MAKE INTERVAL BOUND PROPAGATION GREAT AGAIN

Anonymous authors
Paper under double-blind review

ABSTRACT

In various scenarios motivated by real life, such as medical data analysis, autonomous driving, and adversarial training, we are interested in robust deep networks. A network is robust when a relatively small perturbation of the input cannot lead to drastic changes in output (like change of class, etc.). This falls under the broader scope field of Neural Network Certification (NNC). Two crucial problems in NNC are of profound interest to the scientific community: how to calculate the robustness of a given pre-trained network and how to construct robust networks. The common approach to constructing robust networks is Interval Bound Propagation (IBP). This paper demonstrates that IBP is sub-optimal in the first case due to its susceptibility to the wrapping effect. Even for linear activation, IBP gives strongly sub-optimal bounds. Consequently, one should use strategies immune to the wrapping effect to obtain bounds close to optimal ones. We adapt two classical approaches dedicated to strict computations – Dubleton Arithmetic and Affine Arithmetic – to mitigate the wrapping effect in neural networks. These techniques yield precise results for networks with linear activation functions, thus resisting the wrapping effect. As a result, we achieve bounds significantly closer to the optimal level than IBPs.

1 INTRODUCTION

Deep neural networks find application in medical data analysis, autonomous driving, and adversarial training (Zhang et al., 2023) where safety-critical and robustness guarantees against adversarial examples (Biggio et al., 2013; Szegedy et al., 2014) are extremely important. The rapid development of artificial intelligence models does not correspond to their robustness (Luo et al., 2024). Therefore, certifiable robustness (Zhang et al., 2022; Ferrari et al., 2022) becomes an important task in deep learning. The aim of Neural Network Certification lies in rigorous validation of a classifier's robustness within a specified input region.

Most commonly applied certification method is interval bound propagation (IBP) (Gowal et al., 2018; Mirman et al., 2018). It is based on application of interval arithmetic, which allows to propagate the input intervals through a neural network. If in the case of classification tasks such propagation gives an unambiguous output then all elements of the interval inputs are guaranteed to have identical prediction. Therefore, we can control the behavior of predictions of a neural network in an explicit neighborhood of the input data. Among the approaches studied most extensively in robustness of neural networks is Certified Training (Singh et al., 2018; Mao et al., 2024). These certified training methods try to estimate and optimize the worst-case loss approximations of a network across an input domain defined by adversary specifications. They achieve this by computing an over approximation of the network reachable set through symbolic bound propagation techniques (Singh et al., 2019; Gowal et al., 2018). Interestingly, training techniques based on the least accurate bounds derived from interval-bound propagation (IBP) have delivered the best empirical performance (Shi et al., 2021; Mao et al., 2024).

The certification process uses the network's upper bound of the propagated input interval. Although classical IBP gives reasonable estimations in robust training, it ultimately fails in the certification of classically trained neural networks. In practice, for a given pre-trained networks, intervals that store intermediate values in a neural network evaluation increase exponentially with respect to number of layers, see Theorem 2.1. This phenomena is known as *the wrapping effect* (Neumaier, 1993), which in the context of neural networks applications was previously an unexplored area. Such a growth of obtained bounds makes them often dramatically sub-optimal in practical applications.

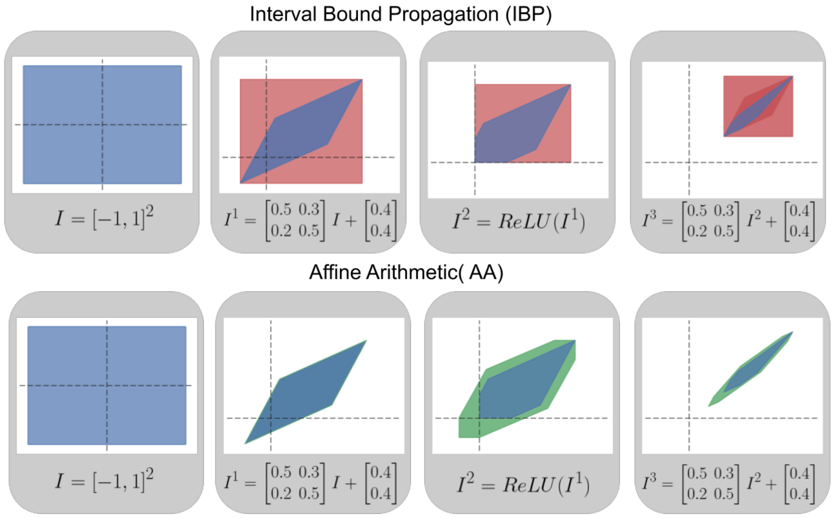

Figure 1: The figure presents how the interval is propagated throughout linear layers. By red color we marked wrapping obtain by IBP and by green by Affine Arithmetic. As we can see, Affine Arithmetic produces significantly lower wrapping effects. In the case of linear transformations, Affine Arithmetic gives an exact approximation. We can work with more complex objects than hyper-cubes from IBP and obtain bounds close to optimal ones. In Fig. 3 we present the procedure used in Affine Arithmetic to obtain $\mathrm{ReLU}(I^1)$.

The aim of this paper is to analyse and adapt two existing methods for reduction of the wrapping effect to the context of neural networks applications: Doubleton Arithmetics (DA) (Mrozek & Zgliczyński, 2000) and Affine Arthmetic (AA) (de Figueiredo & Stolfi, 2004).

Doubletons is very special family of subsets of $\mathbb{R}^n$, which has been extensively used to reduce and control the wrapping effect in validated solvers to initial value problems of ODEs (Kapela et al., 2021). Direct application of interval arithmetics to propagate sets along trajectories, that is enclosing them into the Cartesian product of closed intervals, leads to accumulated overestimation known as the wrapping effect. This overestimation becomes larger and larger when we use smaller time steps h of the underlying ODE solver. Lohner (Lohner, 1992) observed, that one can propagate coordinate system (approximate space derivative of the flow) between subsequent time steps along trajectories of the flow. This method proved to be very efficient and one of the reasons is that the for small time steps the mapping defined as a time shift along trajectories is close to identity. In this paper we adopt Doubleton Arithmetics to the context of neural networks. The main difference in comparison to ODEs is that the dimensions of subsequent layers in a network are usually different, while in ODEs we have a fixed dimension of the phase space. Moreover, in neural networks we often deal with non-smooth functions, such as ReLU. In Section 3 we will formally define doubleton representation and give algorithm for propagation of non-smooth functions in this arithmetics.

The second method, called Affine Arthmetic (AA) (de Figueiredo & Stolfi, 2004), is a special case of Taylor Models by (Berz & Makino, 1999; Makino & Berz, 2009). Here subsets of $\mathbb{R}^n$ are represented as a range of an affine map (often sparse) over a cube $[-1, 1]^m$, where m in general is not related to n. Similarly to DA, evaluation of affine layers in AA causes no wrapping effect and it is sharp. In Section 3 and Appendix we show how to implement ReLU and $\mathrm{softmax}$ functions over a set represented in this way. The main numerical drawback of AA is that its computational cost is non-constant and depends on actual input arguments. Our experiments show that AA outperforms IBP in obtained bounds, see Fig. 1. Although AA and DA provide bounds of comparable sizes, AA is much faster (orders of magnitude) than DA on large networks. Thus, we recommend AA for evaluation of interval inputs through neural networks.

To make our approach completely certifiable, we need to have the full control over the numerical and rounding errors appearing in floating point arithmetics (Kahan, 1996). To obtain this we have decided to switch from Python-based networks to interval arithmetics (IEEE Std 1788.1-2017, 2018) in C++ with the use of the CAPD library (Kapela et al., 2021), which gives us certifiable control over the rounding errors. Consequently, to the best of the authors' knowledge, the presented approach is the first model that deals with rounding errors and obtains guaranteed boundaries.

Our contributions can be summarized as follows:

- We theoretically analyze the wrapping effect in the Neural Network certification task and show that classical IBP is sub-optimal even for linear transformations.
- We adapt two approaches, Doubleton Arithmetic and Affine Arithmetic, with full control over numerical and rounding errors to the neural network certification task.
- Using empirical evaluation, we show that Affine Arithmetic gives the best bounds of the neural network output and significantly outperforms classical IBP.

2 IBP AND WRAPPING EFFECT

In this section, we examine how interval bounds propagate through a linear layer. We show that the appearance of *wrapping effect*, even in the case of isometric transformations, leads to an exponential growth of interval bounds. Wrapping effect is typically studied in the context of strict estimations for solutions of dynamical systems, where the propagated set is at each iteration “wrapped” in the minimal interval bound (Neumaier, 1993). Therefore, applying the standard interval bound propagation layer after layer leads to an exponential increase of bounds.

Given a bounded set $X \subset \mathbb{R}^n$, by $\mathrm{IB}(X)$ (interval bounds) we denote the smallest interval bounding box for X. The aim of IBP (Interval Bound Propagation) lies in obtaining the IB for the processing of X through a network, i.e. a series of possibly nonlinear maps. In the case of linear map $A = [a_{ij}]$, the optimal bounds are given by

$$\mathrm{IB}(A(x + [-r, r])) = Ax + [-|A|r, |A|r], \tag{1}$$

where $x + [-r, r] = \prod_i [x_i - r_i, x_i + r_i]$ and $|A| = [|a_{ij}|]$. To propagate an interval through the ReLU activation, we propagate the lower and upper bound separately: $\mathrm{ReLU}([x, y]) = [\mathrm{ReLU}(x), \mathrm{ReLU}(y)]$. We can propagate intervals through the standard network Φ, which is represented as a sequence of mappings corresponding to the successive layers $y = \Phi(x) = \phi_k \circ \ldots \circ \phi_1(x)$. The aim of IBP is to obtain the estimate of $\mathrm{IB}(\Phi(x + [-r, r]))$, where commonly we restrict to the case when $r = \varepsilon \mathbb{1}$:

$$\mathrm{IB}(\Phi(x + \varepsilon[-\mathbb{1}, \mathbb{1}])), \text{ where } \mathbb{1} = (1, \ldots, 1) \in \mathbb{R}^n.$$

The standard classical approach used for IBP in the networks uses the naive iterative approach, where we process through each layer the interval bounds obtained from the previous one:

$$[I = x + [-r, r]] \to [I^1 = \mathrm{IB}(\phi_1(I))] \to [I^2 = \mathrm{IB}(\phi_2(I^1))] \to \ldots \to [y = \mathrm{IB}(\phi_k(I^k))].$$

Since we compute interval bound in each stage, the estimations are far from optimal; see Fig. 1. In practice, wrapping effects appear in neural networks. We will show that intervals grow exponentially, even for linear networks. We consider linear orthogonal ones, as they can be seen as the natural initialization of the deep network (Nowak et al.).

We will need the following lemma which proof is given in the Appendix.

Lemma 2.1. *Let $V = (V_1, \ldots, V_n)$ be a random vector uniformly chosen from the unit sphere in $\mathbb{R}^n$. Let R be a random variable given by $R = |V_1| + \ldots + |V_n|$. Then*

$$\mathbb{E}(R) = \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n} + O(1/\sqrt{n}),\ \mathbb{V}(R) = 1 + \frac{1}{\pi} + O(1/n).$$

Now we will show how a uniform interval bound is processed through an orthogonal map (isometry).

Proposition 2.1. *Let U be a randomly chosen orthogonal map in $\mathbb{R}^n$. Then*

$$\mathrm{IB}(U([-\mathbb{1}, \mathbb{1}])) \approx \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n} \cdot ([-\mathbb{1}, \mathbb{1}] + O(1/\sqrt{n})).$$

Proof. For each fixed $i = 1 \ldots, n$ the i-th row of U is a random vector uniformly chosen from the unit sphere. Thus $U(x) = [U_1(x), \ldots, U_n(x)]^T$. By (1), $\mathrm{IB}(U_i([-\mathbb{1}, \mathbb{1}])) = [-R_i, R_i]$, where R_i is a random variable given by $R_i = |U_{i1}| + \ldots + |U_{in}|$. Now by Lemma 2.1, $\mathbb{E}(R_i) = \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n} + O(1/n)$, $\mathbb{V}(R_i) = 1 + \frac{1}{\pi} + O(1/n)$. By the Chebyshev inequality,

$$\mathrm{P}(|R_i - \mathbb{E}[R_i]| \geqslant a) \leqslant \frac{\mathbb{V}([R_i])}{a^2},$$

and consequently asymptotically for large n

$$\mathrm{P}(|R_i - \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n}| \geqslant a) \leqslant \frac{1 + \frac{1}{\pi} + O(1/n)}{a^2}.$$

Consequently, with an arbitrary large probability

$$U_i \approx \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n} \cdot [-1, 1] + O(1) = \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n} \cdot \left([-1, 1] + O(1/\sqrt{n})\right).$$

$\square$

The following theorem shows that the standard IBP leads to an exponential increase of the bound with respect to the number of layers, even when the true optimal bound does not increase. We obtain the formal proof for the linear layers with orthogonal activations.

Theorem 2.1. *Let $U_1, \ldots, U_k$ be a sequence of orthogonal maps in $\mathbb{R}^n$, and let $U = U_k \circ \ldots \circ U_1$. Let $B_0 = [-\mathbb{1}, \mathbb{1}]$. Then*

$$\mathrm{IB}(U(B_0)) \approx \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n}([-\mathbb{1}, \mathbb{1}] + O(1/\sqrt{n}).$$

Let B_i be defined iteratively by $B_i = \mathrm{IB}(U_i(B_{i-1}))$. Then

$$B_k \approx (\frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n})^k([-\mathbb{1}, \mathbb{1}] + O(1/\sqrt{n}))$$

Proof. The proof follows from the recursive use of the previous proposition. $\square$

Observe, that the above theorem says, that applying standard interval bounds propagation layer after layer leads to exponential increase in the bound, as compared to the true optimal bound. This paper modifies two Doubleton and Affine Arithmetic models, which provide optimal bounds for linear transformations.

3 Doubleton and Affine Arithmetics

As shown in the previous section, classical interval bound propagation leads to an exponential increase in the bounds, even for the case of most superficial linear networks, which implies that it is suboptimal for pre-trained networks. Consequently, we postulate that we should develop methods that obtain strict estimation in the case of linear networks. In this paper, we propose adapting two Doubleton and Affine Arithmetics models for deep neural networks.

Doubleton Arithmetics Doubleton is a class of subsets of $X \subset \mathbb{R}^n$ that are represented in the following form

$$X = \{x + Cr + Qq : r \in \mathbf{r}, q \in \mathbf{q}\},$$

for some $x \in \mathbb{R}^n$, $C \in \mathbb{R}^{n \times m}$, $Q \in \mathbb{R}^{n \times k}$ and $\mathbf{r} \subset \mathbb{R}^m$, $\mathbf{q} \subset \mathbb{R}^k$ are interval vectors (product of intervals) containing zero. For a possibly nonlinear function $f : \mathbb{R}^m \to \mathbb{R}^n$ and a compact set $W \subset \mathbb{R}^m$ we use doubletons to enclose range $f(W)$. The component $x + C\mathbf{r}$ is supposed to store linear approximation to f, while $Q\mathbf{q}$ stores accumulated errors (usually bounds on nonlinear terms) in certain (often orthogonal) coordinate system.

Such a family is one of the most frequently used in validated integration of ODEs (Lohner, 1992; Mrozek & Zgliczyński, 2000) and provides a good balance between accuracy (size of overestimation) add time complexity of operations on such objects. Here we would like to adopt it to the special case of neural networks. We have to extend doubleton arithmetics to functions with different dimensions of domain and codomain and also for non-smooth ReLU frequently used as an activate function in neural networks.

In the context of neural networks Doubleton Arithmetics is promising since we obtain sharp bound when mapping a doubleton by an affine transformation.

Theorem 3.1. *Evaluation of an affine function $A(t) = x_0 + Lt$ over a doubleton $X = x + C\mathbf{r} + Q\mathbf{q}$ is exact, that is*

$$A(X) = \left\{ \tilde{x} + \tilde{C}r + \tilde{Q}q : q \in \mathbf{q}, r \in \mathbf{r} \right\}, \text{ where } \tilde{x} = x_0 + Lx, \quad \tilde{C} = LC, \quad \tilde{Q} = LQ.$$

Consequently, we can process sets described by doubletons through linear layers without any wrapping effect. Enclosing a classical neural network activation function in Doubleton Arithmetics is more challenging. Below we proceed with the formulation how it can be done for a general nonlinear map.

Assume that $f : \mathbb{R}^n \to \mathbb{R}^d$ is a nonlinear function (even not continuous) and assume that for $z \in X = x + C\mathbf{r} + Q\mathbf{q}$ there holds

$$f(z) = x_0 + L(z - x) + e(z)$$

and let us assume that we have computed a bound $e(z) \in \mathbf{e}$ for $z \in X$. Then we have

$$\begin{aligned} f(z) &= x_0 + (LC)r + (LQ)q + e(z) = (x_0 + \mathrm{mid}(\mathbf{e})) + (LC)r + (LQ)q + (e(z) - \mathrm{mid}(\mathbf{e})) \\ &\in \tilde{x} + \tilde{C}\mathbf{r} + \tilde{Q}\tilde{\mathbf{q}}, \end{aligned}$$

where $\tilde{x} = x_0 + \mathrm{mid}(\mathbf{e}) \in \mathbb{R}^d$, $\tilde{C} = LC \in \mathbb{R}^{d \times m}$ and the term $\tilde{Q}\tilde{\mathbf{q}}$ is computed as follows. To simplify notation put $\Delta = \mathbf{e} - \mathrm{mid}(\mathbf{e})$. Let $\tilde{Q} \in \mathbb{R}^{d \times n}$ and $A \in \mathbb{R}^{n \times d}$ be arbitrary matrices so that $\tilde{Q}A = \mathrm{Id}_d$. Then for $q \in \mathbf{q}$ and $\delta \in \Delta$ we have

$$(LQ)q + \delta = \tilde{Q}A(LQq + \delta) = \tilde{Q}\left((ALQ)q + A\delta\right)$$

Now we define $\tilde{\mathbf{q}} := (ALQ)\mathbf{q} + A\Delta \subset \mathbb{R}^d$. It should be emphasized, that it is very important to first evaluate the product of matrices (ALQ) and then multiply the result by the interval vector $\mathbf{q}$. Here is the place when we can reduce wrapping effect provided we make a *good choice* of $\tilde{Q}$ and A. There are various strategies for that.

Strategy 1. If $n = d$ and $LQ \in \mathbb{R}^{n \times n}$ is nonsingular then we may set $\tilde{Q} = LQ$ and $A = \tilde{Q}^{-1}$. Then $\tilde{\mathbf{q}} := \mathbf{q} + \tilde{Q}^{-1}\Delta \subset \mathbb{R}^n$.

Strategy 2 - QR-decomposition. If $d \geqslant n$ then we can first compute QR-decomposition of $LQ = \tilde{Q}R$, where $\tilde{Q} \in \mathbb{R}^{d \times d}$ is orthonormal. Then $A = \tilde{Q}^T$ and $R = ALQ \in \mathbb{R}^{d \times k}$. We can set $\tilde{\mathbf{q}} := R\mathbf{q} + \tilde{Q}^T\Delta \subset \mathbb{R}^d$. If $d < n$ then we may compute QR-decomposition of the leading $d \times d$ block of LQ and proceed as before.

Strategy 3 - QR-decomposition with pivots. In **Strategy 2** we add a preconditioning step – that is permutation of columns of LQ before QR-factorization. The permutation should take into account widths of components $\mathbf{q_i}$ and Δ_i.

We can also try hybrid strategies in the case $n = d$. For instance, we can start from **Strategy 1** and if LQ is singular or close to singular we switch to **Strategy 3**.

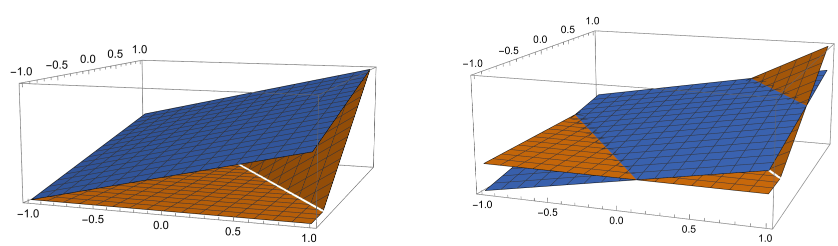

Figure 2: Graphs of ReLU over a hyperplane crossing zero. (Left) first affine approximation of ReLU with $\tau = 1$, that is $\tilde{b}_0 + c\sum a_i t_i$ and (right) its final affine approximation $b_0 + c\sum a_i t_i$.

We can implement ReLU and $\mathrm{softmax}$ in Doubleton Arithmetics thanks to the above methodology. Consequently, we can track how the input interval is propagated through the neural network. Since the transformation by linear mapping does not cause the wrapping effect, we get much better estimates than the classical IBP.

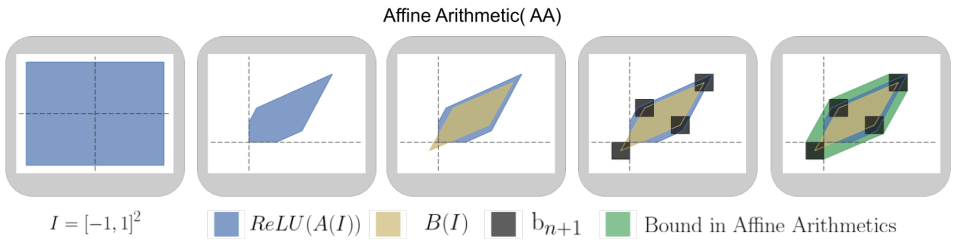

Figure 3: Affine Arithmetic works with more complicated shapes than hypercubes from IBP. In the example, we take Interval $I = [-1, 1]^2$ and see how AA produces an approximation of output from the linear layer with ReLu activation. We use affine transformation from Fig. 1. To approximate AA output from $\mathrm{ReLU}(A(I))$, we first approximate nonlinear function $\mathrm{ReLU}(A(\cdot))$ by linear $B(\cdot)$. Then, we propagate input interval I through $B(\cdot)$. Then we add interval correction, which is equal to the maximal error between $\mathrm{ReLU}(A(I))$ and $B(\cdot)$ denoted by b_{n+1}. Finally, we obtain bound in Affine Arithmetic in the case of mapping interval through a linear layer with linear activation.

Affine Arithmetic The main drawback of the Doubleton Arithmetics is that it is expensive, because it involves multiplication of full dimensional non-sparse matrices. Affine arithmetics (de Figueiredo & Stolfi, 2004) is a concept of reducing overestimation in evaluating an expression in interval arithmetics coming from dependency, that is multiple occurrence of a variable in an expression.

Affine arithmetics keeps track of linear dependencies between variables through evaluation of an expression. Affine Arithmetic gives sharp bounds for linear transformations.

In affine aritmetics we represent subsets of $\mathbb{R}^n$ as a range of an affine functions $A([-1, 1]^m)$ for some affine map $A : \mathbb{R}^m \to \mathbb{R}^n$. Clearly the composition of A with another affine map $B : \mathbb{R}^n \to \mathbb{R}^k$ is again an affine map $B \circ A : \mathbb{R}^m \to \mathbb{R}^k$ and thus the image of $[-1, 1]^m$ via $B \circ A$ is represented as an affine expression with no overestimation.

Let us present on an easy example the main property of affine arithmetics, which shows its superiority over interval arithemtics. Assume we have two expressions $A(x, y) = 1+x+2y$ and $B(x, y, z) = 1-x-2y+z$, where $x, y, z \in I := [-1, 1]$ and we would like to compute a bound on $A(I, I)+B(I, I, I)$. Evaluation in interval arithmetics gives

$$A(I, I) + B(I, I, I) \subset (1 + [-1, 1] + 2[-1, 1]) + (1 - [-1, 1] - 2[-1, 1] + [-1, 1]) = [-5, 9].$$

We see that multiple occurrence of a variable in an expression leads to large overestimation. In affine arithmetics we keep linear track of variables and only in the end we evaluate expression in interval arithmetics. This gives the following (sharp) bound

$$\begin{aligned} A(x, y) + B(x, y, z) &= 2 + z, \\ A(I, I) + B(I, I, I) &= 2 + [-1, 1] = [1, 3]. \end{aligned}$$

Because different affine functions may have different number of arguments it is convenient to treat them (formally) as functions $A : \ell^0 \to \mathbb{R}$, where ℓ^0 is a set of sequences with all but finite number of non-zero elements. Then we have a straightforward interpretation of addition of such functions and multiplication of affine function by a scalar.

To adapt Affine Arithmetic to neural networks we need to implement ReLU and $\mathrm{softmax}$ functions. In the case of nonlinear transformation in Affine Arithmetic we approximate our nonlinear mapping by an affine function with known precision, see Fig. 2. Then we propagate our input interval throught this affine transformation and add a new interval equal to upper bound of the difference between linear approximation and original function, see Fig. 3.

In order to implement ReLU in Affine Arithmetic let us consider an affine function

$$A(t_1, \cdots, t_n) = a_0 + \sum_{i=1}^{n} a_i t_i$$

defined on the cube $t = (t_1, \ldots, t_n) \in [-1, 1]^n =: I^n$ and assume $0 \in A(I^n)$. Clearly the composition $\mathrm{ReLU} \circ A$ is nonlinear and the set $\mathrm{ReLU}(A(I^n))$ cannot be represented exactly as a range of an affine function.

Our strategy is to find an affine map $B : \mathbb{R}^n \to \mathbb{R}$, which approximates well the composition $\mathrm{ReLU} \circ A$ on the hypercube I^n. Bound on the difference

$$\max_{t \in I^n} |\mathrm{ReLU}(A(t)) - B(t)|$$

will be treated as a new variable and finally the range $\mathrm{ReLU}(A(I^n))$ will be covered by a range of an affine function but with $n+1$ variables. This scenario is visualised in Fig. 3.

We impose that B is of the form

$$B(t) = b_0 + \sum_{i=1}^{n} (ca_i)t_i$$

for some $c \in \mathbb{R}$, that is $b_i = ca_i$ for $i > 0$. Put $S := \sum_{i=1}^{n} |a_i|$ and let $M := \sup_{t \in I^n} A(t) = a_0 + S$. Let $\tau \in [0, 1]$ be a parameter to be specified later. We impose that B vanishes for its all arguments being -1, while it reaches maximum value in I^n equal to τM for all arguments equal to 1 – see Fig.2 left panel. This gives the following system of equations with two unknowns

$$\widetilde{b}_0 - cS = 0, \qquad \widetilde{b}_0 + cS = \tau \cdot M.$$

The solution is $\widetilde{b}_0 = \frac{1}{2}\tau M$ and $c = \frac{1}{2}\tau M/S$. The graph of the first affine approximation $\widetilde{B}(t) = \widetilde{b}_0 + c\sum_{i=1}^{n} a_i t_i$ of $\mathrm{ReLU}(A(t))$ is shown in Fig. 2 left panel.

Now, we have to bound the difference between $\widetilde{B}$ and $\mathrm{ReLU} \circ A$ on $[-1, 1]^n$. By the choice of c and $\widetilde{b}_0$ the maximal value of $\widetilde{B}(t)$ in the cube $[-1, 1]^n$ is τM. Hence

$$D_+ := \max_{t \in I^n} \left(\mathrm{ReLU}(A(t)) - \widetilde{B}(t)\right) = \max_{t \in I^n} \left(A(t) - \widetilde{B}(t)\right) = M - \tau M = M(1 - \tau). \quad (2)$$

The minimal value of this difference is achieved, when $(a_0 + \sum a_i t_i) = 0$, that is $\sum a_i t_i = -a_0$ – see Fig. 2 (left panel). This minimal value is then

$$D_- = \min_{t \in I^n} \left(\mathrm{ReLU}(A(t)) - \widetilde{B}(t)\right) = \min_{t \in I^n} \left(0 - \widetilde{B}(t)\right) = -\widetilde{b}_0 + ca_0 = ca_0 - \frac{1}{2}\tau M. \quad (3)$$

Gathering (2)-(3) we obtain

$$\left(\mathrm{ReLU}(A(t)) - \widetilde{B}(t)\right) \in [D_-, D_+], \qquad \text{for } t \in [-1, 1]^n.$$

The above considerations lead to an algorithm for computation of ReLU in the affine arithmetics. Given coefficients $(a_0, \ldots, a_n)$ of an affine function $A(t_1, \ldots, t_n)$ we compute coefficients $(b_0, \ldots, b_{n+1})$ of an affine function $B(t_1, \ldots, t_{n+1})$ so that the range of $B(t)$, $t \in [-1, 1]^{n+1}$ covers the range of $\mathrm{ReLU}(A(t))$, $t \in [-1, 1]^n$ in the following way

$$\begin{aligned} S &= \sum_{i=1}^{n} |a_i|, \quad M = a_0 + S, \quad c = \frac{1}{2}\tau M/S, \quad D_+ = M(1 - \tau), \quad D_- = ca_0 - \frac{1}{2}\tau M, \\ b_0 &= \frac{1}{2}\left(\tau M + D_+ + D_-\right), \quad b_{n+1} = \frac{1}{2}(D_+ - D_-), \quad b_i = ca_i, \quad i = 1, \ldots, n. \end{aligned}$$

There remains to explain how we choose the parameter $\tau \in [0, 1]$. Set $U = \max_{t \in [-1,1]^n} A(t) = a_0 + S$ and $L = \min_{t \in [-1,1]^n} A(t) = a_0 - S$. Experimentally we have found that the choice $\tau \approx \frac{U}{U-L} = \frac{a_0+S}{2S}$ gives reasonable small overestimation of $\mathrm{ReLU} \circ A$ and it is very fast to compute.

The computation of softmax in affine arithmetics is presented in Appendix.

4 Experiments

In this section we present the results obtained by our two proposed methods: Affine and Doubleton Arithmetics. For a fixed point x from the dataset, we define a box $B = x + \varepsilon[-\mathbb{1}, \mathbb{1}]$ and then we compare bounds on the output of a neural network $\Phi(B)$ obtained by means of IBP, DA and AA methods. Additionally we compute Lower Bound (LB) on $\Phi(B)$ as the smallest box (interval hull) containing the set $\{\Phi(\xi_k)\}_{k=1}^{1000}$, where $\{\xi_k\}_{k=1}^{1000} \subset B$ are randomly chosen points. All the experiments

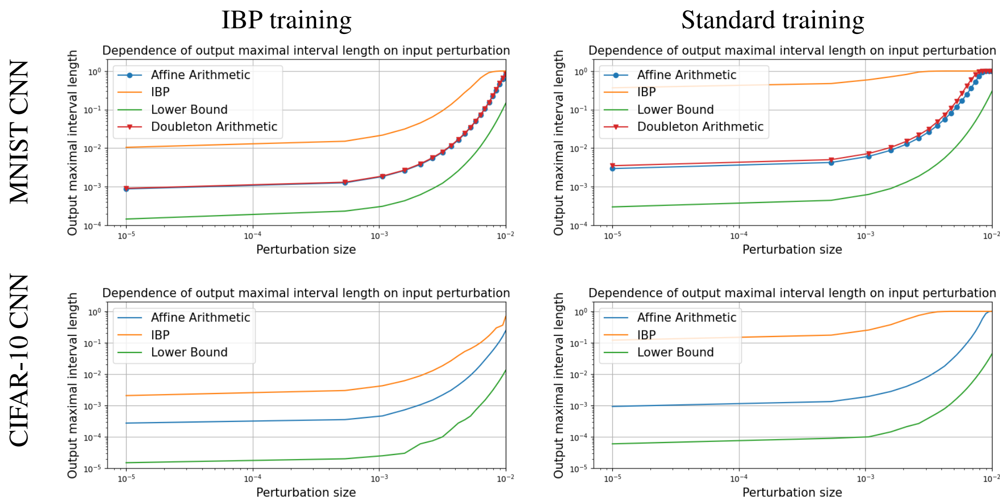

Figure 4: The average maximal diameter of the NN output measured for points near the classification boundary. The X axis represents the perturbation size applied to the data points, while the Y axis shows the average maximal diameter of the NN output in the logarithmic scale. As we can see, the AA and DA methods give better approximation of interval bounds than the IBP method. Note that the DA cannot be calculated for large CNN architectures according to CPU constraints. We can see that IBP training in relation to standard training allows to reduce wrapping effect.

are implemented in C++ with the full control over numerical and rounding errors obtained due to the use of CAPD library (Kapela et al., 2021).

The results presented for the MNIST, CIFAR-10, and SVHN datasets are shown only for the small CNN architecture unless stated otherwise. The results for the Digits dataset are presented using an MLP architecture. For more details about the training hyperparameters, architectures, datasets, and hardware used – see Section C in Appendix.

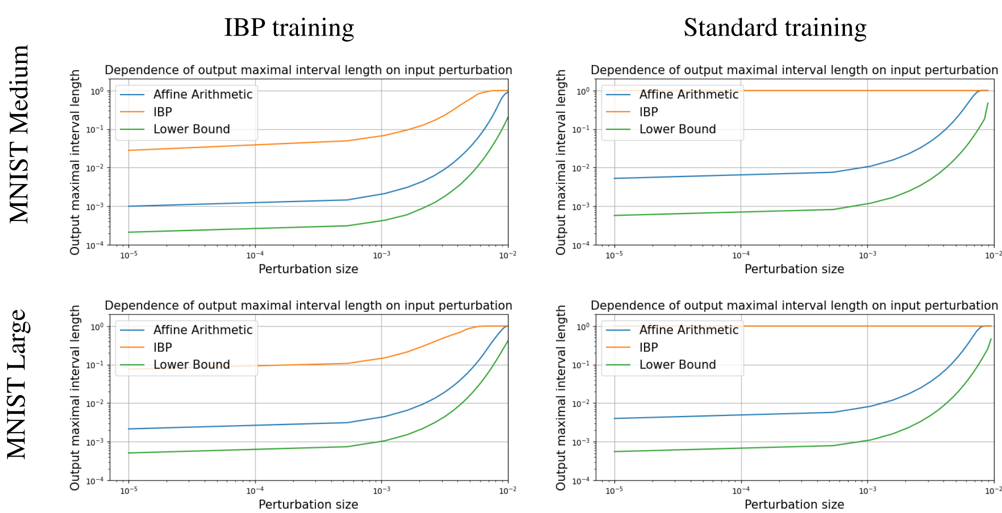

Figure 5: The average maximal diameter of the NN output measured for points near the classification boundary for the medium and large CNN architectures. The X axis represents the perturbation size applied to the data points, while the Y axis shows the average maximal diameter of the NN output in the logarithmic scale.

Interval bounds for points sampled near the decision boundary We compare our methods on points sampled near the decision boundary. We start by selecting one point from each class. From these points, we sample one point and connect it to the remaining points with line segments. For each of these line segments, we sample a point that lies near the decision boundary. We then calculate the interval neural network output for each of these selected points and average maximal diameters of these intervals to assess the model's uncertainty. It is important to emphasize that these selected points may not be actual points from the real dataset, as they are, in fact, convex combinations of points from the real dataset. The experiments are conducted on the MNIST and CIFAR-10. We provide results for neural networks with weights obtained through a classical training procedure (without IBP training) and weights obtained through IBP training as well, for comparison.

As shown in Fig. 4, for a neural network trained using the IBP method ($\epsilon_{\text{train}} = 0.01$) and standard training, the AA and DA methods perform significantly better compared to the IBP method. It is important to highlight that both the AA and DA methods produce nearly identical results. We would like to emphasize, that the AA method gives useful answer even for large perturbation size ϵ, while the IBP even for not very large perturbations gives useless outputs of length 1, which means that the probability is somewhere between 0 and 1. Such phenomenon is well visible in each experiment we conducted – see Figs. 4, 5 and 6.

Influence of network size on interval bounds It is a fair question how interval bounds change depending on the size of a neural network architecture. We address this question by using medium, and large CNN architectures for the MNIST dataset. The architectures were trained using the IBP method with $\epsilon_{\text{train}} = 0.01$, as well as without the IBP method for comparison.

For the medium and large CNN architectures trained on the MNIST dataset (Fig. 5), the AA method produces results close to those of the LB method, significantly outperforming the IBP method. These differences are particularly noticeable when IBP training is not applied. In this case, the AA method once again outperforms the IBP method, while the differences between the LB and AA methods are only slightly worse compared to the scenario when IBP training is used. It is also worth emphasizing that when medium and large CNN architectures are used without IBP training, the neural network becomes extremely uncertain about the investigated data points.

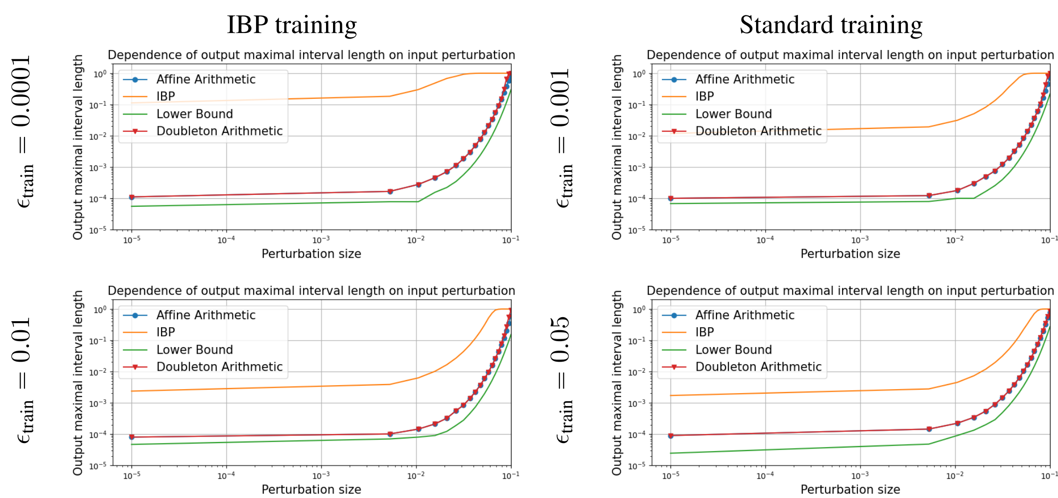

Figure 6: The average maximal diameter of the NN output measured for points near the classification boundary for network trained with different interval lengths $\epsilon_{\text{train}}$for the Digits dataset. The X axis represents the perturbation size applied to the data points, while the Y axis shows the average maximal diameter of the NN output in the logarithmic scale.

Influence of various perturbation sizes used in IBP-based training on the resulting interval bounds Generally, the presented plots in Fig. 6 show that the larger the perturbation size applied during IBP training, the smaller the difference between the results obtained using the IBP and AA/DA methods. However, it is important to emphasize that increasing the perturbation size during IBP training makes training a neural network more difficult, leading to challenges in achieving satisfactory accuracy. Therefore, our proposed methods offer a much easier way to reduce the wrapping effect, and they can be applied regardless of whether IBP training is used, making them both more practical and efficient for real-world applications.

5 Conclusion

This paper analyzes *wrapping effect* in a neural network. We show that for linear models, interval bounds can grow exponentially. Such effects have a strong influence on the IBP certification of neural networks. To solve such a problem, we propose adapting two models from strict numerical calculations: Doubleton and Affine Arithmetics. Both models give sharp bounds for linear transformations. The experimental section shows that Affine Arithmetic returns bounds close to optimal within reasonable computational time.

Limitations Doubleton Arithmetics provides near-optimal bounds, but the computational complexity is $O(n^3)$, where n is the largest dimension of the hidden layers. Even for the small CNN architecture on the CIFAR-10 and SVHN datasets, the computation time was unacceptably high.

References

Martin Berz and Kyoko Makino. New methods for high-dimensional verified quadrature. *Reliable Computing*, 5(1):13–22, 1999. ISSN 1385-3139. doi: 10.1023/A:1026437523641.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*, pp. 387–402. Springer, 2013.

Luiz Henrique de Figueiredo and Jorge Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1):147–158, Dec 2004. ISSN 1572-9265. doi: 10.1023/B:NUMA.0000049462.70970.b6. URL https://doi.org/10.1023/B:NUMA.0000049462.70970.b6.

Claudio Ferrari, Mark Niklas Muller, Nikola Jovanovic, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *ICLR*, 2022.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

William Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.

Tomasz Kapela, Marian Mrozek, Daniel Wilczak, and Piotr Zgliczyński. CAPD:: DynSys: a flexible C++ toolbox for rigorous numerical analysis of dynamical systems. *Communications in nonlinear science and numerical simulation*, 101:105578, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Rudolf J. Lohner. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In *Computational ordinary differential equations (London, 1989)*, volume 39 of *Inst. Math. Appl. Conf. Ser. New Ser.*, pp. 425–435. Oxford Univ. Press, New York, 1992.

Siwen Luo, Hamish Ivison, Soyeon Caren Han, and Josiah Poon. Local interpretations for explainable natural language processing: A survey. *ACM Computing Surveys*, 56(9):1–36, 2024.

Kyoko Makino and Martin Berz. Rigorous integration of flows and odes using taylor models. In *Proceedings of the 2009 Conference on Symbolic Numeric Computation*, SNC '09, pp. 79–84, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-664-9. doi: 10.1145/1577190.1577206.

Yuhao Mao, Mark Niklas Müller, Marc Fischer, and Martin Vechev. Understanding certified training with interval bound propagation. In *ICLR*, 2024.

IEEE Std 1788.1-2017. Ieee standard for interval arithmetic (simplified). *IEEE Std 1788.1-2017*, pp. 1–38, 2018. doi: 10.1109/IEEESTD.2018.8277144.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pp. 3578–3586. PMLR, 2018.

Marian Mrozek and Piotr Zgliczyński. Set arithmetic and the enclosing problem in dynamics. *Ann. Polon. Math.*, 74:237–259, 2000. ISSN 0066-2216.

Arnold Neumaier. *The wrapping effect, ellipsoid arithmetic, stability and confidence regions*. Springer, 1993.

Aleksandra Nowak, Łukasz Gniecki, Filip Szatkowski, and Jacek Tabor. Sparser, better, deeper, stronger: Improving static sparse training with exact orthogonal initialization. In *Forty-first International Conference on Machine Learning*.

Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Fast certified robust training with short warmup. *Advances in Neural Information Processing Systems*, 34:18335–18349, 2021.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. *Advances in neural information processing systems*, 31, 2018.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems*, 32, 2019.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. *Dive into deep learning*. Cambridge University Press, 2023.

Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *Advances in neural information processing systems*, 35:1656–1670, 2022.

A Integral computations

We will show the following lemma, which gives a detailed estimations for Lemma 2.1.

Lemma A.1. *Let $V = (V_1, \ldots, V_n)$ be a random vector uniformly chosen from the unit sphere in $\mathbb{R}^n$. Let R be a random variable given by*

$$R = |V_1| + \ldots + |V_n|.$$

Then

$$ER = \frac{2n}{n-1} \frac{\Gamma(\frac{n}{2})}{\sqrt{\pi}\Gamma(\frac{n-1}{2})}, \; ER^2 = 1 + \frac{2}{\pi}(n - \frac{1}{n-2}), \; VR = ER^2 - (ER)^2.$$

Moreover, we have the asymptotics

$$ER = \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n} + \frac{1}{2\sqrt{2\pi n}} + O(n^{-3/2}), \; VR = 1 + \frac{1}{\pi} + O(1/n).$$

Proof. To calculate ER, we compute

$$ER = \frac{1}{S_{n-1}} \int_{x:\|x\|=1} |x|_1 dS(x) = \frac{1}{S_{n-1}} \int_{x:\|x\|=1} x_1 + \ldots + x_n dS(x) =$$

$$\frac{2^n}{S_{n-1}}\int_{x:\|x\|=1,x_1,\dots,x_n\geqslant 0} x_1+\ldots+x_n dS(x) = \frac{n2^n}{S_{n-1}}\int_{x:\|x\|=1,x_1,\dots,x_n\geqslant 0} x_1 dS(x) =$$
$$\frac{2n}{S_{n-1}}\int_{x:\|x\|=1,x_1\geqslant 0} x_1 dS(x) = \frac{2n}{S_{n-1}}\int_0^1 x_1 S_{n-2}(\sqrt{1-x_1^2})^{n-2}\cdot\frac{1}{\sqrt{1-x_1^2}}dx_1 =$$
$$\frac{2nS_{n-2}}{S_{n-1}}\int_0^1 x_1(1-x_1^2)^{\frac{n-3}{2}}dx_1 = \frac{nS_{n-2}}{S_{n-1}}\int_0^1 t^{\frac{n-3}{2}}dt =$$
$$= \frac{2nS_{n-2}}{(n-1)S_{n-1}} = \frac{2n}{n-1}\frac{\Gamma(\frac{n}{2})}{\sqrt{\pi}\Gamma(\frac{n-1}{2})}$$

Finally we obtain the assymptotic expansion

$$\approx \frac{2}{\sqrt{\pi}}[\frac{\sqrt{n}}{\sqrt{2}} + \frac{1}{4\sqrt{2n}} + O(1/n^{3/2})]. = \frac{\sqrt{2}}{\sqrt{\pi}}\sqrt{n} + \frac{1}{2\sqrt{2\pi n}} + O(n^{-3/2}).$$

We proceed to computation of R^2. We have

$$ER^2 = \frac{1}{S_{n-1}}\int_{x:\|x\|=1}(|x|_1+\ldots+|x_n|)^2 dS(x)$$
$$= \frac{1}{S_{n-1}}\int_{x:\|x\|=1}|x|_1^2+\ldots+|x_n|^2 dS(x) + \frac{n(n-1)}{S_{n-1}}\int_{x:\|x\|=1}|x|_1|x_2|dS(x) =$$
$$= 1 + \frac{4n(n-1)}{S_{n-1}}\int_{x:\|x\|=1,x_1,x_2\geqslant 0} x_1x_2 dS(x)$$

Now

$$\int_{x:\|x\|=1,x_1,x_2\geqslant 0} x_1x_2 dS(x) = \int_{x_1,x_2\geqslant 0, x_1^2+x_2^2\leqslant 1} x_1x_2 S_{n-3}\sqrt{1-(x_1^2+x_2^2)}^{n-3}\frac{1}{\sqrt{1-(x_1^2+x_2^2)}}dx_1dx_2$$

Now

$$\int_{x_1,x_2\geqslant 0, x_1^2+x_2^2\leqslant 1} x_1x_2(1-(x_1^2+x_2^2))^{n/2-2}dx_1dx_2 = \int_0^{\pi/2}\int_0^1 r\sin\phi r\cos\phi(1-r^2)^{n/2-2}rdrd\phi$$
$$= \int_0^{\pi/2}\frac{\sin 2\phi}{2}d\phi\cdot\frac{1}{2}\int_0^1(1-t)t^{n/2-2}dt = \frac{1}{4}\cdot(\frac{2}{n-2}-\frac{2}{n}) = \frac{1}{n(n-2)}.$$

Finally

$$ER^2 = 1 + 4\frac{n-1}{n-2}\frac{S_{n-3}}{S_{n-1}} = 1 + 4\frac{n-1}{n-2}\frac{n/2}{\pi} = 1 + \frac{2}{\pi}(n - \frac{1}{n-2}).$$

Clearly $VR = ER^2 - (ER)^2$, which trivially yields the asymptotic expansion

$$VR = ER^2 - (ER)^2 = 1 + \frac{1}{\pi} + O(1/n).$$

□

B Doubleton and affine arithmetics

Softmax in affine arithmetics Assume we have an affine function $At = x + Lt$ defined on the cube $t = (t_1, \ldots, t_n) \in [-1, 1]^n =: I^n$, with $x \in \mathbb{R}^m$, $L \in \mathbb{R}^{m\times n}$. Our goal is to find an affine map

$$Bt = \tilde{x} + \tilde{L}t$$

and a vector $e \in \mathbb{R}^m$, so that for $i = 1, \ldots, m$ there holds

$$\max_{t\in I^n} |(\mathrm{softmax}(A(t)) - B(t))_i| \leqslant e_i.$$

The vector $\tilde{x}$ and the matrix $\tilde{L}$ will be computed from first order Taylor expansion of softmax. A bound on error term e will be computed from second derivatives.

Recall, that for $z \in \mathbb{R}^m$

$$\mathrm{softmax}(z) = (s_1, \ldots, s_m) := \left(\frac{\exp(z_i)}{\sum_{j=1}^m \exp(z_j)}, \ldots, \frac{\exp(z_m)}{\sum_{j=1}^m \exp(z_j)}\right).$$

In order to avoid numerical instabilities in evaluation of the above expression we take $R = \|z\|_\infty$ and compute $\mathrm{softmax}(z) = \mathrm{softmax}(z_1 - R, \ldots, z_m - R)$.

It is well known that the Jacobian of softmax is given by

$$J(z) := D\mathrm{softmax}(z) = \begin{bmatrix} s_1(1-s_1) & -s_1 s_2 & \ldots & -s_1 s_m \\ -s_1 s_2 & s_2(1-s_2) & \ldots & -s_2 s_m \\ \vdots & \vdots & \ddots & \vdots \\ -s_m s_1 & \cdots & -s_{m-1}s_m & s_m(1-s_m) \end{bmatrix}.$$

Thus, we can compute a linear approximation of softmax by

$$B(t) = \tilde{x} + \tilde{L}t = \mathrm{softmax}(x) + (J(x)L)\, t.$$

In the above $\mathrm{softmax}(x)$ and $J(x)$ are evaluated at a single point and therefore neither dependency error nor wrapping effect is present.

The error term e_i can be bounded using second order Taylor expansion. We would like to find a bound

$$\left|(I^n)^T D^2 g_i(I^n) I^n\right| \leqslant e_i, \quad i = 1, \ldots, m,$$

where $g(t) = \mathrm{softmax}(x + Lt)$. Differentiation of $Dg(t) = J(x + Lt)L$ gives

$$D^2 g_i(t) \quad = \quad L^T D J_i(x + Lt)L.$$

There remains to derive formula for $DJ_i(z)$. Differentiation gives

$$\begin{aligned} \frac{\partial J_{ij}(z)}{\partial z_c} &= \frac{\partial}{\partial z_c}\left(\delta_{ij}s_i - s_i s_j\right) \\ &= (\delta_{ijc}s_i - \delta_{ij}s_i s_c) - (\delta_{ic}s_i - s_i s_c)s_j - s_i(\delta_{jc}s_j - s_j s_c) \\ &= \delta_{ijc}s_i - \delta_{ij}s_i s_c - \delta_{ic}s_i s_j - \delta_{jc}s_i s_j + 2 s_i s_j s_c. \end{aligned}$$

Evaluation of the above formula in interval arithmetics leads to a rough bound on the error term e_i. We will show, however, that increasing time complexity we can significantly reduce dependency problem in this expression.

Evaluation of g_i and products $g_i g_c$ and $g_i g_c g_r$. We have

$$g_i(t) = \frac{\exp\left(x_i + L_{i1}t_1 + \ldots + L_{in}t_n\right)}{\sum_{j=1}^m \exp\left(x_j + L_{j1}t_1 + \ldots L_{jn}t_n\right)} \tag{4}$$

Dependency in (4) can be reduced using equivalent formula

$$g_i(t) = \frac{\exp\left(y_i\right)}{\sum_{j=1}^m \exp\left(y_j + (L_{j1} - L_{i1})t_1 + \ldots + (L_{jn} - L_{in})t_n\right)}, \tag{5}$$

where $R = \max_{i=1,\dots,m} x_i$ and $y_i = x_i - R$.

Let us recall an important in this context property of interval arithmetics. It is well known that multiplication is not distributive, that is for intervals a, b, c there holds $a(b+c) \subset ab + ac$. However, if all intervals are nonnegative then we have equality. Such situation appears in evaluation of the product

$$\begin{aligned} g_i(t)g_c(t) &= \left(\frac{\exp(y_i)}{\sum_{j=1}^m \exp\left(y_j + \sum_{p=1}^n (L_{jp} - L_{ip})t_p\right)}\right)\left(\frac{\exp(y_c)}{\sum_{k=1}^m \exp\left(y_k + \sum_{p=1}^n (L_{kp} - L_{cp})t_p\right)}\right) \\ &= \frac{\exp(y_i + y_c)}{\sum_{j,k=1}^m \exp\left(y_j + y_k + \sum_{p=1}^n (L_{jp} + L_{kp} - L_{ip} - L_{cp})t_p\right)}. \end{aligned}$$

The above two expressions, when evaluated in interval arithmetics, may lead to different bounds (of course they can be intersected). Time complexity of the second evaluation is $O(M^2N)$ while direct evaluation (first expression) is of order $O(MN)$. However, $\mathrm{softmax}$ is applied to the output of last layer in a neural network, which is usually of low dimension and therefore this should not be a serious additional cost.

Similarly, we can evaluate the product of three functions as

$$g_i(t)g_c(t)g_r(t) = \frac{\exp(y_i + y_c + y_r)}{\sum_{j,k,s=1}^m \exp\left(y_j + y_k + y_s + \sum_{p=1}^n (L_{jp} + L_{kp} + L_{sp} - L_{ip} - L_{cp} - L_{rp})t_p\right)}$$

and intersect the result with direct multiplication of three intervals $g_i(t)$, $g_c(t)$ and $g_r(t)$.

C Experimental setting

Datasets We use the following publicly available datasets: 1) MNIST dataset, consisting of 60,000 training and 10,000 testing 28×28 pixel gray-scale images of 10 classes of digits; 2) CIFAR-10 dataset, consisting of 50,000 training and 10,000 testing 32×32 colour images in 10 classes; 3) SVHN dataset, consisting of 600,000 32×32 pixel colour images of 10 classes of digits; 4) Digits dataset, consisting of 1797 8×8 pixel gray-scale images of 10 classes of digits.

Architectures We use three CNN architectures (small, medium and large) as defined in Table 1 in Gowal et al. (2018). Additionally, we consider an MLP architecture consisting of four hidden layers with 100 neurons per layer. A classification head is added on top of these layers.

Training parameters During training, we use the Adam optimizer Kingma & Ba (2017) with the default configuration of $\beta_1 = 0.9$ and $\beta_2 = 0.999$, but with different learning rates (lr) across all datasets. We consistently use the ReLU activation function. Whenever a scheduler is mentioned, we apply the MultiStepLR scheduler with a default multiplicative learning rate decay factor set to 0.1. The scheduler steps are applied twice: once after $\frac{1}{3}$ of the total number of iterations and once after $\frac{2}{3}$ of the total number of iterations. Additionally, there is a parameter κ scheduled over the entire training process as $\kappa_i = \max\{1 - 0.00005 \cdot i, \kappa_{max}\}$, where i denotes the current training iteration and κ_{max} is set to 0.5. A perturbation value ϵ grows linearly from 0 at the beginning of training to the ϵ_{max} hyperparameter value at the midpoint of the total number of iterations. The considered ϵ_{max} values are from the set $\{0.0001, 0.001, 0.01, 0.05, 0.1\}$ and remain the same regardless of the architecture used. We use 10% of the training samples as the validation set.

- For the MNIST, SVHN, and CIFAR-10 datasets, we train small, medium, and large CNNs using the best set of hyperparameters identified in Gowal et al. (2018). We apply the same normalization and augmentation scheme. The only differences are in the epsilons ϵ used during training and the number of epochs. We decreased the total number of epochs for the CIFAR-10 and SVHN datasets to 100 for the large CNN.
- For Digits, we train the MLP for 50 epochs with batch sizes of 32. No normalization or augmentation is applied. The rest of the hyperparameters remain the same as for the MNIST, SVHN, and CIFAR-10 datasets.

Hardware and software resources used The implementation is done in Python 3.10.13, utilizing libraries such as PyTorch 2.3.1 with CUDA support, NumPy 1.26.4, Pandas 2.1.1, and others. Most computations are performed on an NVIDIA GeForce RTX 4090 GPU, with some training sessions also conducted on NVIDIA GeForce RTX 3080 and NVIDIA DGX GPUs. The experiments involving Affine and Doubleton Arithmetics were implemented using the CAPD library (Kapela et al., 2021).

D EXPERIMENTAL RESULTS

Interval bounds for partially masked data In this subsection, we aim to present the interval bounds obtained through a neural network for data from the Digits and SVHN datasets. We sample 10 points, each belonging to a single class, and apply a mask where 50% of the values are masked (replaced by zero) and the remaining values stay unchanged. We then measure the average diameter of the neural network output.

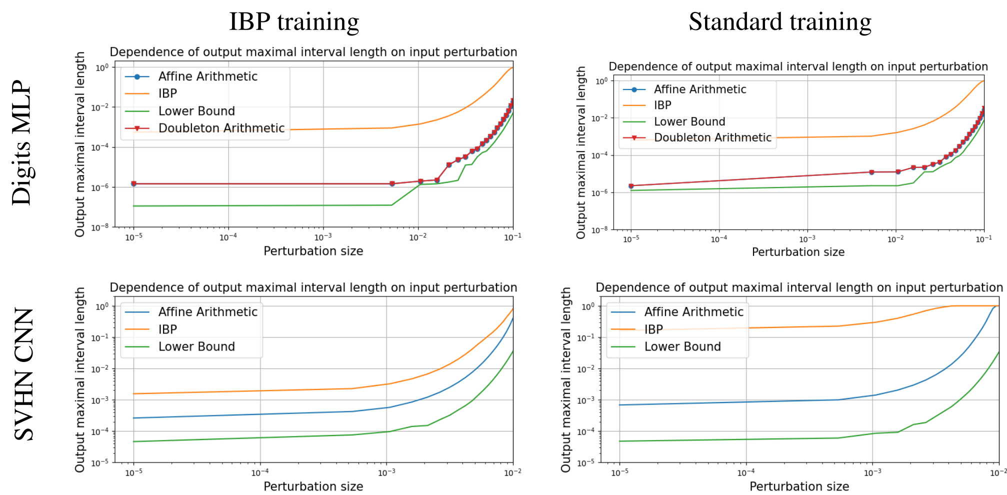

Figure 7: The average maximal diameter of the NN output measured for points near the classification boundary in the case where parts of the images was masked. The X axis represents the perturbation size applied to the data points, while the Y axis shows the average maximal diameter of the NN output in the logarithmic scale.

Even for partially masked data, the output interval bounds obtained using the AA and DA methods are very close to those of the LB methods (Fig. 7), significantly outperforming the IBP method.

These results indicate that the AA and DA methods, compared to the IBP method, effectively minimizes the wrapping effect in neural networks. Consequently, these methods can be regarded as a viable approach for quantifying the uncertainty of a neural network's output when some pixels of an image are masked with a value of 0.