Reachability is NP-Complete Even for the Simplest Neural Networks

Marco Sälzer(✉) and Martin Lange

School of Electr. Eng. and Computer Science, University of Kassel, Kassel, Germany
{marco.saelzer,martin.lange}@uni-kassel.de
https://www.uni-kassel.de/eecs/fmv

Abstract. We investigate the complexity of the reachability problem for (deep) neural networks: does it compute valid output given some valid input? It was recently claimed that the problem is NP-complete for general neural networks and conjunctive input/output specifications. We repair some flaws in the original upper and lower bound proofs. We then show that NP-hardness already holds for restricted classes of simple specifications and neural networks with just one layer, as well as neural networks with minimal requirements on the occurring parameters.

Keywords: Machine learning · Computational complexity · Formal specification and verification

1 Introduction

Deep learning has proved to be very successful for highly challenging or even otherwise intractable tasks in a broad range of applications such as image recognition [11] or natural language processing [5] but also safety-critical applications like autonomous driving [4], medical applications [12], or financial matters [2]. These naturally come with safety concerns and the need for certification methods. Recent such methods can be divided into (I) Adversarial Attack and Defense, (II) Testing, and (III) Formal Verification. A comprehensive survery about all three categories is given in [6].

The former two cannot guarantee the absence of errors. Formal verification of neural networks (NN) is a relatively new area of research which ensures completeness of the certification procedure. Recent work on sound and complete verification algorithms for NN are mostly concerned with efficient solutions to their reachability problem NNReach [1,3,8,13]: given an NN and symbolic specifications of valid inputs and outputs, decide whether there is some valid input such that the corresponding output is valid, too. This corresponds to the understanding of reachability in classical software verification: valid sets of inputs and outputs are specified and the question is whether there is a valid input that leads to a valid output. Put differently, the question is whether the set of valid outputs is reachable from the set of valid inputs. The difference to classical reachability problems in discrete state-based programs is that there reachability is a matter

P. C. Bell et al. (Eds.): RP 2021, LNCS 13035, pp. 149–164, 2021.
https://doi.org/10.1007/978-3-030-89716-1_10

of *lengths* of a connection. In NN this is given by the number of layers, and it is rather the *width* of the continuous state space which may cause unreachability.

Solving NNReach is interesting for practical purposes. An efficient algorithm can be used to ensure that no input from some specified set of inputs is misclassified or that some undesired class of outputs is never reached. In applications like autonomous-driving, where classifiers based on neural networks are used to make critical decisions, such safeguards are indispensable.

However, all known algorithms for NNReach show the same drawback: a lack of scalability to networks of large size which, unfortunately, are featured typically in real-world applications [10]. This is not a big surprise as the problem is NP-complete. This result was proposed by Katz et al. [8] for NN with ReLU and identity activations, and later also by Ruan et al. [14]. While there is no reason to doubt the NP-completeness claim, the proofs are not stringent and contain flaws.

The argument for the upper bound in [8] misses the fact that real inputs are not necessarily polynomially bounded in size. In fact, guessing values in $\mathbb{R}$ is not even effective without a bound on the size of their representation. Such a bound is closely linked to the question whether such values can be approximated upto some precision. The proof by Katz et al. makes no argument for any bound on the representation of such values, let alone a polynomial one.[1]

The arguments for the lower bound by a reduction from 3Sat in [8] and [15] rely on a discretisation of real values to model Boolean values. This does not work for the signum function σ used by Ruan et al. as it is not congruent for sums: e.g. $\sigma(-3) = \sigma(-1)$ but $\sigma(2 + (-3)) \neq \sigma(2 + (-1))$, showing that one cannot simply interpret any negative number as the Boolean value *false* etc. As a consequence, completeness of the construction fails as there are (real) solutions to NNReach which do not correspond to (discrete) satisfying 3Sat assignments. Katz et al. seem to be aware of this and use a slightly more elaborate discretisation in their reduction, but unfortunately it still suffers from similar problems.[2]

We start our investigations into the complexity of NNReach by fixing these issues in Sect. 3. We provide a different argument for membership in NP which shows that the need for nondeterminism is not to be sought in the input values but in the use of ReLU nodes. As a corollary we obtain polynomial decidability for NN with a bounded number of such nodes. We also address the issue of discretisation of real values in the lower bound proof, fixing the construction given by Katz et al. We do not address the one by Ruan et al. further, as this does not provide any further insights or new results.

We then observe that the reduction from 3Sat constructs a very specific class of NNReach instances which we call $\mathcal{C}(3\text{Sat})$. NN from this class have a fixed amount of layers but scaling input and output dimension as well as layer size. This raises the question whether, in comparison to the networks from $\mathcal{C}(3\text{Sat})$,

[1] While this paper was being processed, Katz et al. published an extended version of their original paper [9]. Unfortunately, the flaws concerning the upper bound are still present in this version.

[2] These problems are repaired in [9], but in a slightly different way than we do.

reducing the amount of layers or fixing dimensionality leads to a class of networks for which NNReach is efficiently solvable. In Sect. 4 we show that the answer to this is mostly negative: NP-hardness of NNReach holds for NN with just one layer and an output dimension of one. While this provides minimal requirements on the structure of NN for NNReach to be NP-hard, we also give minimal criteria on the weights and biases in NN for NP-hardness to hold. Thus, the computational difficulty of NNReach in the sense of NP-completeness is quite robust. The requirements on the structure or parameters of an NN that are needed for NP-hardness to occur are easily met in practical applications. Due to space restrictions, some technical proof details are deferred to the appendix.

We conclude in Sect. 5 with references to possible future work.

2 Preliminaries

Definition 1. *A* neural network *(NN) N is a layered graph that represents a function of type $\mathbb{R}^n \to \mathbb{R}^m$.*

The first layer $l = 0$ is called the input layer and consists of n nodes. The i-th node computes the output $y_{0i} = x_i$ where x_i is the i-th input to the overall network. Thus, the output of the input layer $(y_{00}, \ldots, y_{0(n-1)})$ is identical to the input of N.

A layer $1 \leq l \leq L-2$ is called hidden *and consists of k nodes. Note that k must not be uniform across the hidden layers of N. Then, the i-th node of layer l computes the output $y_{li} = \sigma_{li}(\sum_j c_{ji}^{(l-1)} y_{(l-1)j} + b_{li})$ where j iterates over the output dimensions of the previous layer, $c_{ji}^{(l-1)}$ are real constants which are called* weights*, b_{li} is a real constant which is called* bias *and σ_{li} is some (typically non-linear) function called* activation*. The outputs of all nodes of layer l combined gives the output $(y_{l0}, \ldots, y_{l(k-1)})$ of the hidden layer.*

The last layer $l = L-1$ is called the output layer *and consists of m nodes. The i-th node computes an output $y_{(L-1)i}$ in the same way as a node in a hidden layer. The output of the output layer $(y_{(L-1)0}, \ldots, y_{(L-1)(m-1)})$ is considered as the output of the network N.*

The output of a neural network N under input $\boldsymbol{x}$ is denoted $N(\boldsymbol{x})$. If a node in a layer $l > 0$ has less inputs than there are outputs in layer $l-1$ then we assume that the unconsidered outputs of $l-1$ are weighted with zero. We only consider networks where nodes in hidden layers have the identity or the ReLU function, and nodes in the output layer have the identity as activation. The *ReLU function* is defined as $x \mapsto \max(0, x)$. Nodes with ReLU or identity activation are called ReLU nodes or identity nodes, respectively. Given some input to the NN, we say that a ReLU node is *active*, resp. *inactive* if the input for its activation function is greater, resp. less than or equal to 0. We visualize an NN as a directed graph with weighted edges. An example is given in Fig. 1.

Our main interest lies in the validity of specifications over the output values of NN given specifications over their input values. These specifications are expressed as conjunctions of linear constraints on the input and output variables of a network.

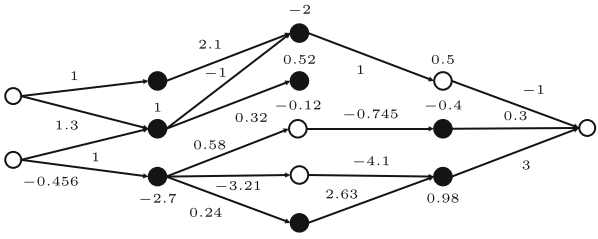

Fig. 1. Schema of a neural network with five layers, input dimension of two and output dimension of one. Filled nodes are ReLU nodes, empty nodes are identity nodes. An edge between two nodes u and v with label w denotes that the output of u is weighted with w in the computation of v. No edge between u and v implies $w = 0$. The bias of a node is depicted by a value above or below the node. If there is no such value then the bias is zero.

Definition 2. *A* specification φ *for a given set of variables* X *is defined by the following grammar:*

$$\varphi ::= \varphi \wedge \varphi \mid t \leq b \qquad\qquad t ::= c \cdot x \mid t + t$$

where b, c *are rational constants and* $x \in X$ *is a variable.*

We use $t \geq b$ and $t = b$ as syntactic sugar for $-t \leq -b$ and $t \leq b \wedge -t \leq -b$. Furthermore, we use $\top$ for $x + (-x) = 0$ and $\bot$ for $x + (-x) = 1$ where x is some variable. We call a specification φ *simple* if for all $t \leq b$ it holds that $t = c \cdot x$ for some rational constant c and variable x.

Definition 3. *Specification* $\varphi(x_0, \ldots, x_{n-1})$ *is* true *under* $\boldsymbol{x} = (r_0, \ldots, r_{n-1}) \in \mathbb{R}^n$ *if each inequality in* φ *is satisfied in real arithmetic with each* x_i *set to* r_i*.*

We write $\varphi(\boldsymbol{x})$ for the application of $\boldsymbol{x}$ to the variables of φ. If there are less variables in φ than dimensions in $\boldsymbol{x}$ we ignore the additional values of $\boldsymbol{x}$. If we consider a specification φ in context of a neural network N we call it an *input or output specification* and assume that the set of variables occurring in φ is a subset of the input respectively output variables of N.

Definition 4. *The decision problem* NNReach *is the following: given a neural network* N*, input specification* $\varphi_{in}(x_0, \ldots, x_{n-1})$ *and output specification* $\varphi_{out}(y_0, \ldots, y_{m-1})$*, is there* $\boldsymbol{x} \in \mathbb{R}^n$ *such that* $\varphi_{in}(\boldsymbol{x})$ *and* $\varphi_{out}(N(\boldsymbol{x}))$ *are true?*

3 NNReach is NP-Complete

3.1 Membership in NP

The argument used by Katz et al. to show membership of NNReach in NP can be summarized as follows: nondeterministically guess an input vector $\boldsymbol{x}$ as a witness, compute the output $N(\boldsymbol{x})$ of the network and check that $\varphi_{\text{in}}(\boldsymbol{x}) \wedge$

$\varphi_{\text{out}}(N(\boldsymbol{x}))$ holds. It is indisputable that the computation and check of this procedure are polynomial in the size of N, φ_{in}, φ_{out} *and* the size of $\boldsymbol{x}$. However, for inclusion in NP we also need the size of $\boldsymbol{x}$ to be polynomially bounded in the size of the instance given as $(N, \varphi_{\text{in}}, \varphi_{\text{out}})$. There may be an argument for this, for instance based on the correspondence between size of $\boldsymbol{x}$ and required approximation precision for such values. However, we are not aware of such an argument, let alone a striking one, and there is also a simpler way of obtaining the upper bound.

Definition 5. *A* ReLU-linear program *over a set* $X = \{x_0, \ldots, x_{n-1}\}$ *of variables is a set* Φ *of (in-)equalities of the form*

$$b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji} \leq x_j \qquad \text{or} \qquad \mathit{ReLU}(b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji}) = x_j$$

where $x_{ji}, x_j \in X$ *and* $c_{ji}, b_j \in \mathbb{Q}$. *Equations of the second form are called* ReLU-equations. *A* solution *to* Φ *is a vector* $\boldsymbol{x} \in \mathbb{R}^n$ *which satisfies all (in-)equalities when each variable* $x_i \in X$ *is replaced by* $\boldsymbol{x}(i)$. *A ReLU-equality* $\mathit{ReLU}(b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji}) = x_j$ *is satisfied by* $\boldsymbol{x}$ *if*

- $b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji} \geq 0$ *and* $x_j = b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji}$, *or*
- $b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji} \leq 0$ *and* $x_j = 0$.

The problem of solving *a ReLU-linear program is: given* Φ, *decide whether there is a solution to it.*

Any ReLU-linear program without ReLU-equalities is a linear program in the usual sense, and linear programs are known to be solvable in polynomial time [7].

Lemma 1. *The problem of solving a ReLU-linear program is in NP.*

Proof. Suppose a ReLU-linear program Φ with l ReLU-equalities is given. Existence of a solution can be decided as follows. Guess, for each ReLU-equation χ_k of the form $\mathit{ReLU}(b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji}) = x_j$, some $a_k \in \{0, 1\}$. Let $\boldsymbol{a} = (a_0, \ldots, a_{l-1})$. Next, let $\Phi_{\boldsymbol{a}}$ result from Φ by replacing each χ_k by the following (in-)equalities.

$$b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji} \geq 0 \ , \ b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji} = x_j \qquad \text{if } a_k = 1$$

$$b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji} \leq 0 \ , \ x_j = 0 \qquad \text{if } a_k = 0$$

The following is not hard to see: (I) Using standard transformations, $\Phi_{\boldsymbol{a}}$ can be turned into a linear program of size linear in Φ. (II) Any solution to $\Phi_{\boldsymbol{a}}$ is also a solution to Φ, (III) If Φ has a solution, then there is $\boldsymbol{a} \in \{0, 1\}^l$ such that $\Phi_{\boldsymbol{a}}$ has a solution. This can be created as follows. Let $\boldsymbol{x}$ be a solution to Φ. For each ReLU-equation χ_k as above, let $a_k = 1$ if the corresponding sum

is non-negative, otherwise let $a_k = 0$. Then $\boldsymbol{x}$ is also a solution for $\Phi_{\boldsymbol{a}}$. Thus, ReLU-linear programs can be solved in nondeterministic polynomial time by guessing $\boldsymbol{a}$, and then constructing the linear program $\Phi_{\boldsymbol{a}}$ and solving it. □

With this definition of a ReLU-linear program and the corresponding lemma at hand, we are set to prove NP-membership of NNReach.

Theorem 1. NNReach *is in NP.*

Proof. Let $\mathcal{I} = (N, \varphi_{\text{in}}, \varphi_{\text{out}})$. We construct a ReLU-linear program $\Phi_{\mathcal{I}}$ of size linear in $|N| + |\varphi_{\text{in}}| + |\varphi_{\text{out}}|$ which is solvable iff there is a solution for $\mathcal{I}$. The ReLU-linear program $\Phi_{\mathcal{I}}$ contains the following (in-)equalities.

- φ_{in} and φ_{out} (with each conjunct seen as one (in-)equality),
- for each non-ReLU node v_{li} computing $\sum_j c_{ji}^{(l-1)} y_{(l-1)j} + b_{li}$ add the equality $\sum_j c_{ji}^{(l-1)} y_{(l-1)j} + b_{li} = y_{li}$ (in the form of two inequalities of appropriate form),
- for each ReLU node v_{li} computing $ReLU(\sum_j c_{ji}^{(l-1)} y_{(l-1)j} + b_{li})$ add the ReLU-equality $ReLU(\sum_j c_{ji}^{(l-1)} y_{(l-1)j} + b_{li}) = y_{li}$.

The claim on the size of $\Phi_{\mathcal{I}}$ should be clear. Moreover, note that a solution $\boldsymbol{x}$ to $\mathcal{I}$ can be extended to an assignment $\boldsymbol{x}'$ of real values at every node of N, including values $\boldsymbol{y}$ for the output nodes of N s.t., in particular $N(\boldsymbol{x}) = \boldsymbol{y}$. Then $\boldsymbol{x}'$ is a solution to $\Phi_{\mathcal{I}}$. Likewise, a solution to $\Phi_{\mathcal{I}}$ can be turned into a solution to $\mathcal{I}$ by projection to the input variables.

Hence, NNReach polynomially reduces to the problem of solving ReLU-linear programs which, by Lemma 1 is in NP. □

It is interesting to point out the role of witnesses for positive instances of the NNReach problem: it is tempting to regard values to the input nodes of the NN as potential witnesses as done by Katz et al. but, as mentioned before, for as long as there is no argument for their polynomial boundedness these are *not* suitable witnesses in an NP procedure. Instead, Theorem 1 above shows that an assignment to the ReLU nodes as being in-/active can serve as such a witness. This immediately yields a polynomial fragment of NNReach.

Corollary 1. *The reachability problem for NN with a bounded number of ReLU nodes is decidable in polynomial time.*

3.2 NP-Hardness

Katz et al. try to build a polynomial-time reduction from 3Sat to NNReach. The underlying idea is to encode the structure of a 3Sat formula in a neural network and the existence of a satisfying assignment for this formula in the corresponding input- and output-specifications. Consider the 3Sat instance

$$\psi = (X_0 \vee X_1 \vee X_1) \wedge (\neg X_0 \vee X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$$

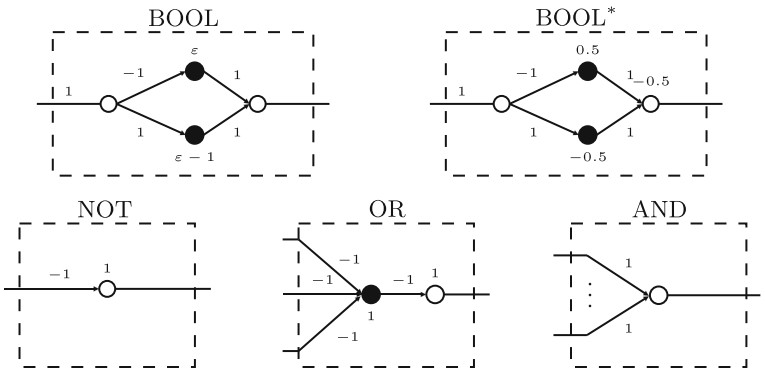

Fig. 2. Gadgets used in the reduction from 3SAT to NNREACH. A non-weighted outgoing edge of a gadget is connected to a weighted incoming edge of another gadget in the actual construction or is considered an output of the overall neural network.

with four propositional variables and three clauses, and let $(N, \varphi_{\text{in}}, \varphi_{\text{out}})$ be the NNREACH instance resulting from the mapping of ψ according to the reduction. To understand the structure of N we make use of so-called *gadgets*, specified in Fig. 2. Each gadget is a compact NN and is used to describe a functional subcomponent of N. Using these gadgets, the network N is depicted in Fig. 3.

Ignoring the BOOL-gadgets for the moment, assume that input values are taken from $\{0, 1\}$ instead of $\mathbb{R}$. The function computed by N is described as follows. Each of the three OR-gadgets together with their connected NOT-gadgets represent one of the clauses in ψ. From Fig. 2 we can infer that the NOT-gadgets negate their inputs and that the OR-gadgets output 1 if at least one input is 1 and 0 otherwise. Hence, if an OR-gadget outputs 1 then the current input, viewed as an assignment to the propositional variables in ψ, satisfies the corresponding clause. The AND-gadget simply sums up all of its inputs and, thus, we get that y is equal to 3 iff each OR-gadget outputs one. Therefore, with the output specification $\varphi_{\text{out}} := y = 3$, we get a reduction from 3SAT to NNREACH, provided that input values are externally restricted to $\{0, 1\}$.

But NN are defined for all real-valued inputs, so we need further adjustments to make the reduction complete. First, note that it is impossible to write an input specification $\varphi_{\text{in}}(\boldsymbol{x})$ which is satisfied by $\boldsymbol{x}$ iff $\boldsymbol{x} \in \{0, 1\}^n$ because $\{0, 1\}^n$ is not a hyperrectangle in $\mathbb{R}^n$ but conjunctions of inequalities only specify hyperrectangles. This is where we make use of BOOL-gadgets. Let ε be a very small constant. A BOOL-gadget with input x and output z computes $z = \max(0, \varepsilon - x) + \max(0, x - 1 + \varepsilon)$. Now, Katz et al. claim the following: if $x \in [0; 1]$ then we have $z \in [0; \varepsilon]$ iff $x \in [0; \varepsilon]$ or $x \in [1 - \varepsilon; 1]$. Thus, by connecting a BOOL-gadget to each input x_i in N and using the simple specifications

$$\varphi_{\text{in}} := \bigwedge_{i=0}^{3} x_i \geq 0 \wedge x_i \leq 1 \qquad \varphi_{\text{out}} := \bigwedge_{i=0}^{3} z_i \geq 0 \wedge z_i \leq \varepsilon \wedge y \geq 3(1 - \varepsilon) \wedge y \leq 3$$

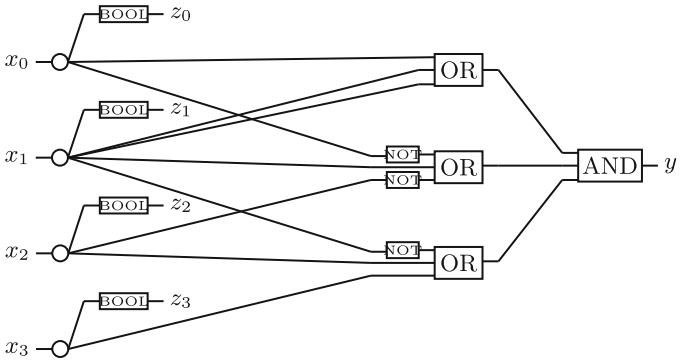

Fig. 3. Schema of a neural network resulting from the reduction of the 3SAT-formula $(X_0 \vee X_1 \vee X_1) \wedge (\neg X_0 \vee X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$. Note that no weights are depicted as these are specified inside the gadgets.

we would get a correct translation of ψ. Note that the constraint on y is no longer $y = 3$ as the valid inputs to N, determined by the BOOL-gadgets and their output constraints, are not exactly 0 or exactly 1. However, the claim about BOOL-gadgets is wrong. Consider a BOOL-gadget with very small ε such that it is safe to assume $\varepsilon < 2\varepsilon < 1 - \varepsilon$. Then, for $x = 2\varepsilon$ we have $z = 0$, which contradicts the claim. In fact, it can be shown that for each $\varepsilon \leq \frac{1}{2}$ and each input $x \in [0; 1]$ the output z is an element of $[0; \varepsilon]$. Clearly, this is not the intended property of these gadgets. But with some adjustments to the BOOL-gadgets we can make the reduction work.

A BOOL*-*gadget* is a neural network with functional form $\mathbb{R} \to \mathbb{R}$ shown in Fig. 2. It computes the function

$$z = \max\left(0, \frac{1}{2} - x\right) + \max\left(0, x - \frac{1}{2}\right) - \frac{1}{2},$$

where x is the input variable and z is the output variable. For this BOOL*-gadget we can show a similar statement as it was intended for the BOOL-gadgets in the original proof.

Lemma 2. *In a* BOOL**-gadget with input x and output z we have $z = 0$ if and only if $x = 0$ or $x = 1$.*

Proof. Note that $z = \max\left(0, \frac{1}{2} - x\right) + \max\left(0, x - \frac{1}{2}\right) - \frac{1}{2}$ is equivalent to

$$z = \begin{cases} -x & \text{if } x < \frac{1}{2}, \\ x - 1 & \text{otherwise.} \end{cases}$$

From this we immediately get that $z = 0$ if $x = 0$ or $x = 1$, and $z \neq 0$ for all other values of x. □

Now, replacing all BOOL-gadgets with BOOL*-gadgets in the construction and using the simple specifications $\varphi_{\text{in}} = \top$ and $\varphi_{\text{out}} = \bigwedge_{i=0}^{n-1} z_i = 0 \wedge y = m$ for a 3SAT-instance with n propositional variables and m clauses, we get a correct reduction from 3SAT to NNREACH.

Theorem 2. NNREACH *is NP-hard.*

One could argue that the networks resulting from the reduction of 3SAT are not typical feed-forward neural networks as they do not follow a layerwise structure. A reason for this is that some inputs are connected to NOT-gadgets where some are not and that the outputs z_i are not in the same layer as the output y. This can of course be fixed by introducing additional dummy nodes.

4 NP-Hardness Holds in Very Restricted Cases Already

Let $\mathcal{C}(3\text{SAT})$ be the class of NNREACH instances which are obtained as images under the reduction presented in the previous section. Note that the NN of $\mathcal{C}(3\text{SAT})$ are already quite restricted; they possess only a fixed number of layers. In this section we strengthen the NP-hardness result by constructing even simpler classes of NN for which NNREACH is NP-hard already. Section 4.1 studies the possibility to make these NN structurally as simple as possible; Sect. 4.2 shows that requirements on weights and biases can be relaxed whilst retaining NP-hardness.

4.1 Neural Networks of a Simple Structure

We consider NN with just one hidden layer of ReLU nodes and an output dimension of one. As before, we can establish a reduction from 3SAT.

Theorem 3. NNREACH *is NP-hard for NN with output dimension one, a single hidden layer and simple specifications.*

Proof. Let ψ be a 3SAT formula with n propositional variables X_i and m clauses l_j. We slightly modify the construction of a network N in the proof of Theorem 2. First, we remove the last identity node of all BOOL*-gadgets in N and directly connect the two outputs of their ReLU nodes to the AND-gadget, weighted with 1. Additionally, we merge NOT-gadgets and OR-gadgets in N. Consider the OR-gadget corresponding to some clause l_j. The merged gadget has three inputs $x_{j_0}, x_{j_1}, x_{j_2}$ and computes $\max\big(0, 1 - \sum_{k=0}^{2} f_j(x_{j_k})\big)$ where $f_j(x_{j_k}) = x_{j_k}$ if X_{j_k} occurs positively in l_j and $f_j(x_{j_k}) = 1 - x_{ij}$ if it occurs negatively. It is straightforward to see that the output of such a gadget is 0 if at least one positively (resp. negatively) weighted input is 0, resp. 1, and that the output is 1 if all positively weighted inputs are 1 and all negatively weighted inputs are 0. These merged gadgets are connected with weight -1 to the AND-gadget. Once done for all BOOL*-, NOT- and OR-gadgets, the overall output y of N is given by

$$\sum_{i=0}^{n-1} \max\big(0, \frac{1}{2} - x_i\big) + \max\big(0, x_i - \frac{1}{2}\big) - \sum_{i=0}^{m-1} \max\big(0, 1 - \sum_{j=0}^{2} f_i(x_{ij})\big).$$

Note that N has input dimension n, a single hidden layer of $2n+m$ ReLU nodes and output dimension 1.

Now take the simple specifications $\varphi_{\text{in}} = \bigwedge_{i=0}^{n-1} x_i \geq 0 \wedge x_i \leq 1$ and $\varphi_{\text{out}} = y = \frac{n}{2}$. We argue that the following holds for a solution to $(N, \varphi_{\text{in}}, \varphi_{\text{out}})$: (I) all x_i are either 0 or 1, and (II) the output of each merged OR-gadget is 0. To show (I), we assume the opposite, i.e. there is a solution with $x_k \in (0;1)$ for some k. This implies that $\sum_{i=0}^{n-1} \max\left(0, \frac{1}{2} - x_i\right) + \max\left(0, x_i - \frac{1}{2}\right) < \frac{n}{2}$ as for all $x_i \in [0;1]$ we have $\max\left(0, \frac{1}{2} - x_i\right) + \max\left(0, x_i - \frac{1}{2}\right) \leq \frac{1}{2}$, and for x_k we have $\max\left(0, \frac{1}{2} - x_k\right) + \max\left(0, x_k - \frac{1}{2}\right) < \frac{1}{2}$. Furthermore, we must have $-\sum_{i=0}^{m-1} \max\left(0, 1 - \sum_{j=0}^{2} f(x_{ij})\right) \leq 0$. Therefore, this cannot be a solution for $(N, \varphi_{\text{in}}, \varphi_{\text{out}})$ as it does not satisfy $y = \frac{n}{2}$.

To show (II), assume there is a solution such that one merged OR-gadget outputs a value different from 0. Then, $-\sum_{i=0}^{m-1} \max\left(0, 1 - \sum_{j=0}^{2} f(x_{ij})\right) < 0$ which in combination with (I) yields $y < \frac{n}{2}$. Again, this is a contradiction.

Putting (I) and (II) together, a solution for $(N, \varphi_{\text{in}}, \varphi_{\text{out}})$ implies the existence of a model for ψ. For the opposite direction assume that ψ has a model I. Then, a solution for $(N, \varphi_{\text{in}}, \varphi_{\text{out}})$ is given by $x_i = 1$ if $I(X_i)$ is true and $x_i = 0$ otherwise. □

In the previous section, especially in the arguments of Corollary 1, we pointed out that the occurrence of ReLU nodes is crucial for the NP-hardness of NNReach. Thus, it is tempting to assume that any major restriction to these nodes leads to efficiently solvable classes.

Theorem 4. NNReach *is NP-hard for NN where all ReLU nodes have at most one non-zero weighted input and simple specifications.*

Proof. We prove NP-hardness via a reduction from 3SAT. The reduction works in the same way as in the proof of Theorem 2, but with the following adjustments. We replace the OR-gadgets with simple identity-nodes, we do not include the AND-gadget, and we set the output specification to $\varphi_{\text{out}} = \bigwedge_{i=0}^{n-1} z_i = 0 \wedge \bigwedge_{i=0}^{m} y_i \geq 1$, where y_i is the output of the i-th identity-node replacing the former i-th OR-gadget, z_i is the output of the i-th BOOL-gadget, n is the number of propositional variables and m the number of clauses in the considered 3SAT-instance. Note that this is a simple specification and that the only ReLU nodes in this network are inside the BOOL*-gadgets, which have only one non-zero input. Now, if each $z_i = 0$ then the value of an output y_i is equivalent to the number of inputs equal to 1. The correctness of this reduction is argued int the exaxt same way as in in the original one. □

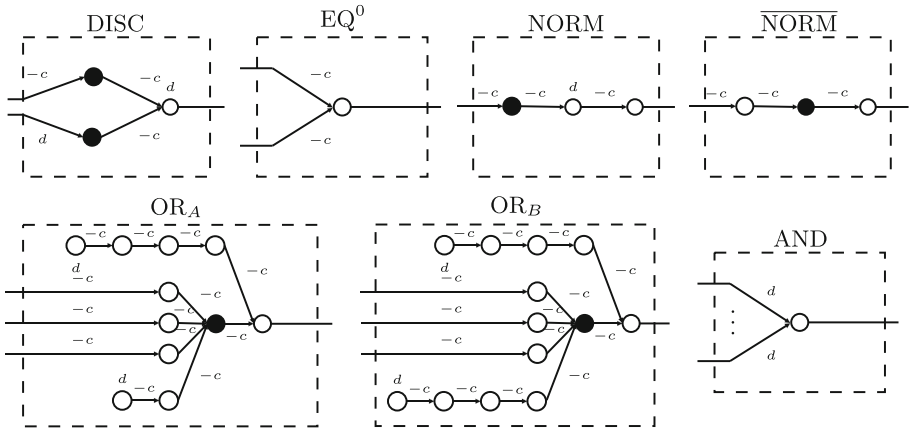

Fig. 4. Gadgets used to show that NNREACH is NP-hard if restricted to $\mathcal{C}(\{-c, 0, d\})$. A non-weighted outgoing edge of a gadget is connected to a weighted incoming one of another gadget in the actual construction or are considered as outputs of the overall neural networks.

4.2 Neural Networks with Simple Parameters

One could argue that the NP-hardness results in Theorem 2 and 3 are only partially applicable to real world problems as the constructed NN use very specific combinations of weights and biases, namely $-1, 0, \frac{1}{2}$ and 1, which may be unlikely to occur in this exact combination in real-world applications. We show that NNREACH is already NP-hard in cases where only very weak assumptions are made on the set of occurring weights and biases.

For $P \subseteq \mathbb{Q}$ let $\mathcal{C}(P)$ be the class of NNREACH instances whose NN only use weights and biases from P and simple specifications. We will show that NP-hardness already occurs when P contains three values: 0, some positive and some negative value. We make use of the same techniques as in Sect. 3 and assume that the general idea of gadgets and the reduction from 3SAT to NNREACH are known.

Definition 6. *Let $c, d \in \mathbb{Q}^{>0}$ and ψ be a* 3SAT*-formula with n propositional variables X_i and m clauses l_j. The network $N_{-c,d,\psi}$ is a network with $2n$ inputs, two for each X_i, called x_i and $\overline{x_i}$. We describe the structure of $N_{-c,d,\psi}$ using the gadgets from Fig. 4:*

- *Each input x_i is connected to both inputs of a* DISC*-gadget and this gadget is connected with weight $-c$ to a chain of five identity nodes interconnected with weight $-c$. We call the output of the last node of this chain z_i.*
- *Each pair x_i and $\overline{x_i}$ is connected to an* EQ^0*-gadget and this gadget is connected with weight $-c$ to a chain of six identity nodes interconnected with weight $-c$. We call the output of the last node of this chain e_i.*
- *Each input x_i is connected to a* NORM*-gadget. Analogously, each $\overline{x_i}$ is connected to a* $\overline{\text{NORM}}$*-gadget.*

- *If $c \geq 1$ (resp. $c < 1$) then there are m OR_A-gadgets (resp. OR_B-gadgets), one for each l_j s.t. if X_i occurs positively in l_j then the output of the* NORM*-gadget connected to x_i is connected and if X_i occurs negatively the output of the* $\overline{\text{NORM}}$*-gadget conntected to $\overline{x_i}$ is connected.*
- *The outputs of all OR_A-gadgets respectively OR_B-gadgets are connected to a single* AND*-gadget. We denote the output of this* AND*-gadget with y.*

Note that each $N_{-c,d,\psi}$ has eight layers and output dimension $2n+1$. Moreover, $N_{-c,d,\psi} \in \mathcal{C}(\{-c, 0, d\})$. Next, we need to clarify some properties of the used gadgets.

Lemma 3. *Let x_0, x_1, x_2 denote inputs for some gadget. The following statements hold:*

1. *If $x_0 = x_1$ then the output of a* DISC*-gadget is 0 if and only if $x_0 = x_1 = -\frac{d}{c^2}$ or $x_0 = x_1 = \frac{1}{c}$.*
2. *If $x_0 = -\frac{d}{c^2}$ then the output of a* NORM*-gadget is 0 and if $x_0 = \frac{1}{c}$ then the output is $-dc$.*
3. *If $x_0 = \frac{d}{c^2}$ then the output of $\overline{\text{NORM}}$-gadget is $-dc$ and if $x_0 = -\frac{1}{c}$ then the output is 0.*
4. *If $x_0 = x_1 = x_2 = 0$ then the output of an OR_A-gadget is $dc^4 - dc^3$. If at least one input is $-dc$ while the others are 0 then the output is dc^4. The same holds for an OR_B-gadget with the difference that if $x_0 = x_1 = x_2 = 0$ then the output is $dc^4 - dc^5$.*

Proof. We start with Property 3.1 and assume that the inputs x_0, x_1 are equal. We can infer from the depiction in Fig. 4 that the output of a DISC-gadget is given by $d - c\max(0, dx_0) - c\max(0, -cx_1)$. At this point we make a case distinction. If $x_0 = x_1 < 0$ then the output is given by $d + c^2 x_1$ and equal to zero if and only if $x_1 = -\frac{d}{c^2}$. If $x_0 = x_1 > 0$ then the output is given by $d - cdx_0$ and equal to zero if and only if $x_0 = \frac{1}{c}$. The last case, namely $x_0 = x_1 = 0$, leads to an output of d.

The Properties 3.2, 3.3 and 3.4 are easily argued. We can infer from Fig. 4 that the output of a NORM-gadget is given by $-c(d - c\max(0, -cx_0))$, the output of a $\overline{\text{NORM}}$-gadget given by $-c\max(0, c^2 x_0)$, the output of an OR-A-gadget given by $dc^4 - c\max(0, dc^2 + c^2 \sum_{i=0}^{2} x_i)$ and the output of an OR-B-gadget given by $dc^4 - c\max(0, dc^4 + c^2 \sum_{i=0}^{2} x_i)$. Then the statements about these gadgets follow by inserting the mentioned values and solving the equations. □

With these properties at hand, we are suited to prove our main statement of this section.

Theorem 5. *Let $c, d \in \mathbb{Q}^{>0}$.* NNREACH *restricted to $\mathcal{C}(\{-c, 0, d\})$ is NP-hard.*

Proof. Let $c, d \in \mathbb{Q}^{>0}$. Take a 3SAT-formula ψ and consider $(N_{-c,d,\psi}, \varphi_{\text{in}}, \varphi_{\text{out}})$ with $N_{-c,d,\psi}$ defined above, $\varphi_{\text{in}} = \top$ and $\varphi_{\text{out}} = \bigwedge_{i=0}^{n-1} z_i = 0 \wedge e_i = 0 \wedge y = m \cdot d^2 c^4$. Obviously, these specifications are simple.

Clearly, $(N_{-c,d,\psi}, \varphi_{\text{in}}, \varphi_{\text{out}})$ can be constructed in time polynomial in the size of ψ. For the correctness of the construction assume that ψ has a model I. We claim that $(N_{-c,d,\psi}, \varphi_{\text{in}}, \varphi_{\text{out}})$ is solved with $x_i = \frac{1}{c}$ if $I(X_i)$ is true, $x_i = -\frac{d}{c^2}$ otherwise, and $\overline{x_i} = -x_i$. Note that φ_{in} is trivially satisfied.

So apply these inputs to $N_{-c,d,\psi}$. According to Lemma 3.1, all outputs z_i are 0. It is easily verified that all outputs e_i are 0 as well. Thus, it is left to argue that $y = m \cdot d^2c^4$. Consider one of the $\text{OR}_{\text{A|B}}$-gadgets occurring in $N_{-c,d,\psi}$, corresponding to a clause l_j. Its inputs are given by the NORM- and $\overline{\text{NORM}}$-gadgets connected to the inputs x_i, resp. $\overline{x_i}$ corresponding to the X_i occurring in l_j. According to Lemma 3.2 and 3 these inputs are either 0 or $-dc$. If l_j is satisfied by I then there is at least one input to the $\text{OR}_{\text{A|B}}$-gadget that is equal to $-dc$. From the fact that ψ is satisfied by I and Lemma 3.4 it follows that each $\text{OR}_{\text{A|B}}$-gadget outputs dc^4. Therefore, the output y of $N_{-c,d,\psi}$ is $m \cdot d^2c^4$. This means that φ_{out} is valid as well.

Consider now the converse direction. A solution for $(N_{-c,d,\psi}, \varphi_{\text{in}}, \varphi_{\text{out}})$ must yield that all x_i are $\frac{1}{c}$ or $-\frac{d}{c^2}$ and $x_i = \overline{x_i}$ as all z_i and e_i have to equal 0. Therefore, all m $\text{OR}_{\text{A|B}}$-gadgets have to output dc^4 as y must equal $m \cdot d^2c^4$. This implies that each $\text{OR}_{\text{A|B}}$-gadget has at least one input that is $-dc$ which in turn means that there is at least one indirectly connected x_i or $\overline{x_i}$ that is $\frac{1}{c}$ resp. $\frac{d}{c^2}$. Thus, ψ is satisfied by setting X_i true if $x_i = \frac{1}{c}$ and false if $x_i = -\frac{d}{c^2}$. □

If $d = c$ and we allow for arbitrary specifications we can show that 0 as a value for weights or biases is unnecessary to keep the lower bound.

Theorem 6. *Let $c \in \mathbb{Q}^{>0}$.* NNREACH *is NP-hard for NN in $\mathcal{C}(\{-c, c\})$ and arbitrary specifications.*

Proof. This is done in the same way as the proof of Theorem 5 with some slight modifications. We only sketch this reduction by describing the differences compared to the instances $(N_{-c,c,\psi}, \varphi_{\text{in}}, \varphi_{\text{out}})$ resulting from the reduction used in Theorem 5.

We do not use EQ^0-gadgets in the network but add for each input x_i the conjunct $x_i = -\overline{x_i}$ to the input specification φ_{in}. This also means that we do not include $\bigwedge_{i=0}^{n-1} e_i = 0$ in the output specification φ_{out}. Consider the weights between the input and the first hidden layer. If the inputs x_i and $\overline{x_i}$ were originally weighted with zero we set the weights corresponding to x_i and $\overline{x_i}$ to be c. In combination with the input constraint $x_i = -\overline{x_i}$ this is equal to weighting x_i and $\overline{x_i}$ with zero. If x_i ($\overline{x_i}$) was originally weighted with c we have to set the weight of $\overline{x_i}$ (x_i) to be $-c$. If it was weighted with $-c$ we have to set the weight of its counterpart to be c. This leads to the case that all non-zero inputs of a node in the first hidden layer are doubled compared to the same inputs in a network $N_{-c,c,\psi}$. Consider now the weights between two layers l and $l+1$ with $l > 0$. For each node in l we add a node in the same layer with the same input weights. If the output of a node in layer l was originally weighted with zero then we weight it with c and the corresponding output of its copy with $-c$. If the output was originally weighted with weight c ($-c$) then we weight the output of the copy node with c ($-c$), too. As before, this doubles the input values at the

nodes in layer $l+1$, which means that compared to a network $N_{-c,c,\psi}$ the output value of our modified network is multiplied by 2^7. Thus, we have to change the output constraint of y to be $y = 2^7(m \cdot c^6)$. Note that these modifications give a network using only the weights $-c$ and c.

To get rid of zero bias, we add the inputs $x_{\text{bias},1}, \overline{x_{\text{bias},1}}, \ldots, x_{\text{bias},7}, \overline{x_{\text{bias},7}}$ to the network and add the input constraints $x_{\text{bias},i} = -\sum_{j=0}^{i-1} \frac{1}{2^{j+1}c^j}$ and $x_{\text{bias},i} = -\overline{x_{\text{bias},i}}$ to φ_{in}. Then, we set the bias of all nodes which originally had a zero bias to be c. For $x_{\text{bias,i}}$ with $i > 1$ we add a chain of $i-1$ identity nodes each with bias c and interconnected with weight c and connect this chain with weight c to $x_{\text{bias},i}$ and $-c$ to $\overline{x_{\text{bias},i}}$. All other weights are assumed to be zero which is realized using the same techniques as described in the previous paragraph. If a node in the first hidden layer originally had a zero bias we weight the input $x_{\text{bias},1}$ with c and $\overline{x_{\text{bias},i}}$ with $-c$. If the input specification holds then the bias plus these inputs sums up to zero. If a node in some layer $l \in \{2, \ldots, 7\}$ originally had a zero bias we weight the output of the last node of the chain corresponding to $x_{\text{bias},l}$ and its copy with c. Again, if the input specification holds, the bias value of this node is nullified. This modification ensures that the network is from $\mathcal{C}(\{-c, c\})$. □

5 Conclusion

We investigated the computational complexity of the reachability problem for NN with ReLU and identity activations. We revised the original proof of its NP-completeness, fixing flaws in both the upper and lower bound, and showed that the parameter driving NP-hardness is the number of ReLU nodes. Furthermore, we showed that NNReach is difficult for very restricted classes of small NN already, respectively that three parameters of different signum occurring as weights and biases suffice for NP-hardness. This indicates that finding non-trivial classes of NN with practical relevance and polynomial NNReach is unlikely.

It remains to be seen whether NP-hardness can be strengthened, for instance for classes of NN with a single hidden layer and a maximum of two non-zero inputs to ReLU nodes, or only one arbitrary positive and only one arbitrary negative weight and bias value. However, possible results here are only of theoretical interest.

From a practical perspective, it would be interesting to see if pure ReLU networks, where every node in a hidden layer has a ReLU activation, lead to similar results as these are more common in practice. Also, investigating the fixed-parameter tractability of the problem more broadly could be promising. It remains to be seen whether there are parameters other than the number of ReLU nodes, like structural properties or dimensionality, whose fixing leads to polynomial decidability. This could yield efficiently solvable classes of NN that are also of practical interest.

References

1. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3–8 December 2018, Montréal, Canada, pp. 4795–4804 (2018). https://proceedings.neurips.cc/paper/2018/hash/be53d253d6bc3258a8160556dda3e9b2-Abstract.html
2. Dixon, M., Klabjan, D., Bang, J.H.: Classification-based financial markets prediction using deep neural networks. Algorithmic Finance **6**(3–4), 67–77 (2017). https://doi.org/10.3233/AF-170176
3. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_19
4. Grigorescu, S.M., Trasnea, B., Cocias, T.T., Macesanu, G.: A survey of deep learning techniques for autonomous driving. J. Field Robot. **37**(3), 362–386 (2020). https://doi.org/10.1002/rob.21918
5. Hinton, G., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012). https://doi.org/10.1109/MSP.2012.2205597
6. Huang, X., et al.: A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability. Comput. Sci. Rev. **37**, 100270 (2020). https://doi.org/10.1016/j.cosrev.2020.100270
7. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Comb. **4**(4), 373–396 (1984). https://doi.org/10.1007/BF02579150
8. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
9. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: a calculus for reasoning about deep neural networks. Form Methods Syst. Des. (2021). https://doi.org/10.1007/s10703-021-00363-7
10. Khan, A., Sohail, A., Zahoora, U., Qureshi, A.S.: A survey of the recent architectures of deep convolutional neural networks. Artif. Intell. Rev. **53**(8), 5455–5516 (2020). https://doi.org/10.1007/s10462-020-09825-6
11. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017). https://doi.org/10.1145/3065386
12. Litjens, G., et al.: A survey on deep learning in medical image analysis. Med. Image Anal. **42**, 60–88 (2017). https://doi.org/10.1016/j.media.2017.07.005
13. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, 2–7 February 2018, pp. 6615–6624. AAAI Press (2018). https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16898

14. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, 13–19 July 2018, Stockholm, Sweden, pp. 2651–2659. ijcai.org (2018). https://doi.org/10.24963/ijcai.2018/368
15. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. CoRR abs/1805.02242 (2018). http://arxiv.org/abs/1805.02242