Perspectives on the Verification of Artificial Neural Networks

Patrick Musau
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
Email: patrick.musau@vanderbilt.edu

***Abstract*—Artificial Neural Networks have demonstrated an effective and powerful ability to solve problems in numerous contexts such as adaptive control, autonomous vehicles, evolutionary robotics, non-linear system identification, and image and pattern recognition. Despite achieving results that are comparable to human levels of accuracy and performance, there have been reservations about incorporating them into safety-critical systems due to an inability to formally reason about the underlying models that govern their behavior. In light of this challenge, there has been a significant amount of work towards the creation of verification methods that are capable of identifying erroneous behaviors and providing formal guarantees about the safe operation of neural network applications. The following paper presents a survey of these works from three perspectives in an effort to trace the intellectual progression of this field and stimulate the creation of advanced verification methods that are capable of addressing the major challenges currently being faced within this context.**

Keywords*—Artificial Intelligence, Neural Networks, Automated Verification, Formal Methods

I. Context and Origins

In recent years, the success of Artificial Neural Networks as frameworks for solving complex problems such as pattern and image recognition [1], machine translation [2], and function approximation [3], has led researchers to believe that these models will revolutionize the development of robust and intelligent systems in a diverse set of application domains [4]. Despite these achievements, there have been reservations in utilizing them within high assurance systems due to their susceptibility to unexpected and errant behavior caused by slight perturbations in their inputs [5]. In a famous study by Chrisitian Szegedy et al. [6], the authors demonstrated that by carefully applying a hardly perceptible modification to an input image, one could cause a successfully trained neural network to produce an incorrect classification. These inputs are known as *adversarial examples*, and their discovery has caused concern over the safety, reliability, and security of neural network applications [7]. As a result, there has been a large research impetus towards obtaining an explicit understanding of neural network behavior.

Neural networks are often viewed as "black boxes," whose underlying operation is often incomprehensible, and the last several years have witnessed a large number of promising verification methods proposed towards reasoning about the correctness of their behavior. However, it has been demonstrated that neural network verification is an NP-complete problem [8], and while current state-of-the-art verification methods have been able to deal with small networks, they are incapable of dealing with the complexity and scale of networks utilized in practice ([9]–[11]). The following paper presents a brief survey of neural network verification from three perspectives in order to trace the intellectual progression of this rapidly growing field and stimulate the development of efficient and effective methods capable of being utilized within real-life applications. The first perspective presents an analysis and classification of the existing verification methods, the second presents an experimental evaluation of two state-of-the-art software tools, and the third perspective classifies verification approaches into two main modes of thought.

II. Neural Network Preliminaries

Artificial Neural Networks consist of a number of interconnected neurons where each neuron can be perceived as a processing element that reacts to the weighted sum of the inputs it receives. The neurons are typically structured into three types of layers: an input layer, an output layer, and one or multiple hidden layers. Each connection between neurons is typically labeled with a real-valued weight that is determined during a training process that seeks to maximize the networks prediction accuracy [12]. The overall structure of the network varies greatly depending on the specific architecture being considered, and while there are numerous architectures such as recurrent networks [13], radial basis function networks [14], long-term short-memory networks [15], and self organizing maps [16], the following manuscript will focus on feed-forward and convolutional neural networks as they are the most prevalent in the verification literature.

A. Feed-Forward Neural Networks

Feed-forward Neural Networks (FNN) are networks in which the connections between neurons extend forward in a single direction. Each layer is fully connected to the next one, but connections between neurons within the same layer are prohibited. The output of a neuron is computed by calculating the weighted sum of the outputs of neurons in the previous layer and applying a non-linear activation function to that sum shifted by a bias. The action of a single neuron can be described by:

$$u_i = f(\sum_{j=1}^{n} w_{ij}x_j + b_i) \tag{1}$$

where x_j is the j-th input of the i-th neuron, w_{ij} is the connection weight from the j-th input to the i-th neuron, b_i is the bias of the i-th neuron, u_i is the output of i-th neuron, and $f(\cdot)$ is the activation function [3]. There are a number of different activation functions used in practice, however, the three most popular activation functions are the logistic sigmoid

$\sigma(x) = 1/(1+e^{-x})$, hyperbolic tangent $\tanh(x)$, and the rectified linear unit (ReLU) $\max(0,x)$[1].

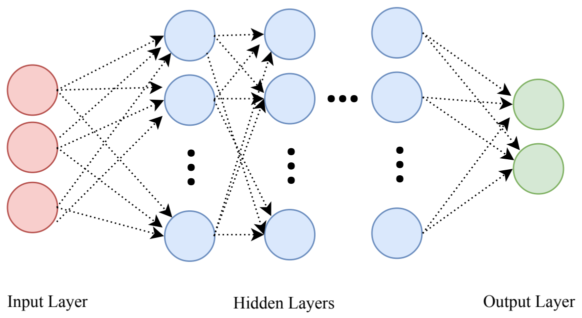

Figure 1: Simple Feed-forward Architecture

Formally, the operation of a neural network can be viewed as a function, $v : I^n \rightarrow O^m$, that maps an n-dimensional input, I^n, to an m-dimensional output, O^m [17]. For a neural network with L layers, where $1 \leq \ell \leq L$, let s_ℓ denote the size of the layer ℓ, s_1 denote the size of the input layer, $v_{\ell,j}$ denote the value of the j-th neuron of layer ℓ, b_ℓ be a bias vector of size $s_\ell \times 1$, and $U_\ell = [u_{\ell,1}, \ldots, u_{\ell,s_\ell}]^T$ be a column vector of neuron activations for layer ℓ ([3], [8]). For each layer $2 < \ell < L$, there is an associated weight matrix, W_ℓ, of size $s_\ell \times s_{\ell-1}$. Thus, the neuron activation vector, U_ℓ, can be described by:

$$U_\ell = f(W_\ell \cdot U_{\ell-1} + b_l) \tag{2}$$

where the activation function $f(\cdot)$ is applied component-wise [8]. Given an input, U_1, the output of the whole network is given by applying rule (2) repeatedly until U_ℓ is calculated.

B. *Convolutional Neural Networks*

Convolutional neural networks (CNN) are very similar to feed-forward networks but make the explicit assumption that the inputs are images [18]. Thus, their structure is tailored towards dealing with images in an efficient manner. Since images can be thought of as being organized into three dimensions, namely in terms of height, width, and depth (typically RGB values), CNNs have neurons that are organized with respect to these dimensions. These layers are called *convolutional layers* and consist of a set of learn-able filters that we convolve over the image in order to compute neuron activations. Additionally, in order to limit the number of parameters that must be learned during training, CNNs also include layers called *pooling layers* that perform a down sampling operation on the input volume of the previous layer ([18], [19]). We refer the reader to the following papers for an in-depth discussion of this class of network ([18], [19]).

III. Neural Network Verification

A. *Verification Problem*

The verification problem for neural networks refers to the process of proving that the output of a network satisfies some desirable property for every choice of input within a bounded set [20]. To accomplish this, the problem is posed as a search for an input that causes the negation of a desired property to be true. If the search is successful, then we conclude that the property does not hold and the obtained solution serves as a counterexample demonstrating this failure [21]. Otherwise, the search fails and we conclude that the property is true. Formally, the problem can be stated as: given a neural network that maps an input, x_1, to an output, u_1, via $u_1 = v(x_1)$, along with an input domain, C, and a property P, we wish to prove:

$$\forall x_1 \in C,\; u_1 = v(x_1) \Rightarrow P(u_1) \tag{3}$$

where $P(u_1)$ describes the undesired behavior for the network ([20], [21]). Ultimately, the verification problem is obtained by encoding the network as a constraint system as in (3) and determining whether there exists an assignment that satisfies all of the the constraints [17].

The problem posed in (3) is a deeply challenging problem and it has been demonstrated that proving even simple properties about the behavior of neural networks is an NP-complete problem [8]. This challenge is primarily caused by the presence of nonlinear activation functions. While, these functions enable neural networks to learn complex mappings, they also make the verification process more difficult [22]. The principal challenge, as in other realms of verification, is developing techniques that will scale to the size of networks encountered in practice [11]. Currently, the most successful methods deal with feed-forward and convolutional neural networks that make use of piece-wise linear activation functions, such as ReLU and max pooling [23]. The remainder of this section provides an introduction to the procedures commonly used in solving verification problems posed as (3).

B. *Satisfiability Modulo Theories*

Satisfiability Modulo Theories (SMT) generalizes the Boolean satisfiability (SAT) problem, into the problem of finding a series of assignments, called an interpretation, such that a first-order logic formula, with respect to some background theory evaluates to true [17]. A background theory is a set of sentences in a formal language that can be used to restrict the interpretations considered in the SMT decision problem [24]. Commonly considered background theories include: the theory of integers, the theory of real numbers, and the theory of linear arithmetic [22]. To determine the satisfiability of a given SMT problem, SMT solvers typically build a boolean abstraction of the formula by replacing each constraint in the formula with a boolean variable and use a state-of-the-art SAT solver to search for a satisfying assignment [17]. If the solver fails to find a satisfying assignment, then the solver concludes that the SMT formula is unsatisfiable. However, if the SAT solver finds a suitable interpretation, a theory solver is used to determine whether or not the interpretation is consistent within the context of the background theory [24]. If the solution is inconsistent, the theory solver provides an explanation of the inconsistency and refines the boolean abstraction of the SMT formula [17]. This process occurs continuously and terminates either when a theory-consistent interpretation has been found or when the boolean abstraction of the formula is found to be unsatisfiable [17].

There are a number of powerful and efficient off-the-shelf SMT solvers [24] that are well-suited to the verification of

[1]A discussion of the influence of activation functions in neural networks can be found in [12]

neural networks with piecewise linear activation functions. By encoding properties as in (3), using SMT formulas, researchers have been able to verify feed-forward neural networks with about 1800 neurons [8], and convolutional neural networks with up to 1.25 million learnable parameters [19].

C. Linear and Mixed Integer Programming

Neural network verification can also be posed as an optimization problem using *Linear* (LP) and *Mixed Integer Programming* (MIP). Mixed Integer Programming is an extension of Linear Programming that involves the optimization of a function over a set of integer and real-valued variables subject to a set of linear constraints [17]. Traditionally, the problem is described by variables known as *decision variables*, and a linear objective function that we wish to minimize or maximize. Formally this can be expressed as:

$$\min \sum_{j=1}^{n} c_i x_i \tag{4}$$

$$\text{subject to} \sum_{j=1}^{n} a_{ji} x_i \geq b_j, \quad j \in [1, m] \tag{5}$$

where $x_i \in [a_i, b_i] \cap \mathbb{Z}$ are the bounded decision variables, (4) is the objective function, and (5) are the set of linear constraints that need to be upheld ([17], [21]).

To formulate the neural network verification problem as an MIP problem the network must employ the use of piece-wise linear activation functions and the property considered must also be expressible as a set of linear set of constraints. An in-depth discussion of the process of encoding neural networks into MIP problems can be found in the following papers ([23], [25], [26]).

D. Other Approaches

While the SMT and MIP formulations of the neural network verification problem are the most commonly used techniques within the research literature, there are several other notable techniques not characterized by the philosophy of (3). These methods include: *test coverage methods*, and *concolic testing*. Test coverage methods, which draw inspiration from software engineering, are white-box testing methodologies aimed at generating exhaustive test cases to gauge the correctness of a neural network's behavior. There have been a number of promising techniques proposed within this realm and a discussion of these results can be found in the following survey [7]. Concolic testing methods are similar to test coverage methods, however, they combine program execution and symbolic methods in order to explore paths that are hard to cover via test coverage techniques[2]. These approaches have largely been successful in testing networks for adversarial input robustness, but do not provide comprehensive assurance that a neural network is free from defect [7]. The remainder of this paper will focus on SMT and MIP based verification.

[2]We refer readers to the following paper for a discussion of concolic testing methods [27]

IV. Perspectives on Neural Network Verification

This section presents neural network verification from three perspectives: the first perspective presents a survey and classification of the existing verification methods, the second presents an experimental evaluation of two state-of-the-art software tools designed for networks with piece-wise linear activation functions, and the third perspective classifies verification approaches into two main modes of thought: complete verification techniques and incomplete verification techniques.

A. Leofante et al., Invariance, Invertibility, and Equivalence

In their survey, Francesco Leofante et al. present a comprehensive categorization of neural network verification schemes into three distinct classes, based on the particular type of property being considered for verification. The three properties that the authors consider are: invariance, invertibility, and equivalence [17].

1) Invariance: Verification methods that deal with invariance are described by (3), and Leofante et al. note that the majority of neural network verification methods deal with properties of this nature [7]. As a result, Leofante et al. note that there are two different notions of invariance: global invariance and local invariance. Techniques that cater to local invariance properties are often concerned with searching for adversarial examples and seek to demonstrate the invariance of a classification decision within a small neighbourhood of a single input ([19], [28]). These techniques are often either posed as a MIP optimization problem, or as an SMT problem.

Global invariance techniques, on the other hand, deal with properties that consider all inputs within a bounded domain, and as a result, these properties are harder to demonstrate. Currently, the most successful methods have been able to verify neural networks with piece-wise linear activation functions on the order of thousands of neurons ([8], [21], [23]). Constrastingly, for networks that make use of more general activation functions, the most successful techniques have only been able to deal with networks on the order of tens to hundreds of neurons [29]. It is also worth noting that the vast majority of invariance verification techniques deal with networks with feed-forward architectures and Leofante et al. cite only one article capable of dealing with convolutional architectures [19].

2) Invertibility: The second class of methods that Leofante et al. consider are methods that deal with invertibility. Properties relating to invertibility can be posed similarly to falsification problems within the context of formal methods [7]. Given an output $u_1 = v(x_1)$ that satisfies some property $P(u_1)$ the task is to find an input, x_1, such that the input is part of a particular bounded input domain C [17]. In this regime, the central focus is obtaining a specific input counterexample that causes the network to exhibit some undesired behavior described by P. At the time of Leofante et al.'s publication, the authors cited only two articles by Rudiger Ehlers et al. [30] and Svyatoslav Korneev [31], that specifically dealt with neural network invertibility. However, since then, there have been several proposed verification approaches such as ([32], [33]). We refer readers to the following survey for a comprehensive exploration of the aforementioned techniques [7].

3) Equivalence: Lastly, Leofante et al. examine verification methods that deal with network equivalence. Network equivalence properties involve two distinct networks, and aim to find an input such that the outputs of the networks are dissimilar. Properties relating to network equivalence are popular within studies that consider adversarial input robustness, and in this context, researchers wish to guage whether or not an adversarial example discovered for a specific neural network, will also be incorrectly classified by a network that is trained on a separate set of data. Formally, this can be described by:

$$\forall x_1 \in C, \; (u_1 = v(x_1) \Rightarrow P(u_1) \wedge \\ u_2 = v'(x_1) \Rightarrow Q(u_2)) \Rightarrow u1 = u_2 \tag{6}$$

where x_1 is a bounded input, $v(x)$ and $v'(x)$ are the functions implemented by the two networks, and $Q(x)$ and $P(x)$ are properties relating to the the network outputs u_1 and u_2 [17]. Although there are several adversarial input robustness studies that consider a local flavor of this problem, Leofante et al. cite only work by Narodystka et al. [28] that specifically deals with global formulation of this problem as given by (3). Thus, the authors cite this as a potential area of interest for further investigation.

B. Bunel et al.'s Unified View of Piece-wise Linear Neural Network Verification

While Leofante et al.'s survey presents a comprehensive categorization of verification methodologies, another survey by Rudy Bunel et al. [23], provides an experimental evaluation of two efficient verification software tools for piece-wise linear neural networks, on a set of benchmark problems with closely related properties. The authors argue that verification methods that deal with networks with piece-wise linear activation functions can be viewed as special cases of branch and bound optimization. By formulating the verification problem in this fashion, the authors designed a novel software tool that surpassed the performance of state-of-the-art methods at that time.

1) Branch and Bound Optimization: Bunel et al. assert that verification problems formulated as satisfiability problems, as in (3), can be reduced to a global optimization problem where the satisfiability of a formula will be obtained by checking the sign of the minimum. Thus, given a property $P(u_1)$ that can be expressed as boolean formula over linear inequalities, the authors add an additional layer to the network with one output that represents the property considered for verification [23]. As an example, Bunel et al. state, that for a property expressed as linear inequality given by $P(u_1) = c \cdot u_1 \geq b$, one can add a final layer with connection weights given by the vector, c, and a bias vector given by $-b$ [23]. With the addition of this layer, the verification problem is reduced to finding an input such that the output of the network is negative. If the output is negative, then the value given by an MIP optimizer is a counterexample. Otherwise, if the output is positive, then we can conclude that the property is satisfied. The authors assert that the advantage of formulating the problem in this fashion, is that one can construct and prove several complex properties at once without having to create and run multiple instances of (3) as in other approaches ([8], [30]). An in-depth discussion of these techniques as well as the reduction of piece-wise linear verification procedures to branch and bound optimization can be found in [23].

2) Reluplex and Planet: The two verification tools that Bunel et al. consider for experimental evaluation are Reluplex,[3] designed by Guy Katz et al. [8], and Planet,[4] developed by Rüdiger Ehlers et al. [30]. Reluplex is a specialized SMT solver for verifying neural networks that make use of ReLU activation functions. The tool is based on an extension of the simplex algorithm [34] in linear optimization, that allows the authors to efficiently deal with SMT formulas that encode ReLU constraints. During the search for a satisifiable assignment to a query such as in (3), the principle of Reluplex is to always maintain an assignment to all of the variables even if some of the ReLU constraints are violated [8]. As the search progresses, at each step, Reluplex tries to fix a violated constraint and terminates either when a satisfying example is obtained or if the search fails to find a solution. In their paper, the authors demonstrate that Reluplex can handle networks with up to 1800 neurons ([8] [17]).

Correspondingly, Ehlers et al.'s tool, Planet, is based on an approach that leverages both SMT and MIP solvers [30]. In their framework, the authors make use of a linear approximation of the neural network's behavior, in order to prune the search space that the solvers must consider during execution. Given a particular linear constraint system encoding of a network, Planet tries to find a satisfying interpretation, and at each step, the tool assigns a value of 0 or 1 to each variable associated with ReLU $(\max(0, x))$ or pooling functions. If a satisfying solution is found, the tool returns the solution as counterexample. Otherwise, if there are no solutions corresponding to the current partial assignment, the tool backtracks, and attempts a different assignment. This search is done iteratively until either a satisfying solution is found, or if the solver concludes that no solution exists ([21], [30]). In their work, Ehlers et al. demonstrate that they are able to prove properties for networks with over 1341 neurons.

3) Experimental Results: To assess the performance of the aformentioned tools, Bunel et al. utilized three benchmark data sets with a diverse set of properties considered for verification and of varying dimensionality [23]. The results of their experiments, demonstrated that by expressing neural network verification as a branch and bound optimization problem, their software tool was able to surpass the performance of Reluplex and Planet [23]. In some cases, their tool executed with a speedup of over two orders of magnitude. However, despite significant improvements in run-time, the authors did not present an evaluation of whether their tool achieved better scalability than the methods proposed by Katz et al. and Ehlers et al. Thus, it is unclear whether or not Bunel et al.'s methods are superior to those of Reluplex and Planet. This is an interesting question that merits further investigation[5].

[3]Guy Katz et al.'s verification tool Reluplex is available at https://github.com/guykatzz/ReluplexCav2017

[4]Rüdiger Ehlers et al.'s verification tool Planet is available at https://github.com/progirep/planet

[5]A detailed summary of the experimental results of Bunel's experiments can be found online at https://github.com/oval-group/PLNN-verification, and in the following paper [23]

Tool Name	Network Type	Verification Approach	Largest Network Considered	Completeness	Type of Property
Reluplex [8]	FNN	SMT	1800 neurons	Complete	Invariance
Planet [30]	FNN, CNN	SMT, MIP	1341 neurons	Complete	Invariance, Invertibility
PLNN [23]	FNN, CNN	Branch and Bound	1485	Complete	Invariance
Sherlock [35]	FNN	MIP	3822	Complete	Invariance
VeriDeep/DLV [19]	FNN, CNN	SMT	1.25×10^6 parameters	Incomplete	Invariance
NNAF [25]	CNN	LP	60000 parameters	Incomplete	Invariance
MIPVerify.jl [26]	FNN, CNN	MIP	500 neurons	Incomplete	Invariance, Invertibility
AI^2 [36]	FNN, CNN	Abstract Interpretation	53000 neurons	Incomplete	Invariance

Table I: Summary of several available software tools for network verification present in the research literature. A more comprehensive list of tools can be found in [7].

C. Modes of thought within Neural Network Verification

The last perspective we consider in this paper is a categorization of neural network verification methods into two main modes of thought, as described by Guy Katz in [8]. One of the greatest challenges in verification is designing methods that scale well to real world applications. To address this challenge, two types of techniques have arisen in the research literature: methods that are sound and complete, and methods that are sound and incomplete. [21]. Intuitively, a sound method is a method that always returns a correct answer, and a complete method is one that is guaranteed to find a solution if one exists [21]. Katz asserts that techniques which are sound and complete are typically limited in terms of scalability. As a result these methods have been able to deal with networks with sizes on the order of hundreds to thousands of neurons. As an example, Reluplex and Planet are techniques that are both sound and complete, and there are a number of other approaches that fall within this category ([7], [26], [35], [37]). A summary of the existing software tools can be found in Table I.

On the other hand, sound and incomplete techniques exhibit better scalability and can handle networks with thousands to tens of thousands of neurons [8]. While these techniques are superior in terms of scalability, they may not return an answer to the specific verification problem posed by the researcher. However, these approaches often work well in practice and there have been a number of notable approaches proposed in recent years. A discussion of these methods can be found in [19] and [7]. For instance, in their paper, Osbert Bastani et al. present a method for deriving adversarial inputs with some neighborhood of an input image [25]. While their methods can generate adversarial examples, if they exist, they cannot guarantee that there are no other adversarial examples within this region to be discovered. Katz notes that comparing complete and incomplete techniques is difficult, and in the future, an analysis of the benefits and drawbacks of both methods merits investigation.

V. Major Challenges and Future Work

The last several years have witnessed a significant increase in the the number of proposed verification methods for neural networks [7]. Currently the vast majority of research efforts have focused on designing methods that scale to networks utilized in real world applications. Unfortunately, networks such as GoogLeNet, AlexNet, and VGGNet that contain millions of neurons ([38], [39]) are currently out of reach. The current limiting factor in these efforts is an effective treatment of the non-linearities introduced by activation functions, and at the present time, the vast majority of methods can only handle functions that are piecewise linear [22]. Regrettably, it is not clear how this problem can be overcome, and thus there is an urgent need for the additional research towards the development of methods that will work well in practice.

In addition to creating more scalable methods, there are several other open problems within this realm. Presently, there is a lack of effective methodology for comparing the existing verification techniques due to the diversity of the underlying SMT and SAT solvers utilized in state-of-the-art tools, a lack of standardized benchmarks, distinct verification modes of thought, and a great variety of properties considered for verification. In fact, Guy Katz notes that often the property being considered for verification has a large influence on the complexity of the problem [8]. Moreover, there is a lack of a standardized input format describing neural network verification problems for the available software tools within the community. As a result, it is difficult for researchers to efficiently and accurately evaluate the success of their techniques.

Lastly, while the vast majority of the existing verification methods can provide examples that are realizations of undesired behavior, they have not yet provided an intuitive way of creating models that are free from defect. Due to the opaqueness of neural network models, techniques that describe how to modify a network's structure to cause the network to satisfy some desirable property would be extremely useful [8].

In light of these challenges, there are several interesting directions for future work including:

- The creation of standardized benchmark sets and input formats for the effective evaluation of proposed verification methods,
- Constructing SMT and SAT solvers that are specifically designed to handle formulas that involve neural network constraints such as Reluplex ([8], [22]),
- A comprehensive cost-benefit analysis of complete and incomplete neural network verification approaches,
- Interdisciplinary studies towards the creation of interpretable neural network models from domain experts from the Artificial Intelligence and Automated Verification communities
- And further research into other forms of verification such as run-time monitoring for systems that incorporate neural networks into their architecture.

If the research community can obtain acceptable solutions to the aforementioned challenges, they will stimulate the development of robust and intelligent systems with the potential to bring unparalleled benefits to numerous application domains. Moreover, they will provide the basis for theoretical and practical advances within artificial intelligence and automated verification.

REFERENCES

[1] J. Schumann, P. Gupta, and Y. Liu, *Application of Neural Networks in High Assurance Systems: A Survey*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–19. [Online]. Available: https://doi.org/10.1007/978-3-642-10690-3_1

[2] S. J. Russell, D. Dewey, and M. Tegmark, "Research Priorities for Robust and Beneficial Artificial Intelligence," *CoRR*, vol. abs/1602.03506, 2016. [Online]. Available: http://arxiv.org/abs/1602.03506

[3] W. Xiang and T. T. Johnson, "Reachability Analysis and Safety Verification for Neural Network Control Systems," *CoRR*, vol. abs/1805.09944, 2018. [Online]. Available: http://arxiv.org/abs/1805.09944

[4] S. Russell, D. Dewey, and M. Tegmark, "Research Priorities for Robust and Beneficial Artificial Intelligence," *AI Magazine*, vol. 36, no. 4, p. 105, 2015.

[5] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *CoRR*, vol. abs/1607.02533, 2016. [Online]. Available: http://arxiv.org/abs/1607.02533

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013. [Online]. Available: http://arxiv.org/abs/1312.6199

[7] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. Rosenfeld, and T. T. Johnson, "Verification for Machine Learning, Autonomy, and Neural Networks Survey," *CoRR*, vol. abs/1810.01989, 2018. [Online]. Available: https://arxiv.org/abs/1810.01989

[8] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," *CoRR*, vol. abs/1702.01135, 2017. [Online]. Available: http://arxiv.org/abs/1702.01135

[9] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A Survey of Deep Neural Network Architectures and their Applications," *Neurocomputing*, vol. 234, pp. 11 – 26, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231216315533

[10] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A Dual Approach to Scalable Verification of Deep Networks," *CoRR*, vol. abs/1803.06567, 2018. [Online]. Available: http://arxiv.org/abs/1803.06567

[11] R. Alur, "Formal Verification of Hybrid Systems," in *Proceedings of the Ninth ACM International Conference on Embedded Software*, ser. EMSOFT '11. New York, NY, USA: ACM, 2011, pp. 273–278. [Online]. Available: http://doi.acm.org/10.1145/2038642.2038685

[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[13] Z. C. Lipton, "A Critical Review of Recurrent Neural Networks for Sequence Learning," *CoRR*, vol. abs/1506.00019, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019

[14] S. E. V.T. and Y. C. Shin, "Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems," *IEEE Transactions on Neural Networks*, vol. 5, no. 4, pp. 594–603, July 1994.

[15] H. Sak, A. W. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," *CoRR*, vol. abs/1402.1128, 2014. [Online]. Available: http://arxiv.org/abs/1402.1128

[16] T. Kohonen, "The Self-Organizing Map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, Sept 1990.

[17] F. Leofante, N. Narodytska, L. Pulina, and A. Tacchella, "Automated Verification of Neural Networks: Advances, Challenges and Perspectives," *CoRR*, vol. abs/1805.09938, 2018. [Online]. Available: http://arxiv.org/abs/1805.09938

[18] A. Karpathy, "Convolutional Neural Networks (CNNs / ConvNets)," http://cs231n.github.io/convolutional-networks/. [Online]. Available: http://cs231n.github.io/convolutional-networks/

[19] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety Verification of Deep Neural Networks," *CoRR*, vol. abs/1610.06940, 2016. [Online]. Available: http://arxiv.org/abs/1610.06940

[20] K. Dvijotham, R. Stanforth, S. Gowal, T. A. Mann, and P. Kohli, "A Dual Approach to Scalable Verification of Deep Networks," *CoRR*, vol. abs/1803.06567, 2018. [Online]. Available: http://arxiv.org/abs/1803.06567

[21] G. Katz, ""Verification of Machine Learning Programs"." St Anne's College (Oxford): 2nd School on Foundations of Programming and Software Systems, 7 2018, st Anne's College (Oxford), Oxford, England, July 4, 2018.

[22] L. Kuper, G. Katz, J. Gottschlich, K. Julian, C. Barrett, and M. J. Kochenderfer, "Toward Scalable Verification for Safety-Critical Deep Networks," *CoRR*, vol. abs/1801.05950, 2018. [Online]. Available: http://arxiv.org/abs/1801.05950

[23] R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and M. P. Kumar, "Piecewise Linear Neural Network verification: A Comparative Study," *CoRR*, vol. abs/1711.00455, 2017. [Online]. Available: http://arxiv.org/abs/1711.00455

[24] C. Barrett and C. Tinelli, "Satisfiability Modulo Theories," in *Handbook of Model Checking*, E. Clarke, T. Henzinger, and H. Veith, Eds., 2018, in preparation. [Online]. Available: http://www.cs.stanford.edu/~barrett/pubs/BT14.pdf

[25] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, "Measuring Neural Net Robustness with Constraints," *CoRR*, vol. abs/1605.07262, 2016. [Online]. Available: http://arxiv.org/abs/1605.07262

[26] V. Tjeng and R. Tedrake, "Verifying Neural Networks with Mixed Integer Programming," *CoRR*, vol. abs/1711.07356, 2017. [Online]. Available: http://arxiv.org/abs/1711.07356

[27] Y. Sun, X. Huang, and D. Kroening, "Testing Deep Neural Networks," *CoRR*, vol. abs/1803.04792, 2018. [Online]. Available: http://arxiv.org/abs/1803.04792

[28] N. Narodytska, S. Prasad Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying Properties of Binarized Deep Neural Networks," *ArXiv e-prints*, Sep. 2017.

[29] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 243–257.

[30] R. Ehlers, "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks," *CoRR*, vol. abs/1705.01320, 2017. [Online]. Available: http://arxiv.org/abs/1705.01320

[31] S. Korneev, N. Narodytska, L. Pulina, A. Tacchella, N. Bjorner, and M. Sagiv, "Constrained Image Generation Using Binarized Neural Networks with Decision Procedures," *ArXiv e-prints*, Feb. 2018.

[32] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," *CoRR*, vol. abs/1710.07859, 2017. [Online]. Available: http://arxiv.org/abs/1710.07859

[33] T. Dreossi, S. Jha, and S. A. Seshia, "Semantic Adversarial Deep Learning," *CoRR*, vol. abs/1804.07045, 2018. [Online]. Available: http://arxiv.org/abs/1804.07045

[34] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965. [Online]. Available: http://dx.doi.org/10.1093/comjnl/7.4.308

[35] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output Range Analysis for Deep Neural Networks," *CoRR*, vol. abs/1709.09130, 2017. [Online]. Available: http://arxiv.org/abs/1709.09130

[36] T. Gehr, M. Mirman, D. D. Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI^2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation," *IEEE Symposium on Security and Privacy*, vol. 39, May 2018. [Online]. Available: https://www.srl.inf.ethz.ch/publications.php

[37] A. Lomuscio and L. Maganti, “An approach to reachability analysis for feed-forward ReLU neural networks,” *CoRR*, vol. abs/1706.07351, 2017. [Online]. Available: http://arxiv.org/abs/1706.07351

[38] P. Ballester and R. M. Araujo, “On the performance of googlenet and alexnet applied to sketches,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI’16. AAAI Press, 2016, pp. 1124–1128. [Online]. Available: http://dl.acm.org/citation.cfm?id=3015812.3015979

[39] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556