Three Analog Neurons Are Turing Universal

Jiří Šíma*

Institute of Computer Science, Czech Academy of Sciences,
P. O. Box 5, 18207 Prague 8, Czech Republic, sima@cs.cas.cz

Abstract. The languages accepted online by binary-state neural networks with rational weights have been shown to be context-sensitive when an extra analog neuron is added (1ANNs). In this paper, we provide an upper bound on the number of additional analog units to achieve Turing universality. We prove that any Turing machine can be simulated by a binary-state neural network extended with *three* analog neurons (3ANNs) having rational weights, with a linear-time overhead. Thus, the languages accepted offline by 3ANNs with rational weights are recursively enumerable, which refines the classification of neural networks within the Chomsky hierarchy.

Keywords: neural computing · Turing machine · Chomsky hierarchy

1 Introduction

The computational power of (recurrent) neural networks with the saturated-linear activation function[1] depends on the descriptive complexity of their weight parameters [12, 19]. Neural nets with *integer* weights, corresponding to binary-state networks, coincide with finite automata [1, 3, 4, 8, 16, 21]. *Rational* weights make the analog-state networks computationally equivalent to Turing machines [4, 14], and thus (by a real-time simulation [14]) polynomial-time computations of such networks are characterized by the complexity class P. Moreover, neural nets with arbitrary *real* weights can even derive "super-Turing" computational capabilities [12]. In particular, their polynomial-time computations correspond to the nonuniform complexity class P/poly while any input/output mapping (including undecidable problems) can be computed within exponential time [13]. In addition, a proper hierarchy of nonuniform complexity classes between P and P/poly has been established for polynomial-time computations of neural nets with increasing Kolmogorov complexity of real weights [2].

As can be seen, our understanding of the computational power of super-recursive (super-Turing) neural networks is satisfactorily fine-grained when changing from rational to arbitrary real weights. In contrast, there is still a gap between integer and rational weights which results in a jump from regular to recursively enumerable languages in the Chomsky hierarchy. In the effort of refining

* Research was done with institutional support RVO: 67985807 and partially supported by the grant of the Czech Science Foundation No. P202/12/G061.

[1] The results are valid for more general classes of activation functions [6, 11, 15, 22] including the logistic function [5].

the analysis of subrecursive neural nets we have introduced a model of binary-state networks extended with one extra analog-state neuron (1ANNs) [17], as already a few additional analog units allow for Turing universality [4, 14]. Although this model of 1ANNs has been inspired by theoretical issues, neural networks with different types of units/layers are widely used in practical applications, e.g. in deep learning [10] which also demands for the analysis.

In our previous work, we have characterized syntactically the class of languages accepted online by the 1ANNs [18] in terms of so-called cut languages [20]. The *online* input/output protocol means that a (potentially infinite) input word $\mathbf{x}$ is sequentially read symbol after symbol, each being processed with a constant-time overhead, while a neural network simultaneously signals via its output neuron whether the prefix of $\mathbf{x}$ that has been read so far, belongs to the underlying language [3, 21]. By using this characterization we have shown that the languages recognized online by the 1ANNs with rational weights are context-sensitive, and we have presented explicit examples of such languages that are not context-free. In addition, we have formulated a sufficient condition when a 1ANN accepts only a regular language in terms of quasi-periodicity of its real weight parameters. For example, 1ANNs with weights from the smallest field extension $\mathbb{Q}(\beta)$ over rational numbers including a Pisot number $\beta = 1/w$ (e.g. $\beta \in \mathbb{Z}$ is an integer or the plastic constant $\beta = \left(\sqrt[3]{9-\sqrt{69}} + \sqrt[3]{9+\sqrt{69}}\right) / \sqrt[3]{18} \approx 1.324718$) where w is the self-loop weight of the analog unit, have only a power of finite automata. These results refine the classification of subrecursive neural networks with the weights between integer and rational weights, within the Chomsky hierarchy.

A natural question arises concerning an upper bound on the number of extra analog units with rational weights that are sufficient for simulating a Turing machine. In this paper, we prove that any language accepted by a Turing machine in time $T(n)$ can be accepted offline by a binary-state neural network with *three* extra analog units (3ANNs) having rational weights in time $O(T(n))$. The offline input/output protocol assumes that an input word $\mathbf{x}$ of length n is read by a neural network at the beginning of a computation, either sequentially symbol after symbol in time $O(n)$, or $\mathbf{x}$ is already encoded in an initial state of an analog unit. The neural network then carries out its computation until it possibly halts and decides whether $\mathbf{x}$ belongs to the underlying language, which is indicated by its output neurons [14]. Thus, for rational weights, the languages accepted online by 1ANNs or offline by 3ANNs are context-sensitive or recursively enumerable, respectively, while the classification of offline 1ANNs or even 2ANNs (with two analog units) within the Chomsky hierarchy remains open for further research.

The proof exploits the classical technique of implementing two stacks of a pushdown automaton by two analog units, which is a model equivalent to Turing machine, including the encoding of stack contents based on a Cantor-like set [14]. In order to minimize the number of analog neurons, the first stack allows only for the `push` operation while the second one realizes the `top` and `pop` operations. To compensate for these restrictions, the third analog unit is introduced which is used to `swap` the contents of these two stacks by adding and subtracting their codes appropriately.

The paper is organized as follows. In Section 2, we introduce a formal model of binary-state neural networks with three extra analog units. Section 3 shows a simulation of Turing machine by a 3ANN with rational weights. We present some open problems in Section 4.

2 Neural Networks with Three Analog Units

In this section, we specify a formal computational model of *binary-state neural networks with three extra analog units* (shortly, 3ANN), $\mathcal{N}$, which will be used for simulating a Turing machine. The network $\mathcal{N}$ consists of $s \geq 3$ *units (neurons)*, indexed as $V = \{1, \ldots, s\}$. All the units in $\mathcal{N}$ are assumed to be binary-state (shortly *binary*) neurons except for the first three neurons $1, 2, 3 \in V$ which are analog-state (shortly *analog*) units. The neurons are connected into a directed graph representing an *architecture* of $\mathcal{N}$, in which each edge $(i, j) \in V^2$ leading from unit i to j is labeled with a real *weight* $w(i, j) \in \mathbb{R}$. The absence of a connection within the architecture corresponds to a zero weight between the respective neurons, and vice versa. The *computational dynamics* of $\mathcal{N}$ determines for each unit $j \in V$ its *state (output)* $y_j^{(t)}$ at discrete time instants $t = 0, 1, 2, \ldots$. The outputs $y_1^{(t)}, y_2^{(t)}, y_3^{(t)}$ from analog units $1, 2, 3 \in V$ are real numbers from the unit interval $\mathbb{I} = [0, 1]$, whereas the states $y_j^{(t)}$ of the remaining $s-3$ neurons $j \in V \setminus \{1, 2, 3\}$ are binary values from $\{0, 1\}$. This establishes the *network state* $\mathbf{y}^{(t)} = \left(y_1^{(t)}, y_2^{(t)}, y_3^{(t)}, y_4^{(t)}, \ldots, y_s^{(t)}\right) \in \mathbb{I}^3 \times \{0, 1\}^{s-3}$ at each discrete time instant $t \geq 0$.

Without loss of efficiency [9], we assume a synchronous fully parallel mode for notational simplicity. At the beginning of a computation, the neural network $\mathcal{N}$ is placed in an *initial state* $\mathbf{y}^{(0)} \in \mathbb{I}^3 \times \{0, 1\}^{s-3}$ which may also include an external input. At discrete time instant $t \geq 0$, an *excitation* of any neuron $j \in V$ is defined as $\xi_j^{(t)} = \sum_{i=0}^{s} w(i, j) y_i^{(t)}$, including a real *bias* value $w(0, j) \in \mathbb{R}$ which can be viewed as the weight from a formal constant unit input $y_0^{(t)} = 1$ for every $t \geq 0$ (i.e. $0 \in V$). At the next instant $t + 1$, all the neurons $j \in V$ compute their new outputs $y_j^{(t+1)}$ in parallel by applying an *activation function* $\sigma_j : \mathbb{R} \longrightarrow \mathbb{I}$ to $\xi_j^{(t)}$, $y_j^{(t+1)} = \sigma_j\left(\xi_j^{(t)}\right)$. The analog units $j \in \{1, 2, 3\}$ employ the *saturated-linear* function $\sigma_j(\xi) = \sigma(\xi)$ where

$$\sigma(\xi) = \begin{cases} 1 & \text{for } \xi \geq 1 \\ \xi & \text{for } 0 < \xi < 1 \\ 0 & \text{for } \xi \leq 0\,, \end{cases} \tag{1}$$

while for neurons $j \in V \setminus \{1, 2, 3\}$ with binary states $y_j \in \{0, 1\}$, the *Heaviside* activation function $\sigma_j(\xi) = H(\xi)$ is used where

$$H(\xi) = \begin{cases} 1 & \text{for } \xi \geq 0 \\ 0 & \text{for } \xi < 0\,. \end{cases} \tag{2}$$

In this way, the new network state $\mathbf{y}^{(t+1)} \in \mathbb{I}^3 \times \{0,1\}^{s-3}$ at time $t+1$ is determined.

The computational power of neural networks has been studied analogously to the traditional models of computations so that the networks are exploited as acceptors of formal languages $L \subseteq \Sigma^*$ [19]. For simplicity, we assume the binary alphabet $\Sigma = \{0,1\}$ and we use the following offline input/output protocol [14]. For a finite network $\mathcal{N}$, an input word (string) $\mathbf{x} = x_1 \dots x_n \in \{0,1\}^n$ of arbitrary length $n \geq 0$ can be encoded by the initial state of the analog *input* unit $\mathrm{inp} \in \{1,2,3\}$, using the encoding $\gamma' : \{0,1\}^* \longrightarrow \mathbb{I}$, that is,

$$y_{\mathrm{inp}}^{(0)} = \gamma'(x_1 \dots x_n)\,. \tag{3}$$

We assume that the encoding γ' could be evaluated by $\mathcal{N}$ in linear time $O(n)$ if $\mathbf{x}$ is read online and stored, bit after bit, in the state of inp. After $\mathcal{N}$ carries its computation deciding about the input word whether it belongs to L within the computational time of $T(n)$ updates, $\mathcal{N}$ halts and produces the result, which is indicated by the two neurons $\mathrm{halt}, \mathrm{out} \in V$ as

$$y_{\mathrm{halt}}^{(t)} = \begin{cases} 1 & \text{if } t = T(n) \\ 0 & \text{if } t \neq T(n) \end{cases} \qquad\qquad y_{\mathrm{out}}^{(T(n))} = \begin{cases} 1 & \text{if } \mathbf{x} \in L \\ 0 & \text{if } \mathbf{x} \notin L\,. \end{cases} \tag{4}$$

Note that the computation of $\mathcal{N}$ over $\mathbf{x}$ may not terminate. We say that a language $L \subseteq \{0,1\}^*$ is *accepted by 3ANN* $\mathcal{N}$, which is denoted by $L = \mathcal{L}(\mathcal{N})$ if for any input word $\mathbf{x} \in \{0,1\}^*$, $\mathbf{x} \in L$ iff $\mathcal{N}$ halts and accepts $\mathbf{x}$.

3 Simulating a Turing Machine

The following theorem shows how to simulate a Turing machine by a 3ANN with rational weights and a linear-time overhead.

Theorem 1. *Given a Turing machine $\mathcal{M}$ that accepts a language $L = \mathcal{L}(\mathcal{M})$ in time $T(n)$, there is a 3ANN $\mathcal{N}$ with rational weights, which accepts the same language $L = \mathcal{L}(\mathcal{N})$ in time $O(T(n))$.*

Proof. Without loss of generality, we assume that a given Turing machine $\mathcal{M}$ satisfies the following technical conditions. Its tape is arbitrarily extendable to the left and to the right, and the tape alphabet is $\{0,1\}$ which is sufficient for encoding the *blank* symbol uniquely (e.g. each symbol is encoded by two bits) so that there is the infinite string 0^ω to the left and to the right of the tape. At startup, $\mathcal{M}$ begins with an input word $\mathbf{x} = x_1 \dots x_n \in \{0,1\}^n$ written on the tape so that x_1 is under the tape head.

We will construct a 3ANN $\mathcal{N}$ with the set of neurons V, simulating the Turing machine $\mathcal{M}$ by using two stacks s_1 and s_2. One stack holds the contents of the tape to the left of the head of $\mathcal{M}$ while the other stack stores the right part of the tape. We assume that the first stack s_1 implements only the `push`(b) operation adding a given element b to the top of s_1, whereas the second stack

s_2 allows only for the `top` and `pop` operation which reads and removes the top element of s_2, respectively. In addition, the top element of s_2 models a symbol currently under the head of $\mathcal{M}$. In order to compensate for these restriction, we introduce the `swap` operation which exchanges the contents of s_1 and s_2, while the control unit remembers in its current state which part of the tape contents $c_{\rm cur} \in \{L, R\}$, either to the left of the head for $c_{\rm cur} = L$ or to the right for $c_{\rm cur} = R$, is stored in the second stack s_2.

We show how to implement one instruction of $\mathcal{M}$ by using the two stacks s_1 and s_2 and their operations `push`(b), `top`, `pop`, and `swap`. The transition function δ of $\mathcal{M}$ specifies for its current state $q_{\rm cur}$ and for a symbol x under the head, its new state $q_{\rm new}$, a symbol b to overwrite x, and a direction $d \in \{L, R\}$ for the tape head to move, which is either to the left for $d = L$ or to the right for $d = R$, that is, $\delta(q_{\rm cur}, x) = (q_{\rm new}, b, d)$. The transition from $q_{\rm cur}$ to $q_{\rm new}$ is realized by the control unit, while the tape update takes place in the stacks, for which we distinguish two cases, a so-called *short* and *long* instruction.

The short instruction applies when $d = c_{\rm cur}$. In this case, the two operations

$$\texttt{push}(b);\ \texttt{pop} \tag{5}$$

implement the corresponding update of the tape contents so that x under the head of $\mathcal{M}$ is overwritten by b, the head moves to a new symbol which is next in the desired direction d and appears at the top of s_2, while $c_{\rm new} = c_{\rm cur}$ is preserved. For the long instruction when $d \neq c_{\rm cur}$, the following sequence of five operations

$$\texttt{push}(\texttt{top});\ \texttt{pop};\ \texttt{swap};\ \texttt{push}(b);\ \texttt{pop} \tag{6}$$

is employed where the first two operations `push`$(\texttt{top})$; `pop` shift the current symbol $x = \texttt{top}$ under the head of $\mathcal{M}$ from the top of s_2 to the top of s_1. Then the `swap` operation exchanges the contents of s_1 and s_2 so that x is back at the top of s_2. Now, $c_{\rm new} = d \neq c_{\rm cur}$, which ensures the conversion to the previous case, and hence, the last two operations of (6) coincide with the short instruction (5).

The stacks s_1 and s_2 are implemented by analog neurons of $\mathcal{N}$ having the same name, $s_1, s_2 \in V$. The contents $\mathbf{a} = a_1 \ldots a_p \in \{0,1\}^*$ of stack s_k for $k \in \{1, 2\}$, where a_1 is the top element of s_k, is represented by the analog state of neuron s_k, using the encoding $\gamma : \{0,1\}^* \longrightarrow \mathbb{I}$,

$$y_{s_k} = \gamma\,(a_1 \ldots a_p) = \sum_{i=1}^{p} \frac{2^6\,(a_i + 1) - 1}{(2^7)^{i+1}} \in \left[0\,,\,\tfrac{1}{2^7}\right) \subset \mathbb{I}\,. \tag{7}$$

Note that the empty string ε is encoded by zero. All the possible analog state values generated by the encoding (7) create a Cantor-like set so that two strings with distinct top symbols are represented by two sufficiently separated numbers [14]. In particular, for $\mathbf{a} \neq \varepsilon$, we have

$$a_1 = \begin{cases} 0 & \text{if } \gamma\,(a_1 \ldots a_p) \in \left[\frac{2^6-1}{2^{14}}\,,\,\frac{1}{2^8}\right) \\ \\ 1 & \text{if } \gamma\,(a_1 \ldots a_p) \in \left[\frac{2^7-1}{2^{14}}\,,\,\frac{1}{2^7}\right)\,, \end{cases} \tag{8}$$

which can be used for reading the top element from the stack s_2 (which models a current tape symbol under the head of $\mathcal{M}$) by a binary neuron employing the Heaviside activation function (2):

$$\texttt{top} = 1 \quad \text{iff} \quad -1 + 2^8 y_{s_2} \geq 0\,. \tag{9}$$

Furthermore, the $\texttt{push}(b)$ and $\texttt{pop}$ operation can be implemented by the analog neuron s_1 and s_2, respectively, employing the linear part of the activation function (1), as

$$\begin{aligned} \texttt{push}(b):\, y_{s_1}^{\text{new}} = \xi_{s_1}^{\text{cur}} &= \frac{2^6\,(b+1)-1}{2^{14}} + \frac{1}{2^7}\cdot y_{s_1}^{\text{cur}} \\ &= \frac{2^6-1}{2^{14}} + \frac{1}{2^8}\cdot b + \frac{1}{2^7}\cdot y_{s_1}^{\text{cur}} \in \left[0\,,\,\tfrac{1}{2^7}\right) \qquad (10) \\ \texttt{pop}:\, y_{s_2}^{\text{new}} = \xi_{s_2}^{\text{cur}} &= \frac{1-2^6(\texttt{top}+1)}{2^7} + 2^7\cdot y_{s_2}^{\text{cur}} \\ &= \frac{1-2^6}{2^7} - \frac{1}{2}\cdot\texttt{top} + 2^7\cdot y_{s_2}^{\text{cur}} \in \left[0\,,\,\tfrac{1}{2^7}\right) \qquad (11) \end{aligned}$$

according to (7), where $y_{s_k}^{\text{new}}$ and $y_{s_k}^{\text{cur}}$ ($\xi_{s_k}^{\text{cur}}$) for $k \in \{1,2\}$, denotes the analog state (excitation) of neuron s_k, encoding the new and current contents of stack s_k, respectively.

According to [3], one can construct a binary-state (size-optimal) neural network $\mathcal{N}'$ with integer weights that implements the finite control of Turing machine $\mathcal{M}$ (i.e. a finite automaton). In particular, $\mathcal{N}'$ is a subnetwork of the 3ANN $\mathcal{N}$ with binary neurons in $V' \subset V$, which evaluates the transition function δ of $\mathcal{M}$ within four time steps by using the method of threshold circuit synthesis [7] (cf. [16]). Moreover, one can ensure that $\mathcal{N}'$ operates in the fully parallel mode by using the technique of [9]. Thus, $\mathcal{N}'$ holds internally a current state q_{cur} of $\mathcal{M}$ and receives a current symbol $x \in \{0,1\}$ under the tape head of $\mathcal{M}$ (which is stored at the top of stack s_2) via the neuron $\text{hd} \in V'$ implementing the $\texttt{top}$ operation. Then, $\mathcal{N}'$ computes $\delta\,(q_{\text{cur}}, x) = (q_{\text{new}}, b, d)$ within four computational steps, replaces the current state q_{cur} with q_{new}, and outputs a symbol $b \in \{0,1\}$ to overwrite x, via the neuron $\text{ow} \in V'$. In addition, $\mathcal{N}'$ holds a current value of $c_{\text{cur}} \in \{L, R\}$ which together with the calculated direction of head move $d \in \{L, R\}$, decides if a short or long instruction applies, depending on whether or not $c_{\text{cur}} = d$.

At the beginning of a computation, $\mathcal{N}'$ holds the initial state of $\mathcal{M}$ and the stacks s_1, s_2 contain the initial tape contents including an input word $\mathbf{x} = x_1 \ldots x_n \in \{0,1\}^n$ which is encoded by the input neuron $\text{inp} = s_2$ according to (3) and (7). Thus, $y_{s_1}^{(0)} = \gamma\,(0^\omega) = \sum_{i=1}^{\infty} \frac{2^6-1}{(2^7)^{i+1}} = \frac{2^6-1}{2^7(2^7-1)} = \frac{63}{16256} \in \left[0\,,\,\frac{1}{2^7}\right)$ and $y_{s_2}^{(0)} = \gamma'\,(\mathbf{x}) = \gamma\,(\mathbf{x}0^\omega) = \sum_{i=1}^{n} \frac{2^6(x_i+1)-1}{(2^7)^{i+1}} + \frac{2^6-1}{2^{n+1}(2^7-1)} \in \left[0\,,\,\frac{1}{2^7}\right)$, which $\mathcal{N}$ could evaluate in linear time $O(n)$ by using (10).

One computational step of $\mathcal{M}$ is simulated within one *macrostep* of $\mathcal{N}$ which takes 7 computational steps for a short instruction, while a long one consumes 18 steps of $\mathcal{N}$. Hereafter, the computational time t of $\mathcal{N}$ is related to the

macrostep. At the beginning of the macrostep when $t = 0$, the states of analog neurons s_1, s_2 encode the stack contents according to (7), that is, $y_{s_k}^{(0)} = z_k \in \left[0, \frac{1}{2^7}\right)$ for $k \in \{1, 2\}$. Then, $\mathcal{N}'$ reads the top element of s_2 via the neuron $\mathrm{hd} \in V'$ at time instant $t = 1$ of the macrostep, which is implemented by the integer weights

$$w(0, \mathrm{hd}) = -1\,, \qquad w(s_2, \mathrm{hd}) = 2^8\,, \tag{12}$$

implying $y_{\mathrm{hd}}^{(1)} = \mathtt{top}$ by (9). On the other hand, $\mathcal{N}'$ outputs a symbol $b \in \{0, 1\}$ to overwrite the current tape cell under the head via the neuron $\mathrm{ow} \in V'$ either at time instant $t = 6$ for a short instruction (i.e. $y_{\mathrm{ow}}^{(6)} = b$), or at time instant $t = 17$ for a long one (i.e. $y_{\mathrm{ow}}^{(17)} = b$), whereas the state of $\mathrm{ow} \in V'$ is 0 at other times, thus producing the sequence $0^5 b\, 0$ or $0^{16} b\, 0$, respectively.

We further extend $\mathcal{N}'$ with the four control neurons $c_1, c_2, c_3, c_4 \in V'$ for synchronizing the stack operations. Within each macrostep of $\mathcal{N}$, the control neurons c_1, c_2, c_3, c_4 produce the four sequences of binary output values, either 1111111, 1111011, 1111111, 0000010 of length 7 for a short instruction, or $1^4 01^{13}$, $1^{15} 01^2$, $1^6 01^{11}$, $0^5 10^{10} 10$ of length 18 for a long instruction, respectively, which can easily be implemented by a finite automaton and incorporated within $\mathcal{N}'$.

For realizing the stack operations, the binary neurons $\mathrm{pop}_1, \mathrm{pop}_2, \mathrm{bias} \in V$ and the third auxiliary analog unit $s_3 \in V$ are introduced in $\mathcal{N}$. In Table 1, the incoming rational weights to the neurons in $V \setminus V'$ are defined in the form of weight matrix with the entry $w(i, j) \in \mathbb{Q}$ in the ith row and jth column, where the analog neurons are separated from the binary ones by the double lines. For example, the weight of the connection from the control neuron $c_1 \in V'$ and from the analog neurons s_2 to $\mathrm{pop}_1 \in V \setminus V'$ is $w(c_1, \mathrm{pop}_1) = -2^3$ and $w(s_2, \mathrm{pop}_1) = 2^3$, respectively, whereas the bias of pop_1 is $w(0, \mathrm{pop}_1) = -1$.

We will verify the implementation of the long instruction including the short one, within one macrostep of $\mathcal{N}$ which is composed of 18 network state updates. The state evolution of neurons during the macrostep is presented in Table 2 which also shows the short instruction when the block bounded by the horizontal double lines corresponding to the time interval from $t = 6$ to $t = 16$ within the long instruction, is skipped. Moreover, alternatives for the short instruction are presented after the slash symbol, e.g. $t = 17/6$ means the seventeenth/sixth computational step of the long/short instruction within the macrostep.

Observe that for every $t = 1, \ldots, 18$ and $k \in \{1, 2\}$,

$$y_{\mathrm{pop}_k}^{(t)} = 0 \quad \text{if} \quad (k = 1 \,\&\, t \neq 6) \text{ or } (k = 2 \,\&\, t \neq 17) \tag{13}$$

since

$$\begin{aligned} \xi_{\mathrm{pop}_k}^{(t-1)} &= w(0, \mathrm{pop}_k) + w(c_k, \mathrm{pop}_k) y_{c_k}^{(t-1)} + w(s_2, \mathrm{pop}_k) y_{s_2}^{(t-1)} \\ &= -1 - 2^3 y_{c_k}^{(t-1)} + 2^3 y_{s_2}^{(t-1)}\,, \end{aligned} \tag{14}$$

by Table 1, reducing to $\xi_{\mathrm{pop}_k}^{(t-1)} = -1 - 2^3 + 2^3 y_{s_2}^{(t-1)} < 0$ for $y_{c_k}^{(t-1)} = 1$ which holds for $(k = 1 \,\&\, t \neq 6)$ or $(k = 2 \,\&\, t \neq 17)$. Similarly, we have

$$y_{\mathrm{bias}}^{(t)} = 1 \quad \text{iff} \quad y_{c_3}^{(t-1)} = 0 \quad \text{iff} \quad t = 8 \quad \text{for every } t = 1, \ldots, 18\,, \tag{15}$$

Table 1. The weight matrix with $w(i,j)$ in the ith row and jth column for $j \in V \setminus V'$

	pop_1	pop_2	bias	s_1	s_2	s_3
0	-1	-1	0	0	0	$\frac{1}{4}$
ow	0	0	0	$\frac{1}{2^8}$	0	0
c_1	-2^3	0	0	0	0	0
c_2	0	-2^3	0	0	0	0
c_3	0	0	-1	0	0	-5
c_4	0	0	0	$\frac{2^6-1}{2^{14}}$	$\frac{1-2^6}{2^7}$	0
pop_1	0	0	0	$\frac{1}{2^8}$	$-\frac{1}{2}$	0
pop_2	0	0	0	0	$-\frac{1}{2}$	0
bias	0	0	0	$-\frac{1}{4}$	$\frac{1}{4}$	0
s_1	0	0	0	$\frac{1}{2}$	0	$-\frac{1}{4}$
s_2	2^3	2^3	0	0	2	4
s_3	0	0	0	1	-1	0

because $\xi^{(t-1)}_{\text{bias}} = w(c_3, \text{bias}) y^{(t-1)}_{c_3} = -y^{(t-1)}_{c_3} \geq 0$ iff $y^{(t-1)}_{c_3} = 0$. Furthermore,

$$y^{(t)}_{s_3} = 0 \quad \text{if} \quad t \neq 8 \quad \text{for every } t = 1, \ldots, 18\,, \tag{16}$$

since $\xi^{(t-1)}_{s_3} = w(0, s_3) + w(c_3, s_3) y^{(t-1)}_{c_3} + w(s_1, s_3) y^{(t-1)}_{s_1} + w(s_2, s_3) y^{(t-1)}_{s_2} = \frac{1}{4} - 5y^{(t-1)}_{c_3} - \frac{1}{4} y^{(t-1)}_{s_1} + 4y^{(t-1)}_{s_2}$ which implies $\xi^{(t-1)}_{s_3} < 0$ for $y^{(t-1)}_{c_3} = 1$ holding for $t \neq 8$.

For a given symbol under the head of $\mathcal{M}$ held in $y^{(1)}_{\text{hd}}$ at time instant $t = 1$ according to (12), the binary-state subnetwork $\mathcal{N}'$ evaluates the transition function δ of $\mathcal{M}$ during four computational steps for $t = 2, 3, 4, 5$, deciding whether a long or short instruction occurs, which is indicated through the state $y^{(5)}_{c_1}$ of control neuron c_1 at time instant $t = 5$, that is, $y^{(5)}_{c_1} = 0$ iff a long instruction applies. In the meantime, the state of analog unit s_k for $k \in \{1, 2\}$, starting with $y^{(0)}_{s_k} = z_k \in \left[0, \frac{1}{2^7}\right)$, is multiplied by its self-loop weight $w(s_k, s_k)$ at each time instant $t = 1, \ldots, 6$, producing

$$y^{(t)}_{s_k} = w(s_k, s_k)^t z_k = \begin{cases} \frac{z_1}{2^t} \in \left[0, \frac{1}{2^{t+7}}\right) & \text{if } k = 1 \\ 2^t z_2 \in \left[0, \frac{1}{2^{7-t}}\right) & \text{if } k = 2 \end{cases} \quad \text{for } t = 0, \ldots, 6\,, \tag{17}$$

since $y^{(t)}_{\text{ow}} = y^{(t)}_{c_4} = y^{(t)}_{\text{pop}_1} = y^{(t)}_{\text{pop}_2} = y^{(t)}_{\text{bias}} = y^{(t)}_{s_3} = 0$ for every $t = 0, \ldots, 5$ due to (13), (15), and (16).

For a long instruction, we have $y^{(5)}_{c_1} = 0$ which implies

$$\xi^{(5)}_{\text{pop}_1} = -1 - 2^3 y^{(5)}_{c_1} + 2^3 y^{(5)}_{s_2} = -1 + 2^8 z_2 \tag{18}$$

Table 2. The macrostep of 3ANN $\mathcal{N}$ simulating one long/short instruction of TM $\mathcal{M}$

t	$y_{\mathrm{hd}}^{(t)}$	$y_{\mathrm{ow}}^{(t)}$	$y_{c_1}^{(t)}$	$y_{c_2}^{(t)}$	$y_{c_3}^{(t)}$	$y_{c_4}^{(t)}$	$y_{\mathrm{pop}_1}^{(t)}$	$y_{\mathrm{pop}_2}^{(t)}$	$y_{\mathrm{bias}}^{(t)}$	$y_{s_1}^{(t)}$	$y_{s_2}^{(t)}$	$y_{s_3}^{(t)}$
0		0	1	1	1	0	0	0	0	z_1	z_2	0
1	top	0	1	1	1	0	0	0	0	$\frac{z_1}{2}$	$2z_2$	0
2		0	1	1	1	0	0	0	0	$\frac{z_1}{2^2}$	2^2z_2	0
3		0	1	1	1	0	0	0	0	$\frac{z_1}{2^3}$	2^3z_2	0
4		0	1	1	1	0	0	0	0	$\frac{z_1}{2^4}$	2^4z_2	0
5		0	0/1	1/0	1	0	0	0	0	$\frac{z_1}{2^5}$	2^5z_2	0
6		0	1	1	1	1	top	0	0	$\frac{z_1}{2^6}$	2^6z_2	0
7		0	1	1	0	0	0	0	0	z_1' (19)	z_2' (20)	0
8		0	1	1	1	0	0	0	1	$\frac{z_1'}{2}$	$2z_2'$	$\frac{1}{4}-\frac{z_1'}{4}+4z_2'$
9		0	1	1	1	0	0	0	0	$4z_2'$	$\frac{z_1'}{4}$	0
10		0	1	1	1	0	0	0	0	$2z_2'$	$\frac{z_1'}{2}$	0
11		0	1	1	1	0	0	0	0	z_2'	z_1'	0
12		0	1	1	1	0	0	0	0	$\frac{z_2'}{2}$	$2z_1'$	0
13		0	1	1	1	0	0	0	0	$\frac{z_2'}{2^2}$	$2^2z_1'$	0
14		0	1	1	1	0	0	0	0	$\frac{z_2'}{2^3}$	$2^3z_1'$	0
15		0	1	1	1	0	0	0	0	$\frac{z_2'}{2^4}$	$2^4z_1'$	0
16		0	1	0	1	0	0	0	0	$\frac{z_2'}{2^5}$	$2^5z_1'$	0
17/6		b	1	1	1	1	0	top	0	$\frac{z_2'}{2^6}/\frac{z_1}{2^6}$	$2^6z_1'/2^6z_2$	0
$18/7\equiv 0$		0	1	1	1	0	0	0	0	z_1'' (29)	z_2'' (30)	0

according to (14) and (17). Hence, $y_{\mathrm{pop}_1}^{(6)} = \mathtt{top}$ by (9), which gives

$$\begin{aligned} y_{s_1}^{(7)} &= w(\mathrm{ow}, s_1)y_{\mathrm{ow}}^{(6)} + w(c_4, s_1)y_{c_4}^{(6)} + w(\mathrm{pop}_1, s_1)y_{\mathrm{pop}_1}^{(6)} \\ &\quad + w(\mathrm{bias}_1, s_1)y_{\mathrm{bias}}^{(6)} + w(s_1, s_1)y_{s_1}^{(6)} + w(s_3, s_1)y_{s_3}^{(6)} \\ &= \frac{2^6-1}{2^{14}} + \frac{1}{2^8}\cdot\mathtt{top} + \frac{z_1}{2^7} = z_1' \in \left[0\,,\,\tfrac{1}{2^7}\right) \end{aligned} \tag{19}$$

by Table 1, since $y_{\mathrm{ow}}^{(6)} = y_{\mathrm{bias}}^{(6)} = y_{s_3}^{(6)} = 0$, $y_{c_4}^{(6)} = 1$, and $y_{s_1}^{(6)} = \frac{z_1}{2^6}$ due to (15), (16), and (17). It follows from (10) and (19) that z_1' encodes the contents of the stack s_1 after the first operation $\mathtt{push}(\mathtt{top})$ of long instruction (6) has been applied to $\gamma^{-1}(z_1)$. Similarly,

$$\begin{aligned} y_{s_2}^{(7)} &= w(c_4, s_2)y_{c_4}^{(6)} + w(\mathrm{pop}_1, s_2)y_{\mathrm{pop}_1}^{(6)} + w(\mathrm{pop}_2, s_2)y_{\mathrm{pop}_2}^{(6)} + w(s_2, s_2)y_{s_2}^{(6)} \\ &= \frac{1-2^6}{2^7} - \frac{1}{2}\cdot\mathtt{top} + 2^7z_2 = z_2' \in \left[0\,,\,\tfrac{1}{2^7}\right) \end{aligned} \tag{20}$$

due to $y^{(6)}_{\mathrm{pop}_2} = 0$ and $y^{(6)}_{s_2} = 2^6 z_2$ by (13) and (17), respectively. According to (11) and (20), we thus know that z'_2 encodes the contents of the stack s_2 after the second operation `pop` of long instruction (6) has been applied to $\gamma^{-1}(z_2)$.

The `swap` operation starts at time instant $t = 8$ when

$$y^{(8)}_{s_1} = w(s_1, s_1) y^{(7)}_{s_1} = \frac{z'_1}{2} \in \left[0\,,\, \tfrac{1}{2^8}\right) \tag{21}$$

$$y^{(8)}_{s_2} = w(s_2, s_2) y^{(7)}_{s_2} = 2z'_2 \in \left[0\,,\, \tfrac{1}{2^6}\right) \tag{22}$$

$$\begin{aligned} y^{(8)}_{s_3} &= w(0, s_3) + w(s_1, s_3) y^{(7)}_{s_1} + w(s_2, s_3) y^{(7)}_{s_2} \\ &= \frac{1}{4} - \frac{z'_1}{4} + 4z'_2 \in \left[\tfrac{2^7-1}{2^9}\,,\, \tfrac{2^3+1}{2^5}\right) \end{aligned} \tag{23}$$

according to (19), (20), and Table 1, since $y^{(7)}_{\mathrm{ow}} = y^{(7)}_{c_3} = y^{(7)}_{c_4} = y^{(7)}_{\mathrm{pop}_1} = y^{(7)}_{\mathrm{pop}_2} = y^{(7)}_{\mathrm{bias}} = 0$ due to (13) and (15). At time instant $t = 9$, we have

$$\begin{aligned} y^{(9)}_{s_1} &= w(\mathrm{bias}, s_1) y^{(8)}_{\mathrm{bias}} + w(s_1, s_1) y^{(8)}_{s_1} + w(s_3, s_1) y^{(8)}_{s_3} \\ &= -\frac{1}{4} + \frac{z'_1}{4} + \frac{1}{4} - \frac{z'_1}{4} + 4z'_2 = 4z'_2 \in \left[0\,,\, \tfrac{1}{2^5}\right) \end{aligned} \tag{24}$$

$$\begin{aligned} y^{(9)}_{s_2} &= w(\mathrm{bias}, s_2) y^{(8)}_{\mathrm{bias}} + w(s_2, s_2) y^{(8)}_{s_2} + w(s_3, s_2) y^{(8)}_{s_3} \\ &= \frac{1}{4} + 4z'_2 - \frac{1}{4} + \frac{z'_1}{4} - 4z'_2 = \frac{z'_1}{4} \in \left[0\,,\, \tfrac{1}{2^9}\right) \end{aligned} \tag{25}$$

by (21)–(23) and Table 1, since $y^{(8)}_{\mathrm{ow}} = y^{(8)}_{c_4} = y^{(8)}_{\mathrm{pop}_1} = y^{(8)}_{\mathrm{pop}_2} = 0$ and $y^{(8)}_{\mathrm{bias}} = 1$ due to (13) and (15). This means that the respective multiples of z'_1 and z'_2 are exchanged between s_1 and s_2, cf. (21), (22) and (24), (25), respectively. Similarly to (17), the state of analog unit s_k for $k \in \{1, 2\}$, starting with $y^{(9)}_{s_k}$ in (24) and (25), respectively, is further multiplied by its self-loop weight $w(s_k, s_k)$ at each time instant $t = 10, \ldots, 17$, producing

$$y^{(t)}_{s_k} = w(s_k, s_k)^t z_k = \begin{cases} \frac{z'_2}{2^{t-11}} \in \left[0, \frac{1}{2^{t-4}}\right) & \text{if } k = 1 \\ 2^{t-11} z'_1 \in \left[0, \frac{1}{2^{18-t}}\right) & \text{if } k = 2 \end{cases} \quad \text{for } t = 9, \ldots, 17\,, \tag{26}$$

since $y^{(t)}_{\mathrm{ow}} = y^{(t)}_{c_4} = y^{(t)}_{\mathrm{pop}_1} = y^{(t)}_{\mathrm{pop}_2} = y^{(t)}_{\mathrm{bias}} = y^{(t)}_{s_3} = 0$ for every $t = 9, \ldots, 16$ due to (13), (15), and (16). Thus, the `swap` operation is finished at time instant $t = 11$ when $y^{(11)}_{s_1} = z'_2$ and $y^{(11)}_{s_2} = z'_1$.

Analogously to (18), $y^{(16)}_{c_2} = 0$ ensures

$$\xi^{(16)}_{\mathrm{pop}_2} = -1 - 2^3 y^{(16)}_{c_2} + 2^3 y^{(16)}_{s_2} = -1 + 2^8 z'_1 \tag{27}$$

according to (14) and (26), which implies

$$y^{(17)}_{\mathrm{pop}_2} = \mathtt{top} \tag{28}$$

by (9). At time instant $t = 18$, (19) reads as

$$\begin{aligned} y_{s_1}^{(18)} &= w(\mathrm{ow}, s_1) y_{\mathrm{ow}}^{(17)} + w(c_4, s_1) y_{c_4}^{(17)} + w(s_1, s_1) y_{s_1}^{(17)} \\ &= \frac{2^6 - 1}{2^{14}} + \frac{1}{2^8} \cdot b + \frac{z_2'}{2^7} = z_1'' \in \left[0\,,\, \tfrac{1}{2^7}\right) , \end{aligned} \tag{29}$$

since $y_{\mathrm{pop}_1}^{(17)} = y_{\mathrm{bias}}^{(17)} = y_{s_3}^{(17)} = 0$, $y_{\mathrm{ow}}^{(17)} = b$, $y_{c_4}^{(17)} = 1$, and $y_{s_1}^{(17)} = \frac{z_2'}{2^6}$ due to (13), (15), (16), and (26). It follows from (10) and (29) that z_1'' encodes the contents of the stack s_1 after the fourth operation $\texttt{push}(b)$ of long instruction (6) has been applied to $\gamma^{-1}(z_2')$. Similarly to (20),

$$\begin{aligned} y_{s_2}^{(18)} &= w(c_4, s_2) y_{c_4}^{(17)} + w(\mathrm{pop}_2, s_2) y_{\mathrm{pop}_2}^{(17)} + w(s_2, s_2) y_{s_2}^{(17)} \\ &= \frac{1 - 2^6}{2^7} - \frac{1}{2} \cdot \texttt{top} + 2^7 z_1' = z_2'' \in \left[0\,,\, \tfrac{1}{2^7}\right) \end{aligned} \tag{30}$$

by (26) and (28). According to (11) and (30), we thus know that z_2'' encodes the contents of the stack s_2 after the fifth operation $\texttt{pop}$ of (6) has been applied to $\gamma^{-1}(z_1')$, which completes the macrostep of $\mathcal{N}$ for a long instruction. For a short instruction when $y_{c_1}^{(5)} = 1$, $y_{c_2}^{(5)} = 0$, $y_{s_1}^{(5)} = \frac{z_1}{2^5}$ and $y_{s_2}^{(5)} = 2^5 z_2$, which coincides with a long instruction at time instant $t = 16$, the $\texttt{push}(b)$ and $\texttt{pop}$ operations of (5) are implemented analogously.

Finally, if $\mathcal{M}$ reaches its final state at the computational time $T(n)$ and accepts the input word $\mathbf{x}$, then this is indicated by the two neurons $\mathrm{halt}, \mathrm{out} \in V'$ of subnetwork $\mathcal{N}'$ during the macrostep $T(n)$ according to (4). Hence, $\mathcal{N}$ simulates $\mathcal{M}$ in time $O(T(n))$ because each macrostep takes only constant number of network's updates, which completes the proof of the theorem. □

4 Conclusion

In this paper, we have achieved an upper bound on the number of extra analog units that are sufficient to make binary-state neural networks Turing universal. We have proven that three additional analog neurons with rational weights suffices for simulating a Turing machine with a linear-time overhead, which complements the lower bound that neural networks with one extra analog unit and rational weights accept online context-sensitive languages. It is an open question whether the upper bound can be improved, that is, if only one or two extra rational-weight analog units suffice for simulating any Turing machine.

Another challenge for further research is to generalize the characterization of languages [18] that are accepted offline to 2ANNs employing two (or even more) extra analog units. Nevertheless, the ultimate goal is to prove a proper "natural" hierarchy of neural networks between integer and rational weights similarly as it is known between rational and real weights [2] and possibly, map it to known hierarchies of regular/context-free languages. This problem is related to a more general issue of finding suitable complexity measures of subrecursive neural networks establishing the complexity hierarchies, which could be employed in practical neurocomputing, e.g. the precision of weight parameters, energy complexity [16], temporal coding etc.

References

1. Alon, N., Dewdney, A.K., Ott, T.J.: Efficient simulation of finite automata by neural nets. Journal of the ACM **38**(2), 495–514 (1991)
2. Balcázar, J.L., Gavaldà, R., Siegelmann, H.T.: Computational power of neural networks: A characterization in terms of Kolmogorov complexity. IEEE Transactions on Information Theory **43**(4), 1175–1183 (1997)
3. Horne, B.G., Hush, D.R.: Bounds on the complexity of recurrent neural network implementations of finite state machines. Neural Networks **9**(2), 243–252 (1996)
4. Indyk, P.: Optimal simulation of automata by neural nets. In: Proceedings of the STACS 1995 Twelfth Annual Symposium on Theoretical Aspects of Computer Science. LNCS, vol. 900, pp. 337–348 (1995)
5. Kilian, J., Siegelmann, H.T.: The dynamic universality of sigmoidal neural networks. Information and Computation **128**(1), 48–56 (1996)
6. Koiran, P.: A family of universal recurrent networks. Theoretical Computer Science **168**(2), 473–480 (1996)
7. Lupanov, O.B.: On the synthesis of threshold circuits. Problemy Kibernetiki **26**, 109–140 (1973)
8. Minsky, M.: Computations: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
9. Orponen, P.: Computing with truly asynchronous threshold logic networks. Theoretical Computer Science **174**(1-2), 123–136 (1997)
10. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks **61**, 85–117 (2015)
11. Siegelmann, H.T.: Recurrent neural networks and finite automata. Journal of Computational Intelligence **12**(4), 567–574 (1996)
12. Siegelmann, H.T.: Neural Networks and Analog Computation: Beyond the Turing Limit. Birkhäuser, Boston (1999)
13. Siegelmann, H.T., Sontag, E.D.: Analog computation via neural networks. Theoretical Computer Science **131**(2), 331–360 (1994)
14. Siegelmann, H.T., Sontag, E.D.: On the computational power of neural nets. Journal of Computer System Science **50**(1), 132–150 (1995)
15. Šíma, J.: Analog stable simulation of discrete neural networks. Neural Network World **7**(6), 679–686 (1997)
16. Šíma, J.: Energy complexity of recurrent neural networks. Neural Computation **26**(5), 953–973 (2014)
17. Šíma, J.: The power of extra analog neuron. In: Proceedings of the TPNC 2014 Third International Conference on Theory and Practice of Natural Computing. LNCS, vol. 8890, pp. 243–254 (2014)
18. Šíma, J.: Neural networks between integer and rational weights. In: Proceedings of the IJCNN 2017 Thirties International Joint Conference on Neural Networks. pp. 154–161. IEEE (2017)
19. Šíma, J., Orponen, P.: General-purpose computation with neural networks: A survey of complexity theoretic results. Neural Computation **15**(12), 2727–2778 (2003)
20. Šíma, J., Savický, P.: Quasi-periodic β-expansions and cut languages. Theoretical Computer Science **720**, 1–23 (2018)
21. Šíma, J., Wiedermann, J.: Theory of neuromata. Journal of the ACM **45**(1), 155–178 (1998)
22. Šorel, M., Šíma, J.: Robust RBF finite automata. Neurocomputing **62**, 93–110 (2004)