Automated Verification of Neural Networks: Advances, Challenges and Perspectives

Francesco Leofante[1,4], **Nina Narodytska**[2], **Luca Pulina**[3], **Armando Tacchella**[1]
[1] University of Genoa , [2] VMware Research
[3] University of Sassari, [4] RWTH Aachen University
leofante@cs.rwth-aachen.de, n.narodytska@vmware.com, lpulina@uniss.it , armando.tacchella@unige.it

Abstract

Neural networks are one of the most investigated and widely used techniques in Machine Learning. In spite of their success, they still find limited application in safety- and security-related contexts, wherein assurance about networks' performances must be provided. In the recent past, automated reasoning techniques have been proposed by several researchers to close the gap between neural networks and applications requiring formal guarantees about their behavior. In this work, we propose a primer of such techniques and a comprehensive categorization of existing approaches for the automated verification of neural networks. A discussion about current limitations and directions for future investigation is provided to foster research on this topic at the crossroads of Machine Learning and Automated Reasoning.

1 Introduction

Neural Networks (NNs) are powerful learning models that can achieve impressive results in many applications, such as image classification [Taigman *et al.*, 2014] or speech recognition [Yu *et al.*, 2012], with some architectures even claimed to be matching the cognitive abilities of humans [LeCun *et al.*, 2015]. In spite of some exceptions — see, *e.g.*, [Jorgensen, 1997] and more recently [Bojarski *et al.*, 2016; Julian *et al.*, 2016] — traditional applications of NNs have been mostly confined to systems without safety or security requirements, due to the absence of effective methods to guarantee the correct behavior of such models.

There has long been an interest in the rigorous verification of NNs, with first attempts made in the early 2000s [Zakrzewski, 2001; Pullum *et al.*, 2007], mostly motivated by applications in avionic systems. This line of research was recently refueled by critical discoveries made in [Szegedy *et al.*, 2014; Goodfellow *et al.*, 2015]: machine learning models, including state-of-the-art Deep Neural Networks (DNNs), can be unstable with respect to *adversarial perturbations*. Such perturbations represent minimal changes to correctly classified input data that can cause a network to respond in unexpected and incorrect ways. These discoveries confirmed the worthiness of efforts to develop techniques to provide guarantees about the behavior of NNs and other learning models.

Among potential approaches to ensure correct behavior of NNs, those based on Automated Reasoning show some promise. Since NNs are complex implements, it is unlikely that their performances can be checked and corrected manually. Techniques such as Adversarial Training [Goodfellow *et al.*, 2015] have been proposed with the intent to steer learning in the direction of making resulting networks more robust to adversarial attacks. However, recent results [Carlini and Wagner, 2017] have shown that existing methods still lack thorough evaluations and often they are even unable to detect adversarial examples. On the other hand, automated reasoning tools can be applied to NNs "out of the box" to perform verification of desired properties, *e.g.*, robustness, safety, and equivalence. As with any algorithmic technique, the challenge shifts towards the computational needs of automated verification, and the problem of scaling to networks of relevant size arises.

Starting from the seminal contribution of [Pulina and Tacchella, 2010], verification of NNs is not just a theoretical possibility, but it has witnessed diverse proposals based on a variety of automated reasoning techniques, including Boolean satisfiability (SAT) solvers, Satisfiability Modulo Theories (SMT) solvers and Mixed Integer Programming (MIP) solvers. The contributions to be found in the literature consider verification of diverse models, from conventional NNs, to networks apt for representation learning, *i.e.*, those "*...allowing a machine to be fed with raw data and automatically discover the representations needed for detection or classification*" [LeCun *et al.*, 2015]. Following the common usage found in the literature, we associate the term *deep* to networks apt for representation learning; by contrast, we use the term *shallow* to denote networks designed within a conventional learning framework. As a matter of fact, while all NNs are arranged in layers of elementary computation units, conventional networks are indeed shallow since they rarely consist of several layers beyond input and output ones. From the initial challenges and limitations presented in [Pulina and Tacchella, 2012], mostly related to the application of SMT solvers to prove properties of shallow NNs, several contributions have focused on the challenge of scaling SMT, as well as SAT and MIP techniques to deep networks. In this work, we present a survey of such literature, and we contribute a

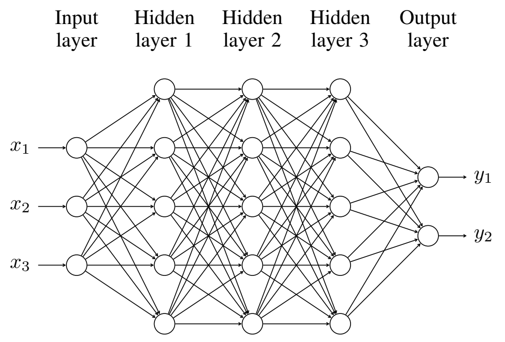


Figure 1: A simple network containing 3 input nodes, 2 output nodes and 3 hidden layers containing 5 nodes each.

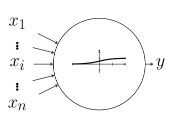
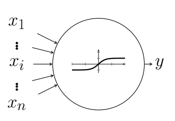
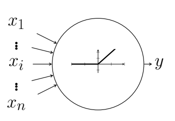
Logistic	**Hyperbolic tangent**	**ReLU**
		
$y = \dfrac{1}{1+e^{-\Sigma_i x_i}}$	$y = tanh\,(\Sigma_i\, x_i)$	$y = max\,(0, \Sigma_i\, x_i)$

Figure 2: Examples of commonly used activation functions, assuming that input weights are fixed to one.

categorization of existing approaches based on properties and networks. To the best of our knowledge, this is the first work attempting to put in perspective this body of work within the communities of Machine Learning and Automated Reasoning. Our contribution supports comparative assessments among applications, but it also helps in identifying open directions for future research. With the categorization herewith proposed, we hope to lay the foundations for further innovation in this interdisciplinary domain.

The remainder of this paper is organized as follows. In Section 2 we briefly introduce basic terms and definitions about NNs; in Section 3 we provide a short introduction to the main automated reasoning techniques that have been considered so far to verify NNs. Section 4 provides a classification of the current relevant literature and Section 5 describes the current challenges and provides some potential directions for future research.

2 Feed-forward Neural Networks

This survey will focus on approaches developed for the verification of feed-forward NNs. In such networks, *neurons* are arranged in disjoint *layers*, with each layer being fully connected with the next one, but without connection between neurons in the same layer. We call a layer without incoming connections *input* layer, a layer without outgoing connections *output* layer, while all other layers are referred to as *hidden* layers. Connections between neurons in the network are labeled with *weights* which can take, in the most general case, real values[1]. Furthermore, each neuron is characterized by an *activation function* defining the input-output relation for that particular neuron. A pictorial representation of the architecture described can be see in Fig. 1.

At a high level, these networks can be seen as functions $\nu : I^n \to O^m$, mapping an n-dimensional *input domain* I^n ($n > 0$) to a m-dimensional *output domain* O^m ($m > 0$). We argue that this representation captures most cases of practical interest. For instance, a network computing an approximation of some function $f : \mathbb{R}^n \to \mathbb{R}$ would have $I = O = \mathbb{R}$, whereas a network classifying 8-bit images of size $h \times v$ in two classes would be defined as $\nu : \{0, \ldots, 255\}^{h \cdot v} \to \{0, 1\}$ with $I = \{0, \ldots, 255\}$ and $O = \{0, 1\}$. The mapping is performed by feeding an input $e \in I^n$ to the network through its input layer, which is then propagated to the output layer by successively computing linear combinations of values from nodes in the preceding layer and applying activation functions to the result. Several types of activations exist, each of them having different properties which determine the expressive power of the network. Most commonly used activation functions include, *e.g.*, logistic sigmoid, hyperbolic tangent and rectified linear units (ReLU) – see Fig. 2.

[1]We do not discuss here how NNs are learned. Frameworks such as `Keras` [Chollet and others, 2015] provide ready-to-use solutions for learning networks from data.

3 Decision procedures

Several approaches have been proposed to verify different classes of networks. Even though such approaches might differ in several aspects, they all tackle the verification problem by encoding networks to constraint systems. In this section we briefly introduce the decision procedures that have been most commonly used to solve constraint systems encoding neural networks. For further details we refer the interested reader to [Biere *et al.*, 2009] and [Schrijver, 1999].

SAT SAT solving aims to check the satisfiability of a propositional logic formula φ represented as Boolean combinations of atomic (Boolean) propositions. Although several algorithms have been proposed to solve the boolean satisfiability problem, here we introduce CDCL-style SAT solving algorithm (see Fig. 3), being the most commonly implemented in state-of-the-art SAT solvers.

The CDCL algorithm proceeds as follows. Starting from an input CNF formula, the algorithm explores the search space by iteratively making decisions, *i.e.*, it assigns truth values to some heuristically chosen propositions. After each such decision, the algorithm applies Boolean Constraint Propagation (BCP) to determine further variable assignments that are implied by the last decision. If BCP leads to a conflict, *i.e.*, if the value of a proposition is implied to be true as well as false at the same time, *conflict-driven clause-learning* and *non-chronological backtracking* are applied: the algorithm follows back the chain of implications and applies resolution to derive a reason for the conflict in form of a conflict clause, which is added to the solver's clause set. Backtracking removes previous decisions and their implications until the conflict clause can be satisfied.

If the input has clauses consisting of a single literal, these literals will be directly assigned. Therefore, the algorithm starts with BCP to detect implications. If BCP leads to a conflict, the algorithm tries to resolve the conflict. If the conflict cannot be resolved, the input formula is unsatisfiable. Otherwise, if the conflict is successfully resolved, the algorithm backtracks and continues with BCP. If BCP can be completed without any conflicts, a new decision is made if there are any unassigned propositions. Otherwise, a satisfying solution is found.

SMT Satisfiability Modulo Theories is the problem of deciding the satisfiability of a first-order formula with respect to some decidable theory $\mathcal{T}$. In particular, SMT generalizes the boolean satisfiability problem (SAT) by adding background theories such as the theory of real numbers, the theory of integers, and the theories of data structures (*e.g.*, lists, arrays and bit vectors).

To decide the satisfiability of an input formula φ in CNF, SMT solvers (see Fig. 4) typically first build a *Boolean abstraction* $abs(\varphi)$ of φ by replacing each constraint by a fresh Boolean variable (proposition), *e.g.*,

$$\begin{array}{rcccccccc} \varphi & : & \underbrace{x \geq y} & \wedge\ (& \underbrace{y > 0} & \vee & \underbrace{x > 0} &)\ \wedge & \underbrace{y \leq 0} \\ abs(\varphi) & : & A & \wedge\ (& B & \vee & C &)\ \wedge & \neg B \end{array}$$

where x and y are real-valued variables, and A, B and C are propositions.
A SAT solver searches for a satisfying assignment S for $abs(\varphi)$, *e.g.*, $S(A) = 1$, $S(B) = 0$, $S(C) = 1$ for the above example. If no such assignment exists then the input formula φ is unsatisfiable. Otherwise, the consistency of the assignment in the underlying theory is checked by a *theory solver*. In our example, we check whether the set $\{x \geq y,\ y \leq 0,\ x > 0\}$ of linear inequalities is feasible, which is the case. If the constraints are consistent then a satisfying solution (*model*) is found for φ. Otherwise, the theory solver returns a theory lemma φ_E giving an *explanation* for the conflict, *e.g.*, the negated conjunction some inconsistent input constraints. The explanation is used to refine the Boolean abstraction $abs(\varphi)$ to $abs(\varphi) \wedge abs(\varphi_E)$. These

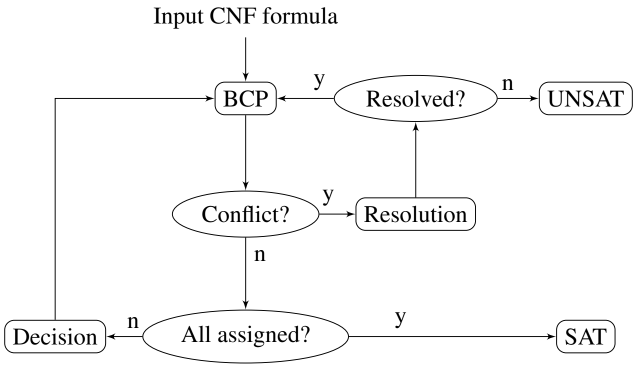


Figure 3: The CDCL framework.

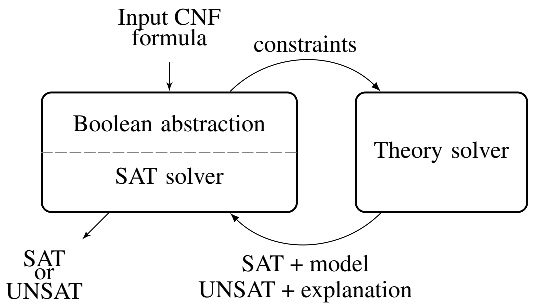


Figure 4: The SMT solving framework.

steps are iteratively executed until either a theory-consistent Boolean assignment is found, or no more Boolean satisfying assignments exist.

MIP Mixed Integer Linear Programming (MIP) solves linear problems over a set of integer and real valued variables. MIP contains a set of *decision variables*, a set of *linear constraints* over these variables and an *objective function* to be optimized (minimized or maximized) that is linear in decision variables. Without loss of generality we consider a minimization formulation of a MIP and assume that all variables are integers that take values in a given interval. Let $x_1, \ldots, x_n$ be a set of decision variables, an integer linear program can be written as

$$\begin{aligned} \min \sum_{i=1} c_i x_i \\ \text{subject to } \sum_{i=1} a_{ji} x_i \geq b_j, j \in [1, m] \\ x_i \in [a_i, b_i] \cap \mathbb{Z}, i \in [1, n] \end{aligned}$$

Values c_i, a_{ij} and b_j are constants that are specified during problem formulation. One general approach to solving MIPs is by using the *branch-and-cut* method that employs *branch-and-bound* and *cutting planes* techniques. The branch-and-bound method performs two main steps. First, it solves a linear relaxation of MIP where all integrality constraints are relaxed. In other words, we assume that all integer variables can take real values. The cost of a solution of the relaxed problem gives a lower bound on the optimal solution of the original problem. However, this solution can contain fractional values. Therefore, the MIP solver has to branch on one of variables with the fractional value, splitting the search space into two parts. For example, if $x_2 = 0.5$ in a solution then the split is $x_2 = 0$ or $x_2 = 1$. Based on these decision points, the search procedure builds a branching tree and stores the best solution found in each node. These solutions are used to prune future branches. The cutting planes technique is used to cut off fractional solutions. These cutting inequalities are learned during search and help to improve the quality of Linear Programming relaxations.

4 State of the art: a bird's eye view

To describe properties of NNs defined as $\nu : I^n \rightarrow O^m$, let $pre(x)$ and $post(y)$ be sorted first order logic formulas, with

Network	Invariance		Invertibility	Equivalence
	Local	Global		
BNN	[Narodytska *et al.*, 2017]	[Cheng *et al.*, 2017b]	[Korneev *et al.*, 2018]	[Narodytska *et al.*, 2017]
DNN(ReLU)	[Gopinath *et al.*, 2017; Katz *et al.*, 2017; Bastani *et al.*, 2016]	[Katz *et al.*, 2017; Dutta *et al.*, 2017; Kuper *et al.*, 2018]	—	—
DNN(ReLU+Pooling)	[Ehlers, 2017; Tjeng and Tedrake, 2017]	[Bunel *et al.*, 2017; Fischetti and Jo, 2017]	[Ehlers, 2017]	—
DNN	[Huang *et al.*, 2017]	—	—	—
NN	[Pulina and Tacchella, 2012; Cheng *et al.*, 2017a]	[Pulina and Tacchella, 2010; Scheibler *et al.*, 2015]	—	—

Table 1: Literature classified by network type (rows) and properties considered (columns). For invariance properties, a further distinction is made between papers dealing with local or global invariance. "BNN" row collects references dealing specifically with Binarized (deep) NNs. Deep network references are organized according to the kind of nodes for which the techniques thereto proposed are applicable: "DNN(ReLU)" for networks made up of ReLUs only, "DNN(ReLU+Pooling)" if also pooling nodes are considered, and "DNN" if no restriction is placed on nodes. "NN "row collects references which, in principle, consider any kind of NN, but are mostly concerned with shallow networks.

x and y occurring as free variables of sort S_I and S_O, respectively. Informally, S_I is the input type and S_O is the output type required by the network thought as a function in some programming language. We say that pre defines *preconditions* on the input of a network, and $post$ defines *postconditions* on its output. We consider interpretations that map sort S_I to the input domain I^n and sort S_O to the output domain O^m. For instance, an interpretation in a network defined as $\nu : \mathbb{R} \to \{0, 1\}$, could map S_I = *real* and S_O = *boolean* to corresponding domains $\mathbb{R}$ and $\{0, 1\}$. Interpretations map variables x and y to values in the domains I^n and O^m. We write $\mathcal{I}(x \to e)$ to denote that variable x is mapped to value $e \in I^n$ by interpretation $\mathcal{I}$, and $\varphi^{\mathcal{I}}$ to denote the value of expression φ under interpretation $\mathcal{I}$. We also consider the predicates "=", "$\neq$", and "$<$" with the usual semantics.

Properties. To the extent of our knowledge, all the studies published so far about automated verification of NNs, focused on three kinds of properties:

- *Invariance*. For specific conditions pre and $post$, asserting an invariance property for a network ν amounts to state
$$\forall x.\forall y.(pre(x) \wedge y = \nu(x)) \implies post(y) \qquad (1)$$
The goal of automated verification is to prove (1) or find a *counterexample*, *i.e.*, some value $e \in I^n$ such that $(pre(x) \wedge \neg post(y))^{\mathcal{I}(x \to e, y \to \nu(e))}$ is true.
- *Invertibility*. For specific conditions pre and $post$, asserting an invertibility property for a network ν amounts to state
$$\forall y.\exists x.(post(y) \wedge y = \nu(x)) \implies pre(x) \qquad (2)$$
Proving invertibility might be less interesting than actually finding a specific *realization*, *i.e.*, given an output pattern $p \in O^m$ find an input pattern $e \in I^n$ such that $(post(y) \wedge y = \nu(x) \wedge pre(x))^{\mathcal{I}(x \to e, y \to p)}$ is true.
- *Equivalence*. While invariance and invertibility refer to a single network, equivalence is a property involving two networks ν and ν'. For specific conditions pre and $post$, it is defined as
$$\begin{aligned}&\forall x.\forall y.\forall w(\,pre(x) \wedge \\ &\qquad y = \nu(x) \wedge post(y) \wedge \\ &\qquad w = \nu'(x) \wedge post(w)) \implies y = w\end{aligned} \qquad (3)$$
The property can be either proved as such or a *counterexample* can be produced, *i.e.*, some $e \in I^n$ such that
$$\begin{aligned}&(pre(x) \wedge \\ &\; post(y) \wedge post(w) \wedge y \neq w)^{\mathcal{I}(x \to e, y \to \nu(e), w \to \nu'(e))}\end{aligned}$$
is true. In contexts wherein strict equality might be inappropriate, we can replace the term $y = w$ in (3) with the term $||y - w|| < \epsilon$, assuming that $|| \cdot ||^{\mathcal{I}}$ is a norm over O^m and $\epsilon^{\mathcal{I}} \in O^m$ is a a *tolerance*, *i.e.*, a threshold under which the response of the networks is considered to be indistinguishable.

As stated in (1) to (3), the scope of the properties is *global*, *i.e.*, interpretations range over full domains I^n and O^m. Researchers have also considered *local* versions wherein interpretations range over specific regions of the input and output domains.

Literature. In Table 1 we have organized all the contributions found in the literature, wherein the techniques described in Section 3 are utilized to prove properties about NNs. For invariance, we have considered an additional classification into global and local versions (first two columns of the Table), whereas for invertibility and equivalence, we did not make a distinction due to the limited number of references available. In particular, [Ehlers, 2017] mentions a local flavor of invertibility considering DNNs with ReLU and MaxPool nodes, whereas [Korneev *et al.*, 2018] considers a global flavor of the same property in the context of binarized (deep) NNs (BNNs). As for equivalence, we found only one contribution about BNNs in [Narodytska *et al.*, 2017].

The first paper to consider automated verification for shallow networks is [Pulina and Tacchella, 2010]. Indeed, some authors refer to [Zakrzewski, 2001] as the first attempt to scrutinize NNs in order to give formal guarantees about their performances. We must stress here that while [Zakrzewski, 2001] put forth the first analytical approach to verify network's accuracy, it did not consider algorithmic verification which, on the contrary, is the focus of [Pulina and Tacchella, 2010] and the ensuing literature herewith considered. In particular, [Pulina and Tacchella, 2010] focused on a global invariance condition for multi-input, single-output networks involving non-linear activation functions, whereby given $\nu : I^n \to O$, as long as the input $e \in I^n$ is guaranteed to range within some prescribed interval, then $\nu(e) \in [a, b]$ with $a, b \in O$. Also for the first time, [Pulina and Tacchella, 2010] deals with verification-triggered network *repair*, *i.e.*, how to modify network's weights in order to meet the invariance condition. The initial contribution is extended in [Pulina and Tacchella, 2012] to consider other invariance conditions, including local invariance and global sensitivity. The main limitation of these early attempts is that the number of activations functions in the networks is relatively small (in the order of tens), and the proposed encoding hardly scales for the kind of (deep) networks considered nowadays in applications. Other attempts at proving invariance properties of shallow NNs include [Scheibler *et al.*, 2015] and [Cheng *et al.*, 2017a]. The former approach leverages SMT technology over non-linear arithmetic — with essentially the same limitations exposed in [Pulina and Tacchella, 2012] — whereas the latter considers MIP techniques.

More recently, due to the growing interest in DNNs, several papers came out proposing approaches that, to some extent, can deal with networks having nodes in the order of thousands, *e.g.*, 1800 ReLU nodes in [Katz *et al.*, 2017], and millions of parameters, *e.g.*, about 1.25 million parameters in the CIFAR experiment considered in [Huang *et al.*, 2017]. Most of the contributions proposed in the literature focus on invariance properties only and consider restrictions of DNNs, *i.e.*, they consider only some kind of activation function. ReLUs, possibly with MaxPool nodes, are by far the most common target: to the best of our knowledge, the only contribution dealing with convolutional, ReLU, maxpooling, and softmax layers is [Huang *et al.*, 2017], whose goal is to verify (local) adversarial robustness using SMT technology black-box. Noticeably, [Tjeng and Tedrake, 2017; Bastani *et al.*, 2016] and [Fischetti and Jo, 2017] are the only contributions using MIP encodings to prove adversarial robustness in a local and global fashion, respectively. Other contributions instead consider variations of SMT technology wherein the theory solver is specialized to deal with the problem at hand. This is the case of [Gopinath *et al.*, 2017; Katz *et al.*, 2017; Kuper *et al.*, 2018] which consider Reluplex, a Simplex-based decision procedure specialized to tackle constraints arising from the verification of deep networks comprised of ReLU nodes. Uniquely among other surveyed works, [Gopinath *et al.*, 2017] proposed a combination between inductive and deductive techniques to improve scaling on large networks: a global invariance check is reduced to a series of local invariance check using clustering techniques on the input space. In [Ehlers, 2017] networks made of ReLUs and MaxPool nodes are considered. Also in this case, an original combination of SAT and a theory solver (an Integer Linear Programming engine) is considered to enable verification of a network containing 1341 nodes.

Overall, looking at Table 1, it is clear that existing literature leaves a lot of potential areas of interest to be covered. Firstly, full-fledged DNNs are considered only in one contribution, namely [Huang *et al.*, 2017], and the approach therewith proposed is still in its prototypical stage. This is in stark contrast with the development of tools for *learning* neural networks, many of which are available off-the-shelf, reaching a substantial degree of sophistication. Secondly, network equivalence and invertibility remain mostly uncharted, but they could support effective application of neural networks in many ways. For instance, showing that a small-footprint network is equivalent to a large-footprint one, memory and energy could be saved by running the smallest network. Also showing that a given output pattern may not be produced by any input pattern, could help debug networks that could otherwise reach production based solely on empirical tests.

5 Challenges and perspectives

The first and foremost challenge in automated verification of NNs, is to coordinate the efforts of two "separated at birth" AI communities: Machine Learning and Automated Reasoning. On one side, Machine Learning has made remarkable progresses in the last decade, and it is now one of the mainstream AI domains, backed by substantial funding and success stories, *e.g.*, Deepmind's AlphaGo[2], Facebook's DeepFace [Taigman *et al.*, 2014], and Neurala's[3] Lifelong-DNN™, to cite only some. The existing need for explanation and certification associated with learning algorithms has been recognized by the community, *e.g.*, by a number of recent workshops dedicated to the topic [4] and DARPA research programs [5]. For NNs, most researchers in Machine Learning agree that new techniques are needed to understand, trust and manage networks that might be used in regulated areas, such as law or medicine, or in safety- and security-critical applications. As noted by Percy Liang during his invited talk at AAAI 2018[6], current means to assess the quality of NNs focus on accuracy only, encouraging behaviors that are good on

[2]https://deepmind.com/blog/alphagos-next-move/.

[3]https://www.neurala.com/press-releases/edge-deep-learning-without-cloud.

[4]Explainable Artificial Intelligence at IJCAI 2017, http://home.earthlink.net/~dwaha/research/meetings/ijcai17-xai/ and Interpretable Machine Learning at NIPS 2017, http://interpretable.ml/

[5]Explainable Artificial Intelligence (XAI)https://www.darpa.mil/program/explainable-artificial-intelligence

[6]"How Should We Evaluate Machine Learning for AI?", https://aaai.org/Conferences/AAAI-18/invited-speakers/

average. This leaves out a number of important properties and creates vulnerabilities in NNs which can be exploited, *e.g.*, in cyber-attacks to AI systems. On the other side, Automated Reasoning has made, possibly less spectacular, but steady progresses as well. Automated reasoning tools provide staple techniques for hardware, software and protocol verification in research as well as industrial applications — see, *e.g.*, [Harrison, 2009]. Researchers expanded the reach beyond the traditional "comfort zone" of discrete finite-state models to deal with cyber-physical systems incorporating continuous and stochastic dynamics — see, *e.g.*, [Clarke and Zuliani, 2011; Kumar *et al.*, 2012]. On top of this, the community has been actively providing interesting contributions towards solving the problem of NN verification as described in Section 4. Therefore, we believe that it is worth to merge the two streams of research, by now largely independent, in order to harness the power of automated verification whenever learning methods require (formal) assurances related to their performances.

The second important challenge is to mate precision with scalability. As shown, *e.g.*, by [Pulina and Tacchella, 2012; Scheibler *et al.*, 2015], the main barrier to applying off-the-shelf automated reasoning tools to analyze NNs is that such tools hardly scale to deal with current state-of-the art models: DNNs like AlexNet or GoogleLeNet feature millions of parameters, resulting in prohibitively large search spaces for automated reasoning algorithms. This is why researchers either restrict the scope of application of general-purpose tools — see, *e.g.*, [Huang *et al.*, 2017] — or develop special-purpose solvers — see, *e.g.*, [Katz *et al.*, 2017; Ehlers, 2017]. Even if SAT and MIP techniques are known to scale very well on problems of considerable size, researchers using them also report limitations in dealing with networks of moderate size given current demands. Noticeably, a lot of research in the ML community has been carried out to reduce the size of ML models that are known to be highly redundant in terms of the number of parameters. Two examples of such techniques are model reduction and knowledge distillation. Model reduction methods take an original model and use, for example, quantization techniques to reduce its size while preserving accuracy. An extreme case of such quantization is binarizing most parameters of the network [Courbariaux *et al.*, 2015; Hubara *et al.*, 2016]. Similarly, knowledge distillation algorithms start with a full network and build a new smaller network without a significant accuracy loss. Conversely, on the Automated Reasoning side, there is a need of effective tools. For instance, large classes of NNs, *e.g.*, NNs with ReLU activation functions, can be encoded using linear arithmetic constraints enhanced with simple logical constraints — an approach considered by [Ehlers, 2017]; other examples of specialization include the Reluplex algorithm that analyzes only NNs with ReLU activation functions [Katz *et al.*, 2017] and, as mentioned in [Narodytska *et al.*, 2017; Korneev *et al.*, 2018], the usage of pseudo-Boolean constraints to encode binarized NNs in order to leverage pseudo-Boolean solvers rather than a fully fledged SMT solvers. Additionally, verification tools should exploit structural properties of NNs, *e.g.*, the layered graph structure of the network, and the parameter sharing features of convolutional layers. For example, abstraction techniques or clever decompositions can be used to verify only critical parts of the models — similarly to what [Huang *et al.*, 2017] proposed.

Overall, we believe that the synergy between Machine Learning and Automated Reasoning communities can be very fruitful. Automated verification of NNs could be the new driving force for theoretical and practical advancements in Automated Reasoning and, at the same time, ML could benefit from powerful verification techniques to generate proofs of correctness for NNs. Such techniques could even be used to explain models, making them more amenable to application in regulated contexts. Hybrid solutions could also be adopted, where verification could be embedded into the learning process, and learning could support effective verification. In the following, we briefly outline some of the directions that can be explored to improve on the current state of the art.

Common standards and APIs. Appropriate standards are to be developed such that learning tools can be easily and interchangeably connected to verification ones. APIs are required in cases where integration should be tight: for instance in [Pulina and Tacchella, 2010], repairing the NN required communication between verification and learning engines to leverage counterexamples and improve on learning.

Benchmark collections. Right now, most contributions cited in Section 4 consider their own case studies wherefore they provide specific results — with MNIST database images being an exception. The creation and the dissemination of a library of benchmarks —- respecting the standards defined above — will enable communities to share challenges and measure progress.

Hybrid reasoning methods. It is unlikely that complex, implements trained on dedicated hardware like modern DNNs can be verified by general purpose automated reasoning tools and hardware. Developers of automated reasoning techniques must realize this, and push innovation towards tools that can deal with, *e.g.*, non linearities, non-algebraic activation functions, and complex architectures, possibly harnessing the power of the same computing devices used to train NNs.

Transparent learning models. The tendency towards creating networks whose size and complexity makes them hardly explainable should probably be reconsidered, at least for networks whose safety or security must be guaranteed in all expected working conditions. Therefore, we need the ability to design and train networks whose effectiveness is uncompromised, but whose verification is feasible. This is a goal that the Machine Learning community must acknowledge and consider [Kuper *et al.*, 2018].

References

[Bastani *et al.*, 2016] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *NIPS*, pages 2613–2621, 2016.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[Bojarski *et al.*, 2016] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[Bunel *et al.*, 2017] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and M. Pawan Kumar. Piecewise linear neural network verification: A comparative study. *CoRR*, abs/1711.00455, 2017.

[Carlini and Wagner, 2017] Nicholas Carlini and David A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *AISec@CCS*, pages 3–14, 2017.

[Cheng *et al.*, 2017a] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *ATVA*, pages 251–268. Springer, 2017.

[Cheng *et al.*, 2017b] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Verification of binarized neural networks. *CoRR*, abs/1710.03107, 2017.

[Chollet and others, 2015] François Chollet et al. Keras. ”https://github.com/keras-team/keras”, 2015.

[Clarke and Zuliani, 2011] Edmund M Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *ATVA*, pages 1–12. Springer, 2011.

[Courbariaux *et al.*, 2015] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, pages 3123–3131, 2015.

[Dutta *et al.*, 2017] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks. *CoRR*, abs/1709.09130, 2017.

[Ehlers, 2017] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *ATVA*, pages 269–286, 2017.

[Fischetti and Jo, 2017] Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *CoRR*, abs/1712.06174, 2017.

[Goodfellow *et al.*, 2015] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[Gopinath *et al.*, 2017] Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark Barrett. Deepsafe: A data-driven approach for checking adversarial robustness in neural networks. *CoRR*, abs/1710.00486, 2017.

[Harrison, 2009] John Harrison. *Handbook of practical logic and automated reasoning*. Cambridge University Press, 2009.

[Huang *et al.*, 2017] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV*, pages 3–29, 2017.

[Hubara *et al.*, 2016] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NIPS*, pages 4107–4115, 2016.

[Jorgensen, 1997] Charles C Jorgensen. Direct adaptive aircraft control using dynamic cell structure neural networks. 1997.

[Julian *et al.*, 2016] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. Policy compression for aircraft collision avoidance systems. In *DASC*, pages 1–10. IEEE, 2016.

[Katz *et al.*, 2017] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*, pages 97–117, 2017.

[Korneev *et al.*, 2018] Svyatoslav Korneev, Nina Narodytska, Luca Pulina, Armando Tacchella, Nikolaj Bjorner, and Mooly Sagiv. Constrained image generation using binarized neural networks with decision procedures. *CoRR*, abs/1802.08795, 2018.

[Kumar *et al.*, 2012] Pratyush Kumar, Dip Goswami, Samarjit Chakraborty, Anuradha Annaswamy, Kai Lampka, and Lothar Thiele. A hybrid approach to cyber-physical systems verification. In *DAC*, pages 688–696. ACM, 2012.

[Kuper *et al.*, 2018] Lindsey Kuper, Guy Katz, Justin Gottschlich, Kyle Julian, Clark Barrett, and Mykel J. Kochenderfer. Toward scalable verification for safety-critical deep networks. In *SysML*, 2018.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[Narodytska *et al.*, 2017] Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. *CoRR*, abs/1709.06662, 2017.

[Pulina and Tacchella, 2010] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *CAV*, pages 243–257, 2010.

[Pulina and Tacchella, 2012] Luca Pulina and Armando Tacchella. Challenging SMT solvers to verify neural networks. *AI Commun.*, 25(2):117–135, 2012.

[Pullum *et al.*, 2007] Laura L. Pullum, Brian J. Taylor, and Marjorie A. Darrah. *Guidance for the Verification and Validation of Neural Networks (Emerging Technologies)*. Wiley-IEEE Computer Society, 2007.

[Scheibler *et al.*, 2015] Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker. Towards verification of artificial neural networks. In *MBMV*, pages 30–40, 2015.

[Schrijver, 1999] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

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

[Taigman *et al.*, 2014] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, pages 1701–1708, 2014.

[Tjeng and Tedrake, 2017] Vincent Tjeng and Russ Tedrake. Verifying neural networks with mixed integer programming. *CoRR*, abs/1711.07356, 2017.

[Yu *et al.*, 2012] Dong Yu, Geoffrey E. Hinton, Nelson Morgan, Jen-Tzung Chien, and Shigeki Sagayama. Introduction to the special section on deep learning for speech and language processing. *IEEE Trans. Audio, Speech & Language Processing*, 20(1):4–6, 2012.

[Zakrzewski, 2001] R. R. Zakrzewski. Verification of a trained neural network accuracy. In *IJCNN*, volume 3, pages 1657–1662 vol.3, 2001.