Maximum Resilience of Artificial Neural Networks

Chih-Hong Cheng(✉), Georg Nührenberg(✉), and Harald Ruess

fortiss - An-Institut Technische Universität München,
Guerickestr. 25, 80805 Munich, Germany
{cheng,nuehrenberg,ruess}@fortiss.org

Abstract. The deployment of Artificial Neural Networks (ANNs) in safety-critical applications poses a number of new verification and certification challenges. In particular, for ANN-enabled self-driving vehicles it is important to establish properties about the resilience of ANNs to noisy or even maliciously manipulated sensory input. We are addressing these challenges by defining resilience properties of ANN-based classifiers as the maximum amount of input or sensor perturbation which is still tolerated. This problem of computing maximum perturbation bounds for ANNs is then reduced to solving mixed integer optimization problems (MIP). A number of MIP encoding heuristics are developed for drastically reducing MIP-solver runtimes, and using parallelization of MIP-solvers results in an almost linear speed-up in the number (up to a certain limit) of computing cores in our experiments. We demonstrate the effectiveness and scalability of our approach by means of computing maximum resilience bounds for a number of ANN benchmark sets ranging from typical image recognition scenarios to the autonomous maneuvering of robots.

1 Introduction

The deployment of Artificial Neural Networks (ANNs) in safety-critical applications such as medical image processing or semi-autonomous vehicles poses a number of new assurance, verification, and certification challenges [2,5]. For ANN-based end-to-end steering control of self-driving cars, for example, it is important to know how much noisy or even maliciously manipulated sensory input is tolerated [12]. Here we are addressing these challenges by establishing maximum and verified bounds for the resilience of given ANNs on these kinds of input disturbances.

More precisely, we are defining and computing safe perturbation bounds for multi-class ANN classifiers. This measure compares the relative ratio-ordering of multiple, so-called *softmax* output neurons for capturing scenarios where one only wants to consider inputs that classify to a certain class with high probability. The problem of finding minimal perturbation bounds is reduced to solving a corresponding *mixed-integer programming* (MIP). In particular, the encoding of some non-linear functions such as *ReLU* and *max-pooling* nodes require the

D. D'Souza and K. Narayan Kumar (Eds.): ATVA 2017, LNCS 10482, pp. 251–268, 2017.
DOI: 10.1007/978-3-319-68167-2_18

introduction of integer variables. These integer constraints are commonly handled by off-the-shelf MIP-solvers such as CPLEX[1] which are based on branch-and-bound algorithms. In the MIP reduction, a number of nonlinear expressions are linearized using a variant of the well-known *big-M* [8] encoding strategy. We also define a dataflow analysis [6] for generating relatively small big-M as the basis for speeding up MIP solving. Other important heuristics in encoding the MIP problem include the usage of solving several substantially simpler MIP problems for speeding up the overall generation of satisfying instances by the solver. Lastly, branch-and-bound is run in parallel on a number of computing cores.

We demonstrate the effectiveness and scalability of our approach and encoding heuristics by computing maximum perturbation bounds for benchmark sets such as MNIST [13] and agent games [14]. These cases studies include ANNs for image recognition and for high-level maneuver decisions for autonomous control of a robot. Using the heuristic encodings outlined above we experienced a speed-up of about two orders of magnitude compared with vanilla MIP encodings. Moreover, parallelization of branch-and-bound [23] on different computing cores can yield, up to a certain threshold, linear speed-ups using a high-performance parallelization framework.

The practical advantages of our approach for validating and qualifying ANNs for safety-relevant applications are manifold. First, perturbation bounds provide a formal interface between sensor sets and ANNs in that they provide a maximum tolerable bound on possible sensor errors. These *assume-guarantee* interfaces therefore form the basis for decoupling the design of sensor sets from the design of the classifier itself. Second, our method also computes minimally perturbed inputs of different classification, which might be included into ANN training sets for potentially improving classification results. Third, maximum perturbation bounds are a useful measure of the resilience of an ANN towards (adversarial) perturbation, and also for objectively comparing different ANNs. Last, large perturbation bounds are intuitively inversely related with the problem of *overfitting*, that is poor generalization to new inputs, which is a common issue with ANNs.

An overview of concrete problems and various approaches to the safety of machine learning is provided in [2]. We compare our results only with work that is most closely related to ours. Techniques including the generation of test cases [7,15,16] or strengthening the resistance of a network with respect to adversarial perturbation [17] are used for validating and improving ANNs. In contrast to our work, these methods do not actually establish verified properties on the input-output behavior of ANNs. Formal methods-based approaches for verifying ANNs include abstraction-refinement based approaches [18], bounded model checking for neural network for control problems [21] and neural network verification using SMT solvers or other specialized solvers [9,11,19]. Instead we rely on solving MIP problems and parallelization of branch-and-bound algorithms. In contrast to previous approaches we also go beyond verification and solve

[1] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

optimization problems for ANNs for establishing maximum perturbation bounds. These kinds of problems might also be addressed in SMT-based approaches either by using binary search over SMT or by using SMT solvers that support optimization such as νZ [4], but it is not clear how well these approaches scale to complex ANNs. Recent work also targets ReLU [11] or application of a single image [3,9] (point-wise robustness or computing measures by taking samples). Our proposed resilience measure for ANNs goes beyond [3,9,11] in that it applies to multi-classification network using the softmax descriptor. Moreover, our proposed measure is a property of the classification network itself rather than just a property of a single image (as in [9]) or by only taking samples from the classifier without guarantee (as in [3]).

The paper is structured as follows. Section 2 reviews the foundations of feed-forward ANNs. Section 3 presents an encoding of various neurons in terms of linear constraints. Section 4 defines our measure for quantifying the resilience of an ANN, that is, its capability to tolerate random or even adversarial input perturbations. Section 5 summarizes our MIP encoding heuristics for substantially increasing the performance of the MIP-solver in establishing in minimal perturbation bounds of ANN. Finally, we present the results of some of our experiments in Sect. 6, and we describe possible improvements and extensions in Sect. 7.

2 Preliminaries

We introduce some basic concepts of *feed-forward artificial neural networks* (ANN) [1]. These networks consist of a sequence of layers labeled from $l = 0, 1, \ldots, L$, where 0 is the index of the *input layer*, L is the *output layer*, and all other layers are so-called *hidden layers*. For the purpose of this paper we assume that each input is of bounded domain. Superscripts $^{(l)}$ are used to index layer l-specific variables, but these superscripts may be omitted for input layers. Layers l are comprised of *nodes* $n_i^{(l)}$ (so-called neurons), for $i = 0, 1, \ldots, d^{(l)}$, where $d^{(l)}$ is the *dimension* of the layer l. By convention nodes of index 0 have a constant output 1; these *bias nodes* are commonly used for encoding activation thresholds of neurons. In a feed-forward net, nodes $n_j^{(l-1)}$ of layer $l-1$ are connected with nodes $n_i^{(l)}$ in layer l by means of directed edges of *weight* $w_{ji}^{(l)}$. For the purpose of this paper we are assuming that all weights in a network have fixed values, since we do not consider re-learning. Figure 1 illustrates a small feed-forward network structure with four layers, where each layer comes with a different type of node functions, which are also main ingredients of *convolutional neural networks*. These node functions are specified in Fig. 2. The first hidden layer of the network in Fig. 1 is a *fully-connected ReLU layer*. Node $n_2^{(1)}$, for example, computes the weighted linear sum of all inputs from the previous layer as $\mathsf{im}_2^{(1)}$, and outputs the maximum of 0 and this weighted sum. The second hidden layer is using *max-pooling* for down-sampling an input representation by reducing its dimensionality; node $n_1^{(2)}$, for example, just outputs the maximum of its inputs. Node $n_1^{(3)}$ in the output layer applies the sigmoid-shaped tan^{-1} on the weighted linear input sum.

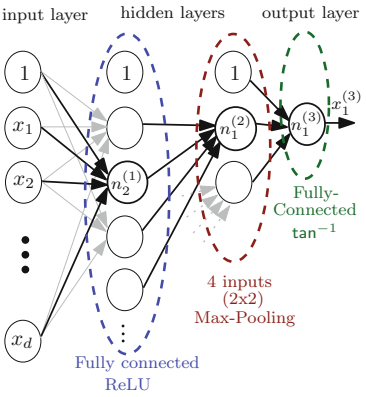

Fig. 1. An illustration of how a neural network is defined.

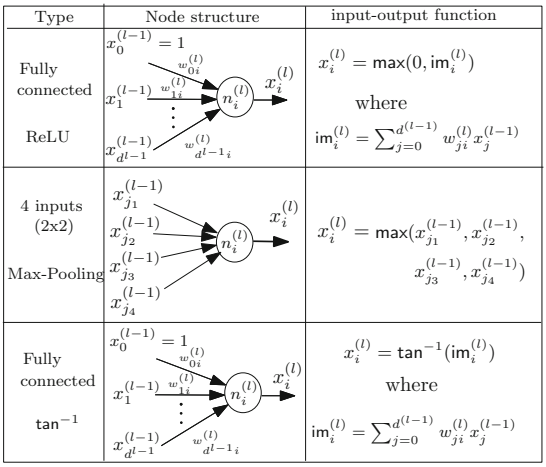

Fig. 2. Input-output function neurons.

Given an input to the network these node functions are applied successively from layer 0 to $L-1$ for computing the corresponding network output at layer L. For $l = 1$ to L we use $x_i^{(l)}$ to denote the output value of node $n_i^{(l)}$ and $x_i^{(l)}(a_1, \ldots, a_d)$ denotes the output value $x_i^{(l)}$ for the input $a_1, \ldots, a_d$, sometimes abbreviated by $x_i^{(l)}(a)$.

For the purpose of multi-class classification, outputs in layer L are often transformed into a *probability distribution* by means of the softmax function

$$\frac{\mathsf{e}^{x_i^{(L-1)}}}{\sum_{j=1,\ldots,d^L} \mathsf{e}^{x_j^{(L-1)}}}.$$

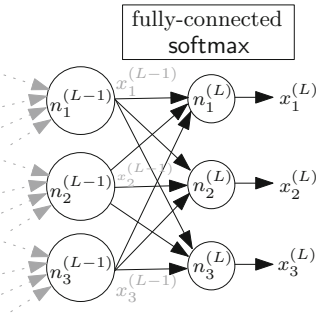

Fig. 3. Topological structure for an output layer with 3 neurons using softmax.

In this way, the output $x_i^{(L)}$ is interpreted as the probability of the input to be in class i. For the inputs $x_1^{(L-1)} = -1$, $x_2^{(L-1)} = 2$, $x_3^{(L-1)} = 3$ of the nodes in Fig. 3, for example, the corresponding outputs $(0.0132, 0.2654, 0.7214)$ for $(x_1^{(L)}, x_2^{(L)}, x_3^{(L)})$ sum up to 1.

3 Arithmetic Encoding of Artificial Neural Networks

In a first step, we are encoding the behavior of ANNs in terms of linear arithmetic constraints. In addition to [11] we are also considering tan^{-1}, max-pooling and

softmax nodes as commonly found in many ANNs in practice. These encodings are based on the input-output behavior of every node in the network, and the main challenge is to handle the non-linearities, which are arising from non-linear activation functions (e.g., ReLU and $\tan^{-1}$), max-pooling and softmax nodes.

Constraints for ReLU and $\tan^{-1}$ nodes as defined in Fig. 2 are separated into, first, an equality constraint (1) for the intermediate value $\mathsf{im}_i^{(l)}$ and, second, several linear constraints for encoding the non-linear behavior of these nodes.

$$\mathsf{im}_i^{(l)} = \sum_{j=0,\ldots,d^{(l-1)}} w_{ji}^{(l)} x_j^{(l-1)} \tag{1}$$

We now describe the encoding of the non-linear functions ($x_i^{(l)} = \mathsf{max}(0, \mathsf{im}_i^{(l)})$ or $x_i^{(l)} = \tan^{-1}(\mathsf{im}_i^{(l)})$).

Encoding ReLU activation function. The non-linearity in ReLU constraints $x_i^{(l)} = \mathsf{max}(0, \mathsf{im}_i^{(l)})$ is handled using the well-known big-M method [8], which introduces a binary integer variable $b_i^{(l)}$ together with a positive constant $M_i^{(l)}$ such that $-M_i^{(l)} \leq \mathsf{im}_i^{(l)}$ and $x_i^{(l)} \leq M_i^{(l)}$ for all possible values of $\mathsf{im}_i^{(l)}$ and $x_i^{(l)}$. A derivation of the following reduction is listed in the appendix.

Proposition 1. $x_i^{(l)} = \mathit{max}(0, \mathit{im}_i^{(l)})$ *iff the constraints* (2a) *to* (4b) *hold.*

$$x_i^{(l)} \geq 0 \tag{2a}$$
$$x_i^{(l)} \geq \mathit{im}_i^{(l)} \tag{2b}$$

$$\mathit{im}_i^{(l)} - b_i^{(l)} M_i^{(l)} \leq 0 \tag{3a}$$
$$\mathit{im}_i^{(l)} + (1 - b_i^{(l)}) M_i^{(l)} \geq 0 \tag{3b}$$

$$x_i^{(l)} \leq \mathit{im}_i^{(l)} + (1 - b_i^{(l)}) M_i^{(l)} \tag{4a}$$
$$x_i^{(l)} \leq b_i^{(l)} M_i^{(l)} \tag{4b}$$

The efficiency of a MIP-solver via big-M encoding heavily depends on the size of $M_i^{(l)}$, because MIP-solvers typically relax binary integer variables to real-valued variables, resulting in a weak LP-relaxation for large big-Ms. It is therefore essential to choose relatively small values for $M_i^{(l)}$. We apply static analysis [6] based on interval arithmetic for propagating the bounded input values through the network, as the basis for generating "good" values for $M_i^{(l)}$.

Max-Pooling. The output $x_i^{(l)}$ of a max-pooling node is rewritten as $x_i^{(l)} = \mathsf{max}(\mathsf{im}_1, \mathsf{im}_2)$, where $\mathsf{im}_1 = \mathsf{max}(x_{j_1}^{(l-1)}, x_{j_2}^{(l-1)})$ and $\mathsf{im}_2 = \mathsf{max}(x_{j_3}^{(l-1)}, x_{j_4}^{(l-1)})$. Encoding the $\mathsf{max}(x_1, x_2)$ function into MIP constraints is accomplished by introducing three binary integer variables to encode $y = \mathsf{max}(x_1, x_2)$ using the big-M method.

Property-directed encoding of softmax. The exponential function in the definition of softmax, of course, can not be encoded into a linear MIP constraint. However, using the proposition below, one confirms that if the property to be analyzed does not consider the concrete value of output values from neurons but only the ratio ordering, then (1) it suffices to omit the construction of the output layer, and (2) one may rewrite the property by replacing each $x_i^{(L)}$ by $x_i^{(L-1)}$.

Proposition 2. *Given a feed-forward ANN with* softmax *output layer and a constant* $\alpha > 0$*, then for all* $i, j \in \{1, \ldots, d^{(L)}\}$*:*

$$x_{i_1}^{(L)} \geq \alpha\, x_{i_2}^{(L)} \Leftrightarrow x_{i_1}^{(L-1)} \geq \ln(\alpha) + x_{i_2}^{(L-1)}.$$

This equivalence is simply derived by using the definition of softmax, multiplying by the positive denominator, and by applying the logarithm and the resulting inequality. The derivation is listed in the appendix.

Encoding tan^{-1} *with error bounds.* The handling of non-linearity in $\mathsf{tan}^{-1}(\mathsf{im})$ is based on results in digital signal processing for piece-wise approximating $\mathsf{tan}^{-1}(\mathsf{im})$ with quadratic constraints and error bounds. In case $-1 \leq \mathsf{im} \leq 1$ the quadratic approximation methods (Eq. (7) of [20]) are used, and $\mathsf{tan}^{-1}(\mathsf{im})$ is approximated by $\frac{\pi}{4}\mathsf{im} + 0.273\,\mathsf{im}(1 - |\mathsf{im}|)$ with a maximum error smaller than 0.0038. The absolute value $|\mathsf{im}|$ in the formula is removed by encoding case splits between $\mathsf{im} \geq 0$ and $\mathsf{im} < 0$ using big-M methods. Otherwise, when considering the case $\mathsf{im} > 1$ or $\mathsf{im} < -1$, the symmetry condition of tan^{-1} [22] states that (1) if $\mathsf{im} > 0$ then $\mathsf{tan}^{-1}(\mathsf{im}) + \mathsf{tan}^{-1}(\frac{1}{\mathsf{im}}) = \frac{\pi}{2}$, and (2) if $\mathsf{im} < 0$ then $\mathsf{tan}^{-1}(\mathsf{im}) + \mathsf{tan}^{-1}(\frac{1}{\mathsf{im}}) = -\frac{\pi}{2}$. This implies that we can create a variable im_{inv} with a constraint that $\mathsf{im}_{inv}\,\mathsf{im} = 1$, i.e., variable im_{inv} is the inverse of im. By utilizing the fact that $-1 \leq \mathsf{im}_{inv} \leq 1$, the value of $\mathsf{tan}^{-1}(\mathsf{im}_{inv})$ can be computed by the formula in (i).

Moreover, case splits are encoded using the big-M method as outlined above. Since quadratic terms are used, our approach for handling tan^{-1} nodes requires solving *mixed integer quadratic constraint problem* (MIQCP) problems.

Using these approximations for $\mathsf{tan}^{-1}(\mathsf{im}_i)$, we obtain lower and upper bounds for the value of the node variable x_i, where the interval between lower and upper bound is determined by the approximation error of tan^{-1}. Since the approximation error propagates through the network and using lower and upper bounds instead of an equality constraint relaxes the problem, our method computes approximations for the measure when it is used for ANNs with tan^{-1} as activation function.

Pre-processing based on dataflow analysis. We use interval arithmetic to obtain relatively small values for big-M, in order to avoid a weak LP-relaxation of the MIP. Interval bounds for the values of $x_i^{(l)}$ are denoted by $[\mathsf{Lo}(x_i^{(l)}), \mathsf{Up}(x_i^{(l)})]$. We are assuming that all input values (at layer $l = 0$) are bounded, and the output of bias nodes is restricted by the singleton $[1, 1]$ (the value of the bias is given by the weight of a bias node). Interval bounds for the values of node outputs

$x_i^{(l)}$ are obtained from the interval bounds of connected nodes from the previous layers by means of interval arithmetic.

The output $x_i^{(l)}$ of ReLU nodes is defined by $\mathsf{im}_i^{(l)} = \sum_{j=0,\ldots,d^{(l-1)}} w_{ji}^{(l)} x_j^{(l-1)}$ and the ReLU function $\mathsf{max}(0, \mathsf{im}_i^{(l)})$. Therefore, interval bounds for $x_i^{(l)}$ are computed by first considering the interval bounds $\mathsf{Lo}(\mathsf{im}_i^{(l)})$ and $\mathsf{Up}(\mathsf{im}_i^{(l)})$, which are determined by weights of the linear sum and the bounds on $x_j^{(l-1)}$. The bounds $\mathsf{Lo}(\mathsf{im}_i^{(l)})$ and $\mathsf{Up}(\mathsf{im}_i^{(l)})$ are obtained from interval arithmetic as follows:

$$\mathsf{Lo}(\mathsf{im}_i^{(l)}) = \sum_{j=0,\ldots,d^{(l-1)}} \mathsf{min}\left(w_{ij}^{(l)} \cdot \mathsf{Lo}(x_j^{(l-1)}), w_{ij}^{(l)} \cdot \mathsf{Up}(x_j^{(l-1)})\right)$$

$$\mathsf{Up}(\mathsf{im}_i^{(l)}) = \sum_{j=0,\ldots,d^{(l-1)}} \mathsf{max}\left(w_{ij}^{(l)} \cdot \mathsf{Lo}(x_j^{(l-1)}), w_{ij}^{(l)} \cdot \mathsf{Up}(x_j^{(l-1)})\right).$$

Given $\mathsf{Lo}(\mathsf{im}_i^{(l)})$ and $\mathsf{Up}(\mathsf{im}_i^{(l)})$ the bounds on $x_j^{(l)}$ are derived using the definition of ReLU, i.e.,

$$[\mathsf{Lo}(x_i^{(l)}), \mathsf{Up}(x_i^{(l)})] = [\mathsf{max}(0, \mathsf{Lo}(\mathsf{im}_i^{(l)})), \mathsf{max}(0, \mathsf{Up}(\mathsf{im}_i^{(l)}))].$$

Note that if $\mathsf{Lo}(x_i^{(l)}) \geq 0$ or $\mathsf{Up}(x_i^{(l)}) \leq 0$ these bounds suffice to determine which case of the piece-wise linear ReLU function applies. In this way, the constraints (2)–(4) maybe dropped and the value of $x_i^{(l)}$ is directly encoded using linear constraints, which reduces the number of binary variables. See Fig. 4 for an example of dataflow analysis.

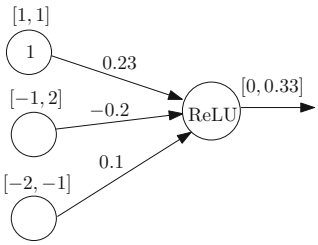

Fig. 4. Dataflow analysis for bounding computed values in a neural network.

In the case of max-pooling nodes, the output $x_i^{(l)}$ is simply the maximum $\mathsf{max}(x_{j_1}^{(l-1)}, x_{j_2}^{(l-1)}, x_{j_3}^{(l-1)}, x_{j_4}^{(l-1)})$ of its four inputs. Therefore, the bounds $\mathsf{Lo}_{x_i^{(l)}}$ and $\mathsf{Up}_{x_i^{(l)}}$ on the output are given by the maximum of the lower and uppers bounds of the four inputs respectively. Interval bounds of the outputs for tan^{-1} are obtained using a polynomial approximation for tan^{-1}. Finally, the output of softmax nodes is a probability in $[0, 1]$ which might also be further refined using interval arithmetic. These bounds on softmax nodes, however, are not used in our encodings, because of the property-driven encoding of softmax output layers as described previously.

4 Perturbation Bounds

We define concrete measures for quantifying the resilience of multi-classification neural networks with softmax output neurons. This measure for resilience is defined over all possible inputs of the network. In particular, our developments

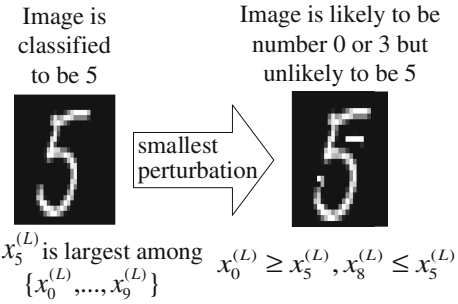

Fig. 5. Finding the smallest possible perturbation for a multi-class classifier to loose confidence.

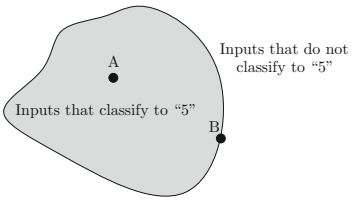

Fig. 6. Two images A, B that both classify to number 5.

do not depend on probability distributions of training and test data as in previous work [3]. Maximum resilience of these ANNs is obtained by means of solving corresponding MIP problems (or MIQCPs in the case of $\tan^{-1}$ activation functions).

We illustrate the underlying principles of maximum resilience using examples from the MNIST database [13] for digit recognition of input images (see Fig. 5). Input images in MNIST are of dimension 24×24 and are represented as a vector $a_1, \ldots, a_{576}$. Input layers of ANN-based multi-digit classifiers for MNIST therefore consist of 576 input neurons, and the output layer is comprised of 10 softmax neurons. Let the output $x_0^{(L)}, \ldots, x_9^{(L)}$ at the last layer be the computed probabilities for an input image to be classified to characters '0' to '9'.

To formally define a perturbation, we allow each input a_i $(i = 1, \ldots, d)$ to have a small disturbance ϵ_i, so the input after perturbation is $(a_1+\epsilon_1, \ldots, a_d+\epsilon_d)$. We sometimes use the concise notation of $a + \epsilon := (a_1 + \epsilon_1, \ldots, a_d + \epsilon_d)$ for the perturbed input. The global value of the perturbation is obtained by taking the sum of the absolute values of each disturbance ϵ_i, i.e., $|\epsilon_1| + |\epsilon_2| + \ldots + |\epsilon_d|$.

Definition 1 (Maximum Perturbation Bound for m-th classifier). *For a given ANN with $d^{(L)}$ neurons in a* softmax *output layer and given constants $\alpha \geq 1$ and $k \in \{1, \ldots, d^{(L)} - 1\}$, we define the* maximum perturbation bound for the m-th classifier*, denoted by Φ_m,*[2] *to be the maximum value such that:*

> *For all inputs $a = (a_1, \ldots, a_d)$ where $x_m^{(L)}(a) \geq \alpha \cdot x_j^{(L)}(a)$ on all other classes $j \in \{1, \ldots, d^{(L)}\} \setminus \{m\}$, we have that for all perturbations $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_d)$ where $|\epsilon_1|+|\epsilon_2|+\ldots+|\epsilon_d| < \Phi_m$, there exist at most $k-1$ classes $j' \in \{0, 1 \ldots, d^{(L)}\}$ such that $x_m^{(L)}(a+\epsilon) \leq x_{j'}^{(L)}(a+\epsilon)$.*

Intuitively, the bound Φ_m guarantees that for all inputs that strongly (defined by α) classify to class m, if the total amount of perturbation is limited to a value

[2] For clarity, we usually omit the dependency of Φ_m from α.

strictly below Φ_m, then either (1) the perturbed input can still be classified as m, or (2) the probability of classifying to m is among the k highest probabilities. Dually, Φ_m is the smallest value such that there exists an input that originally classifies to m, for which the computed probability for class m may *not be among the k highest* after being perturbed with value greater than or equal to Φ_m. Figure 5 illustrates an example of an MNIST image being perturbed, where the neural network considers the perturbed image to be '0' or '3' with at least the probability of being a '5'. The "not among the k highest" property is an indicator that the confidence of classifying to class m has decreased under perturbation, as the perturbed input can be interpreted as at least k other classes. In our experiment evaluations below we used the fixed value $k = 2$.

Constant $\alpha \geq 1$ may be interpreted as indicating the level of confidence of being classified to a class m. When setting α to 1, the analysis takes all inputs for which the probability of class m is greater than or equal to the probabilities of the other classes. Since there might exist an image that has the same probability for all classes, setting $\alpha = 1$ may result in a maximum perturbation of zero. Increasing k helps to avoid this effect, because it requires that at most $k - 1$ other classes have probabilities greater than or equal to the probility of m. By picking an $\alpha > 1$ low-confidence inputs are removed and part (II) of Definition 1 forces the perturbation to be greater than zero. E.g., assume if point B in Fig. 6 is classified to '5' with probability 0.35 and to '0' with probability 0.34, then even by setting $\alpha = 1.1$, point B will not be considered in the analysis. By setting α to 25 one already only considers inputs that classifies to m with probability higher than 0.95.

Provided that Φ_m can be computed for each class m (as shown below), one defines a measure for safe perturbation by taking the minimum of all Φ_m, and the measure is computed by computing each Φ_m independently.

Definition 2 (Perturbation Bound for ANN). *For an ANN with L layers and $d^{(L)}$ softmax neurons in the output layer, a given $\alpha \geq 1$, $k \in \{1, \ldots, d^{(L)} - 1\}$, and Φ_m the perturbation bound for the m-th classifier of this ANN from Definition 1, the* perturbation bound for ANN *is defined as $\Xi := \mathit{min}(\Phi_1, \ldots, \Phi_{d^L})$.*

Based on the dual interpretation above of Definition 1 we are now ready to encode the problem of finding Φ_m in terms of the following optimization problem, where $a = (a_1, \ldots, a_d)$ and $a + \epsilon = (a_1 + \epsilon_1, \ldots, a_d + \epsilon_d)$.

$$\begin{aligned} &\text{minimize} \sum_{i=1,\ldots,d} |\epsilon_i| \\ &\text{subject to} \\ &\qquad x_m^{(L)}(a) \geq \alpha x_i^{(L)}(a) \qquad\qquad \forall i \in \{1, \ldots, d^L\} \setminus m \\ &\bigvee_{\substack{I \subseteq \{1,\ldots,d^L\} \setminus m \\ |I| = k}} \bigwedge_{\forall i \in I} x_m^{(L)}(a + \epsilon) \leq x_i^{(L)}(a + \epsilon) \end{aligned} \tag{5}$$

and subject to constraints (1)–(4) for ANN encoding.

Proposition 3. *For a given $\alpha \geq 1$ and $k \in \{1, \ldots, d^{(L)} - 1\}$, the optimal value of the optimization problem (5) as stated above equals Φ_m. For ANNs using* tan^{-1} *problem (5) yields an under-approximation $\Phi'_m \leq \Phi_m$, because the feasible region is relaxed due to the approximation of* tan^{-1}*.*

The first set of conjunctive constraints specifies that the input $a = (a_1, \ldots, a_d)$ strongly classifies to m (i.e., satisfies condition I in Definition 1), while the second set of disjunctive constraints specifies that by feeding the image after perturbation, the neural network outputs that at least k classes in I are more likely (or equally likely) than class m (i.e., the second condition in Definition 1 is violated). Therefore, for input $a = (a_1, \ldots, a_d)$ and its associated perturbation $\epsilon = (\epsilon_1, \ldots, \epsilon_d)$, we have that $\sum_{i=1,\ldots,d} |\epsilon_i| \geq \Phi_m$. By computing the minimum objective of $\sum_{i=1,\ldots,d} |\epsilon_i|$ satisfying the constraints we obtain $\sum_{i=1,\ldots,d} |\epsilon_i| = \Phi_m$.

We now address the following issues in order to transform optimization problem (5) into a MIP: (1) the objective is not linear due to the introduction of the absolute value function, (2) the non-linearity of $\mathsf{softmax}$ due to the function $x_i^{(L)} = \mathsf{e}^{x_i^{(L-1)}} / \sum_{j=1,\ldots,d^L} \mathsf{e}^{x_j^{(L-1)}}$, and (3) the disjunction in the second set of constraints.

(i) Transforming objectives. Since the objective $|\epsilon_1| + |\epsilon_2| \ldots, |\epsilon_d|$ in problem (5) is not linear, we create new variables $\epsilon_i^{\mathsf{abs}}$ in optimization problem (6), where $i \in \{1, \ldots, d\}$, such that every $\epsilon_i^{\mathsf{abs}}$ is greater than ϵ_i and $-\epsilon_i$. Whenever the value is minimized, we have that $\epsilon_i^{\mathsf{abs}} = |\epsilon_i|$.

(ii) Removing $\mathsf{softmax}$ *output layer.* Optimization problem (5) contains the inequality $x_m^{(L)}(a_1, \ldots, a_d) \geq \alpha x_i^{(L)}(a_1, \ldots, a_d)$. It follows from Proposition 2 that replacing this inequality with $x_m^{(L-1)}(a_1, \ldots, a_d) \geq \ln(\alpha) + x_i^{(L-1)}(a_1, \ldots, a_d)$ is sufficient, thereby omitting the exponential function.

(iii) Transforming disjunctive constraints. The disjunctive constraint in problem (5) guarantees at least k classifications with probability equal or higher as m. We rewrite it by introducing a binary variable c_i for each class $i \neq m$. Then we use (1) an integer constraint $\sum_{i=1,\ldots,d, i \neq m} c_i \geq k$ to select k classifications and (2) the big-M method to enforce that if classification i is selected (i.e., $c_i = 1$), the probability of classifying to i is higher or equal to the probability of classifying to m.

By applying the transformations (i)–(iii) to the optimization problem (5) we obtain problem (6), which is a MIP, and it follows from Proposition 3 that maximum perturbations bounds can be obtained by solving the MIP in (6).

Theorem 1. *For a given $\alpha \geq 1$ and $k \in \{1, \ldots, d^{(L)} - 1\}$, the optimum of the MIP in (6) equals Φ_m for ANNs with ReLU nodes and* $\mathsf{softmax}$ *output layer. For ANNs using* tan^{-1} *it yields an under-approximation.*

$$
\begin{aligned}
&\text{minimize} && \Phi_m := \textstyle\sum_{i\in\{1,\dots,d\}} \epsilon_i^{\mathsf{abs}} && \\
&\text{subject to} && && \\
& && x_m^{(L-1)}(a) \geq \ln(\alpha) + x_i^{(L-1)}(a) && \forall i \in \{1,\dots,d^L\} \setminus m \\
& && \sum_{i\in\{1,\dots,d^L\}\setminus m} c_i \geq k && \\
& && x_i^{(L-1)}(a+\epsilon) \geq x_m^{(L-1)}(a+\epsilon) - M(1-c_i) && \forall i \in \{1,\dots,d^L\} \setminus m \\
& && \epsilon_i^{\mathsf{abs}} \geq \epsilon_i && \forall i \in \{1,\dots,d\} \\
& && \epsilon_i^{\mathsf{abs}} \geq -\epsilon_i && \forall i \in \{1,\dots,d\} \\
& && c_i \in \{0,1\} && \forall i \in \{1,\dots,d^L\} \setminus m
\end{aligned}
$$

and subject to constraints (1)–(4) for ANN encoding. (6)

5 Heuristic Problem Encodings

We list some simple but essential heuristics for efficiently solving MIP problems for the verification of ANNs. Notice that these heuristics are not restricted to computing the resilience of ANNs, and may well be applicable for other verification tasks involving ANNs.

1. Smaller big-M s by looking back at multiple layers. The dataflow analysis in Sect. 3 essentially views neurons at the same layer to be independent. Here we propose a more fine-grained analysis by considering a fixed number of predecessor layers at once. Finding the bound for the output of a neuron $x_i^{(l)}$, for example, can be understood as solving a substantially smaller MIP problem by considering neurons from layer $l-1$ and $l-2$ when considering two preceding layers. These MIP problems are independent for each node in these layers and can therefore be solved in parallel. For each node, we first set the upper bound as a variable to be maximized in the objective, and trigger the MIP-solver to find such a value. Relations over integer binary variables can be derived by applying similar techniques. Notice that these MIPs only generate correct lower and upper bounds if they can be solved to optimality.

2. Branching priorities. This encoding heuristics uses the given structure of feed-forward ANNs in that binary integer variables originating from lower layers are prioritized for branching. Intuitively, variables from the first hidden layer only depend on the input and it influences all other binary integer variables corresponding to neurons in deeper layers.

3. Constraint generation from samples and solver initialization. For computing Φ_m on complex systems via MIP, we use the following three-step process. First, find an input assignment $(a_1^{\mathsf{ini}}, \ldots, a_d^{\mathsf{ini}})$ such that the probability of classifying to m is α times larger, i.e., $x_m^{(L)}(a_1^{\mathsf{ini}}, \ldots, a_d^{\mathsf{ini}}) \geq \alpha x_j^{(L)}(a_1^{\mathsf{ini}}, \ldots, a_d^{\mathsf{ini}})$ for all $j = 1, \ldots, d^{(L)}, j \neq m$. Finding $(a_1^{\mathsf{ini}}, \ldots, a_d^{\mathsf{ini}})$ is equivalent to solving a substantially simpler MIP problem without introducing variables $\epsilon_1, \ldots, \epsilon_d$ and $\epsilon_1^{\mathsf{abs}}, \ldots, \epsilon_d^{\mathsf{abs}}$. Second, use Eq. (6) to compute the minimum perturbation by considering the domain to be size 1, i.e., $\{(a_1^{\mathsf{ini}}, \ldots, a_d^{\mathsf{ini}})\}$. As the domain is restricted to a single input, all variables $a_1^{\mathsf{ini}}, \ldots, a_d^{\mathsf{ini}}$ in Eq. (6) are replaced by constants $a_1^{\mathsf{ini}}, \ldots, a_d^{\mathsf{ini}}$. This also yields substantially simpler MIP problems, and the computed bound is denoted by Φ_m^{ini}. Third, and finally, initialize the MIP-solver by using the computed values from steps 1 and 2, such that the search directly starts with a feasible solution with objective Φ_m^{ini}. Also, the constraint $-\Phi_m^{\mathsf{ini}} \leq \sum_{i=1,\ldots,d} \epsilon_i \leq \Phi_m^{\mathsf{ini}}$, as $\sum_{i=1,\ldots,d} \epsilon_i \leq \sum_{i=1,\ldots,d} |\epsilon_i| = \Phi_m \leq \Phi_m^{\mathsf{ini}}$, can be further added to restrict the search space.

6 Implementation and Evaluation

We implemented an experimental platform in C++ for verifying and computing perturbation bounds for neural networks, which is based on IBM CPLEX Optimization Studio 12.7 (academic version) for MIP solving. We used three different benchmark sets as the basis for our evaluations: (1) MNIST[3] for number characterization, (2) agent games[4], and (3) deeptraffic for simulating highway overtaking scenarios[5]. These benchmarks are denoted by $\mathrm{I}_{\mathrm{MNIST}}$, $\mathrm{I}_{\mathrm{Agent}}$, and $\mathrm{I}_{\mathrm{deeptraffic}}$ respectively, in the following. For each of the benchmarks we created neural networks with different numbers of hidden layers and numbers of neurons, which are shown in Tables 1 and 2. All the networks were trained using ConvNetJS [10].

- Agents in agent games have 9 sensors, each pointing into a different direction and returning the distances to an apple, poison or a wall, which amounts to the 27 inputs. Neural networks of various size were trained for an agent that gets rewarded for eating red things (apples) and gets negative reward when it eats green things (poison).
- deeptraffic is used as a gamified simulation environment for highway traffic. The controller is trained based on a grid sensor map, and it outputs high-level driving decisions to be taken such as switch lane, accelerate or decelerate.
- For MNIST digit recognition [13] has 576 input nodes for the pixels of a gray-scale image, where we trained three networks with different numbers of neurons in the hidden layers.

[3] http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html.
[4] http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html.
[5] http://selfdrivingcars.mit.edu/deeptrafficjs/.

Table 1. Execution time for verifying perturbation problem over a single input instance. Time out (`t.o.`) is set to be 1 h. Agent games turn out to be quite simple to solve, therefore no heuristics are being applied (`n.a.`).

ID	Instance & output m	# inputs; # neurons in hidden layers	δ	Status	Time(s) $M = 10^4$	Time(s) dataflow	Time(s) heuristic 1.+2.
0	I_{Agent} m=0	27; 300	0.025	`inf`	1.9	0.1	`n.a.`
			0.05	`feas`	7.2	26.9	`n.a.`
1	$I^{2\times50}_{\text{MNIST}}$ m=0	576; 100	0.075	`inf`	477.8	186.8	35.1
2			0.1	`inf`	`t.o.`	`t.o.`	2015.9
3	$I^{2\times50}_{\text{MNIST}}$ m=1	576; 100	0.025	`inf`	516.8	763.9	40.5
4			0.05	`feas`	0.5	0.3	328.3
5	$I^{2\times50}_{\text{MNIST}}$ m=3	576; 100	0.025	`inf`	0.3	0.3	18.7
6			0.05	`inf`	303.9	405.1	68.9
7			0.075	`feas`	0.3	0.4	151.6
8	$I^{2\times50}_{\text{MNIST}}$ m=8	576; 100	0.025	`inf`	0.3	0.3	16.5
9			0.05	`inf`	146.0	193.5	37.2
10			0.075	`feas`	1.1	1.2	185.3
11	$I^{4\times50}_{\text{MNIST}}$ m=0	576; 200	0.025	`inf`	464.7	489.4	38.08
12			0.05	`inf`	`t.o.`	`t.o.`	65.5
13	$I^{4\times50}_{\text{MNIST}}$ m=1	576; 200	0.025	`inf`	`t.o.`	`t.o.`	128.21
14			0.05	`feas`	`t.o.`	261.4	3197.6
15	$I^{4\times50}_{\text{MNIST}}$ m=2	576; 200	0.025	`inf`	`t.o.`	`t.o.`	54.32
16			0.05	`unkown`	`t.o.`	`t.o.`	`t.o.`
17	$I^{4\times50}_{\text{MNIST}}$ m=3	576; 200	0.025	`feas`	2.7	2.7	45.88
18			0.05	`feas`	12.5	18.8712	115.1
19	$I^{4\times50}_{\text{MNIST}}$ m=4	576; 200	0.025	`inf`	`t.o.`	`t.o.`	66.43
20			0.05	`unkown`	`t.o.`	`t.o.`	`t.o.`

In our experimental validation we focus on efficiency gains of our MIP encodings and parallelization for verifying neural networks, and the computation of perturbation bound by means of the optimization problem stated in Eq. (6).

Evaluation of MIP Encodings. To understand how dataflow analysis and our heuristic encodings reduce the overall execution time, we have created synthetic benchmarks where for each example, we only ask for a given input instance (e.g., an image) that classifies to m, whether the perturbation bound is below δ. By restricting ourselves to only verify a single input instance and by not minimizing δ, the problem under verification (*local robustness* related to an input) is substantially simpler and is similar to those stated in [3,11]. Table 1 gives a summary over results being evaluated using Google Computing Engine (16 CPU and 60 GB RAM) by only allowing 12 threads to be used. Compared to a naïve approach that sets $M_i^{(l)}$ uniformly to a large constant, applying dataflow analysis can bring benefits for instances that take a longer time to solve. The first

two heuristics we have implemented are useful for solving some very difficult problems. Admittedly, it can also result in longer solutions times for simpler instances, but as our ultimate goal is for scalability such an issue is currently minor. More difficult instances (see $I^{4\times 50}_{\text{MNIST}}$ in Table 1) could only be solved using heuristic 1. for preprocessing.

Effects of Parallelization. For I_{MNIST} we further measured the solution time for local robustness with $\epsilon = 0.01$ for 10 test inputs using 8, 16, 24, 32 and 64 threads on machines that have at least as many CPUs as we allow CPLEX to have threads. The results are shown in Fig. 7. It is clearly visible that using more threads can bring a significant speed-up till 32 cores, especially for instances that cannot be solved fast with few threads. Interestingly, one can also observe that for this particular problem (200 neurons in hidden layers), increasing the number of threads from 32 to 64 does not improve performance (many lines just flatten from 32 cores onwards). However, for some other problems (e.g., 400 neurons in hidden layers in hidden layers or computing resilience), the parallelization effect can last longer to some larger number of threads. We suspect that for problems that have reached a certain level of simplicity, adding additional parallelization may not further help.

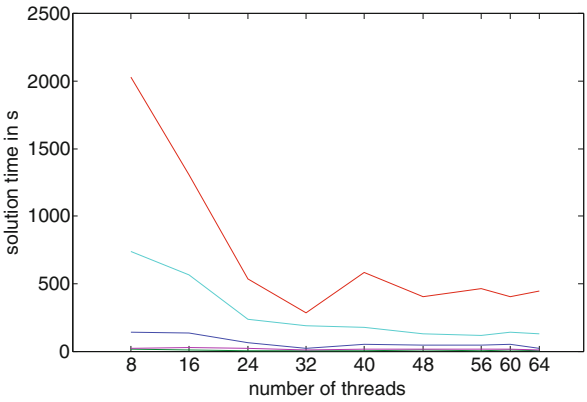

Fig. 7. Execution time vs. the number of threads of five test inputs for I_{MNIST} with $\epsilon = 0.01$.

ComputingΦ_m by solving problem (6). Table 2 shows the result of computing precise Φ_m. For simpler problems, we can observe from the first 4 rows of Table 2 that the computed Φ_m increases, when the value of the parameter α increases. This is a natural consequence - for inputs being classified with higher confidence, it should allow for more perturbation to bring to ambiguity. Notably, using a value of α above its maximum makes the problem infeasible, because there does not exist an input for which the neural network has such high confidence. For complex problems, by setting α is closer to its maximum (which can be computed

Table 2. Computation time and results for computing the maximum resilience Φ_m.

Net: # input; # neurons in hidden layers, output m	α	# of parallelization	Φ_m	Time (s)
I_{RL}: 27;15 $m := 0$	1.1	12	0.1537	0.4
	1.2	12	0.3006	0.3
	1.5	12	0.7666	0.1
	1.7	12	1.2730	0.1
I_{RL} 27;15 $m := 3$	1.3	12	0.6904	1.5
$I_{deeptraffic}$: 30;70 $m := 0$	4.022	360	11.3475	421.58
$I_{deeptraffic}$: 30;70 $m := 3$	78.0305	360	69.9109	86.40
$I_{deeptraffic}$: 45;70 $m := 2$	13.5258	360	7.6226	124.46
$I_{deeptraffic}$: 60;70 $m := 2$	2.2704	360	0.8089	2246.8

by solving another substantially simpler MIP that maximizes α for all inputs that classify to class m), one shrinks the complete input space to inputs with high confidence. Currently, scalability of our approach relies on sometimes setting a high value of α, as can be observed in the lower part of Table 2.

7 Concluding Remarks

Our definition and computation of maximum perturbation bounds for ANNs using MIP-based optimization is novel. By developing specialized encoding heuristics and using parallelization we demonstrate the scalability and possible applicability of our verification approach for neural networks in real-world applications. Our verification techniques also allow to formally and quantitatively compare the resilience of different neural networks. Also, perturbation bounds provide a formal assume-guarantee interface for decoupling the design of sensor sets from the design of the neural network itself. In our case, the network assumes a maximum sensor input error for resilience, and the input sensor sets need to be designed to guarantee the given error bound. These kinds of contract-based interfaces may form the basis for constructing more modularized safety cases for autonomous systems.

Nevertheless, we consider the developments in this paper as only a first tiny step towards realizing the full potential of formal verification techniques for artificial neural networks and their deployment for realizing new safety-critical functionalities such as self-driving cars. For simplicity we have restricted ourselves to 1-norms for measuring perturbations but other vector norms may, of course, also be used depending on the specific needs of the application context. Also, the development of specialized MIP solving strategies for verifying ANNs, which go beyond the encoding heuristics provided in this paper, may result in considerable efficiency gains. Notice also that the offline verification

approach as presented here is applied *a posteriori* to fixed and "fully trained" networks, whereas real-world networks are usually trained and improved in the field and during operation. Furthermore, the exact relationship of our perturbation bounds with the common phenomena of over-fitting in a neural network classifier deserves a closer examination, since perturbation may also be viewed as generalization from samples. And, finally, investigation of further measures of the resilience of ANNs is needed, as perturbation bounds do not generally cover the resilience of ANNs to input transformations such as scaling or rotation.

Appendix

Proposition 1. $x_i^{(l)} = \mathit{max}(0, \mathit{im}_i^{(l)})$ *iff constraints* (2a) *to* (4b) *hold.*

First we establish a lemma to assist the proof.

Lemma 1. $b_i^{(l)} = 1 \Leftrightarrow \mathit{im}_i^{(l)} \geq 0$.

Proof. ($\Rightarrow$) Assume $b_i^{(l)} = 1$, then (3a) holds trivially and (3b) implies $\mathsf{im}_i^{(l)} \geq 0$. ($\Leftarrow$) Assume $\mathsf{im}_i^{(l)} \geq 0$, then (3b) holds trivially and (3a) only holds if $b_i^{(l)} = 1$.

Proof (Proposition 1).
First we rewrite the condition $x_i^{(l)} = \mathsf{max}(0, \mathsf{im}_i^{(l)})$ to allow further processing.

$$
\begin{aligned}
& x_i^{(l)} = \mathsf{max}(0, \mathsf{im}_i^{(l)}) \\
\xleftrightarrow{\text{definition of max}} \quad & (\mathsf{im}_i^{(l)} \geq 0 \Rightarrow x_i^{(l)} = \mathsf{im}_i^{(l)}) \wedge (\mathsf{im}_i^{(l)} < 0 \Rightarrow x_i^{(l)} = 0) \\
\xleftrightarrow{\text{Replace } \mathsf{im}_i^{(l)} \text{ by } b_i^{(l)} = 1 \text{ using lemma 1}} \quad & (b_i^{(l)} = 1 \Rightarrow x_i^{(l)} = \mathsf{im}_i^{(l)}) \wedge (b_i^{(l)} = 0 \Rightarrow x_i^{(l)} = 0)
\end{aligned}
$$

($\Rightarrow$) If $(b_i^{(l)} = 1 \Rightarrow x_i^{(l)} = \mathsf{im}_i^{(l)}) \wedge (b_i^{(l)} = 0 \Rightarrow x_i^{(l)} = 0)$ holds, as $b_i^{(l)}$ is a $0-1$ integer variable, we consider both cases:

(case $b_i^{(l)} = 1$) From the left clause we derive $x_i^{(l)} = \mathsf{im}_i^{(l)}$. From Lemma 1 we have $\mathsf{im}_i^{(l)} \geq 0$. By injecting $b_i^{(l)} = 1$, $x_i^{(l)} = \mathsf{im}_i^{(l)}$, and $\mathsf{im}_i^{(l)} \geq 0$ to constraints (2a) to (4b), all constraints hold due to very large $M_i^{(l)}$.

(case $b_i^{(l)} = 0$) From the right clause we derive $x_i^{(l)} = 0$. From Lemma 1 we have $\mathsf{im}_i^{(l)} < 0$. By injecting $b_i^{(l)} = 0$, $x_i^{(l)} = 0$, and $\mathsf{im}_i^{(l)} < 0$ to constraints (2a) to (4b), all constraints hold due to very large $M_i^{(l)}$.

($\Leftarrow$) If all constraints in (2a) to (4b) hold, we do case split to consider cases $b_i^{(l)} = 0$ and $b_i^{(l)} = 1$, and how they make $(b_i^{(l)} = 1 \Rightarrow x_i^{(l)} = \mathsf{im}_i^{(l)}) \wedge (b_i^{(l)} = 0 \Rightarrow x_i^{(l)} = 0)$ hold.

(**case** $b_i^{(l)} = 1$) From (2b) and (4a) we know that $x_i^{(l)} = \mathsf{im}_i^{(l)}$.
(**case** $b_i^{(l)} = 0$) From (2a) and (4b) we know that $x_i^{(l)} = 0$.

In both cases, $(b_i^{(l)} = 1 \Rightarrow x_i^{(l)} = \mathsf{im}_i^{(l)}) \wedge (b_i^{(l)} = 0 \Rightarrow x_i^{(l)} = 0)$ holds.

Proposition 2. *Given a feed-forward ANN with* softmax *output layer and a constant* $\alpha > 0$*, then for all* $i, j \in \{1, \ldots, d^{(L)}\}$*:*

$$x_{i_1}^{(L)} \geq \alpha\, x_{i_2}^{(L)} \Leftrightarrow x_{i_1}^{(L-1)} \geq \ln(\alpha) + x_{i_2}^{(L-1)}.$$

Proof.

$$\begin{aligned} & x_{i_1}^{(L)} \geq && \alpha\, x_{i_2}^{(L)} \\ \Leftrightarrow \quad & \frac{\mathsf{e}^{x_{i_1}^{(L-1)}}}{\sum_{j=1,\ldots,d^L} \mathsf{e}^{x_j^{(L-1)}}} \geq && \alpha \frac{\mathsf{e}^{x_{i_2}^{(L-1)}}}{\sum_{j=1,\ldots,d^L} \mathsf{e}^{x_j^{(L-1)}}} \\ \Leftrightarrow \quad & x_{i_1}^{(L-1)} \geq && \ln(\alpha) + x_{i_2}^{(L-1)} \end{aligned}$$

References

1. Abu-Mostafa, Y.S., Magdon-Ismail, M., Lin, H.-T.: Learning from Data, vol. 4. AMLBook, New York (2012)
2. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in Ai safety. arXiv preprint arXiv:1606.06565 (2016)
3. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., Criminisi, A.: Measuring neural net robustness with constraints. CoRR, abs/1605.07262 (2016)
4. Bjørner, N., Phan, A.-D., Fleckenstein, L.: νZ-an optimizing SMT solver. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 194–199. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_14
5. Bhattacharyya, S., Cofer, D., Musliner, D., Mueller, J., Engstrom, E.: Certification considerations for adaptive systems. In ICUAS, pp. 270–279. IEEE (2015)
6. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In POPL, pp. 238–252. ACM (1977)
7. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
8. Grossmann, I.E.: Review of nonlinear mixed-integer and disjunctive programming techniques. Optim. Eng. **3**(3), 227–252 (2002)
9. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. CoRR, abs/1610.06940 (2016)
10. Karpathy, A.: ConvNetJS: deep learning in your browser (2014). URL http://cs.stanford.edu/people/karpathy/convnetjs
11. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. CoRR, abs/1702.01135 (2017)
12. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533 (2016)
13. LeCun, Y., Cortes, C., Burges, C.J.: The MNIST database of handwritten digits (1998)

14. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
15. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In CPVR, pp. 427–436 (2015)
16. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against deep learning systems using adversarial examples. arXiv preprint arXiv:1602.02697 (2016)
17. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: Oakland, pp. 582–597. IEEE (2016)
18. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 243–257. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14295-6_24
19. Pulina, L., Tacchella, A.: Challenging SMT solvers to verify neural networks. AI Commun. **25**(2), 117–135 (2012)
20. Rajan, S., Wang, S., Inkol, R., Joyal, A.: Efficient approximations for the arctangent function. IEEE Signal Process. Mag. **23**(3), 108–111 (2006)
21. Scheibler, K., Winterer, L., Wimmer, R., Becker, B.: Towards verification of artificial neural networks. In: MBMV, pp. 30–40 (2015)
22. Ukil, A., Shah, V.H., Deck, B.: Fast computation of arctangent functions for embedded applications: a comparative analysis. In ISIE, pp. 1206–1211. IEEE (2011)
23. Xu, Y., Ralphs, T.K., Ladányi, L., Saltzman, M.J.: Computational experience with a software framework for parallel integer programming. INFORMS J. Comput. **21**(3), 383–397 (2009)