The Computational Power of Interactive Recurrent Neural Networks

Jérémie Cabessa
jcabessa@nhrg.org
Hava T. Siegelmann
hava@cs.umass.edu
BINDS Lab, Computer Science Department, University of Massachusetts Amherst, Amherst, MA 01003-9264, U.S.A.

In classical computation, rational- and real-weighted recurrent neural networks were shown to be respectively equivalent to and strictly more powerful than the standard Turing machine model. Here, we study the computational power of recurrent neural networks in a more biologically oriented computational framework, capturing the aspects of sequential interactivity and persistence of memory. In this context, we prove that so-called interactive rational- and real-weighted neural networks show the same computational powers as interactive Turing machines and interactive Turing machines with advice, respectively. A mathematical characterization of each of these computational powers is also provided. It follows from these results that interactive real-weighted neural networks can perform uncountably many more translations of information than interactive Turing machines, making them capable of super-Turing capabilities.

1 Introduction

Understanding the computational and dynamical capabilities of neural networks is an issue of central importance. In this context, much interest has been focused on comparing the computational power of diverse theoretical neural models and abstract computing devices.

The approach was initiated by McCulloch and Pitts (1943), who proposed a modelization of the nervous system as a finite interconnection of logical devices. Neural networks were then considered as discrete abstract machines, and the issue of their computational capabilities was investigated from the automata-theoretic perspective. In this context, Kleene (1956) and Minsky (1967) proved that rational-weighted recurrent neural networks equipped with Boolean activation functions are computationally equivalent to classical finite state automata. Later, Siegelmann and Sontag (1995) showed that extending the activation functions of the cells from Boolean to linear-sigmoid actually drastically increases the computational power of the networks from finite state automata up to Turing capabilities. Kilian and Siegelmann (1996) then generalized the Turing universality of neural networks to a broader class of sigmoidal activation functions. The computational equivalence between so-called rational recurrent neural networks and Turing machines has now become a standard result in the field.

Siegelmann and Sontag (1994) achieved a further breakthrough by considering the computational power of recurrent neural networks from the perspective of analog computation (Siegelmann, 1999). They introduced the concept of an analog recurrent neural network as a classical linear-sigmoid neural net equipped with real- instead of rational-weighted synaptic connections. This analog information processing model turns out to be capable of capturing the nonlinear dynamical properties that are most relevant to brain dynamics, such as Cantor-like encoding and rich chaotic behaviors (Tsuda, 2001, 2009; Yamaguti, Kuroda, Fukushima, Tsukada, & Tsuda, 2011). Moreover, many dynamical and idealized chaotic systems that cannot be described by the universal Turing machine are also well captured within this analog framework (Siegelmann, 1995). In this context, Siegelmann and Sontag (1994) notably proved that the computational capabilities of analog recurrent neural networks turn out to stand beyond the Turing limits. These results support the idea that some dynamical and computational features of neurobiological systems might be beyond the scope of standard artificial models of computation.

However, until now, the issue of the computational capabilities of neural networks has always been considered from the strict perspective of Turing-like classical computation (Turing, 1936): a network is considered as an abstract machine that receives a finite input stream from its environment, processes this input, and then provides a corresponding finite output stream as the answer, without any consideration of the internal or external changes that might happen during previous computations. But this classical computational approach is inherently restrictive and has now been argued to "no longer fully correspond to the current notion of computing in modern systems" especially when it refers to bio-inspired complex information processing systems (van Leeuwen & Wiedermann, 2001a, 2008). Indeed, in the brain (or in organic life in general), information is processed in an interactive way, where previous experience must affect the perception of future inputs and older memories themselves may change with response to new inputs. Hence, neural networks should be conceived as performing sequential interactions or communications with their environments and be provided with memory that remains active throughout the whole computational process rather than proceeding in a closed-box amnesic classical fashion. Accordingly, we propose to study the computational power of recurrent neural networks from the rising perspective of interactive computation (Goldin, Smolka, & Wegner, 2006).

In this letter, we consider a basic paradigm of computation capturing the aspects of sequential interactivity and persistence of memory, and we study the computational power of recurrent neural networks in this context. Our framework is in line with previous ones suggested, for instance, by Goldin, Smolka, Attie, and Sonderegger (2004) and van Leeuwen and Wiedermann (2006), but focused on biological computational considerations. In section 2, we state some preliminary definitions. In section 3, we present the interactive computational paradigm that we consider. In sections 4 and 5, we define the concept of an interactive recurrent neural network and further show that under our interactive computational scenario, the rational- and real-weighted neural networks show the same computational powers as interactive Turing machines and interactive Turing machines with advice, respectively. Moreover, we provide a mathematical characterization of each of these computational powers. It follows from these results that in the interactive just as in the classical framework, analog (i.e., real-weighted) neural networks are capable of super-Turing computational capabilities. Sections 6 and 7 are devoted to the proofs of these results. Finally, section 8 provides some concluding remarks.

2 Preliminaries

Before entering into further considerations, we introduce the following definitions and notations. Given some finite alphabet Σ, we let Σ^*, Σ^+, Σ^n, and Σ^ω denote, respectively, the sets of finite words, nonempty finite words, finite words of length n, and infinite words, all of them over alphabet Σ. We also let $\Sigma^{\leq\omega} = \Sigma^* \cup \Sigma^\omega$ be the set of all possible words (finite or infinite) over Σ. The empty word is denoted λ.

For any $x \in \Sigma^{\leq\omega}$, the length of x is denoted by $|x|$ and corresponds to the number of letters contained in x. If x is nonempty, we let $x(i)$ denote the $(i+1)$th letter of x, for any $0 \leq i < |x|$. The prefix $x(0)\cdots x(i)$ of x is denoted by $x[0{:}i]$ for any $0 \leq i < |x|$. For any $x \in \Sigma^*$ and $y \in \Sigma^{\leq\omega}$, the fact that x is a prefix (resp. strict prefix) of y is denoted by $x \subseteq y$ (resp. $x \subsetneq y$). If $x \subseteq y$, we let $y - x = y(|x|)\cdots y(|y|-1)$ be the suffix of y that is not common to x (we have $y - x = \lambda$ if $x = y$). Moreover, the concatenation of x and y is denoted by $x \cdot y$, or sometimes simply by xy. The word x^n consists of n copies of x concatenated together, with the convention that $x^0 = \lambda$.

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is called monotone if the relation $x \subseteq y$ implies $f(x) \subseteq f(y)$, for all $x, y \in \Sigma^*$. It is called recursive if it can be computed by some Turing machine. Throughout this letter, any function $\varphi : \Sigma^\omega \longrightarrow \Sigma^{\leq\omega}$ will be referred to as an *ω-translation*.

3 Interactive Computation

3.1 The Interactive Paradigm. *Interactive computation* refers to the computational framework where systems may react or interact with each other as well as with their environment during the computation (Goldin et al., 2006). This paradigm was theorized in contrast to classical computation, which proceeds in a closed-box fashion and was argued to "no longer fully corresponds to the current notions of computing in modern systems" (van Leeuwen & Wiedermann, 2008). Interactive computation also provides a particularly appropriate framework for the consideration of natural and bio-inspired complex information processing systems (van Leeuwen & Wiedermann, 2001a, 2008).

In fact, Goldin and Wegner (2008), as well as Wegner (1997, 1998), argued that the intrinsic nature of interactivity shall alone lead to computations beyond the expressiveness of classical Turing machines. Goldin (2000) and Goldin et al. (2004) then introduced the concept of a persistent Turing machine as a possible extension of the classical notion of Turing machine in the interactive context. Van Leeuwen and Wiedermann (2001a), however, consider that "interactivity alone is not sufficient to break the Turing barrier." They introduced the concepts of an interactive Turing machine and an interactive Turing machine with advice as a generalization of their classical counterparts in the interactive context and used them as a tool to analyze the computational power of other interactive systems. In this context, they showed that several interactive models of computation are capable of super-Turing computational capabilities (van Leeuwen & Wiedermann, 2001a, 2001b).

The general interactive computational paradigm consists of a step-by-step exchange of information between a system and its environment. In order to capture the unpredictability of next inputs at any time step, the dynamically generated input streams need to be modeled by potentially infinite sequences of symbols (the case of finite sequences of symbols would necessarily reduce to the classical computational framework) (Wegner, 1998; van Leeuwen & Wiedermann, 2008). Hence, the interactive system receives a potentially infinite input stream of signals bit by bit and produces a corresponding potentially infinite output stream of signals bit by bit. At every time step, the current input bit might depend on intermediate outputs or external sources, and the corresponding output bit depends on the current input as well as on the current internal state of the system. It follows that every output depends on the whole input history that has been processed so far. In this sense, the memory of the system remains active throughout the whole computational process.

Throughout this letter, we consider a basic interactive computational scenario where at every time step, the environment first sends a nonempty input bit to the system (full environment activity condition); the system next updates its current state accordingly and then answers by either producing a corresponding output bit or remaining silent. In other words, the system is not obliged to provide corresponding output bits at every time step, but might instead stay silent for a while (to express the need of some internal computational phase before outputting a new bit) or even forever (to express the case that it has died). Consequently, after infinitely many time steps, the system will have received an infinite sequence of consecutive input bits and translated it into a corresponding finite or infinite sequence of not necessarily consecutive output bits. Accordingly, any interactive system $\mathcal{S}$ realizes an ω-translation $\varphi_{\mathcal{S}} : \{0,1\}^{\omega} \longrightarrow \{0,1\}^{\leq\omega}$.

3.2 Interactive Turing Machines. The concept of an interactive Turing machine was introduced by van Leeuwen and Wiedermann (2001a) as a generalization of the standard Turing machine model in the context of interactive computation.

An interactive Turing machine consists of an interactive abstract device driven by a standard Turing machine program. It receives an infinite stream of bits as input and produces a corresponding stream of bits as output step by step. The input and output bits are processed by corresponding input and output ports rather than tapes. Consequently, at every time step, the machine can no more operate on the output bits that have already been processed.[1] Furthermore, according to our interactive scenario, it is assumed that at every time step, the environment sends a nonsilent input bit to the machine, and the machine might either answer by some corresponding output bit or remain silent.

Formally, an interactive Turing machine (ITM) $\mathcal{M}$ is defined as a tuple $\mathcal{M} = (Q, \Gamma, \delta, q_0)$, where Q is a finite set of states, $\Gamma = \{0, 1, \lambda, \sharp\}$ is the alphabet of the machine, where $\sharp$ stands for the blank tape symbol, $q_0 \in Q$ is the initial state, and

$$\delta : Q \times \Gamma \times \{0,1\} \longrightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow, -\} \times \{0, 1, \lambda\}$$

is the transition function of the machine. The relation $\delta(q, x, b) = (q', x', d, b')$ means that if the machine $\mathcal{M}$ is in state q, the cursor of the tape is scanning the letter $x \in \{0, 1, \sharp\}$, and the bit $b \in \{0,1\}$ is currently received at its input port, then $\mathcal{M}$ will go in next state q'; it will make the cursor overwrite symbol x by $x' \in \{0, 1, \sharp\}$ and then move to direction d, and it will finally output symbol $b \in \{0, 1, \lambda\}$ at its output port, where λ represents the fact that the machine is not outputting any bit at that time step.

According to this definition, for any infinite input stream $s \in \{0,1\}^{\omega}$, we define the corresponding output stream $o_s \in \{0,1\}^{\leq\omega}$ of $\mathcal{M}$ as the finite or infinite subsequence of (non-λ) output bits produced by $\mathcal{M}$ after having processed input s. In this manner, any machine $\mathcal{M}$ naturally induces an ω-translation $\varphi_{\mathcal{M}} : \{0,1\}^{\omega} \longrightarrow \{0,1\}^{\leq\omega}$ defined by $\varphi_{\mathcal{M}}(s) = o_s$, for each $s \in \{0,1\}^{\omega}$. Finally, an ω-translation $\psi : \{0,1\}^{\omega} \longrightarrow \{0,1\}^{\leq\omega}$ is said to be realizable by some interactive Turing machine iff there exists an ITM $\mathcal{M}$ such that $\varphi_{\mathcal{M}} = \psi$.

Van Leeuwen and Wiedermann (2001a) also introduced the concept of an interactive machine with advice as a relevant nonuniform computational

[1] In fact, allowing the machine to erase its previous output bits would lead to the consideration of much more complicated ω-translations.

model in the context of interactive computation. Interactive Turing machines with advice are strictly more powerful than their classical counterpart (i.e., interactive Turing machines without advice) (van Leeuwen & Wiedermann, 2001b, proposition 5; van Leeuwen & Wiedermann, 2001a, lemma 1), and they were shown to be computationally equivalent to several other nonuniform models of interactive computation, like sequences of interactive finite automata, site machines, and Web Turing machines (van Leeuwen & Wiedermann, 2001a).

An interactive Turing machine with advice (ITM/A) $\mathcal{M}$ consists of an interactive Turing machine provided with an advice mechanism. The mechanism comes in the form of an advice function, which consists of a mapping α from $\mathbb{N}$ to $\{0, 1\}^*$. Moreover, the machine $\mathcal{M}$ uses two auxiliary special tapes, an advice input tape and an advice output tape, as well as a designated advice state. During its computation, $\mathcal{M}$ can write the binary representation of an integer m on its input tape, one bit at a time. Yet at time step n, the number m is not allowed to exceed n. Then at any chosen time, the machine can enter its designated advice state and have the string $\alpha(m)$ be written on the advice output tape in one time step, replacing the previous content of the tape. The machine can repeat this process as many times as it wants during its infinite computation.

Once again, according to our interactive scenario, any ITM/A $\mathcal{M}$ induces an ω-translation $\varphi_{\mathcal{M}} : \{0, 1\}^{\omega} \longrightarrow \{0, 1\}^{\leq\omega}$ which maps every infinite input stream s to its corresponding finite or infinite output stream o_s produced by $\mathcal{M}$. Finally, an ω-translation $\psi : \{0, 1\}^{\omega} \longrightarrow \{0, 1\}^{\leq\omega}$ is said to be realizable by some interactive Turing machine with advice iff there exists an ITM/A $\mathcal{M}$ such that $\varphi_{\mathcal{M}} = \psi$.

4 Interactive Recurrent Neural Networks

We consider a natural extension in the interactive framework of the classical model of recurrent neural network, as presented for instance in Siegelmann and Sontag (1994, 1995) and Siegelmann (1995, 1999). We will provide a characterization of the expressive powers of both rational- and real-weighted interactive recurrent neural networks.

A recurrent neural network (RNN) consists of a synchronous network of neurons (or processors) related together in a general architecture—not necessarily loop free or symmetric. The network contains a finite number of neurons $(x_j)_{j=1}^{N}$, as well as M parallel input lines carrying the input stream transmitted by the environment into M of the N neurons, and P designated output neurons among the N whose role is to communicate the output of the network to the environment. At each time step, the activation value of every neuron is updated by applying a linear-sigmoid function to some weighted affine combination of values of other neurons or inputs at a previous time step.

Formally, given the activation values of the internal and input neurons $(x_j)_{j=1}^N$ and $(u_j)_{j=1}^N$ at time t, the activation value of each neuron x_i at time $t+1$ is then updated by

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right), \quad i = 1, \dots, N, \tag{4.1}$$

where all a_{ij}, b_{ij}, and c_i are numbers describing the weighted synaptic connections and weighted bias of the network, and σ is the classical saturated-linear activation function defined by

$$\sigma(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \le x \le 1. \\ 1 & \text{if } x > 1 \end{cases}$$

A rational recurrent neural network (RNN[$\mathbb{Q}$]) denotes a recurrent neural net whose synaptic weights are rational numbers. A real (or analog) recurrent neural network (RNN[$\mathbb{R}$]) is a network whose synaptic weights are real. Since rational numbers are real, note that any RNN[$\mathbb{Q}$] is also a RNN[$\mathbb{R}$] by definition. The converse is obviously not true. In fact, it has been proven that RNN[$\mathbb{Q}$] are Turing equivalent and that RNN[$\mathbb{R}$]s are strictly more powerful than RNN[$\mathbb{Q}$]s and, hence, also than Turing machines (Siegelmann & Sontag, 1994, 1995).

In order to stay consistent with our interactive scenario, we define the notion of an interactive recurrent neural network (IRNN), which adheres to a rigid encoding of the way input and output are interactively processed between the environment and the network.

First, we assume that any IRNN is provided with a single input line u whose role is to transmit to the network the infinite input stream of bits sent by the environment. More precisely, at each time step $t \geq 0$, the input line u admits an activation value $u(t)$ belonging to $\{0, 1\}$ (the full environment activity conditions forces that $u(t)$ never equals λ). Furthermore, we suppose that any IRNN is equipped with two binary output lines, a data line y_d and a validation line y_v.[2] The role of the data line is to carry the output stream of the network, while the role of the validation line is to describe when the data line is active and when it is silent. Accordingly, the output stream transmitted by the network to the environment will be defined as the (finite or infinite) subsequence of successive data bits that occur simultaneously with positive validation bits.

[2]The binary requirement of the output lines y_d and y_v means that the network is designed such that for every input and every time step t, one has $y_d(t) \in \{0, 1\}$ and $y_v(t) \in \{0, 1\}$.

Note that the convention of using two output lines allows us to have all output signals be binary and hence stay close to the framework that Siegelmann and Sontag (1994) developed. Yet one could have used a single output processor y satisfying $y(t) \in \{-1, 0, 1\}$ for every $t \geq 0$, where $y(t) = 0$ means that no signal is present at time t, while $y(t) = \{-1, 1\}$ means that y is transmitting one of the two possible values at time t. The forthcoming results do not depend on the output encoding that we consider.

An interactive rational recurrent neural network (IRNN[$\mathbb{Q}$]) denotes an IRNN whose synaptic weights are all rational numbers and an interactive real (or analog) recurrent neural network (IRNN[$\mathbb{R}$]), an IRNN whose synaptic weights are all real.

If $\mathcal{N}$ is a rational- or real-weighted IRNN with initial activation values $x_i(0) = 0$ for $i = 1, \ldots, N$, then any infinite input stream,

$$s = s(0)s(1)s(2)\cdots \in \{0,1\}^{\omega},$$

transmitted to input line u induces, via equation 4.1, a corresponding pair of infinite streams:

$$\left(y_d(0)y_d(1)y_d(2)\cdots, y_v(0)y_v(1)y_v(2)\cdots\right) \in \{0,1\}^{\omega} \times \{0,1\}^{\omega}.$$

The output stream of $\mathcal{N}$ according to input s is then given by the finite or infinite subsequence o_s of successive data bits that occur simultaneously with positive validation bits:

$$o_s = \langle y_d(i) : i \in \mathbb{N} \text{ and } y_v(i) = 1\rangle \in \{0,1\}^{\leq\omega}.$$

Hence, any IRNN $\mathcal{N}$ naturally induces an ω-translation $\varphi_{\mathcal{N}}$: $\{0,1\}^{\omega} \longrightarrow \{0,1\}^{\leq\omega}$ defined by $\varphi_{\mathcal{N}}(s) = o_s$ for each $s \in \{0,1\}^{\omega}$. Finally, an ω-translation $\psi : \{0,1\}^{\omega} \longrightarrow \{0,1\}^{\leq\omega}$ is said to be realizable by some IRNN iff there exists some IRNN $\mathcal{N}$ such that $\varphi_{\mathcal{N}} = \psi$.

5 The Computational Power of Interactive Recurrent Neural Networks

This section states the main results of the letter. A complete characterization of the computational powers of IRNN[$\mathbb{Q}$]s and IRNN[$\mathbb{R}$]s is provided. More precisely, we show that IRNN[$\mathbb{Q}$]s and IRNN[$\mathbb{R}$]s are computationally equivalent to ITMs and ITM/As, respectively. Furthermore, we provide a precise mathematical characterization of the ω-translations realized by IRNN[$\mathbb{Q}$]s and IRNN[$\mathbb{R}$]s. From these results, it follows that IRNN[$\mathbb{R}$]s are strictly more powerful than ITMs, showing that the super-Turing computational capabilities of analog recurrent neural networks also hold in the framework of interactive computation (Siegelmann & Sontag, 1995).

5.1 The Classical Case. For clarity, we first recall the main results concerning the computational powers of recurrent neural networks in the case of classical computation. In this context, classical rational-weighted recurrent neural networks were proven to be computationally equivalent to Turing machines (Siegelmann & Sontag, 1995). Indeed, on the one hand, any function determined by equation 4.1 and involving rational weights is necessarily recursive, and thus can be computed by some Turing machine. And on the other hand, it was shown that any Turing machine can be simulated in linear time by some rational recurrent neural network. The result can be expressed as follows:

Theorem 1. *Let $L \subseteq \{0,1\}^{+}$ be some language. Then L is decidable by some RNN[$\mathbb{Q}$] if and only if L is decidable by some TM (i.e., iff L is recursive).*

Moreover, classical real-weighted recurrent neural networks were shown to be strictly more powerful than rational recurrent networks, and hence also than Turing machines. More precisely, they turn out to be capable of deciding all possible languages in an exponential time of computation. When restricted to polynomial time of computation, the networks decide precisely the complexity class of languages **P/poly**, that is, the set of all languages decidable in polynomial time by some Turing machine with polynomially long advice (Siegelmann & Sontag, 1994). Note that since **P/poly** strictly includes the class **P** and contains nonrecursive languages, it follows that the real networks are already capable of super-Turing computational power from the polynomial time of computation. These results are summarized in the following theorem:

Theorem 2. *Let $L \subseteq \{0,1\}^{+}$ be some language. Then L is decidable in exponential time by some RNN[$\mathbb{R}$]. Moreover, L is decidable in polynomial time by some RNN[$\mathbb{R}$] iff L is decidable in polynomial time by some Turing machine with polynomially long advice (i.e., iff $L \in$ **P/poly**).*

5.2 The Interactive Case. Similar to the classical framework, the main tools involved in the characterization of the computational powers of interactive neural networks are the concepts of an interactive Turing machine and an interactive Turing machine with advice. Yet in order to provide a mathematical description of that computational power, the following important relationship between monotone functions and ω-translations also needs to be introduced. More precisely, we note that any monotone function $f : \{0,1\}^{*} \longrightarrow \{0,1\}^{*}$ induces in the limit an ω-translation $f_{\omega} : \{0,1\}^{\omega} \longrightarrow \{0,1\}^{\leq\omega}$ defined by

$$f_{\omega}(x) = \lim_{i\geq 0} f(x[0{:}i]),$$

where $\lim_{i\geq 0} f(x[0{:}i])$ denotes the smallest finite word that contains each word of $\{f(x[0{:}i]) : i \geq 0\}$ as a finite prefix if $\lim_{i\to\infty} |f(x[0{:}i])| < \infty$, and $\lim_{i\geq 0} f(x[0{:}i])$ denotes the unique infinite word that contains each word of $\{f(x[0{:}i]) : i \geq 0\}$ as a finite prefix if $\lim_{i\to\infty} |f(x[0{:}i])| = \infty$ (whenever infinite, the word $\lim_{i\geq 0} f(x[0{:}i])$ is also generally denoted by $\bigcup_{i\geq 0} f(x[0{:}i])$; Kechris, 1995). Note that the monotonicity of f ensures that the value $f_\omega(x)$ is well defined for all $x \in \{0, 1\}^\omega$. Intuitively, the value $f_\omega(x)$ corresponds to the finite or infinite word that is ultimately approached by the sequence of growing prefixes $\langle f(x[0{:}i]) : i \geq 0\rangle$.

According to these definitions, in this letter, an ω-translation $\psi : \{0, 1\}^\omega \longrightarrow \{0, 1\}^{\leq\omega}$ will be called continuous[3] if there exists a monotone function $f : \{0, 1\}^* \longrightarrow \{0, 1\}^*$ such that $f_\omega = \psi$; it will be called recursive continuous if there exists a monotone and recursive function $f : \{0, 1\}^* \longrightarrow \{0, 1\}^*$ such that $f_\omega = \psi$.

We now come up to the computational power of interactive recurrent neural networks. More precisely, the following result shows that IRNN[$\mathbb{Q}$]s and ITMs have equivalent computational capabilities. The two models of computation actually realize the class of all ω-translations that can be obtained as limits of monotone recursive functions:

Theorem 3. *IRNN[$\mathbb{Q}$]s and ITMs have the same computational power. More precisely, for any ω-translation $\psi : \{0, 1\}^\omega \longrightarrow \{0, 1\}^{\leq\omega}$, the following conditions are equivalent:*

A. ψ is realizable by some IRNN[$\mathbb{Q}$].
B. ψ is realizable by some ITM.
C. ψ is recursive continuous.

Proof. The proof is a direct consequence of propositions 1 and 2 of section 6.

The next result describes the computational power of interactive real-weighted recurrent neural networks. It states that IRNN[$\mathbb{R}$]s and ITM/As have an equivalent computational power and realize precisely the class of all ω-translations that can be obtained as limits of monotone but not necessarily recursive functions:

Theorem 4. *IRNN[$\mathbb{R}$]s and ITM/As have the same computational power. More precisely, for any ω-translation $\psi : \{0, 1\}^\omega \longrightarrow \{0, 1\}^{\leq\omega}$, the following conditions are equivalent:*

A. ψ is realizable by some IRNN[$\mathbb{R}$].
B. ψ is realizable by some ITM/A.
C. ψ is continuous.

[3]The choice of this name comes from the fact that continuous functions over the Cantor space $\mathcal{C} = \{0, 1\}^\omega$ can be precisely characterized as limits of monotone functions. We chose to extend this term in the broader context here of functions from $\{0, 1\}^\omega$ to $\{0, 1\}^{\leq\omega}$ that can also be expressed as limits of monotone functions.

Proof. The proof is a direct consequence of propositions 3 and 4 in section 7.

Finally, it follows from the two preceding results that as for the case of classical computation, analog recurrent neural networks also have super-Turing computational capabilities in our context of interactive computation:

Theorem 5. *IRNN[$\mathbb{R}$]s are strictly more powerful than ITMs. More precisely, IRNN[$\mathbb{R}$]s can realize uncountably many more ω-translations than ITMs.*

Proof. We first recall that $\aleph_0$ and $2^{\aleph_0}$ denote the cardinalities of the sets of natural and real numbers, respectively, and that the difference set obtained by removing the natural numbers from the real numbers still has cardinality $2^{\aleph_0}$. Any ω-translation ψ realized by some ITM can obviously also be realized by some ITM/A, and hence also by some IRNN[$\mathbb{R}$]. It follows that IRNN[$\mathbb{R}$]s are at least as powerful as ITMs. Moreover, since there are $2^{\aleph_0}$ monotone functions from $\{0,1\}^*$ into $\{0,1\}^*$ but only $\aleph_0$ recursive monotone functions from $\{0,1\}^*$ into $\{0,1\}^*$, there are also $2^{\aleph_0}$ continuous ω-translations whereas only $\aleph_0$ recursive continuous ω-translations. Therefore, theorems 4C and 3C show that IRNN[$\mathbb{R}$]s can realize $2^{\aleph_0}$ many more ω-translations than ITMs.

Theorems 3 and 4 furnish a complete characterization of the computational powers of IRNN[$\mathbb{Q}$]s and IRNN[$\mathbb{R}$]s according to our interactive paradigm of computation. Theorem 5 shows further that IRNN[$\mathbb{R}$]s are actually super-Turing.

More precisely, the equivalence between conditions A and B of theorem 3 provides a proper generalization in our interactive context of the classical equivalence between RNN[$\mathbb{Q}$]s and TMs stated in theorem 1. The equivalence between conditions B and C of theorem 3 corresponds to the translation in the present computational context of the results of van Leeuwen and Wiedermann (2006, theorems 7 and 8) concerning the characterization of partial and total interactive ω-translations from $\{0,1\}^\omega$ to $\{0,1\}^\omega$ in terms of limits of monotone recursive functions. Furthermore, the equivalence between conditions A and B of theorem 4 provides some kind of interactive counterpart to the equivalence in polynomial time of computation between RNN[$\mathbb{R}$]s and TM/poly(A)s stated in theorem 2. In this case, the consideration of polynomial time of computation is no longer relevant since the systems perform a never-ending sequential interactive exchange of information. Condition C of theorem 4 provides a new precise mathematical characterization of the computational power of ITM/A and IRNN[$\mathbb{R}$]s.

Besides, following the approach of van Leeuwen and Wiedermann (2006), we could also have conceived interactive computing devices as performing partial ω-translations from $\{0,1\}^\omega$ to $\{0,1\}^\omega$ rather than total ω-translations from $\{0,1\}^\omega$ to $\{0,1\}^{\leq\omega}$. The partial ω-translation $\varphi_{\mathcal{D}}$ realized by some interactive device $\mathcal{D}$ would be simply defined by $\varphi_{\mathcal{D}}(s) = o_s$ if $o_s \in \{0,1\}^\omega$ and $\varphi_{\mathcal{D}}(s)$ undefined if $o_s \in \{0,1\}^*$, where $o_s \in \{0,1\}^{\leq\omega}$ corresponds to the output produced by $\mathcal{D}$ when receiving input $s \in \{0,1\}^\omega$. In this case, the computational equivalences between IRNN[$\mathbb{Q}$]s and ITMs, as well as between IRNN[$\mathbb{R}$]s and ITM/As, would remain valid, and hence the super-Turing capabilities of the IRNN[$\mathbb{R}$]s will hold true. Moreover, the partial ω-translations performed by ITM/As would correspond precisely to the partial functions $\varphi : \{0,1\}^\omega \longrightarrow \{0,1\}^\omega$ such that $dom(\varphi) \in \boldsymbol{\Pi}^0_2$ and $\varphi|_{dom(\varphi)} : dom(\varphi) \subseteq \{0,1\}^\omega \longrightarrow \{0,1\}^\omega$ is continuous in the classical sense (see Kechris, 1995, for a precise definition of $\boldsymbol{\Pi}^0_2$-sets and continuous functions in the Cantor space $\{0,1\}^\omega$).

6 IRNN[$\mathbb{Q}$]s and ITMs

This section is devoted to the proof of theorem 3. The following proposition establishes the equivalence between conditions B and C of theorem 3:

Proposition 1. *Let ψ be some ω-translation. Then ψ is realizable by some ITM iff ψ is recursive continuous.*

Proof. Let $\varphi_{\mathcal{M}}$ be an ω-translation realized by some ITM $\mathcal{M}$. We show that $\varphi_{\mathcal{M}}$ is recursive continuous. For this purpose, consider the function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$, which maps every finite word u to the unique corresponding finite word produced by $\mathcal{M}$ after $|u|$ steps of computation when $u \cdot x$ is provided as input bit by bit for any suffix $x \in \{0,1\}^\omega$. In other words, $f(u) =$ output string produced by $\mathcal{M}$ after $|u|$ time steps of computation on input $u \cdot x$, for any $x \in \{0,1\}^\omega$.

In order to see that f is well defined, we remark that the definition of f is independent of the choice of x. In fact, by definition of our interactive scenario, after the first $|u|$ time steps of computation, the machine $\mathcal{M}$ working on input $u \cdot x$ has received only the $|u|$ first bits of $u \cdot x$, namely, u which shows that its current output string is so far absolutely not influenced by the suffix x. Hence, the function f is well defined.

Now, since $\mathcal{M}$ is driven by the program of a TM, the function f can be computed by the classical TM $\mathcal{M}'$, which, on any finite input $u \in \{0,1\}^*$, works exactly like $\mathcal{M}$ during the $|u|$ first steps of computations and then halts. It follows that f is recursive. Moreover, if $u \subseteq v$, then since the definition of f is independent of the suffix x and since $u \cdot (v - u) = v$, the values $f(u)$ and $f(v)$ can be seen as the output strings produced by $\mathcal{M}$ after, respectively, $|u|$ and $|v|$ time steps of computation over the same input $u \cdot (v - u) \cdot x$, for some $x \in \{0,1\}^\omega$. Since $|u| \leq |v|$, one necessarily has $f(u) \subseteq f(v)$. Therefore, f is monotone.

We now prove that $\varphi_{\mathcal{M}} = f_\omega$. Given some input stream $s \in \{0,1\}^\omega$, we consider in turn the two possible cases where either $\varphi_{\mathcal{M}}(s) \in \{0,1\}^\omega$ or $\varphi_{\mathcal{M}}(s) \in \{0,1\}^*$. First, suppose that $\varphi_{\mathcal{M}}(s) \in \{0,1\}^\omega$. This means that the sequence of partial output strings produced by $\mathcal{M}$ on input s after i time steps of computation is strictly increasing as i grows to infinity, that is, $\lim_{i\to\infty} |f(s[0{:}i])| = \infty$. Moreover, for any $i \geq 0$, the word $f(s[0{:}i])$ corresponds to the output stream produced by $\mathcal{M}$ after $i+1$ time steps of computation over the input $s[0{:}i] \cdot (s - s[0{:}i]) = s$. Yet since the output stream produced by $\mathcal{M}$ over the input s is by definition $\varphi_{\mathcal{M}}(s)$, it follows that $f(s[0{:}i])$ is a prefix of $\varphi_{\mathcal{M}}(s)$ for all $i \geq 0$. Hence, the two properties $\lim_{i\to\infty} |f(s[0{:}i])| = \infty$ and $f(s[0{:}i]) \subseteq \varphi_{\mathcal{M}}(s) \in \{0,1\}^{\omega}$ for all $i \geq 0$ ensure that $\varphi_{\mathcal{M}}(s)$ is the unique infinite word that contains each word of $\{f(s[0{:}i]) : i \geq 0\}$ as a finite prefix, which is to say, by definition, that $\varphi_{\mathcal{M}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_{\omega}(s)$. Second, suppose that $\varphi_{\mathcal{M}}(s) \in \{0,1\}^*$. This means that the sequence of partial output strings produced by $\mathcal{M}$ on input s after i time steps of computation becomes stationary from time step j onward, that is, $\lim_{i\to\infty} |f(s[0{:}i])| < \infty$. Hence, the entire finite output stream $\varphi_{\mathcal{M}}(s)$ must necessarily have been produced after a finite amount of time, and thus $\varphi_{\mathcal{M}}(s) \in \{f(s[0{:}i]) : i \geq 0\}$. Moreover, as argued in the previous case, $f(s[0{:}i])$ is a prefix of $\varphi_{\mathcal{M}}(s)$ for all $i \geq 0$. Hence, the three properties $\lim_{i\to\infty} |f(s[0{:}i])| < \infty$, $\varphi_{\mathcal{M}}(s) \in \{f(s[0{:}i]) : i \geq 0\}$, and $f(s[0{:}i]) \subseteq \varphi_{\mathcal{M}}(s) \in \{0,1\}^*$ for all $i \geq 0$ ensure that $\varphi_{\mathcal{M}}(s)$ is the smallest finite word that contains each word of $\{f(s[0{:}i]) : i \geq 0\}$ as a finite prefix, which is to say by definition that $\varphi_{\mathcal{M}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_{\omega}(s)$. Therefore, $\varphi_{\mathcal{M}}(s) = f_{\omega}(s)$ for any $s \in \{0,1\}^{\omega}$, that is, $\varphi_{\mathcal{M}} = f_{\omega}$, which means that $\varphi_{\mathcal{M}}$ is recursive continuous.

Conversely, let ψ be a recursive continuous ω-translation. We show that ψ is realizable by some ITM $\mathcal{M}$. Since ψ is recursive continuous, there exists a monotone recursive function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ such that $f_{\omega} = \psi$. Now, consider the following procedure 1:

```
Procedure 1
  Input $s = s(0)s(1)s(2)\cdots \in \{0,1\}^{\omega}$ provided bit by bit
  $i \leftarrow 0$, $u \leftarrow \lambda$, $v \leftarrow \lambda$
  loop
    compute $f(s[0{:}i])$  // rec. step since $f$ is rec. by def.
    $v \leftarrow f(s[0{:}i])$
    if $u \subsetneq v$ then
      output $v - u$ bit by bit
    else
      output $\lambda$
    end if
    $i \leftarrow i + 1$
    $u \leftarrow v$
  end loop
```

Since f is recursive, procedure 1 consists of a never-ending succession of only recursive steps. Hence, there indeed exists some ITM $\mathcal{M}$ that performs procedure 1 in the following way. The machine $\mathcal{M}$ keeps outputting λ symbols while simulating any internal nonoutputting instructions of procedure 1 and then outputs the current word $v-u$ bit by bit every time it reaches the instruction "output $v-u$ bit by bit." Therefore, on any infinite input string $s \in \{0,1\}^{\omega}$, procedure 1 and the machine $\mathcal{M}$ will produce the very same sequences of nonsilent output bits $o_s \in \{0,1\}^{\leq\omega}$ after infinitely many time steps.

We now prove that $\varphi_{\mathcal{M}} = \psi$. Note that for any input stream $s \in \{0,1\}^{\omega}$, the finite word that has been output by $\mathcal{M}$ at the end of each instruction output $v-u$ bit by bit corresponds precisely to the finite word $f(s[0{:}i])$ currently stored in the variable v. Hence, after infinitely many time steps, the finite or infinite word $\varphi_{\mathcal{M}}(s)$ output by $\mathcal{M}$ contains all words of $\{f(s[0{:}i]) : i \geq 0\}$ as a finite prefix. Moreover, if $\varphi_{\mathcal{M}}(s)$ is finite, its value necessarily corresponds to some current content of the variable v, that is, to some finite word $f(s[0{:}j])$ for some $j \geq 0$. Hence, irrespective of whether $\varphi_{\mathcal{M}}(s)$ is finite or infinite, one always has $\varphi_{\mathcal{M}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_{\omega}(s)$, for any $s \in \{0,1\}^{\omega}$. Therefore, $\varphi_{\mathcal{M}} = f_{\omega} = \psi$, meaning that ψ is realized by $\mathcal{M}$.

The following result establishes the equivalence between conditions A and C of theorem 3:

Proposition 2. *Let ψ be some ω-translation. Then ψ is realizable by some IRNN[$\mathbb{Q}$] iff ψ is recursive continuous.*

Proof. Let $\varphi_{\mathcal{N}}$ be an ω-translation realized by some IRNN[$\mathbb{Q}$] $\mathcal{N}$. We show that $\varphi_{\mathcal{N}}$ is recursive continuous. For this purpose, consider the function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$, which maps every finite word u to the unique corresponding finite word output by $\mathcal{N}$ after $|u|$ steps of computation when $u \cdot x$ is provided as input bit by bit, for any $x \in \{0,1\}^{\omega}$. First, since $\mathcal{N}$ is an IRNN[$\mathbb{Q}$], the function f can be computed by some RNN[$\mathbb{Q}$] $\mathcal{N}'$, which, on every input u, would behave exactly like $\mathcal{N}$ during the $|u|$ steps of computation and then stops. Hence, the equivalence between RNN[$\mathbb{Q}$]s and TMs ensures that f is recursive (Siegelmann & Sontag, 1995). Moreover, by similar arguments as in the proof of proposition 1, the interactive deterministic behavior of $\mathcal{N}$ ensures that f is monotone and that $\varphi_{\mathcal{N}} = f_{\omega}$. Therefore, $\varphi_{\mathcal{N}}$ is recursive continuous.

Conversely, let $\psi : \{0,1\}^{\omega} \longrightarrow \{0,1\}^{\leq\omega}$ be recursive continuous. We show that ψ is realizable by some IRNN[$\mathbb{Q}$] $\mathcal{N}$. Since ψ is recursive continuous, there exists a monotone recursive function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ such that $f_{\omega} = \psi$. Now, we describe an infinite procedure that, for any infinite word $s = s(0)s(1)s(2)\cdots$ provided bit by bit, eventually produces a corresponding pair of infinite words (p_s, q_s). The procedure uses the successive values of $f(s[0{:}i])$ in order to build the corresponding sequences p_s and q_s block by block. More precisely, at stage $i+1$, the procedure computes $f(s[0{:}i+1])$. By monotonicity of f, the word $f(s[0{:}i+1])$ extends $f(s[0{:}i])$. If this extension is strict, the procedure concatenates this extension to the current value of p_s and concatenates a block of 1's of the same length to the current value of q_s. Otherwise the procedure simply concatenates a 0 to the current values of p_s and q_s. Here we give an illustration and pseudo-code of this procedure:

s	0	1	1	0	1	1	0	$\cdots$
$f(s[0{:}i])$	λ	λ	10	10	10	101	101100	$\cdots$
p_s	0	0	10	0	0	1	100	$\cdots$
q_s	0	0	11	0	0	1	111	$\cdots$

Since f is recursive, procedure 2 consists of a succession of recursive computational steps:

```
Procedure 2
  Input $s = s(0)s(1)s(2)\cdots \in \{0,1\}^\omega$ provided bit by bit
  $i \leftarrow 0, u \leftarrow \lambda, v \leftarrow \lambda, p_s \leftarrow \lambda, q_s \leftarrow \lambda$
  loop
    compute $f(s[0{:}i])$
    $v \leftarrow f(s[0{:}i])$
    if $u \subsetneq v$ then
      $p_s \leftarrow p_s \cdot (v - u)$
      $q_s \leftarrow q_s \cdot 1^{|v-u|}$
    else
      $p_s \leftarrow p_s \cdot 0$
      $q_s \leftarrow q_s \cdot 0$
    end if
    $i \leftarrow i + 1$
    $u \leftarrow v$
  end loop
```

Hence, according to the equivalence between RNN[$\mathbb{Q}$]s and TMs, there indeed exists some IRNN[$\mathbb{Q}$] $\mathcal{N}$ that performs procedure 2 in the following way: the network $\mathcal{N}$ keeps outputting pairs of $(0,0)$s every time it simulates some internal nonoutputting recursive computational instruction of procedure 2, and then outputs the current pair $(v-u, 1^{|v-u|})$ bit by bit every time it reaches up the instructions $p_s \leftarrow p_s \cdot (v-u)$ and $q_s \leftarrow q_s \cdot 1^{|v-u|}$.

We finally prove that $\varphi_{\mathcal{N}} = \psi$. A similar argument as in the proof of proposition 1 shows that $\varphi_{\mathcal{N}}(s) = \lim_{i \geq 0} f(s[0{:}i]) = f_\omega(s)$, for any $s \in \{0, 1\}^\omega$. Therefore, $\varphi_{\mathcal{N}} = f_\omega = \psi$, meaning that ψ is realized by $\mathcal{N}$.

7 IRNN[$\mathbb{R}$]s and ITM/As

This section is devoted to the proof of theorem 4. The following proposition establishes the equivalence between conditions B and C of theorem 4:

Proposition 3. *Let ψ be some ω-translation. Then ψ is realizable by some ITM/A iff ψ is continuous.*

Proof. The proof resembles that of proposition 1. First, let $\varphi_{\mathcal{M}}$ be an ω-translation realized by some TM/A $\mathcal{M}$. We show that $\varphi_{\mathcal{M}}$ is continuous. For this purpose, consider the function $f : \{0, 1\}^* \longrightarrow \{0, 1\}^*$, which maps every finite word u to the unique corresponding finite word output by $\mathcal{M}$ after $|u|$ steps of computation when $u \cdot x$ is provided as input bit by bit, for any $x \in \{0, 1\}^\omega$. By similar arguments as in the proof of proposition 1, the interactive deterministic behavior of $\mathcal{N}$ ensures that f is monotone and that $\varphi_{\mathcal{M}} = f_\omega$. Therefore, $\varphi_{\mathcal{M}}$ is continuous.

Conversely, let ψ be a continuous ω-translation. We show that ψ is realizable by some ITM/A $\mathcal{M}$. The key idea is the following. Since ψ is continuous, there exists a monotone function $f : \{0, 1\}^* \longrightarrow \{0, 1\}^*$ such that $f_\omega = \psi$. Hence, we consider the ITM/A $\mathcal{M}$, which contains a precise description of f in its advice and simulates the behavior of f step by step. The ω-translation $\varphi_{\mathcal{M}}$ eventually induced by $\mathcal{M}$ will then satisfy $\varphi_{\mathcal{M}} = f_\omega = \psi$, showing that ψ is indeed realized by $\mathcal{M}$.

More precisely, for each $i \geq 0$, let $(z_{i,j})_{j=1}^{2^i}$ be the lexicographic enumeration of the words of $\{0, 1\}^i$, and let $\alpha' : \mathbb{N} \longrightarrow \{0, 1, \sharp\}^*$ be the function that maps every integer i to the concatenation of all successive values $f(z_{i,j})$ separated by $\sharp$'s. For instance, $\alpha'(2) = \sharp f(00) \sharp f(01) \sharp f(10) \sharp f(11) \sharp$. Furthermore, let $\alpha : \mathbb{N} \longrightarrow \{0, 1\}^*$ be the advice function that maps every integer i to some suitable recursive binary encoding of $\alpha'(i)$, and consider the following procedure 3, which precisely uses the advice function α:

```
Procedure 3
  Input $s = s(0)s(1)s(2) \cdots \in \{0,1\}^\omega$ provided bit by bit
  $i \leftarrow 0$, $u \leftarrow \lambda$, $v \leftarrow \lambda$
  loop
    query $\alpha(i+1)$ and decode $f(s[0{:}i])$ from it
    $v \leftarrow f(s[0{:}i])$
    if $u \subsetneq v$ then
      output $v - u$ bit by bit
```

```
        else
            output $\lambda$
        end if
        $i \leftarrow i + 1$
        $u \leftarrow v$
end loop
```

Note that procedure 3 consists of a never-ending succession of recursive steps and extrarecursive advice calls. Hence, there indeed exists some ITM/A $\mathcal{M}$ that performs procedure 3 in the following way: the machine $\mathcal{M}$ keeps outputting λ symbols while simulating any internal nonoutputting computational instructions of procedure 3 and then outputs the current word $v-u$ bit by bit every time it reaches up the instruction "output $v-u$ bit by bit."

We now prove that $\varphi_{\mathcal{M}} = \psi$. A similar argument as in the proof of proposition 1 shows that $\varphi_{\mathcal{M}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_{\omega}(s)$, for any $s \in \{0,1\}^{\omega}$. Therefore, $\varphi_{\mathcal{M}} = f_{\omega} = \psi$, meaning that ψ is realized by $\mathcal{M}$.

We now proceed to the equivalence between conditions A and C of theorem 4. The proof is conceptually similar to that of proposition 3 but requires more work to be achieved. More precisely, in order to prove that any continuous ω-translation ψ can be realized by some IRNN[$\mathbb{R}$], we first consider a monotone function f that precisely implies ψ in the limit, such that $f_{\omega} = \psi$; then recursively encode f into some real number $r(f)$; and finally prove the existence of an IRNN[$\mathbb{R}$] $\mathcal{N}$, which, thanks to the synaptic weight $r(f)$, is able to simulate the behavior of f step by step. The ω-translation $\varphi_{\mathcal{N}}$ eventually induced by $\mathcal{N}$ will then satisfy $\varphi_{\mathcal{N}} = f_{\omega} = \psi$, showing that ψ is indeed realized by $\mathcal{N}$. The encoding and decoding approach is inspired by the method that Siegelmann and Sontag (1994) described.

First, we need to show that any function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ can be suitably encoded by some real number $r(f)$. For this purpose, for any finite word $z \in \{0,1\}^*$, let $\ulcorner z \urcorner \in \{1,3,5\}^+$ be the word obtained by doubling and adding 1 to each successive bit of z if $z \neq \lambda$, and being equal to 5 if $z = \lambda$—for instance, $\ulcorner 0100 \urcorner = 1311$. Accordingly, each value $f(z) \in \{0,1\}^*$ of f can be associated with the finite word $\ulcorner f(z) \urcorner \in \{1,3,5\}^+$. Each finite word $\ulcorner f(z) \urcorner$ can then be encoded by the rational number $r(f(z)) \in [0,1]$ given by the interpretation of $\ulcorner f(z) \urcorner$ in base 8, namely,

$$r(f(z)) = \sum_{i=0}^{|f(z)|-1} \frac{\ulcorner f(z) \urcorner(i)}{8^{i+1}}.$$

Similarly, the whole function f can be associated with the infinite word $\ulcorner f \urcorner \in \{1, 3, 5, 7\}^{\omega}$ defined by

$$\ulcorner f \urcorner = 7 \ulcorner f(0) \urcorner 7 \ulcorner f(1) \urcorner 7 \ulcorner f(00) \urcorner 7 \ulcorner f(01) \urcorner 7 \ulcorner f(10) \urcorner 7 \\ \ulcorner f(11) \urcorner 7 \ulcorner f(000) \urcorner 7 \cdots,$$

where the successive values of f are listed in lexicographic order of their arguments and separated by 7's. The infinite word $\ulcorner f \urcorner$ can then be encoded by the real number $r(f) \in [0, 1]$ given by the interpretation of $\ulcorner f \urcorner$ in base 8:

$$r(f) = \sum_{i=0}^{\infty} \frac{\ulcorner f \urcorner(i)}{8^{i+1}}.$$

The real $r(f)$ provides a nonambiguous encoding of the function f (see Siegelmann & Sontag, 1994, for more details about such encoding).

An analogous result to Siegelmann and Sontag (1994, lemma 3.2) shows that for any function $f : \{0, 1\}^* \longrightarrow \{0, 1\}^*$, there exists a corresponding (noninteractive) RNN[$\mathbb{R}$] $\mathcal{N}_f$, which, given a suitable encoding of any finite word $z \in \{0, 1\}^*$ as input, is able to retrieve the rational encoding $r(f(z))$ as output. We let $(z_i)_{i>0}$ denote the lexicographic enumeration of the words of $\{0, 1\}^+$:

Lemma 1. *Let $f : \{0, 1\}^* \longrightarrow \{0, 1\}^*$ be some function. Then there exists an RNN[$\mathbb{R}$] $\mathcal{N}_f$ containing one continuous input cell, one continuous output cell, and a synaptic real weight equal to $r(f)$, and such that, starting from the zero initial state, and given the input signal $(1 - 2^{-k})0^{\omega}$, produces an output of the form $0^* r(f(z_k))0^{\omega}$.*

Proof. We give a sketch of the proof (see Siegelmann & Sontag, 1994, lemma 3.2, for more details). The idea is that the network $\mathcal{N}_f$ first stores the integer k in memory. Then $\mathcal{N}_f$ decodes step by step the infinite sequence $\ulcorner f \urcorner$ from its synaptic weight $r(f)$ until reaching the $(k + 1)$th letter 7 of that sequence. After that, $\mathcal{N}_f$ knows that it has gone through the suitable block $\ulcorner f(z_k) \urcorner$ of the sequence $\ulcorner f \urcorner$ and proceeds to a reencoding of that last block into the rational number $r(f(z_k))$. The value $r(f(z_k))$ is finally provided as output. The technicality of the proof resides in showing that the decoding and encoding procedures are indeed performable by such an RNN[$\mathbb{R}$]. This property results from the fact that both procedures are recursive, and any recursive function can be simulated by some rational-weighted network, as Siegelmann and Sontag (1995) show. Note that $\mathcal{N}_f$ contains only $r(f)$ as nonrational weight.

The previous lemma enables us to prove the equivalence between conditions A and C of theorem 4:

Proposition 4. *Let ψ be some ω-translation. Then ψ is realizable by some IRNN[$\mathbb{R}$] iff ψ is continuous.*

Proof. The proof resembles that of proposition 2. First, let $\varphi_{\mathcal{N}}$ be an ω-translation realized by some IRNN[$\mathbb{R}$] $\mathcal{N}$. We show that $\varphi_{\mathcal{N}}$ is continuous. For this purpose, consider the function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$, which maps every finite word u to the unique corresponding finite word output by $\mathcal{N}$ after $|u|$ steps of computation when $u \cdot x$ is provided as input bit by bit, for any $x \in \{0,1\}^\omega$. By similar arguments as in the proof of proposition 1, the interactive deterministic behavior of $\mathcal{N}$ ensures that f is monotone and that $\varphi_{\mathcal{N}} = f_\omega$. Therefore, $\varphi_{\mathcal{N}}$ is continuous.

Conversely, let $\psi : \{0,1\}^\omega \longrightarrow \{0,1\}^{\leq\omega}$ be continuous. We show that ψ is realizable by some IRNN[$\mathbb{R}$] $\mathcal{N}$. For this purpose, let $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ be a monotone function such that $f_\omega = \psi$, and let $\mathcal{N}_f$ be the corresponding RNN[$\mathbb{R}$] described in lemma 1. Once again, $(z_i)_{i>0}$ denotes the lexicographic enumeration of the words of $\{0,1\}^+$, and $num : \{0,1\}^+ \longrightarrow \mathbb{N}$ is the function that maps any nonempty word x to its corresponding numbering in the the enumeration $(z_i)_{i>0}$, that is, $num(x) = i$ iff $x = z_i$.

Now we describe an infinite procedure very similar to that of the proof of Proposition 2, which, for any infinite word $s = s(0)s(1)s(2)\cdots$ provided bit by bit, eventually produces a corresponding pair of infinite words (p_s, q_s). The procedure uses the successive values of $f(s[0{:}i])$ in order to build the corresponding sequences p_s and q_s block by block. More precisely, at stage $i+1$, the procedure computes $f(s[0{:}i+1])$ by involving the capabilities of the RNN[$\mathbb{R}$] $\mathcal{N}_f$. By the monotonicity of f, the word $f(s[0{:}i+1])$ extends $f(s[0{:}i])$. If this extension is strict, the procedure concatenates this extension to the current value of p_s and concatenates a block of 1's of the same length to the current value of q_s. Otherwise the procedure simply concatenates a 0 to the current values of p_s and q_s. Here we give an illustration and pseudo-code of this procedure:

s	0	1	1	0	1	1	0	$\cdots$
$f(s[0{:}i])$	λ	λ	10	10	10	101	101100	$\cdots$
p_s	0	0	10	0	0	1	100	$\cdots$
q_s	0	0	11	0	0	1	111	$\cdots$

Note that procedure 4 consists of a succession of recursive computational steps as well as extrarecursive calls to the RNN[$\mathbb{R}$] $\mathcal{N}_f$ provided by lemma 1:

```
Procedure 4
  Input $s = s(0)s(1)s(2)\cdots \in \{0,1\}^\omega$ provided bit by bit
  $i \leftarrow 0, u \leftarrow \lambda, v \leftarrow \lambda, p_s \leftarrow \lambda, q_s \leftarrow \lambda$
  loop
    $k \leftarrow num(s[0{:}i])$                    // i.e. $s[0{:}i] = z_k$
    submit input $(1 - 2^{-k})$ to $\mathcal{N}_f$
    get output $r(f(z_k))$ from $\mathcal{N}_f$
    decode $f(z_k) = f(s[0{:}i])$ from $r(f(z_k))$
    $v \leftarrow f(z_k) = f(s[0{:}i])$
    if $u \subsetneq v$ then
      $p_s \leftarrow p_s \cdot (v - u)$
      $q_s \leftarrow q_s \cdot 1^{|v-u|}$
    else
      $p_s \leftarrow p_s \cdot 0$
      $q_s \leftarrow q_s \cdot 0$
    end if
    $i \leftarrow i + 1$
    $u \leftarrow v$
  end loop
```

Hence, there indeed exists some IRNN[$\mathbb{R}$] $\mathcal{N}$ that contains $\mathcal{N}_f$ as a subnetwork and performs procedure 4 in the following way: The network $\mathcal{N}$ keeps outputting pairs of $(0, 0)$s every time it simulates some internal nonoutputting computational instruction of procedure 4 and then outputs the current pair $(v - u, 1^{|v-u|})$ bit by bit every time it reaches the instructions "$p_s \leftarrow p_s \cdot (v - u)$" and "$q_s \leftarrow q_s \cdot 1^{|v-u|}$."

We finally prove that $\varphi_{\mathcal{N}} = \psi$. A similar argument as in the proof of proposition 1 shows that $\varphi_{\mathcal{N}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_\omega(s)$, for any $s \in \{0,1\}^\omega$. Therefore, $\varphi_{\mathcal{N}} = f_\omega = \psi$, meaning that ψ is realized by $\mathcal{N}$.

8 Conclusion

This letter provides a study of the computational powers of recurrent neural networks in a basic context of interactive and active memory computational paradigm. More precisely, we proved that rational and analog interactive neural networks have the same computational capabilities as interactive Turing machine and interactive Turing machines with advice, respectively. We also provided a precise characterization of each of these computational powers. It follows from these results that in the interactive just as in the classical framework, analog neural networks turn out to reveal super-Turing computational capabilities.

Our characterization of the computational power of interactive recurrent neural networks (see theorems 3, 4, and 5) is more than a simple interactive generalization of the previous work by Siegelmann and Sontag (1994, 1995), summarized by theorems 1 and 2 of this letter. Indeed, we believe that the consideration of an interactive computational framework represents an important step toward the modeling of a more biologically oriented paradigm of information processing in neural networks.

Also, theorems 3, 4, and 5 do not appear to us as straightforward generalizations of theorems 1 and 2, since the interactive situation contrasts with the classical one on many significant aspects. From a technical point of view, the mathematical tools involved in the modeling of the classical and interactive computational frameworks are notably different. The classical situation involves languages of finite binary strings, whereas the interactive situation involves translations of infinite binary strings. The two approaches clearly appeal to distinct kinds of reasoning. Only the encoding and decoding procedures used in the proofs are similar. In addition, the proof techniques themselves are different in spirit. In the classical situation, the equivalence between the two computational models is obtained by simulating any device of one class by a device of the other class, and conversely. In the interactive context, the equivalence is obtained by proving that both models of computation realize the same class of ω-translations. This alternative approach is used on purpose in order to obtain more complete results in the sense that an additional purely mathematical characterization of the computational powers of IRNN[$\mathbb{Q}$]s, ITMs, IRNN[$\mathbb{R}$]s, and ITM/As is also provided in this way. Furthermore, as opposed to the classical situation, a simple counting argument shows that IRNN[$\mathbb{R}$]s do not actually have unbounded computational power. Indeed, there are $2^{2^{\aleph_0}}$ possible ω-translations, whereas there are only $2^{\aleph_0}$ IRNN[$\mathbb{R}$]s, meaning that there necessarily exist uncountably many ω-translations that cannot be realized by some IRNN[$\mathbb{R}$]. This feature actually makes the interactive results more interesting than the classical ones since the model of IRNN[$\mathbb{R}$]s never becomes pathologically (unboundedly) powerful under some specific condition.

This work can be extended in several directions. First, in the perspective of evolving interactive systems presented by van Leeuwen and Wiedermann (2001a), it is envisioned to consider the concept of a interactive recurrent neural network with synaptic plasticity as a neural network whose synaptic weights would be able to evolve and change over time. It is conjectured that such networks would be equivalent to interactive analog neural networks and interactive machines with advice, thus realizing precisely the class of all continuous ω-translations. More generally, we also envision extending the possibility of evolution to several important aspects of the architecture of the networks, for example, the numbers of neurons (to capture neural birth and death), and the connectivity. Ultimately the combination of all such evolving features would provide a better understanding of the computational power of more and more biologically oriented models of interactive neural networks.

A more general interactive paradigm could also be considered, where not only the device but also the environment would be allowed to stay silent during the computation. In such a framework, any interactive device $\mathcal{D}$ would perform a no more functional yet relational ω-translation of information $R_{\mathcal{D}} \subseteq \{0,1\}^{\leq\omega} \times \{0,1\}^{\leq\omega}$ (induced by the total function $\varphi_{\mathcal{D}} : \{0,1,\lambda\}^{\omega} \longrightarrow \{0,1,\lambda\}^{\omega}$ achieved by the device $\mathcal{D}$). A precise understanding of either the function $\varphi_{\mathcal{D}}$ or the relation $R_{\mathcal{D}}$ preformed by ITMs and ITM/As would be of specific interest. We believe that the computational equivalences between ITMs and IRNN[$\mathbb{Q}$]s, as well as between ITM/As and IRNN[$\mathbb{R}$]s, still hold in this case. However, a precise mathematical characterization of that computational power remains unclear.

An even more general interactive framework could also be considered where the machines would be able to keep control of the bits that have already been output. In other words, at any time step of the computation, the machine would be allowed to erase one or several bits that have previously been output in order to come back on its decision and replace them by other bits. This approach could be justified from a machine learning perspective. Indeed, the erasing decision of the machine could be interpreted as the possibility for the machine to reconsider and correct its previous output behavior from the perspective of its current learning level. In such a machine learning interactive framework, the considered machines would certainly be able to compute ω-translations that are strictly more complicated than continuous. A better comprehension of such functions could be of interest.

Finally, we believe that the study of the computational power of more realistic neural models involved in more biologically-oriented interactive computational contexts might bring further insights to the understanding of brain functioning in general.

Acknowledgments

We gratefully acknowledge research support from the Swiss National Science Foundation under grant number PBLAP2-132975 and from the Office of Naval Research under grant number N00014-09-1-0069.

References

Goldin, D. (2000). Persistent Turing machines as a model of interactive computation. In K.-D. Schewe & B. Thalheim (Eds.), *Foundations of information and knowledge systems* (pp. 116–135). Berlin: Springer.

Goldin, D., Smolka, S. A., Attie, P. C., & Sonderegger, E. L. (2004). Turing machines, transition systems, and interaction. *Inf. Comput.*, *194*, 101–128.

Goldin, D., Smolka, S. A., & Wegner, P. (2006). *Interactive computation: The new paradigm*. New York: Springer-Verlag.

Goldin, D., & Wegner, P. (2008). The interactive nature of computing: Refuting the strong Church–Turing thesis. *Minds Mach.*, *18*, 17–38.

Kechris, A. S. (1995). *Classical descriptive set theory*. New York: Springer-Verlag.

Kilian, J., & Siegelmann, H. T. (1996). The dynamic universality of sigmoidal neural networks. *Inf. Comput.*, *128*(1), 48–56.

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In C. E. Shannon & J. McCarthy (Eds.), *Automata studies* (pp. 3–42). Princeton, NJ: Princeton University Press.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133.

Minsky, M. L. (1967). *Computation: Finite and infinite machines*. Upper Saddle River, NJ: Prentice Hall.

Siegelmann, H. T. (1995). Computation beyond the Turing limit. *Science*, *268*(5210), 545–548.

Siegelmann, H. T. (1999). *Neural networks and analog computation: Beyond the Turing limit*. Cambridge, MA: Birkhauser.

Siegelmann, H. T., & Sontag, E. D. (1994). Analog computation via neural networks. *Theor. Comput. Sci.*, *131*(2), 331–360.

Siegelmann, H. T., & Sontag, E. D. (1995). On the computational power of neural nets. *J. Comput. Syst. Sci.*, *50*(1), 132–150.

Tsuda, I. (2001). Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems. *Behav. Brain Sci.*, *24*(5), 793–847.

Tsuda, I. (2009). Hypotheses on the functional roles of chaotic transitory dynamics. *Chaos*, *19*, 015113-1–015113-10.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, *2*(42), 230–265.

van Leeuwen, J., & Wiedermann, J. (2001a). Beyond the Turing limit: Evolving interactive systems. In L. Pacholski & P. Ružicka (Eds.), *SOFSEM 2001: Theory and practice of informatics* (pp. 90–109). Berlin: Springer.

van Leeuwen, J., & Wiedermann, J. (2001b). The Turing machine paradigm in contemporary computing. In B. Engquist & W. Schmid (Eds.), *Mathematics unlimited: 2001 and beyond* (pp. 1139–1155). Berlin: Springer-Verlag.

van Leeuwen, J., & Wiedermann, J. (2006). A theory of interactive computation. In D. Goldin, S. A. Smolka, & P. Wegner (Eds.), *Interactive computation* (pp. 119–142). Berlin: Springer.

van Leeuwen, J., & Wiedermann, J. (2008). How we think of computing today. In A. Beckmann, C. Dimitracopoulos, & B. Löwe (Eds.), *Logic and theory of algorithms* (pp. 579–593). Berlin: Springer.

Wegner, P. (1997). Why interaction is more powerful than algorithms. *Commun. ACM*, *40*, 80–91.

Wegner, P. (1998). Interactive foundations of computing. *Theor. Comput. Sci.*, *192*, 315–351.

Yamaguti, Y., Kuroda, S., Fukushima, Y., Tsukada, M., & Tsuda, I. (2011). A mathematical model for Cantor coding in the hippocampus. *Neural Networks*, *24*(1), 43–53.

Received June 15, 2011; accepted October 18, 2011.