Reachability in Simple Neural Networks

Marco Sälzer[*]**, Martin Lange**

School of Electr. Eng. and Computer Science

University of Kassel, Germany

{*marco.saelzer, martin.lange*}*@uni-kassel.de*

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 specifications over the input/output dimension given by conjunctions of linear inequalities. We recapitulate the proof and repair some flaws in the original upper and lower bound proofs. Motivated by the general result, we show that NP-hardness already holds for restricted classes of simple specifications and neural networks. Allowing for a single hidden layer and an output dimension of one as well as neural networks with just one negative, zero and one positive weight or bias is sufficient to ensure NP-hardness. Additionally, we give a thorough discussion and outlook of possible extensions for this direction of research on neural network verification.

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 [1] or natural language processing [2] but also safety-critical applications like autonomous driving [3], medical applications [4], or financial matters [5]. 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].

---

[*]Address for correspondence: School of Electr. Eng. and Computer Science, University of Kassel, Germany

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 is mostly concerned with efficient solutions to their reachability problem REACH [7–10]: 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 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 REACH 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 REACH show the same drawback: a lack of scalability to networks of large size which, unfortunately, are typically featured in real-world scenarios. This is not a big surprise as the problem is NP-complete. This result was proposed by Katz et al. [7] for NN with ReLU and identity activations, and later also by Ruan et al. mentioned in [11] and done in [12]. 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 [7] claims inclusion in NP via the standard guess-and-verify approach: guess some valid inputs, pass them through the NN and check whether the resulting numbers are valid outputs. This is flawed, though: it misses the fact that the guessed witnesses need to be polynomially bounded in size, i.e. in the size of the NN and the input and output specifications. No argument is given in [7] for a bound on the representation of such values, let alone a polynomial one.[1] In fact, guessing values in $\mathbb{R}$ is not even effective without a bound on the size of their representation, hence this approach does not even show recursive enumerability of the reachability problem. On the other hand, obtaining a polynomial bound on the values to be guessed is closely linked to the question whether such values can be approximated up to some precision, for which no argument is given in [7] either.

The arguments for the lower bound by a reduction from 3SAT in [7] and [12] 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 REACH which do not correspond to (discrete) satisfying 3SAT assignments. Katz et al. seem to be aware of this in [7] and use a slightly more elaborate discretisation in their reduction, but unfortunately it still suffers from similar problems. These problems are repaired in [13].

We start our investigations into the complexity of REACH by fixing these issues in Section 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 nodes with sophisticated activation functions

---

[1] A follow-up paper by Katz et al. contains an extended version of the original article [13] without this flaw being corrected.

like ReLU. Somehow surprisingly, we obtain the missing polynomial bound for the witnesses used by Katz et al. as a corollary of our observations. Moreover, we show that NP membership is preserved if we allow the use of arbitrary piecewise linear activation functions. We also address the issue of discretisation of real values in the lower bound proof, fixing the construction given by Katz et al. in [7] and presenting an alternative fix compared to [13]. We do not address the one by Ruan et al. from [11] any further, as this does not provide further insights or new results.

We then observe that the reduction from 3SAT constructs a very specific class of neural networks. NN from this class have a fixed number of layers but scaling input and output dimension as well as layer size. This raises the question whether, in comparison to these networks, reducing the number of layers or fixing dimensionality leads to a class of networks for which REACH is efficiently solvable. In Section 4 we show that the answer to this is mostly negative: NP-hardness of REACH holds for NN with just one layer and an output dimension of one. While this provides minimal requirements on the structure of NN for REACH 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 REACH 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.

We conclude in Section 5 with references to possible future work which focuses on further research regarding computational complexity and decidability issues on the reachability problems of NN or deep learning models in a broader sense.

This article is a revised and expanded version of [14], with the following noteworthy differences: proofs and explanations, especially in Section 4, are enriched with helpful explanations and descriptions, the NP membership of REACH is shown for neural networks with arbitrary, piecewise linear activation functions (see Section 3.1), we provide a direct proof for the polynomial bound missing in the arguments of [7] (see Corollary 3.4), it is shown that NP-hardness holds for neural networks using ReLU activations only (see Corollary 3.11) and a thorough outlook on possible further steps is provided (see Section 5).

2.   Preliminaries

A *neural network* (NN) N can be seen as a layered graph, representing 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 $1 \le l \le L-2$ 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 rational constants which are called *weights*, b_{li} is a rational 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. We denote the output of N given $\boldsymbol{x}$ by $N(\boldsymbol{x})$.

We consider neural networks using piecewise linear activations in this work. A function $f : \mathbb{R} \to \mathbb{R}$ is called *piecewise linear (PWL)* if there are $a_0, b_0 \in \mathbb{Q}$ such that $f(x) = a_0 x + b_0$ or there are $m \in \mathbb{N}, m \geq 2$ linear functions $a_i x + b_i$ with $a_i, b_i \in \mathbb{Q}$ and $m - 1$ breakpoints $t_j \in \mathbb{R}$ such that

$$f(x) = \begin{cases} a_0 x + b_0 & \text{if } x < t_0, \\ a_i x + b_i & \text{if } t_i \leq x < t_{i+1} \\ a_{m-1} x + b_{m-1} & \text{if } t_{m-1} \leq x. \end{cases}$$

We denote the i-th linear part of f by f^i. The *ReLU function*, commonly used as an activation in all hidden layers, is defined as $x \mapsto \max(0, x)$ or $x \mapsto 0$ if $x < 0$ and $x \mapsto x$ otherwise. The second definition makes clear that ReLU is a PWL function. Obviously, the same holds for the identity function. In order to make our results about lower bounds as strong as possible, we focus on neural networks using ReLU (and identity as a shortcut) activations only. Nodes with ReLU (identity) activation are called ReLU (identity) nodes. 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 or equal, resp. less than 0.

We make use of two ways to represent NN in this work. The first one is visualizing an NN as a directed graph with weighted edges. An example is given in Figure 1. The second one is referring to a NN N by its computed function f_N. Obviously, this second way is ambiguous as there are infinitely many NN that compute f_N. To keep things easy, we assume that it is clear in the respective situation which NN is referred to.
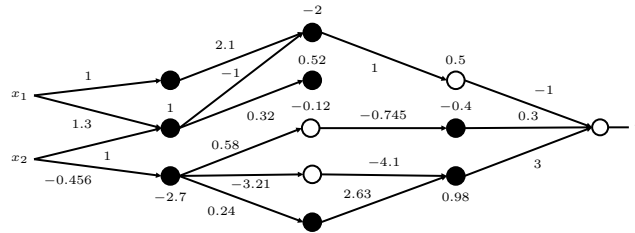

Figure 1.   Schema of a neural network with five layers, input dimension of two and output dimension of one. Filled nodes are ReLU nodes, empty nodes are identity nodes. An edge between two nodes u and v with label w denotes that the output of u is weighted with w in the computation of v. No edge between 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.

Our main interest lies in the validity of specifications over the output values of NN given specifications over their input values. 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. A 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*, respectively *output specification* and assume that the set of variables occurring in φ is a subset of the input respectively output variables of N.

The decision problem REACH is defined as follows: given a neural network N with PWL activations, input specification $\varphi_{\text{in}}(x_0, \ldots, x_{n-1})$ and output specification $\varphi_{\text{out}}(y_0, \ldots, y_{m-1})$, is there $\boldsymbol{x} \in \mathbb{R}^n$ such that $\varphi_{\text{in}}(\boldsymbol{x})$ and $\varphi_{\text{out}}(N(\boldsymbol{x}))$ are true?

3. REACH is NP-complete

3.1. Membership in NP

The argument used by Katz et al. to show membership of REACH 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 $(N, \varphi_{\text{in}}, \varphi_{\text{out}})$, which is not provided in [7]. We first present an alternative proof for REACH $\in$ NP, circumventing the missing argument of a polynomial bound for the size of satisfying inputs, by establishing a connection between NN with PWL activations and an extended notion of linear programs. Note that this subsumes the claim of Katz et al., as ReLU and identity are PWL activations. Furthermore, this proof directly implies the polynomial bound in question, which we will see at the end of this section.

Definition 3.1. A *PWL-linear program* over a set $X = \{x_0, \ldots, x_{n-1}\}$ of variables is a set Φ of (in-)equalities of the form $b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji} \leq x_j$ and $f(b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji}) = x_j$ where $x_{ji}, x_j \in X$, $c_{ji}, b_j \in \mathbb{Q}$ and f is a PWL function. We call the second kind a *PWL-equality*. 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 PWL-equality $f(b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji}) = x_j$, where f has m linear parts $a_l x + b_l$ and $m - 1$ breakpoints t_l, is satisfied by $\boldsymbol{x}$ with $t_l \leq b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji} < t_{l+1}$ if $a_l(b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji}) + b_l = x_j$. The problem of *solving* a PWL-linear program is: given Φ, decide whether there is a solution to it.

It is well-known that regular linear programming can be done in polynomial time [15]. We use this to show that solving PWL-linear programs can be done in nondeterministic polynomial time.

Lemma 3.2. The problem of solving a PWL-linear program is in NP.

Proof:
Suppose a PWL-linear program Φ with l PWL-equalities is given. First, note that there is a maximum number k of linear parts a PWL function f occurring in Φ consists of. Then, the existence of a solution can be decided as follows. Guess, for each PWL-equality χ_h of the form $f_h(b_j + \sum_{i=1}^{m} c_{ji} \cdot x_{ji}) = x_j$

where f_h has $k' \leq k$ parts, some $w_h \in \{0, \ldots, k'-1\}$, leading to the witness $\boldsymbol{w} = (w_0, \ldots, w_{l-1})$. It is not hard to see that its size is polynomially bounded by the size of Φ. Next, let the linear program $\Phi_{\boldsymbol{w}}$ result from Φ by replacing each χ_h by the (in-)equalities $b_j + \sum_{i=1}^m c_{ji} \cdot x_{ji} \geq t_{w_h}$, $b_j + \sum_{i=1}^m c_{ji} \cdot x_{ji} \leq t_{w_h+1} + z_h$ and $a_{w_h}(b_j + \sum_{i=1}^m c_{ji} \cdot x_{ji}) + b_{w_h} = x_j$ where z_h is a fresh variable.

The following is straightforward: (I) $\Phi_{\boldsymbol{w}}$ is linear in the size of Φ. (II) An optimal solution for the minimization of $\boldsymbol{z} = (z_1, \ldots, z_l)$ over the constraints given by $\Phi_{\boldsymbol{w}}$ where all z_i are strictly negative is a witness for a solution of Φ. (III) If Φ has a solution, then there is $\boldsymbol{w} \in \{0, \ldots k-1\}^l$ such that $\Phi_{\boldsymbol{w}}$ has a solution. This can be created as follows. Let $\boldsymbol{x}$ be a solution to Φ. For each PWL-equality χ_h as above, let $w_h = j$ if the active linear part is the j-th. Then $\boldsymbol{x}$ is also a solution for $\Phi_{\boldsymbol{w}}$. Thus, PWL-linear programs can be solved in nondeterministic polynomial time by guessing $\boldsymbol{w}$, constructing the linear program $\Phi_{\boldsymbol{w}}$, finding the optimal solution minimizing $\boldsymbol{z}$ and checking if all z_i are negative.   □

Using the definition of a PWL-linear program and the corresponding lemma at hand, we are set to prove NP-membership of REACH.

Theorem 3.3. REACH is in NP.

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

- φ_{in} and φ_{out} (with each conjunct seen as one (in-)equality) and

- for each node v_{li} computing the function $f(\sum_j c_{ji}^{(l-1)} y_{(l-1)j} + b_{li})$ it contains the PWL-equality $f(\sum_j c_{ji}^{(l-1)} y_{(l-1)j} + b_{li}) = y_{li}$.

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, REACH polynomially reduces to the problem of solving PWL-linear programs which, by Lemma 3.2 is in NP.   □

Taking a look at our translation from REACH to PWL-linear programs, one could think that Theorem 3.3 can also be proven by translating REACH instances into *mixed-integer linear programs* (MILP) [16]. Roughly speaking, an MILP is a LP with additional integer variables. Deciding feasability of MILP is known to be NP-complete [17]. Translating REACH instance into MILP works similarly as described in the proof of Theorem 3.3 up to the point of PWL nodes. For example, consider a node v_j computing the function $ReLU(b_j + \sum_{i=1}^m c_{ji} \cdot x_{ji})$. First, we add the (in-)equalities $b_j + \sum_{i=1}^m c_{ji} \cdot x_{ji} = y_j - s_j$, $y_j \geq 0$ and $s_j \geq 0$ to our MILP. Second, we add the inequalities $y_j \leq M_j^+(1 - z_j)$ and $s_j \leq M_j^- z_j$ where z_j is an integer variable and M_j^+, M_j^- are sufficiently large positive constants such that $-M_j^- \leq b_j + \sum_{i=1}^m c_{ji} \cdot x_{ji} \leq M_j^+$ holds. This ensures that $z_j = 0$ corresponds to v_j being active and $z_j = 1$ corresponds to v_j being inactive. For general PWL activation

functions the idea is the same, making use of switch variables like z_j and bounds like M^-, M^+ in a more elaborate fashion. However, this translation requires that suitable bounds exist for each node, which in general is only guaranteed if each input dimension is bounded by the input specification.

As stated in the beginning, our proof implies the polynomial bound missing in the claim of Katz et al. We recall the missing argument in [7]: it is not obvious, that there is an input $\boldsymbol{x}$ for each positive REACH instance, witnessing the validity I, which is polynomial bounded by the size of I. Taking a look at the proof of Lemma 3.2, we see that the solution of a PWL-linear program Φ is determined by the solution of a regular linear program $\Phi_{\boldsymbol{w}}$ linear in the size of Φ. Furthermore, from the definition of neural networks and specifications follows that all coefficients occurring in $\Phi_{\boldsymbol{w}}$ are rational. Then, we can use a standard result about linear programs: if a linear program L with rational coefficients has an optimal solution then it has an optimal one of size polynomially bounded by L (for example, see Theorem 4.4 in [18]). Thus, our proof of Lemma 3.2 implies that each PWL-linear program, resulting from the translation of a REACH instance, has a solution of polynomial size and, thus, using the arguments of the proof of Theorem 3.3, we retrieve the missing bound.

Corollary 3.4. Let $I = (N, \varphi_{\text{in}}, \varphi_{\text{out}})$ be a REACH instance. If I is a positive instance of REACH, then there is an input $\boldsymbol{x}$ satisfying φ_{in} such that $N(\boldsymbol{x})$ satisfies φ_{out} and the size of $\boldsymbol{x}$ is polynomially bounded by the size of I.

Additionally, it is interesting to point out the role of witnesses for positive instances of the REACH problem: the arguments leading to the result of Theorem 3.3 above show that an assignment to the PWL nodes determining their active linear part serves as a witness. This immediately yields a polynomial fragment of REACH.

Corollary 3.5. REACH for NN with a bounded number of nodes with PWL activation is decidable in polynomial time.

3.2. NP-hardness

A natural candidate for a lower bound proof for REACH is a polynomial reduction from 3SAT, as attempted by Katz et al. [7] and Ruan et al. [11]. 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. To ease the definition of the NN resulting from this reduction Katz et al. introduced the notion of gadgets.

Definition 3.6. Let $\varepsilon \in \mathbb{R}$ be a small constant and $n \in \mathbb{N}$. We call NN, which compute one of the following functions, *gadgets*:

$$not(x) = id(1 - x)$$
$$or(x_1, x_2, x_3) = id(1 - ReLU(1 - x_1 - x_2 - x_3))$$
$$and_n(x_1, \ldots, x_n) = id(x_1 + \cdots + x_n)$$
$$bool_\varepsilon(x) = id(ReLU(\varepsilon - x) + ReLU(x + \varepsilon - 1))$$
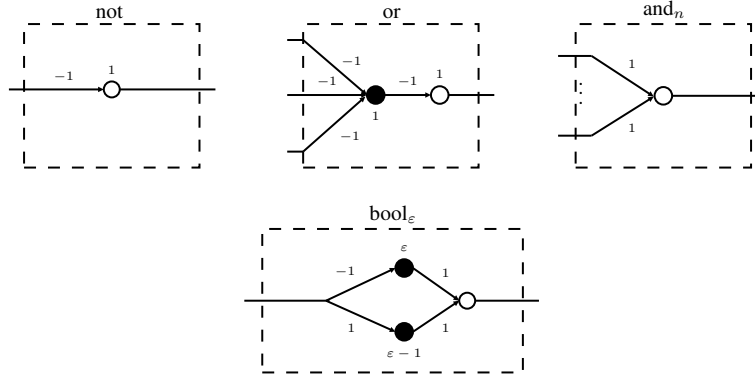
Figure 2.   Gadgets used in the reduction from 3SAT to REACH. A non-weighted outgoing edge of a gadget is connected to a weighted incoming edge of another gadget in the actual construction or is considered an output of the overall neural network.

The obvious candidates for these gadgets are visualized in Figure 2. The name of the first three gadgets gives a direct indication of their intended properties.

Lemma 3.7. Let $n \in \mathbb{N}$ be a natural number and all $r_i \in \{0, 1\}$. It holds that

1. $not(1) = 0$ and $not(0) = 1$,

2. $or(0, 0, 0) = 0$,

3. $or(r_1, r_2, r_3) = 1$ if and only if at least one r_i is equal to 1,

4. $and_n(r_1, \ldots, r_n) = n$ if and only if all r_i are equal to 1.

Proof:
These are straightforward implications of the functions computed by the respective gadgets.   □

To understand how the reduction works, we take the 3SAT instance $\psi = (X_0 \vee X_1 \vee X_2) \wedge (\neg X_0 \vee X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$ with four propositional variables and three clauses as an example. Furthermore, for the moment we assume that NN can take values from $\{0, 1\}$ as inputs only. Then, the resulting network $N_\psi : \mathbb{R}^4 \to \mathbb{R}$ computes

$$and_3(or(x_0, x_1, x_2), or(not(x_0), x_1, not(x_2)), or(not(x_1), x_2, x_3))$$

and is visualized in Figure 3. Note that N_ψ includes redundant identity nodes matching included *not*-gadgets in order to ensure a layerwise structure. The function computed by N_ψ is described as follows. Each of the three *or*-gadgets together with their connected *not*-gadgets and *id*-nodes represents one of the clauses in ψ. From case (3) in Lemma 3.7 we obtain that, if an *or*-gadget outputs 1, then its current input, viewed as an assignment to the propositional variables in ψ, satisfies the corresponding clause. The and_3-gadget simply sums up all of its inputs and, with case (4) of Lemma 3.7, we get that y is

equal to 3 iff each *or*-gadget outputs one. Therefore, with the output specification $\varphi_{out}(y) := y = 3$ and trivial input specification $\varphi_{in} = \top$, we get a reduction from 3SAT to REACH, provided that input values are externally restricted to $\{0, 1\}$.
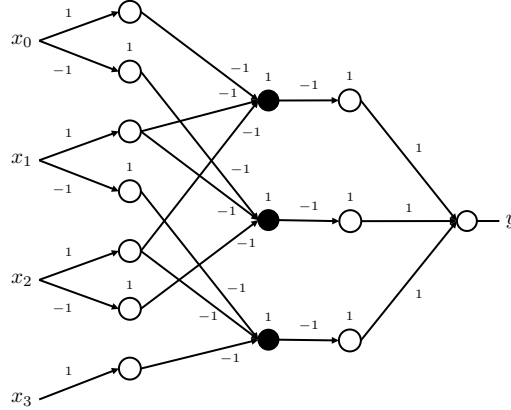

Figure 3. NN resulting from the reduction of the 3SAT-formula $(X_0 \vee X_1 \vee X_2) \wedge (\neg X_0 \vee X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$ under the assumption that NN are defined over $\{0, 1\}$ only.

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 $\varphi_{in}(\boldsymbol{x})$ which is satisfied by $\boldsymbol{x}$ iff $\boldsymbol{x} \in \{0, 1\}^n$ because $\{0, 1\}^n$ is not a hyperrectangle in $\mathbb{R}^n$ and conjunctions of inequalities only specify hyperrectangles. This is where we make use of $bool_\varepsilon$-gadgets. 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 $bool_\varepsilon$-gadget to each input x_i in N_ψ including adding further identity nodes for ensuring a layerwise structure and using the simple specifications $\varphi_{in} := \bigwedge_{i=0}^{3} x_i \geq 0 \wedge x_i \leq 1$ and $\varphi_{out} := \bigwedge_{i=0}^{3} z_i \geq 0 \wedge z_i \leq \varepsilon \wedge y \geq 3(1 - \varepsilon) \wedge y \leq 3$ 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_ψ, determined by the $bool_\varepsilon$-gadgets and their output constraints, are not exactly 0 or exactly 1.

However, the claim about $bool_\varepsilon$-gadgets is wrong. Consider a $bool_\varepsilon$-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 $bool_\varepsilon$-gadgets we can make the reduction work.

Lemma 3.8. Let $bool(x) = id\left(ReLU\left(\frac{1}{2} - x\right) + ReLU\left(x - \frac{1}{2}\right) - \frac{1}{2}\right)$ be a gadget. It holds that $bool(x) = 0$ if and only if $x = 0$ or $x = 1$.

Proof:
Note that the function $bool(x)$ is equivalent to

$$bool(x) = \begin{cases} -x & \text{if } x < \frac{1}{2} \\ x - 1 & \text{otherwise.} \end{cases}$$

From this we immediately get that $bool(x) = 0$ if $x = 0$ or $x = 1$, and $bool(x) \neq 0$ for all other values of x. $\qquad \square$

Remember that id and $ReLU$ are PWL functions. Now, replacing all $bool_\varepsilon$-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 REACH.

Theorem 3.9. REACH is NP-hard.

One could argue that a network N resulting from the reduction of 3SAT is not a typical feed-forward neural network. The reason is that it is common to use the same activation function across all layers , which is not the case for NN resulting from this reduction as they include both ReLU- and identity nodes in hidden layers. However, we can modify the reduction with the help of the following lemma.

Lemma 3.10. Let N be a NN and v be an identity node in N computing $\sum_{i=1}^{k} c_i x_i + b$ and which is connected with weights $c'_1, \dots, c'_l$ to nodes in a following layer of N. We can replace v with two ReLU nodes v' and v'' such that the resulting network N' computes the same function as N.

Proof:
Let N and v be as stated above. Then, v' computes $ReLU(\sum_{i=1}^{k} c_i x_i + b)$ and v'' computes the function $ReLU(\sum_{i=1}^{k} -c_i x_i - b)$. Furthermore, v' is connected with weights $c'_1, \dots, c'_l$ to the same nodes as v and v'' with weights $-c'_1, \dots, -c'_l$. It is straightforward that the resulting network N' still computes the same function. $\qquad \square$

We can use Lemma 3.10 to adjust the gadgets of Figure 2, resulting in gadgets using ReLU-activations only, leading to an even stronger lower bound.

Corollary 3.11. REACH is NP-hard if restricted to instances with networks using ReLU activations only.

4. NP-hardness holds in very restricted cases already

Neural networks and specifications resulting from the reduction of 3SAT to REACH presented in the previous section are already quite restricted; the NN possess only a small, fixed number of layers and output size of one and the specifications are simple. In this section we strengthen the NP-hardness result by constructing even simpler classes of instances for which REACH is NP-hard already. Our main focus is on simplifying the structure of the NN rather than the specifications. However, when trying to simplify the networks, it is tempting to increase complexity in the specifications. To avoid this, we aim for keeping the specifications simple.

Section 4.1 studies the possibility to make the 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. In the same way as in the previous section, we can establish a reduction from 3SAT.

Theorem 4.1. REACH 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 3.9. 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_n-gadget, weighted with 1. Additionally, we merge *not*-gadgets and *or*-gadgets in N. Consider an *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 $ReLU\left(1 - \sum_{k=0}^{2} f_j(x_{j_k})\right)$ 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_n-gadget. Once done for all *bool*-, *not*- and *or*-gadgets, the resulting network N' computes the function

$$id\left(\sum_{i=0}^{n-1} ReLU\left(\tfrac{1}{2} - x_i\right) + ReLU\left(x_i - \tfrac{1}{2}\right) - \sum_{i=0}^{m-1} ReLU\left(1 - \sum_{j=0}^{2} f_i(x_{ij})\right)\right).$$

Note that N' has input dimension n, a single hidden layer of $2n+m$ ReLU nodes and output dimension 1 which we refer to by y.

Now take the simple specifications $\varphi_{in} = \bigwedge_{i=0}^{n-1} x_i \geq 0 \wedge x_i \leq 1$ and $\varphi_{out} = y = \tfrac{n}{2}$. We argue that the following holds for a solution to $(N', \varphi_{in}, \varphi_{out})$: (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 $\sum_{i=0}^{n-1} ReLU\left(0, \tfrac{1}{2} - x_i\right) + ReLU\left(0, x_i - \tfrac{1}{2}\right) < \tfrac{n}{2}$ as for all $x_i \in [0;1]$ we have $ReLU\left(0, \tfrac{1}{2} - x_i\right) + ReLU\left(0, x_i - \tfrac{1}{2}\right) \leq \tfrac{1}{2}$, and for x_k we have $ReLU\left(0, \tfrac{1}{2} - x_k\right) + ReLU\left(0, x_k - \tfrac{1}{2}\right) < \tfrac{1}{2}$. Furthermore, we must have $-\sum_{i=0}^{m-1} ReLU\left(0, 1 - \sum_{j=0}^{2} f(x_{ij})\right) \leq 0$. Therefore, this cannot be a solution for $(N', \varphi_{in}, \varphi_{out})$ as it does not satisfy $y = \tfrac{n}{2}$.

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

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

We can use Lemma 3.10 to argue that the same holds for pure ReLU-networks. The idea here is that replacing identity- with ReLU nodes does increase the layer sizes of a NN but not the depth of it.

In the previous section, especially in the arguments of Corollary 3.5, we pointed out that the occurrence of ReLU nodes is crucial for the NP-hardness of REACH. Thus, it is tempting to assume that any major restriction to these nodes leads to efficiently solvable classes.

Theorem 4.2. REACH 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. Let ψ be a 3SAT formula with n propositional variables X_i and m clauses l_j. The reduction works in the same way as in the proof of Theorem 3.9, but we use plain identity-nodes instead of or-gadgets and we do not include an and_n-gadget. The resulting network $N_\psi : \mathbb{R}^n \to \mathbb{R}^{m+1}$ computes the function

$$\left(id\left(\sum_{i=1}^{3} f_1(x_{1i}) \right), \ldots, id\left(\sum_{i=1}^{3} f_m(x_{mi}) \right), id\left(\sum_{i=1}^{n} bool(x_i) \right) \right),$$

where $f_j(x)$ is either $id(x)$ or $not(x)$, determined by the j-th clause of ψ. Furthermore, the input specification remains $\varphi_{\text{in}} = \top$ and we set the output specification to $\varphi_{\text{out}} = \bigwedge_{j=1}^{m} y_j \geq 1 \land z = 0$, where y_j is the output of the j-th identity-node corresponding to the j-th clause of ψ and z is the output corresponding to the sum of all $bool$-gadgets. First, note that the only ReLU nodes occurring in N_ψ are inside the $bool$-gadgets, which according to Lemma 3.8 have only one non-zero input. Second, note that the specifications φ_{in} and φ_{out} are simple. Now, if $z = 0$ then the value of an output y_j is equivalent to the number of inputs equal to 1. The remaining argument for the correctness of this reduction is the same as in the original NP-hardness proof.   □

This result does not hold for the case of pure ReLU networks as the resulting networks include identity-nodes. Additionally, we cannot make use of Lemma 3.10 here as it would result in NN including ReLU nodes with at least three inputs. However, we can derive a slightly weaker result for the pure ReLU case by modifying the reduction again.

Theorem 4.3. REACH is NP-hard for NN without identity-nodes and where all nodes have at most two non-zero weighted inputs and simple specifications.

Proof:
The proof works similar to the original NP-hardness argument but we use adjusted or-gadgets. Such a modified gadget $or' : \mathbb{R}^3 \to \mathbb{R}$ computes the function $or'(x_1, x_2, x_3) = ReLU(1 - ReLU(ReLU(1 - x_1 - x_2) - x_3))$. It is not hard to see that an or'-gadget fullfills the same property as an or-gadget, namely that for $x_i \in \{0, 1\}$ its output is 1 if at least one x_i is 1 otherwise its 0. Furthermore, we can see that this function can be represented by a gadget where all ReLU nodes have two non-zero inputs only. The remaining identity-nodes in other gadgets can safely be eliminated using Lemma 3.10.   □

4.2. Neural Networks with simple parameters

One could argue that the NP-hardness results of Sections 3 and 4.1 are only of limited relevance 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. To conquer such concerns, we show that REACH 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 $NN(P)$ be the class of NN which only use weights and biases from P. Our main goal is to 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 Section 3 and start with defining further but similar gadgets.

Definition 4.4. Let $c, d \in \mathbb{Q}^{>0}$. We call NN, which compute one of the following functions, *gadgets*:

$$disc(x) = id\,(d - c\,ReLU\,(-cx) - c\,ReLU\,(dx))$$
$$eq_0(x_1, x_2) = id(-cx_1 - cx_2)$$
$$norm(x) = id\,(-c\,id\,(d - c\,ReLU(-cx)))$$
$$\overline{norm}(x) = id\,(-c\,ReLU\,(-c\,id(-cx)))$$
$$or_{<1}(x_1, x_2, x_3) = id\big([d \cdot c^4] - c\,ReLU\big([d \cdot c^4] + \textstyle\sum_{i=1}^{3} -cx_i\big)\big)$$
$$or_{\geq 1}(x_1, x_2, x_3) = id\big([d \cdot c^4] - c\,ReLU\big([d \cdot c^2] + \textstyle\sum_{i=1}^{3} -cx_i\big)\big)$$
$$and_n(x_1, \ldots, x_n) = id(dx_1 + \cdots + dx_n)$$

where $[d \cdot c^n]$ with even n is an abbreviation for $\underbrace{-c \cdot id(-c \cdot id(\cdots (-c \cdot id(d)) \cdots))}_{n}$.

The natural candidates for these gadgets are visualized in Figure 4. Similar to the gadgets defined in Section 3 each of the above ones fullfills a specific purpose.
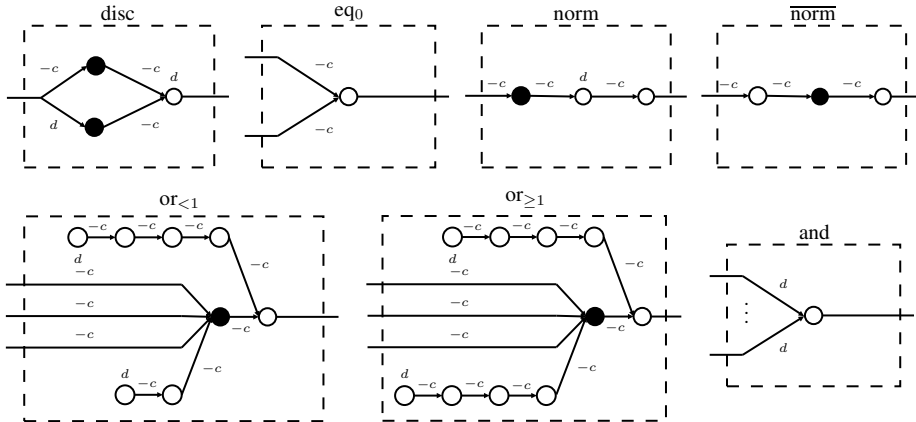

Figure 4. Gadgets used to show that REACH is NP-hard if restricted to NN from $NN\,(\{-c, 0, d\})$. A non-weighted outgoing edge of a gadget is connected to a weighted incoming one of another gadget in the actual construction or are considered as outputs of the overall neural networks.

Lemma 4.5. Let $c, d \in \mathbb{Q}^{>0}$. The following statements hold:

1. $disc(r) = 0$ if and only if $r = -\frac{d}{c^2}$ or $r = \frac{1}{c}$.

2. $eq_0(r_1, r_2) = 0$ if and only if $r_1 = -r_2$.

3. $norm(-\frac{d}{c^2}) = 0$ and $norm(\frac{1}{c}) = -dc$.

4. $\overline{norm}(\frac{d}{c^2}) = -dc$ and $\overline{norm}(-\frac{1}{c}) = 0$.

5. $or_{<1}(0,0,0) = dc^4 - dc^5$ and $or_{\geq 1}(0,0,0) = dc^4 - dc^3$

6. If all $r_i \in \{-dc, 0\}$ and $c < 1$ then $or_{<1}(r_1, r_2, r_3) = dc^4$ if at least one $r_i = -dc$.

7. If all $r_i \in \{-dc, 0\}$ and $c \geq 1$ then $or_{\geq 1}(r_1, r_2, r_3) = dc^4$ if at least one $r_i = -dc$.

Proof:
We start with (1) and make a case distinction. If $r \leq 0$ then $disc(r) = d + c^2 r$. It is easy to see that this becomes zero if and only if $r = -\frac{d}{c^2}$. The next case is $r = 0$ which leads to $disc(0) = d$. And for the last case $r > 0$ we get that $disc(r) = d - cdr$. Again, it is easy to see that this becomes zero if and only if $r = \frac{1}{c}$.

The remaining properties are straightforward implications of the function computed by the respective gadgets. $\qquad \square$

From cases (6) and (7) of Lemma 4.5 we can see that the properties of $or_{\geq 1}$ and $or_{<1}$ are the same, the only difference is that $or_{\geq 1}$ fullfills it for the case that $c \geq 1$ and $or_{<1}$ fullfills it in the $c < 1$ case.

Let ψ be some 3SAT-formula. We construct a network $N_{\psi, c, d}$, using the gadgets above, which is later used in a reduction from 3SAT to REACH restricted to NN using only $-c, 0$ and d as weights or bias.

Definition 4.6. Let ψ be a 3SAT-formula with n propositional variables X_i and m clauses, $c, d \in \mathbb{Q}^{>0}$ and $i \in \{1, \ldots, n\}$. The network $N_{\psi, c, d}$ computes a function $\mathbb{R}^{2n} \rightarrow \mathbb{R}^{2n+1}$ with two input dimensions x_i and $\overline{x_i}$ per propositional variable X_i. We describe the structure of $N_{\psi, c, d}$ by defining the function computed for each output dimension:

$$z_i = id(-c\, id(-c\, id(-c\, id(-c\, id(-c\, disc(x_i))))))$$
$$e_i = id(-c\, id(-c\, id(-c\, id(-c\, id(-c\, id(-c\, eq_0(x_i, \overline{x_i})))))))$$
$$y = \begin{cases} and_m\big(or_{\geq 1}(id(f_{i_1^1}), id(f_{i_2^1}), id(f_{i_3^1})), \ldots, or_{\geq 1}(id(f_{i_1^m}), id(f_{i_2^m}), id(f_{i_3^m}))\big) & \text{, if } c \geq 1 \\ and_m\big(or_{<1}(id(f_{i_1^1}), id(f_{i_2^1}), id(f_{i_3^1})), \ldots, or_{<1}(id(f_{i_1^m}), id(f_{i_2^m}), id(f_{i_3^m}))\big) & \text{, otherwise,} \end{cases}$$

where $f_{i_k^j} = norm(x_i)$ if the variable X_i is at the k-th position in the j-th clause and occurs positively and $f_{i_k^j} = \overline{norm}(\overline{x_i})$ if it occurs negatively.

Note that the redundant identity nodes ensure a layerwise structure and that each such network $N_{\psi, c, d}$ has eight layers. Moreover, from the definition of the gadgets and the construction above follows that $N_{\psi, c, d} \in NN(\{-c, 0, d\})$.

With this definition at hand, we are suited to prove our main statement of this section. To keep the proof clear and w.l.o.g., we do not distinguish $or_{\geq 1}$- and $or_{<1}$-gadgets and simply call them or-gadgets.

Theorem 4.7. Let $c, d \in \mathbb{Q}^{>0}$. REACH restricted to neural networks from $NN(\{-c, 0, d\})$ and simple specifications is NP-hard.

Proof:
We prove the result above via reduction from 3SAT. Take a 3SAT-formula ψ with n propositional variables X_i and m clauses l_j and consider an REACH-instance $(N_{\psi,c,d}, \varphi_{\text{in}}, \varphi_{\text{out}})$ with $N_{\psi,c,d}$ defined above, $\varphi_{\text{in}} = \top$ and $\varphi_{\text{out}} = \bigwedge_{i=1}^{n} z_i = 0 \wedge e_i = 0 \wedge y = m \cdot d^2 c^4$. Obviously, these specifications are simple and clearly $(N_{\psi,c,d}, \varphi_{\text{in}}, \varphi_{\text{out}})$ can be constructed in polynomial time in the size of ψ.

For the correctness of the construction we start with assuming that ψ has a model I. We claim that a solution for $(N_{\psi,c,d}, \varphi_{\text{in}}, \varphi_{\text{out}})$ is given by $x_i = \frac{1}{c}$ 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 by this assignment. So apply these inputs to $N_{\psi,c,d}$. According to cases (1) and (2) of Lemma 4.5, all outputs z_i and e_i are 0. Thus, it is left to argue that $y = m \cdot d^2 c^4$. Consider one of the or-gadgets occurring in $N_{\psi,c,d}$, corresponding to a clause l_j. Its inputs are given by the $norm$- and $\overline{norm}$-gadgets connected to the inputs x_i, resp. $\overline{x_i}$ corresponding to the X_i occurring in l_j. According to cases (3) and (4) of Lemma 4.5, these inputs are either 0 or $-dc$. If l_j is satisfied by I then there is at least one input to the or-gadget that is equal to $-dc$. From the fact that ψ is satisfied by I and the cases (6) and (7) of Lemma 4.5 we get that each or-gadget outputs dc^4. Therefore, the output y of $N_{\psi,c,d}$ is $m \cdot d^2 c^4$. This means that φ_{out} is valid as well.

Consider now the converse direction. A solution for $(N_{\psi,c,d}, \varphi_{\text{in}}, \varphi_{\text{out}})$ 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 the output of each $norm$- and $\overline{norm}$ in $N_{\psi,c,d}$ is either $-dc$ or 0. This implies that all m or-gadgets have to output dc^4 as y must equal $m \cdot d^2 c^4$. According to cases (5), (6) and (7) of Lemma 4.5, each or-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 not required to keep the lower bound of REACH.

Theorem 4.8. Let $c \in \mathbb{Q}^{>0}$. REACH is NP-hard restricted to neural networks from $NN(\{-c, c\})$ and arbitrary specifications.

Proof:
This is done in the same way as the proof of Theorem 4.7 with some slight modifications to the resulting NN and specifications. We only sketch this reduction by describing the differences compared to the instances $(N_{\psi,c,c}, \varphi_{\text{in}}, \varphi_{\text{out}})$ resulting from the reduction used in Theorem 4.7. Our goal is to eliminate all zero weights and bias in $N_{\psi,c,c}$ while ensuring that the modified network $N'_{\psi,c,c} \in NN(\{-c, c\})$ has the same functional properties. Therefore, we split the argument into two parts: first, we argue how to eliminate zero weights, and second, we argue how to eliminate zero bias.

To get rid of zero weights, we add for each x_i the conjunct $x_i = -\overline{x_i}$ to the input specification. As a side effect, this makes the usage of eq_0-gadgets obsolete, though we do not include them and the following chain of id-nodes and we do not include $\bigwedge_{i=0}^{n-1} e_i = 0$ in the output specification as well. Now, consider the weights between the input and the first hidden layer of $N'_{\psi,c,c}$. If the inputs

x_i and $\overline{x_i}$ were originally weighted with zero in some node in the first hidden layer of $N_{\psi,c,c}$, we set the weights corresponding to x_i and $\overline{x_i}$ to be c in $N'_{\psi,c,c}$. 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 weighted with c in $N_{\psi,c,c}$, we set the weight of $\overline{x_i}$ (x_i) to be $-c$ and if it was weighted with $-c$, we set the weight of its counterpart to be c in $N'_{\psi,c,c}$. 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 $N_{\psi,c,c}$. Consider now the weights between two layers l and $l+1$ with $l > 0$ of $N_{\psi,c,c}$. For each node in l we add an additional node in the layer l of $N'_{\psi,c,c}$ with the same input weights. If the output of a node in layer l was originally weighted with zero in $N_{\psi,c,c}$ then we weight it with c and the corresponding output of its copy with $-c$ in $N'_{\psi,c,c}$. If the output was originally weighted with weight c ($-c$) in $N_{\psi,c,c}$ then we weight the output of the copy node with c ($-c$) $N'_{\psi,c,c}$, too. As before, this doubles the input values at the nodes in layer $l+1$ of $N'_{\psi,c,c}$ compared to $N_{\psi,c,c}$, which means that in comparison to $N_{\psi,c,c}$ the output value of $N'_{\psi,c,c}$ is multiplied by 2^7. Thus, we have to change the output constraint of y to be $y = 2^7(m \cdot c^6)$.

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 $N'_{\psi,c,c}$ and the 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 the input specification. Next, 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 incoming weights of this chain are assumed to be zero which is realized using the same techniques as described in the previous paragraph. Now, if a node in the first hidden layer of $N_{\psi,c,c}$ has a zero bias in $N_{\psi,c,c}$, we give weight c to the input $x_{\text{bias},1}$ and $-c$ to $\overline{x_{\text{bias},1}}$ in $N'_{\psi,c,c}$. If the input specification holds then the bias of the node plus these inputs sum up to zero. If a node in some layer $l \in \{2, \ldots, 7\}$ has a zero bias in $N_{\psi,c,c}$, we give weight c to the output of the last node of the chain corresponding to $x_{\text{bias},l}$ and its copy, resulting from the zero weight elimination of the previous paragraph. Again, if the input specification holds, the bias value of this node is nullified.

In summary, we get the specifications $\varphi'_{\text{in}} = \bigwedge_{i=1}^{n} x_i = \overline{x_i} \wedge \bigwedge_{i=1}^{7} x_{\text{bias},i} = -\sum_{j=0}^{i-1} \frac{1}{2^{j+1}c^j} \wedge x_{\text{bias},i} = -\overline{x_{\text{bias},i}}$ and $\varphi'_{\text{out}} = \bigwedge_{i=1}^{n} z_i = 0 \wedge y = 2^7(m \cdot c^6)$. Note that φ'_{in} is no longer simple and that $N'_{\psi,c,c} \in NN(\{-c, c\})$. □

5.　Conclusion and outlook

We investigated the computational complexity of the reachability problem for NN with PWL activations, including the common activation function ReLU, and input/output specifications given by conjunctions of linear inequalities over the respective input/output dimensions. 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 PWL nodes. Furthermore, we showed that REACH is difficult for very restricted classes of small neural networks already, respectively two parameters of different signum and 0 occurring as weights and biases suffice for NP-hardness. This indicates that finding non-trivial classes of NN and specifications with practical relevance and polynomial REACH is unlikely.

Additionally, modern uses of deep learning techniques limit the practical applicability of findings about REACH even more. There are two reasons for this. On the one hand, the deep learning frame-

work circumvents far more techniques and models than our considered NN model. On the other hand, usual deep learning applications are concerned with data of very high complexity and dimensionality so that simple linear specifications as they are considered here are hardly expressive enough for formalising meaningful properties of the input data and possibly also the output values.

The former concern is tackled by extending the analysis to similar reachability problems of different models, like Convolutional Neural Networks (CNN) [19] or Graph Neural Networks (GNN) [20]. The former incorporates different kinds of layers like convolutional layers or pooling layers. However, in the classical sense, these are just layers consisting of nodes with PWL activations like ReLU, which makes it straightforward to argue that reachability for CNN is NP-complete, too. The case of GNN is more challenging as these are neural network based models which compute functions over graphs. First results on this matter are provided in [21], showing that there are computational limits for the possibility of formal verifying reachability properties of common GNN models.

The latter concern is best understood by example. Consider the problem REACH in the context of an image classification task like classifying whether an x-ray picture shows a broken or healthy bone. It immediately becomes clear that our current definition of specifications is not suited to express properties like "the picture shows a bone with a hairline fracture". But expressing such properties is obviously desired for tasks like this, in order to rule out misclassification of severely damaged bones. Thus, we need specifications of higher expressibility which directly leads to the question whether the upper complexity bound, namely the membership to NP, holds for the reachability problem with specifications extended in a particular way. Starting from our current definition of specifications, a natural extension is allowing for full boolean logic with linear inequalities as propositions, which means adding disjunction and negation operators to our current definition. It is not hard to see that adding disjunction does not change the upper bound. The idea is that we guess for each occurring disjunction which disjunct is satisfied and include this guess in the witness for the NP membership. Obviously, this part of the witness is polynomially bounded by the number of disjunctions in the specifications and therefore by the size of the specifications. As soon as we allow for disjunctions, adding the negation operator does not harm the upper bound either. We can bring each such specification into a normal form (for example conjunctive normal form) where all negations are directly in front of propositions like $\neg(x \leq 2)$ which we interpret as $x > 2$. We provided an argument how to handle strict inequalities in the proof of Lemma 3.2. In summary, full boolean expessibility in the specifications does not change the upper bound for the reachability problem. The obvious next step is adding existential and universal quantifiers to the specifications. Allowing for pure existential quantification in the specifications just leads to an equivalent reformulation of the reachability problem itself. Currently, REACH is formulated such that it existentially quantifies over the free variables in the input and output specifications. Using existential quantifiers, a possible reformulation leads to a normal form of specifications like $\varphi = \exists x_1.\exists x_2.x_1 - x_2 \geq 3 \wedge N(x_1, x_2) \geq 0$ where N is the considered NN and the question is whether φ is satisfiable. It is not hard to see that such a reformulation does not harm the membership in NP. Such specifications with pure universal quantification are not more difficult then, either, because of $\forall x.\psi \equiv \neg\exists x.\neg\psi$. Hence, we simply get that reachability with purely universal specifications is Co-NP-complete. It remains to be seen whether specifications of certain quantifier alternation depth will yield NN-reachability problems that are complete for the corresponding classes in the polynomial hierarchy [22].

References

[1] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. *Commun. ACM*, 2017. **60**(6):84–90. doi:10.1145/3065386.

[2] Hinton G, Deng L, Yu D, Dahl GE, Mohamed Ar, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Process. Mag.*, 2012. **29**(6):82–97. doi:10.1109/MSP.2012.2205597.

[3] Grigorescu SM, Trasnea B, Cocias TT, Macesanu G. A survey of deep learning techniques for autonomous driving. *J. Field Robotics*, 2020. **37**(3):362–386. doi:10.1002/rob.21918.

[4] Litjens G, Kooi T, Bejnordi BE, Setio AAA, Ciompi F, Ghafoorian M, van der Laak JAWM, van Ginneken B, Sánchez CI. A survey on deep learning in medical image analysis. *Medical Image Anal.*, 2017. **42**:60–88. doi:10.1016/j.media.2017.07.005.

[5] Dixon M, Klabjan D, Bang JH. Classification-based financial markets prediction using deep neural networks. *Algorithmic Finance*, 2017. **6**(3-4):67–77. doi:10.3233/AF-170176.

[6] Huang X, Kroening D, Ruan W, Sharp J, Sun Y, Thamo E, Wu M, Yi X. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Comput. Sci. Rev.*, 2020. **37**:100270. doi:10.1016/j.cosrev.2020.100270.

[7] Katz G, Barrett CW, Dill DL, Julian K, Kochenderfer MJ. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: Majumdar R, Kuncak V (eds.), Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I, volume 10426 of *Lecture Notes in Computer Science*. Springer, 2017 pp. 97–117. doi:10.1007/978-3-319-63387-9_5.

[8] Ehlers R. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In: D'Souza D, Kumar KN (eds.), Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings, volume 10482 of *Lecture Notes in Computer Science*. Springer, 2017 pp. 269–286. doi:10.1007/978-3-319-68167-2_19.

[9] Narodytska N, Kasiviswanathan SP, Ryzhyk L, Sagiv M, Walsh T. Verifying Properties of Binarized Deep Neural Networks. In: McIlraith SA, Weinberger KQ (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, February 2-7, 2018. AAAI Press, 2018 pp. 6615–6624. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16898.

[10] Bunel R, Turkaslan I, Torr PHS, Kohli P, Mudigonda PK. A Unified View of Piecewise Linear Neural Network Verification. In: Bengio S, Wallach HM, 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, December 3-8, 2018, Montréal, Canada. 2018 pp. 4795–4804. URL https://proceedings.neurips.cc/paper/2018/hash/be53d253d6bc3258a8160556dda3e9b2-Abstract.html.

[11] 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, July 13-19, 2018, Stockholm, Sweden. ijcai.org, 2018 pp. 2651–2659. doi:10.24963/ijcai.2018/368.

[12] Ruan W, Huang X, Kwiatkowska M. Reachability Analysis of Deep Neural Networks with Provable Guarantees. *CoRR*, 2018. **abs/1805.02242**. 1805.02242, URL http://arxiv.org/abs/1805.02242.

[13] Katz G, Barrett CW, Dill DL, Julian K, Kochenderfer MJ. Reluplex: a calculus for reasoning about deep neural networks. *Form Methods Syst Des*, 2021. doi:10.1007/s10703-021-00363-7.

[14] Sälzer M, Lange M. Reachability is NP-Complete Even for the Simplest Neural Networks. In: Reachability Problems - 15th International Conference, RP 2021, Liverpool, UK, October 25-27, 2021, Proceedings, volume 13035 of *Lecture Notes in Computer Science*. Springer, 2021 pp. 149–164. doi:10.1007/978-3-030-89716-1_10.

[15] Karmarkar N. A new polynomial-time algorithm for linear programming. *Comb.*, 1984. **4**(4):373–396. doi:10.1007/BF02579150.

[16] Akintunde M, Lomuscio A, Maganti L, Pirovano E. Reachability Analysis for Neural Agent-Environment Systems. In: Thielscher M, Toni F, Wolter F (eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018. AAAI Press, 2018 pp. 184–193. URL https://aaai.org/ocs/index.php/KR/KR18/paper/view/17991.

[17] Karp RM. Reducibility Among Combinatorial Problems. In: Miller RE, Thatcher JW (eds.), Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, The IBM Research Symposia Series. Plenum Press, New York, 1972 pp. 85–103. doi:10.1007/978-1-4684-2001-2_9.

[18] Korte B, Vygen J. Combinatorial Optimization. Springer Berlin, Heidelberg, 2006. ISBN 978-3-540-29297-5. doi:10.1007/3-540-29297-7.

[19] Khan A, Sohail A, Zahoora U, Qureshi AS. A survey of the recent architectures of deep convolutional neural networks. *Artif. Intell. Rev.*, 2020. **53**(8):5455–5516. doi:10.1007/s10462-020-09825-6.

[20] Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.*, 2021. **32**(1):4–24. doi:10.1109/TNNLS.2020.2978386.

[21] Sälzer M, Lange M. Fundamental Limits in Formal Verification of Message-Passing Neural Networks. In: The Eleventh International Conference on Learning Representations. 2023 URL https://openreview.net/forum?id=WlbG820mRH-.

[22] Stockmeyer LJ. The polynomial-time hierarchy. *Theor. Comp. Sci.*, 1976. **3**(1):1–22.

This figure "image001.png" is available in "png" format from:

http://arxiv.org/ps/2203.07941v4