Proceedings of International Joint Conference on Neural Networks, San Jose, California, USA, July 31 – August 5, 2011

Evolving Recurrent Neural Networks are Super-Turing

Jérémie Cabessa
Computer Science Department
University of Massachusetts Amherst
jcabessa@cs.umass.edu

Hava T. Siegelmann
Computer Science Department
University of Massachusetts Amherst
hava@cs.umass.edu

Abstract— **The computational power of recurrent neural networks is intimately related to the nature of their synaptic weights. In particular, neural networks with static rational weights are known to be Turing equivalent, and recurrent networks with static real weights were proved to be super-Turing. Here, we study the computational power of a more biologically-oriented model where the synaptic weights can evolve rather than stay static. We prove that such evolving networks gain a super-Turing computational power, equivalent to that of static real-weighted networks, regardless of whether their synaptic weights are rational or real. These results suggest that evolution might play a crucial role in the computational capabilities of neural networks.**

I. Introduction

Neural networks' most interesting feature is their ability to change. Biological networks tune their synaptic strengths constantly. This mechanism – referred to as synaptic plasticity – is widely assumed to be intimately related to the storage and encoding of memory traces in the central nervous system [1], and synaptic plasticity provides the basis for most models of learning and memory in neural networks [2]. Moreover, this adaptive feature has also been translated to the artificial neural network context and used as a machine learning tool in many relevant applications [3].

As a first step towards the analysis of the computational power of such evolving networks, we consider a model of first-order recurrent neural networks provided with the additional property of evolution of synaptic weights which can update at any computational step. We prove that such evolving networks gain a super-Turing computational power.

More precisely, recurrent neural networks with unchanging rational weights were shown to be computationally equivalent to Turing machines, and their real-weighted counterparts are known to be super-Turing [4], [5], [6]. Here, we prove that allowing for the additional possibility for the synaptic weights to evolve also causes the corresponding networks to gain super-Turing capabilities. In fact, the evolving networks are capable of deciding all possible languages in exponential time of computation, and when restricted to polynomial time of computation, the networks decide precisely the complexity class of languages **P/poly**. Moreover, such evolving networks do not increase their computational power when translated from the rational to the real-weighted context. Therefore, both classes of rational and real-weighted evolving networks are super-Turing, and equivalent to real-weighted static recurrent networks. The results suggest that evolution might play a crucial role in the computational capabilities of neural networks.

This work was supported by the Swiss National Science Foundation (SNSF) Grant No. PBLAP2-132975, and by the Office of Naval Research (ONR) Grant No. N00014-09-1-0069.

II. Static Recurrent Neural Networks

We consider the classical model of first-order recurrent neural network presented in [4], [5], [6].

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

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

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

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

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

A *rational recurrent neural network* (RNN[$\mathbb{Q}$]) denotes a recurrent neural net whose all synaptic weights are rational numbers. An *real recurrent neural network* (RNN[$\mathbb{R}$]) is a network whose all synaptic weights are real. It has been proved that RNN[$\mathbb{Q}$] are Turing equivalent, and that RNN[$\mathbb{R}$]'s are strictly more powerful than RNN[$\mathbb{Q}$]'s, and hence also than Turing machines [4], [5].

The formal proofs of these results involve the consideration of a specific model of *formal network* that performs recognition and decision of formal languages, and thus allows mathematical comparison with the languages computed by Turing machines.

More precisely, the considered neural networks are equipped with two binary input processors: a data line u_d and a validation line u_v. The data line is used to carry the binary incoming input string; it carries the binary signal as long as it is present, and switches to value 0 when no more signal is present. The validation line is used to indicated when the data line is active; it takes value 1 as long as the incoming input string is present, and switches to value 0 thereafter.

Similarly, the networks are equipped with two binary output processors: a data line y_d and a validation line y_v. The data line provides the decision answer of the network concerning the current input string; it takes value 0 as long as no answer is provided, then possibly outputs 0 or 1 in order to accept or reject the current input, and next switches to value 0 thereafter. The validation line indicates the only moment when the data line is active; it takes value 1 at the precise decision time step of the network, and takes value 0 otherwise.

These formal networks can perform recognition and decision of formal languages[1]. Indeed, given some formal network $\mathcal{N}$ 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 given the input streams

$$\begin{aligned} u_d(0)u_d(1)u_d(2)\cdots &= u_0 \cdots u_k 000 \cdots \\ u_v(0)u_v(1)u_v(2)\cdots &= \underbrace{1\cdots 1}_{k+1} 000\cdots \end{aligned}$$

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

$$\begin{aligned} y_d(0)y_d(1)y_d(2)\cdots &= \underbrace{0\cdots 0}_{\tau-1} \eta_u 000\cdots \\ y_v(0)y_v(1)y_v(2)\cdots &= \underbrace{0\cdots 0}_{\tau-1} 1000\cdots \end{aligned}$$

where $\eta_u \in \{0,1\}$. The word u is said to be *accepted* or *rejected* by $\mathcal{N}$ if $\eta_u = 1$ or $\eta_u = 0$, respectively. The set of all words accepted by $\mathcal{N}$ is called the language *recognized* by $\mathcal{N}$. Moreover, for any proper complexity function $f : \mathbb{N} \longrightarrow \mathbb{N}$ and any language $L \subseteq \{0,1\}^+$, we say that L is *decided* by $\mathcal{N}$ in time f if and only if every word $u \in \{0,1\}^+$ is classified by $\mathcal{N}$ in time $\tau \leq f(|u|)$, and $u \in L \Leftrightarrow \eta_u = 1$. Naturally, a given language L is then said to be *decidable* by some network in time f if and only if there exists a RNN that decides L in time f.

Rational-weighted recurrent neural networks were proved to be computationally equivalent to Turing machines [5]. Indeed, on the one hand, any function determined by Equation (1) and involving rational weights is necessarily recursive, and thus can be computed by some Turing machine, and on the other hand, it was proved that any Turing machine can be simulated in linear time by some rational recurrent neural network. The result can be expressed as follows.

Theorem 1: Let L be some language. Then L is decidable by some RNN[$\mathbb{Q}$] if and only if L is decidable by some TM (i.e. L is recursive).

Furthermore, real-weighted recurrent neural networks were proved to be strictly more powerful than rational recurrent networks, and hence also than Turing machines. More precisely, they turn out to be capable of deciding all possible languages in exponential time of computation. When restricted to polynomial time of computation, the networks decide precisely the complexity class of languages **P/poly** [4].[2] Note that since **P/poly** strictly includes the class **P**, and even contains non-recursive languages [7], the networks are capable of super-Turing computational power already from polynomial time of computation. These results are summarized in the following theorem.

Theorem 2: (a) For any language L, there exists some RNN[$\mathbb{R}$] that decides L in exponential time.
(b) Let L be some language. Then $L \in$ **P/poly** if and only if L is decidable in polynomial time by some RNN[$\mathbb{R}$].

III. Evolving Recurrent Neural Networks

In the neural model governed by Equation (1), the number of neurons, the connectivity patterns between the neurons, and the strengths of the synaptic connections all remain *static* over time. We will now consider first-order recurrent neural networks provided with evolving (or adaptive) synaptic weights. This abstract neuronal model intends to capture the important notion of synaptic plasticity observed in various kind of neural networks. We will further prove that evolving (rational and real) recurrent neural network are computationally equivalent to (non-evolving) real recurrent neural networks. Therefore, evolving nets might also achieve super-Turing computational capabilities.

Formally, an *evolving recurrent neural network* (Ev-RNN) is a first-order recurrent neural network whose dynamics is governed by equations of the form

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

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. The boundness condition formally states that there exist two real constants s and s' such that $a_{ij}(t), b_{ij}(t), c_i(t) \in [s, s']$ for every $t \geq 0$. The values s and s' represent two extremal synaptic strengths that the network might never be able to overstep along its evolution.

An *evolving rational recurrent neural network* (Ev-RNN[$\mathbb{Q}$]) denotes an evolving recurrent neural net whose all

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

[2]The complexity class **P/poly** consists of the set of all languages decidable in polynomial time by some Turing machine with polynomially long advice (TM/poly(A)).

synaptic weights are rational numbers. An *evolving real recurrent neural network* (Ev-RNN[$\mathbb{R}$]) is an evolving network whose all synaptic weights are real.

Given some Ev-RNN $\mathcal{N}$, the description of the synaptic weights of network $\mathcal{N}$ at time t will be denoted by $\mathcal{N}(t)$. Moreover, we suppose that Ev-RNN's satisfy the formal input-output encoding presented in previous section. Therefore, the notions of language recognition and decision can be naturally transposed in the present case. Accordingly, we will provide a precise characterization of the computational power of Ev-RNN's.

For this purpose, we need a result that will be involved in the proof of forthcoming Lemma 3. The result is a straightforward generalization of the so-called "linear-precision suffices lemma" [4, Lemma 4.1], which plays a crucial role in the proof that RNN[$\mathbb{R}$]'s compute in polynomial time the class of languages **P/poly**. Before stating the result, the following definition is required. Given some Ev-RNN[$\mathbb{R}$] $\mathcal{N}$ and some proper complexity function f, an *f-truncated family over* $\mathcal{N}$ is a family of Ev-RNN[$\mathbb{Q}$]'s $\{\mathcal{N}_{f(n)} : n > 0\}$ such that: firstly, each net $\mathcal{N}_{f(n)}$ has the same processors and connectivity patterns as $\mathcal{N}$; secondly, for each $n > 0$, the rational synaptic weights of $\mathcal{N}_{f(n)}(t)$ are precisely those of $\mathcal{N}(t)$ truncated after $C \cdot f(n)$ bits, for some constant C (independent of n); thirdly, when computing, the activation values of $\mathcal{N}_{f(n)}$ are all truncated after $C \cdot f(n)$ bits at every time step. We then have the following result.

Lemma 1: Let $\mathcal{N}$ be some Ev-RNN[$\mathbb{R}$] that computes in time f. Then there exists an f-truncated family $\{\mathcal{N}_{f(n)} : n > 0\}$ of Ev-RNN[$\mathbb{Q}$]'s over $\mathcal{N}$ such that, for every input u and every $n > 0$, the binary output processors of $\mathcal{N}$ and $\mathcal{N}_{f(n)}$ have the very same activation values for all time steps $t \leq f(n)$.
Proof:[sketch] The proof is a generalization of that of [4, Lemma 4.1]. The idea is the following: since the evolving synaptic weights of $\mathcal{N}$ are by definition bounded over time by some constant W, then the truncation of the weights and activation values of $\mathcal{N}$ after $log(W) \cdot f(n)$ bits would indeed provide more and more precise approximation of the real activation values of of $\mathcal{N}$ as $f(n)$ increases, i.e. as n increases (f is a proper complexity function, hence monotone). Consequently, one can find a constant C related to $log(W)$ such that each "$(C \cdot f(n))$-truncated network" $\mathcal{N}_{f(n)}$ computes precisely like $\mathcal{N}$ up to time step $f(n)$. □

IV. The computational power of Evolving Recurrent Neural Networks

In this section, we first show that both rational and real Ev-RNN's are capable of deciding all possible languages in exponential time of computation. We then prove that the class of languages decided by rational and real Ev-RNN's in polynomial time corresponds precisely to the complexity class **P/poly**. It will directly follow from Theorem 2 that Ev-RNN[$\mathbb{Q}$]'s, Ev-RNN[$\mathbb{R}$]'s, and RNN[$\mathbb{R}$]'s have equivalent super-Turing computational powers both in polynomial time as well as exponential time of computation. We make the whole proof for the case of Ev-RNN[$\mathbb{Q}$]'s. The same results concerning Ev-RNN[$\mathbb{R}$]'s will directly follow.

Proposition 1: For any language $L \subseteq \{0,1\}^+$, there exists some Ev-RNN[$\mathbb{Q}$] that decides L in exponential time.
Proof: The main idea of the proof is is illustrated in Figure 1. First of all, for every $n > 0$, we need to encode the subset $L \cap \{0,1\}^n$ of words of lenght n of L into a rational number $q_{L,n}$. We proceed as follows. Given the lexicographical enumeration $w_1, \ldots, w_{2^n}$ of $\{0,1\}^n$, we first encode the set $L \cap \{0,1\}^n$ into the finite word $w_{L,n} = w_1\varepsilon_1 w_2\varepsilon_2 \cdots w_{2^n}\varepsilon_{2^n}$, where ε_i is the *L-characteristic bit* $\chi_L(w_i)$ of w_i given by $\varepsilon_i = 1$ if $w_i \in L$ and $\varepsilon_i = 0$ if $w_i \notin L$. Note that $length(w_{L,n}) = 2^n \cdot (n+1)$. Then, we consider the following rational number

$$q_{L,n} = \sum_{i=1}^{2^n \cdot (n+1)} \frac{2 \cdot w_{L,n}(i) + 1}{4^i}.$$

Note that $q_{L,n} \in]0,1[$ for all $n > 0$. Also, the encoding procedure ensures that $q_{L,n} \neq q_{L,n+1}$, since $w_{L,n} \neq w_{L,n+1}$, for all $n > 0$. Moreover, it can be shown that the finite word $w_{L,n}$ can be decoded from the value $q_{L,n}$ by some Turing machine, or equivalently, by some rational recurrent neural network [4], [5].

We provide the description of an Ev-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ that decides L in exponential time. The network $\mathcal{N}_L$ actually consists of one evolving and one non-evolving rational sub-network connected together. More precisely, the evolving rational-weighted part of $\mathcal{N}_L$ is made up of a single designated processor x_e. The neuron x_e receives as sole incoming synaptic connection a background activity of evolving intensity $c_i(t)$. The synaptic weight $c_i(t)$ successively takes the rational bounded values $q_{L,1}, q_{L,2}, q_{L,3}, \ldots$, by switching from value $q_{L,k}$ to $q_{L,k+1}$ after every K time steps, for some suitable constant $K > 0$ to be described.

Moreover, the non-evolving rational-weighted part of $\mathcal{N}_L$ is designed in order to perform the following recursive procedure: for any finite input u provided bit by bit, the sub-network first stores in its memory the successive incoming bits $u(0), u(1), \ldots$ of u, and simultaneously counts the number of bits of u as well as the number of successive distinct values $q_{L,1}, q_{L,2}, q_{L,3}, \ldots$ taken by the activation values of the neuron x_e. After the input has finished being processed, the sub-network knows the length n of u. It then waits for the n-th value $q_{L,n}$ to appear, then stores the value $q_{L,n}$ in its memory in one time step when it occurs (this can be done whatever the complexity of $q_{L,n}$), next decodes the finite word $w_{L,n}$ from the value $q_{L,n}$, and finally outputs the L-characteristic bit $\chi_L(u)$ of u written in the word $w_{L,n}$. Note that a constant time of K time steps between any $q_{L,i}$ and $q_{L,i+1}$ can indeed be chosen in order to provide enough time for the sub-network to successfully decide if the current value $q_{L,i}$ has to be stored or not, and if yes, to be able to store it before the next value $q_{L,i+1}$ has occurred. Note also that the equivalence between Turing machines and rational recurrent neural networks ensures that the above recursive procedure can indeed be performed by some non-evolving rational recurrent neural sub-network [5].

The network $\mathcal{N}_L$ clearly decides the language L, since it finally outputs the L-characteristic bit of the incoming input. Moreover, since the word $w_{L,n}$ has length $2^n \cdot (n+1)$, the decoding procedure of $w_{L,n}$ works in time $O(2^n)$, for any input of length n. All other tasks take no more than $O(2^n)$ time steps. Therefore, the network $\mathcal{N}_L$ decides the language L in exponential time. ☐

We now prove that the class of languages decidable by Ev-RNN[$\mathbb{Q}$]'s in polynomial time corresponds precisely to the complexity class of languages **P/poly**.

Lemma 2: Let $L \subseteq \{0,1\}^+$ be some language. If $L \in$ **P/poly**, then there exists an Ev-RNN[$\mathbb{Q}$] that decides L in polynomial time.

Proof: The present proof resembles the proof of Proposition 1. The main idea of the proof is illustrated in Figure 2. First of all, since $L \in$ **P/poly**, there exists a Turing machine with polynomially long advice (TM/poly(A)) $\mathcal{M}$ that decides L in polynomial time. Let $\alpha : \mathbb{N} \longrightarrow \{0,1\}^+$ be the polynomially long advice function of $\mathcal{M}$, and for each $n > 0$, consider the following rational number

$$q_{\alpha(n)} = \sum_{i=1}^{length(\alpha(n))} \frac{2 \cdot \alpha(n)(i) + 1}{4^i}.$$

We can assume without loss of generality that the advice function of $\mathcal{M}$ satisfies $\alpha(n) \neq \alpha(n+1)$ for all $n > 0$, and thus the encoding procedure ensures that $q_{\alpha(n)} \neq q_{\alpha(n+1)}$ for all $n > 0$. Moreover, $q_{\alpha(n)} \in]0,1[$ for all $n > 0$, and the finite word $\alpha(n)$ can be decoded from the value $q_{\alpha(n)}$ in a recursive manner [4], [5].

We now provide the description of an Ev-RNN[$\mathbb{Q}$] $\mathcal{N}_L$ that decides L in polynomial time. Once again, the network $\mathcal{N}_L$ consists of one evolving and one non-evolving rational sub-network connected together. The evolving rational-weighted part of $\mathcal{N}_L$ is made up of a single designated processor x_e. The neuron x_e receives as sole incoming synaptic connection a background activity of evolving intensity $c_i(t)$. The synaptic weight $c_i(t)$ successively takes the rational bounded values $q_{\alpha(1)}, q_{\alpha(2)}, q_{\alpha(3)}, \ldots$, by switching from value $q_{\alpha(k)}$ to $q_{\alpha(k+1)}$ after every K time steps, for some large enough constant $K > 0$.

Moreover, the non-evolving rational-weighted part of $\mathcal{N}_L$ is designed in order to perform the following recursive procedure: for any finite input u provided bit by bit, the sub-network first stores in its memory the successive incoming bits $u(0), u(1), \ldots$ of u, and simultaneously counts the number of bits of u as well as the number of successive distinct values $q_{\alpha(1)}, q_{\alpha(2)}, q_{\alpha(3)}, \ldots$ taken by the activation values of the neuron x_e. After the input has finished being processed, the sub-network knows the length n of u. It then waits for the n-th synaptic value $q_{\alpha(n)}$ to occur, then stores $q_{\alpha(n)}$ in its memory in one time step when it appears, next decodes the finite word $\alpha(n)$ from the value $q_{\alpha(n)}$, simulates the behavior of the TM/poly(A) $\mathcal{M}$ on u with $\alpha(n)$ written on its advice tape, and finally outputs the answer of that computation. Note that the equivalence between Turing machines and rational recurrent neural networks ensures that the above recursive procedure can indeed be performed by some non-evolving rational recurrent neural sub-network [5].

Since $\mathcal{N}_L$ outputs the same answer as $\mathcal{M}$ and $\mathcal{M}$ decides the language L, it follows that $\mathcal{N}_L$ clearly also decides L. Besides, since the advice is polynomial, the decoding procedure of the advice word performed by $\mathcal{N}_L$ can be done in polynomial time in the input size. 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 [5]. Consequently, $\mathcal{N}_L$ decides L in polynomial time. ☐

Lemma 3: Let $L \subseteq \{0,1\}^+$ be some language. If there exists an Ev-RNN[$\mathbb{Q}$] that decides L in polynomial time, then $L \in$ **P/poly**.

Proof: The main idea of the proof is illustrated in Figure 3. Suppose that L is decided by some Ev-RNN[$\mathbb{Q}$] $\mathcal{N}$ in polynomial time p. Since $\mathcal{N}$ is by definition also an Ev-RNN[$\mathbb{R}$], Lemma 1 applies and shows the existence of a p-truncated family of Ev-RNN[$\mathbb{Q}$]'s over $\mathcal{N}$. Hence, for every n, there exists an Ev-RNN[$\mathbb{Q}$] $\mathcal{N}_{p(n)}$ such that: firstly, the network $\mathcal{N}_{p(n)}$ has the same processors and connectivity pattern as $\mathcal{N}$; secondly, 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; thirdly, on every input of lenght 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 $\mathcal{N}_{p(n)}$ and $\mathcal{N}$ still have the very 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, consider the oracle function $\alpha : \mathbb{N} \longrightarrow \{0,1\}^+$ given by $\alpha(i) = Encoding(\langle \mathcal{N}_{p(i)}(t) : 0 \leq t \leq p(i) \rangle)$, where $Encoding(\langle \mathcal{N}_{p(i)}(t) : 0 \leq t \leq p(i) \rangle)$ 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 $p(i)$ 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 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 of $\mathcal{N}_{p(n)}$ by $\mathcal{M}$ are performed in polynomial time in n.

Now, since any u of lenght n is classified by $\mathcal{N}$ in time $p(n)$, Lemma 1 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 $L \in$ **P/poly**, which concludes the proof. $\square$

Lemmas 2 and 3 directly induce the following characterization of the computational power of Ev-RNN[$\mathbb{Q}$]'s in polynomial time.

Proposition 2: Let $L \subseteq \{0,1\}^+$ be some language. Then L is decidable by some Ev-RNN[$\mathbb{Q}$] in polynomial time if and only if $L \in$ **P/poly**

Now, propositions 1 and 2 show that Ev-RNN[$\mathbb{Q}$]'s are capable of super-Turing computational capabilities both in polynomial as well as in exponential time of computation. Since (non-evolving) RNN[$\mathbb{Q}$]'s were only capable of Turing capabilities, these features suggests that evolution might play a crucial role in the computational capabilities of neural networks. The results are summarized in the following theorem.

Theorem 3: (a) For any language L, there exists some Ev-RNN[$\mathbb{Q}$] that decides L in exponential time.

(b) Let L be some language. Then $L \in$ **P/poly** if and only if L is decidable in polynomial time by some Ev-RNN[$\mathbb{Q}$].

Furthermore, since any Ev-RNN[$\mathbb{Q}$] is also by definition an Ev-RNN[$\mathbb{R}$], it follows that Proposition 1 and Lemma 2 can directly be generalized in the case of Ev-RNN[$\mathbb{R}$]'s. Also, since Lemma 1 is originally stated for the case of Ev-RNN[$\mathbb{R}$]'s, it follows that Lemma 3 can also be generalized in the context of Ev-RNN[$\mathbb{R}$]'s. Therefore, propositions 1 and 2 also hold for the case of Ev-RNN[$\mathbb{R}$]'s, meaning that rational and real evolving recurrent neural networks have an equivalent super-Turing computational power both in polynomial as well as in exponential time of computation. Finally, theorems 2 and 3 show that this computational power is the same as that of RNN[$\mathbb{R}$]'s, as stated by the following result.

Theorem 4: RNN[$\mathbb{R}$]'s, Ev-RNN[$\mathbb{Q}$]'s, and Ev-RNN[$\mathbb{R}$]'s have equivalent super-Turing computational powers both in polynomial as well as in exponential time of computation.

V. Conclusion

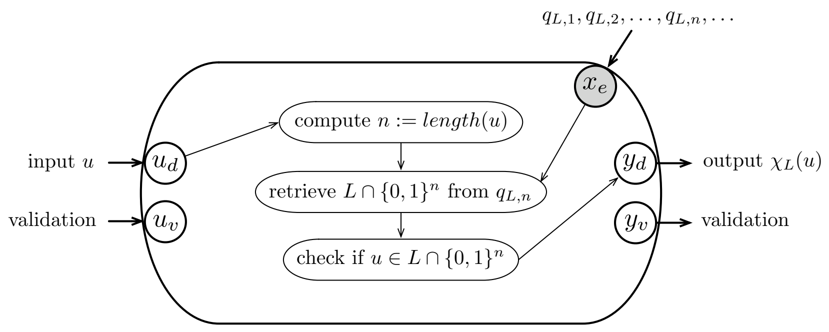
We proved that evolving recurrent neural networks are super-Turing. They are capable of deciding all possible languages in exponential time of computation, and they decide in polynomial time of computation the complexity class of languages **P/poly**. It follows that evolving rational networks, evolving real networks, and static real networks have the very same super-Turing computational powers both in polynomial as well as exponential time of computation. They are all strictly more powerful than rational static networks, which are Turing equivalent. These results indicate that evolution might play a crucial role in the computational capabilities of neural networks.

Of specific interest is the rational-weighted case, where the evolving property really brings up an additional super-Turing computational power to the networks. These capabilities arise from the theoretical possibility to consider non-recursive evolving patters of the synaptic weights. Indeed, the consideration of restricted evolving patterns driven by recursive procedures would necessarily constrain the corresponding networks to Turing computational capabilities. Therefore, according to our model, the existence of super-Turing capabilities of the networks depends on the possibility of having non-recursive evolving patterns in nature. Moreover, it has been shown that the super-Turing computational powers revealed by the consideration of, on the one hand, static real synaptic weights, and on the other hand, evolving rational synaptic weights turn out to be equivalent. This fact can be explained as follows: on the one side, the whole evolution of a rational-weighted synaptic connection can indeed be encoded into a single static real synaptic weight; one the other side, any static real synaptic weight can be approximated by a converging evolving sequence of more and more precise rational weights.

In the real-weighted case, the evolving property doesn't bring any additional computational power, since the static networks were already super-Turing. This feature can be explained by the fact that any infinite sequence of evolving real weights can be encoded to a single static real weight. This feature reflects the fundamental difference between rational and real numbers: limit points of rational sequences are not necessarily rational, whereas limit points of real sequences are always real.

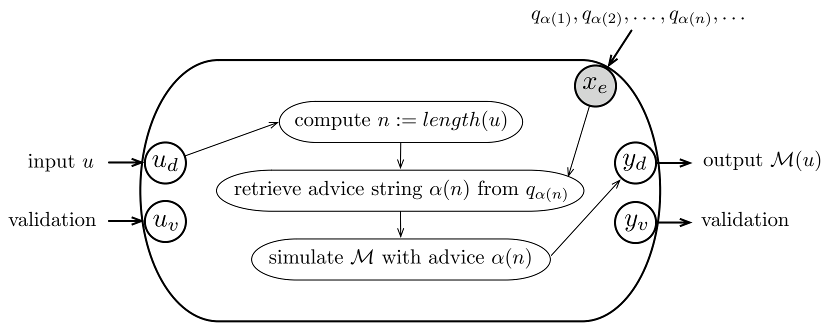
Furthermore, the fact that Ev-RNN[$\mathbb{Q}$]'s are strictly stronger than RNN[$\mathbb{Q}$]'s but still not stronger than RNN[$\mathbb{R}$]'s provides a further evidence in supportive the Thesis of Analog Computation [4], [8]. This thesis is analogous to the Church-Turing thesis, but in the realm of analog computation. It state that no reasonable abstract analog device can be more powerful than RNN[$\mathbb{R}$]'s.

The present work can be extended significantly. As a first step, we intend to study other specific evolving paradigms of weighted-connections. For instance, the consideration of an input dependent evolving framework could be of specific interest, for it would bring us closer to the important concept of adaptability of networks. More generally, we also envision to extend the possibility of evolution to other important aspects of the architectures of the networks, like the number of neurons (to capture neural birth and death), etc. Ultimately, the combination of all such evolving features would provide a better understanding of the computational power of more and more biologically-oriented models of neural networks.



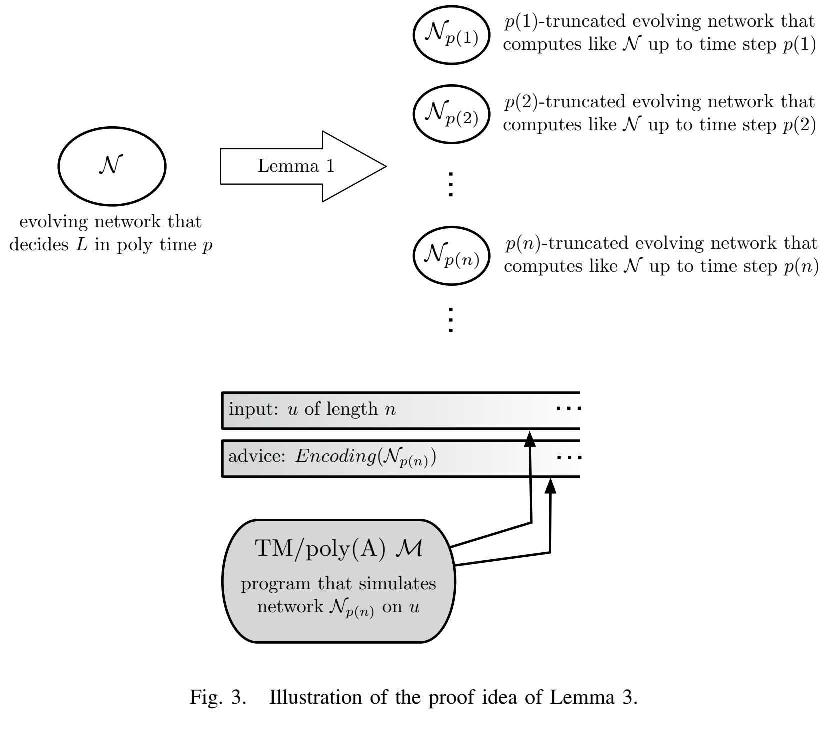
$q_{L,1}, q_{L,2}, \ldots, q_{L,n}, \ldots$
x_e
input u
u_d
validation
u_v
compute $n := length(u)$
retrieve $L \cap \{0,1\}^n$ from $q_{L,n}$
check if $u \in L \cap \{0,1\}^n$
y_d
output $\chi_L(u)$
y_v
validation

Fig. 1. Illustration of the network $\mathcal{N}_L$ described in the proof of Proposition 1.



$q_{\alpha(1)}, q_{\alpha(2)}, \ldots, q_{\alpha(n)}, \ldots$
x_e
input u
u_d
validation
u_v
compute $n := length(u)$
retrieve advice string $\alpha(n)$ from $q_{\alpha(n)}$
simulate $\mathcal{M}$ with advice $\alpha(n)$
y_d
output $\mathcal{M}(u)$
y_v
validation

Fig. 2. Illustration of the network $\mathcal{N}_L$ described in the proof of Lemma 2.



$\mathcal{N}_{p(1)}$
$p(1)$-truncated evolving network that computes like $\mathcal{N}$ up to time step $p(1)$
$\mathcal{N}_{p(2)}$
$p(2)$-truncated evolving network that computes like $\mathcal{N}$ up to time step $p(2)$
$\mathcal{N}$
evolving network that decides L in poly time p
Lemma 1
$\mathcal{N}_{p(n)}$
$p(n)$-truncated evolving 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. 3. Illustration of the proof idea of Lemma 3.

References

[1] S. J. Martin, P. D. Grimwood, and R. G. M. Morris, “Synaptic Plasticity and Memory: An Evaluation of the Hypothesis,” *Annual Review of Neuroscience*, vol. 23, pp. 649–711, 2000.

[2] L. F. Abbott and S. B. Nelson, “Synaptic plasticity: taming the beast,” *Nature Neuroscience*, vol. 3, pp. 1178–1183, 2000.

[3] B. Widrow and M. Lehr, “30 years of adaptive neural networks: perceptron, madaline, and backpropagation,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, Sep. 1990.

[4] H. T. Siegelmann and E. D. Sontag, “Analog computation via neural networks,” *Theor. Comput. Sci.*, vol. 131, no. 2, pp. 331–360, 1994.

[5] ——, “On the computational power of neural nets,” *J. Comput. Syst. Sci.*, vol. 50, no. 1, pp. 132–150, 1995.

[6] H. T. Siegelmann, *Neural networks and analog computation: beyond the Turing limit*. Cambridge, MA, USA: Birkhauser Boston Inc., 1999.

[7] O. Goldreich, *Introduction to Complexity Theory: Lecture notes*. Unpublished lecture notes, 1999.

[8] H. T. Siegelmann, “Computation beyond the Turing limit,” *Science*, vol. 268, no. 5210, pp. 545–548, 1995.