_{t-c} < 0$ for all $t \neq c$. From Lemma 1 this means that f is safe for c with respect to $\psi_{\boldsymbol{x}}$ and the algorithm terminates.

Otherwise, the solver's assignments $\boldsymbol{x}^t$ are treated as potential counter-examples. Indeed, as the bounds from ESIP are not exact, we may obtain spurious counter-examples such that $\boldsymbol{y}_l(\boldsymbol{x}^t)_{t-c} \geq 0$ even though $f(\boldsymbol{x}^t)_k < f(\boldsymbol{x}^t)_c$ for all $k \neq c$. The next phase tests whether this is the case and, if so, attempts to locate a valid counter-example.

Local search. The third block of the procedure is devoted to checking whether the assignments $\boldsymbol{x}^t$ from the LP-solver constitute valid counter-examples by running them on the network and determining whether $f(\boldsymbol{x}^t)_k > f(\boldsymbol{x}^t)_c$ for any $k \neq c$. If this is not the case, then a gradient descent-based local search is launched in an attempt to locate a valid counter-example by minimising the loss function $L(\boldsymbol{x}) = f(\boldsymbol{x})_c - f(\boldsymbol{x})_t$ with respect to $\boldsymbol{x}$. After each step of the gradient descent, the assignment $\boldsymbol{x}^i$ is clipped to the input bounds $\psi_{\boldsymbol{x}}$ and checked to see whether it is a valid counter-example. If this is the case, then f is unsafe for c with respect to $\psi_{\boldsymbol{x}}$ and the algorithm terminates. Otherwise, the local search terminates after a predetermined number of steps or when the loss changes less than a given fraction.

Branch and bound. If the local search does not find a valid counter-example and at least one of the calls to the LP-solver is satisfiable, then the verification problem is still undefined at this point. In this case a branch and bound refinement phase is launched by splitting the input to a node.

The method aims at splitting the node with the most impact on the lower bound of the given classification $\boldsymbol{y}_l(\boldsymbol{x})_c$ and the upper bounds of other classifications $\boldsymbol{y}_u(\boldsymbol{x})_t$ for $t \neq c$. This is done in an attempt to satisfy the condition for Lemma 1, $\boldsymbol{y}_l(\boldsymbol{x})_{t-c} \geq 0$ for all $t \neq c$ and $\boldsymbol{x}$ satisfying $\psi_{\boldsymbol{x}}$. We use a novel heuristic to quantify the impact of a hidden node on these bounds.

Definition 5. *Let $\langle f, \psi_{\boldsymbol{x}}, c\rangle$ be the verification input as defined in Definition 1 and let E^m, $\boldsymbol{y}_l(\boldsymbol{x})_k$ and $\boldsymbol{y}_u(\boldsymbol{x})_k$ be the associated error matrix and bounds at the output layer as calculated by ESIP. Moreover, let $\mathcal{H}$ be the set of all hidden nodes* *in the network and let $\mathcal{T} = \{t \in \{1, ..., m\} \mid t \neq c, (\min_{\boldsymbol{x}|\psi_{\boldsymbol{x}}}(\boldsymbol{y}_l(\boldsymbol{x})_c) < \max_{\boldsymbol{x}|\psi_{\boldsymbol{x}}}(\boldsymbol{y}_u(\boldsymbol{x})_k))\}$ be the set of potential counter-example classifications. Then the impact-score $s : \mathcal{H} \rightarrow \mathbb{R}$ for node h is defined as $s(h) = |\mathcal{T}||\min(E^m_{c,h}, 0)| + \sum_{t \in \mathcal{T}} \max(E^m_{t,h}, 0)$, where $|\mathcal{T}|$ is the cardinality of $\mathcal{T}$.*

During ESIP, the negative values of $E^m_{c,h}$ are added to $\boldsymbol{y}_l(\boldsymbol{x})_c$ and positive values of $E^m_{t,h}$ are added to $\boldsymbol{y}_u(\boldsymbol{x})_t$ as explained in Section 2 , so the impact-score indicates how the relevant bounds change after a split. Notice that improving $\boldsymbol{y}_l(\boldsymbol{x})_c$ works towards proving $\boldsymbol{y}_u(\boldsymbol{x})_t < \boldsymbol{y}_l(\boldsymbol{x})_c$ for all other classifications $t \neq c$ at once, which is the reason for the weighing factor $|\mathcal{T}|$ in the impact-score.

After calculating the impact-score for all hidden nodes, the node k with the largest impact-score is split by constraining the node's input in a lower and upper bounding branch. If $\boldsymbol{z}_l(\boldsymbol{x})_k$ and $\boldsymbol{z}_u(\boldsymbol{x})_k$ are the lower and upper symbolic bounds for the input to node k as calculated by ESIP and $p \in \mathbb{R}$ is the split-value, then the split-constraints $\boldsymbol{z}_l(\boldsymbol{x})_k \leq p$ and $\boldsymbol{z}_u(\boldsymbol{x})_k \geq p$ are added to the LP-solver in the lower and upper branch, respectively. For ReLU nodes we use $p = 0$ and for s-shaped activation functions $\sigma : [z_l, z_u] \rightarrow \mathbb{R}$ we use the midpoint such that $\sigma(p) = (\sigma(z_u) + \sigma(z_l))/2$.

Furthermore, the split-constraints are used during ESIP to improve the concrete bounds used to calculate linear relaxations. So, for the lower branch the maximum of the lower bound z_l calculated by ESIP and the split-value p is used as a lower bound for the split-node, and analogously for the upper branch with the upper bound.

After adding the split constraints, both branches are restarted from the symbolic interval propagation with two optimisations. First, only the symbolic bounds for nodes in layers after the split-node change, so the symbolic interval propagation is only performed for those layers. Second, if the satisfiability call to the LP-solver is determined to be unsatisfiable for a classification t, then we know from Lemma 1 that no counter-example $\boldsymbol{x}^t$ exists such that $f(\boldsymbol{x}^t)_t \geq f(\boldsymbol{x}^t)_c$, so classification t is not reconsidered in subsequent branches.

Soundness and completeness. We end this section by showing that the proposed algorithm is sound for all networks and complete for ReLU networks.

Theorem 1. *Let $\mathcal{V}$ be the set of possible inputs on the form $\langle f, \psi_{\boldsymbol{x}}, c\rangle$ as defined in Definition 1 and let $A : \mathcal{V} \rightarrow \{$"safe", "unsafe", "timeout"$\}$ be the algorithm described in this section. Furthermore, let $\mathcal{Y}_o = f(\{\boldsymbol{x}|\psi_{\boldsymbol{x}}\})$ be the set of possible outputs and $\mathcal{Y}_m = \{\boldsymbol{y} \in \mathbb{R}^m \mid \exists t \neq c, \boldsymbol{y}_t \geq \boldsymbol{y}_c\}$ be the set of misclassified outputs. If $A(\langle f, \psi_{\boldsymbol{x}}, c\rangle) =$ "safe" then $\mathcal{Y}_o \cap \mathcal{Y}_m = \emptyset$.*

Proof sketch. Before branching, $\mathcal{A}(\langle f, \psi_{\boldsymbol{x}}, c\rangle) =$ "safe" requires that the satisfiability call to the LP-solver with the constraints $\{\psi_{\boldsymbol{x}}, \boldsymbol{y}_u(\boldsymbol{x})_{t-c} \geq 0\}$ is unsatisfiable. It follows directly from Lemma 1 that $\mathcal{Y}_o \cap \mathcal{Y}_m = \emptyset$.

The proof after branching is similar, using the fact that the branching exhaustively explores all options and the algorithm only returns "safe" if all branches are "safe". □

Theorem 2. *Let $A, \mathcal{V}, \mathcal{Y}_o$ and $\mathcal{Y}_m$ be defined as in Theorem 1. If f_{ReLU} is an FFNN with only ReLU activation functions, then there exists a finite timeout setting such that $\mathcal{Y}_o \cap \mathcal{Y}_m = \emptyset$ implies that $A(\langle f_{ReLU}, \psi_{\boldsymbol{x}}, c\rangle) =$ "safe".*

Proof sketch. It is clear that $A(\langle f_{ReLU}, \psi_{\boldsymbol{x}}, c\rangle) \neq$ "unsafe" when $\mathcal{Y}_o \cap \mathcal{Y}_m = \emptyset$ since all potential counter-examples are checked in the local search. The only thing remaining is to prove that the solution is determined in finite time. This follows from the fact that splitting a ReLU node at 0 results in two branches where the node operates linearly in each branch, thus removing all overestimation for that node. So, by splitting all nodes the symbolic interval propagation becomes exact, which means that the problem can be solved by exploring a maximum of 2^N branches where N is the number of nodes. □

In summary, the algorithm proposed in this section is complete for ReLU networks, and sound for all other networks. Furthermore, it extends current symbolic interval propagation based algorithms [25, 24] with three major novel contributions. Firstly, we introduce a local search to detect counter-examples. Secondly, we use a novel adaptive splitting strategy aiming at always splitting the most influential node. Finally, with the optimal linear relaxations derived in the previous chapter, our algorithm also supports Sigmoid and Tanh activation functions. In the following section we show that a toolkit implementing this algorithm significantly outperforms comparable state-of-the-art verification toolkits.

5 IMPLEMENTATION AND EXPERIMENTAL RESULTS

We implemented the algorithm presented in Section 4 into VERINET [7], a toolkit for robustness verification. The toolkit takes as input a problem specification as defined in Definition 1 and determines whether or not the network is robust by outputting either "safe" or "unsafe". The algorithm may also output either "timeout" or "underflow", thereby indicating that a timeout has been reached (see Section 4) in the first case, and that the algorithm finished splitting all nodes without finding a solution due to floating point precision, in the latter case. If "unsafe" is outputted, then a counter-example is also returned.

VERINET utilises NumPy with the OpenBLAS-backend [17] for an efficient implementation of the ESIP method, the Gurobi solver version 8.1.1 [5] for the LP-phase, and the PyTorch implementation of the Adam optimiser [11] for the local search. Furthermore, parallelisation is implemented with Python's multiprocessing module to take advantage of the highly parallel nature of the branching phase.

We evaluated the performance of VERINET against three state-of-the-art verification toolkits, NEURIFY [23][2], MARABOU [10], and ERAN [21] on several networks trained on the MNIST [14] and CIFAR-10 [13] datasets. Due to the extensive runtime of the experiments we used two benchmarking environments. For the experiments on fully-connected MNIST networks, we used a workstation with an Intel Core i9 9900X 3.5 GHz 10-core CPU, 128 GB ram running Fedora 30 with Linux kernel 5.3.6. Experiments on CIFAR-10 and MNIST with convolutional layers were run on a workstation with a Ryzen 3700X 3.6 GHz 8-core CPU, 64 GB ram running Ubuntu 18.04 with Linux kernel 4.18.0. All experiments were performed with $L_\infty \leq \epsilon$ bounds on the input and a timeout of 3600 seconds.

The MNIST fully-connected experiments consider the three fully-connected networks from [24] with 48, 100, and 1024 ReLU nodes. We used $\epsilon \in \{1, 2, 5, 10, 15\}$ and 100 images for each ϵ-network combination, except for the 1024 node network where we used 50 images for $\epsilon \in \{5, 10, 15\}$. A total of 810 cases were verified as safe, 453 as unsafe and 81 timed out in both toolkits. Neurify did not terminate after reaching the one hour timeout for the remaining six data-points; so we decided to not include them. The verification-times for safe and unsafe cases are plotted in Figure 3

[2] In communication with the author, we found a bug in Neurify which could have resulted in incorrect results for convolutional networks. We have fixed this bug to the best of our ability.

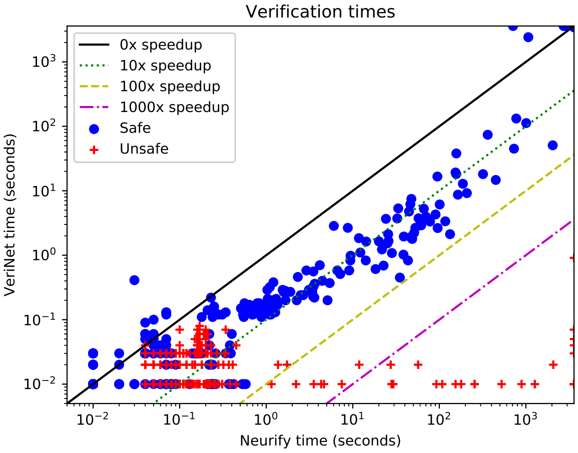


Figure 3: Verification times of NEURIFY vs VERINET for the MNIST fully-connected experiments. Blue dots are data-points verified as safe, while red crosses are data-points verified as unsafe.

Note from Figure 3 that VERINET found all unsafe cases in less than one second, while NEURIFY timed-out for several of those cases. The total speed-up by calculating the timed-out points as 3600 seconds was ×4068. VERINET only branched for one of the unsafe data-points, so almost all of the speed-up can be attributed to the local search functionality.

For the cases verified as safe, most images that required branching were solved around an order of magnitude faster in VERINET. As mentioned in the related work section, NEURIFY implements a hierarchical splitting strategy, starting from the first layer and moving on to the next layer only after all nodes in the previous layer have been split. These experiments indicate that adaptive splitting generally is favourable for fully-connected networks. However, the real strength of adaptive splitting will be evident in the convolutional experiments.

Finally, we also compared our results against MARABOU [10] on the smallest MNIST network. We found that MARABOU is not optimised for high-dimensional input networks and did not perform as well as VERINET or NEURIFY even in the case of small ϵ.

The MNIST Convolutional experiments were performed on the network from [24], which has two convolutional layers followed by two fully-connected layers and a total of 4804 ReLU nodes. The results are provided in Table 1.

We found that NEURIFY started splitting at the fully-connected layers, skipping the convolutional layers in the beginning. As a result, NEURIFY is not complete for this network and returns ”Underflow” for a significant amount of cases. In contrast, the adaptive splitting approach employed by VERINET is complete and can split nodes in small layers, which experiments indicate leads to a performance advantage. As a result, even in the case of safe cases, VERINET solved problems that NEURIFY failed to check, see e.g., $\epsilon = 5$. For $\epsilon > 5$, VERINET's performance further increased due to its ability of performing local search, thereby solving 38% of the cases for $\epsilon = 15$ against 24% only for NEURIFY.

The CIFAR-10 Convolutional experiments considered a new network with 56,384 ReLU nodes, 7 convolutional layers, 6 batch normalisation layers, and a test-set accuracy of 85.56%. These networks cannot be checked by NEURIFY nor MARABOU. VERINET was able to verify several problems on this network as shown in Table 2. To the best of our knowledge, this is the largest network ever verified by a complete method.

Table 1: MNIST-convolutional results

		VeriNet			Neurify		
ϵ	Total	Safe	Unsafe	Undec	Safe	Unsafe	Undec
1	50	49	1	0	49	1	0
2	50	49	1	0	49	1	0
5	50	42	2	6	37	2	11*
10	50	1	6	43	1	4	45*
15	50	0	19	31	0	12	38*

Results for the 4804 ReLU node MNIST convolutional network.
* NEURIFY reported an underflow for the images that could not be verified.

Table 2: CIFAR-10 convolutional results for VERINET

ϵ	Total	Safe	Unsafe	Undec
0.05	50	49	1	0
0.1	50	43	5	2
0.2	50	37	9	4
0.5	50	0	13	37
1	50	0	23	27

The MNIST Sigmoid and Tanh experiments were performed on the networks from [20] with 3000 hidden nodes. To the best of our knowledge, existing verification algorithms with iterative refinement that support the Sigmoid and Tanh activation functions (e.g. [18]) do not scale to large networks. So, we benchmarked against the ERAN toolkit [21] paired with the sound but incomplete DeepPoly [20] algorithm. DeepPoly does not locate counter-examples, so we only compared the number of cases determined to be safe.

VERINET determined significantly more safe-cases, which is most likely due to the optimal linear relaxations employed. Note, however, that VERINET, NEURIFY and MARABOU are not sound with respect to floating point-arithmetic while DeepPoly is. We are not aware of cases from the literature where floating-point correctness has been an issue in robustness verification.

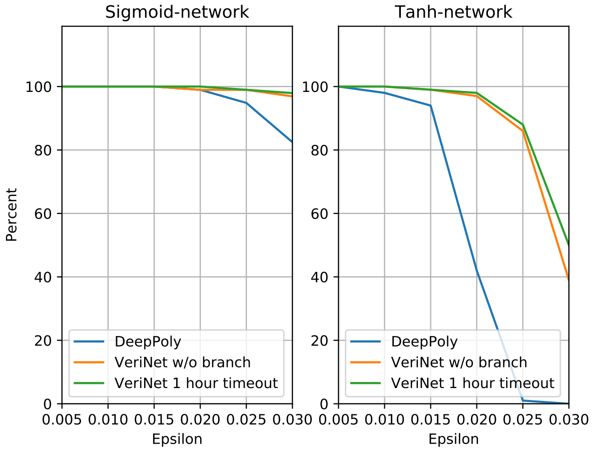


Figure 4: Safe cases proven by VERINET and DeepPoly for the Sigmoid and Tanh networks. In these experiments, pixel values range from 0 to 1 to keep the notation as in [20].

In conclusion, VERINET was the most performing among the complete toolkits checked and offered functionality (such as support for Sigmoid functions) not present in other toolkits.

6 CONCLUSIONS

It is recognised that for FFNNs to be used in safety-critical AI applications, such as autonomous vehicles, they need to be verifiable. However, the high degree of non-linearity exhibited by FFNNs makes this particularly challenging. State-of-the-art methods cannot analyse the networks used in practical applications; this is due to limited scalability and unsupported architectures or activation functions.

In this paper we have put forward an algorithm based on symbolic interval propagation approach [24] extending it in several directions. The experiments demonstrate that the novel local search and adaptive splitting significantly increase the scalability of the method, particularly in the case of unsafe cases. Differently from related work, the proposed method also includes support for Sigmoid and Tanh activation functions by deriving their optimal linear relaxations. Furthermore, the contribution is complete for ReLU networks and sound for Sigmoid and Tanh networks.

In future work we plan to investigate further verification problems, such as transformational robustness [12].

ACKNOWLEDGEMENTS

This work is partly funded by DARPA under the Assured Autonomy programme (FA8750-18-C-0095) and the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (grant number EP/S023356/1). Alessio Lomuscio is supported by a Royal Academy of Engineering Chair in Emerging Technologies.

REFERENCES

[1] M. E. Akintunde, A. Kevorchian, A. Lomuscio, and E. Pirovano, 'Verification of RNN-based neural agent-environment systems', in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI19)*, pp. 6006–6013. AAAI Press, (2019).

[2] M. E. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano, 'Reachability analysis for neural agent-environment systems', in *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning (KR18)*, pp. 184–193. AAAI Press, (2018).

[3] R. Ehlers, 'Formal verification of piece-wise linear feed-forward neural networks', in *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA17)*, volume 10482 of *Lecture Notes in Computer Science*, pp. 269–286. Springer, (2017).

[4] A. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, volume 1, MIT press Cambridge, 2016.

[5] Inc. Gurobi Optimization. Gurobi optimizer reference manual. http://www.gurobi.com, 2019.

[6] H.D.Tran, F. Cai, M.L. Diego, P. Musau, T.T. Johnson, and X.Koutsoukos, 'Safety verification of cyber-physical systems with reinforcement learning control', *ACM Transactions on Embedded Computing Systems (TECS18)*, **18**(5s), 1–22, (2019).

[7] P. Henriksen and A. Lomuscio. Verinet. https://vas.doc.ic.ac.uk/software/verinet.html, 2019.

[8] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, 'Verisig: verifying safety properties of hybrid systems with neural network controllers', *arXiv preprint arXiv:1811.01828*, (2018).

[9] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, 'Reluplex: An efficient SMT solver for verifying deep neural networks', in *Proceedings of the 29th International Conference on Computer Aided Verification (CAV17)*, volume 10426 of *Lecture Notes in Computer Science*, pp. 97–117. Springer, (2017).

[10] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, 'The marabou framework for verification and analysis of deep neural networks', in *Proceedings of the 31st International Conference on Computer Aided Verification (CAV19)*, volume 11561 of *Lecture Notes in Computer Science*, pp. 443–452. Springer, (2019).

[11] D. P. Kingma and J. L. Ba, 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980*, (2014).

[12] P. Kouvaros and A. Lomuscio, 'Formal verification of cnn-based perception systems', *arXiv preprint arXiv:1811.11373*, (2018).

[13] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. online: http://www.cs.toronto.edu/kriz/cifar.html, 2014.

[14] Y. Lecun. The mnist database of handwritten digits. https://ci.nii.ac.jp/naid/10027939599/en/, 1999.

[15] C. Liu, T. Arnon, C. Lazaru, C. Barrett, and M.J. Kochenderfer, 'Algorithms for verifying deep neural networks', *arXiv preprint arXiv:1903.06758*, (2019).

[16] A. Lomuscio and L. Maganti, 'An approach to reachability analysis for feed-forward relu neural networks', *arXiv preprint arXiv:1706.07351*, (2017).

[17] OpenBLAS. Openblas. https://www.openblas.net, 2019.

[18] L. Pulina and A. Tacchella, 'An abstraction-refinement approach to verification of artificial neural networks', in *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV10)*, volume 6184 of *Lecture Notes in Computer Science*, pp. 243–257. Springer, (2010).

[19] G Singh, T. Gehr, M. Püschel, and M. Vechev, 'Boosting robustness certification of neural networks', in *Proceedings of the 7th International Conference on Learning Representations (ICLR19)*, (2019).

[20] G. Singh, T. Gehr, M. Püschel, and P. Vechev, 'An abstract domain for certifying neural networks', in *Proceedings of the ACM on Programming Languages (POPL19)*, volume 3, pp. 1–30. ACM Press, (2019).

[21] G. Singh, M. Mirman, T. Gehr, A. Hoffman, P. Tsankov, D. Drachsler-Cohen, M. Püschel, and M. Vechev. Eth robustness analyzer for neural networks (eran). https://github.com/eth-sri/eran, 2019.

[22] V. Tjeng, K. Xiao, and R. Tedrake, 'Evaluating robustness of neural networks with mixed integer programming', in *Proceedings of the 7th International Conference on Learning Representations (ICLR19)*, (2019).

[23] S. Wang, K. Pei, W. Justin, J. Yang, and S. Jana. Neurify. https://github.com/tcwangshiqi-columbia/Neurify, 2019.

[24] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, 'Efficient formal safety analysis of neural networks', in *Proceedings of the 31st Annual Conference on Neural Information Processing Systems 2018 (NeurIPS2018)*, pp. 6367–6377. Curran Associates, Inc., (2018).

[25] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, 'Formal security analysis of neural networks using symbolic intervals', in *Proceedings of the 27th USENIX Security Symbosium (USENIX18)*, (2018).

[26] W. Winston, *Operations research: applications and algorithms*, Duxbury Press, 1987.

[27] W. Xiang, H. Tran, J. A. Rosenfeld, and T. T. Johnson, 'Reachable set estimation and safety verification for piecewise linear systems with neural network controllers', in *2018 Annual American Control Conference (ACC18)*, pp. 1574–1579. AACC, (2018).

[28] H. Zhang, T. Weng, P. Chen, C. Hsieh, and L. Daniel, 'Efficient neural network robustness certification with general activation functions', in *Proceedings of the 31st Annual Conference on Neural Information Processing Systems 2018 (NeurIPS2018)*, pp. 4944–4953. Curran Associates, Inc., (2018).