Information Sciences 711 (2025) 122104

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Refined Kolmogorov complexity of analog, evolving and stochastic recurrent neural networks

Jérémie Cabessa [a,b,*], Yann Strozecki [a]

[a] Laboratoire DAVID, UVSQ – University Paris-Saclay, 78035 Versailles, France
[b] Institute of Computer Science of the Czech Academy of Sciences, 18207 Prague 8, Czech Republic

ARTICLE INFO

Keywords:
Recurrent neural networks
Echo state networks
Computational power
Computability theory
Analog computation
Stochastic computation
Kolmogorov complexity
Quantization

ABSTRACT

Kolmogorov complexity measures the compressibility of real numbers. We provide a refined characterization of the hypercomputational power of analog, evolving, and stochastic neural networks based on the Kolmogorov complexity of their real weights, evolving weights, and real probabilities, respectively. First, we retrieve the infinite hierarchy of complexity classes of analog networks, defined in terms of the Kolmogorov complexity of their real weights. This hierarchy lies between the complexity classes **P** and **P/poly**. Next, using a natural identification between real numbers and infinite sequences of bits, we generalize this result to evolving networks, obtaining a similar hierarchy of complexity classes within the same bounds. Finally, we extend these results to stochastic networks that employ real probabilities as randomness, deriving a new infinite hierarchy of complexity classes situated between **BPP** and **BPP/log***. Beyond providing examples of such hierarchies, we describe a generic method for constructing them based on classes of functions of increasing complexity. As a practical application, we show that the predictive capabilities of recurrent neural networks are strongly impacted by the quantization applied to their weights. Overall, these results highlight the relationship between the computational power of neural networks and the intrinsic information contained by their parameters.

1. Introduction

Certain brain processes can be regarded as computational in nature. As Churchland and Sejnowski note, "The idea that brains are computational in nature has spawned a range of explanatory hypotheses in theoretical neurobiology" [1]. In this context, the question of the computational capabilities of neural networks naturally arises, among many others.

Since the early 1940s, theoretical research on neural computation has focused on comparing the computational power of neural network models to that of abstract computing machines. In 1943, McCulloch and Pitts modeled the nervous system as a finite interconnection of logical devices and analyzed the computational capabilities of "nets of neurons" from a logical perspective [2]. Following this approach, Kleene and Minsky demonstrated that recurrent neural networks composed of McCulloch-Pitts (Boolean) neurons are computationally equivalent to finite-state automata [3,4]. These results laid the foundation for a subsequent line of research focused on implementing abstract machines on parallel hardware architectures (see e.g., [5,6]).

In 1948, Turing introduced the B-type unorganized machine, a kind of neural network composed of interconnected NAND neuronal-like units [7]. He suggested that sufficiently large B-type unorganized machines could simulate the behavior of a uni-

* Corresponding author.
E-mail addresses: jeremie.cabessa@uvsq.fr (J. Cabessa), yann.strozecki@uvsq.fr (Y. Strozecki).

https://doi.org/10.1016/j.ins.2025.122104
Received 13 July 2023; Received in revised form 15 March 2025; Accepted 15 March 2025

Available online 19 March 2025

versal Turing machine with limited memory. The Turing universality of neural networks involving infinitely many Boolean neurons has been further investigated [8]. Moreover, Turing anticipated the concepts of "learning" and "training", which would later become central to machine learning. These concepts took shape with the introduction of the *perceptron*, a formal neuron that can be trained to discriminate inputs using Hebbian-like learning [9,10]. However, the computational limitations of the perceptron tempered enthusiasm for artificial neural networks [11]. This led to a prolonged decline in interest in the field until the 1980s, when the popularization of the backpropagation algorithm, among other factors, paved the way for the success of deep learning [12].

In the late 1950s, von Neumann proposed an alternative perspective on brain information processing, combining digital and analog computation [13]. Building on this idea, Siegelmann and Sontag investigated the computational capabilities of *sigmoidal neural networks* – as opposed to Boolean ones – and demonstrated that recurrent neural networks composed of linear-sigmoid neurons with rational synaptic weights are Turing complete [14]. This result was later extended to a broader class of sigmoidal networks [15].

Building on advancements in analog computation [16], Siegelmann and Sontag proposed that the variables involved in underlying chemical and physical phenomena could be modeled using continuous, rather than discrete (rational) numbers. They introduced the concept of an *analog neural network* – a sigmoidal recurrent neural network with real, rather than rational, weights. They demonstrated that analog neural networks are computationally equivalent to Turing machines with advice, thus operating in polynomial time within the complexity class **P/poly** [17]. Consequently, analog networks possess *super-Turing* capabilities, enabling them to capture chaotic dynamical features beyond the reach of Turing machines [18]. Based on these findings, Siegelmann and Sontag formulated the Thesis of Analog Computation – analogous to the Church-Turing thesis in the realm of analog computation – asserting that no reasonable abstract analog device can surpass the computational power of first-order analog recurrent neural networks [16,17]. These results were later generalized to the context of infinite computation [19].

Inspired by the learning process in neural networks, Cabessa and Siegelmann explored the computational capabilities of *evolving neural networks* [20]. They demonstrated that evolving neural networks with binary, rational or real evolving weights are computationally equivalent to analog neural networks, and hence, decide the class **P/poly** in polynomial time.

The computational power of stochastic neural networks has been studied in detail. For rational-weighted networks, the introduction of a discrete source of stochasticity increases the computational power from **P** to **BPP/log***. However, for real-weighted networks, the computational capabilities remain unchanged at the **P/poly** level [21]. In contrast, the presence of analog noise significantly reduces the computational power of these systems, limiting them to the capabilities of finite state automata, or even below [22,23].

The refined computational capabilities of recurrent neural networks have been investigated as well. On the one hand, the sub-Turing capabilities of Boolean rational-weighted networks with 0, 1, 2, or 3 additional sigmoidal cells have been explored [24,25]. On the other hand, the super-Turing computational power of analog neural networks has been refined in terms of the Kolmogorov complexity of their underlying real weights [26]. As the Kolmogorov complexity of the weights increases, the capabilities of these analog networks span the gap between the complexity classes **P** and **P/poly**.

The computational capabilities of spiking neural networks (as opposed to sigmoidal ones) have also been extensively studied [27]. In this approach, computational states are encoded in the temporal differences between spikes, rather than in the activation values of the cells. Maass characterized both lower and upper bounds on the complexity of networks composed of classical and noisy spiking neurons [28,29]. Additionally, he demonstrated that networks of spiking neurons are capable of simulating analog recurrent neural networks [28]. In a more bio-inspired context, automata and Turing completeness have been achieved with spiking neural networks composed of cell assemblies [30,31].

In the early 2000s, Păun introduced the concept of a P system – a highly parallel abstract model of computation inspired by the membrane-like structure of the biological cell – leading to the emergence of a highly active field of research [32]. The capabilities of various models of so-called neural P systems have been studied [33]. In particular, neural P systems, equipped with a bio-inspired source of acceleration, have been shown to possess hypercomputational capabilities, spanning all levels of the arithmetical hierarchy [34].

In terms of practical applications, recurrent neural networks are natural candidates for sequential tasks, involving time series or textual data for instance. Classical recurrent architectures, like LSTM and GRU, have been applied with great success in many contexts [35]. A three-level formal hierarchy of the sub-Turing expressive capabilities of these architectures, based on the notions of space complexity and rational recurrence, has been established [36]. Echo state networks (ESNs), which are the machine learning counterpart to liquid state machines (LSMs), are another type of recurrent neural network that has gained increasing popularity due to their training efficiency [37,38]. The universality of ESNs has been studied from the perspective of functional analysis, showing that they are capable of approximating different classes of filters for infinite discrete-time signals [39].

In this paper, we revisit and extend the Kolmogorov-based complexity approach to recurrent neural networks, introduced by Balcázar et al. [26]. Specifically, Kolmogorov complexity measures the compressibility of real numbers. In this context, we provide a refined characterization of the super-Turing computational power of analog, evolving, and stochastic neural networks, based on the Kolmogorov complexity of their real weights, evolving weights, and real probabilities, respectively. First, we retrieve the infinite hierarchy of complexity classes of analog networks, defined in terms of the Kolmogorov complexity of their underlying real weights [26]. This hierarchy lies between the complexity classes **P** and **P/poly**. Next, using a natural identification between real numbers and infinite sequences of bits, we generalize this result to evolving networks, obtaining a novel similar hierarchy of complexity classes within the same bounds. Finally, we extend these results to stochastic networks that employ real probabilities as a source of randomness, and derive a new infinite hierarchy of complexity classes situated between **BPP** and **BPP/log***. In practical settings, where real-valued weights cannot be manipulated exactly, weight approximation and compression techniques serve as practical counterparts to the theoretical concept of Kolmogorov complexity. In a context of time series prediction, we show that the predictive capabilities of recurrent neural networks are strongly impacted by the amount of quantization applied to their weights.

Beyond establishing the existence and providing concrete examples of such hierarchies, we describe a general method for constructing them based on function classes of increasing complexity. Technically, the separability between non-uniform complexity classes is achieved through a diagonalization argument, generalizing the previous approach [26]. Overall, these results provide a unified perspective on the refined computational capabilities of analog, evolving, and stochastic neural networks. They emphasize the relationship between the computational power of recurrent neural networks and the intrinsic information encoded in their parameters. They further reinforce the view of recurrent neural networks as a natural model for hypercomputation [40].

This paper is organized as follows: Section 2 discusses the related works. Section 3 introduces the mathematical concepts necessary for this study. Section 4 presents recurrent neural networks within the framework of echo state networks (ESNs). Section 5 introduces the various models of analog, evolving, and stochastic recurrent neural networks, highlighting their connections to non-uniform complexity classes defined by Turing machines with advice. Section 6 presents the hierarchy theorems, which lead to the description of strict hierarchies of analog, evolving, and stochastic neural network classes. Section 7 explores the practical implications of our theoretical insights by examining the predictive performance of ESNs as a function of their quantized weights. Finally, Section 8 provides a discussion and concluding remarks.

2. Related works

The computational capabilities of recurrent neural networks (RNNs) have been extensively studied over the years. Kleene and Minsky first established the equivalence between Boolean recurrent neural networks and finite state automata [3,4]. Later, Siegelmann and Sontag proved that rational-weighted neural networks are Turing universal [14]. This result was further extended to encompass a broader class of sigmoidal neural networks, as well as a synaptic-based computational paradigm [15]. In the context of analog computation, Siegelmann and Sontag characterized the super-Turing capabilities of real-weighted neural networks [17,18]. Cabessa et al. extended these findings to evolving neural networks [20] and infinite computation [19]. The computational power of various stochastic [21] and noisy [22,23] neural networks has also been analyzed. Sima investigated the sub-Turing capabilities of Boolean networks augmented with a small number of sigmoidal neurons [24,25], while Balcázar et al. introduced a hierarchy of analog networks based on the Kolmogorov complexity of their real weights [26].

Furthermore, the computational power of spiking neural networks has been extensively investigated [27]. The computational power of bio-inspired neural architectures has also been studied [30,31]. Additionally, since the early 2000s, research on P systems – particularly neural P systems – has expanded rapidly, leading to a proliferation of models that are generally Turing complete [32,33].

Regarding modern architectures, a hierarchy of the sub-Turing expressive power of LSTM and GRU networks has been established [36]. Additionally, the universality of echo state networks has been analyzed through the lens of universal approximation theorems [39].

3. Preliminaries

The binary alphabet is denoted by $\Sigma = \{0, 1\}$, and the set of finite words, finite words of length n, infinite words, and finite or infinite words over Σ are denoted by Σ^*, Σ^n, Σ^ω, and $\Sigma^{\leq\omega}$, respectively. Given some finite or infinite word $w \in \Sigma^{\leq\omega}$, the i-th bit of w is denoted by w_i, the sub-word from index i to index j is $w[i : j]$, and the length of w is $|w|$, with $|w| = \infty$ if $w \in \Sigma^\omega$.

A *Turing machine (TM)* is defined in the usual way. A *Turing machine with advice (TM/A)* is a TM provided with an additional advice tape and function $\alpha : \mathbb{N} \to \Sigma^*$. On every input $w \in \Sigma^n$ of length n, the machine first queries its advice function $\alpha(n)$, writes this word on its advice tape, and then continues its computation according to its finite program. The advice α is said to be *prefix* if $m \leq n$ implies that $\alpha(m)$ is a prefix of $\alpha(n)$, for all $m, n \in \mathbb{N}$. The advice α is called *unbounded* if the length of the successive advice words tends to infinity, i.e., if $\lim_{n\to\infty} |\alpha(n)| = \infty$.[1] In this work, we assume that every advice α is prefix and unbounded, which ensures that the infinite word $\lim_{n\to\infty} \alpha(n) \in \Sigma^\omega$ is well-defined. For any non-decreasing function $f : \mathbb{N} \to \mathbb{N}$, the advice α is said to be of *size* f if $|\alpha(n)| = f(n)$, for all $n \in \mathbb{N}$. Let poly be the set of univariate polynomials with integer coefficients and log be the set of functions of the form $n \to C \log(n)$ where $C \in \mathbb{N}$. The advice α is called *polynomial* or *logarithmic* if it is of size $f \in$ poly or $f \in$ log, respectively. In this wwork, any TM/A $\mathcal{M}$ equipped with some prefix unbounded advice is assumed to satisfy the following additional consistency property: for any input $w \in \Sigma^n$ of length n, $\mathcal{M}$ accepts w using advice $\alpha(n)$ iff $\mathcal{M}$ accepts w using advice $\alpha(n')$, for all $n' \geq n$.

The class of languages decidable in polynomial time by some TM is $\mathbf{P}$. The class of languages decidable in time $t : \mathbb{N} \to \mathbb{N}$ by some TM/A with advice α is denoted by $\mathbf{TMA}[\alpha, t]$. Given some class of advice functions $\mathcal{A} \subseteq (\Sigma^*)^{\mathbb{N}}$ and some class of time functions $\mathcal{T} \subseteq \mathbb{N}^{\mathbb{N}}$, we naturally define

$$\mathbf{TMA}\left[\mathcal{A}, \mathcal{T}\right] = \bigcup_{\alpha\in\mathcal{A}} \bigcup_{t\in\mathcal{T}} \mathbf{TMA}\left[\alpha, t\right] .$$

The class of languages decidable in polynomial time by some TM/A with polynomial prefix and non-prefix advice are $\mathbf{P/poly}^*$ and $\mathbf{P/poly}$, respectively. It can be noticed that $\mathbf{P/poly}^* = \mathbf{P/poly}$.

A *probabilistic Turing machine (PTM)* is a TM with two transition functions. At each computational step, the machine chooses one or the other transition function with probability $\frac{1}{2}$, independently from all previous choices, and updates its state, tapes' contents, and heads accordingly. A PTM $\mathcal{M}$ is assumed to be a *decider*, meaning that for any input w, all possible computations of $\mathcal{M}$ end up

[1] Note that if α is not unbounded, then it can be encoded into the program of a TM, and thus, doesn't add any computational power to the TM model.

either in an accepting or in a rejecting state. The random variable corresponding to the decision (0 or 1) that $\mathcal{M}$ makes at the end of its computation over w is denoted by $\mathcal{M}(w)$. Given some language $L \subseteq \Sigma^*$, we say that the PTM $\mathcal{M}$ *decides* L *in time* $t : \mathbb{N} \to \mathbb{N}$ if, for every $w \in \Sigma^*$, $\mathcal{M}$ halts in $t(|w|)$ steps regardless of its random choices, and $Pr[\mathcal{M}(w) = 1] \geq \frac{2}{3}$ if $w \in L$ and $Pr[\mathcal{M}(w) = 0] \geq \frac{2}{3}$ if $w \notin L$. The class of languages decidable in polynomial time by some PTM is **BPP**. A *probabilistic Turing machine with advice (PTM/A)* is a PTM provided with an additional advice tape and function $\alpha : \mathbb{N} \to \Sigma^*$. The class of languages decided in time t by some PTM/A with advice α is denoted by $\mathbf{PTMA}[\alpha, t]$. Given some class of advice functions $\mathcal{A} \subseteq (\Sigma^*)^{\mathbb{N}}$ and some class of time functions $\mathcal{T} \subseteq \mathbb{N}^{\mathbb{N}}$, we also define

$$\mathbf{PTMA}\left[\mathcal{A}, \mathcal{T}\right] = \bigcup_{\alpha \in \mathcal{A}} \bigcup_{t \in \mathcal{T}} \mathbf{PTMA}\left[\alpha, t\right].$$

The class of languages decidable in polynomial time by some PTM/A with logarithmic prefix and non-prefix advice are $\mathbf{BPP/log}^*$ and $\mathbf{BPP/log}$, respectively. In this probabilistic case however, it can be shown that $\mathbf{BPP/log}^* \subsetneq \mathbf{BPP/log}$.

In the following, we will examine the size of the advice functions. Hence, we define the following *non-uniform complexity classes*.[2] Given a class of languages (or associated machines) $C \subseteq 2^{\Sigma^*}$ and a function $f : \mathbb{N} \to \mathbb{N}$, we say that $L \in C/f^*$ if there exist some $L' \in C$ and some prefix advice function $\alpha : \mathbb{N} \to \Sigma^*$ such that, for all $n \in \mathbb{N}$ and for all $w \in \Sigma^n$, the following properties hold:

(i) $|\alpha(n)| = f(n)$, for all $n \geq 0$;
(ii) $w \in L \Leftrightarrow \langle w, \alpha(n)\rangle \in L'$;
(iii) $\langle w, \alpha(n)\rangle \in L' \Leftrightarrow \langle w, \alpha(k)\rangle \in L'$, for all $k \geq n$.

Given a class of functions $\mathcal{F} \subseteq \mathbb{N}^{\mathbb{N}}$, we naturally set

$$C/\mathcal{F}^* = \bigcup_{f \in \mathcal{F}} C/f^*.$$

For instance, the class of languages decidable in polynomial time by some Turing machine (resp. probabilistic Turing machines) with prefix advice of size f is $\mathbf{P}/f^*$ (resp. $\mathbf{BPP}/f^*$). The non-starred complexity classes C/f and $C/\mathcal{F}$ are defined analogously, except that the prefix property of α and the last condition are not required.

For any word $w = w_0 w_1 w_2 \cdots \in \Sigma^{\leq\omega}$, the *base-2* and *base-4 encoding* functions $\delta_2 : \Sigma^{\leq\omega} \to [0,1]$ and $\delta_4 : \Sigma^{\leq\omega} \to [0,1]$ are respectively defined by

$$\delta_2(w) = \sum_{i=0}^{|w|} \frac{w_i + 1}{2^{i+1}} \quad \text{and} \quad \delta_4(w) = \sum_{i=0}^{|w|} \frac{2w_i + 1}{4^{i+1}}.$$

It can be shown that $\delta_4 : \Sigma^{\leq\omega} \to [0,1]$ is injective, which means that no two words have the same base-4 encoding [14]. Defining $\Delta := \delta_4(\Sigma^\omega) \subseteq [0,1]$ ensures that the restriction $\delta_4 : \Sigma^{\leq\omega} \to \Delta$ is bijective, which guarantees that δ^{-1} is well-defined on the domain Δ. For any real $r \in \Delta$, its *base-4 expansion* is defined as $\bar{r} = \delta_4^{-1}(r) = r_0 r_1 r_2 \cdots \in \Sigma^\omega$. For any $R \subseteq \Delta$, we define $\bar{R} = \delta_4^{-1}(R) = \{\bar{r} : r \in R\} \subseteq \Sigma^\omega$.

Finally, for any probability space Ω and any events $A_1, \ldots, A_n \subseteq \Omega$, the probability that at least one of the events A_i occurs can be bounded using the *union bound*:

$$\Pr\left(\bigcup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} \Pr(A_i).$$

4. Recurrent neural networks

We consider a specific model of recurrent neural networks that follows the *echo state network* architecture [37]. More specifically, a *recurrent neural network* consists of an input layer, a pool of interconnected neurons – often referred to as the *reservoir* – and an output layer, as illustrated in Fig. 1. These networks process finite words over the alphabet $\Sigma = \{0, 1\}$ using an input-output encoding described below. Consequently, they are capable of recognizing formal languages.

Definition 1. *A* rational-weighted recurrent neural network (RNN) *is a tuple*

$$\mathcal{N} = \left(\mathbf{x}, \mathbf{h}, \mathbf{y}, \mathbf{W_{in}}, \mathbf{W_{res}}, \mathbf{W_{out}}, \mathbf{h}^0\right)$$

where

- $\mathbf{x} = (x_0, x_1)$ *is a sequence of two input cells, the data input* x_0 *and the validation input* x_1*;*
- $\mathbf{h} = (h_0, \ldots, h_{K-1})$ *is a sequence of* K *hidden cells, referred to as the reservoir;*

[2] This definition is non-standard. Usually, non-uniform complexity classes are defined with respect to a class of advice functions $\mathcal{H} \subseteq (\Sigma^*)^{\mathbb{N}}$ instead of a class of advice functions' size $\mathcal{F} \subseteq \mathbb{N}^{\mathbb{N}}$.

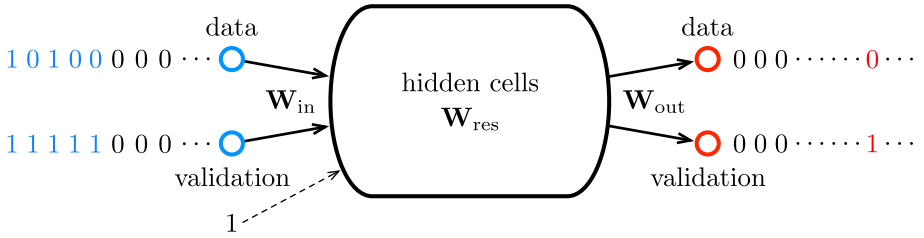

Fig. 1. A recurrent neural network. The network is composed of two Boolean input cells (data and validation), two Boolean output cells (data and validation) and a set of K hidden cells, the reservoir, that are recurrently interconnected. The weight matrices $\mathbf{W_{in}}$, $\mathbf{W_{res}}$, and $\mathbf{W_{out}}$ labeling the connections between these layers are represented. In this illustration, the network reads the finite word 10100 by means of its data and validation input cells (blue), and eventually rejects it, as shown by the pattern of the data and validation output cells (red).

- $\mathbf{y} = (y_0, y_1)$ *is a sequence of two output cells, the data output* y_0 *and the validation output* y_1*;*
- $\mathbf{W_{in}} \in \mathbb{Q}^{K\times(2+1)}$ *is a matrix of input weights and biases, where* w_{ij} *is the weight from input* x_j *to cell* h_i*, for* $j \neq 2$*, and* w_{i2} *is the bias of* h_i*;*
- $\mathbf{W_{res}} \in \mathbb{Q}^{K\times K}$ *is a matrix of internal weights, where* w_{ij} *is the weight from cell* h_j *to cell* h_i*;*
- $\mathbf{W_{out}} \in \mathbb{Q}^{2\times K}$ *is a matrix of output weights, where* w_{ij} *is the weight from cell* h_j *to output* y_i*;*
- $\mathbf{h}^0 = (h_0^0, \ldots, h_{K-1}^0) \in [0,1]^K$ *is the initial state of* $\mathcal{N}$*, where each component* h_i^0 *is the initial activation value of cell* h_i*.*

The *activation value* of the cell x_i, h_j and y_k at time t is denoted by $x_i^t \in \{0,1\}$, $h_j^t \in [0,1]$ and $y_k^t \in \{0,1\}$, respectively. Note that activation values of input and output cells are Boolean, as opposed to those of the hidden cells. The *input*, *output* and *(hidden) state* of $\mathcal{N}$ at time t are the vectors

$$\mathbf{x}^t = (x_0^t, x_1^t) \in \mathbb{B}^2, \quad \mathbf{y}^t = (y_0^t, y_1^t) \in \mathbb{B}^2 \text{ and } \mathbf{h}^t = (h_0^t, \ldots, h_{K-1}^t) \in [0,1]^K,$$

respectively. Given some input $\mathbf{x}^t$ and state $\mathbf{h}^t$ at time t, the state $\mathbf{h}^{t+1}$ and the output $\mathbf{y}^{t+1}$ at time $t+1$ are computed by the following equations:

$$\mathbf{h}^{t+1} = \sigma\left(\mathbf{W_{in}}(\mathbf{x}^t : 1) + \mathbf{W_{res}}\mathbf{h}^t\right) \tag{1}$$

$$\mathbf{y}^{t+1} = \theta\left(\mathbf{W_{out}}\mathbf{h}^{t+1}\right) \tag{2}$$

where $(\mathbf{x}^t : 1)$ denotes the vector $(x_0^t, x_1^t, 1)$, and σ and θ are the *linear sigmoid* and the *hard-threshold* functions respectively given by

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

The constant value 1 in the input vector $(\mathbf{x}^t : 1)$ ensures that the hidden cells $\mathbf{h}$ receive the last column of $\mathbf{W_{in}} \in \mathbb{Q}^{K\times(2+1)}$ as biases at each time step $t \geq 0$. The bias of cell h_i will be denoted as w_{i2}.

An *input* $\mathbf{x}$ of length n for the network $\mathcal{N}$ is an infinite sequence of inputs at successive time steps $t = 0, 1, 2, \ldots$, where the first n data bits correspond to the finite word transmitted to $\mathcal{N}$, the first n validation bits are set to 1, and the remaining data and validation bits are set to 0. Formally, an *input* $\mathbf{x}$ is defined as

$$\mathbf{x} = \mathbf{x}^0\mathbf{x}^1 \cdots \mathbf{x}^{n-1}\mathbf{0}^\omega \in \left(\mathbb{B}^2\right)^\omega$$

where $\mathbf{x}^i = (x_0^i, 1)$ with $x_0^i \in \{0,1\}$ for $i = 0, \ldots, n-1$, and $\mathbf{0} = (0,0)$. Suppose that the network $\mathcal{N}$ starts in the initial state $\mathbf{h}^0$ and processes the input $\mathbf{x}$ step by step. The dynamics given by Equations (1) and (2) ensures that $\mathcal{N}$ generates the sequences of states and outputs

$$\mathbf{h} = \mathbf{h}^0\mathbf{h}^1\mathbf{h}^2 \cdots \in \left([0,1]^K\right)^\omega$$

$$\mathbf{y} = \mathbf{y}^1\mathbf{y}^2\mathbf{y}^3 \cdots \in \left(\mathbb{B}^2\right)^\omega$$

step by step, where $\mathbf{y} := \mathcal{N}(\mathbf{x})$ is the *output* of $\mathcal{N}$ associated with input $\mathbf{x}$.

Let $w = w_0 w_1 \cdots w_{n-1} \in \Sigma^n$ be a finite word of length n and let $\tau \in \mathbb{N}$ be an integer. The word $w \in \Sigma^*$ can naturally be associated with the input

$$\mathbf{w} = \mathbf{w}^0\mathbf{w}^1 \cdots \mathbf{w}^{n-1}\mathbf{0}^\omega \in \left(\mathbb{B}^2\right)^\omega$$

defined by $\mathbf{w}^i = (x_0^i, x_1^i) = (w_i, 1)$, for $i = 0, \ldots, n-1$. In other words, the input $\mathbf{w}$ corresponds to the pattern

$$\begin{array}{llllllllllll} x_0^0 & x_0^1 & x_0^2 & \cdots & = & w_0 & w_1 & \cdots & w_{n-1} & 0 & 0 & 0 & \cdots \\ x_1^0 & x_1^1 & x_1^2 & \cdots & = & 1 & 1 & \cdots & 1 & 0 & 0 & 0 & \cdots \end{array}$$

where data bits represent the successive input bits transmitted to the network and the validation bits indicate whether a data bit is actively being processed or not (see Fig. 1). The word w is said to be *accepted* or *rejected by* $\mathcal{N}$ *in time* τ if the output

$$\mathcal{N}(\mathbf{w}) = \mathbf{y} = \mathbf{y}^1\mathbf{y}^2 \cdots \mathbf{y}^\tau \mathbf{0}^\omega \in \left(\mathbb{B}^2\right)^\omega$$

satisfies $\mathbf{y^i} = (0,0)$, for $i = 1, \ldots, \tau - 1$, and $\mathbf{y}^\tau = (1,1)$ or $\mathbf{y}^\tau = (0,1)$, respectively. In this case, the output $\mathbf{y}$ corresponds to the pattern

$$\begin{array}{llllllllllll} y_0^0 & y_0^1 & y_0^2 & \cdots & = & 0 & 0 & \cdots & y_1^\tau & 0 & 0 & \cdots \\ y_1^0 & y_1^1 & y_1^2 & \cdots & = & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots \end{array}$$

where $y_1^\tau = 1$ or $y_1^\tau = 0$, respectively.

Let $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function. The word w is said to be *accepted* or *rejected by* $\mathcal{N}$ *in time* f if it is accepted or rejected in time $\tau \leq f(n)$, respectively.[3] A language $L \subseteq \Sigma^*$ is *decided by* $\mathcal{N}$ *in time* f if for every word $w \in \Sigma^*$,

$w \in L$ implies that $\mathbf{w}$ is accepted by $\mathcal{N}$ in time f and

$w \notin L$ implies that $\mathbf{w}$ is rejected by $\mathcal{N}$ in time f.

A language $L \subseteq \Sigma^*$ is *decided by* $\mathcal{N}$ *in polynomial time* if L is decided by $\mathcal{N}$ in time f, for some polynomial function $f : \mathbb{N} \to \mathbb{N}$. If it exists, the *language decided by* $\mathcal{N}$ *in time* f is denoted by $L_f(\mathcal{N})$. Besides, a network $\mathcal{N}$ is said to be a *decider* if any finite word is eventually accepted or rejected by it. In this case, the *language decided by* $\mathcal{N}$ is unique and is denoted by $L(\mathcal{N})$. We will assume that all the networks that we consider are deciders.

Recurrent neural networks with rational weights have been proven to be computationally equivalent to Turing machines [14].

Theorem 1. *Let $L \subseteq \Sigma^*$ be a language. The following conditions are equivalent:*

(i) L is decidable by some TM;
(ii) L is decidable by some RNN.

Proof. (sketch) (ii) → (i): The dynamics of any RNN $\mathcal{N}$ is governed by Equations (1) and (2), which involves only rational weights, and hence, can clearly be simulated by some Turing machine $\mathcal{M}$.

(i) → (ii): We provide a sketch of the original proof [14]. This proof is based on the fact that any finite (and infinite) binary word can be encoded into the activation value of a neuron, and decoded from this activation value bit by bit. This idea will be utilized in subsequent proofs. First of all, recall that any TM $\mathcal{M}$ is computationally equivalently to, and can be simulated in real time by, some p-stack machine $\mathcal{S}$ with $p \geq 2$. We thus show that any p-stack machine $\mathcal{S}$ can be simulated by some RNN $\mathcal{N}$. To this end, we encode every stack content

$$w = w_0 \cdots w_{n-1} \in \{0,1\}^*$$

as the rational number

$$q_w = \delta_4(w) = \sum_{i=0}^{n-1} \frac{2 \cdot w_i + 1}{4^{i+1}} \in [0,1].$$

For instance, $w = 1110$ is encoded into $q_w = \frac{3}{4} + \frac{3}{16} + \frac{3}{64} + \frac{1}{256}$. With this base-4 encoding, the required stack operations can be performed by simple functions involving the sigmoid-linear function σ, as described below:

- Reading the top of the stack: $\mathrm{top}(q_w) = \sigma(4q_w - 2)$
- Pushing 0 into the stack: $\mathrm{push}_0(q_w) = \sigma(\frac{1}{4}q_w + \frac{1}{4})$
- Pushing 1 into the stack: $\mathrm{push}_1(q_w) = \sigma(\frac{1}{4}q_w + \frac{3}{4})$
- Popping the stack: $\mathrm{pop}(q_w) = \sigma(4q_w - (2\mathrm{top}(q_w) + 1))$
- Testing emptiness of the stack: $\mathrm{empty}(q_w) = \sigma(4q_w)$

Hence, the content w of each stack can be encoded into the rational activation value q_w of a stack neuron, and the stack operations (reading the top, pushing 0 or 1, popping and testing the emptiness) can be performed by simple neural circuits implementing the functions described above.

We can therefore design an RNN $\mathcal{N}$ which correctly simulates the p-stack machine $\mathcal{S}$. The network $\mathcal{N}$ contains 3 neurons per stack: one for storing the encoded content of the stack, one for reading the top element of the stack, and one for storing the answer

[3] The choice of the letter f (instead of t) for referring to a computation time is deliberate, since the computation time of the networks will be related to the advice length of the Turing machines.

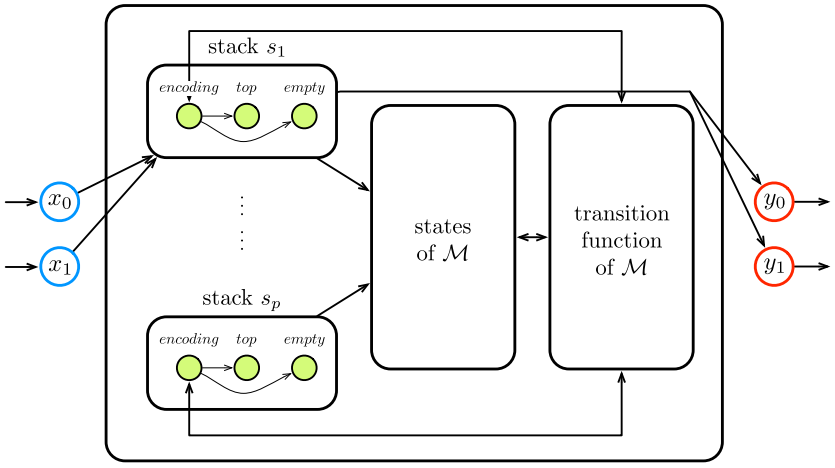

Fig. 2. Construction of an RNN simulating a p-stack machine.

of the emptiness test of the stack. Moreover, $\mathcal{N}$ contains two pools of neurons implementing the computational states and transition function of $\mathcal{S}$, respectively. For any computational state, input bit and stack contents, $\mathcal{N}$ computes the next computational state and updates the stack contents according to the transition function of $\mathcal{S}$. In this way, the network $\mathcal{N}$ simulates the behavior of the p-stack machine $\mathcal{S}$ correctly. The network $\mathcal{N}$ is illustrated in Fig. 2. □

The simulation process described in the proof of Theorem 1 is performed in real time. More precisely, if a language $L \subseteq \Sigma^*$ is decided by some TM in time $f(n)$, then L is decided by some RNN in time $f(n) + O(n)$. Hence, when restricted to polynomial time of computation, RNNs decide the complexity class **P**.

Corollary 2. *Let $L \subseteq \Sigma^*$ be a language. The following conditions are equivalent:*

(i) $L \in \mathbf{P}$;
(ii) *L is decidable by some RNN in polynomial time.*

5. Analog, evolving and stochastic recurrent neural networks

We now introduce analog, evolving and stochastic recurrent neural networks, which are all variants of the RNN model. In polynomial time, these models capture the complexity classes **P/poly**, **P/poly** and **BPP/log***, respectively, which strictly contain the class **P** and include non-recursive languages. Based on these considerations, these augmented models have been qualified as *super-Turing*. A tight characterization of these models in terms of Turing machines with specific kinds of advice is provided.

5.1. Analog networks

An *analog recurrent neural network (ANN)* is an RNN as defined in Definition 1, except that the weight matrices are real instead of rational [17]. Formally, an ANN is an RNN

$$\mathcal{N} = (\mathbf{x}, \mathbf{h}, \mathbf{y}, \mathbf{W_{in}}, \mathbf{W_{res}}, \mathbf{W_{out}}, \mathbf{h}^0)$$

such that

$$\mathbf{W_{in}} \in \mathbb{R}^{K\times(2+1)}, \ \mathbf{W_{res}} \in \mathbb{R}^{K\times K} \ \text{ and } \ \mathbf{W_{out}} \in \mathbb{R}^{2\times K}.$$

The definitions of word acceptance and rejection, as well as language recognition, are the same as for RNNs.

It can be shown that any ANN $\mathcal{N}$ containing the irrational weights $r_1, \ldots, r_k \in \mathbb{R} \setminus \mathbb{Q}$ is computationally equivalent to some ANN $\mathcal{N}'$ that uses only a single irrational weight $r \in \mathbb{R} \setminus \mathbb{Q}$, where $r \in \Delta \subseteq [0, 1]$ and r serves as the bias w_{02} of the hidden cell h_0 [17]. Hence, without loss of generality, we focus on such networks. Let $r \in \Delta$ and $R \subseteq \Delta$.

- ANN[r] denotes the class of ANNs such that all weights but w_{02} are rational and $w_{02} = r$.
- ANN[R] denotes the class of ANNs such that all weights but w_{02} are rational and $w_{02} \in R$.

In this definition, r is allowed to be a rational number. In this case, an ANN[r] is just a specific RNN.

In exponential time, analog recurrent neural networks can decide all possible languages. Specifically, any language $L \subseteq \Sigma^*$ can be encoded into the infinite word $\bar{r}_L \in \Sigma^\omega$, where the i-th bit of $\bar{r}_L$ is 1 if and only if the i-th word of Σ^* belongs to L, according to

some enumeration of Σ^*. Thus, we can construct an ANN containing the real weight $r_L = \delta_4(\bar{r}_L)$, which, for any input w, determines whether $w \in L$ or $w \notin L$ by decoding $\bar{r}_L$ and reading the corresponding bit. In polynomial time, however, ANNs decide the complexity class **P/poly** and are therefore computationally equivalent to Turing machines with polynomial advice (TM/poly(A)). The following result holds [17]:

Theorem 3. *Let $L \subseteq \Sigma^*$ be a language. The following conditions are equivalent:*

(i) $L \in \mathbf{P/poly}$;
(ii) *L is decidable by some ANN in polynomial time.*

Given an ANN $\mathcal{N}$ and some $q \in \mathbb{N}$, the *truncated network* $\mathcal{N}|_q$ is defined as the network $\mathcal{N}$ in which all weights and activation values are truncated to q precision bits at each step of the computation. The following result shows that, up to time q, the network $\mathcal{N}$ can restrict itself to $O(q)$ precision bits without affecting the outcome of its computation [17].

Lemma 4. *Let $\mathcal{N}$ be an ANN computing in time $f : \mathbb{N} \to \mathbb{N}$. Then, there exists a constant $c > 0$ such that, for every $n \in \mathbb{N}$ and every input $w \in \Sigma^n$, the networks $\mathcal{N}$ and $\mathcal{N}|_{cf(n)}$ produce the same outputs up to time $f(n)$.*

The computational relationship between analog neural networks and Turing machines with advice can actually be strengthened. To this end, for any non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ and any class of such functions $\mathcal{F}$, we define the following classes of languages that are decided by analog neural networks in time f and $\mathcal{F}$, respectively:

$$\mathbf{ANN}[r, f] = \left\{ L \subseteq \Sigma^\omega : L = L_f(\mathcal{N}) \text{ for some } \mathcal{N} \in \mathrm{ANN}[r] \right\}$$
$$\mathbf{ANN}[R, \mathcal{F}] = \bigcup_{r \in R} \bigcup_{f \in \mathcal{F}} \mathbf{ANN}[r, f].$$

In addition, for any real $r \in \Delta$ with base-4 expansion $\bar{r} = r_0 r_1 r_2 \cdots \in \Sigma^\omega$ and any function $f : \mathbb{N} \to \mathbb{N}$, the prefix advice $\alpha(\bar{r}, f) : \mathbb{N} \to \Sigma^*$ of length f associated with r is defined by

$$\alpha(\bar{r}, f)(n) = r_0 r_1 \cdots r_{f(n)-1}$$

for all $n \in \mathbb{N}$. For any set of reals $R \subseteq \Delta$ with base-4 expansions $\bar{R}$ and any class of functions $\mathcal{F} \subseteq \mathbb{N}^{\mathbb{N}}$, we naturally set

$$\alpha(\bar{R}, \mathcal{F}) = \bigcup_{\bar{r} \in \bar{R}} \bigcup_{f \in \mathcal{F}} \{\alpha(\bar{r}, f)\}$$

Conversely, note that any prefix unbounded advice $\alpha : \mathbb{N} \to \Sigma^*$ is of the form $\alpha(\bar{r}, f)$, where $\bar{r} = \lim_{n \to \infty} \alpha(n) \in \Sigma^\omega$ and $f : \mathbb{N} \to \mathbb{N}$ is defined by $f(n) = |\alpha(n)|$.

The following result clarifies the tight relationship between analog neural networks with real weights and Turing machines with related advices. Note that the real weights of the networks correspond precisely to the advice used by the machines, and the computation time of the networks is related to the advice length of the machines.

Proposition 5. *Let $r \in \Delta$ be a real weight and $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function.*

(i) $\mathbf{ANN}[r, f] \subseteq \mathbf{TMA}\left[\alpha(\bar{r}, cf), O(f^3)\right]$, *for some $c > 0$.*
(ii) $\mathbf{TMA}[\alpha(\bar{r}, f), f] \subseteq \mathbf{ANN}[r, O(f)]$.

Proof. (i) Let $L \in \mathbf{ANN}[r, f]$. Then, there exists an ANN[r] $\mathcal{N}$ such that $L_f(\mathcal{N}) = L$. By Lemma 4, there exists a constant $c > 0$ such that the network $\mathcal{N}$ and the truncated network $\mathcal{N}|_{cf(n)}$ produce the same outputs up to time step $f(n)$, for all $n \in \mathbb{N}$. Now, consider Procedure 1 below. In this procedure, all instructions except the query one (line 1) are recursive. Besides, the simulation of each step of $\mathcal{N}|_{cf(n)}$ involves a constant number of multiplications and additions of rational numbers, all representable by $cf(n)$ bits, and can thus be performed in time $O(f^2(n))$ (for the products). Consequently, the simulation of the $f(n)$ steps of $\mathcal{N}|_{cf(n)}$ can be performed in time $O(f^3(n))$. Hence, Procedure 1 can be simulated by some TM/A $\mathcal{M}$ using advice $\alpha(\bar{r}, cf)$ in time $O(f^3(n))$. In addition, Lemma 4 ensures that w is accepted by $\mathcal{M}$ iff w is accepted by $\mathcal{N}$, for all $w \in \Sigma^*$. Hence, $L(\mathcal{M}) = L(\mathcal{N}) = L$, and therefore $L \in \mathbf{TMA}\left[\alpha(\bar{r}, cf), O(f^3)\right]$.

(ii) Let $L \in \mathbf{TMA}[\alpha(\bar{r}, f), f]$. Then, there exists a TM/A $\mathcal{M}$ with advice $\alpha(\bar{r}, f)$ such that $L_f(\mathcal{M}) = L$. We show that $\mathcal{M}$ can be simulated by some analog neural network $\mathcal{N}$ with real weight r. The network $\mathcal{N}$ simulates the advice tape of $\mathcal{M}$ as described in the proof of Theorem 1: the left and right contents of the tape are encoded and stored into two stack neurons x_l and x_r, respectively, and the tape operations are simulated using appropriate neural circuits. On every input $w \in \Sigma^n$, the network $\mathcal{N}$ works as follows. First, $\mathcal{N}$ copies its real bias $r = \delta_4(r_0 r_1 r_2 \cdots)$ into neuron x_r. Every time $\mathcal{M}$ reads a new advice bit r_i, then $\mathcal{N}$ pops r_i from x_r, which thus updates its activation value to $\delta_4(r_{i+1} r_{i+2} \cdots)$, and pushes r_i into neuron x_l. This process is performed in constant time. At this point, neurons x_l and x_r contain the encodings of $r_0 r_1 \cdots r_i$ and $r_{i+1} r_{i+2} \cdots$, respectively. Then, $\mathcal{N}$ can simulate the recursive instructions of $\mathcal{M}$ in the usual way, in real time, until the next bit r_{i+1} is read [14]. Overall, $\mathcal{N}$ simulates the behavior of $\mathcal{M}$ in time $O(f(n))$.

Procedure 1

Input: input $w \in \Sigma^n$
Query the advice $\alpha(\bar{r}, cf)(n) = r_0 r_1 \cdots r_{cf(n)-1}$
for $t = 0, 1, \ldots, f(n) - 1$ **do**
  Simulate the truncated network $\mathcal{N}|_{cf(n)}$ which uses the rational approximation $\tilde{r} = \delta_4(r_0 r_1 \cdots r_{cf(n)-1})$ of r as its weight;
return Output of $\mathcal{N}|_{cf(n)}$ over w at time step $f(n)$

We now show that $\mathcal{M}$ and $\mathcal{N}$ output the same decision for input w. If $\mathcal{M}$ does not reach the end of its advice word $\alpha(n)$, the behaviors of $\mathcal{M}$ and $\mathcal{N}$ are identical, and so are their outputs. If at some point, $\mathcal{M}$ reaches the end of $\alpha(n)$ and reads successive blank symbols, then $\mathcal{N}$ continues to pop the successive bits $r_{|\alpha(n)|} r_{|\alpha(n)|+1} \cdots$ from neuron x_r, to push them into neuron x_l, and to simulates the behavior of $\mathcal{M}$. In this case, $\mathcal{N}$ simulates the behavior of $\mathcal{M}$ working with some extension of the advice $\alpha(n)$, which, by the consistency property of $\mathcal{M}$ (cf. Section 3), produces the same output as if $\mathcal{M}$ were working with the advice $\alpha(n)$. In this way, w is accepted by $\mathcal{M}$ iff w is accepted by $\mathcal{N}$, and thus $L(\mathcal{M}) = L(\mathcal{N})$. Therefore, $L \in \mathbf{ANN}[r, O(f)]$. □

The following corollary shows that the classes of languages decided in polynomial time by analog networks with real weights and by Turing machines with related advices are the same.

Corollary 6. *Let $r \in \Delta$ be a real weight and $R \subseteq \Delta$ be a set of real weights.*

(i) $\mathbf{ANN}\left[r, \text{poly}\right] = \mathbf{TMA}\left[\alpha(\bar{r}, \text{poly}), \text{poly}\right]$.
(ii) $\mathbf{ANN}\left[R, \text{poly}\right] = \mathbf{TMA}\left[\alpha(\bar{R}, \text{poly}), \text{poly}\right]$.

Proof. (i) Let $L \in \mathbf{ANN}\left[r, \text{poly}\right]$. Then, there exists $f \in \text{poly}$ such that $L \in \mathbf{ANN}[r, f]$. By Proposition 5–(i), $L \in \mathbf{TMA}[\alpha(\bar{r}, cf), O(f^3)]$, for some $c > 0$. Thus $L \in \mathbf{TMA}\left[\alpha(\bar{r}, \text{poly}), \text{poly}\right]$. Conversely, let $L \in \mathbf{TMA}\left[\alpha(\bar{r}, \text{poly}), \text{poly}\right]$. Then, there exist $f, f' \in \text{poly}$ such that $L \in \mathbf{TMA}\left[\alpha(\bar{r}, f'), f\right]$. By the consistency property of the TM/A, we can assume without loss of generality that $f' = f$. By Proposition 5–(ii), $L \in \mathbf{ANN}[r, O(f)]$, and hence $L \in \mathbf{ANN}\left[r, \text{poly}\right]$.
(ii) This point follows directly from point (i) by taking the union over all $r \in R$. □

5.2. Evolving networks

An *evolving recurrent neural network (ENN)* is an RNN where the weight matrices can evolve over time within a bounded space instead of remaining static [20]. Formally, an ENN is a tuple

$$\mathcal{N} = \left(\mathbf{x}, \mathbf{h}, \mathbf{y}, \left(\mathbf{W}^t_{\mathbf{in}}\right)_{t\in\mathbb{N}}, \left(\mathbf{W}^t_{\mathbf{res}}\right)_{t\in\mathbb{N}}, \left(\mathbf{W}^t_{\mathbf{out}}\right)_{t\in\mathbb{N}}, \mathbf{h}^0\right)$$

where $\mathbf{x}, \mathbf{h}, \mathbf{y}, \mathbf{h}^0$ are defined as in Definition 1, and

$$\mathbf{W}^t_{\mathbf{in}} \in \mathbb{Q}^{K\times(2+1)}, \ \mathbf{W}^t_{\mathbf{res}} \in \mathbb{Q}^{K\times K} \ \text{ and } \ \mathbf{W}^t_{\mathbf{out}} \in \mathbb{Q}^{2\times K}.$$

are input, reservoir and output weight matrices at time t, such that $\|\mathbf{W}^t_{\mathbf{in}}\|_{\max} = \|\mathbf{W}^t_{\mathbf{res}}\|_{\max} = \|\mathbf{W}^t_{\mathbf{out}}\|_{\max} \leq C$ for some constant $C > 1$ and for all $t \in \mathbb{N}$. The boundedness condition expresses the fact that synaptic weights are confined within a certain range of values imposed by the biological constitution of neurons. The successive values of an evolving weight w_{ij} are denoted by $(w^t_{ij})_{t\in\mathbb{N}}$. The dynamics of an ENN are given by the following adapted equations:

$$\mathbf{h}^{t+1} = \sigma\left(\mathbf{W}^t_{\mathbf{in}}(\mathbf{x}^t : 1) + \mathbf{W}^t_{\mathbf{res}}\mathbf{h}^t\right) \tag{3}$$

$$\mathbf{y}^{t+1} = \theta\left(\mathbf{W}^t_{\mathbf{out}}\mathbf{h}^{t+1}\right). \tag{4}$$

The definitions of word acceptance and rejection, as well as of language decision, are the same as those used for RNNs.

In this case, it can also be shown that any ENN $\mathcal{N}$ containing the evolving weights $\bar{e}_1, \ldots, \bar{e}_n \in [-C, C]^\omega$ is computationally equivalent to some ENN $\mathcal{N}'$ containing only one evolving weight $\bar{e} \in [-C, C]^\omega$, such that $\bar{e}$ evolves only between the binary values 0 and 1, i.e., $\bar{e} \in \Sigma^\omega$, and $\bar{e}$ represents the evolving bias $(w^t_{02})_{t\in\mathbb{N}}$ of the hidden cell h_0 [20]. Hence, without loss of generality, we restrict our attention to such networks. Let $\bar{e} \in \Sigma^\omega$ be a binary evolving weight and let $\bar{E} \subseteq \Sigma^\omega$.

- ENN$[\bar{e}]$ denoted the class of ENNs such that all weights but w_{02} are static, and $(w^t_{02})_{t\in\mathbb{N}} = \bar{e}$.
- ENN$[\bar{E}]$ denotes the class of ENNs such that all weights but w_{02} are static, and $(w^t_{02})_{t\in\mathbb{N}} \in \bar{E}$.

Like analog networks, evolving recurrent neural networks can decide any possible language in exponential time. In polynomial time, they decide the complexity class $\mathbf{P/poly}$, and are thus computationally equivalent to Turing machines with polynomial advice (TM/poly(A)). The following result holds [20]:

Theorem 7. *Let $L \subseteq \Sigma^*$ be a language. The following conditions are equivalent:*

(i) $L \in \mathbf{P/poly}$;
(ii) L *is decidable by some ENN in polynomial time.*

An analogous version of Lemma 4 holds for the case of evolving networks [20]. Note that the boundedness condition on the weights is involved in this result.

Lemma 8. *Let $\mathcal{N}$ be an ENN computing in time $f : \mathbb{N} \to \mathbb{N}$. Then, there exists a constant c such that, for every $n \in \mathbb{N}$ and every input $w \in \Sigma^n$, the networks $\mathcal{N}$ and $\mathcal{N}|_{cf(n)}$ produce the same outputs up to time $f(n)$.*

Once again, the computational relationship between evolving neural networks and Turing machines with advice can be strengthened. To this end, we define the following classes of languages decided by evolving neural networks in time f and $\mathcal{F}$, respectively:

$$\mathbf{ENN}[\bar{e}, f] = \left\{ L \subseteq \Sigma^\omega : L = L_f(\mathcal{N}) \text{ for some } \mathcal{N} \in \mathrm{ENN}[\bar{e}] \right\}$$
$$\mathbf{ENN}\left[\bar{E}, \mathcal{F}\right] = \bigcup_{\bar{e} \in \bar{E}} \bigcup_{f \in \mathcal{F}} \mathbf{ENN}[\bar{e}, f].$$

For any $\bar{e} \in \Sigma^\omega$ and any function $f : \mathbb{N} \to \mathbb{N}$, we consider the prefix advice $\alpha(\bar{e}, f) : \mathbb{N} \to \Sigma^*$ associated with e and f defined by

$$\alpha(\bar{e}, f)(n) = e_0 e_1 \cdots e_{f(n)-1}$$

for all $n \in \mathbb{N}$. Conversely, any prefix advice $\alpha : \mathbb{N} \to \Sigma^*$ is clearly of the form $\alpha(\bar{e}, f)$, where $\bar{e} = \lim_{n\to\infty} \alpha(n) \in \Sigma^\omega$ and $f(n) = |\alpha(n)|$ for all $n \in \mathbb{N}$.

The following relationships between neural networks with evolving weights and Turing machines with related advice hold:

Proposition 9. *Let $e \in \Sigma^\omega$ be a binary evolving weight and $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function.*

(i) $\mathbf{ENN}[\bar{e}, f] \subseteq \mathbf{TMA}\left[\alpha(\bar{e}, cf), O(f^3)\right]$, *for some $c > 0$.*
(ii) $\mathbf{TMA}[\alpha(\bar{e}, f), f] \subseteq \mathbf{ENN}[\bar{e}, O(f)]$.

Proof. The proof is very similar to that of Proposition 5.

(i) Let $L \in \mathbf{ENN}[\bar{e}, f]$. Then, there exists an $\mathrm{ENN}[\bar{e}]$ $\mathcal{N}$ such that $L_f(\mathcal{N}) = L$. By Lemma 8, there exists a constant $c > 0$ such that the network $\mathcal{N}$ and the truncated network $\mathcal{N}|_{cf(n)}$ produce the same outputs up to time step $f(n)$, for all $n \in \mathbb{N}$. Now, consider Procedure 2 below. In this procedure, all instructions except the query one are recursive. Procedure 2 can be simulated by some TM/A $\mathcal{M}$ using advice $\alpha(\bar{e}, f)$ in time $O(f^3(n))$, as described in the proof of Proposition 5. In addition, $\mathcal{M}$ and $\mathcal{N}$ decide the same language L, and therefore $L \in \mathbf{TMA}\left[\alpha(\bar{e}, f), O(f^3)\right]$.

Procedure 2

Input: input $w \in \Sigma^n$
Query the advice $\alpha(\bar{e}, f)(n) = e_0 e_1 \cdots e_{cf(n)-1}$
for $t = 0, 1, \ldots, f(n)$ **do**
  Simulate the truncated network $\mathcal{N}|_{cf(n)}$ which uses $w_{02}^t = e_t$ as the current value if its evolving weight;
return Output of $\mathcal{N}|_{cf(n)}$ over w at time step $f(n)$

(ii) Let $L \in \mathbf{TMA}[\alpha(\bar{e}, f), f]$. Then, there exists some TM/A $\mathcal{M}$ with advice $\alpha(\bar{e}, f)$ such that $L_f(\mathcal{M}) = L$. The machine $\mathcal{M}$ can be simulated by the network $\mathcal{N}$ with evolving weight $\bar{e} = e_0 e_1 e_2 \cdots$ as follows. First, $\mathcal{N}$ simultaneously counts and pushes into a stack neuron x_a the successive bits of $\bar{e}$ as they arrive. Then, for $k = 1, 2, \ldots$ and until it produces a decision, $\mathcal{N}$ proceeds as follows. If necessary, $\mathcal{N}$ waits for x_a to contain more than 2^k bits, copies the content $e_0 e_1 \cdots e_{2^k} \cdots$ of x_a in reverse order into another stack neuron $x_{a'}$, and simulates $\mathcal{M}$ with advice $e_0 e_1 \cdots e_{2^k} \cdots$ in real time. Every time $\mathcal{M}$ reads a new advice bit, $\mathcal{N}$ tries to access it from its stack $x_{a'}$. If $x_{a'}$ does not contain this bit, then $\mathcal{N}$ restarts the whole process with $k + 1$. When $k = \log(f(n))$, the stack x_a contains $2^k = f(n)$ bits, which ensures that $\mathcal{N}$ properly simulates $\mathcal{M}$ with advice $e_0 e_1 \cdots e_{f(n)-1}$. Hence, the whole simulation process is achieved in time $O(\sum_{k=1}^{\log(f(n))} 2^k) = O(2^{\log(f(n))+1}) = O(f(n))$. In this way, $\mathcal{M}$ and $\mathcal{N}$ decide the same language L, and $\mathcal{M}$ is simulated by $\mathcal{N}$ in time $O(f)$. Therefore, $L \in \mathbf{ENN}[\bar{e}, O(f)]$. □

The classes of languages decided in polynomial time by evolving networks and by Turing machines using related advices are the same.

Corollary 10. *Let $\bar{e} \in \Sigma^\omega$ be a binary evolving weight and $\bar{E} \subseteq \Sigma^\omega$ be a set of binary evolving weights.*

(i) $\mathbf{ENN}\left[\bar{e}, \mathrm{poly}\right] = \mathbf{TMA}\left[\alpha(\bar{e}, \mathrm{poly}), \mathrm{poly}\right]$.
(ii) $\mathbf{ENN}\left[\bar{E}, \mathrm{poly}\right] = \mathbf{TMA}\left[\alpha(\bar{E}, \mathrm{poly}), \mathrm{poly}\right]$.

Proof. The proof is similar to that of Corollary 6. □

5.3. Stochastic networks

A *stochastic recurrent neural network (SNN)* is an RNN as defined in Definition 1, except that the network contains additional stochastic cells as inputs [16]. Formally, an SNN is an RNN

$$\mathcal{N} = \left(\mathbf{x}, \mathbf{h}, \mathbf{y}, \mathbf{W_{in}}, \mathbf{W_{res}}, \mathbf{W_{out}}, \mathbf{h}^0\right)$$

such that $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_{k+1})$, where x_0 and x_1 are the data and validation input cells, respectively, and $x_2, \ldots, x_{k+1}$ are k additional stochastic cells. The weights are assumed to be rational, and the dimension of the input weight matrix is adapted, namely $\mathbf{W_{in}} \in \mathbb{Q}^{K\times((k+2)+1)}$. Each stochastic cell x_i is associated with a *real* probability $p_i \in [0,1]$, and at each time step $t \geq 0$, the activation of the cell x_i^t takes value 1 with probability p_i, and value 0 with probability $1 - p_i$. The dynamics of an SNN is then governed by Equations (1) and (2), but using $\mathbf{x}^t = (x_0^t, x_1^t, x_2^t, \cdots, x_{k+1}^t) \in \mathbb{B}^{k+2}$ as inputs, for all $t \geq 0$.

Consider some SNN $\mathcal{N}$ and some input $w = w_0 w_1 \cdots w_{n-1} \in \Sigma^n$. We assume that w is decided by $\mathcal{N}$ in the same amount of time $\tau(n)$, regardless of the random pattern of the stochastic cells $x_i^t \in \{0,1\}$, for $i = 2, \ldots, k+1$. Hence, the number of possible computations of $\mathcal{N}$ over w is finite. The input w is *accepted* (resp. *rejected*) by $\mathcal{N}$ if the number of accepting (resp. rejecting) computations out of the total number of computations on w is greater than or equal to $2/3$. This means that the error probability of $\mathcal{N}$ is bounded by $1/3$. If $f : \mathbb{N} \to \mathbb{N}$ is a non-decreasing function, we say that w is *accepted* or *rejected* by $\mathcal{N}$ *in time* f if it is accepted or rejected in time $\tau(n) \leq f(n)$, respectively. We assume that any SNN is a decider. The definition of language decision is the same as in the case of RNNs.

Once again, any SNN is computationally equivalent to some SNN with only one stochastic cell x_2 associated with a real probability $p \in \Delta$ [16]. Without loss of generality, we restrict our attention to such networks. Let $p \in \Delta$ be a probability and let $P \subseteq \Delta$.

- SNN$[p]$ denotes the class of SNNs such that the probability of the stochastic cell x_2 is equal to p.
- SNN$[P]$ denotes the class of SNNs such that the probability of the stochastic cell x_2 is equal to some $p \in P$.

In polynomial time, the SNNs with rational probabilities decide the complexity class **BPP**. In contrast, the SNNs with real probabilities decide the complexity class **BPP/log***, and hence, are computationally equivalent to probabilistic Turing machines with logarithmic advice (PTM/log(A)). If real weights are allowed (which is not the case in our definition), SNNs become equivalent to ANNs, regardless of whether their probabilities are rational or real, and hence, decide the class **P/poly**. Here, we focus on the following result [16]:

Theorem 11. *Let $L \subseteq \Sigma^*$ be a language. The following conditions are equivalent:*

(i) $L \in$ **BPP/log***;
(ii) *L is decidable by some SNN in polynomial time.*

As with the two previous models, we define the following classes of languages decided by stochastic neural networks in time f and $\mathcal{F}$, respectively:

$$\mathbf{SNN}[p, f] = \left\{ L \subseteq \Sigma^\omega : L = L_f(\mathcal{N}) \text{ for some } \mathcal{N} \in \text{SNN}[p] \right\}$$
$$\mathbf{SNN}[P, \mathcal{F}] = \bigcup_{p\in P} \bigcup_{f\in\mathcal{F}} \mathbf{SNN}[p, f].$$

The tight relationships between stochastic neural networks using real probabilities and Turing machines with related advices can now be described in detail. Note that, in this case, the advice of the machines is logarithmically related to the computation time of the networks.

Proposition 12. *Let $p \in \Delta$ be a real probability and $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function.*

(i) $\mathbf{SNN}[p, f] \subseteq \mathbf{PTMA}\left[\alpha(\bar{p}, \log(5f)), O(f^3)\right]$.
(ii) $\mathbf{PTMA}\left[\alpha(\bar{p}, \log(f)), f\right] \subseteq \mathbf{SNN}\left[p, O(f^2)\right]$.

Proof. (i) Let $L \in \mathbf{SNN}[p, f]$. Then, there exists an SNN$[p]$ $\mathcal{N}$ deciding L in time f. Note that the stochastic network $\mathcal{N}$ can be considered as a classical rational-weighted RNN with an additional input cell x_2. Since $\mathcal{N}$ has rational weights, it can be noticed that, up to time $f(n)$, the activation values of its neurons are always representable by rational numbers with $O(f(n))$ bits. Now, consider Procedure 3 below. This procedure can then be simulated by some PTM/A $\mathcal{M}$ using advice $\alpha(\bar{p}, \log(5f))$ in time $O(f^3)$, as described in the proof of Proposition 5.

It remains to show that $\mathcal{N}$ and $\mathcal{M}$ decide the same language L. For this purpose, consider a hypothetical device $\mathcal{M}'$ working as follows: at each time t, $\mathcal{M}'$ takes the sequence of bits $\bar{b}$ generated by Procedure 3 and concatenates it with some infinite sequence

Procedure 3

Input: input $w \in \Sigma^n$
Query the advice $\alpha(\bar{p}, \log(5f))(n) = \bar{p}_{\mathcal{M}} = p_0 p_1 \cdots p_{\log(5f(n))-1}$
for $t = 0, 1, \ldots, f(n) - 1$ **do**
  Draw a sequence of independent fair bits $\bar{b} := b_0 b_1 \cdots b_{\log(5f(n))-1}$
  if $\bar{b} <_{lex} \bar{p}_{\mathcal{M}}$ **then** $c_t = 1$
  else $c_t = 0$
  Simulate network $\mathcal{N}|_{cf(n)}$ at time t with $x_2^t = c_t$
return Output of $\mathcal{N}|_{cf(n)}$ over w at time step $f(n)$

of bits $\bar{b}' \in \Sigma^\omega$ drawn independently with probability $\frac{1}{2}$, producing the infinite sequence $\bar{b}\bar{b}' \in \Sigma^\omega$. Then, $\mathcal{M}'$ generates the bit $c'_t = 1$ iff $\bar{b}\bar{b}' <_{lex} \bar{p}$, which happens precisely with probability p, since $p = \delta_2(\bar{p})$ [41]. Finally, $\mathcal{M}'$ simulates the behavior of $\mathcal{N}$ at time t using the stochastic bit $x_2^t = c'_t$. Clearly, $\mathcal{M}'$ and $\mathcal{N}$ produce random bits with same probability p, behave in the same way, and thus decide the same language L. We now evaluate the error probability of $\mathcal{M}$ in deciding L by comparing the behaviors of $\mathcal{M}$ and $\mathcal{M}'$. Let $w \in \Sigma^n$ be some input and let

$$\bar{p}_{\mathcal{M}} := \alpha(\bar{p}, \log(5f))(n) = p_0 p_1 \cdots p_{\log(5f(n))-1} \quad \text{and} \quad p_{\mathcal{M}} = \delta_2(\bar{p}_{\mathcal{M}}).$$

According to Procedure 3, at each time step t, the machine $\mathcal{M}$ generates $c_t = 1$ iff $\bar{b} <_{lex} \bar{p}_{\mathcal{M}}$, which happens precisely with probability $p_{\mathcal{M}}$, since $p_{\mathcal{M}} = \delta_2(\bar{p}_{\mathcal{M}})$ [41]. On the other hand, $\mathcal{M}'$ generates $c'_t = 1$ with probability p, showing that $\mathcal{M}$ and $\mathcal{M}'$ might differ in their decisions. Since $\bar{p}_{\mathcal{M}}$ is a prefix of $\bar{p}$, it follows that $p_{\mathcal{M}} \leq p$ and

$$p - p_{\mathcal{M}} = \sum_{i=\log(5f(n))}^{\infty} \frac{p_i}{2^{i+1}} \leq \frac{1}{2^{\log(5f(n))}} = \frac{1}{5f(n)}.$$

In addition, the bits c_t and c'_t are generated by $\mathcal{M}$ and $\mathcal{M}'$ based on the sequences $\bar{b}$ and $\bar{b}\bar{b}'$ satisfying $\bar{b} <_{lex} \bar{b}\bar{b}'$. Hence,

$$\Pr(c_t \neq c'_t) = \Pr(\bar{p}_{\mathcal{M}} <_{lex} \bar{b}\bar{b}' <_{lex} \bar{p}) = p - p_{\mathcal{M}} \leq \frac{1}{5f(n)}.$$

By a union bound argument, the probability that the sequences $\bar{c} = c_0 c_1 \cdots c_{f(n)-1}$ and $\bar{c}' = c'_0 c'_1 \cdots c'_{f(n)-1}$ generated by $\mathcal{M}$ and $\mathcal{M}'$ differ satisfy

$$\Pr(\bar{c} \neq \bar{c}') \leq \frac{1}{5f(n)} f(n) = \frac{1}{5} \quad \text{and thus} \quad \Pr(\bar{c} = \bar{c}') \geq 1 - \frac{1}{5}.$$

Since $\mathcal{M}'$ classifies w correctly with probability at least $\frac{2}{3}$, it follows that $\mathcal{M}$ classifies w correctly with probability at least $(1 - \frac{1}{5})\frac{2}{3} = \frac{8}{15} > \frac{1}{2}$. This probability can be increased above $\frac{2}{3}$ by repeating Procedure 3 a constant number of time and taking the majority of the decisions as output [41]. Consequently, the devices $\mathcal{M}$, $\mathcal{M}'$ and $\mathcal{N}$ all decide the same language L, and therefore, $L \in \mathbf{PTMA}\left[\alpha(\bar{p}, \log(5f)), O(f^3)\right]$.

(ii) Let $L \in \mathbf{PTMA}[\alpha(\bar{p}, \log(f)), f]$. Then, there exists a PTM/log(A) $\mathcal{M}$ with logarithmic advice $\alpha(\bar{p}, \log(f))$ deciding L in time f. For simplicity purposes, let the advice of $\mathcal{M}$ be denoted by $\bar{p} = p_0 \cdots p_{\log(f(n))-1}$ ($\bar{p}$ is not anymore the binary expansion of p from now on). Now, consider Procedure 4 below. The first for loop computes an estimation $\bar{p}'$ of the advice $\bar{p}$ defined by

$$\bar{p}' = p'_0 \cdots p'_{\log(f(n))-1} = \delta_2^{-1}(p')[0 : \log(f(n)) - 1]$$

where

$$p' = \frac{1}{k(n)} \sum_{i=0}^{k(n)-1} b_i \quad \text{and} \quad k(n) = \ulcorner 10p(1-p)f^2(n) \urcorner$$

and the b_i are drawn according to a Bernouilli distribution of parameter p. The second for loop computes a sequence of random choices

$$\bar{c} = c_0 \cdots c_{f(n)-1}$$

using von Neumann's trick to simulate a fair coin with a biased one [41]. The third loop simulates the behavior of the PTM/log(A) $\mathcal{M}$ using the alternative advice $\bar{p}'$ and the sequence of random choices $\bar{c}$. This procedure can clearly be simulated by some SNN[p] $\mathcal{N}$ in time $O(k + 2f) = O(f^2)$, where the random samples of the bits b_i's are given by the stochastic cell and the remaining recursive instructions are simulated by a rational-weighted sub-network.

It remains to show that $\mathcal{M}$ and $\mathcal{N}$ decide the same language L. For this purpose, we estimate the error probability of $\mathcal{N}$ in deciding language L. First, we show that $\bar{p}'$ is a good approximation of the advice $\bar{p}$ of $\mathcal{M}$. Note that, since $\bar{p}$ and $\bar{p}'$ have length $\log(f(n))$, one has that $\bar{p}' \neq \bar{p}$ iff $|p' - p| > \frac{1}{2^{\log(f(n))}} = \frac{1}{f(n)}$. Note also that, by definition, $p' = \frac{\#1}{k(n)}$, where $\#1 \sim \mathcal{B}(k(n), p)$ is a binomial random variable of parameters $k(n)$ and p with $\mathrm{E}(\#1) = k(n)p$ and $\mathrm{Var}(\#1) = k(n)p(1-p)$. It follows that

Procedure 4

Input: input $w \in \Sigma^n$
1 **for** $i = 0, \dots, k(n) := \ulcorner 10p(1-p)f^2(n) \urcorner$ **do**
2 Draw a random bit b_i with probability p
3 Compute the estimation of the advice of $\mathcal{M}$ $\bar{p}' = p'_0 \cdots p'_{\log(f(n))-1} = \delta_2^{-1}(\frac{1}{k(n)} \sum_{i=0}^{k(n)-1} b_i)[0 : \log(f(n)) - 1]$
4 **for** $t = 0, \dots, f(n) - 1$ **do**
5 $c_t = 0$;
6 **for** $i = 0, \dots, \ulcorner \frac{-4-\log(f(n))}{\log(p^2+(1-p)^2)} \urcorner$ **do**
7 Draw 2 random bits b and b' with probability p
8 **if** $bb' = 01$ **then** $c_t = 0$; break
9 **if** $bb' = 10$ **then** $c_t = 1$; break
10 **for** $t = 0, \dots, f(n) - 1$ **do**
11 Simulate the PTM/log(A) $\mathcal{M}$ using the advice $\bar{p}' = p'_0 \cdots p'_{\log(f(n))-1}$ and the sequence of random choices $\bar{c} = c_0 \cdots c_{f(n)-1}$
12 **return** Output of $\mathcal{M}$ over w at time step $f(n)$

$$\begin{aligned}\Pr\left(\bar{p}' \neq \bar{p}\right) &= \Pr\left(|p' - p| > \frac{1}{f(n)}\right) \\ &= \Pr\left(|k(n)p' - k(n)p| > \frac{k(n)}{f(n)}\right) \\ &= \Pr\left(|\#1 - \mathrm{E}(\#1)| > \frac{k(n)}{f(n)}\right).\end{aligned}$$

The Chebyshev's inequality ensures that

$$\Pr\left(\bar{p}' \neq \bar{p}\right) \leq \frac{\mathrm{Var}(\#1)f^2(n)}{k^2(n)} = \frac{p(1-p)f^2(n)}{k(n)} < \frac{1}{10}$$

since $k(n) > 10p(1-p)f^2(n)$. Therefore, $\Pr\left(\bar{p}' = \bar{p}\right) \geq \frac{9}{10}$.

We now estimate the source of error coming from the simulation of a fair coin by a biased one in Procedure 4 (loop of Line 6). Note that, at each step i, if the two bits bb' are different (01 or 10), then c_t is drawn with fair probability $\frac{1}{2}$, like in the case of the machine $\mathcal{M}$. Hence, the sampling process of $\mathcal{N}$ and $\mathcal{M}$ differ in probability precisely when all of the $K = \frac{-4-\log(f(n))}{\log(p^2+(1-p)^2)}$ draws produce identical bits bb' (00 or 11). The probability that the two bits bb' are identical at step i is $p^2 + (1-p)^2$, and hence, the probability that the K independent draws all produce identical bits bb' satisfy

$$\left(p^2 + (1-p)^2\right)^K \leq 2^{-4-\log(f(n))} = \frac{1}{16f(n)}.$$

by using the fact that $x^{1/\log(x)} \geq 2$ and $-4 - \log(f(n)) < 0$. By a union bound argument, the probability that some c_t in the sequence $c_0 \cdots c_{f(n)-1}$ is not drawn with a fair probability $\frac{1}{2}$ is bounded by $\frac{1}{16}$. Equivalently, the probability that all random randoms bits c_t of the sequence $c_0 \cdots cs_{f(n)-1}$ are drawn with fair probability $\frac{1}{2}$ is at least $\frac{15}{16}$.

To safely estimate the error probability of $\mathcal{N}$, we assume that $\mathcal{N}$ always makes errors when its behavior differs from that of $\mathcal{M}$, i.e., when either $\bar{p}' \neq \bar{p}$ or some c_t is not drawn with fair probability. The complementary events that $\bar{p}' = \bar{p}$ and all c_t's being drawn with fair probability are independent and of probabilities at least $\frac{9}{10}$ and at least $\frac{15}{16}$, respectively. Hence, $\mathcal{M}$ and $\mathcal{N}$ agree on input w with probability at least $\frac{9}{10} \cdot \frac{15}{16} > \frac{4}{5}$. Consequently, the probability of $\mathcal{N}$ deciding correctly whether $w \in L$ or not is bounded by $\frac{4}{5} \cdot \frac{2}{3} > \frac{1}{2}$. As before, this probability can be made larger than $\frac{2}{3}$ by repeating Procedure 4 a constant number of times and taking the majority of the decisions as output [41]. This shows that $L(\mathcal{N}) = L(\mathcal{M}) = L$, and therefore, $L \in \mathbf{SNN}\left[p, O(f^2)\right]$. □

The class of languages decided in polynomial time by stochastic networks using real probabilities and Turing machines using related advices are the same. In this case, however, the length of the advice is logarithmic instead of polynomial.

Corollary 13. *Let $p \in \Delta$ be a real probability and $P \subseteq \Delta$ be a set of real probabilities.*

(i) $\mathbf{SNN}[p, \mathrm{poly}] = \mathbf{PTMA}[\alpha(\bar{p}, \log), \mathrm{poly}]$.
(ii) $\mathbf{SNN}[P, \mathrm{poly}] = \mathbf{PTMA}[\alpha(\bar{P}, \log), \mathrm{poly}]$.

Proof. The proof is similar to that of Corollary 6. □

6. Hierarchies

In this section, we provide a refined characterization of the super-Turing computational power of analog, evolving, and stochastic neural networks, based on the Kolmogorov complexity of their real weights, evolving weights, and real probabilities, respectively. More specifically, we demonstrate the existence of infinite hierarchies of classes of analog and evolving neural networks positioned between **P** and **P/poly**. Additionally, we establish the existence of an infinite hierarchy of classes of stochastic neural networks between **BPP** and **BPP/log***. Beyond proving the existence and providing examples of such hierarchies, we present a generic method for constructing them, based on classes of functions of increasing complexity.

To this end, we define the *Kolmogorov complexity* of a real number as outlined in a related work [26]. Let $\mathcal{M}_U$ be a universal Turing machine, $f, g : \mathbb{N} \to \mathbb{N}$ be two functions, and $\alpha \in \Sigma^\omega$ be some infinite word. We say that $\alpha \in \bar{K}_g^f$ if there exists $\beta \in \Sigma^\omega$ such that, for all but finitely many n, the machine $\mathcal{M}_U$ with inputs $\beta[0:m-1]$ and n will output $\alpha[0:n-1]$ in time $g(n)$, for all $m \geq f(n)$. In other words, $\alpha \in \bar{K}_g^f$ if its n first bits can be recovered from the $f(n)$ first bits of some β in time $g(n)$. The notion becomes relevant when $f(n) \leq n$, in which case $\alpha \in \bar{K}_g^f$ means that every n-long prefix of α can be compressed into and recovered from a smaller $f(n)$-long prefix of β. Given two classes of functions $\mathcal{F}$ and $\mathcal{G}$, we define

$$\bar{K}_{\mathcal{G}}^{\mathcal{F}} = \bigcup_{f \in \mathcal{F}} \bigcup_{g \in \mathcal{G}} \bar{K}_g^f .$$

Finally, for any real number $r \in \Delta$ with associated binary expansion $\bar{r} = \delta_4^{-1}(r) \in \Sigma^\omega$, we say that $r \in K_g^f$ iff $\bar{r} \in \bar{K}_g^f$ and

$$K_{\mathcal{G}}^{\mathcal{F}} = \bigcup_{f \in \mathcal{F}} \bigcup_{g \in \mathcal{G}} K_g^f .$$

Let $\mathcal{F} \subseteq \mathbb{N}^{\mathbb{N}}$ be a class of functions. We say $\mathcal{F}$ is a class of *reasonable advice bounds* if the following conditions hold:

- Sub-linearity: for all $f \in \mathcal{F}$, then $f(n) \leq n$ for all $n \in \mathbb{N}$.
- Dominance by a polynomially computable function: for all $f \in \mathcal{F}$, there exists $g \in \mathcal{F}$ such that $f \leq g$ and g is computable in polynomial time.
- Closure by polynomial composition on the right: For all $f \in \mathcal{F}$ and for all $p \in \text{poly}$, there exist $g \in \mathcal{F}$ such that $f \circ p \leq g$.

For instance, log is a class of reasonable advice bounds. All properties in this definition are necessary for our separation theorems. The first and second conditions are essential for defining Kolmogorov reals associated with advice of bounded size. The third condition arises from the fact that RNNs can access polynomial number of bits from their weights during polynomial time computation. Note that our definition is slightly weaker than that of Balcázar et al., who further assume that the class should be closed under $O(\cdot)$ [26].

The following theorem relates non-uniform complexity classes based on polynomial time of computation **P** and reasonable advice bounds $\mathcal{F}$, to classes of analog and evolving networks using weights within $K_{\text{poly}}^{\mathcal{F}}$ and $\bar{K}_{\text{poly}}^{\mathcal{F}}$, respectively.

Theorem 14. *Let $\mathcal{F}$ be a class of reasonable advice bounds, and let $K_{\text{poly}}^{\mathcal{F}} \subseteq \Delta$ and $\bar{K}_{\text{poly}}^{\mathcal{F}} \subseteq \Sigma^\omega$ be the sets of Kolmogorov reals associated with $\mathcal{F}$. Then*

$$\mathbf{P}/\mathcal{F}^* = \mathbf{ANN}\left[K_{\text{poly}}^{\mathcal{F}}, \text{poly}\right] = \mathbf{ENN}\left[\bar{K}_{\text{poly}}^{\mathcal{F}}, \text{poly}\right].$$

Proof. We prove the first equality. By definition, $\mathbf{P}/\mathcal{F}^*$ is the class of languages decided in polynomial time by some TM/A using any possible prefix advice of length $f \in \mathcal{F}$, namely,

$$\mathbf{P}/\mathcal{F}^* = \mathbf{TMA}\left[\alpha(\Sigma^\omega, \mathcal{F}), \text{poly}\right].$$

In addition, Corollary 6 ensures that

$$\mathbf{ANN}\left[K_{\text{poly}}^{\mathcal{F}}, \text{poly}\right] = \mathbf{TMA}\left[\alpha\left(\bar{K}_{\text{poly}}^{\mathcal{F}}, \text{poly}\right), \text{poly}\right].$$

Hence, we need to show that

$$\mathbf{TMA}\left[\alpha(\Sigma^\omega, \mathcal{F}), \text{poly}\right] = \mathbf{TMA}\left[\alpha\left(\bar{K}_{\text{poly}}^{\mathcal{F}}, \text{poly}\right), \text{poly}\right]. \tag{5}$$

Equation (5) can be understood as follows: in polynomial time of computation, the TM/As using small advices (of size $\mathcal{F}$) are equivalent to those using larger but compressible advices (of size poly and inside $\bar{K}_{\text{poly}}^{\mathcal{F}}$).

For the sake of simplicity, we suppose that the polynomial time of computation of the TM/As is clear from the context by introducing the following abbreviations:

$$\mathbf{TMA}\left[\alpha(\Sigma^\omega, \mathcal{F}), \text{poly}\right] := \mathbf{TMA}\left[\alpha(\Sigma^\omega, \mathcal{F})\right]$$
$$\mathbf{TMA}\left[\alpha\left(\bar{K}_{\text{poly}}^{\mathcal{F}}, \text{poly}\right), \text{poly}\right] := \mathbf{TMA}\left[\alpha\left(\bar{K}_{\text{poly}}^{\mathcal{F}}, \text{poly}\right)\right].$$

We show the backward inclusion of Eq. (5). Let

$$L \in \mathbf{TMA}\left[\alpha\left(\bar{K}^{\mathcal{F}}_{\mathrm{poly}}, \mathrm{poly}\right)\right].$$

Then, there exists a TM/A $\mathcal{M}$ using advice $\alpha(\bar{r}, p_1)$, where $\bar{r} \in \bar{K}^{\mathcal{F}}_{\mathrm{poly}}$ and $p_1 \in \mathrm{poly}$, deciding L in time $p_2 \in \mathrm{poly}$. Since $\bar{r} \in \bar{K}^{\mathcal{F}}_{\mathrm{poly}}$, there exist $\beta \in \Sigma^\omega$, $f \in \mathcal{F}$ and $p_3 \in \mathrm{poly}$ such that the $p_1(n)$ bits of $\bar{r}$ can be computed from the $f(p_1(n))$ bits of β in time $p_3(p_1(n))$. Hence, the TM/A $\mathcal{M}$ with advice $\alpha(\bar{r}, p_1)$ can be simulated by the TM/A $\mathcal{M}'$ with advice $\alpha(\beta, f \circ p_1)$ working as follows: on every input $w \in \Sigma^n$, $\mathcal{M}'$ first queries its advice string $\beta_0\beta_1 \cdots \beta_{f(p_1(n))-1}$, then reconstructs the advice $r_0 r_1 \ldots r_{p_1(n)-1}$ in time $p_3(p_1(n))$, and finally simulates the behavior of $\mathcal{M}$ over input w in real time. Clearly, $L(\mathcal{M}') = L(\mathcal{M}) = L$. In addition, $p_3 \circ p_1 \in \mathrm{poly}$, and since $\mathcal{F}$ is a class of reasonable advice bounds, there is $g \in \mathcal{F}$ such that $f \circ p_1 \leq g$. Therefore,

$$L \in \mathbf{TMA}\left[\alpha(\beta, g)\right] \subseteq \mathbf{TMA}\left[\alpha\left(\Sigma^\omega, \mathcal{F}\right)\right].$$

We now prove the forward inclusion of Eq. (5). Let

$$L \in \mathbf{TMA}\left[\alpha\left(\Sigma^\omega, \mathcal{F}\right)\right].$$

Then, there exists a TM/A $\mathcal{M}$ with advice $\alpha(\bar{r}, f)$, with $\bar{r} \in \Sigma^\omega$ and $f \in \mathcal{F}$, deciding L in time $p_1 \in \mathrm{poly}$. Since $\mathcal{F}$ is a class of reasonable advice bounds, there exists $g \in \mathcal{F}$ such that $f \leq g$ and g is computable in polynomial time. We now define $\bar{s} \in \bar{K}^{\mathcal{F}}_{\mathrm{poly}}$ using $\bar{r}$ and g as follows: for each $i \geq 0$, let $\bar{r}_i$ be the sub-word of $\bar{r}$ defined by

$$\bar{r}_i = \begin{cases} r_0 r_1 \cdots r_{g(0)-1} & \text{if } i = 0 \\ r_{g(i-1)} r_{g(i-1)+1} \cdots r_{g(i)-1} & \text{if } i > 0 \end{cases}$$

and let

$$\bar{s} = \bar{r}_0 0 \bar{r}_1 0 \bar{r}_2 0 \cdots.$$

Given the first $g(n)$ bits of $\bar{r}$, we can construct the first $g(n) + n \geq n$ bits of $\bar{s}$ by computing $g(i)$ and the corresponding block $\bar{r}_i$ (which may be empty) for all $i \leq n$, and then interleaving these with 0's. This process can be done in polynomial time, since g is computable is polynomial time. Therefore, $\bar{s} \in \bar{K}^{\mathcal{F}}_{\mathrm{poly}}$.

Let $q(n) = 2n$. Since $\mathcal{F}$ is a class of reasonable advice bounds, $g(n) \leq n$, and thus $q(n) = 2n \geq g(n) + n$. Now, consider the TM/A $\mathcal{M}'$ with advice $\alpha(\bar{s}, q)$ working as follows. On every input $w \in \Sigma^n$, the machine $\mathcal{M}'$ first queries its advice $\alpha(\bar{s}, q)(n) = s_0 s_1 \cdots s_{q(n)-1}$. Then, $\mathcal{M}'$ reconstructs the string $r_0 r_1 \cdots r_{g(n)-1}$ by computing $g(i)$ and then removing n 0's from $\alpha(\bar{s}, q)(n)$ at positions $g(i) + i$, for all $i \leq n$. This is done in polynomial time, since g is computable in polynomial time. Finally, $\mathcal{M}'$ simulates $\mathcal{M}$ with advice $r_0 r_1 \cdots r_{g(n)-1}$ in real time. Clearly, $L(\mathcal{M}') = L(\mathcal{M}) = L$. Therefore,

$$L \in \mathbf{TMA}\left[\alpha(\bar{s}, q)\right] \subseteq \mathbf{TMA}\left[\alpha\left(\bar{K}^{\mathcal{F}}_{\mathrm{poly}}, \mathrm{poly}\right)\right].$$

The property that have just been established together with Corollary 10 proves the second equality. □

We now prove the analogous of Theorem 14 for the case of probabilistic complexity classes and machines. In this case, however, the class of advice bounds does not anymore correspond exactly to the Kolmogorov space bounds of the real probabilities. Instead, a logarithmic correcting factor needs to be introduced. Given some class of functions $\mathcal{F}$, we define $\mathcal{F} \circ \log = \{f \circ \log \mid f \in \mathcal{F}\}$.

Theorem 15. *Let $\mathcal{F}$ be a class of reasonable advice bounds, then*

$$\mathbf{BPP}/(\mathcal{F} \circ \log)^* = \mathbf{SNN}\left[K^{\mathcal{F}}_{\mathrm{poly}}, \mathrm{poly}\right].$$

Proof. By definition, $\mathbf{BPP}/(\mathcal{F} \circ \log)^*$ is the class of languages decided in polynomial time by PTM/A using prefix advices of length $f \in \mathcal{F} \circ \log$:

$$\mathbf{BPP}/(\mathcal{F} \circ \log)^* = \mathbf{PTMA}\left[\alpha\left(\Sigma^\omega, \mathcal{F} \circ \log\right), \mathrm{poly}\right].$$

Moreover, Corollary 13 ensures that

$$\mathbf{SNN}\left[K^{\mathcal{F}}_{\mathrm{poly}}, \mathrm{poly}\right] = \mathbf{PTMA}\left[\alpha\left(\bar{K}^{\mathcal{F}}_{\mathrm{poly}}, \log\right), \mathrm{poly}\right].$$

Hence, we need to prove the following equality:

$$\mathbf{PTMA}\left[\alpha\left(\bar{K}^{\mathcal{F}}_{\mathrm{poly}}, \log\right), \mathrm{poly}\right] = \mathbf{PTMA}\left[\alpha\left(\Sigma^\omega, \mathcal{F} \circ \log\right), \mathrm{poly}\right]. \tag{6}$$

We first prove the forward inclusion of Eq. (6). Let

$$L \in \mathbf{PTMA}\left[\alpha\left(\bar{K}^{\mathcal{F}}_{\mathrm{poly}}, \log\right), \mathrm{poly}\right].$$

Then, there exists a PTM/A $\mathcal{M}$ using advice $\alpha(\bar{r}, c\log)$, where $\bar{r} \in \bar{K}^{\mathcal{F}}_{\text{poly}}$ and $c > 0$, that decides L in time $p_1 \in \text{poly}$. Since $\bar{r} \in \bar{K}^{\mathcal{F}}_{\text{poly}}$, there exist $\beta \in \Sigma^\omega$ and $f \in \mathcal{F}$ such that $\bar{r}[0 : n-1]$ can be computed from $\beta[0 : f(n)-1]$ in time $p_2(n) \in \text{poly}$, for all $n \geq 0$. Consider the PTM/A $\mathcal{M}'$ with advice $\alpha(\beta, f \circ c\log)$ working as follows. First, $\mathcal{M}'$ queries its advice $\beta[0 : f(c\log(n))-1]$, then it computes $\bar{r}[0 : c\log(n)-1]$ from this advice in time $p_2(\log(n))$, and finally it simulates $\mathcal{M}$ with advice $\bar{r}[0 : c\log(n)-1]$ in real time. Consequently, $\mathcal{M}'$ decides the same language L as $\mathcal{M}$, and works in time $O(p_1 + p_2 \circ \log) \in \text{poly}$. Therefore,

$$L \in \mathbf{PTMA}\left[\alpha\left(\Sigma^\omega, \mathcal{F} \circ \log\right), \text{poly}\right].$$

We now prove the backward inclusion of Eq. (6). Let

$$L \in \mathbf{PTMA}\left[\alpha\left(\Sigma^\omega, \mathcal{F} \circ \log\right), \text{poly}\right].$$

Then, there exists a PTM/A $\mathcal{M}$ using advice $\alpha(\bar{r}, f \circ c\log)$, where $\bar{r} \in \Sigma^\omega$, $f \in \mathcal{F}$ and $c > 0$, that decides L in time $p_1 \in \text{poly}$. Using the same argument as in the proof of Theorem 14, there exist $\bar{s} \in \bar{K}^{\mathcal{F}}_{\text{poly}}$ and $g \in \mathcal{F}$ such that $f \leq g$ and the smaller word $\bar{r}[0 : g(n)-1]$ can be retrieved from the larger one $\bar{s}[0 : 2n-1]$ in time $p_2(n) \in \text{poly}$, for all $n \geq 0$. Now, consider the PTM/A $\mathcal{M}'$ using advice $\alpha(\bar{s}, 2c\log)$ and working as follows. First, $\mathcal{M}'$ queries its advice $\bar{s}[0 : 2c\log(n)-1]$, then it reconstructs $\bar{r}[0 : g(c\log(n))-1]$ from this advice in time $O(p_2(\log(n)))$, and finally, it simulates $\mathcal{M}$ with advice $\bar{r}[0 : g(c\log(n))-1]$ in real time. Since $\bar{r}[0 : g(c\log(n))-1]$ extends $\bar{r}[0 : f(c\log(n))-1]$, $\mathcal{M}'$ and $\mathcal{M}$ decide the same language L. In addition, $\mathcal{M}'$ works in time $O(p_1 + p_2 \circ \log) \in \text{poly}$. Therefore,

$$\mathbf{PTMA}\left[\alpha\left(\bar{K}^{\mathcal{F}}_{\text{poly}}, \log\right), \text{poly}\right]. \quad \square$$

We now prove the separation of non-uniform complexity classes of the form $\mathcal{C}/f$. To this end, we assume that each class of languages $\mathcal{C}$ is defined based on a set of machines that decide these languages. For simplicity, we naturally identify $\mathcal{C}$ with its associated class of machines. For instance, **P** and **BPP** are identified with the set of Turing machines and probabilistic Turing machines working in polynomial time, respectively. In this context, we demonstrate that as soon as the advice is increased by a single bit, the capabilities of the corresponding machines are also increased. To achieve this result, two weak conditions are required. First, $\mathcal{C}$ must contain machines capable of reading their full inputs (of length n) and advices (of length $f(n)$), since otherwise, any additional advice bit would not lead to any change. Thus, $\mathcal{C}$ must at least include machines that operate in $O(n + f(n))$ time. Second, the advice length $f(n)$ should be smaller than 2^n, otherwise, the advice could encode any possible language, and the corresponding machine would have full computational power. The following result is first proven for machines with general (i.e., non-prefix) advices, before being stated for the special case of machines with prefix advices.

Theorem 16. *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two increasing functions such that $f(n) < g(n) \leq 2^n$, for all $n \in \mathbb{N}$. Let $\mathcal{C}$ be a set of machines containing the Turing machines working in time $O(n + f(n))$. Then $\mathcal{C}/f \subsetneq \mathcal{C}/g$.*

Proof. Note that any Turing machine M with advice α of size f can be simulated by some Turing machine M' with an advice α' of size $g > f$. Indeed, take $\alpha'(n) = 1^{g(n)-f(n)-1}0\alpha(n)$. Then, for any input $w \in \Sigma^n$, the machine M' queries its advice $\alpha'(n)$, erases all 1's up to and including the first encountered 0, and subsequently simulates M with advice $\alpha(n)$. Clearly, M and M' decide the same language.

To prove the strictness of the inclusion, we proceed by diagonalization. Recall that the set of (probabilistic) Turing machines is countable. Let $M_0, M_1, M_2, \ldots$ be an enumeration of the machines in $\mathcal{C}$. For any $M_k \in \mathcal{C}$ and any advice $\alpha : \mathbb{N} \to \Sigma^*$ of size f, let M_k/α be the associated (probabilistic) machine with advice, and let $L(M_k/\alpha)$ be its associated language. The language $L(M_k/\alpha)$ can be written as the union of its sub-languages of words of length n, i.e.

$$L(M_k/\alpha) = \bigcup_{n\in\mathbb{N}} L(M_k/\alpha)^n.$$

For each $k, n \in \mathbb{N}$, consider the set of sub-languages of words of length n decided by M_k/α, for all possible advices α of size f, i.e.:

$$\mathcal{L}^n_k = \left\{L(M_k/\alpha)^n : \alpha \text{ is an advice of size } f\right\}.$$

Since there are at most $2^{f(n)}$ advice strings of length $f(n)$, it follows that $|\mathcal{L}^n_k| \leq 2^{f(n)}$, for all $k \in \mathbb{N}$, and in particular, that $|\mathcal{L}^n_n| \leq 2^{f(n)}$. By working on the diagonal $D = \left(\mathcal{L}^n_n\right)_{n\in\mathbb{N}}$ of the sequence $\left(\mathcal{L}^n_k\right)_{k,n\in\mathbb{N}}$ (illustrated in Table 1), we will build a language $A = \bigcup_{n\in\mathbb{N}} A^n$ that cannot be decided by any Turing machine in $\mathcal{C}$ with advice of size f, but can be decided by some Turing machine in $\mathcal{C}$ with advice of size g. It follows that $A \in (\mathcal{C}/g) \setminus (\mathcal{C}/f)$, and therefore, $\mathcal{C}/f \subsetneq \mathcal{C}/g$.

Let $n \in \mathbb{N}$. For each $i < 2^n$, let $b(i) \in \Sigma^n$ be the binary representation of i over n bits. For any subset $\mathcal{L} \subseteq \mathcal{L}^n_n$, let

$$\mathcal{L}\left(b(i)\right) = \{L \in \mathcal{L} : b(i) \in L\} \quad \text{and} \quad \bar{\mathcal{L}}\left(b(i)\right) = \{L \in \mathcal{L} : b(i) \notin L\}.$$

Consider the sequence $(\mathcal{L}^n_{n,0}, \ldots, \mathcal{L}^n_{n,f(n)+1})$ of decreasing subsets of $\mathcal{L}^n_n$ and the sequence $(A^n_0, \ldots, A^n_{f(n)+1})$ of sub-languages of words of length n defined by induction for every $0 \leq i \leq f(n)$ as follows

$$\mathcal{L}^n_{n,0} = \mathcal{L}^n_n \text{ and } \mathcal{L}^n_{n,i+1} = \begin{cases} \mathcal{L}^n_{n,i}\left(b(i)\right) & \text{if } |\mathcal{L}^n_{n,i}\left(b(i)\right)| < |\bar{\mathcal{L}}^n_{n,i}\left(b(i)\right)| \\ \bar{\mathcal{L}}^n_{n,i}\left(b(i)\right) & \text{otherwise} \end{cases}$$

Table 1
Illustration of the sequence $(\mathcal{L}_k^n)_{k,n\in\mathbb{N}}$. Each set $\mathcal{L}_k^n$ satisfies $|\mathcal{L}_k^n| \leq 2^{f(n)}$. The diagonal $D = (\mathcal{L}_n^n)_{n\in\mathbb{N}}$ is highlighted.

	0	1	2	⋯	n	⋯
0	$\mathcal{L}_0^0$	$\mathcal{L}_0^1$	$\mathcal{L}_0^2$	⋯	$\mathcal{L}_0^n$	⋯
1	$\mathcal{L}_1^0$	$\mathcal{L}_1^1$	$\mathcal{L}_1^2$	⋯	$\mathcal{L}_1^n$	⋯
2	$\mathcal{L}_2^0$	$\mathcal{L}_2^1$	$\mathcal{L}_2^2$	⋯	$\mathcal{L}_2^n$	⋯
⋮	⋮	⋮	⋮	⋱	⋮	⋯
n	$\mathcal{L}_n^0$	$\mathcal{L}_n^1$	$\mathcal{L}_n^2$	⋯	$\mathcal{L}_n^n$	⋯
⋮	⋮	⋮	⋮	⋯	⋮	⋱

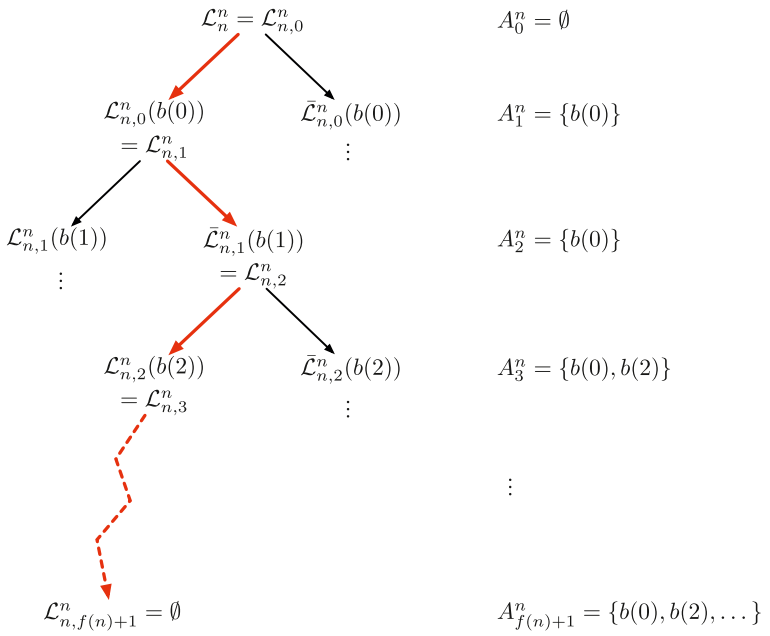

Fig. 3. Inductive construction of the sequences $(\mathcal{L}_{n,i}^n)_{i=0}^{f(n)+1}$ and $(A_i^n)_{i=0}^{f(n)+1}$.

$$A_0^n = \emptyset \text{ and } \mathcal{A}_{i+1}^n = \begin{cases} \mathcal{A}_i^n \cup \{b(i)\} & \text{if } |\mathcal{L}_{n,i}^n(b(i))| < |\bar{\mathcal{L}}_{n,i}^n(b(i))| \\ \mathcal{A}_i^n & \text{otherwise.} \end{cases}$$

This construction is illustrated in Fig. 3.

Note that the n-bit representation $b(i)$ of i is well-defined, since $0 \leq i \leq f(n) < 2^n$. In addition, the construction ensures that $\mathcal{L}_{n,i+1}^n \subseteq \mathcal{L}_{n,i}^n$ and $|\mathcal{L}_{n,i+1}^n| \leq \frac{1}{2}|\mathcal{L}_{n,i}^n|$, for all $0 \leq i \leq f(n)$, and since $|\mathcal{L}_{n,0}^n| = |\mathcal{L}_n^n| \leq 2^{f(n)}$, it follows that $|\mathcal{L}_{n,f(n)+1}^n| = 0$, meaning that $\mathcal{L}_{n,f(n)+1}^n = \emptyset$. Now, towards, a contradiction, suppose that $A_{f(n)+1}^n \in \mathcal{L}_n^n = \mathcal{L}_{n,0}^n$. The construction and the previous properties imply that

$$A_{f(n)+1}^n \in \bigcap_{0\leq i\leq f(n)} \mathcal{L}_{n,i}^n = \mathcal{L}_{n,f(n)+1}^n = \emptyset$$

which is a contradiction. Therefore, $A_{f(n)+1}^n \notin \mathcal{L}_n^n$, for all $n \in \mathbb{N}$.

Now, consider the language

$$A = \bigcup_{n\in\mathbb{N}} A_{f(n)+1}^n .$$

By construction, $A_{f(n)+1}^n$ is the set of words of length n of A, i.e., $A_{f(n)+1}^n = A^n$, for all $n \in \mathbb{N}$. We now show that A cannot be decided by any machine in C with advice of size f. Towards a contradiction, suppose that $A \in C/f$. Then, there exist $M_k \in C$ and $\alpha : \mathbb{N} \to \Sigma^*$ of size f such that $L(M_k/\alpha) = A$, and thus $L(M_k/\alpha)^k = A^k$. On the one hand, the definition of $\mathcal{L}_k^k$ ensures that $L(M_k/\alpha)^k \in \mathcal{L}_k^k$. On the other hand, $L(M_k/\alpha)^k = A^k = A_{f(k)+1}^k \notin \mathcal{L}_k^k$, which is a contradiction. Therefore, $A \notin C/f$.

We now show that $A \in C/g$. Consider the advice function $\alpha : \mathbb{N} \to \Sigma^*$ of size $g = f + 1$ given by $\alpha(n) = \alpha_0^n \alpha_1^n \cdots \alpha_{f(n)}^n$, where

$$\alpha_i^n = \begin{cases} 1 & \text{if } b(i) \in A^n \\ 0 & \text{otherwise,} \end{cases}$$

for all $0 \leq i \leq f(n)$. Note that the advice string $\alpha(n)$ encodes the sub-language A^n, for all $n \in \mathbb{N}$, since the latter is a subset of $\{b(i) : 0 \leq i \leq f(n)\}$, by definition. Consider the Turing machine with advice M/α which, on every input $w = w_0 w_1 \cdots w_{n-1}$ of length n, moves its advice head up to the i-th bit α_i^n of $\alpha(n)$, where $i = b^{-1}(w)$, if this bit exists (note that $i < 2^n$ and $|\alpha(n)| = f(n) + 1$), and accepts w if and only if $\alpha_i^n = 1$. Note that these instructions can be computed in time $O(f(n) + n)$. In particular, moving the advice head up to the i-th bit of $\alpha(n)$ does not require to compute $i = b^{-1}(w)$ explicitly, but can be achieved by moving simultaneously the input head from the end of the input to the beginning and the advice head from left to right, in a suitable way. It follows that

$$w \in L(M/\alpha)^n \text{ iff } \alpha^n_{b^{-1}(w)} = 1 \text{ iff } b(b^{-1}(w)) \in A^n \text{ iff } w \in A^n .$$

Hence, $L(M/\alpha)^n = A^n$, for all $n \in \mathbb{N}$, and thus

$$L(M/\alpha) = \bigcup_{n\in\mathbb{N}} L(M/\alpha)^n = \bigcup_{n\in\mathbb{N}} A^n = A .$$

Therefore, $A \in C/g$. The argument can be generalized in a straightforward way to any advice size g such that $f(n) + 1 \leq g(n) \leq 2^n$.

Finally, the two properties $A \notin C/f$ and $A \in C/g$ imply that $C/f \subsetneq C/g$. □

We now prove the analogous of Theorem 16 for the case of machines with prefix advice. In this case, however, a stronger condition on the advice lengths f and g is required: $f \in o(g)$ instead of $f \leq g$.

Theorem 17. *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two increasing functions such that $f \in o(g)$ and $\lim_{n\to\infty} g(n) = +\infty$. Let C be a set of machines containing the Turing machines working in time $O(n + f(n))$. Then $C/f^* \subsetneq C/g^*$.*

Proof. The proof is similar to that of Theorem 16, except that the separation language A is constructed based on a sequence of integers $(n_i)_{i\in\mathbb{N}}$. Consider the sequence $(n_i)_{i\in\mathbb{N}}$ defined for all $i \geq 0$ as follows

$$n_0 = \min\{n \in \mathbb{N} : 2(f(n) + 1) \leq g(n)\}$$
$$n_{i+1} = \min\left\{n \in \mathbb{N} : 2\sum_{j=0}^{i}(f(n_j) + 1) + 2(f(n) + 1) \leq g(n)\right\}.$$

We show by induction that the sequence $(n_i)_{i\in\mathbb{N}}$ is well-defined. Since $f \in o(g)$ and $\lim_{n\to\infty} g(n) = +\infty$, the following limits hold

$$\lim_{n\to\infty} \frac{2(f(n) + 1)}{g(n)} = 2 \lim_{n\to\infty} \frac{f(n)}{g(n)} + \lim_{n\to\infty} \frac{2}{g(n)} = 0$$
$$\lim_{n\to\infty} \frac{2\sum_{j=0}^{i}(f(n_j) + 1) + 2(f(n) + 1)}{g(n)} = 2 \lim_{n\to\infty} \frac{f(n)}{g(n)} + \lim_{n\to\infty} \frac{C + 2}{g(n)} = 0$$

where $C = 2\sum_{j=0}^{i}(f(n_j) + 1)$, and thus n_0 and n_{i+1} are well-defined.

For each $k, i \in \mathbb{N}$, consider the set of sub-languages of words of length n_i decided by the machine $M_k \in C$ using any possible advice α of size f, i.e.,

$$\mathcal{L}_k^{n_i} = \left\{L(M_k/\alpha)^{n_i} : \alpha \text{ is an advice of size } f\right\}.$$

Consider the diagonal $\mathcal{D} = \left(\mathcal{L}_i^{n_i}\right)_{i\in\mathbb{N}}$ of the set $\left(\mathcal{L}_k^{n_i}\right)_{k,n_i\in\mathbb{N}}$. Since there are at most $2^{f(n_i)}$ advice strings of length $f(n_i)$, it follows that $|\mathcal{L}_i^{n_i}| \leq 2^{f(n_i)}$. Using a similar construction as in the proof of Theorem 16, we can define by induction a sub-language $A^{n_i}_{f(n_i)+1} \subseteq \Sigma^{n_i}$ such that $A^{n_i}_{f(n_i)+1} \notin \mathcal{L}_i^{n_i}$. Then, consider the language

$$A := \bigcup_{i\in\mathbb{N}} A^{n_i} = \bigcup_{i\in\mathbb{N}} A^{n_i}_{f(n_i)+1} .$$

Once again, a similar argument as in the proof of Theorem 16 ensures that $A \notin C/f$. Since $C/f^* \subseteq C/f$, it follows that $A \notin C/f^*$.

We now show that $A \in C/g^*$. Recall that, by construction, $A^{n_i} \subseteq \{b(j) : 0 \leq j \leq f(n_i)\}$. Consider the word homomorphism $h : \Sigma^* \to \Sigma^*$ induced by the mapping $0 \mapsto 00$ and $1 \mapsto 11$, and define the symbol $\# = 01$. For each $i \in \mathbb{N}$, consider the encoding $\beta_0^{n_i}\beta_1^{n_i} \cdots \beta^{n_i}_{f(n_i)}$ of A^{n_i} given by

$$\beta_j^{n_i} = \begin{cases} 1 & \text{if } b(j) \in A^{n_i}, \\ 0 & \text{otherwise,} \end{cases}$$

for all $0 \leq j \leq f(n_i)$, and let $\beta(n_i) = h(\beta_0^{n_i} \beta_1^{n_i} \cdots \beta_{f(n_i)}^{n_i})$. Note that $|\beta(n_i)| = 2(f(n_i)+1)$. Now, consider the advice function $\alpha : \mathbb{N} \to \Sigma^*$ given by the concatenation of the encodings of the successive A^{n_i} separated by symbols #. Formally,

$$\alpha(n) = \begin{cases} \beta(n_0)\#\beta(n_1)\#\cdots\#\beta(n_i) & \text{if } n = n_i \text{ for some } i \geq 0 \\ \beta(n_0)\#\beta(n_1)\#\cdots\#\beta(n_i)\# & \text{if } n_i < n < n_{i+1}. \end{cases}$$

Note that $|\alpha(n)| = 2\sum_{j=0}^{i}(f(n_j)+1) \leq g(n_i) \leq g(n)$ for $n_i \leq n < n_{i+1}$ and α satisfies the prefix property: $m \leq n$ implies $|\alpha(m)| \leq |\alpha(n)|$. If necessary, the advice strings can be extended by dummy symbols 10 in order to achieve the equality $|\alpha(n)| = g(n)$, for all $n \geq 0$ (assuming without loss of generality that $g(n)$ is even). Now, consider the machine with advice M/α which, on every input w of length n, first reads its advice string $\alpha(n)$. If the last symbol of $\alpha(n)$ is #, then it means that $|w| \neq n_i$ for all $i \geq 0$, and the machine rejects w. Otherwise, the input is of length n_i for some $i \geq 0$. Hence, the machine moves its advice head back up to the last # symbol, and then moves one step to the right. At this point, the advice head points at the beginning of the advice substring $\beta(n_i)$. Then, the machine decodes $\beta_0^{n_i} \beta_1^{n_i} \cdots \beta_{f(n_i)}^{n_i}$ from $\beta(n_i)$ by removing one out of two bits. Next, as in the proof of Theorem 16, the machine moves its advice head up to the j-th bit $\beta_j^{n_i}$, where $j = b^{-1}(w)$, if this bit exists (note that $j < 2^{n_i}$ and $|\beta(n_i)| = f(n_i)+1$), and accepts w if and only if $\beta_j^{n_i} = 1$. These instructions can be computed in time $O(2\sum_{j=0}^{i}(f(n_j)+1)+n_i)$. It follows that $w \in L(M/\alpha)^{n_i}$ iff $\beta_j^{n_i} = 1$ iff $b(j) \in A^{n_i}$ iff $b\left(b^{-1}(w)\right) \in A^{n_i}$ iff $w \in A^{n_i}$. Thus $L(M/\alpha)^{n_i} = A^{n_i}$, for all $i \in \mathbb{N}$, and hence

$$L(M/\alpha) = \bigcup_{i \in \mathbb{N}} L(M/\alpha)^{n_i} = \bigcup_{i \in \mathbb{N}} A^{n_i} = A$$

Therefore, $A \in C/g^*$.

Finally, the two properties $A \notin C/f^*$ and $A \in C/g^*$ imply that $C/f^* \subsetneq C/g^*$. □

The separability between classes of analog, evolving, and stochastic recurrent neural networks using real weights, evolving weights, and probabilities of different Kolmogorov complexities, respectively, can now be obtained.

Corollary 18. *Let $\mathcal{F}$ and $\mathcal{G}$ be two classes of reasonable advice bounds. Suppose that there exists $g \in \mathcal{G}$ such that $f \in o(g)$ and $\lim_{n\to\infty} g(n) = +\infty$, for all $f \in \mathcal{F}$. Then*

(i) $\mathbf{ANN}\left[K^{\mathcal{F}}_{\text{poly}}, \text{poly}\right] \subsetneq \mathbf{ANN}\left[K^{\mathcal{G}}_{\text{poly}}, \text{poly}\right]$

(ii) $\mathbf{ENN}\left[\bar{K}^{\mathcal{F}}_{\text{poly}}, \text{poly}\right] \subsetneq \mathbf{ENN}\left[\bar{K}^{\mathcal{G}}_{\text{poly}}, \text{poly}\right]$

(iii) $\mathbf{SNN}\left[K^{\mathcal{F}}_{\text{poly}}, \text{poly}\right] \subsetneq \mathbf{SNN}\left[K^{\mathcal{G}}_{\text{poly}}, \text{poly}\right]$

Proof. (i) and (ii): Theorem 14 shows that

$$\mathbf{P}/\mathcal{F}^* = \mathbf{ANN}\left[K^{\mathcal{F}}_{\text{poly}}, \text{poly}\right] = \mathbf{ENN}\left[\bar{K}^{\mathcal{F}}_{\text{poly}}, \text{poly}\right] \quad \text{and}$$

$$\mathbf{P}/\mathcal{G}^* = \mathbf{ANN}\left[K^{\mathcal{G}}_{\text{poly}}, \text{poly}\right] = \mathbf{ENN}\left[\bar{K}^{\mathcal{G}}_{\text{poly}}, \text{poly}\right].$$

In addition, Theorem 17 ensures that

$$\mathbf{P}/\mathcal{F}^* \subsetneq \mathbf{P}/g^* \subseteq \mathbf{P}/\mathcal{G}^*$$

The strict inclusions of Points (i) and (ii) follow directly.

(iii): Theorem 15 states that

$$\mathbf{BPP}/(\mathcal{F} \circ \log)^* = \mathbf{SNN}\left[K^{\mathcal{F}}_{\text{poly}}, \text{poly}\right]$$

$$\mathbf{BPP}/(\mathcal{G} \circ \log)^* = \mathbf{SNN}\left[K^{\mathcal{G}}_{\text{poly}}, \text{poly}\right].$$

In addition, note that if $f \in \mathcal{F}$ and $g \in \mathcal{G}$ satisfy the hypotheses of Theorem 17, then so do $f \circ l \in \mathcal{F} \circ \log$ and $g \circ l \in \mathcal{G} \circ \log$, for all $l \in \log$. Hence, Theorem 17 ensures that

$$\mathbf{BPP}/(\mathcal{F} \circ \log)^* \subsetneq \mathbf{BPP}/g \circ \log)^* \subseteq \mathbf{BPP}/(\mathcal{G} \circ \log)^*.$$

The strict inclusion of Point (iii) ensues. □

Finally, Corollary 18 provides a way to construct infinite hierarchies of classes of analog, evolving and stochastic neural networks based on the Kolmogorov complexity of their underlying weights and probabilities, respectively. The hierarchies of analog and evolving networks are located between **P** and $\mathbf{P}/\mathbf{poly}$, while those of stochastic networks lie between **BPP** and $\mathbf{BPP}/\mathbf{log}^*$.

For instance, define $\mathcal{F}_i = O\left(\log(n)^i\right)$, for all $i \in \mathbb{N}$. Each $\mathcal{F}_i$ satisfies the three conditions for being a class of reasonable advice bounds (note that the sub-linearity $\log(n)^i$ is satisfied for n sufficiently large). By Corollary 18, the sequence of classes $(\mathcal{F}_i)_{i\in\mathbb{N}}$ induces the following infinite strict hierarchies of classes of neural networks:

$$\begin{array}{ccccccc}
\mathbf{ANN}\left[K^{\mathcal{F}_0}_{\text{poly}}, \text{poly}\right] & \subsetneq & \mathbf{ANN}\left[K^{\mathcal{F}_1}_{\text{poly}}, \text{poly}\right] & \subsetneq & \mathbf{ANN}\left[K^{\mathcal{F}_2}_{\text{poly}}, \text{poly}\right] & \subsetneq & \cdots \\
\mathbf{ENN}\left[\bar{K}^{\mathcal{F}_0}_{\text{poly}}, \text{poly}\right] & \subsetneq & \mathbf{ENN}\left[\bar{K}^{\mathcal{F}_1}_{\text{poly}}, \text{poly}\right] & \subsetneq & \mathbf{ENN}\left[\bar{K}^{\mathcal{F}_2}_{\text{poly}}, \text{poly}\right] & \subsetneq & \cdots \\
\mathbf{SNN}\left[K^{\mathcal{F}_0}_{\text{poly}}, \text{poly}\right] & \subsetneq & \mathbf{SNN}\left[K^{\mathcal{F}_1}_{\text{poly}}, \text{poly}\right] & \subsetneq & \mathbf{SNN}\left[K^{\mathcal{F}_2}_{\text{poly}}, \text{poly}\right] & \subsetneq & \cdots
\end{array}$$

We provide another example of hierarchy for stochastic networks only. In this case, it can be noticed that the third condition for being a class of reasonable advice bounds can be relaxed: we only need that $\mathcal{F} \circ \log$ is a class of reasonable advice bounds and that the functions of $\mathcal{F}$ are bounded by n. Accordingly, consider some infinite sequence of rational numbers $(q_i)_{i\in\mathbb{N}}$ such that $0 < q_i < 1$ and $q_i < q_{i+1}$, for all $i \in \mathbb{N}$, and define $\mathcal{F}_i = O(n^{q_i})$, for all $i \in \mathbb{N}$. Each $\mathcal{F}_i$ satisfies the required conditions. By Corollary 18, the sequence of classes $(\mathcal{F}_i)_{i\in\mathbb{N}}$ induces the following infinite strict hierarchy of classes of neural networks:

$$\mathbf{SNN}\left[K^{\mathcal{F}_0}_{\text{poly}}, \text{poly}\right] \subsetneq \mathbf{SNN}\left[K^{\mathcal{F}_1}_{\text{poly}}, \text{poly}\right] \subsetneq \mathbf{SNN}\left[K^{\mathcal{F}_2}_{\text{poly}}, \text{poly}\right] \subsetneq \cdots$$

7. Application

The Kolmogorov complexity measures the compressibility of real numbers. Accordingly, our study finds its full relevance in the context of theoretical analog computation [42]. In practice, real-valued weights cannot be manipulated exactly, making approximation and compression techniques essential. These compression techniques can be viewed as practical counterparts to the theoretical concept of Kolmogorov complexity. Among them, *quantization* has become a critical technique in recent years, actively explored in research for its ability to enhance computational efficiency during training while preserving, and sometimes even improving, network performance [43,44]. In the case of echo state networks (ESNs), where only the output weights are trained, we show that the predictive capabilities of the networks are strongly impacted by the degree of quantization applied to their weights. The code is freely available on GitHub.

We consider *leaky echo state networks* (ESNs) whose dynamical equations are obtained by slight modifications of Equations (1) and (2):

$$\begin{cases} \mathbf{x}^{t+1} = (1-\alpha)\,\mathbf{x}^t + \alpha\,\sigma\left(\mathbf{W_{in}}\mathbf{u}^{t+1} + \mathbf{W_{res}}\mathbf{x}^t + \mathbf{b}\right) \\ \mathbf{y}^{t+1} = \mathbf{W_{out}}\mathbf{x}^{t+1} \end{cases}$$

where $\sigma = \tanh$, $\mathbf{x}^t \in \mathbb{R}^N$ is the reservoir state at time t, $\mathbf{u}^{t+1} \in \mathbb{R}^K$ is the input at time $t+1$, $\mathbf{W_{in}} \in \mathbb{R}^{N\times K}$, $\mathbf{W_{res}} \in \mathbb{R}^{N\times N}$ and $\mathbf{W_{out}} \in \mathbb{R}^{N\times N}$ are the input, reservoir and output weight matrices, respectively, $\mathbf{b} \in \mathbb{R}^N$ is the bias, and $\alpha \in (0, 1]$ is the leaking rate, controlling the speed of the state update.

The hyperparameters of ESNs play a crucial role on their memory capacity, stability and predictive performance [38]. Since we are primarily interested in studying the effect of weight quantization, we limit ourselves to fixed hyperparameters. The reservoir weights $\mathbf{W}_{\text{res}}$ are rescaled to have a spectral radius (largest absolute eigenvalue) of 0.99 and a sparsity of 90%. The input weights $\mathbf{W}_{\text{in}}$ have an input scaling (magnitude of the entries) of 1.0 and a sparsity of 80%. The leaking rate is set to $\alpha = 0.3$.

To study the effect of weight compression, we implement a *quantization* process on the input, reservoir, and output weights of the ESN. Given a specified number of precision bits $b \in \{2, 4, 8, 16, 32, 64\}$, we round the weights to the closest values within a finite set of 2^b quantization levels. These levels are evenly distributed between the minimum and maximum values of the weights in question. As a result, each quantized weight corresponds to the discrete level determined by the original weight's value and is thus representable by b precision bits.

Given a time series $\bar{\mathbf{u}} = \{\mathbf{u}^t\}_{t\geq 0}$, we train a leaky ESN on a τ-step look-ahead prediction task. At each time step t, the network is provided with the current input $\mathbf{u}^t$ and its task is to predict the future value $\mathbf{y}^t := \mathbf{u}^{t+\tau}$. This formulation encourages the network to capture long-term dependencies in the data.

Specifically, we begin by feeding the initial 100 values of $\bar{\mathbf{u}}$ to the network to wash out the influence of the initial state on the reservoir. After this warm-up phase, we train the network using the subsequent 2000 values of $\bar{\mathbf{u}}$, collecting the corresponding reservoir states and targets into the matrices $\mathbf{X}$ and $\mathbf{Y}_{\text{target}}$, respectively (row-wise concatenation). The output weights are then learned using a Ridge regression between $\mathbf{X}$ and $\mathbf{Y}_{\text{target}}$, whose closed-form solution is given by

$$\mathbf{W}_{\text{out}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{Y}_{\text{target}}\,.$$

Finally, we evaluate the trained network by running it on the next $m = 500$ time steps, collecting the corresponding predictions and targets into the matrices $\mathbf{Y}_{\text{pred}}$ and $\mathbf{Y}_{\text{target}}$, respectively, and computing the normalized root mean squared error (NRMSE) given by

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left\|\mathbf{Y}_{\text{pred}}(i) - \mathbf{Y}_{\text{target}}(i)\right\|^2}}{\max_{i,j}\left\|\mathbf{Y}_{\text{target}}(i) - \mathbf{Y}_{\text{target}}(j)\right\|}\,.$$

Table 2
NRMSE of ESNs evaluated on the Mackey-Glass (MG), Lorenz (LO), and Hénon Map (HM) time series for a 10-step look-ahead forecasting task. The top part (ALL) presents results where all weights are quantized to b precision bits, with $b \in \{2, 4, 8, 16, 32, 64\}$. The bottom part (RES) shows results where only the reservoir weights are quantized.

ALL	2	4	8	16	32	64
MG	5.504×10^{3}	8.408×10^{3}	4.245×10^{-1}	1.511×10^{-3}	$\mathbf{6.039 \times 10^{-4}}$	$\mathbf{6.039 \times 10^{-4}}$
LO	8.944×10^{3}	3.377×10^{2}	4.701×10^{0}	2.748×10^{-2}	$\mathbf{1.419 \times 10^{-2}}$	$\mathbf{1.419 \times 10^{-2}}$
HM	1.358×10^{3}	1.654×10^{3}	6.435×10^{0}	2.110×10^{-1}	$\mathbf{2.073 \times 10^{-1}}$	$\mathbf{2.073 \times 10^{-1}}$
RES	2	4	8	16	32	64
MG	2.114×10^{-1}	3.231×10^{-2}	$\mathbf{6.026 \times 10^{-4}}$	6.039×10^{-4}	6.039×10^{-4}	6.039×10^{-4}
LO	7.605×10^{-2}	1.761×10^{-2}	$\mathbf{1.410 \times 10^{-2}}$	1.419×10^{-2}	1.419×10^{-2}	1.419×10^{-2}
HM	2.057×10^{-1}	2.064×10^{-1}	$\mathbf{2.073 \times 10^{-1}}$	$\mathbf{2.073 \times 10^{-1}}$	$\mathbf{2.073 \times 10^{-1}}$	$\mathbf{2.073 \times 10^{-1}}$

In the teacher forcing paradigm, during training, the actual target $\mathbf{y}_{\text{target}}^{t+\tau-1}$ is provided as part of the input to help stabilize state evolution and improve learning. During testing, however, the predicted value $\mathbf{y}_{\text{pred}}^{t+\tau-1}$ is fed back into the input instead. To reduce the impact of randomness in weight initialization, each experiment is repeated 20 times with different random seeds, and the results are averaged over these trials.

We evaluate the performance of echo state networks (ESNs) on three benchmark time series that exhibit rich, chaotic dynamics. The Mackey-Glass time series is generated by the following delay differential equation:

$$\frac{dx(t)}{dt} = \beta \frac{x(t-\tau)}{1 + x(t-\tau)^n} - \gamma x(t),$$

where τ is the delay, and β, γ, and n are system parameters. For the parameters values $\tau = 17$, $\beta = 0.2$, $\gamma = 0.1$, $n = 10$, the system exhibits chaotic behavior. The Lorenz system is a set of three coupled ordinary differential equations given by

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases}$$

with the canonical parameter values $\sigma = 10$, $\rho = 28$, and $\beta = \frac{8}{3}$. The Lorenz system displays chaotic behavior characterized by sensitive dependence on initial conditions and the formation of a strange attractor. The Hénon map is a discrete-time dynamical system defined by

$$\begin{cases} x_{n+1} = 1 - a x_n^2 + y_n \\ y_{n+1} = b x_n \end{cases}$$

with parameters set to $a = 1.4$ and $b = 0.3$. Despite its simplicity, the Hénon map produces a fractal attractor and exhibits robust chaotic behavior.

The results of our experiments are presented in Fig. 4 and Table 2 (top). As expected, the predictive performance of the networks improves significantly as the number of quantization bits increases (from 2 to 64). The forecast horizon also affects the networks performance, to a lesser extent. The logarithmic scale of the heatmaps highlights the substantial gains achieved by increasing weight precision, or equivalently, the drastic losses induced by greater weight quantization. Overall, these results demonstrate that in echo state networks – where the reservoir primarily functions as a dynamical encoder and predictions rely heavily on the output weights $\mathbf{W}_{\text{out}}$ – the level of quantization plays a crucial role in the network's capabilities. This contrasts with deep feedforward architectures, such as Transformers and convolutional neural networks (CNNs), where weight approximation and compression techniques have been successfully applied [43,44]. We conjecture that this difference arises from the ability of deep architectures to distribute approximation errors across successive layers, thereby mitigating their impact on overall performance.

In contrast, the predictive capabilities of the networks are significantly less affected when only the reservoir weights are quantized, while the output weights keep full precision. Fig. 4 and Table 2 (bottom) show that, across all configurations, reservoirs quantized to 4, 8, 16, 32, and 64 precision bits yield nearly identical results. This robustness stems from the reservoir's ability to encode input sequences as superimposed dynamical patterns, functioning like a complex Morse code, thereby reducing the need for high precision. However, accurately decoding these reservoir dynamics requires highly precise output weights. These findings align with the liquid state machine (LSM) paradigm, which typically employs binary reservoirs composed of spiking neural networks [45].

8. Conclusion

In this work, we provide a refined characterization of the super-Turing computational power of analog, evolving, and stochastic recurrent neural networks through the lens of Kolmogorov complexity of their underlying real-valued weights, evolving rational weights, and real-valued probabilities, respectively. For the first two models, we establish infinite hierarchies of analog and evolving networks situated between **P** and **P/poly**. For the stochastic model, we construct an infinite hierarchy of network classes positioned

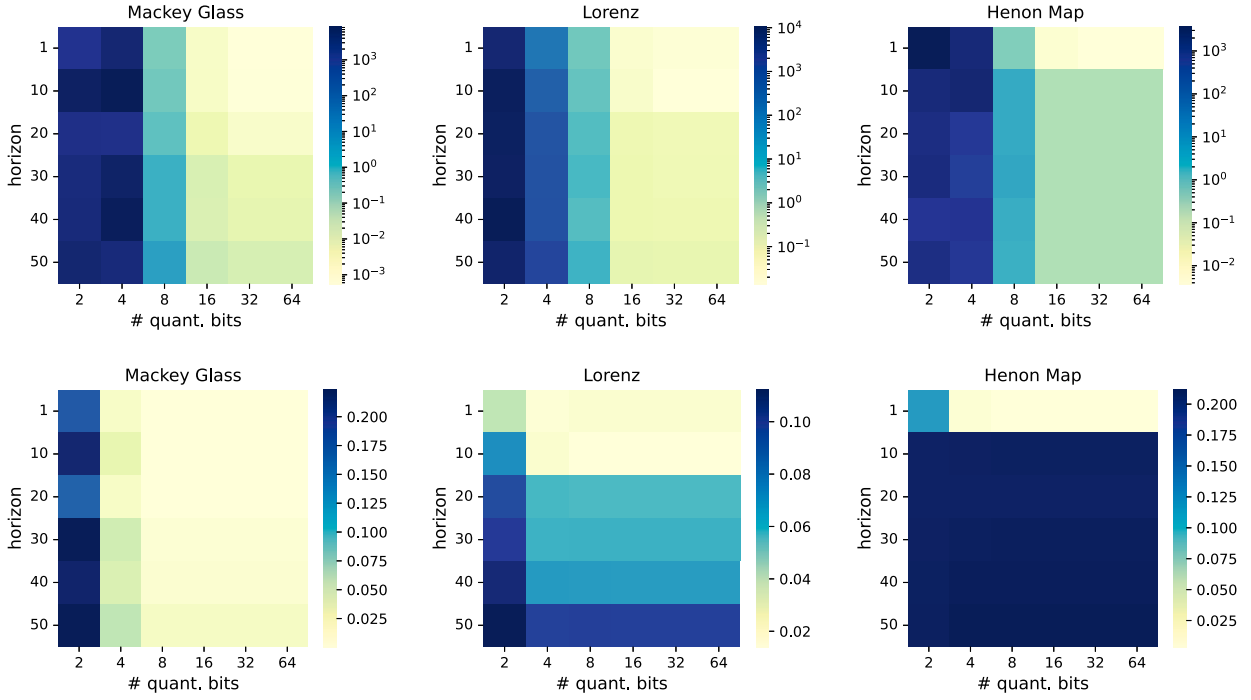

Fig. 4. Normalized root mean squared error (NRMSE) of ESNs evaluated on the Mackey-Glass, Lorenz, and Hénon Map time series. The ESNs are evaluated on look-ahead forecasting tasks with horizons $h \in \{1, 10, 20, 30, 40, 50\}$, and their weights are quantized to b precision bits, where $b \in \{2, 4, 8, 16, 32, 64\}$. The forecasting task becomes more challenging as the number of precision bits decreases and the horizon increases. **(Top)** All weights are quantized. **(Bottom)** Only the reservoir weight $\mathbf{W}_{\mathbf{res}}$ is quantized.

between **BPP** and **BPP/log***. Beyond proving the existence and providing concrete examples of these hierarchies, Corollary 18 offers a general method for constructing them based on function classes that satisfy reasonable advice bound conditions.

This work extends the study of Balcázar et al. [26] on the Kolmogorov-based hierarchization of analog neural networks. In particular, the proof of Theorem 14 is strongly inspired by their Theorem 6.2 [26]. However, our approach takes a different perspective by making the computational relationships between recurrent neural networks and Turing machines with advice more explicit. In this regard, Propositions 5, 9, and 12 characterize precisely how real weights, evolving weights, and real probabilities correspond to advice of varying lengths in the associated Turing machines. In contrast, Balcázar et al. frame these relationships indirectly through an alternative model – Turing machines with tally oracles. Another key distinction is that our separability results (Theorems 16 and 17) are derived using a diagonalization argument that applies to any non-uniform complexity class, making it of independent interest. Our approach does not rely on the existence of reals with high Kolmogorov complexity. Finally, our conditions for reasonable advice bounds are slightly weaker than those of Balcázar et al., broadening the scope of our results.

The computational equivalence between stochastic neural networks and Turing machines with advice builds on the results of Siegelmann [21]. Our Proposition 12 is inspired by their Lemma 6.3 [21]. However, while their result establishes the equivalence between two models of bounded-error Turing machines, our Proposition 12 makes the relationship between stochastic neural networks and Turing machines with logarithmic advice explicit. Additionally, our use of union bound arguments provides a technical simplification of their original approach.

The main message of our study is twofold: (1) the complexity of real or evolving weights is crucial to the computational power of analog and evolving neural networks, and (2) the complexity of the randomness source similarly influences the capabilities of stochastic neural networks. These theoretical insights contrast with the flourishing research on approximation techniques – such as precision scaling and quantization – that often yield significant efficiency gains [43,44]. In the framework of theoretical analog computation, less compressible weights or randomness sources encode more information, ultimately enhancing the computational power of the underlying neural networks.

Our results strongly rely on the Turing completeness of recurrent neural networks (RNNs) [14]. Accordingly, they are framed within the context of echo state networks (ESNs), which represent the simplest and most general form of vanilla RNNs. However, these results can be safely generalized to other recurrent architectures, provided they are Turing complete. For instance, any universal architecture incorporating LSTM, GRU, or ReLU units would fall within the scope of our analysis.[4]

In practice, real-valued weights or probabilities cannot be manipulated with exact precision. However, one could envision a system where a rational-weighted recurrent neural network (RNN) is coupled with a quantum bit generator, which inherently produces a

[4] Note, however, that the Turing completeness of RNNs composed *exclusively* of LSTM or GRU units has not been established.

Kolmogorov incompressible real number, bit by bit. According to our theory, the class of RNNs utilizing these bits could solve strictly more problems than those based on rational weights. Furthermore, the quantum bit generator could be harnessed to generate real numbers of varying Kolmogorov complexities by producing subsequent bits based on the incompressible properties of the preceding ones. This would allow for the construction of RNNs belonging to more refined complexity classes, based on these real numbers.

For future work, hierarchies based on alternative notions beyond Kolmogorov complexity could be investigated. Additionally, while the universality of echo state networks has been extensively studied from a functional analysis perspective [39], their computational universality remains an open question from a probabilistic standpoint. Specifically, given an echo state network $\mathcal{N}$ that satisfies appropriate reservoir conditions, and a computable function f, it remains to be determined whether output weights can be found such that $\mathcal{N}$ approximates f with high probability.

Finally, the proposed study aims to bridge existing gaps and offer a unified perspective on the enhanced capabilities of analog, evolving, and stochastic recurrent neural networks. The question of whether the super-Turing computational power of neural networks can be practically exploited is a complex issue that falls outside the scope of this paper, aligning instead with the philosophical discussions on hypercomputation [40]. Nonetheless, we believe this study contributes to advancing the field of analog computation [42].

CRediT authorship contribution statement

Jérémie Cabessa: Writing – review & editing, Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization. **Yann Strozecki:** Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The research was done with institutional supports RVO: 67985807 and partially supported by the Czech Science Foundation grants AppNeCo GA22-02067S and LEDNeCo GA25-15490S.

Data availability

No data was used for the research described in the article.

References

[1] P.S. Churchland, T.J. Sejnowski, The Computational Brain, The MIT Press, 2016.
[2] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133.
[3] S.C. Kleene, Representation of events in nerve nets and finite automata, in: C. Shannon, J. McCarthy (Eds.), Automata Studies, Princeton University Press, Princeton, NJ, 1956, pp. 3–41.
[4] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1967.
[5] B.G. Horne, D.R. Hush, Bounds on the complexity of recurrent neural network implementations of finite state machines, Neural Netw. 9 (2) (1996) 243–252.
[6] C.W. Omlin, C.L. Giles, Constructing deterministic finite-state automata in recurrent neural networks, J. ACM 43 (6) (1996) 937–972.
[7] A.M. Turing, Intelligent machinery, Technical report, National Physical Laboratory, Teddington, UK, 1948.
[8] J.B. Pollack, On connectionist models of natural language processing, Ph.D. thesis, Computing Reseach Laboratory, New Mexico State University, Las Cruces, NM, 1987.
[9] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (6) (1958) 386–408.
[10] D.O. Hebb, The Organization of Behavior: A Neuropsychological Theory, John Wiley & Sons Inc., 1949.
[11] M.L. Minsky, S. Papert, Perceptrons: An Introduction to Computational Geometry, MIT Press, Cambridge, MA, USA, 1969.
[12] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[13] J.v. Neumann, The Computer and the Brain, Yale University Press, New Haven, CT, USA, 1958.
[14] H.T. Siegelmann, E.D. Sontag, On the computational power of neural nets, J. Comput. Syst. Sci. 50 (1) (1995) 132–150.
[15] J. Kilian, H.T. Siegelmann, The dynamic universality of sigmoidal neural networks, Inf. Comput. 128 (1) (1996) 48–56.
[16] H.T. Siegelmann, Neural Networks and Analog Computation: Beyond the Turing Limit, Birkhauser Boston Inc., Cambridge, MA, USA, 1999.
[17] H.T. Siegelmann, E.D. Sontag, Analog computation via neural networks, Theor. Comput. Sci. 131 (2) (1994) 331–360.
[18] H.T. Siegelmann, Computation beyond the Turing limit, Science 268 (5210) (1995) 545–548.
[19] J. Cabessa, O. Finkel, Computational capabilities of analog and evolving neural networks over infinite input streams, J. Comput. Syst. Sci. 101 (2019) 86–99.
[20] J. Cabessa, H.T. Siegelmann, The super-Turing computational power of plastic recurrent neural networks, Int. J. Neural Syst. 24 (8) (2014).
[21] H.T. Siegelmann, Stochastic analog networks and computational complexity, J. Complex. 15 (4) (1999) 451–475.
[22] W. Maass, P. Orponen, On the effect of analog noise in discrete-time analog computations, Neural Comput. 10 (5) (1998) 1071–1095.
[23] W. Maass, 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.
[24] J. Šíma, Subrecursive neural networks, Neural Netw. 116 (2019) 208–223, https://doi.org/10.1016/j.neunet.2019.04.019.
[25] J. Šíma, Analog neuron hierarchy, Neural Netw. 128 (2020) 199–215, https://doi.org/10.1016/j.neunet.2020.05.006.
[26] J.L. Balcázar, R. Gavaldà, H.T. Siegelmann, Computational power of neural networks: a characterization in terms of Kolmogorov complexity, IEEE Trans. Inf. Theory 43 (4) (1997) 1175–1183.
[27] W. Maass, Computing with spiking neurons, in: W. Maass, C.M. Bishop (Eds.), Pulsed Neural Networks, MIT Press, Cambridge, MA, USA, 1999, pp. 55–85.

[28] W. Maass, Lower bounds for the computational power of networks of spiking neurons, Neural Comput. 8 (1) (1996) 1–40.
[29] W. Maass, On the computational power of noisy spiking neurons, in: D.S. Touretzky, M. Mozer, M.E. Hasselmo (Eds.), Advances in Neural Information Processing Systems 8, NIPS Conference, MIT Press, Denver, CO, November 27-30, 1995, 1995, pp. 211–217.
[30] J. Cabessa, A. Tchaptchet, Automata complete computation with Hodgkin-Huxley neural networks composed of synfire rings, Neural Netw. 126 (2020) 312–334.
[31] C.H. Papadimitriou, S.S. Vempala, D. Mitropolsky, M. Collins, W. Maass, Brain computation by assemblies of neurons, Proc. Natl. Acad. Sci. 117 (25) (2020) 14464–14472.
[32] G. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143.
[33] G. Păun, M.J. Pérez-Jiménez, A. Salomaa, Spiking neural P systems: an early survey, Int. J. Found. Comput. Sci. 18 (3) (2007) 435–455.
[34] M. Gheorghe, M. Stannett, Membrane system models for super-Turing paradigms, Nat. Comput. 11 (2) (2012) 253–259.
[35] K. Greff, R.K. Srivastava, J. Koutnìk, B.R. Steunebrink, J. Schmidhuber, LSTM: a search space odyssey, IEEE Trans. Neural Netw. Learn. Syst. 28 (10) (2017) 2222–2232.
[36] W. Merrill, G. Weiss, Y. Goldberg, R. Schwartz, N.A. Smith, E. Yahav, A formal hierarchy of RNN architectures, in: D. Jurafsky, J. Chai, N. Schluter, J.R. Tetreault (Eds.), Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020, Association for Computational Linguistics, 2020, pp. 443–459.
[37] H. Jaeger, H. Haas, Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication, Science 304 (5667) (2004) 78–80.
[38] M. Lukoševičius, H. Jaeger, Reservoir computing approaches to recurrent neural network training, Comput. Sci. Rev. 3 (3) (2009) 127–149.
[39] L. Grigoryeva, J. Ortega, Echo state networks are universal, Neural Netw. 108 (2018) 495–508, https://doi.org/10.1016/j.neunet.2018.08.025.
[40] B.J. Copeland, Hypercomputation: philosophical issues, Theor. Comput. Sci. 317 (1–3) (2004) 251–267.
[41] S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2006, https://theory.cs.princeton.edu/complexity/book.pdf.
[42] O. Bournez, A. Pouly, A Survey on Analog Models of Computation, Springer International Publishing, Cham, 2021, pp. 173–226.
[43] A. Gholami, S. Kim, Z. Dong, Z. Yao, M.W. Mahoney, K. Keutzer, A survey of quantization methods for efficient neural network inference, CoRR, arXiv:2103.13630 [abs], 2021, arXiv:2103.13630, https://arxiv.org/abs/2103.13630.
[44] R. Jin, J. Du, W. Huang, W. Liu, J. Luan, B. Wang, D. Xiong, A comprehensive evaluation of quantization strategies for large language models, in: L. Ku, A. Martins, V. Srikumar (Eds.), Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024, ACL, 2024, pp. 12186–12215.
[45] W. Maass, H. Markram, On the computational power of circuits of spiking neurons, J. Comput. Syst. Sci. 69 (4) (2004) 593–616.