Formal verification of neural networks

Georg Nührenberg

fortiss · Landesforschungsinstitut des Freistaats Bayern
nuehrenberg@fortiss.org

Abstract. In the last few years, neural networks have become the most powerful tool for perception tasks, especially in image processing, and superior performances in these tasks sparked the desire to use them in safety-critical systems, e.g., autonomous vehicles. However, verifying the safety of systems that are using neural networks remains a challenge, because neural networks raise certain dependability concerns (such as adversarial inputs). Resulting from this need, the research topic of formal verification of neural networks emerged. We identify some of the main challenges of this field and discuss how to address them.

1 Introduction

In recent years, neural networks have become the most popular and powerful tool for perception tasks such as image classification [9] and object detection [10, 12]. The superior performance of neural networks in these tasks has led to the desire to use them in safety-critical systems, e.g., autonomous vehicles. However, it remains a challenge to verify the safety of systems that are using neural networks, since traditional methods of safety-engineering do not transfer well to neural networks [3]. Although performing well, neural networks exhibit dependability concerns, the most prominent one being so-called adversarial inputs, which fool neural network classifiers [6]. Resulting from these needs, the research topic of formal verification of neural networks emerged. The first result of using an SMT-solver to verify multi-layer perceptrons with logistic activation functions [11], was followed by a few years of inactivity. Then there was a recent burst of works focussing on neural networks with piecewise-linear activation functions (e.g., ReLU) [7, 8, 5, 2]. We will briefly introduce the neural network verification problem for piecewise-linear activation functions, outline the state-of-the-art and propose a few directions for advancing the topic.

A neural network is a function $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$, where n_0 and n_L denote the dimension of the input and the output space respectively. Let $\boldsymbol{x}^{(0)} \in \mathbb{R}^{n_0}$ and $\boldsymbol{x}^{(L)} \in \mathbb{R}^{n_L}$ denote the input and output of the neural network: $\boldsymbol{x}^{(L)} = f(\boldsymbol{x}^{(0)})$. The neural network is comprised of L layers, where each layer $l \in \{0, \ldots, L\}$ has n_l nodes. Each layer is a function $f_l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$ with input $\boldsymbol{x}^{(l-1)}$ and output $\boldsymbol{x}^{(l)}$ for all $l \in \{1, \ldots, L\}$, except the input layer where $l = 0$. The function of a layer is defined by the layers weights $\boldsymbol{w}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, biases $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ and activation function $\phi^{(l)} : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$.

$$\boldsymbol{x}^{(l)} = \phi^{(l)}(\boldsymbol{w}^{(l)} \cdot \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}) \qquad \forall l \in \{1, \ldots, L\} \tag{1}$$

We consider piecewise-linear activation functions, e.g., $\mathrm{ReLU}(x) = \max(x, 0)$. A single ReLU-node is given by the following equation:

$$\boldsymbol{x}_i^{(l)} = \max(\boldsymbol{w}_{i\cdot}^{(l)} \cdot \boldsymbol{x}^{(l-1)} + \boldsymbol{b}_i^{(l)}, 0) \qquad \forall i \in \{1, \dots, n_l\} \tag{2}$$

2 The neural network verification problem

The neural network verification problem has first been defined in [11]. Summarizing multiple works [8, 5, 2], we give the following definition:

Problem 1 (Neural network verification problem). Given an input set $\mathcal{I} \subseteq \mathbb{R}^{n_0}$ and an output set $\mathcal{O} \subseteq \mathbb{R}^{n_L}$, which are defined by a finite number of linear constraints, and a neural network f the neural network verification problem is to find an input $\boldsymbol{x}^{(0)} \in \mathcal{I}$ such that $f(\boldsymbol{x}^{(0)}) \in \mathcal{O}$ or prove that no such input exists.

The neural network verification problem is NP-hard [8] and the existing methods to encode and solve it are summarized in [1]. The definition of Problem 1 allows the encoding of various properties, where two cases can be distinguished: (i) The neural network has interpretable input variables, such as measurements from single sensors (e.g. ReLUplex case study [8], collision detection benchmark [5]), which allows to encode safety-properties. (ii) Single input variables are not interpretable, which applies to most cases where neural networks are used for image processing (e.g., image classification, object detection) and the inputs are the pixels of the image. For these cases the only known application of formal verification is to verify robustness properties of the neural network, i.e., prove the presence or absence of so-called adversarial examples.

There are four existing versions for solving Problem 1: Encoding it for an SMT-solver [5, 8], encoding for MILP-solver [2], the adapted simplex algorithm ReLUplex [8] and transforming it to an optimization problem and applying branch-and-bound directly [1]. Currently, all methods achieve roughly the same sizes of benchmarks [1]. The methods are based on the fact that (2) is piecewise-linear, and Problem 1 can be written as a conjunction of linear equations and piecewise linear equations:

$$\boldsymbol{x}^{(0)} \in \mathcal{I} \tag{3a}$$

$$\boldsymbol{x}^{(L)} \in \mathcal{O} \tag{3b}$$

$$\boldsymbol{x}_i^{(l)} = \max(\boldsymbol{x}_i^{(l)\prime}, 0) \qquad \forall i \in \{1, \dots, n_l\} \quad \forall l \in \{1, \dots, L\} \tag{3c}$$

$$\boldsymbol{x}_i^{(l)\prime} = \boldsymbol{w}_{i\cdot}^{(l)} \cdot \boldsymbol{x}^{(l-1)} + \boldsymbol{b}_i^{(l)} \qquad \forall i \in \{1, \dots, n_l\} \quad \forall l \in \{1, \dots, L\} \tag{3d}$$

Modeling the ReLU activation functions (3c) for an SMT-solver is straightforward using `if-then-else`, for MILPs it is done by introducing a binary variable encoding the two cases of the piecewise linearity and ReLUplex introduces a special derivation rule to the simplex algorithm for these constraints. A further crucial part exploited by all algorithms are lower and upper bounds on the variables $\boldsymbol{x}^{(l)}$, which can be derived from lower and upper bounds on the input $\boldsymbol{x}^{(0)}$,

denoted by $\boldsymbol{l}^{(0)}, \boldsymbol{u}^{(0)} \in \mathbb{R}^{n_0}$ such that $\boldsymbol{l}^{(0)} \leq \boldsymbol{x}^{(0)} \leq \boldsymbol{u}^{(0)}$. These input bounds are usually given by the application domain of the neural network. It turns out that the quality of the encoding heavily depends on the tightness of these bounds [5, 8, 2].

Even though neural network verification proves to be a hard to solve problem, the hope is to speed-up the verification times by exploiting the structure of the problem, in order to achieve applicability on industrial scale instances. Judging the current methods in terms of potential for performance improvement, SMT and MILP encodings have the drawback that they rely on solvers, which offer limited capabilities to steer the solving process. Thus, we focus on improvement strategies for the direct branch-and-bound technique [1], which is briefly outlined in the next section. However, most of the proposed improvements have potential benefits for all the verification methods, since all of them are exploiting the piecewise linearity and benefit from tight variable bounds.

3 Branch-and-bound verification

In order to apply direct branch-and-bound the verification problem is transformed into an optimization problem. Intuitively speaking, we optimize by "how much" a property can be violated. The property encoded by $\mathcal{I}$ and $\mathcal{O}$ is modeled by appending additional layers to the neural network, where the final network has only one output variable x^L. If the minimum of x^L is less than zero (resp. greater than or equal to zero) the original property is unsatisfiable (resp. satisfiable) [1].

$$\min x^L \tag{4a}$$

$$\text{subject to constraints (3)} \tag{4b}$$

$$\boldsymbol{l}^{(0)} \leq \boldsymbol{x}^{(0)} \leq \boldsymbol{u}^{(0)} \tag{4c}$$

The branch-and-bound method can be used to solve the optimization problem (4) [1]. The branching is carried out over the input domain given by (4c), by splitting this hyper-rectangle into smaller hyper-rectangles to obtain branches. Branch-and-bound is based on computing a lower and upper bound on the objective x_L for each branch. The goal is to prune large parts of the branch-and-bound tree that cannot contain the optimum because their lower bound is greater than the upper bound of another part of the tree. In the next section we will discuss some approaches to improve the bounds, which is likely to lead to more pruning, thus, enhancing the performance of the branch-and-bound algorithm.

4 Improvement strategies for branch-and-bound

4.1 Lower bounds

Lower bounds are computed by solving an LP-relaxation of (4), which is given by replacing constraints (3c) with their convex hull (requiring pre-computed

bounds on the node-variables) [5]. Since the convex hull relaxation is already tight for each node individually, further strengthening the LP-relaxation of the whole problem could be done by exploiting information about the relationship between different nodes. Furthermore, tightening lower and upper bounds on individual variables and increasing the performance of computing these bounds is a major area for improvement, because it directly effects the quality of the LP-relaxation.

Another promising method for computing a lower bound for (4) is based on weak duality for mixed-integer programming [4].

4.2 Upper bounds

The method proposed by Bunel et al. [1] is randomly sampling points from the input domain (4c) of the current node and evaluating the objective value (4a) to obtain an upper bound. (Note that fixing $\boldsymbol{x}^{(0)}$ fixes all other variables, which is equivalent to running the neural network inference.) We propose a simple heuristic, which often yields a better upper bound: Instead of randomly sampling a value for $\boldsymbol{x}^{(0)}$, we re-use a valuation of $\boldsymbol{x}^{(0)}$ that is a result of solving the LP-relaxation of optimization problem (4) for the lower bound computation. Then, as previously, an upper bound is obtained by evaluating the objective (4a) for a fixed $\boldsymbol{x}^{(0)}$. We have experimented this method on the PLANET collision detection benchmark [5] and compared the original algorithm of [1] to our modified version using the minimum of random sampling and our upper bound heuristic. In total there was a speed-up of about 12%, however, the speed-up is mainly achieved on the SAT-instances, which appear to be easier to verify than UNSAT-instances (see Table 1).

Furthermore, an idea to be investigated is to use gradient-descent to improve a given upper bound. Initial experiments indicate that gradient-descent can provide better upper bounds, but it has to be carefully implemented to achieve a performance improvement.

	BaB_orig	Bab_ub_heuristic
SAT-instances	837 s	337 s
UNSAT-instances	3578 s	3394 s
time out	1200 s	1200 s
total verification time	5615 s	4931 s

Table 1. Preliminary evaluation of the simple heuristic for improved upper bounds

4.3 Gradient-guided branching

Even the order, in which the branches are explored in the branch-and-bound algorithm, may influence how much of the search tree can be pruned. We aim to study branching strategies that should reach low objective values early by using heuristics such as analyzing neural network gradients to decide which branch to explore next.

References

[1] Rudy Bunel et al. "A Unified View of Piecewise Linear Neural Network Verification". In: *arXiv preprint arXiv:1711.00455* (2017).

[2] Chih-Hong Cheng, Georg Nührenberg, and Harald Rueß. "Maximum Resilience of Artificial Neural Networks". In: *Automated Technology for Verification and Analysis - 15rd International Symposium , ATVA*. 2017.

[3] C.-H. Cheng et al. "Neural Networks for Safety-Critical Applications - Challenges, Experiments and Perspectives". In: *ArXiv e-prints* (Sept. 2017). arXiv: 1709.00911 [cs.SE].

[4] Krishnamurthy Dvijotham et al. "A Dual Approach to Scalable Verification of Deep Networks". In: *arXiv preprint arXiv:1803.06567* (2018).

[5] Ruediger Ehlers. "Formal verification of piece-wise linear feed-forward neural networks". In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2017, pp. 269–286.

[6] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[7] Xiaowei Huang et al. "Safety Verification of Deep Neural Networks". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. 2017, pp. 3–29. DOI: 10.1007/978-3-319-63387-9_1. URL: https://doi.org/10.1007/978-3-319-63387-9_1.

[8] Guy Katz et al. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. 2017, pp. 97–117. DOI: 10.1007/978-3-319-63387-9_5.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[10] Wei Liu et al. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[11] Luca Pulina and Armando Tacchella. "An abstraction-refinement approach to verification of artificial neural networks". In: *Computer Aided Verification*. Springer. 2010, pp. 243–257.

[12] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 6517–6525.