The Reachability Problem for Neural-Network Control Systems

Christian Schilling[1][0000−0003−3658−1065] and
Martin Zimmermann[1][0000−0002−8038−2453]

Aalborg University, Aalborg, Denmark
{christianms,mzi}@cs.aau.dk

Abstract. A control system consists of a plant component and a controller which periodically computes a control input for the plant. We consider systems where the controller is implemented by a feedforward neural network with ReLU activations. The reachability problem asks, given a set of initial states, whether a set of target states can be reached. We show that this problem is undecidable even for trivial plants and fixed-depth neural networks with three inputs and outputs. We also show that the problem becomes semi-decidable when the plant as well as the input and target sets are given by automata over infinite words.

1 Introduction

Cyber-physical systems consist of digital (cyber) and physical components. A common instance of this paradigm is a control system, consisting of a physical *plant* and a *controller* whose purpose is to steer the plant to a desired behavior [7]. Control theory studies the automatic synthesis of such controllers, which, for nonlinear systems, is a difficult task. Machine learning has long been successfully applied to tackle this task, where the learned controller was first represented by a (shallow) feedforward neural network [19] and more recently by a *deep neural network* (DNN) [15]. We call a control system with a DNN controller a *neural-network control system* (NNCS). Due to their black-box nature, DNNs have raised concerns about their correctness and safety, in particular in terms of the worst-case behavior [28]. However, as they are deployed in safety-critical applications, proving machine-learned NNCS correct is of utmost importance, and considerable resources have been invested into their verification [16, 17].

In this paper, we are concerned with the fundamental problem of safety for NNCS: given a set of initial states and a set of bad states of the plant, does the controller prevent the plant to reach a bad state when started in an initial state? Note that the failure of safety is captured by a reachability property: does there exist an initial state from which a bad state is reachable? Thus, in the following, we study the reachability problem for NNCS.

Recall that an NNCS is a combination of a DNN (the controller) and a plant. It is known that the reachability problem is already undecidable for sufficiently complex plants, even without any controller [13]. So the question becomes: is

there a simple but expressive class of plants for which the reachability problem is tractable? Inspired by similar results for recurrent neural networks [27, 10], we show in Section 3 that the answer is negative: the reachability problem is undecidable even for trivial plants. Intuitively, a DNN can simulate one computational step of a two-counter machine. Thus, a recurrent neural network can simulate a two-counter machine. As a DNN controlling a plant is essentially recurrent (as it bases its control decisions on the current state of the plant), undecidability follows.

On the positive side, we show in Section 4 that the reachability problem is at least semi-decidable for plants whose behavior can be captured by automata over infinite words: Sälzer et al. showed that the behavior of DNNs can be captured by such automata [23]. Hence, relying on standard automata-theoretic constructions, the composition of a DNN and an automata-definable plant can also be captured by automata. The class of automata-definable plants includes, for instance, plants that are described by multi-mode linear maps. Such maps are able to express, for example, the dynamics of adaptive cruise controls [14].

1.1 Related work

Reachability in NNCS is generally challenging. Existing approaches typically combine techniques developed for dynamical systems (the plant) [2] and neural networks [16]. Tools such as CORA [1, 12], JuliaReach [5, 25], and NNV [18] compete in the ARCH-COMP friendly competition, and we refer to the report [17] for typical examples of NNCS.

Undecidability of questions about unbounded computations with piecewise-linear (PWL) functions is long known, e.g., periodicity in iterated 2D maps [22] or reachability for linear hybrid automata [9]. Similar results have been shown for DNNs. Siegelmann and Sontag showed undecidability for unbounded computations in DNNs with activations given by a PWL approximation of the sigmoid function (which effectively is the ReLU function truncated at 1) [27]. Later, Hyotyniemi showed an encoding of two-counter machines in recurrent neural networks (RNNs) with ReLU activations [10]. While an RNN can conceptually be seen as the special case of an NNCS without a plant, the formalism differs. We thus consider our encoding of two-counter machines in NNCS of independent (yet mainly pedagogical) value. Cabessa showed an encoding of two-counter machines in a variant of RNNs with conditional weights called *spike-timing dependent plasticity* [6]. Recently, we also showed an encoding of two-counter machines in decision-tree control systems [26], where the DNN is replaced by a decision tree with simple conditions $x \leq c$ for some variable x and constant c.

Katz et al. studied the problem of reachability in a ReLU DNN without iteration and showed that, given polyhedral (i.e., described by linear constraints) input and output sets, the reachability problem is NP-complete [11]. Sälzer and Lange later fixed some issues in the proof, mainly related to the effective representation of real numbers [24].

Sälzer et al. recently presented an encoding of a DNN in a weak Büchi automaton [23]. We build on this encoding for the analysis of semi-decidability.

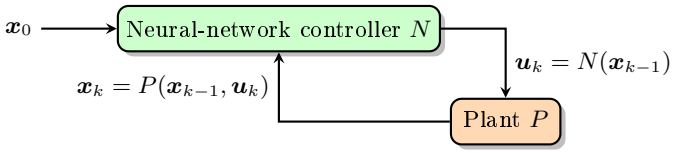


Fig. 1: Neural-network control system.

2 Preliminaries

We start by formally introducing the type of DNN under study.

Definition 1 (Deep neural network). *A* neuron *is a function* $\nu\colon \mathbb{R}^m \to \mathbb{R}$ *with* $\nu(\boldsymbol{x}) = \sigma(\sum_{i=1}^{m} c_i x_i + b)$*, where* m *is the input dimension, the* $c_i \in \mathbb{Q}$ *are the weights,* $b \in \mathbb{Q}$ *is the bias, and* $\sigma\colon \mathbb{R} \to \mathbb{R}$ *is the activation function of* ν*, which is either the identity function or the rectified linear unit (ReLU)* $y \mapsto \max\{y, 0\}$*.*

A layer *is a sequence of neurons* $(\nu_1, \ldots, \nu_n)$*, all of the same input dimension* m*, computing the function* $\ell\colon \mathbb{R}^m \to \mathbb{R}^n$ *given by* $\ell(\boldsymbol{x}) = (\nu_1(\boldsymbol{x}), \ldots, \nu_n(\boldsymbol{x}))$*. The dimensions* m *and* n *are the input resp. output dimension of the layer.*

A deep neural network *(DNN)* N *is a sequence of layers* $(\ell_1, \ldots, \ell_k)$ *such that the output dimension of* ℓ_i *is the input dimension of* ℓ_{i+1} *for all* $i = 1, \ldots, k-1$*. The last layer is the output layer and all other layers are called hidden layers. If* m *is the input dimension of* ℓ_1 *and* n *is the output dimension of* ℓ_k*, then the DNN computes the function* $N\colon \mathbb{R}^m \to \mathbb{R}^n$ *defined as*

$$N(\boldsymbol{x}) = \ell_k(\ell_{k-1}(\ldots \ell_1(\boldsymbol{x}) \ldots)).$$

Next, we define control systems as depicted in Fig. 1. The system consists of a plant and a controller (here: a DNN) and acts in iterations: first, the controller computes a control input $\boldsymbol{u}$ for the plant based on the current state $\boldsymbol{x}$ of the plant. Next, from its inputs $\boldsymbol{x}$ and $\boldsymbol{u}$, the plant computes a new state. Then the process repeats. For the plant, we restrict ourselves to discrete time, i.e., we are only interested in its output and not its intermediate states. For now, we also abstract from the concrete type of plant and just view it as a general function.

Definition 2 (Neural-network control system). *A neural-network control system (NNCS) is a tuple* (P, N) *with a plant* $P\colon \mathbb{R}^{d+c} \to \mathbb{R}^d$ *and a controller given by a DNN* $N\colon \mathbb{R}^d \to \mathbb{R}^c$*, i.e.,* d *is the dimension of the states of* P *and* c *is the dimension of the control vectors computed by* N*.*

The semantics of an NNCS are given as a sequence of states $\boldsymbol{x}_k$ *and control inputs* $\boldsymbol{u}_k$*, induced by some initial state* $\boldsymbol{x}_0 \in \mathbb{R}^d$ *via*

$$\begin{aligned} \boldsymbol{u}_k &= N(\boldsymbol{x}_{k-1}) \\ \boldsymbol{x}_k &= P(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \end{aligned}$$

We introduce a shorthand to express one iteration of the control loop in Fig. 1, i.e., the composition of the DNN followed by the plant, to compute $\boldsymbol{x}_k$

from $\boldsymbol{x}_{k-1}$:

$$C_{P,N}(\boldsymbol{x}_{k-1}) = P(\boldsymbol{x}_{k-1}, N(\boldsymbol{x}_{k-1}))$$

We will focus on sets of states represented by linear constraints. Given $a \in \mathbb{Q}^n$, $b \in \mathbb{Q}$, the set $H_{a,b} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \langle a, \boldsymbol{x}\rangle \leq b\}$ is a linear constraint, where "$\langle\cdot,\cdot\rangle$" denotes the scalar product. A polyhedron is a finite intersection of linear constraints. Let $\mathcal{P}(n)$ denote the set of all polyhedra in n dimensions.

We are now ready to define the reachability problem for NNCS.

Problem 1 (Reachability problem for NNCS). Given a DNN $N\colon \mathbb{R}^d \to \mathbb{R}^c$, a plant $P\colon \mathbb{R}^{d+c} \to \mathbb{R}^d$, a polyhedron $X_0 \in \mathcal{P}(d)$ of initial states, and a polyhedron $\varphi \in \mathcal{P}(d)$ of target states, does there exist an initial state $\boldsymbol{x}_0 \in X_0$ and a $k \in \mathbb{N}$ such that $(C_{P,N})^k(\boldsymbol{x}_0) \in \varphi$?

3 Undecidability

In this section, we prove that the NNCS reachability problem is undecidable. The proof is by a reduction from the halting problem for two-counter machines.

Formally, a two-counter machine $\mathcal{M}$ is a sequence

$$(0 : \mathtt{I}_0)(1 : \mathtt{I}_1)\cdots(k-2 : \mathtt{I}_{k-2})(k-1 : \mathtt{STOP}),$$

where the first element of a pair $(\ell : \mathtt{I}_\ell)$ is the line number and $\mathtt{I}_\ell$ for $0 \leq \ell < k-1$ is an instruction of the form

- $\mathtt{INC}(i)$ with $i \in \{0,1\}$,
- $\mathtt{DEC}(i)$ with $i \in \{0,1\}$, or
- $\mathtt{JZ}(i,\ell')$ with $i \in \{0,1\}$ and $\ell' \in \{0,\ldots,k-1\}$.

A configuration of $\mathcal{M}$ is of the form (ℓ, c_0, c_1) with $\ell \in \{0,\ldots,k-1\}$ (the current value of the program counter) and $c_0, c_1 \in \mathbb{N}$ (the current contents of the two counters). The initial configuration is $(0,0,0)$ and the unique successor configuration of a configuration (ℓ, c_0, c_1) is defined as follows:

- If $\mathtt{I}_\ell = \mathtt{INC}(i)$, then the successor configuration is $(\ell{+}1, c_0', c_1')$ with $c_i' = c_i{+}1$ and $c_{1-i}' = c_{1-i}$.
- If $\mathtt{I}_\ell = \mathtt{DEC}(i)$, then the successor configuration is $(\ell+1, c_0', c_1')$ with $c_i' = \max\{c_i - 1, 0\}$ and $c_{1-i}' = c_{1-i}$.
- If $\mathtt{I}_\ell = \mathtt{JZ}(i,\ell')$ and $c_i = 0$, then the successor configuration is (ℓ', c_0, c_1).
- If $\mathtt{I}_\ell = \mathtt{JZ}(i,\ell')$ and $c_i > 0$, then the successor configuration is $(\ell+1, c_0, c_1)$.
- If $\mathtt{I}_\ell = \mathtt{STOP}$, then (ℓ, c_0, c_1) has no successor configuration.

The unique run of $\mathcal{M}$ (starting in the initial configuration) is defined as the maximal sequence $\gamma_0\gamma_1\gamma_2\cdots$ of configurations $\gamma_j \in \mathbb{N}^3$ where γ_0 is the initial configuration and γ_{j+1} is the successor configuration of γ_j, if γ_j has a successor configuration. This run is either finite (line $k{-}1$ is reached) or infinite (line $k{-}1$ is never reached). In the former case, we say that $\mathcal{M}$ terminates. The halting problem for two-counter machines asks, given a two-counter machine $\mathcal{M}$, whether $\mathcal{M}$ terminates when started in the initial configuration.

Proposition 1 ([20]). *The halting problem for two-counter machines is undecidable.*

In the following, we show that the halting problem for two-counter machines can be reduced to the reachability problem for NNCS by simulating the semantics of a two-counter machine by a NNCS.

Theorem 1. *The reachability problem for NNCS is undecidable.*

Proof. Fix some two-counter machine $\mathcal{M}$ with k instructions. We show how to construct a gadget for each instruction of $\mathcal{M}$ (except for the `STOP` instruction), which we then combine into a DNN simulating one configuration update of $\mathcal{M}$. Thus, the reachability problem for NNCS (which involves the iterated application of the DNN) then allows to simulate the full run of $\mathcal{M}$.

Formally, the DNN implements a function from $\mathbb{R}^3 \to \mathbb{R}^3$ with the following property: If the three inputs encode a non-stopping configuration of the two-counter machine, then the three outputs encode the successor configuration. Note that, since the weights and biases of the DNN we construct are integral, the outputs given integral inputs are also integral. In the following, we often implicitly assume that inputs are integral when we explain the intuition behind our construction.

Our construction of the DNN fits into the common architecture [8] that all hidden neurons use ReLU activations and all output neurons use identity activations. We let the plant component simply turn the control input into the new state ($P(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{u}$), as the DNN already simulates $\mathcal{M}$.

In some more detail, for every instruction $(\ell; \mathtt{I}_\ell)$ of $\mathcal{M}$, we construct one gadget simulating this instruction. All these gadgets will be executed in parallel in one iteration of the DNN, but only one of them (determined by the current value of the program counter) will actually perform a computation. The other gadgets just compute the identity function for each of their inputs. Thus, in the end we need to subtract $(k-2) \cdot v$ from each output v.

All gadgets have inputs named pc (representing the current value of the program counter), and c_0 and c_1 (representing the current counter values), as well as three outputs named pc' (representing the value of the program counter of the successor configuration), and c_0' and c_1' (representing the counter values of the successor configuration). To simplify our construction, we use an additional gadget that conceptually checks whether the value of the program counter is equal to some fixed line number ℓ. This gadget is shown in Fig. 2. The output a_ℓ of this gadget (which has only one input pc) satisfies

$$a_\ell = \begin{cases} 1 & pc = \ell, \\ 0 & pc \neq \ell. \end{cases}$$

The outputs a_ℓ of these auxiliary gadgets (we have one for each line number ℓ) will be fed into the other gadgets simulating the instructions.

Next, we describe the instruction gadgets, where we restrict ourselves to the counter with index zero; the counter with index one is treated in the analogous

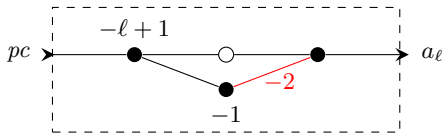


Fig. 2: Auxiliary gadget for instruction ℓ. Here and in all later illustrations of DNNs, dots denote neurons, where filled dots use ReLU activations and empty dots can use either identity or ReLU activations (the choice is irrelevant since the value before the activation is nonnegative). Sometimes, as in this case, the empty dots are only present for a fully-connected architecture. Edge colors only serve the visual association with the weights. We omit weight 1 and bias 0 as well as connections with weight 0.

way. Fig. 3 shows these gadgets together with the possible outputs. It is easy to verify that each gadget performs the corresponding computation whenever the input pc is equal to ℓ, and the identity function otherwise. Let us stress that each gadget we construct depends both on the line number and the instruction.

The final layout of the gadgets is shown in Fig. 4. Essentially, each auxiliary gadget is wired to the corresponding instruction gadget, and at the end we need to subtract the inputs $k-2$ times as described above. Note that there is no gadget for the `STOP` instruction (instruction $k-1$ in $\mathcal{M}$). When pc is equal to $k-1$, then the DNN computes the identity function: First, all a_ℓ are equal to 0; Hence, each of the $k-1$ instruction gadgets I_ℓ computes the identity function; after the subtraction, we are indeed left with the identity.

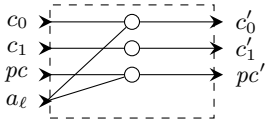
Finally, the initial input to the DNN is $\boldsymbol{x}_0 = (0,0,0)$ (representing the initial configuration) and the target set is $\varphi = \{(k-1, c_0, c_1) \mid c_0, c_1 \geq 0\}$, where $k-1$ is the last instruction number (`STOP`) of $\mathcal{M}$. Clearly, $\mathcal{M}$ terminates if and only if the NNCS reaches a state satisfying φ when started in $X_0 = \{\boldsymbol{x}_0\}$. □

We note that the DNNs simulating two-counter machines are rather simple.

Corollary 1. *The NNCS reachability problem remains undecidable for DNNs with integral weights, 3 input and output dimensions, 6 hidden layers, a singleton initial set, and a target set $o = v$ for some output neuron o and constant $v \in \mathbb{N}$.*

One may wonder whether the six hidden layers are necessary. In general, one cannot hope to obtain a small neural network when removing layers [4]. However, since we can iterate the NNCS, and the plant model is not interfering, we can reduce one iteration of a DNN N with six hidden layers (constructed in the proof above) to seven iterations of a DNN N' with one hidden layer.[1] Fig. 5 shows a sketch of the construction idea. Essentially, we take the hidden layers of N and stack them as one wide hidden layer in N'. (For instance, layer ℓ_1 has width $k+1$.) We refer to each of these hidden layers as a *track*. For each track, we need to add input and output dimensions corresponding to the number of

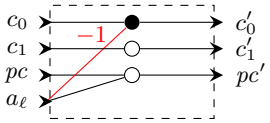
[1] Typically, neural networks with only one hidden layer are not called *deep*.



(a) ℓ : INC(0) gadget.

Condition	x'	y'	pc'
$pc = \ell$	$c_0 + 1$	c_1	$pc + 1$
$pc \neq \ell$	c_0	c_1	pc

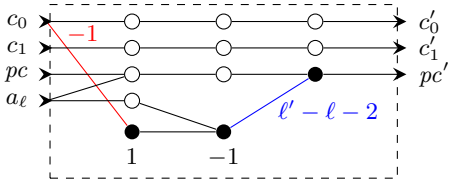
(b) ℓ : INC(0) gadget's possible output values.



(c) ℓ : DEC(0) gadget.

Condition	c_0'	c_1'	pc'
$pc = \ell$	$\max\{c_0 - 1, 0\}$	c_1	$pc + 1$
$pc \neq \ell$	c_0	c_1	pc

(d) ℓ : DEC(0) gadget's possible output values.



(e) ℓ : JZ$(0, \ell')$ gadget.

Condition	c_0'	c_1'	pc'
$pc = \ell$ and $c_0 = 0$	c_0	c_1	ℓ'
$pc = \ell$ and $c_0 \neq 0$	c_0	c_1	$pc + 1$
$pc \neq \ell$	c_0	c_1	pc

(f) ℓ : JZ$(0, \ell')$ gadget's possible output values.

Fig. 3: The gadgets for the three instructions. See Fig. 2 for further explanations.

neurons in the respective previous and next hidden layers. The output of track j is fed to track $j + 1$ (and the output of the last track is fed to the first track).

When presented with an input vector $\boldsymbol{x}_0$ of appropriate size, the first track performs the computation of the first hidden layer and feeds its output to the second track, and so on. After seven iterations, the output of the last track will equal the output of the seven-layer DNN N after the first iteration. This output is then used as the input of the first track again and the process continues.

Finally, we need to make sure that the other tracks do not accidentally produce an output that leads to a target state between multiples of seven iterations. In order to only consider outputs in every seventh iteration, we use the additional gadget shown in Fig. 5(b). This gadget has seven inputs and outputs and is to be stacked below the other DNN. When the initial input is $(1, 0, 0, 0, 0, 0, 0)$, the 1 is propagated to the second index, and so on, until it arrives back at the first index after seven iterations.

The target set φ now simply needs to get extended to arbitrary values in the auxiliary dimensions, except for the seven-last entry (m_i^1), which has to equal one. Formally: $\varphi' = (\varphi \wedge m_i^1 \leq 1 \wedge -m_i^1 \leq -1)$.

In summary, by scaling the number of inputs and outputs with $\mathcal{M}$, we obtain a DNN with one (wide) hidden layer.

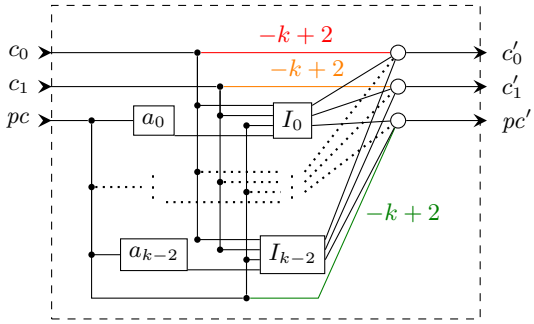


Fig. 4: Complete construction. Each box represents an auxiliary gadget a_ℓ resp. an instruction gadget I_ℓ. Small dots denote junctions of connections and have no further semantics. The last layer is the output layer (with identity activations).

Corollary 2. *The NNCS reachability problem remains undecidable for DNNs with integral weights, one hidden layer, a singleton initial set, and a target set $o = v$ for some output neuron o and constant $v \in \mathbb{N}$.*

4 Semi-decidability

In this section, we show that the NNCS reachability problem is semi-decidable for a particular class of plants. Indeed, from a single initial state $\boldsymbol{x}_0$, we can enumerate all states $C_{P,N}(\boldsymbol{x}_0)^k$ reachable in k iterations and for each of them decide membership in the target polyhedron φ. However, since we allow for an initial set X_0, this algorithm is not effective.

The image of a polyhedron under a ReLU DNN is a (finite) union of polyhedra [21]. If we choose a class of plants with the same property, we obtain an effective algorithm again. In what follows, we show a more general result by using an automaton encoding of DNNs from [23]. This will allow us to more abstractly consider a class of plants that is definable in the same automaton formalism.

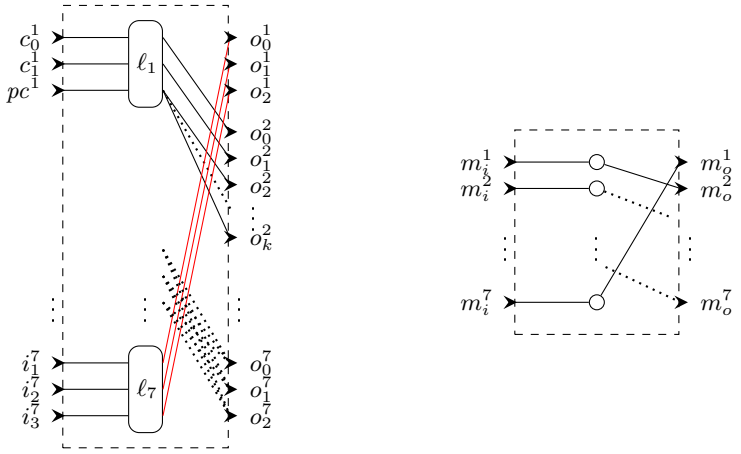
We slightly deviate from the original approach by Sälzer et al. [23] in that we use a more expressive automaton model, as we are (unlike them) not bothered with efficiency considerations (since our problem is undecidable).

Definition 3 (Büchi automaton). *A (nondeterministic)* Büchi automaton *(NBA) $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ consists of a finite set Q of states, a finite alphabet Σ, an initial state $q_0 \in Q$, a transition relation $\delta \subseteq Q \times \Sigma \times Q$, and a set of accepting states $F \subseteq Q$.*

A run on an infinite word $w = a_0 a_1 \dots$ is an infinite sequence of states $q_0, q_1, \dots$ starting in the initial state and satisfying $(q_i, a_i, q_{i+1}) \in \delta$ for all $i \geq 0$. A run is accepting if $q_i \in F$ for infinitely many i. The language of $\mathcal{A}$ is

$$L(\mathcal{A}) = \{w \in \Sigma^\omega \mid \mathcal{A} \text{ has an accepting run on } w\}.$$

A language is ω-regular if there exists an NBA that accepts it.



(a) Stacking of previous hidden layers. (b) Modulo-7 counter.

Fig. 5: Construction with a single hidden layer.

In the following, we recall an effective encoding of real numbers in NBA from [23]. Let $\Sigma = \{+, -, 0, 1, .\}$. A word $w = sa_n \dots a_0.b_0b_1 \dots$ with $n \geq 0$, $s \in \{+, -\}$, $a_i, b_i \in \{0, 1\}$ encodes the real value

$$dec(w) = (-1)^{\text{sign}(s)} \cdot \left(\sum_{i=0}^{n} a_i \cdot 2^i + \sum_{i=0}^{\infty} b_i \cdot 2^{-(i+1)} \right)$$

where $\text{sign}(s) = 0$ if $s = +$ and $\text{sign}(s) = 1$ if $s = -$. As usual, the word encoding is not unique, but the decoding is [23, Page 5].

Now, we switch to a word encoding of multiple numbers by using a product alphabet. A symbol over this product alphabet Σ^k is a k-vector of symbols. A word over Σ^k is well-formed if it is of the form

$$w = \begin{bmatrix} s_1 \\ \vdots \\ s_k \end{bmatrix} \begin{bmatrix} a_{1,n} \\ \vdots \\ a_{k,n} \end{bmatrix} \cdots \begin{bmatrix} a_{1,0} \\ \vdots \\ a_{k,0} \end{bmatrix} \begin{bmatrix} . \\ \vdots \\ . \end{bmatrix} \begin{bmatrix} b_{1,0} \\ \vdots \\ b_{k,0} \end{bmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{k,1} \end{bmatrix} \cdots$$

where $s_i \in \{+, -\}$, $a_{i,j}, b_{i,h} \in \{0, 1\}$ for $i = 1, \dots, k$, $j = 0, \dots, n$, and $h = 0, 1, \dots$. In other words, the signs and the point are aligned, which can be achieved by filling up with leading zeros. The language WF_k of well-formed words is ω-regular [23]. The selection of a single component $i \in \{1, \dots, k\}$ is obtained in the obvious way:

$$w_i = s_i a_{i,n} \dots a_{i,0} \, . \, b_{i,0} b_{i,1} \dots$$

If an NBA over Σ^k accepts only well-formed words, then we can understand its language as a relation over $\mathbb{R}^k$. Furthermore, linear constraints are also ω-regular [23]. Thus, as NBA are closed under intersection and union, (finite unions

of) polyhedra are also ω-regular. Finally, we can also use NBAs to encode functions $f\colon \mathbb{R}^m \to \mathbb{R}^n$ via their graphs, which are relations over $\mathbb{R}^{m+n}$.

Sälzer et al. showed that every function computable by a DNN can be represented by an NBA.[2]

Proposition 2 (Theorem 1 in [23]). *Let $N\colon \mathbb{R}^d \to \mathbb{R}^c$ be a DNN. There exists an NBA $\mathcal{A}_N$ over Σ^{d+c} with*

$$L(\mathcal{A}_N) = \{w \in WF_{d+c} \mid N(dec(w_1), \dots, dec(w_d)) = (dec(w_{d+1}), \dots, dec(w_{d+c}))\}.$$

For our application, we need to slightly modify the automaton $\mathcal{A}_N$ from Proposition 2 so that it also copies its input for further use. This modification can be implemented by replacing each transition label $(a_1, \dots, a_n, a'_1, \dots, a'_m)$ by $(a_1, \dots, a_n, a_1, \dots, a_n, a'_1, \dots, a'_m)$.

Corollary 3. *Let $N\colon \mathbb{R}^d \to \mathbb{R}^c$ be a DNN. There exists an NBA $\widehat{\mathcal{A}}_N$ over Σ^{d+d+c} with*

$$\begin{aligned} L(\widehat{\mathcal{A}}_N) = \{w \in WF_{d+d+c} \mid (w_1, \dots, w_d) = (w_{d+1}, \dots, w_{d+d}) \text{ and} \\ N((dec(w_1), \dots, dec(w_d)) = (dec(w_{d+d+1}), \dots, dec(w_{d+d+c}))\}. \end{aligned}$$

Thus, the semantics of DNNs can be captured by NBAs. This is in general not true for plants. Hence, in the following, we restrict ourselves to plants that can also be captured by NBAs.

Definition 4 (ω-regular plant). *A plant $P\colon \mathbb{R}^{d+c} \to \mathbb{R}^d$ is* ω-regular *if there exists an NBA $\mathcal{A}_P$ over Σ^{d+c+d} such that*

$$\begin{aligned} L(\mathcal{A}_P) = \{w \in WF_{d+c+d} \mid \\ P(dec(w_1), \dots, dec(w_{d+c})) = (dec(w_{d+c+1}), \dots, dec(w_{d+c+d}))\}. \end{aligned}$$

Now, both the DNN and the plant are given by NBA. Hence, we can apply standard automata-theoretic constructions to capture a bounded number of applications of the control loop by repeatedly composing the NBA for the DNN and the NBA for the plant. To this end, we introduce the (parametric) composition operator $\circ_n$ constructing from two NBAs $\mathcal{A}_1$ and $\mathcal{A}_2$, which accept the graphs of two functions $f_1\colon \mathbb{R}^{k_1} \to \mathbb{R}^k$ and $f_2\colon \mathbb{R}^k \to \mathbb{R}^{k_2}$, an NBA $\mathcal{A}_1 \circ_k \mathcal{A}_2$ accepting the graph of $\boldsymbol{x} \mapsto f_2(f_1(\boldsymbol{x}))$.

Lemma 1 (Lemma 4 of [23]). *Let $k, k_1, k_2 \geq 0$ and let A_1 and $\mathcal{A}_2$ be two NBAs over Σ^{k_1+k} and Σ^{k+k_2}, respectively. Then, there exists an NBA $\mathcal{A}_1 \circ_k \mathcal{A}_2$ over $\Sigma^{k_1+k_2}$ accepting the language*

$$\begin{aligned} \{(u_1, \dots, u_{k_1}, w_{k+1}, \dots, w_{k+k_2}) \mid \exists (v_1, \dots, v_k) \text{ s.t.} \\ (u_1, \dots, u_{k_1}, v_1, \dots, v_k) \in L(\mathcal{A}_1) \text{ and} \\ (v_1, \dots, v_k, w_{k+1}, \dots, w_{k+k_2}) \in L(\mathcal{A}_2)\}. \end{aligned}$$

[2] They actually proved the result for the more restrictive class of eventually-always weak NBA. But for us it is more prudent to consider the more general class of NBA.

Now we are ready to prove our main result of this section: NNCS reachability restricted to ω-regular plants is semi-decidable. Note that this is tight, as the problem is undecidable as shown in Theorem 1: the plant just returning its control input as output is ω-regular.

Theorem 2. *The NNCS reachability problem is semi-decidable when restricted to ω-regular plants.*

Proof. We are given a problem instance (N, P, X_0, φ) and need to (semi)-decide whether there exists a $k \geq 0$ such that $(C_{P,N})^k(\boldsymbol{x}_0) \in \varphi$ for some $\boldsymbol{x}_0 \in X_0$. Let d be the dimension of the states of P and c be the dimension of the control vectors computed by N, respectively.

Let $\mathcal{A}_N$ (over Σ^{d+d+c}) and $\mathcal{A}_P$ (over Σ^{d+c+d}) be the NBAs as in Corollary 3 and Definition 4. Then, we define I_0 to be an NBA accepting the graph of the d-ary identity function

$$L(I_0) = \{w \in WF_{d+d} \mid (w_0, \ldots, w_d) = (w_{d+1}, \ldots, w_{d+d})\}$$

and, for $k \geq 1$, $I_k = I_{k-1} \circ_d (\widehat{\mathcal{A}}_N \circ_{d+c} \mathcal{A}_P)$.

By construction, we have

$$(w_1, \ldots, w_d, w_{d+1}, \ldots, w_{d+d}) \in L(I_k)$$

if and only if

$$(dec(w_{d+1}), \ldots, dec(w_{d+d})) \in (C_{P,N})^k(dec(w_1), \ldots, dec(w_d)).$$

There are NBAs $\mathcal{A}_0$ and $\mathcal{A}_\varphi$ accepting X_0 and φ, as they are polyhedra. Both these NBAs have alphabet Σ^d, while each I_k has alphabet Σ^{d+d} where the first d components encode the inputs and the last d components encode the outputs. Hence, to restrict $\mathcal{A}_0$ and $\mathcal{A}_\varphi$ to X_0 and φ, we need to widen $\mathcal{A}_0$ and $\mathcal{A}_\varphi$ to NBA with alphabet Σ^{d+d}. Formally, let NBAs $\widehat{\mathcal{A}}_0$ and $\widehat{\mathcal{A}}_\varphi$ (both over Σ^{d+d}) such that

- $L(\widehat{\mathcal{A}}_0)$ contains the encodings of all vectors $(x_1, \ldots, x_{d+d}) \in WF_{d+d}$ such that $(x_1, \ldots, x_d)$ is in $X_0 \subseteq \mathbb{R}^d$ and $(x_{d+1}, \ldots, x_{d+d}) \in \mathbb{R}^d$ is arbitrary, and
- $L(\widehat{\mathcal{A}}_\varphi)$ contains the encodings of all vectors $(x_1, \ldots, x_{d+d}) \in WF_{d+d}$ such that $(x_1, \ldots, x_d) \in \mathbb{R}^d$ is arbitrary and $(x_{d+1}, \ldots, x_{d+d})$ is in $\varphi \subseteq \mathbb{R}^d$.

Now, there exist an $\boldsymbol{x}_0 \in X_0$ and a $k \geq 0$ such that $(C_{P,N})^k(\boldsymbol{x}_0) \in \varphi$ if and only if the language of $\widehat{\mathcal{A}}_0 \cap I_k \cap \widehat{\mathcal{A}}_\varphi$ is nonempty.

In summary, to semi-decide the NNCS reachability problem for ω-regular plants, we iteratively construct I_k for $k \geq 0$ and check $\widehat{\mathcal{A}}_0 \cap I_k \cap \widehat{\mathcal{A}}_\varphi$ for nonemptiness. □

Let us remark that the construction in Theorem 2 does not require the initial set X_0 and the target set φ to be polyhedral. It is sufficient that they are ω-regular to effectively decide (non)emptiness of the intersection. The class of ω-regular languages is strictly more expressive than polyhedral sets.[3] Thus, our result is more general than the statement of Theorem 2.

[3] For example, the set of natural numbers is ω-regular (in the encoding used here), but it is not a polyhedron.

4.1 Multi-mode linear plants

In this subsection, we give an example of a plant model that falls into the class of ω-regular languages. Our example is inspired by linear hybrid automata [3], which are finite state machines with constant-term ordinary differential equations (ODEs) in the modes (states) and guard conditions on the transitions.

Hybrid automata have two sources of nondeterminism: an enabled transition need not be taken (may-semantics), and multiple transitions may be enabled at the same time. Because we have restricted ourselves to deterministic plants in this work, we need to introduce some restrictions. First, we assume a fixed rational control period, and a transition can only be taken at the end of such a period. Then, the solution of the ODEs is a linear map, which can be analytically computed, and our system becomes discrete-time. Second, we require that exactly one guard is enabled, i.e., in each mode, all guards are pairwise-disjoint and their union is the universe. To simplify the presentation, we do not include discrete updates with the transitions but note that these can easily be added. We call the resulting model a multi-mode linear map.

Definition 5 (Multi-mode linear map). *A* multi-mode linear map *is a tuple $\mathcal{H} = (M, E, d, c, F, G)$ consisting of a finite set $M \subseteq \mathbb{N}$ of modes, a set of edges $E \subseteq M \times M$, input and control dimensions d and c, a flow function $F\colon M \to \mathbb{Q}^{d\times d} \times \mathbb{Q}^{d\times c} \times \mathbb{Q}^d$ (mapping a mode to two matrices and a vector), and a guard function $G\colon E \to \mathcal{FUP}(d+c)$ (where $\mathcal{FUP}$ denotes the set of finite unions of polyhedra), satisfying*

- *if $(m, m') \in E$ and $(m, m'') \in E$, then $G(m, m') \cap G(m, m'') = \emptyset$, and*
- *$\bigcup_{m'\in M} G(m, m') = \mathbb{R}^{d+c}$ for all $m \in M$.*

The function $f_{\mathcal{H}}\colon M \times \mathbb{R}^{d+c} \to M \times \mathbb{R}^d$ computed by $\mathcal{H}$ is defined as

$$f_{\mathcal{H}}(m, x_1, \ldots, x_d, u_1, \ldots, u_c) = (m', x'_1, \ldots, x'_d)$$

where $F(m) = (A, B, c)$,

$$(x'_1, \ldots, x'_d) = A \cdot (x_1, \ldots, x_d)^T + B \cdot (u_1, \ldots, u_c)^T + c,$$

and m' is the unique mode such that

$$(x'_1, \ldots, x'_d, u_1, \ldots, u_c) \in G(m, m').$$

Note that the first component of inputs for $f_{\mathcal{H}}$ is restricted to modes of $\mathcal{H}$, not arbitrary reals as stipulated by the definition of plants. However, this is not an issue as long as the initial input has a mode in the first component, as $f_{\mathcal{H}}$ also returns only outputs that have a mode in the first component.

Lemma 2. *Multi-mode linear maps are ω-regular plants.*

Proof (Sketch). The following operations can be implemented by NBAs [23]:

- Multiplication of real inputs with constants in $\mathbb{Q}$ and addition of reals. These two operations allow us to compute the output $(x'_1, \ldots, x'_d)$ from $(x_1, \ldots, x_d)$ and $(u_1, \ldots, u_c)$.
- Checking membership of a vector of reals in a fixed polyhedron. This allows us to compute the next mode m' from the current mode m, the current state $(x_1, \ldots, x_d)$, and the current input $(u_1, \ldots, u_c)$, as m' is determined by the membership of $(x_1, \ldots, x_d, u_1, \ldots, u_c)$ in a finite union of polyhedra.

This allows us to build an NBA that accepts the graph of $f_{\mathcal{H}}$ for every given multi-mode linear map $\mathcal{H}$. □

Corollary 4. *The NNCS reachability problem is semi-decidable when the plant is restricted to multi-mode linear maps.*

5 Conclusion

In this paper, we studied the reachability problem for dynamical systems controlled by deep neural networks. We showed that, for the common ReLU activations, the problem is undecidable even when the plant is trivial and the network is restricted to integral weights and a singleton initial set; furthermore, we can either fix the input and output dimensions to 3 and the number of hidden layers to 6, or use a single hidden layer. We then turned to the question when the problem can be semi-decided; here we extended a recent encoding of neural networks in Büchi automata and showed that ω-regular plants as well as input and target sets are sufficient for a semi-decision procedure; as an example, we demonstrated that a model akin to linear hybrid automata is ω-regular.

Acknowledgments

We thank the participants of AISoLA 2023 for suggesting to study the NNCS reachability problem with one hidden layer.

This research was partly supported by the Independent Research Fund Denmark under reference number 10.46540/3120-00041B, DIREC - Digital Research Centre Denmark under reference number 9142-0001B, and the Villum Investigator Grant S4OS under reference number 37819.

Bibliography

[1] Althoff, M.: An introduction to CORA 2015. In: ARCH. EPiC Series in Computing, vol. 34, pp. 120–151. EasyChair (2015), https://doi.org/10.29007/zbkv

[2] Althoff, M., Frehse, G., Girard, A.: Set propagation techniques for reachability analysis. Annu. Rev. Control. Robotics Auton. Syst. **4**, 369–395 (2021), https://doi.org/10.1146/annurev-control-071420-081941

[3] Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Hybrid Systems. LNCS, vol. 736, pp. 209–229. Springer (1992), https://doi.org/10.1007/3-540-57318-6_30

[4] Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. In: ICLR. OpenReview.net (2018), https://openreview.net/forum?id=B1J_rgWRW

[5] Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., Schilling, C.: JuliaReach: a toolbox for set-based reachability. In: HSCC. pp. 39–44. ACM (2019), https://doi.org/10.1145/3302504.3311804

[6] Cabessa, J.: Turing complete neural computation based on synaptic plasticity. PloS one **14**(10), e0223451 (2019), https://doi.org/10.1371/journal.pone.0223451

[7] Doyle, J.C., Francis, B.A., Tannenbaum, A.R.: Feedback control theory. Dover Publications (2013)

[8] Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016), http://www.deeplearningbook.org/

[9] Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**(1), 94–124 (1998), https://doi.org/10.1006/jcss.1998.1581

[10] Hyötyniemi, H.: Turing machines are recurrent neural networks. STeP **96**, 13–24 (1996)

[11] Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: CAV. LNCS, vol. 10426, pp. 97–117. Springer (2017), https://doi.org/10.1007/978-3-319-63387-9_5

[12] Kochdumper, N., Schilling, C., Althoff, M., Bak, S.: Open- and closed-loop neural network verification using polynomial zonotopes. In: NFM. LNCS, vol. 13903, pp. 16–36. Springer (2023), https://doi.org/10.1007/978-3-031-33170-1_2

[13] Koiran, P., Moore, C.: Closed-form analytic maps in one and two dimensions can simulate universal turing machines. Theor. Comput. Sci. **210**(1), 217–223 (1999), https://doi.org/10.1016/S0304-3975(98)00117-0

[14] Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Safe and optimal adaptive cruise control. In: Correct System Design. LNCS, vol. 9360, pp. 260–277. Springer (2015), https://doi.org/10.1007/978-3-319-23506-6_17
[15] Le, D.M., Greene, M.L., Makumi, W.A., Dixon, W.E.: Real-time modular deep neural network-based adaptive control of nonlinear systems. IEEE Control. Syst. Lett. **6**, 476–481 (2022), https://doi.org/10.1109/LCSYS.2021.3081361
[16] Liu, C., Arnon, T., Lazarus, C., Strong, C.A., Barrett, C.W., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. Found. Trends Optim. **4**(3-4), 244–404 (2021), https://doi.org/10.1561/2400000035
[17] Lopez, D.M., Althoff, M., Forets, M., Johnson, T.T., Ladner, T., Schilling, C.: ARCH-COMP23 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In: ARCH. EPiC Series in Computing, vol. 96. EasyChair (2023), https://doi.org/10.29007/x38n
[18] Lopez, D.M., Choi, S.W., Tran, H., Johnson, T.T.: NNV 2.0: The neural network verification tool. In: CAV. LNCS, vol. 13965, pp. 397–412. Springer (2023), https://doi.org/10.1007/978-3-031-37703-7_19
[19] Miller, W.T., Sutton, R.S., Werbos, P.J.: Neural networks for control. MIT press (1995)
[20] Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall (1967)
[21] Montúfar, G., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: NeurIPS. pp. 2924–2932 (2014), https://proceedings.neurips.cc/paper/2014/hash/109d2dd3608f669ca17920c511c2a41e-Abstract.html
[22] Moore, C.: Unpredictability and undecidability in dynamical systems. Phys. Rev. Lett. **64**(20), 2354–2357 (1990), https://doi.org/10.1103/PhysRevLett.64.2354
[23] Sälzer, M., Alsmann, E., Bruse, F., Lange, M.: Verifying and interpreting neural networks using finite automata. CoRR **abs/2211.01022** (2022), https://doi.org/10.48550/arXiv.2211.01022
[24] Sälzer, M., Lange, M.: Reachability in simple neural networks. Fundam. Informaticae **189**(3-4), 241–259 (2022), https://doi.org/10.3233/FI-222160
[25] Schilling, C., Forets, M., Guadalupe, S.: Verification of neural-network control systems by integrating Taylor models and zonotopes. In: AAAI. pp. 8169–8177. AAAI Press (2022), https://doi.org/10.1609/aaai.v36i7.20790
[26] Schilling, C., Lukina, A., Demirović, E., Larsen, K.G.: Safety verification of decision-tree policies in continuous time. In: NeurIPS. vol. 36, pp. 14750–14769. Curran Associates, Inc. (2023), https://proceedings.neurips.cc//paper_files/paper/2023/hash/2f89a23a19d1617e7fb16d4f7a049ce2-Abstract-Conference.html
[27] Siegelmann, H.T., Sontag, E.D.: Turing computability with neural nets. Applied Mathematics Letters **4**(6), 77–80 (1991), https://doi.org/10.1016/0893-9659(91)90080-F

[28] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: ICLR (2014), http://arxiv.org/abs/1312.6199