International Journal of Neural Systems, Vol. 24, No. 8 (2014) 1450029 (22 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0129065714500294

THE SUPER-TURING COMPUTATIONAL POWER OF PLASTIC RECURRENT NEURAL NETWORKS

JÉRÉMIE CABESSA*
Laboratory of Mathematical Economics (LEMMA)
University of Paris 2 – Panthéon-Assas
75006 Paris, France
jcabessa@nhrg.org

HAVA T. SIEGELMANN
Biologically Inspired Neural and Dynamical Systems Lab
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003-9264, USA
hava@cs.umas.edu

Accepted 8 September 2014
Published Online 30 October 2014

We study the computational capabilities of a biologically inspired neural model where the synaptic weights, the connectivity pattern, and the number of neurons can evolve over time rather than stay static. Our study focuses on the mere concept of plasticity of the model so that the nature of the updates is assumed to be not constrained. In this context, we show that the so-called plastic recurrent neural networks (RNNs) are capable of the precise super-Turing computational power — as the static analog neural networks — irrespective of whether their synaptic weights are modeled by rational or real numbers, and moreover, irrespective of whether their patterns of plasticity are restricted to bi-valued updates or expressed by any other more general form of updating. Consequently, the incorporation of only bi-valued plastic capabilities in a basic model of RNNs suffices to break the Turing barrier and achieve the super-Turing level of computation. The consideration of more general mechanisms of architectural plasticity or of real synaptic weights does not further increase the capabilities of the networks. These results support the claim that the general mechanism of plasticity is crucially involved in the computational and dynamical capabilities of biological neural networks. They further show that the super-Turing level of computation reflects in a suitable way the capabilities of brain-like models of computation.

Keywords: Neural networks; plastic neural networks; neural computation; Turing machines; Turing machines with advice; super-Turing; plasticity; evolvability; adaptability; learning; computational capabilities.

1. Introduction

The brain computes, but it does so differently than today's computers. Synapses update their connectivity patterns continuously, and synaptic plasticity provides the basis for most models of learning[1,2]; nervous cells can die while others regenerate[3]; acquired memory might rely on the dynamical combination of neural assemblies into higher-order constructs[4]; and neural memories themselves are updated when being retrieved in a process called reconsolidation, which causes adaptation to changing conditions.[5] Such features provide evidence of a very flexible neural architecture.

*Corresponding author.

In the context of Artificial Intelligence (AI), the consideration of evolving neural architectures in so-called Evolving Connectionist Systems (ECoS) has proven to be fruitful and significantly increased in applications.[6,7] Yet from a theoretical perspective, the general concepts of evolvability and plasticity are difficult to handle in brain modeling, and have generally been neglected in the classical literature concerning the computational capabilities of brain-like models (see e.g. Refs. 8–17 and the references there as well as more recently Refs. 18 and 19). Hence, the following questions naturally arise: Can we approach the issue of the brain's capabilities from a nonstatic perspective? Can we understand and characterize the computational capabilities of a neural model incorporating the crucial feature of plasticity?

We answer positively by providing a detailed study of the computational capabilities of a plastic neural model where the networks can update their architectures at each discrete time step. Our study focuses on the mere concept of plasticity of the model, with no assumption on the particular environment, so that the nature of the updates is assumed to be not constrained. The consideration of specific mechanisms of plasticity is envisioned for future work. In this general context, we show that plastic recurrent neural networks (RNNs) are not merely capable of breaking the Turing barrier of computation, but achieve the exact so-called "super-Turing" computational level,[20] irrespective of whether their synaptic weights are modeled by rational or real numbers, and moreover, irrespective of whether their patterns of plasticity are restricted to bi-valued updates (namely changes between two possible distinct synaptic weights, like 0 and 1) or expressed by any other more general form of updating. The super-Turing computational level has already been demonstrated to be relevant in the context of various biological and physical systems.[16,20,21]

More precisely, we consider a classical model of first-order rational-weighted RNNs, and show that by adding possibility to update their synaptic weights among only two possible values at each discrete time step, the networks drastically increase their computational capabilities from the Turing[22] to the super-Turing level,[20] thus being equivalent to static real-weighted (analog) neural networks.[13] We further show that the consideration of any other more complex pattern of plasticity would actually not further increase the capabilities of the networks. Following the common results on static neural networks,[13,14] the question then arises whether the super-Turing capabilities of the plastic neural networks would be further increased when stepping from the rational-weighted to the real-weighted context. We prove that this is actually not the case. The computational powers of static and plastic RNNs is summarized in Table 1.

Besides, central in neural computation is the issue of noise, and a natural question to be addressed concerns the robustness of the super-Turing computational power of our plastic RNNs when they are subjected to various kinds of noise. In this context, the presence of analog noise would generally strongly decrease the computational power of the underlying systems,[23–25] whereas the consideration of some discrete source of stochasticity would rather tend to increase or maintain the capabilities of the systems.[15] We show that our plastic neural model falls under the scope of these results. More precisely, on the one hand, both plastic rational and plastic real RNNs have their computational power decreased to regular or definite languages in the presence of analog noise as described in Refs. 23 and 24, respectively. On the other hand, the plastic networks have their capabilities maintained to the super-Turing level in the presence of some discrete source of stochasticity as presented in Ref. 15.

These results allow to drop any kind of analog assumption and replace it by the more biological concept of plasticity. They support the claim that the general mechanism of plasticity is crucially involved in the computational and dynamical capabilities of biological neural networks. In this sense, they provide a theoretical complement to the numerous experimental studies on the importance of the general mechanism of plasticity in the brain's information processing.[1,26,27]

Table 1. Computational power of static and plastic neural networks according to the nature of their synaptic weights and patterns of plasticity.

	Static	Plastic (bi-valued)	Plastic (general)
$\mathbb{Q}$	Turing	super-Turing	super-Turing
$\mathbb{R}$	super-Turing	super-Turing	super-Turing

These results also suggest that some intrinsic features of biological intelligence might fail to be captured by Turing-equivalent artificial models of computation. In fact, Alan Turing himself explained that his machine model is different from the human brain: "Electronic computers are intended to carry out any definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner".[28] Yet, he trusted that other models will exist that will describe intelligence better: "My contention is that machines can be constructed which will simulate the behavior of the human mind very closely".[29] In 1952, Turing suggested a particular direction, based on adaptability and learning: "If the machine is built to be treated only as a domestic pet, and is spoon-fed with particular problems, it will not be able to learn in the varying way in which human beings learn".[30] While Turing died within two years and did not manage to realize his own direction, it is possible that the results described in this paper provide a step forward in the study of more intelligent systems, following Turing 1952's call.

2. The Model

We consider a classical rate model of first-order[a] recurrent[b] neural networks where the synaptic weights can update at each discrete time step.

Following the terminology of Ref. 16, a RNN consists of a synchronous network of neurons (or processors) related together in a general architecture. The network contains a finite number of neurons $(x_i)_{i=1}^N$, M parallel input lines $(u_i)_{i=1}^M$, and P designated output neurons among the N. 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 the previous time step. Moreover, as opposed to the classical static case, the synaptic weights are now supposed to be time-dependent. Hence, given the activation values of cells $(x_j)_{j=1}^N$ and $(u_j)_{j=1}^M$ at time t, the activation value of each cell x_i at time $t+1$ is then updated by the following equation:

$$x_i(t+1) = \sigma\left(\sum_{j=1}^N a_{ij}(t)\cdot x_j(t) + \sum_{j=1}^M b_{ij}(t)\cdot u_j(t) + c_i(t)\right) \qquad (1)$$

for $i = 1,\ldots,N$, and where all $a_{ij}(t)$, $b_{ij}(t)$, and $c_i(t)$ are *bounded* and *time-dependent* synaptic weights, and σ is the classical saturated-linear activation function defined by

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

As a specific innovation of this paper, the time dependence of the synaptic weights captures the plastic capabilities of the network. The boundedness condition expresses the fact that the synaptic strengths are confined into a certain range of values imposed by the biological constitution of the neurons.[16] It formally states that there exist an upper and a lower bound s and s' such that $a_{ij}(t), b_{ij}(t), c_i(t) \in [s, s']$ for every $t \ge 0$.

Note that the present plastic neural model can describe important architectural evolving capabilities other than the sole synaptic plasticity. For instance, creation or deterioration of synapses can be modeled by switching the corresponding synaptic weights on or off, respectively, and cell birth and death are modeled by simultaneously switching on or off all the adjacent synaptic weights of a given cell, respectively.

Throughout this paper, six models of RNNs will be considered. A network will be called *rational* if all its weights are modeled by rational numbers, and *real* if all its weights are modeled by real numbers. It will also be called *static* if all its weights

[a]We recall that first-order RNNs are neural nets where the activation values of every cell are computed by means of weighted sums of other cells' activation values and inputs, as described by Eq. (1). By contrast, in higher-order RNNs, the activation values of every cell are computed by means of polynomials of other cells' activation values and inputs. The degree of the polynomial represents the order of the network.

[b]RNNs refer to neural networks whose architectures might contain loops, as opposed to feedforward neural networks. We recall that without recurrency, neural networks can approximate, adapt to input, and even evolve according to some optimization function. Yet, they cannot compute recursive (i.e. Turing computable) functions, for these latter would require a looped architecture.

remain constant over time, and *plastic* if some of its weights are time-dependent. Furthermore, it will be called *bi-valued plastic* if it is plastic and its time-dependent weights are restricted to only two possible distinct values. According to these definitions, the corresponding notions of *static rational* (St-RNN[$\mathbb{Q}$]), *static real* (St-RNN[$\mathbb{R}$]), *bi-valued plastic rational* (Pl$_2$-RNN[$\mathbb{Q}$]), *bi-valued plastic real* (Pl$_2$-RNN[$\mathbb{R}$]), *plastic rational* (Pl-RNN[$\mathbb{Q}$]), and *plastic real* (Pl-RNN[$\mathbb{R}$]) RNNs will be employed.

Observe that since rational numbers are included in real numbers, any rational network is also a real network by definition. Also, since static weights are particular cases of plastic weights where the updating patterns remain constant over time, it follows that any static network is also a plastic network. Furthermore, by definition, any bi-valued plastic network is a particular plastic network.

3. The Computational Power of Static RNNs: Previous Work

For the sake of clarity, we first recall the main results concerning the computational powers of RNNs in the case of static synaptic weights.

In 1943, McCulloch and Pitts proposed a model of the nervous system as a finite interconnection of logical devices.[8] For the first time, neural networks were considered as discrete abstract machines, and the issue of their computational capabilities investigated from the automata-theoretic perspective. In this context, Kleene and Minsky proved that finite RNNs with Boolean activation functions are computationally equivalent to classical finite state automata.[9,11] Meanwhile, in a seminal 1948 paper, Turing foresaw the possibility of surpassing the capabilities of finite state machines and reaching Turing universality via neural networks called 'B-type unorganised machines'.[31] The networks consisted of a specific interconnection of NAND neurons, and the consideration of infinitely many such cells could simulate the behavior of a Turing machine (TM). The Turing universality of neural networks involving infinitely many binary neurons has further been investigated in many directions, see for instance Refs. 32–36.

More recently, Siegelmann and Sontag proved the Turing universality of first-order rational RNNs involving only finitely many cells and simple short rational weights.[14] They showed that, on the one hand, any function determined by Eq. (1) and involving static rational weights is necessarily recursive, and thus can be computed by some TM. On the other hand, any TM can be simulated in real time by some static rational RNN.[14] The result can be expressed as follows[14,16] (for a precise definition of the notion of language decidability via RNNs, see Appendix A).

Theorem 1. *St-RNN*[$\mathbb{Q}$]*s are Turing equivalent. More precisely, a language L is decidable by some St-RNN*[$\mathbb{Q}$] *if and only if L is decidable by some TM, i.e. iff L is recursive.*

Siegelmann and Sontag made another important breakthrough in the field by showing that static real-weighted RNNs are actually strictly more powerful than their rational counterparts, and hence also than TMs.[13,16] More precisely, the analog neural networks are capable of unbounded capabilities in exponential time of computation (i.e. analog RNNs are capable of deciding all possible binary languages in exponential time), and when restricted to polynomial time of computation, the networks turn out to be computationally equivalent to Turing machines with polynomial-bounded advice[c] (TM/poly(A)), meaning that they decide the complexity class **P/poly**. This precise computational level is referred to as the *super-Turing* level. Note that since **P/poly** strictly includes the class **P** and even contains nonrecursive languages,[d] it follows that analog RNNs are capable of extra-recursive computational power from polynomial time of computation already. These results are summarized by the following theorem.[13,16]

[c]We recall that a *Turing machine with advice* (TM/A) consists of a classical TM provided with an additional advice function $\alpha : \mathbb{N} \to \{0,1\}^+$ as well as an additional advice tape, and such that, on every input u of length n, the machine first copies the advice word $\alpha(n)$ on its advice tape and then continues its computation according to its finite Turing program. A *Turing machine with polynomial-bounded advice* (TM/poly(A)) consists of a TM/A whose advice length is bounded by some polynomial.

[d]The complexity classes **P** and **P/poly** correspond to the classes of languages decided in polynomial time by TMs and TM/poly(A), respectively.

Theorem 2. *St-RNN[$\mathbb{R}$]s are super-Turing. More precisely*:

(a) *A language L is decidable in polynomial time by some St-RNN[$\mathbb{R}$] if and only if L is decidable in polynomial time by some TM/poly(A), i.e. iff $L \in$ **P/poly**.*
(b) *Any language L can be decided in exponential time by some St-RNN[$\mathbb{R}$].*

Theorems 1 and 2 show that the introduction of real synaptic weights in a standard first-order neural model drastically increases the computational power of the networks from Turing to super-Turing capabilities. Besides, Siegelmann showed another way to achieve computational capabilities beyond the Turing limits without resorting to the consideration of real synaptic weights, but via the incorporation of some *discrete source of stochasticity* in the model, as explained in details in Sec. 4.2.

In fact, the super-Turing computational class was demonstrated to be the upper bound of an infinite hierarchy of nonuniform complexity classes starting at the TM level, and made up of the increasing computational powers of analog neural networks with weights of increasing Kolmogorov complexity.[37]

From a general perspective, the relevance of the super-Turing model of computation resides in its capability to capture nonlinear dynamical properties that are most relevant to brain dynamics and that cannot be described by the classical TM model, e.g. Cantor-like encoding and rich chaotic behavior.[20,38–40] These considerations support the idea that some dynamical and computational features of neurobiological systems might lie beyond the scope of current standard artificial models of computation.

4. Results

4.1. *The computational power of plastic RNNs*

This section provides the statements of our main results. First, we show that plastic RNNs are capable of breaking the Turing barrier of computation. Next, we provide a precise characterization of their computational power.

More precisely, forthcoming key Theorem 4 shows that rational-weighted plastic RNNs provided with only bi-valued plastic capabilities do actually achieve the exact super-Turing computational power, and hence, are computationally equivalent to static analog neural networks. The following Theorems 5 and 6 show that plastic RNNs remain super-Turing equivalent irrespective of whether their synaptic weights are modeled by rational or real numbers, and moreover, irrespective of whether their patterns of plasticity are restricted to bi-valued updates or expressed by any other more general form of updating. Consequently, the four models of bi-valued plastic rational-weighted (Pl_2-RNN[$\mathbb{Q}$]s), general plastic rational-weighted (Pl-RNN[$\mathbb{Q}$]s), bi-valued plastic real-weighted (Pl_2-RNN[$\mathbb{R}$]s), and general plastic real-weighted (Pl-RNN[$\mathbb{R}$]s) RNNs are all super-Turing computationally equivalent (Corollary 7). These considerations are summarized in Table 1.

Our results show that the incorporation of bi-valued plastic capabilities into first-order rational RNNs suffices to break the Turing barrier and achieve the super-Turing level of computation. The consideration of more general mechanisms of architectural plasticity or the incorporation of real synaptic weights in the model would actually not further increase the capabilities of the neural networks. Our proofs show that the super-Turing capabilities of plastic RNNs do not come from any process of approximation of real weights by some convergent series of rational ones, but from the intrinsic power of plasticity instead. In fact, the super-Turing capabilities of plastic RNNs emerge from the potential non-recursive patterns of plasticity to which the networks might be subjected.

We now turn to the formal statements of the above mentioned results. The following lemma states that all models of plastic RNNs are capable of breaking the Turing barrier of computation.

Lemma 3. *Pl_2-RNN[$\mathbb{Q}$]s, Pl-RNN[$\mathbb{Q}$]s, Pl_2-RNN[$\mathbb{R}$]s, Pl-RNN[$\mathbb{R}$]s are all strictly more powerful than TMs.*

The proof of this lemma can be obtained as an easy consequence of previous Theorem 1 and forthcoming Theorem 6. It would however be possible to provide an independent proof of it. The argument goes as follows. By Theorem 1, St-RNN[$\mathbb{Q}$]s are Turing equivalent. Since the addition of plastic capabilities to the networks cannot reduce their computational power, it follows that Pl-RNNs (bi-valued or general, rational or real) are at least as powerful as TMs. Now, consider the unary halting problem language $UHALT = \{1^n : n$ encodes a pair $\langle M, x\rangle$ such that M is a TM that halts on input $x\}$. It can be shown that $UHALT \in \mathbf{P/poly}$ (see Ref. 41), and thus, by Theorem 6 (whose forthcoming proof is independent from the present result), $UHALT$ is decidable by some Pl-RNN (bi-valued or general, rational or real). Moreover, it can be shown that $UHALT$ is not Turing-decidable.[41] Therefore, Pl-RNNs are strictly more powerful than TMs.

The following and main result states that bi-valued plastic rational RNNs are precisely super-Turing.

Theorem 4. *Pl$_2$-RNN[$\mathbb{Q}$]s are super-Turing. More precisely*:

(a) *A language L is decidable in polynomial time by some Pl$_2$-RNN[$\mathbb{Q}$] if and only if L is decidable in polynomial time by some TM/poly(A), i.e. iff $L \in$ **P/poly**.*
(b) *Any language L can be decided in exponential time by some Pl$_2$-RNN[$\mathbb{Q}$].*

Proof. Points (b) and (a) are given by forthcoming Propositions 10(i) and 11(i) of Appendix A, respectively. □

The next result shows that the consideration of more general patterns of plasticity would actually not increase the computational capabilities of the neural networks. In other words, the translation from the bi-valued plastic to the general plastic rational context would not provide any additional power to the underlying neural networks.

Theorem 5. *Pl-RNN[$\mathbb{Q}$]s are super-Turing equivalent to Pl$_2$-RNN[$\mathbb{Q}$]s in polynomial as well as in exponential time of computation.*

Proof. The exponential time and polynomial time equivalences are given by forthcoming Propositions 10(ii) and 11(ii) of Appendix A, respectively. □

Now, the computational power of plastic real RNNs can be deduced from the previous results. The following result shows that plastic real neural networks are also computationally equivalent to bi-valued plastic rational ones, irrespective of whether their plastic synaptic weight are restricted to bi-valued patterns of plasticity or expressed by any other more general form of updating. Hence, once again, the translation from the plastic rational to the plastic real context would not provide any additional power to the underlying neural networks.

Theorem 6. *Both models of Pl$_2$-RNN[$\mathbb{R}$]s and Pl-RNN[$\mathbb{R}$]s are super-Turing equivalent to Pl$_2$-RNN[$\mathbb{Q}$]s in polynomial as well as in exponential time of computation.*

Proof. See Appendix A. □

Finally, Theorems 2 and 4–6 directly imply the super-Turing computational equivalence of the five models of Pl$_2$-RNN[$\mathbb{Q}$]s, Pl-RNN[$\mathbb{Q}$]s, Pl$_2$-RNN[$\mathbb{R}$]s, Pl-RNN[$\mathbb{R}$]s, and St-RNN[$\mathbb{R}$]s. This result shows that the super-Turing level of computation is achieved by the model of Pl$_2$-RNN[$\mathbb{Q}$]s, and that the incorporation of any more general patterns of plasticity or any possible real synaptic weights in this model does actually not further increase its computational capabilities.

Corollary 7. *Pl$_2$-RNN[$\mathbb{Q}$]s, Pl-RNN[$\mathbb{Q}$]s, Pl$_2$-RNN[$\mathbb{R}$]s, Pl-RNN[$\mathbb{R}$]s, and St-RNN[$\mathbb{R}$]s are all super-Turing equivalent in polynomial as well as in exponential time of computation.*

4.2. *The computational power of noisy plastic RNNs*

Central in neural computation is the issue of noise, and in particular, the relationship between noise and information processing in biological neural systems appears to be of specific interest.[42] Hence, a natural question to be addressed concerns the robustness of the super-Turing computational power of plastic RNNs when subjected to various kinds of noise. In this context, the presence of "analog noise" — i.e. some phenomenon that perturbs or moves the activation values of the cells according to some probability distribution — would generally strongly reduce the computational power of the neural networks.[23–25] On the other hand, the consideration of some "discrete source of stochasticity" — i.e. some additional binary input cells spiking at each time step with a certain probability — would rather tend to increase or maintain the capabilities of the neural networks.[15]

More precisely, Maass and Orponen showed that general analog computational systems subjected to arbitrarily small amount of analog noise have their computational power reduced to that of finite automata.[23] In particular, in the presence of such arbitrarily small amount of analog noise, both rational- and real-weighted RNNs have their computational capabilities seriously reduced from the Turing-equivalent and the super-Turing levels to the finite automata level, respectively, namely to the recognition of regular languages. Maass and Sontag further showed that in the presence of Gaussian or other common analog noise distribution with sufficiently large support, RNNs have their computational power reduced to even less than finite automata, since they can recognize only definite languages.[e,24] These two results were further generalized to the broader classes of quasi-compact and weakly ergodic Markov computational systems, respectively.[25]

On the other hand, the presence of discrete rather than analog types of noise tends to increase rather than decrease the computational power of the underlying information processing systems. In this context, Siegelmann proved that RNNs subjected to some discrete source of stochasticity would have their computational capabilities either improved or kept unchanged.[15] In polynomial time of computation, rational neural networks have their power enhanced from **P** to **BPP/log***,[f] and real neural nets have their computational capabilities preserved to the class **P/poly**.

All these results extend to the plastic neural networks context, and can be summarized as follows.

Theorem 8. (a) *When subjected to either arbitrarily small or larger amounts of analog noise as described in Refs.* 23 *and* 24, *all models of plastic RNNs* (*bi-valued or general*, *rational or real*) *have their computational capabilities reduced to regular and definite languages*, *respectively.*

(b) *In the presence of some discrete source of stochasticity as described in Ref.* 15, *all models of plastic RNNs* (*bi-valued or general*, *rational or real*) *have their computational capabilities maintained to the super-Turing level* ***P/poly*** *in polynomial time of computation*, *and maintained to the unbounded level in exponential time of computation.*

Proof. See Appendix A. □

Consequently, the computational power of noisy neural networks relies on the specific conception of noise that we consider. The empirical questions concerning the nature of the noise that could occur in real-world neural networks and the effects that the noise could have on the functioning of the networks remain at the moment open.[43]

5. Discussion

We proved that plastic RNNs are super-Turing — hence computationally equivalent to analog RNNs.[20] The plastic neural networks do achieve the exact same super-Turing computational power irrespective of whether their synaptic weights are modeled by rational or real numbers, and irrespective of whether their patterns of plasticity are restricted to bi-valued updates or expressed by any other more general form of updating. As a chief contribution of this paper, we introduced a minimal model (a normal form) of plastic neural networks — the so-called *bi-valued plastic neural networks* — and proved that they are super-Turing equivalent to general plastic neural networks, first studied in Ref. 6. Note that the present results are mathematically stronger and conceptually deeper than those expressed in Ref. 44, for these latter reduce to only a special case of the present ones. Furthermore, while the study in 6 is only concerned with non-noisy plastic networks, the present one also considers three stochastic models of plastic neural networks: two of them are based on analog noise and

[e]A language L is called *definite* if there exists some integer $r > 0$ such that the membership of any word w in L can be decided by just inspecting the last r symbols of w. Over finite alphabets, definite languages are regular.

[f]The complexity class **BPP** represents the class of languages recognized by some polynomial-time probabilistic TM whose error probability is bounded above by some positive constant $\varepsilon < 1/2$. The complexity class **BPP/log*** is the class of languages recognized by such kind of machines which have further access to prefix logarithmic advice functions, i.e. advices that are logarithmic is the size of the input and such that every advice string $\alpha(n)$ works for every inputs of lengths up to n. For a formal definition of this complexity class, see Ref. 15.

one of them on the stochasticity coming from a "random coin". In this context, we showed that, similarly to the case of static neural networks, plastic RNNs have their computational power decreased to regular or definite languages in the presence of analog noise,[23–25] and have their capabilities maintained to the super-Turing level in the presence of some discrete source of stochasticity.[15]

The super-Turing power of plastic neural networks — equivalent to that of static real-weighted neural networks — does actually not come from any process of approximation of real weights by some evolving convergent series of rational ones. Instead, the super-Turing capabilities of plastic neural networks emerge from intrinsic power of plasticity, or alternatively put, from the potential nonrecursive patterns of plasticity to which the networks might be subjected. In fact, Proposition 10 and Lemma 12, which show that plastic neural networks are at least super-Turing, do not involve any such process of approximation in their proofs. Proposition 13, which shows that plastic neural networks are at most super-Turing, uses the fact any plastic real neural networks can be simulated by some family of polynomially describable plastic rational neural networks, which is the purpose of technical Lemma 9.

The most interesting part of our results concerns rational-weighted neural networks. In this context, the translation from the static to the bi-valued plastic framework does bring up an additional super-Turing computational power. It is worth noting that such super-Turing capabilities can only be achieved in cases where the patterns of plasticity are themselves nonrecursive, i.e. non-Turing computable. By contrast, in the case of real-weighted neural networks, the translation from the static to the plastic framework does not bring any additional computational power to the networks. In this case, the super-Turing computational power of the networks might be related to the nonrecursivity of either the real synaptic weights that are involved, or the patterns of plasticity under consideration, or both.

To summarize, Corollary 7 shows that the consideration of either plastic capabilities or real synaptic weights does equivalently lead to the emergence of super-Turing computational capabilities for the underlying neural networks. However, even if the mechanism of *plasticity* on the one hand and the concept of the *power of the continuum* on the other hand turn out to be mathematically equivalent in this sense, they are nevertheless conceptually well distinct. While the power of the continuum is a pure conceptualization of the mind, the plastic capabilities of the networks are by contrast observable in nature.

Our results support the claim that the general mechanism of plasticity is crucially involved in the computational and dynamical capabilities of biological neural networks, and in this sense, provide a theoretical complement to the numerous experimental studies emphasizing the importance of the general mechanism of plasticity in the brain's information processing.[26–28]

From a global perspective, our results suggest that some computational characteristics of biological neural networks fail to be captured by Turing-equivalent models of computation, and that the super-Turing computational model[20] of a TM/poly(A) has a natural fit to capture the capabilities of brain-like models of computation. Indeed, while the Turing computational model requires the brain to be discrete, based on bit-calculations, and fixed in its structure, the super-Turing model, on the other hand, enables us to describe brains that consider levels of chemicals as well as evolving and adaptive architectures. These results also support the Super-Turing Thesis of Natural Computation, which states that every natural computational phenomenon can be captured within the super-Turing computational model.[13]

The achievement of super-Turing potentialities in plastic neural networks depends on the possibility for "nature" to realize nonrecursive patterns of plasticity. Such nonrecursive patterns are likely to occur in unconstrained environments, where the inputs or influences on the system are not driven by a man-made TM. In fact, Turing himself defined computable numbers as those generated by a machine that is built in advance. Besides, the consideration of purely random phenomena — intrinsic to the theory of quantum physics — also necessarily leads to the emergence of nonrecursive behaviors. Consequently, the assumption that nature does not follow only preprogrammed patterns and that biological structures can change according to random or other general kind of mechanisms suffices to acknowledge a theoretical possibility for super-Turing computational capabilities of plastic neural networks.

6. Conclusion

Nowadays, the relevance of artificial neural networks is beyond question.[46–54] For future work, we intend to pursue our study of the computational capabilities of neural models at a more biological-oriented level. For instance, the consideration of further bio-inspired weight updating paradigms along the lines of Hebb's rule, spike-timing-dependent plasticity (STDP) or input-dependent evolving frameworks would be of major interest, and bring us closer to the concept of adaptability of networks. Also, a detailed study of the computational power of spiking neural models provided with various kind of plastic capabilities seems highly promising. In general, the computational and dynamical capabilities of more and more sophisticated neural models capturing key biological phenomena and involved in more bio-inspired paradigms of computation are intended to be studied.

Finally, we believe that the present work presents some interest far beyond the question of the possible existence of some hypercomputational capabilities in nature.[44,45] In fact, comparative studies of the computational power of more and more biologically oriented neural models might ultimately bring further insight to the understanding of the intrinsic natures of biological as well as artificial intelligences. Furthermore, foundational approaches to alternative models of computation might in the long term not only lead to relevant theoretical considerations, but also to practical applications. Similarly to the theoretical work from Turing which played a seminal role in the practical realization of current digital computers, further foundational considerations of alternative models of computation might definitely contribute to the emergence of novel computational technologies and computers, and step by step, open the way to the next computational era.

Acknowledgments

We are thankful for the generous support of the Office of Naval Research (award N00014-09-1-0069) and the National Science Foundation (award ECCS-1201790).

Appendix A. Proofs of the Results

The study of the computational power of RNNs involves the consideration of a specific model of RNNs capable to perform computation and decision of formal languages, hence permitting a mathematical comparison with the languages computed by classical abstract models of computation, like TMs and TM/As in our case.

For this purpose, we consider a notion of *formal RNN* which adheres to a rigid encoding of the way binary strings are processed as input and output between the network and the environment. In our proofs, we will use the same encoding as described in Refs. 13 and 14. Yet note that the consideration of a binary string input/output framework is not restrictive, and the forthcoming results are correct for any possible finite or infinite countable input alphabet of symbols. In fact, any possible finite or infinite countable alphabet of symbols can be encoded by some set of binary strings. In particular, every integer or rational number can be encoded by some binary string. Consequently, any case of an isolated input or a finite sequence of input symbols taken from some countable alphabet can be handled by the binary string encoding context.

More precisely, we assume that formal networks are equipped with two binary input lines u_d and u_v as well as two binary output lines y_d and y_v. The data lines u_d and y_d carry some uninterrupted incoming and outgoing binary data, respectively, and the validation lines u_v and y_v take value 1 to indicate when their corresponding data line is active and take value 0 otherwise. Moreover, the networks are assumed to be designed in such a way that the two designated output lines y_d and y_v are not fed into any other cell. From this point onwards, all considered neural networks will be assumed to be of that formal form. A formal RNN is illustrated in Fig. A.1.

The formal RNNs perform computation over finite input strings of bits[g] as follows: given some formal RNN $\mathcal{N}$ and some input string $u = u_0 \cdots u_k \in \{0,1\}^+$, we say that u is *classified* in time τ by $\mathcal{N}$ if

[g]We recall that the space of all nonempty finite strings of bits is denoted by $\{0,1\}^+$, and for any $n > 0$, the set of all binary strings of length n is denoted by $\{0,1\}^n$. Moreover, any subset $L \subseteq \{0,1\}^+$ is called a *language*.

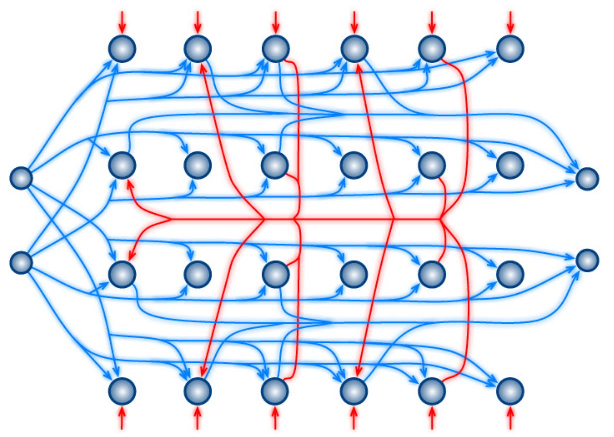


Fig. A.1. (Color online) A formal recurrent neural networks. The two little cells at the very left side represent the input data and validation lines. The two little cells at the very right side represent the output data and validation lines. The forward and recurrent synaptic connections are represented in blue and red, respectively. The background activity connections are represented in red also. The shaded style of the synaptic connections illustrates the fact that the synaptic weights might change over time.

given the input streams

$$u_d(0)u_d(1)u_d(2)\cdots = u_0\cdots u_k000\cdots,$$

$$u_v(0)u_v(1)u_v(2)\cdots = \underbrace{1\cdots1}_{k+1}000\cdots,$$

the network $\mathcal{N}$ produces the corresponding output streams

$$y_d(0)y_d(1)y_d(2)\cdots = \underbrace{0\cdots0}_{\tau-1}\eta_u000\cdots,$$

$$y_v(0)y_v(1)y_v(2)\cdots = \underbrace{0\cdots0}_{\tau-1}1000\cdots,$$

where $\eta_u \in \{0,1\}$. The input string u is said to be *accepted* or *rejected* by $\mathcal{N}$ if $\eta_u = 1$ or $\eta_u = 0$, respectively. Moreover, for any nondecreasing function $f : \mathbb{N}^* \to \mathbb{N}^*$ and any language $L \subseteq \{0,1\}^+$, we say that L is *decided* by $\mathcal{N}$ in time f if and only if every string $u \in \{0,1\}^+$ is classified by $\mathcal{N}$ in time $\tau \le f(|u|)$, and $u \in L \Leftrightarrow \eta_u = 1$. Finally, a given language L is then said to be *decidable in time f* by some network if and only if there exists a RNN $\mathcal{N}$ that decides L in time f. A language L is simply said to be *decidable* by some network if and only if there exist a RNN $\mathcal{N}$ and a nondecreasing function $f : \mathbb{N}^* \to \mathbb{N}^*$ such that $\mathcal{N}$ decides L in time f.

Now, in order to show that the computational power of plastic RNNs does not exceed the super-Turing level, we will need a lemma which we chose to present first, due to its technical nature. It is a generalization of the so-called "linear-precision suffices lemma".[13] Intuitively, the lemma states that for every Pl-RNN[$\mathbb{R}$] $\mathcal{N}$ deciding some language L in time f, there exists a family of Pl-RNN[$\mathbb{Q}$]s $\{\mathcal{N}_{f(n)} : n > 0\}$ such that each network $\mathcal{N}_{f(n)}$ can compute precisely like $\mathcal{N}$ up to time step $f(n)$ by using only about $f(n)$ precision bits to describe its weights and activation values at every time steps. In other words, every Pl-RNN[$\mathbb{R}$] $\mathcal{N}$ can be approximated by some Pl-RNN[$\mathbb{Q}$] $\mathcal{N}_{f(n)}$ which only requires about $f(n)$ precision bits for each instantaneous description but still computes precisely like $\mathcal{N}$ up to time step $f(n)$.

Formally, consider some Pl-RNN[$\mathbb{R}$] $\mathcal{N}$ given by its input neurons u_d and u_v, its output neurons y_d and y_v, the sequence of its internal neurons $(x_i)_{i=1}^{N-2}$, and the sequence of its evolving weights $(a_{ij}(t))_{t\ge0}$, $(b_{ij}(t))_{t\ge0}$, $(c_i(t))_{t\ge0}$. Consider also some nondecreasing function $f : \mathbb{N}^* \to \mathbb{N}^*$. An *f-truncated family over* $\mathcal{N}$ consists of a family of Pl-RNN[$\mathbb{Q}$]s $\{\mathcal{N}_{f(n)} : n > 0\}$ where each network $\mathcal{N}_{f(n)}$ is described as follows (see Fig. A.2):

- the nonoutputting part of network $\mathcal{N}_{f(n)}$ contains the same number of cells as $\mathcal{N}$, denoted by $\tilde{u}_d$, $\tilde{u}_v$, $\tilde{y}_d$, $\tilde{y}_v$, and $(\tilde{x}_i)_{i=1}^{N-2}$, and the same connectivity patterns between those cells as $\mathcal{N}$;
- the outputting part of $\mathcal{N}_{f(n)}$ consists of two additional cells $\tilde{y}'_d$ and $\tilde{y}'_v$ playing the role of output processors and related to $\tilde{y}_d$ and $\tilde{y}_v$ by the relation $\tilde{y}'_d(t+1) = \sigma(2\cdot\tilde{y}_d(t) - \frac{1}{2})$ and $\tilde{y}'_v(t+1) = \sigma(2\cdot\tilde{y}_v(t) - \frac{1}{2})$;
- the dynamics of $\mathcal{N}_{f(n)}$ is given as follows: at each time step, the weights of $\mathcal{N}_{f(n)}$, denoted by $\tilde{a}_{ij}(t)$, $\tilde{b}_{ij}(t)$, $\tilde{c}_i(t)$, correspond to the weights $a_{ij}(t)$, $b_{ij}(t)$, $c_i(t)$ of $\mathcal{N}$ truncated after $K\cdot f(n)$ bits, for some constant K independent of n, and the activation values of all nonoutputting processors of $\mathcal{N}_{f(n)}$ at time t, denoted by $\tilde{x}_i(t)$, $\tilde{y}_d(t)$, $\tilde{y}_v(t)$, are computed only up to $K\cdot f(n)$ precision bits, for the same some constant K independent of n.

The relationship between a Pl-RNN[$\mathbb{R}$] $\mathcal{N}$ and the Pl-RNN[$\mathbb{Q}$] $\mathcal{N}_{f(n)}$ of an f-truncated family over $\mathcal{N}$ is illustrated in Fig. A.2. As explained in more detail in the forthcoming proof of Lemma 9, the networks

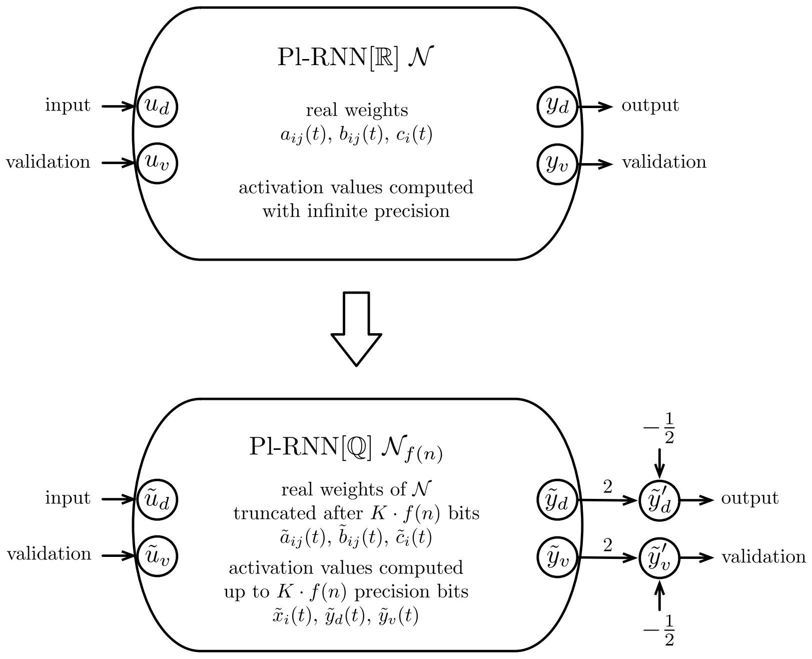

Pl-RNN[ℝ] $\mathcal{N}$
input
validation
u_d
u_v
real weights
$a_{ij}(t)$, $b_{ij}(t)$, $c_i(t)$
activation values computed with infinite precision
y_d
y_v
output
validation
Pl-RNN[ℚ] $\mathcal{N}_{f(n)}$
input
validation
$\tilde{u}_d$
$\tilde{u}_v$
real weights of $\mathcal{N}$ truncated after $K \cdot f(n)$ bits
$\tilde{a}_{ij}(t)$, $\tilde{b}_{ij}(t)$, $\tilde{c}_i(t)$
activation values computed up to $K \cdot f(n)$ precision bits
$\tilde{x}_i(t)$, $\tilde{y}_d(t)$, $\tilde{y}_v(t)$
$\tilde{y}_d$
$\tilde{y}_v$
2
2
$-\frac{1}{2}$
$-\frac{1}{2}$
$\tilde{y}'_d$
$\tilde{y}'_v$
output
validation

Fig. A.2. Relationship between a Pl-RNN[ℝ] $\mathcal{N}$ and the Pl-RNN[ℚ] $\mathcal{N}_{f(n)}$ of an f-truncated family over $\mathcal{N}$.

$\mathcal{N}_{f(n)}$ are designed in such a way that the output processors y_d, y_v and $\tilde{y}'_d, \tilde{y}'_v$ of the respective networks $\mathcal{N}$ and $\mathcal{N}_{f(n)}$ would generate the very same binary output values as long as the activation values between y_d, y_v and $\tilde{y}_d, \tilde{y}_v$ are not too distant (this distance being arbitrarily chosen as $\frac{1}{4}$ in our case).

The following result shows that any Pl-RNN[ℝ] can be perfectly simulated by a family of Pl-RNN[ℚ]s in the following sense.

Lemma 9. *Let $\mathcal{N}$ be some Pl-RNN[ℝ] and $f : \mathbb{N}^* \to \mathbb{N}^*$ be some nondecreasing function. Then there exists an f-truncated family $\{\mathcal{N}_{f(n)} : n > 0\}$ of Pl-RNN[ℚ]s over $\mathcal{N}$ such that, for every input u and every $n > 0$, the output processors of $\mathcal{N}$ and $\mathcal{N}_{f(n)}$ satisfy $y_d(t) = \tilde{y}'_d(t+1)$ and $y_v(t) = \tilde{y}'_v(t+1)$ for all time steps $t \leq f(n)$.*

Proof. By definition of an f-truncated family $\{\mathcal{N}_{f(n)} : n > 0\}$ over $\mathcal{N}$, for each $n > 0$, the dynamics of the nonoutput processors of $\mathcal{N}_{f(n)}$ is defined by $\tilde{x}_i(0) = 0$ and

$$\tilde{x}_i(t+1) = \left[\sigma\left(\sum_{j=1}^{N}[a_{ij}(t)]_{K\cdot f(n)} \cdot \tilde{x}_j(t) + \sum_{j=1}^{M}[b_{ij}(t)]_{K\cdot f(n)} \cdot \tilde{u}_j(t) + [c_i(t)]_{K\cdot f(n)}\right)\right]_{K\cdot f(n)} \quad \text{(A.1)}$$

for $i = 1, \ldots, N$, where $[\alpha]_{K\cdot f(n)}$ denotes the value of α truncated after $K \cdot f(n)$ bits for some constant K (independent of n), and the dynamics of the two output processors $\tilde{y}'_d$ and $\tilde{y}'_d$ is given by $\tilde{y}'_d(t+1) = \sigma(2 \cdot \tilde{y}_d(t) - \frac{1}{2})$ and $\tilde{y}'_v(t+1) = \sigma(2 \cdot \tilde{y}_v(t) - \frac{1}{2})$.

In order to prove the existence of an f-truncated family $\{\mathcal{N}_{f(n)} : n > 0\}$ over $\mathcal{N}$ with the required properties, we need to prove the existence of a constant K such that, for every $n > 0$ and on every input $u \in \{0,1\}^+$, the Pl-RNN[ℚ] $\mathcal{N}_{f(n)}$ and the Pl-RNN[ℝ] $\mathcal{N}$ whose dynamics are respectively governed by Eqs. (A.1) and (1) actually satisfy $y_d(t) = \tilde{y}'_d(t+1)$ and $y_v(t) = \tilde{y}'_v(t+1)$ for all time steps $t \leq f(n)$. Given some $n > 0$, some input $u \in \{0,1\}^+$, and some time step $t \geq 0$, let $u_d(t)$, $u_v(t)$, $(x_i(t))_{i=1}^{N-2}$, $x_{N-1}(t) = y_d(t)$, $x_N(t) = y_v(t)$ be the activation values of $\mathcal{N}$ at time t, and let $a_{ij}(t)$, $b_{ij}(t)$, $c_i(t)$ be the weights of $\mathcal{N}$ at time t, when working on input u; similarly, let $\tilde{u}_d(t)$, $\tilde{u}_v(t)$, $(\tilde{x}_i(t))_{i=1}^{N-2}$, $\tilde{x}_{N-1}(t) = \tilde{y}_d(t)$, $\tilde{x}_N(t) = \tilde{y}_v(t)$, $\tilde{y}'_d(t)$, $\tilde{y}'_v(t)$ denote the activation values of network $\mathcal{N}_{f(n)}$ at time t, and let $\tilde{a}_{ij}(t) = [a_{ij}(t)]_{K\cdot f(n)}$, $\tilde{b}_{ij}(t) = [b_{ij}(t)]_{K\cdot f(n)}$, $\tilde{c}_i(t) = [c_i(t)]_{K\cdot f(n)}$ denote the weights of network $\mathcal{N}_{f(n)}$ at time t, when working on u. Note that since we consider the same input u, one has $u_d(t) = \tilde{u}_d(t)$ and $u_v(t) = \tilde{u}_v(t)$. Let also $W = \max\{|s|, |s'|\}$, where s and s' are the bounds on the weights of $\mathcal{N}$. Furthermore, let the largest truncation errors of the processors and weight at time t as well as the largest accumulated error at time t be given by

$$\begin{aligned}
\delta_p(u,t) &= \max_i \Bigg| \Bigg[\sigma\left(\sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot\tilde{x}_j(t) + \sum_{j=1}^{M} \tilde{b}_{ij}(t)\cdot u_j(t) + \tilde{c}_i(t) \right) \Bigg]_{K\cdot f(n)} \\
&\qquad - \sigma\left(\sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot\tilde{x}_j(t) + \sum_{j=1}^{M} \tilde{b}_{ij}(t)\cdot u_j(t) + \tilde{c}_i(t) \right) \Bigg|, \\
\delta_w(t) &= \max\left\{ \max_{i,j} |\tilde{a}_{ij}(t) - a_{ij}(t)|, \max_{i.j} |\tilde{b}_{ij}(t) - b_{ij}(t)|, \max_i |\tilde{c}_i(t) - c_i(t)| \right\}, \\
\epsilon(t) &= \max_i |\tilde{x}_i(t) - x_i(t)|.
\end{aligned}$$

Now, let δ_p be the supremum of all values $\delta_p(u,t)$ over all possible inputs u and time steps t, i.e. $\delta_p = \sup_{u\in\{0,1\}^+, t\geq 0} \delta_p(u,t)$. Since any possible activation value always belongs to $[0,1]$, one has $\delta_p(u,t) \leq \delta_p \leq 1$. Let also δ_w be the supremum of all values $\delta_w(t)$ over all possible time steps (note that $\delta_w(t)$ does not depend on the input u), i.e. $\delta_w = \sup_{t\geq 0} \delta_w(t)$. Since every possible weight belongs by definition to $[s, s']$, the weight's largest truncation errors cannot exceed $W = \max\{|s|, |s'|\}$, and one thus has $\delta_w(t) \leq \delta_w \leq W$.

According to these definitions, using the global Lipschitz property $|\sigma(x) - \sigma(y)| \leq |x - y|$ and the fact that $u_j(t) = \tilde{u}_j(t)$, one has

$$\begin{aligned}
\epsilon(t+1) &= \max_i |\tilde{x}_i(t+1) - x_i(t+1)| \\
&= \max_i \Bigg| \Bigg[\sigma\left(\sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot\tilde{x}_j(t) + \sum_{j=1}^{M} \tilde{b}_{ij}(t)\cdot u_j(t) + \tilde{c}_i(t) \right) \Bigg]_{K\cdot f(n)} \\
&\qquad - \sigma\left(\sum_{j=1}^{N} a_{ij}(t)\cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t)\cdot u_j(t) + c_i(t) \right) \Bigg| \\
&\leq \max_i \Bigg| \sigma\left(\sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot\tilde{x}_j(t) + \sum_{j=1}^{M} \tilde{b}_{ij}(t)\cdot u_j(t) + \tilde{c}_i(t) \right) \\
&\qquad - \sigma\left(\sum_{j=1}^{N} a_{ij}(t)\cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t)\cdot u_j(t) + c_i(t) \right) \Bigg| + \delta_p(u,t) \\
&\leq \max_i \Bigg| \left(\sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot\tilde{x}_j(t) + \sum_{j=1}^{M} \tilde{b}_{ij}(t)\cdot u_j(t) + \tilde{c}_i(t) \right) \\
&\qquad - \left(\sum_{j=1}^{N} a_{ij}(t)\cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t)\cdot u_j(t) + c_i(t) \right) \Bigg| + \delta_p(u,t)
\end{aligned}$$

$$
\begin{aligned}
&\leq \max_i \left| \sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot \tilde{x}_j(t) - \sum_{j=1}^{N} a_{ij}(t)\cdot x_j(t) \right| + \left| \sum_{j=1}^{M} \tilde{b}_{ij}(t)\cdot u_j(t) - \sum_{j=1}^{M} b_{ij}(t)\cdot u_j(t) \right| \\
&\quad + |\tilde{c}_i(t) - c_i(t)| + \delta_p(u,t) \\
&\leq \max_i \left| \sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot \tilde{x}_j(t) - \sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot x_j(t) \right| + \left| \sum_{j=1}^{N} \tilde{a}_{ij}(t)\cdot x_j(t) - \sum_{j=1}^{N} a_{ij}(t)\cdot x_j(t) \right| \\
&\quad + \left| \sum_{j=1}^{M} \tilde{b}_{ij}(t)\cdot u_j(t) - \sum_{j=1}^{M} b_{ij}(t)\cdot u_j(t) \right| + |\tilde{c}_i(t) - c_i(t)| + \delta_p(u,t) \\
&\leq N\cdot (W + \delta_w(t))\cdot \epsilon(t) + (N+M+1)\cdot \delta_w(t) + \delta_p(u,t) \\
&\leq N\cdot 2\cdot W\cdot \epsilon(t) + (N+M+1)\cdot \delta_w(t) + \delta_p(u,t) \\
&\leq N\cdot 2\cdot W\cdot \epsilon(t) + (N+M+1)\cdot \delta_w + \delta_p \\
&= K_1\cdot \epsilon(t) + K_2\cdot \delta_w + \delta_p,
\end{aligned}
$$

where $K_1 = 2\cdot N\cdot W$ and $K_2 = N+M+1$ are constants. Using the fact that $\epsilon(0) = \max_i |\tilde{x}_i(0) - x_i(0)| = 0$ and the geometric series formula, one has

$$
\begin{aligned}
\epsilon(t+1) &\leq \sum_{i=0}^{t} K_1^i \cdot (K_2\cdot \delta_w + \delta_p) \\
&\leq K_1^{t+1}\cdot (K_2\cdot \delta_w + \delta_p). \qquad (A.2)
\end{aligned}
$$

Now, we want to show the existence of a constant K such that the binary output values $y_d(t), y_v(t)$ and $\tilde{y}'_d(t+1), \tilde{y}'_v(t+1)$ of the respective network $\mathcal{N}$ and truncated network $\mathcal{N}_{f(n)}$ satisfy the relations $y_d(t) = \tilde{y}'_d(t+1)$ and $y_v(t) = \tilde{y}'_v(t+1)$, for all $t \leq f(n)$. In other words, according to the dynamics of $\tilde{y}'_d$ and $\tilde{y}'_v$, one must have for all $t \leq f(n)$:

$$\text{if } y_d(t) = 0 \text{ then } \sigma\left(2\cdot \tilde{y}_d(t) - \frac{1}{2}\right) = 0$$

$$\text{i.e. } 2\cdot \tilde{y}_d(t) - \frac{1}{2} \leq 0,$$

$$\text{if } y_d(t) = 1 \text{ then } \sigma\left(2\cdot \tilde{y}_d(t) - \frac{1}{2}\right) = 1$$

$$\text{i.e. } 2\cdot \tilde{y}_d(t) - \frac{1}{2} \geq 1,$$

$$\text{if } y_v(t) = 0 \text{ then } \sigma\left(2\cdot \tilde{y}_v(t) - \frac{1}{2}\right) = 0$$

$$\text{i.e. } 2\cdot \tilde{y}_v(t) - \frac{1}{2} \leq 0,$$

$$\text{if } y_v(t) = 1 \text{ then } \sigma\left(2\cdot \tilde{y}_v(t) - \frac{1}{2}\right) = 1$$

$$\text{i.e. } 2\cdot \tilde{y}_v(t) - \frac{1}{2} \geq 1,$$

for all $t \leq f(n)$. Note that the above relations hold whenever $|\tilde{y}_d(t) - y_d(t)| \leq \frac{1}{4}$ and $|\tilde{y}_v(t) - y_v(t)| \leq \frac{1}{4}$ for all $0 \leq t \leq f(n)$, and hence in particular whenever $\epsilon(t) \leq \frac{1}{4}$ for all $t \leq f(n)$ (since y_d and y_v belong to the x_i's). This latter inequality is satisfied for $t = 0$ (since $\epsilon(0) = 0 \leq \frac{1}{4}$), and according to inequality (A.2), it also holds for all $0 < t \leq f(n)$ if

$$K_1^t \cdot (K_2\cdot \delta_w + \delta_p) \leq \frac{1}{4} \text{ for all } 0 < t \leq f(n).$$

Now, note that the previous relations hold if δ_w and δ_p are both bounded by all values $\frac{1}{5}(K_1\cdot K_2)^{-t}$, for all $0 < t \leq f(n)$ (for the case $t = 1$, use the fact that $K_2 \geq 5$, and for the cases $t > 1$, use the fact that $K_2 + 1 \leq K_2^t$). Since $\frac{1}{5}(K_1\cdot K_2)^{-f(n)} \leq \frac{1}{5}(K_1\cdot K_2)^{-t}$ for all $t \leq f(n)$, the above relations are also satisfied if δ_w and δ_p are both bounded by $\frac{1}{5}(K_1\cdot K_2)^{-f(n)}$. Moreover, we recall that if the truncations in δ_w and δ_p occur at $K\cdot f(n)$ bits after the decimal point, then δ_w and δ_p are bounded by $2^{-K\cdot f(n)}$. Hence, in order to have δ_w and δ_p bounded by $\frac{1}{5}(K_1\cdot K_2)^{-f(n)}$ as requested, it suffices to have $2^{-K\cdot f(n)} \leq \frac{1}{5}(K_1\cdot K_2)^{-f(n)}$, i.e. to have $K \geq \log(\frac{1}{5}(K_1\cdot K_2))$. Therefore, by taking $K = \ulcorner\log(\frac{1}{5}(K_1\cdot K_2))\urcorner$ (where $\ulcorner x\urcorner$ denotes the least integer above x), the dynamics given by Eq. (A.1) ensures that the output binary values $y_d(t), y_v(t)$ as well as $\tilde{y}'_d(t+1), \tilde{y}'_v(t+1)$ produced by the respective networks $\mathcal{N}$ and $\mathcal{N}_{f(n)}$ will be the very same for all time steps $t \leq f(n)$. This concludes the proof. □

We now show the main results of the paper.

Proposition 10. *Let $L \subseteq \{0,1\}^+$ be some language.*

(i) *There exists some Pl$_2$-RNN[$\mathbb{Q}$] that decides L in exponential time.*
(ii) *There exists some Pl-RNN[$\mathbb{Q}$] that decides L in exponential time.*

Proof. Note that Point (ii) is a direct consequence of Point (i), since any Pl$_2$-RNN[$\mathbb{Q}$] is a particular Pl-RNN[$\mathbb{Q}$]. We now prove Point (i). The main idea of the proof is illustrated in Fig. A.3.

First of all, let $w_1, w_2, w_3, \ldots$ denote the infinite lexicographical enumeration of all words of $\{0,1\}^+$ (i.e. $w_1 = 0$, $w_2 = 1$, $w_3 = 00$, $w_4 = 01$, $w_5 = 10$, $w_6 = 11$, $w_7 = 000$, etc.), and for every $i > 0$, let ε_i be the *L-characteristic bit* $\chi_L(w_i)$ of w_i, i.e. $\varepsilon_i = 1$ iff $w_i \in L$. Now, let w be the binary infinite word defined as the succession of all w_i's and ε_i's separated by 0's, i.e.

$$w = w_1 0 \varepsilon_1 0 w_2 0 \varepsilon_2 0 w_3 0 \varepsilon_3 0 w_4 0 \varepsilon_4 0 \cdots.$$

In words, w represents a description of all successive binary words followed by the information of whether each of these words belongs to L or not. Besides, consider also the binary infinite word z which has the same structure as w except that every sub-word w_i is replaced by a block of 1's of the same length and every bit ε_i is replaced by a 1, i.e.

$$z = 101010101101011010 \cdots.$$

The idea is that the infinite word z acts as a validation line for the infinite word w, i.e. the active bits of z correspond in their positions to the data bits of w. The superposition bit by bit of the words w and z is as illustrated below:

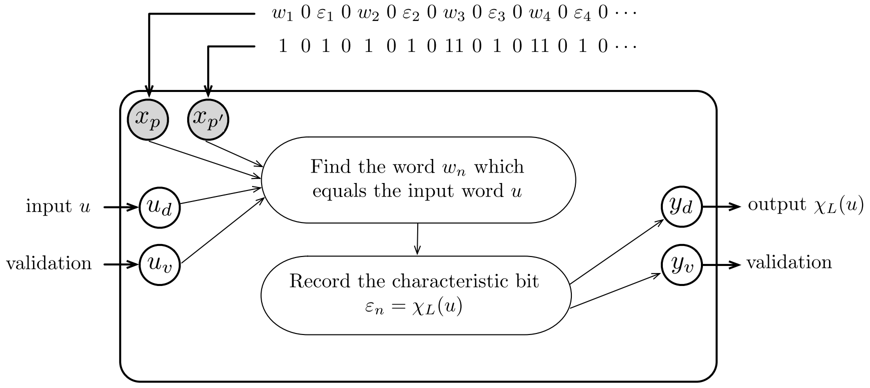
w_1	0	ε_1	0	w_2	0	ε_2	0	w_3	0	ε_3	0	w_4	0	ε_4	0	$\cdots$
1	0	1	0	1	0	1	0	11	0	1	0	11	0	1	0	$\cdots$

We now provide the description of a Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ that decides L in exponential time. The network $\mathcal{N}_L$ actually consists of one plastic and one static rational sub-network connected together.

The plastic rational-weighted part of $\mathcal{N}_L$ is made up of two designated processors x_p and $x_{p'}$. Both neurons x_p and $x_{p'}$ receive as incoming synaptic connections a background activity of changing intensity $c_p(t)$ and $c_{p'}(t)$, respectively. The synaptic weight $c_p(t)$ takes as values the successive bits composing the infinite word w. The synaptic weight $c_{p'}(t)$ takes as values the successive bits composing the infinite word z. At each time step, $c_p(t)$ and $c_{p'}(t)$ switch from one bit to the next. In this way, the synaptic weights $c_p(t)$ and $c_{p'}(t)$ evolve among only two possible values, namely 0 or 1. In words, the neurons x_p and $x_{p'}$ respectively receive as background activities the two binary infinite words w and z in parallel.

We now describe the static rational-weighted part of $\mathcal{N}_L$. This network is designed in order to perform the recursive neural procedure described by Algorithm 1 below. Algorithm 1 receives some finite binary input u bit by bit, uses the information provided by the plastic neurons x_p and $x_{p'}$, and eventually decides whether u belongs to L or not.

Algorithm 1 consists of two subroutines performed in parallel until some return instruction



$w_1\ 0\ \varepsilon_1\ 0\ w_2\ 0\ \varepsilon_2\ 0\ w_3\ 0\ \varepsilon_3\ 0\ w_4\ 0\ \varepsilon_4\ 0 \cdots$
1 0 1 0 1 0 1 0 11 0 1 0 11 0 1 0 ⋯
x_p
$x_{p'}$
input u
u_d
validation
u_v
Find the word w_n which equals the input word u
Record the characteristic bit $\varepsilon_n = \chi_L(u)$
y_d
y_v
output $\chi_L(u)$
validation

Fig. A.3. Illustration of the Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ described in the proof of Proposition 10.

Algorithm 1 Neural procedure

```
Input: finite binary word $u$ provided bit by bit
 1: SUBROUTINE 1:
 2: $c \leftarrow 0$
 3: for all time steps $t \geq 0$ do
 4:     store $x_p(t)$ into neuron $x_w$
 5:     store $x_{p'}(t)$ into neuron $x_z$
 6:     if $x_{p'}(t) = 0$ then
 7:         $c \leftarrow c + 1 \bmod 2$        // $c$ counts modulo 2 the number of 0's occurring at neuron $x_{p'}$
 8:     end if
 9: end for
10: SUBROUTINE 2:
11: for $t = 0$ to $|u|$ do
12:     store $u(t)$ into neuron $x_u$
13: end for        // the activation value of $x_u$ represents an encoding of $u$
14: wait for $c$ (of subroutine 1) to switch from 1 to 0
15: copy the current activation value of neuron $x_w$ (of subroutine 1) into neuron $x_{w'}$
16: copy the current activation value of neuron $x_z$ (of subroutine 1) into neuron $x_{z'}$
    // at that moment $t$, the values of $x_{w'}$ and $x_{z'}$ represent the encodings of two words $w'$ and $z'$ of the forms
    // $w' = w_1 0 \varepsilon_1 0 w_2 0 \varepsilon_2 0 \cdots w_{n(t)} 0 \varepsilon_{n(t)} 0$ and $z' = 1^{|w_1|} 0 \varepsilon_1 0 1^{|w_2|} 0 \varepsilon_2 0 \cdots 1^{|w_{n(t)}|} 0 \varepsilon_{n(t)} 0$, for some $n(t) > 0$
17: if $u = w_i$ for some $i = 1, \ldots, n(t)$ and $\varepsilon_i = 1$ then
18:     return ACCEPT        // in this case, $u \in L$
19: else if $u = w_i$ for some $i = 1, \ldots, n(t)$ and $\varepsilon_i = 0$ then
20:     return REJECT        // in this case, $u \notin L$
21: else
22:     goto instruction 14        // we still don't know whether $u$ belongs to $L$ or not
23: end if
```

is eventually reached. It involves seven designated neurons x_u, x_p, $x_{p'}$, x_w, $x_{w'}$, x_z, $x_{z'}$. Concerning instructions 4, 5 and 12, the way to store successive incoming bits into some designated neuron is described in detail in Ref. 14. Intuitively, the activation value of a neuron can be employed to encode the content a binary stack, and every new incoming bit can be pushed into the stack in constant time.[14] These tasks can be achieved by some simple static rational-weighted RNNs.[14] In order to be able to store binary words of any possible finite length, one needs to dispose of an unbounded memory. This is achieved via the possibility to dispose of an arbitrary precision for the rational activation values of the neurons.[14] For instructions 7 and 14, the implementation of a counter by some static rational RNN is described in detail in Ref. 14. The counter is implemented as a unary stack. Incrementing or decrementing the counter is achieved by pushing or popping an element to and from the stack. The content of the unary stack is encoded by the activation value of a neuron.[14] In instructions 15 and 16, the two copies are achieved by simply triggering two synaptic connections of intensities 1 from x_w to $x_{w'}$, and from x_z to $x_{z'}$, respectively. Instructions 17 to 23 are written in a high-level language, but it is clear that they can be performed by some three-tape TM, where the words u, w', and z' encoded in neurons x_u, $x_{w'}$, and $x_{z'}$ are written on each tape at the beginning of the computation. According to the real-time computational equivalence between TMs and static rational-weighted RNNs proven in Ref. 14, this instruction block can also be simulated by some static rational RNNs with the words u, w' and z' encoded as rational activation values of the three designated neurons x_u, $x_{w'}$ and $x_{z'}$ at the beginning of the computation. The possibility to encode and decode any finite binary word into and from the activation value of some neuron is described in detail in Ref. 14.

Besides, note that every time instructions $14, 15, 16$ are re-executed (via instruction 22), the activation values of $x_{w'}$ and $x_{z'}$ will represent the encodings of two words w' and z' which strictly extend those two involved in the previous execution of these instructions. Now, since $(w_i)_{i>0}$ is an enumeration of $\{0,1\}^+$, there exists some $k > 0$ such that $u = w_k$. By the previous argument, there will necessarily be some execution of instructions $14, 15, 16$ involving a word w' of the form $w' = w_1 0 \varepsilon_1 0 w_2 0 \varepsilon_2 0 \cdots w_n 0 \varepsilon_n 0$, where $n \geq k$. In this case, the word u matches one of the sub-words w_i's of w', meaning that Algorithm 1 will either provide an accepting or a rejecting answer, and therefore necessarily terminate.

Hence, the analysis of each instruction ensures that Algorithm 1 always terminates and can indeed be simulated by some static rational RNN. This RNN represents the static rational-weighted part of $\mathcal{N}_L$.

The Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ consists of the bi-valued plastic and the static rational sub-networks described above. According to Algorithm 1, $\mathcal{N}_L$ clearly decides the language L. Moreover, for any input u of length n, the network has to wait for $O(2^n)$ time steps before the binary word u occurs as a sub-word of w. Therefore, the network $\mathcal{N}_L$ decides the language L in exponential time. □

Proposition 11. *Let $L \subseteq \{0,1\}^+$ be some language.*

(i) *L is decidable in polynomial time by some Pl$_2$-RNN[$\mathbb{Q}$] if and only if $L \in$ **P/poly**.*
(ii) *L is decidable in polynomial time by some Pl-RNN[$\mathbb{Q}$] if and only if $L \in$ **P/poly**.*

The proof is achieved by Lemmas 12 and 13. Note that Lemma 12 concerns Pl$_2$-RNN[$\mathbb{Q}$]s whereas Lemma 13 concerns Pl-RNN[$\mathbb{Q}$]s.

Lemma 12. *Let $L \subseteq \{0,1\}^+$ be some language. If $L \in$ **P/poly**, then there exists a Pl$_2$-RNN[$\mathbb{Q}$] that decides L in polynomial time.*

Proof. The present proof resembles the proof of Proposition 10. The main idea of the proof is illustrated in Fig. A.4. We recall that the notation $|x|$ denotes the length of the word x.

First of all, since $L \in$ **P/poly**, there exists a TM/poly(A) $\mathcal{M}$ that decides L in polynomial time. Let $\alpha : \mathbb{N}^* \to \{0,1\}^+$ be the polynomial-bounded advice function of $\mathcal{M}$. Let w be the binary infinite word defined as the succession of all $\alpha(i)$'s separated by 0's, i.e.

$$w = \alpha(1)0\alpha(2)0\alpha(3)0\alpha(4)0\cdots.$$

Moreover, let z be the binary infinite word which has the same structure as w except that every sub-word $\alpha(w_i)$ is replaced by a block of 1's of the same length, i.e.

$$z = 1^{|\alpha(w_1)|}01^{|\alpha(w_2)|}01^{|\alpha(w_3)|}01^{|\alpha(w_4)|}0\cdots.$$

Once again, the idea is that the infinite word z acts as a validation line for the infinite word w, i.e. the active bits of z correspond in their positions to the data bits of w. The superposition bit by bit of the words w and z is as illustrated below:

$$\begin{array}{ccccccccc} \alpha(1) & 0 & \alpha(2) & 0 & \alpha(3) & 0 & \alpha(4) & 0 & \cdots \\ 1^{|\alpha(1)|} & 0 & 1^{|\alpha(2)|} & 0 & 1^{|\alpha(3)|} & 0 & 1^{|\alpha(4)|} & 0 & \cdots \end{array}$$

We now provide the description of a Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ that decides L in polynomial time. Once again,

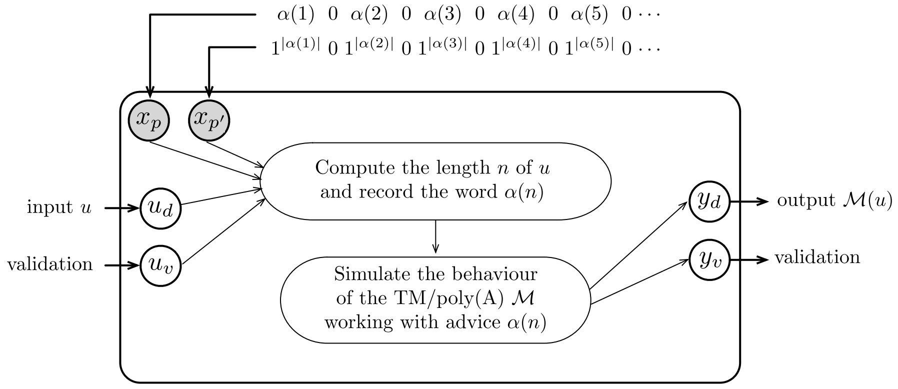

$\alpha(1)\ 0\ \alpha(2)\ 0\ \alpha(3)\ 0\ \alpha(4)\ 0\ \alpha(5)\ 0\cdots$
$1^{|\alpha(1)|}\ 0\ 1^{|\alpha(2)|}\ 0\ 1^{|\alpha(3)|}\ 0\ 1^{|\alpha(4)|}\ 0\ 1^{|\alpha(5)|}\ 0\cdots$
x_p
$x_{p'}$
input u
u_d
validation
u_v
Compute the length n of u and record the word $\alpha(n)$
Simulate the behaviour of the TM/poly(A) $\mathcal{M}$ working with advice $\alpha(n)$
y_d
output $\mathcal{M}(u)$
y_v
validation

Fig. A.4. Illustration of the Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ described in the proof of Lemma 12.

the network $\mathcal{N}_L$ consists of one plastic and one static rational sub-network connected together.

The plastic rational-weighted part of $\mathcal{N}_L$ is made up of two designated processors x_p and $x_{p'}$. The neurons x_p and $x_{p'}$ receive as incoming synaptic connections background activities of changing intensities $c_p(t)$ and $c_{p'}(t)$, each of which taking as values the successive bits of the infinite words w and z, respectively. At each time step, $c_p(t)$ and $c_{p'}(t)$ switch from one bit to the next. In this way, the synaptic weights $c_p(t)$ and $c_{p'}(t)$ evolve among only two possible values, namely 0 or 1. In words, the neurons x_p and $x_{p'}$ respectively receive as background activities the two binary infinite words w and z in parallel.

The static rational-weighted part of $\mathcal{N}_L$ is designed in order to perform the recursive neural procedure described by Algorithm 2 below. Algorithm 2 receives some finite binary input u bit by bit, uses the information provided by the plastic neurons x_p and $x_{p'}$, and eventually decides in polynomial time whether u belongs to L or not.

Algorithm 2 consists of two subroutines performed in parallel until some final answer is provided in instruction 16. It involves eight designated neurons x_u, x_p, $x_{p'}$, x_w, $x_{w'}$, x_z, $x_{z'}$, x_α. Concerning instructions 3, 4 and 8, the way to store successive incoming bits into some designated neuron is described in detail in Ref. 14. Instruction 10 uses a counter which is implemented as a unary stack. For each letter of u, a 1 is pushed into the stack. The content of the stack is encoded in the activation value of a neuron.[14] For instruction 11, the counting procedure is implemented as follows: every time some 0 occurs as a background activity of neuron $x_{p'}$, a 1 is popped from the neuron stack, until the stack becomes empty. In instructions 12 and 13, the two copies are achieved by simply triggering two synaptic connections of intensities 1 from x_w to $x_{w'}$, and from x_z to $x_{z'}$, respectively. The block of instructions 14 and 15 is written in a high-level language, but it is clear that it can be performed by some TM, where the words w' encoded in neurons $x_{w'}$ is written on the tape at the beginning of the computation. According to the real-time computational equivalence between TMs and static rational-weighted RNNs,[14] this sub-procedure can also be simulated by some static rational RNNs with the word w' encoded as the rational activation value of a designated neurons $x_{w'}$ at the

Algorithm 2 Neural procedure

```
Input: finite binary word $u$ provided bit by bit
 1: SUBROUTINE 1:
 2: for all time steps $t \geq 0$ do
 3:     store $x_p(t)$ into neuron $x_w$
 4:     store $x_{p'}(t)$ into neuron $x_z$
 5: end for
 6: SUBROUTINE 2:
 7: for $t = 0$ to $|u|$ do
 8:     store $u(t)$ into neuron $x_u$
 9: end for                // the activation value of $x_u$ represents an encoding of $u$
10: compute and store the value $|u|$ in a neuron
11: from the current time step, wait that $|u|$ occurrences of 0 have appeared at neuron $x_{p'}$
12: copy the current activation value of neuron $x_w$ (of subroutine 1) into neuron $x_{w'}$
13: copy the current activation value of neuron $x_z$ (of subroutine 1) into neuron $x_{z'}$
    // at that point, the activation values of $x_{w'}$ and $x_{z'}$ represent the encodings of two words $w'$ and $z'$
    // that are prefixes of $w$ and $z$ respectively.
    // Since one has waited that at least $|u|$ 0's have occurred as a background activity of neuron $x_{p'}$,
    // we are sure that the finite word $w'$ contains the value $\alpha(|u|)$ as sub-word.
14: decode $\alpha(|u|)$ from the activation value of $x_{w'}$
15: store $\alpha(|u|)$ into neuron $x_\alpha$
16: simulate the behavior of the TM/poly(A) $\mathcal{M}$ working on $u$ with $\alpha(n)$ written on its advice tape
```

beginning of the computation. Concerning instruction 16, the behavior of a TM/poly(A) $\mathcal{M}$ working on u and with the advice string $\alpha(n)$ already written on its advice tape is clearly recursive (only the call to the advice function is not recursive), and therefore, can be simulated by some static rational-weighted RNN.[14]

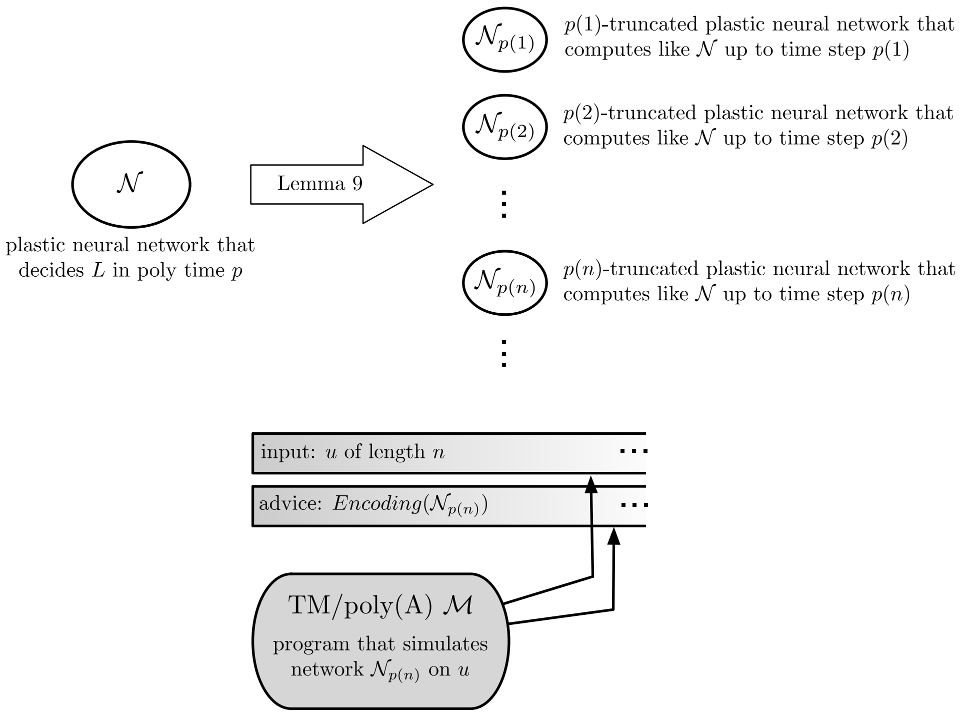
Hence, the analysis of each instruction ensures that Algorithm 2 can indeed be simulated by some static rational RNN, which represents the static rational-weighted part of $\mathcal{N}_L$.

The Pl_2-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ consists of the bi-valued plastic and the static rational sub-networks described above. According to Algorithm 2, the Pl_2-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ outputs the same answer as $\mathcal{M}$. Since $\mathcal{M}$ decides the language L, so does $\mathcal{N}_L$. Besides, since the advice is polynomial-bounded, it follows that for any input u, the network has to wait for polynomially many time steps before the binary word $\alpha(|u|)$ occurs as a sub-word of w. Moreover, since $\mathcal{M}$ decides L in polynomial time, the simulating task of $\mathcal{M}$ by $\mathcal{N}_L$ is also done in polynomial time in the input size.[14] Consequently, $\mathcal{N}_L$ decides L in polynomial time. □

Lemma 13. *Let $L \subseteq \{0,1\}^+$ be some language. If there exists a Pl-RNN[$\mathbb{Q}$] that decides L in polynomial time, then $L \in$* ***P/poly***.

Proof. The main idea of the proof is illustrated in Fig. A.5. Suppose that L is decided by some Pl-RNN[$\mathbb{Q}$] $\mathcal{N}$ in polynomial time p. Since $\mathcal{N}$ is by definition also a Pl-RNN[$\mathbb{R}$], Lemma 9 applies and shows the existence of a p-truncated family of Pl-RNN[$\mathbb{Q}$]s over $\mathcal{N}$. Hence, for every n, there exists a Pl-RNN[$\mathbb{Q}$] $\mathcal{N}_{p(n)}$ such that: first, the network $\mathcal{N}_{p(n)}$ has the same processors and connectivity pattern as $\mathcal{N}$; second, for every $t \leq p(n)$, each rational synaptic weight of $\mathcal{N}_{p(n)}(t)$ can be represented by some sequence of bits of length at most $C \cdot p(n)$, for some constant C independent of n; third, on every input of length n, if one restricts the activation values of $\mathcal{N}_{p(n)}$ to be all truncated after $C \cdot p(n)$ bits at every time step, then the output processors of $\mathcal{N}$ and $\mathcal{N}_{p(n)}$ at respective time steps t and $t+1$ have the same activation values for all time steps $t \leq p(n)$.

We now prove that L can also be decided in polynomial time by some TM/poly(A) $\mathcal{M}$. First of all,



$\mathcal{N}$
plastic neural network that decides L in poly time p
Lemma 9
$\mathcal{N}_{p(1)}$
$p(1)$-truncated plastic neural network that computes like $\mathcal{N}$ up to time step $p(1)$
$\mathcal{N}_{p(2)}$
$p(2)$-truncated plastic neural network that computes like $\mathcal{N}$ up to time step $p(2)$
$\mathcal{N}_{p(n)}$
$p(n)$-truncated plastic neural network that computes like $\mathcal{N}$ up to time step $p(n)$
input: u of length n
advice: $Encoding(\mathcal{N}_{p(n)})$
TM/poly(A) $\mathcal{M}$
program that simulates network $\mathcal{N}_{p(n)}$ on u

Fig. A.5. Illustration of the proof idea of Lemma 13.

consider the advice function $\alpha : \mathbb{N} \to \{0,1\}^+$ given by $\alpha(i) = \text{Encoding}(\langle \mathcal{N}_{p(i)}(t) : 0 \le t \le p(i) \rangle)$, where $\text{Encoding}(\langle \mathcal{N}_{p(i)}(t) : 0 \le t \le p(i) \rangle)$ denotes some suitable recursive encoding of the sequence of successive descriptions of the network $\mathcal{N}_{p(i)}$ up to time step $p(i)$. Note that $\alpha(i)$ consists of the encoding of $p(i)+1$ successive descriptions of the network $\mathcal{N}_{p(i)}$, where each of this description has synaptic weights representable by at most $C \cdot p(i)$ bits. Therefore, the length of $\alpha(i)$ belongs to $O(p(i)^2)$, and thus is still polynomial in i.

Now, consider the TM/poly(A) $\mathcal{M}$ that uses α as advice function, and which, on every input u of length n, first calls the advice word $\alpha(n)$, then decodes this sequence in order to simulate the truncated network $\mathcal{N}_{p(n)}$ on input u up to time step $p(n)$ and in such a way that all activation values of $\mathcal{N}_{p(n)}$ are only computed up to $C \cdot p(n)$ bits at every time step. Note that each simulation step of $\mathcal{N}_{p(n)}$ by $\mathcal{M}$ is performed in polynomial time in n, since the decoding of the current configuration of $\mathcal{N}_{p(n)}$ from $\alpha(n)$ is polynomial in n, and the computation and representations of the next activation values of $\mathcal{N}_{p(n)}$ from its current activation values and synaptic weights are also polynomial in n. Consequently, the $p(n)$ simulation steps of $\mathcal{N}_{p(n)}$ by $\mathcal{M}$ are performed in polynomial time in n.

Now, since any u of length n is classified by $\mathcal{N}$ in time $p(n)$, Lemma 9 ensures that u is also classified by $\mathcal{N}_{p(n)}$ in time $p(n)$, and the behavior of $\mathcal{M}$ ensures that u is also classified by $\mathcal{M}$ in $p(n)$ simulation steps of $\mathcal{N}_{p(n)}$, each of which being polynomial in n. Hence, any word u of length n is classified by the TM/poly(A) $\mathcal{M}$ in polynomial time in n, and the classification answers of $\mathcal{M}$, $\mathcal{N}_{p(n)}$, and $\mathcal{N}$ are the very same. Since $\mathcal{N}$ decides the language L, so does $\mathcal{M}$. Therefore $L \in \mathbf{P/poly}$, which concludes the proof. □

Proof of Proposition 11. Concerning Point (i), the backward implication is given by Lemma 12. For the forward implication, suppose that L is decidable by some Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}$. Then, L is also decidable by some Pl-RNN[$\mathbb{Q}$], namely $\mathcal{N}$ itself. By Lemma 13, $L \in \mathbf{P/poly}$.

Concerning Point (ii), the forward implication is given by Lemma 13. For the backward implication, suppose that $L \in \mathbf{P/poly}$. By Lemma 12, L is decidable by some Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}$. Consequently, L is also decidable by some Pl-RNN[$\mathbb{Q}$], namely $\mathcal{N}$ itself. This concludes the proof. □

Proof of Theorem 6. We first consider the case of the exponential time of computation. Let $L \subseteq \{0,1\}^*$ be some language. Then, by Proposition 10(i), L is decidable in exponential time by some Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}$. Hence, L is also decidable in exponential time by some Pl$_2$-RNN[$\mathbb{R}$] as well as by some Pl-RNN[$\mathbb{R}$], namely by $\mathcal{N}$ itself. This shows that any language L can be decided in exponential time by some Pl$_2$-RNN[$\mathbb{R}$] or some Pl-RNN[$\mathbb{R}$].

We now treat the case of the polynomial time of computation. Suppose that L is decidable in polynomial time by some Pl$_2$-RNN[$\mathbb{R}$] $\mathcal{N}$. Then, L is also decidable in polynomial time by some Pl-RNN[$\mathbb{R}$], namely by $\mathcal{N}$ itself. Furthermore, since Lemma 9 is originally stated for the case of Pl-RNN[$\mathbb{R}$]s, it follows that Lemma 13 — which appeals to Lemma 9 in its proof — can also be generalized in the context of Pl-RNN[$\mathbb{R}$]s. Consequently, $L \in \mathbf{P/poly}$. This provides the two implications from Pl$_2$-RNN[$\mathbb{R}$]s and Pl-RNN[$\mathbb{R}$]s to $\mathbf{P/poly}$.

Conversely, suppose that $L \in \mathbf{P/poly}$. By Theorem 4(a), L is decidable in polynomial time by some Pl$_2$-RNN[$\mathbb{Q}$] $\mathcal{N}$. Hence, L is also decidable in polynomial time by some Pl$_2$-RNN[$\mathbb{R}$] as well as by some Pl-RNN[$\mathbb{R}$], namely by $\mathcal{N}$. This provides the two implications from $\mathbf{P/poly}$ to Pl$_2$-RNN[$\mathbb{R}$]s and Pl-RNN[$\mathbb{R}$]s. □

Proof of Theorem 8. Point (a). Note that the dynamics of plastic RNNs is governed by an equation of the form

$$\mathbf{x(t+1)} = \sigma(\mathbf{A(t)} \cdot \mathbf{x(t)} + \mathbf{b(t)} \cdot \mathbf{u(t)} + \mathbf{c(t)}),$$

where $\mathbf{A(t)}$, $\mathbf{b(t)}$, and $\mathbf{c(t)}$ are the weight matrices and vectors, and σ is the linear-sigmoid activation function applied componentwise. Since weights are time-dependent, the computational states of plastic neural networks are given by a description of both the current activation values of the cells as well as the current synaptic weights, and hence, for a network containing $N+2$ cells (N activation cells plus 2 inputs), can be represented by a K-dimensional vector, with $K = (N+2) + (N^2 + 2N + N)$. Since the activation and weight values are bounded by 0 and 1 and by s and s', respectively, one can take as state space the bounded Borel set

$\Omega = [\min(0, s), \max(1, s')]^K \subseteq \mathbb{R}^K$. The dynamics of the network can thus be expressed by the transition function $f : \Omega \times \{0,1\}^2 \to \Omega$ defined by $f(\mathbf{p}, \mathbf{a}) = \sigma(\mathbf{A}(\mathbf{p}) \cdot \mathbf{x}(\mathbf{p}) + \mathbf{b}(\mathbf{p}) \cdot \mathbf{a} + \mathbf{c}(\mathbf{p}))$, where $\mathbf{x}(\mathbf{p})$ is the activation value vector and $\mathbf{A}(\mathbf{p})$, $\mathbf{b}(\mathbf{p})$, and $\mathbf{c}(\mathbf{p})$ are the weight matrices, all of them extracted from the $(N+2)$ first and (N^2+2N+N) last components of state $\mathbf{p}$, respectively. The function f is clearly continuous (and measurable) for each fixed input $\mathbf{a}$. Therefore, the state space Ω and the transition function f satisfy the requirements described in Refs. 23–25, showing that the computational power of plastic RNNs will indeed be reduced to regular and definite languages in the presence of either arbitrarily small or larger amounts of noise.

Point (b). We begin with the case of polynomial time computation. First, note that the addition of some discrete source of stochasticity to any of the considered plastic neural network model does clearly not decrease the computational power of the model below **P/poly**. Formally, if $L \in$ **P/poly**, then by Corollary 7, L is decidable in polynomial time by some Pl-RNN $\mathcal{N}$, and hence, by incorporating in $\mathcal{N}$ some "inefficient" or "harmless" source of stochasticity, L is also decidable in polynomial time by some stochastic Pl-RNN. We now show that the addition of some discrete source of stochasticity does not increase the computational power of plastic neural networks above **P/poly**. Let L be some language that is ε-recognized[h] in polynomial time by some stochastic Pl-RNN $\mathcal{N}$ of any kind (bi-valued or general, rational or real). By a generalization of Lemma 7.1 of Ref. 15, there exists a family of nonstochastic feedforward St-RNN[$\mathbb{R}$]s $\{\mathcal{F}_n\}_{n=0}^{\infty}$ that decides L in polynomial time p, in the sense that each network $\mathcal{F}_n$ classifies correctly all possible inputs of length n in time $p(n)$. Intuitively, each $\mathcal{F}_n$ is obtained by unfolding the network $\mathcal{N}$ into $p(n)$ layers, copying this unfolding cn times for some constant c (which depends on ε), assigning to each copy some specific fixed values for its stochastic gates which ensure that a majority of the copies always answers correctly on all possible inputs of length n, and gathering as final answer the majority of the answers of the copies. Then, by a direct generalization of Lemma 7.2 of Ref. 40, the behavior of this family of feedforward St-RNN[$\mathbb{R}$]s $\{\mathcal{F}_n\}_{n=0}^{\infty}$ can be simulated by a single St-RNN[$\mathbb{R}$] $\mathcal{N}$. The St-RNN[$\mathbb{R}$] $\mathcal{N}$ decides L with a quadratic slowdown, but still in polynomial time. By Theorem 2(a), $L \in$ **P/poly**.

We now consider the case of exponential time computation. Let $L \subseteq \{0,1\}^+$ be some arbitrary language. By Corollary 7, L is decidable in exponential time by some Pl-RNN $\mathcal{N}$ of any kind (bi-valued or general, rational or real). Hence, by incorporating in $\mathcal{N}$ some "inefficient" or "harmless" source of stochasticity, L is also decidable in exponential time by some stochastic Pl-RNN of any kind (bi-valued or general, rational or real). $\square$

[h]See Ref. 15 for the precise definition of ε-recognition.

References

1. L. F. Abbott and S. B. Nelson, Synaptic plasticity: Taming the beast, *Nat. Neurosci.* **3 Suppl** (2000) 1178–1183.
2. S. J. Martin, P. D. Grimwood and R. G. M. Morris, Synaptic plasticity and memory: An evaluation of the hypothesis, *Annu. Rev. Neurosci.* **23**(1) (2000) 649–711.
3. A. Ertürk, C. P. Mauch, F. Hellal, F. Förstner, T. Keck, K. Becker, N. Jährling, H. Steffens, M. Richter, M. Hübener, E. Kramer, F. Kirchhoff, H. U. Dodt and F. Bradke, Three-dimensional imaging of the unsectioned adult spinal cord to assess axon regeneration and glial responses after injury, *Nat. Med.* **18** (2012) 166–171.
4. R. Perin, T. K. Berger and H. Markram, A synaptic organizing principle for cortical neuronal groups, *Proc. Natl. Acad. Sci. U.S.A.* **108**(13) (2011) 5419–5424.
5. S. McKenzie and H. Eichenbaum, Consolidation and reconsolidation: Two lives of memories? *Neuron* **71**(2) (2011) 224–233.
6. N. Kasabov, *Evolving Connectionist Systems — The Knowledge Engineering Approach*, 2nd edn. (Springer Verlag, 2007).
7. M. J. Watts, A decade of kasabov's evolving connectionist systems: A review. *IEEE Trans. Syst.*, *Man, Cybern. C* **39**(3) (2009) 253–269.
8. W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.* **5** (1943) 115–133.
9. S. C. Kleene, Representation of events in nerve nets and finite automata, in *Automata Studies*, eds. C. Shannon and J. McCarthy (Princeton University Press, Princeton, NJ, 1956), pp. 3–41.
10. J. V. Neumann, *The Computer and the Brain* (Yale University Press, New Haven, CT, USA, 1958).

11. M. L. Minsky, *Computation*: *Finite and Infinite Machines* (Prentice-Hall, Inc., Englewood Cliffs, NJ, 1967).
12. M. L. Minsky and S. Papert, *Perceptrons*: *An Introduction to Computational Geometry* (MIT Press, Cambridge, MA, USA, 1969).
13. H. T. Siegelmann and E. D. Sontag, Analog computation via neural networks, *Theor. Comput. Sci.* **131**(2) (1994) 331–360.
14. H. T. Siegelmann and E. D. Sontag, On the computational power of neural nets, *J. Comput. Syst. Sci.* **50**(1) (1995) 132–150.
15. H. T. Siegelmann, Stochastic analog networks and computational complexity, *J. Complexity* **15**(4) (1999) 451–475.
16. H. T. Siegelmann, *Neural Networks and Analog Computation*: *Beyond the Turing Limit* (Birkhauser Boston Inc., Cambridge, MA, USA, 1999).
17. P. Orponen, An overview of the computational power of recurrent neural networks, in *STeP'00*: *Proc. 9th Finnish AI Conf. 2000*, Vol. 3 (Finnish AI Society, 2000), pp. 89–96.
18. J. Cabessa and H. T. Siegelmann, The computational power of interactive recurrent neural networks, *Neural Comput.* **24**(4) (2012) 996–1019.
19. J. Cabessa and A. E. P. Villa, The expressive power of analog recurrent neural networks on infinite input streams, *Theor. Comput. Sci.* **436** (2012) 23–34.
20. H. T. Siegelmann, Computation beyond the Turing limit, *Science* **268**(5210) (1995) 545–548.
21. H. T. Siegelmann, Turing on super-Turing and adaptivity, *Prog. Biophys. Mol. Biol.* **113**(1) (2013) 117–126.
22. A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc.* **2**(42) (1936) 230–265.
23. W. Maass and P. Orponen, On the effect of analog noise in discrete-time analog computations, *Neural Comput.* **10**(5) (1998) 1071–1095.
24. W. Maass and E. D. Sontag, Analog neural nets with gaussian or other common noise distributions cannot recognize arbitary regular languages, *Neural Comput.* **11**(3) (1999) 771–782.
25. A. Ben-Hur, A. Roitershtein and H. T. Siegelmann, On probabilistic analog automata, *Theor. Comput. Sci.* **320**(2–3) (2004) 449–464.
26. A. Destexhe and E. Marder, Plasticity in single neuron and circuit computations, *Nature* **431** (2004) 789–795.
27. A. Holtmaat and K. Svoboda, Experience-dependent structural synaptic plasticity in the mammalian brain, *Nat. Rev. Neurosci.* **10** (2009) 647–658.
28. A. M. Turing, *Programmers' Handbook for Manchester Electronic Computer. Mark II* (University of Manchester, Manchester, UK, 1951).
29. A. M. Turing, Intelligent machinery, a heretical theory, *Philos. Math.* **4**(3) (1996) 256–260.
30. A. Turing, B. Richard, J. Geoffrey and N. Max, Can automatic calculating machines be said to think? *The Essential Turing*: *Seminal Writings in Computing*, *Logic*, *Philosophy*, *Artificial Intelligence*, *and Artificial Life Plus The Secrets of Enigma*, ed. J. B. Copeland (Oxford University Press, 2004).
31. A. M. Turing, Intelligent machinery, Technical report, National Physical Laboratory, Teddington, UK (1948).
32. J. B. Pollack, On connectionist models of natural language processing, Ph.D. thesis, Computing Reseach Laboratory, New Mexico State University (1987).
33. R. Hartley and H. Szu, A comparison of the computational power of neural network models, in *Proc. IEEE First Int. Conf. Neural Networks*, ed. C. Butler (IEEE, 1987), pp. 17–22.
34. M. Garzon and S. Franklin, Neural computability II, in *Proc. Third Int. Joint Conf. Neural Networks*, ed. O. Omidvar (IEEE, 1989), pp. 631–637.
35. S. Franklin and M. Garzon, Neural computability, in *Progress in Neural Networks*, ed. O. Omidvar (Ablex, Norwood, NJ, USA, 1989), pp. 128–144.
36. J. Schmidhuber, Dynamische neuronale netze und das fundamentale raumzeitliche lernproblem (dynamic neural nets and the fundamental spatiotemporal credit assignment problem), Ph.D. thesis, Institut für Informatik, Technische Universität München (1990).
37. J. L. Balcázar, R. Gavaldà and H. T. Siegelmann, Computational power of neural networks: A characterization in terms of kolmogorov complexity, *IEEE Trans. Inf. Theory* **43**(4) (1997) 1175–1183.
38. I. Tsuda, Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems, *Behav. Brain Sci.* **24**(5) (2001) 793–847.
39. I. Tsuda, Hypotheses on the functional roles of chaotic transitory dynamics, *Chaos* **19** (2009) 015113-1–015113-10.
40. Y. Yamaguti, S. Kuroda, Y. Fukushima, M. Tsukada and I. Tsuda, A mathematical model for Cantor coding in the hippocampus, *Neural Netw.* **24**(1) (2011) 43–53.
41. S. Arora and B. Barak, *Computational Complexity — A Modern Approach* (Cambridge University Press, 2009).
42. Z. F. Mainen and T. J. Sejnowski, Reliability of spike timing in neocortical neurons, *Science* **268**(5216) (1995) 1503–1506.
43. B. J. Copeland, Hypercomputation, *Minds Mach.* **12**(4) (2002) 461–502.
44. J. Cabessa and H. T. Siegelmann, Evolving recurrent neural networks are super-Turing, *2011 Int. Joint Conf. Neural Networks* (*IJCNN*) (IEEE, 2011), pp. 3200–3206.

45. B. J. Copeland, Hypercomputation: Philosophical issues, *Theor. Comput. Sci.* **317**(1–3) (2004) 251–267.
46. A. Alexandridis, Evolving RBF neural networks for adaptive soft-sensor design, *Int. J. Neural Syst.* **23**(6) (2013).
47. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *Int. J. Neural Syst.* **19**(4) (2009) 295–308.
48. C.-M. Lin, A.-B. Ting, C.-F. Hsu and C.-M. Chung, Adaptive control for mimo uncertain nonlinear systems using recurrent wavelet neural network, *Int. J. Neural Syst.* **22**(1) (2012) 37–50.
49. N. R. Luque, J. A. Garrido, J. Ralli, J. J. Laredo and E. Ros, From sensors to spikes: Evolving receptive fields to enhance sensorimotor information in a robot-arm, *Int. J. Neural Syst.* **22**(4) (2012) 1250013-1–1250013–20.
50. A. Mohemmed, S. Schliebs, S. Matsuda and N. Kasabov, Span: Spike pattern association neuron for learning spatio-temporal spike patterns, *Int. J. Neural Syst.* **22**(4) (2012) 1250012-1–1250012-17.
51. A. Panakkat and H. Adeli, Recurrent neural network for approximate earthquake time and location prediction using multiple seismicity indicators, *Comput. Aided Civ. Infrastruct. Eng.* **24**(4) (2009) 280–292.
52. J. L. Rosselló, V. Canals, A. Morro and A. Oliver, Hardware implementation of stochastic spiking neural networks, *Int. J. Neural Syst.* **22**(4) (2012) 1250014-1–1250014-11.
53. D. C. Theodoridis, Y. S. Boutalis and M. A. Christodoulou, Dynamical recurrent neuro-fuzzy identification schemes employing switching parameter hopping, *Int. J. Neural Syst.* **22**(2) (2012) 1250004-1–1250004-16.
54. X. Zeng and Y. Zhang, Development of recurrent neural network considering temporal-spatial input dynamics for freeway travel time modeling, *Comput. Aided Civ. Infrastruct. Eng.* **28**(5) (2013) 359–371.