Computational Power of Neural Networks: A Characterization in Terms of Kolmogorov Complexity

José L. Balcázar, Ricard Gavaldà, and Hava T. Siegelmann

***Abstract*— The computational power of recurrent neural networks is shown to depend ultimately on the complexity of the real constants (weights) of the network. The complexity, or information contents, of the weights is measured by a variant of resource-bounded Kolmogorov complexity, taking into account the time required for constructing the numbers. In particular, we reveal a full and proper hierarchy of nonuniform complexity classes associated with networks having weights of increasing Kolmogorov complexity.**

***Index Terms*— Kolmogorov complexity, neural networks, Turing machines.**

I. Introduction

WE FOCUS on the classical analog recurrent neural networks [6], [8], consisting of a *finite* number of simple processors, each of which computes a scalar—*real-valued*—function, or "activation," of an integrated input. The adjective "analog" refers to the continuity of the activation function. "Recurrent" refers to the existence of feedback loops in the interconnection graph, allowing the processing of arbitrarily long data with a fixed-size network. The formalization of the activation gives rise to a dynamical system.

These networks have attracted much attention lately as computational devices. One particular model of network has been suggested in [24], [27] to capture computation with "analog phase space." This contrasts with the well-known Turing machine model that essentially describes all digital computers. Work so far has distinguished essentially two types of such networks, depending on whether the weights of the networks are rational numbers or real numbers.

If the weights are rational numbers, then networks are equivalent to Turing machines [25], [26]; that is, they are no more powerful than standard digital computers. In the general case, weights are real numbers with infinite precision. Then the networks have been shown to be strictly more powerful than Turing machines [27]. For example, in exponential time they are able to compute every binary function. Even if their computation time is further constrained to polynomial time, they remain computationally strictly stronger than the Turing machine with the same time bounds: they compute the functions in the complexity class denoted by P/poly, which is known to contain some, but not all, recursive binary functions.

In this paper we focus on the reason of this phenomenon. It is clearly due to the fact that both Turing machines and nets with rational weights are finite objects, in the sense that they can be described with a finite amount of information. This is not true for real numbers, hence a neural network containing real numbers with infinite precision has access to a potentially infinite source of information, which may allow it to compute nonrecursive functions. Intuitively, as the real numbers used are richer and richer in information, more and more nonrecursive functions become computable. To formalize this statement, we need a quantitative measure of the information contained in real numbers.

Previous work in the field of information theory [18], [19], [22] has defined the complexity of a sequence as a "measure on the extent to which a given sequence resembles a random one" [14]. One particular line [4], [5], [13] leads to the notion now usually called Kolmogorov complexity of a string. The complexity of a finite sequence is the length of the (shortest) binary string that can be given as input to a universal algorithm to construct (output) the sequence. This can be generalized in several ways to infinite strings, and hence to real numbers. Here we take one of the variants of the notion called resource-bounded Kolmogorov complexity. This is obtained by constraining also the time used by the universal algorithm to construct the numbers (or equivalently, respond to their input strings), making the notion effective.

We prove that, in a sense we make precise, the predictability of processes like these described by neural networks depends essentially on the resource-bounded Kolmogorov complexity of the real numbers defining the process. We find that the equivalence between neural networks and either Turing machines (for rational weights) or classes such as P/poly (for real weights) are but two special cases. By slowly increasing the Kolmogorov complexity of the weights, an infinite proper hierarchy of computational classes is obtained. For polynomial time bounds, these classes are shown to coincide with so-called "advice classes," well studied in complexity theory, of which the classes P and P/poly are the two extreme cases.

Manuscript received December 20, 1994; revised October 20, 1996. The research of J. L. Balcázar and R. Gavaldà was supported in part by the ESPRIT Working Group NeuroCOLT (no. 8556) and by DGICYT, Project PB92–0709. The work of H. T. Siegelmann was supported in part by the Alon fellowship. A preliminary version of the results described here appears as a part of [2].

J. L. Balcázar and R. Gavaldà are with the Department of Software (LSI), Universitat Politècnica de Catalunya, Barcelona 08028, Spain.

H. T. Siegelmann is with the Faculty of Industrial Engineering and Management, Technion–Israel Institute of Technology, Haifa 32000, Israel.

Publisher Item Identifier S 0018-9448(97)03446-9.

A. Organization of the Paper

The paper is technical, and takes the reader step-by-step towards the hierarchy theorem that is stated and proved only in Section VII. Section II consists of preliminaries of complexity theory. In Section III, we describe our neural network model. In Section IV, we define our variant of Kolmogorov complexity. In Section V, we focus on Turing machines that consult oracles as defined in Section III. We prove there the main theorem of this paper: the equivalence between neural networks of various Kolmogorov weights and Turing machines that consult corresponding oracle sets. Section VI considers the connection between neural networks and Turing machines with advice words. As corollaries, we obtain the previous results of Siegelmann and Sontag as special cases. The hierarchy theorem is stated and proved in Section VII using the result and terminology of Section V. We close Section VII with a short summary.

II. Preliminaries: Complexity Theory

Other than the prefix-advice nonuniform complexity classes defined below, all the concepts from Complexity Theory mentioned throughout are standard; see [1, vol. I], for relevant properties and undefined notions.

For any alphabet Σ, Σ^*, and Σ^∞ denote the sets of all finite words and infinite sequences over Σ, respectively. We denote the union of finite and infinite strings as $\Sigma^\# = \Sigma^* \cup \Sigma^\infty$. Strings in Σ^* and $\Sigma^\#$ are ordered by length, and lexicographically within each length. We denote by $\omega_{1:k}$ the word consisting of the first k symbols of ω, and by ω_k the kth symbol of ω; this notation is used when ω is a finite or an infinite sequence. Symbol ϵ denotes the empty string. The length of a finite word ω is denoted by $|\omega|$, and we use the same notation $|A|$ for the cardinality of a finite set A; this should cause no confusion. For an infinite sequence α, we define $|\alpha| = \infty$.

Define $\Sigma^{\leq n}$ as the set of all words of length at most n and, for $A \subseteq \Sigma^*$, $A^{\leq n} = A \cap \Sigma^{\leq n}$; similarly, we have $\Sigma^{=n}$ and $A^{=n}$. Here we will use in particular the alphabets $\Sigma = \{0,1\}$ and $\Sigma = \{0\}$. A tally set is a set of words over the single-letter alphabet $\{0\}$.

If A is a set of words, $\chi_A \in \{0,1\}^\infty$ is the characteristic sequence of A, defined in the standard way: the ith bit of the sequence is 1 if and only if the ith word of Σ^* is in A. Similarly, $\chi_{A^{\leq n}}$ is the characteristic sequence of $A^{\leq n}$ relative to $\Sigma^{\leq n}$. In both cases, Σ is taken as the smallest alphabet containing all the symbols occurring in words of A, so that for a tally set T, χ_T denotes the characteristic sequence of T relative to $\{0\}^*$. Throughout this paper, $\log n$ means the function $\max(1, \lceil \log_2 n \rceil)$.

To encode several strings into one, we use a pairing function $\langle \cdot, \cdot \rangle$: $\Sigma^* \times \Sigma^* \to \Sigma^*$ that is easily computable and invertible. This function is extended to more than two arguments by composition.

We also say that a class $\mathcal{F}$ of functions is *closed under* $O(\cdot)$ if for every f, g, if $g \in O(f)$ and $f \in \mathcal{F}$, then $g \in \mathcal{F}$.

A *(formal) language* over alphabet Σ is a subset of Σ^*. In Complexity Theory, formal languages are classified into complexity classes according to the resources needed by machines of some kind to recognize them.

Complexity class P is the class of languages for which membership can be decided in time polynomial in the length of the input. Overloading the name, we also use P to denote the class of all functions $\Sigma^* \to \Sigma^*$ that are polynomial-time-computable in the same sense. The underlying computation model in both definitions is the standard Turing machine.

Relativized complexity classes are defined using oracle Turing machines. An oracle Turing machine has a specially designated oracle tape and three special states called QUERY, YES, and NO. When the machine enters the QUERY state, it switches in the next step to state YES or state NO depending on whether the string currently written on the oracle tape is or is not in an oracle set A, fixed for the computation. We say that the machine queries the string to oracle A, and that the computation is relative to A.

For a language A, $\mathcal{P}(A)$ is the class of languages that can be decided in polynomial time by oracle Turing machines querying oracle A. If B is in $\mathcal{P}(A)$, we also say that B (Turing-) reduces in polynomial time to A. For a class of languages $\mathcal{A}$, $\mathcal{P}(\mathcal{A})$ is the union of all $\mathcal{P}(A)$ for $A \in \mathcal{A}$.

Our notation for nonuniform complexity classes was introduced in [10]. Given a class of sets C and a class of bounding functions H, the nonuniform class C/H is formed by the sets A such that

$$\forall n\, \exists \omega^n (|\omega^n| \leq h(n)) \forall x (|x| \leq n) [x \in A \Longleftrightarrow \langle x, \omega^n \rangle \in B]$$

where $B \in C$ and $h \in H$. Some frequent H's are

- poly, the class of all functions bounded above by a polynomial;
- log, the class of all functions bounded above by $c \cdot \log n$, for a constant $c > 0$.

Observe that log and poly are closed under $O(\cdot)$.

String ω^n in the definition is called "advice for length n." Observe that ω^n must be useful for all strings of length up to n, not only those of length n. This is not relevant for polynomial advice bounds, but must be taken into account for sublinear bounds; thus here we are departing somewhat from the notation of [10], and use the classes termed "strong" in [11] and "full" in [3].

A restriction gives the classes Pref-C/H, defined in [3]: the definition is the same as that of C/H, with the additional condition that, for all $n \leq m$, ω^n is a prefix of ω^m. Note also that Pref-P/poly = P/poly, but that a similar equality might not hold for smaller function classes. It has been proved, however, that for most cases of interest (namely, H closed under $O(\cdot)$), Pref-P/H coincides with P/H according to our definition [7].

Note that if we allow for exponentially long advice, and allowing exponential time in the length of the input, the nonuniform class contains all binary languages, as the advice could be used to provide the binary response for all exponentially many inputs of length n. Polynomially long advice is not enough to recognize all binary functions. However, in polynomial time it can still recognize, for example, all unary languages and in particular nonrecursive ones, such as a unary encoding of the halting problem.

Our main interest in the advice classes will be the class P/poly that allows polynomial computation time and a polynomially long advice. We will use the following characterization of P/poly (see, e.g., [1]): A set is in P/poly if and only if it is in $\mathcal{P}(T)$ for some tally set T.

The nonuniform complexity class P/poly is a ubiquitous class that has been characterized in many other different ways: for instance, as polynomial-size threshold feedforward neural nets [21], and as Hopfield networks of polynomial size and polynomially high weights [20]. We will see in the next section how it can be defined by the analog recurrent neural network.

Still another interpretation of advice complexity classes is as follows. Consider some sort of process by which, in successive time instants $t = 0, 1, \cdots$, a binary measurement provides observations $v_0, v_1, \cdots$, where $v_t \in \{0, 1\}$. Assume that the process is such that a few (finitely many) real-valued parameters $\vec{\theta}$ allow one to compute feasibly the value of v_t from t and the parameters. Now, if we identify the process with the set $\{\mathrm{bin}\,(t) | v_t = 1\}$, and model "feasibility" by the more formal "in polynomial time," the processes that are predictable in this sense form exactly the complexity class P/poly. Now consider the case when only the first $f(n)$ bits of the real-valued parameters are known, or guaranteed to be correct. This corresponds to reducing the length of the advice from polynomial to roughly $f(n)$.

III. Preliminaries: The Neural Network Model

We focus on the model of recurrent network proposed by Siegelmann and Sontag [26], [27]. It consists of a *finite* number of simple processors. The *activation value* or *state* of each processor is updated at times $t = 1, 2, 3, \cdots$, according to a function of the activations (x_j) and inputs (u_j) at time $t-1$, and a set of real coefficients—also called *weights*—(a_{ij}, b_{ij}, c_i). More precisely, each processor's state is updated by an equation of the type

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

where N is the number of processors, M is the number of external input signals, and σ is a "sigmoid-like" function. This scalar function, or "activation" function, is meant to reflect the graded response of biological neurons to the net sum of excitatory and inhibitory inputs affecting them. We concentrate on a particular, very simple σ function, namely, the saturated-linear function

$$\sigma(x) := \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } 0 \le x \le 1 \\ 1, & \text{if } x > 1. \end{cases} \quad (2)$$

As part of the description, we assume that we have singled out a subset of the N processors, say $x_{i_1}, \cdots, x_{i_p}$; these are the p output processors that communicate the output of the network to the environment. Thus a network is specified by its weights and the set of output processors.

The structure of the network, including the number of neurons and the values of the interconnection weights, does not change in time, but rather remains constant. What changes in time are the activation values, or outputs of each processor. In this sense, our model is "uniform" in contrast with certain models used in the past (e.g., [9]), that allow the number of units to increase over time and often even allow the structure to change, depending on the length of inputs being presented.

Recently, foundational research on the computational power of the analog recurrent neural network model has been developed [2], [25]–[27]. The main results of this line of research are characterizations of classes of formal languages that networks can accept. By standard encoding methods, any other finite, fixed alphabet could be assumed, provided that it has at least two different symbols. The results can also be extended from languages to arbitrary functions from strings to strings.

The focus has been on networks for which input and output channels carry only discrete data. As opposed to the I/O, the computation inside the network in general involves continuous real values.

The computational power of a recurrent network (with a finite number of neurons) depends on the following classes of numbers utilized as weights:

- If the weights are integers, the neurons may assume binary activation values only. Thus the network accepts a regular language.
- If the weights are rational numbers, the network is equivalent in power to the Turing machine model [25]. Two different I/O conventions are suggested in [26]: One that uses input lines and output neurons, and another one where the discrete input and the output are encoded as the state of two fixed neurons. The results for each convention are formally stated as follows.

Theorem 3.1: Let ϕ: $\{0,1\}^* \to \{0,1\}^*$ be any partial recursive function. Then, there exists a network $\mathcal{N}$ with input lines that computes ϕ. Furthermore, if there is a Turing machine (with one input tape and several work tapes) that computes $\phi(\omega)$ in time $T(\omega)$, then some network $\mathcal{N}$ computes $\phi(\omega)$ in time $O(T(\omega) + |\omega|)$. ■

Theorem 3.2: Let M be a Turing machine computing ϕ: $\{0,1\}^* \to \{0,1\}^*$ in time T. Then there exists a network $\mathcal{N}$ without inputs such that the following properties hold: For each $\omega \in \{0,1\}^*$, assume $\mathcal{N}$ is started in the initial state

$$\xi(\omega) = (\delta[\omega], 0, \cdots, 0) \in \mathbb{Q}^N$$

where δ: $\{0,1\}^* \to [0,1]$ is defined by

$$\delta(\omega_1\omega_2\cdots\omega_n) = \sum_{i=1}^{n} (2\omega_i + 1)/4^i.$$

Then, if $\phi(\omega)$ is undefined, the second coordinate $\xi(\omega)_2^t$ of the state after t steps is identically equal to zero, for all t. If, instead, $\phi(\omega)$ is defined, then

$$\xi(\omega)_2^t = 0, \quad \text{for } t = 0, \cdots, T(w) - 1$$
$$\xi(\omega)_2^{T(w)} = 1$$

and

$$\xi(\omega)_1^{T(w)} = \delta[\phi(\omega)].$$

Furthermore, all weights in $\mathcal{N}$ are simple, nonperiodic, and bounded rationals. ■

In both theorems, the size of $\mathcal{N}$ is fixed, independent of the running time T. The proof of the first theorem relies on the second one and the observation that given an input string, the network can encode it in linear time as the initial state in the second theorem, and output the result as given in the second theorem through the output neurons.

- When weights are general real numbers, the network can compute nonrecursive functions, but is still sensitive to resource constraints. In [27] there is an exact characterization of the computational class associated with such nets: In exponential time, networks with real weights can decide any language, but in polynomial time they accept exactly the language class P/poly.

  The main observation that allowed for the result is as follows: Let $\mathcal{N}$ be a network. Then *q-truncation*($\mathcal{N}$) is a network similar to $\mathcal{N}$, but in which the weights have binary precision of q bits only and so are the activations values of the neurons. It is proven in [27] that up to computation of time q, only the first $O(q)$ bits in both weights and activation values influence the result. Hence, there is a constant c so that up to computation time q, *cq-truncation*($\mathcal{N}$) is equivalent to $\mathcal{N}$. This is formalized as follows:

  Lemma 3.3: Let $\mathcal{N}$ be a network computing in time $T(n)$. Then there exists a constant c such that, for every n, *$cT(n)$-truncation*($\mathcal{N}$) computes the same function as $\mathcal{N}$, on inputs of length n.

In this paper we refine these results as follows. (For this informal presentation think of $T(n)$ as a polynomial.) Lemma 3.3 says that the only information accessible to a network in time $T(n)$ is the one contained in the first $O(T(n))$ bits of its weights. Suppose now that the information is in fact much smaller than $T(n)$, i.e., the weights have low Kolmogorov complexity. Then the networks will not be able to compute all of P/poly any longer. We show that they are able to compute the restriction of P/poly when the amount of advice is reduced correspondingly. In this sense, "time-bounded Kolmogorov complexity of weights" and "amount of advice for Turing machines" are equivalent computational resources.

IV. Kolmogorov Complexity and Real Numbers

Kolmogorov complexity attempts to measure the quantity of information contained in an individual object. This approach is quite different from that of classical information theory, where information usually involved some probability distribution on a set of objects.

Intuitively, objects that contain very little information should admit very short descriptions in an appropriate description language. On the other hand, if an object is very complex it should contain almost no redundancy that helps in describing it concisely. As an extreme case, the most economical way to describe an object with no redundancy at all is providing the object itself. A crucial point in the theory is the choice of the description language. Since we are interested in descriptions that are algorithmically useful, we define a description for an object to be a program that builds the object, when fed into some universal Turing machine.

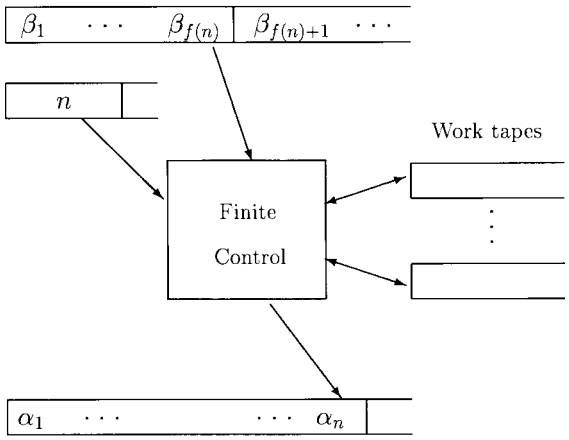

Fig. 1. Our universal Turing machine.

Here we only give the definitions that directly apply to our work. The historical development of Kolmogorov complexity theory, the different variants of the concept, and the many applications it has found in several fields of Computer Science are explained in detail in the survey by Li and Vitányi [15] and their recent book [16]. Suitable references or proofs for the claims in the following paragraphs can be found there.

To define the Kolmogorov complexity of real numbers, we first introduce the definition for infinite binary sequences. Our definition of this concept is a time-bounded analog of Kobayashi's "compressibility" version [12].

Definition 4.1: Fix a universal Turing machine U. Let f and g be any two functions between natural numbers and $\alpha \in \{0,1\}^\infty$. We say that $\alpha \in K[f(n), g(n)]$ if there exists $\beta \in \{0,1\}^\infty$ such that, for all but finitely many n, the universal machine U outputs $\alpha_{1:n}$ in time $g(n)$, when given $\beta_{1:m}$ and n as inputs, for any $m \geq f(n)$. If no condition is imposed on the running time, we say $\alpha \in K[f(n)]$.

Fig. 1 shows a graphical intuition of the idea behind this definition.

Observe that here the length of the output is provided for free to the universal machine; so our definition corresponds to a complexity measure "relative to the length." The reason is that we want simple numbers (e.g., rationals) to have extremely low complexity (e.g., constant), and even the information contained in the length of a string could be higher. However, the definitions are equivalent (modulo small constant factors) for complexities at least logarithmic.

Generally, $K[\mathcal{F}, \mathcal{G}]$ is the set of all infinite binary sequences taken from $K[f, g]$ where $f \in \mathcal{F}$ and $g \in \mathcal{G}$. For example, a sequence is in $K[\log, \text{poly}]$ if its prefixes are computable from logarithmically long prefixes of some other sequence in polynomial time.

It is easy to see that every sequence is in $K[n+O(1), \text{poly}]$: any sequence can be produced from itself, plus a constant-size program for the universal Turing machine that copies its input to its output. A straightforward counting argument shows that there are sequences whose complexity is $K[n - O(1)]$, i.e.,

cannot be compressed beyond a constant number of bits. Such sequences are called *(Kolmogorov) random*, and the name is further justified by the fact that they pass all computable statistical tests for randomness. Informally, this means that they have all properties that can be verified algorithmically and hold with probability 1 for an infinite sequence of bits generated by tossing a fair coin. In fact, if an infinite sequence is chosen at random by such a coin-tossing process, the probability of obtaining a Kolmogorov random sequence is 1.

On the other end of the scale, sequences in $K[O(1)]$ are those that can be computed from a constant amount of information. Not surprisingly, they coincide with "recursive" sequences, i.e., characteristic sequences of recursive sets. This is easy to see from our definition, since it follows Kobayashi's variant, but is also true (although harder to show) for the standard definition [17]. It is also known [12], [17] that all characteristic sequences of recursively enumerable sets are in $K[\log]$. So, the characteristic sequence of the halting problem is not in $K[O(1)]$, but is probably far from being random.

Now we define different classes of real numbers by relating them in two ways to infinite binary sequences. We consider two functions mapping $\{0,1\}^\#$ into the real closed interval $[0,1]$: define the functions

$$\delta_2, \delta_4 \colon \{0,1\}^\# \to [0,1]$$

by the formulas

$$\delta_2(\epsilon) = \delta_4(\epsilon) = 0$$
$$\delta_2(\alpha) = \sum_{i=1}^{|\alpha|} \frac{\alpha_i}{2^i}$$
$$\delta_4(\alpha) = \sum_{i=1}^{|\alpha|} \frac{2\alpha_i + 1}{4^i}.$$

Thus δ_2 corresponds to the standard notion of binary expansion of a real number, with the usual technical problem that an infinite tail of ones preceded by a zero denotes the same number as an infinite tail of zeros preceded by a one. On the other hand, δ_4 is injective on $\{0,1\}^\#$, and its image is the "Cantor-like" set

$$\left\{ \sum_{i=1}^{|\beta|} \frac{\beta_i}{4^i} \Big| \beta \in \{1,3\}^\# \right\}.$$

It is easily seen that this set consists of those reals whose binary expansion has a 1 in each even-numbered digit. The inverse map δ_4^{-1} is well defined there. Let Δ_4 be the range of this function when restricted to the domain of infinite strings. That is,

$$\Delta_4 \equiv \left\{ \sum_{i=1}^{\infty} \frac{\beta_i}{4^i} | \beta \in \{1,3\}^\infty \right\}.$$

The function δ_2 can be used to define the Kolmogorov complexity of numbers in $[0,1]$, and either δ_2 or δ_4 can be used to define the Kolmogorov complexity of numbers in Δ_4: a number $\omega \in \Delta_4$ is said to be in $K[f(n), g(n)]$ if and only if $\delta^{-1}(\omega) \in K[f(n), g(n)]$, where δ is either δ_2 or δ_4. In the cases where δ_2 is not injective, either of the two inverse images can be selected, since they have essentially (up to a small additive term) the same Kolmogorov complexity. The values obtained on Δ_4 by using each function differ only in a small constant factor, and all our results will be independent on the way selected to define it.

The definition of Kolmogorov complexity is extended from the interval $[0,1]$ to all $\mathbb{R}$ as follows. We say that $r \in \mathbb{R}$ is in $K[f(n), g(n)]$ if the fractional part of r is; that is, we disregard the (constant) complexity of the integer part of r. Classes of real numbers $K[\mathcal{F}, \mathcal{G}]$ are defined accordingly.

Finally, we use the notation $A + B$ to denote the set $\{a + b | a \in A \wedge b \in B\}$, for two sets $A, B \subseteq \mathbb{R}$.

V. Equivalence of Turing Machines with Tally Oracles and NN's

The following technical definition will allow us to combine a fixed number of real weights into a single infinite string.

Definition 5.1: Let $S \subseteq \{0,1\}^\#$. The set S is *closed under mixing* if for any finite number $k \in \mathbb{N}$ and for any k strings from S

$$\alpha^1 = \alpha_1^1 \alpha_2^1 \alpha_3^1 \cdots, \alpha^2 = \alpha_1^2 \alpha_2^2 \alpha_3^2 \cdots, \cdots, \alpha^k = \alpha_1^k \alpha_2^k \alpha_3^k \cdots$$

the shuffled string

$$\tilde{\alpha} = \alpha_1^1 \alpha_1^2 \alpha_1^3 \cdots \alpha_1^k \alpha_2^1 \alpha_2^2 \alpha_2^3 \cdots \alpha_2^k \alpha_3^1 \alpha_3^2 \cdots$$

is again an element of S.

Via the characteristic function, we can easily associate a real number to each tally set. That is, the ith bit of the real number determines if the word 0^i is in the set. Actually we have available two forms of it: $\delta_2(\chi_T)$ and $\delta_4(\chi_T)$, the second having the property that the real number obtained belongs to Δ_4, as described in the previous section.

The main theorem of this section formalizes that the power of nets with real weights coincides with that of oracle Turing machines with certain tally oracles related to those weights. It is stated as follows:

Theorem 5.2: Let $S \subseteq \{0,1\}^\infty$ be closed under mixing, and assume that $1^\infty \in S$. Let $\mathcal{T}$ be the class of tally sets

$$\mathcal{T} = \{T | \chi_T \in S\}.$$

Time in the following models is polynomially related:

1) Neural networks that have all weights in the set $\delta_2(S) + \mathbb{Q}$.
2) Oracle Turing machines that consult oracles in $\mathcal{T}$.
3) Neural networks that have all weights in the set $\delta_4(S) + \mathbb{Q}$. ■

Proof: We prove that the model in 1) can be simulated by the model in 2), which in turn is simulated by the model in 3), and that the nets in 3) are a subset of those of 1).

1) **by** 2). Assume that we are given a network $\mathcal{N}$ with weights in $\delta_2(S) + \mathbb{Q}$. The network has a fixed number $k \in \mathbb{N}$ of weights, each of which can be written as the sum of a rational and a real in $\delta_2(S)$. Let these reals have base-two expansions

$$\omega^1 = 0.\omega_1^1 \omega_2^1 \omega_3^1, \cdots, \omega^2 = 0.\omega_1^2 \omega_2^2 \omega_3^2 \cdots,$$
$$\cdots \omega^k = 0.\omega_1^k \omega_2^k \omega_3^k \cdots.$$

Note that some of the ω^i's may have two different base-two expansions (namely, ending in infinitely many 0's and infinitely many 1's). We pick the one (or one of the two) that is in S, which we know to exist because the ω^i's are in $\delta_2(S)$. As S is closed under mixing, the string

$$\alpha = \omega_1^1\omega_1^2\omega_1^3\cdots\omega_1^k\omega_2^1\omega_2^2\omega_2^3\cdots\omega_2^k\omega_3^1\omega_3^2\omega_3^3\cdots$$

is again an element of S.

We show the existence of an oracle Turing machine M that consults a tally set with characteristic string $\chi_T = \alpha$, and simulates the network $\mathcal{N}$ while increasing the time only by a polynomial.

Let $B(n)$ be the running time of $\mathcal{N}$, and c be the constant provided by Lemma 3.3 such that precision $c \cdot B(n)$ in the weights guarantees correct results in the computation. Machine M computes as follows:

1. M receives the input string x.
2. For t:= $1, 2, 3, \cdots$ do
   3. For i =1 to $k \cdot c \cdot t$ do
      query 0^i to T (= the ith bit of α).
   4. Using the weights given by $\alpha_{1:kct}$,
      simulate $\mathcal{N}(x)$ step by step until time t.
   5. If $\mathcal{N}$ has produced an output by this time,
      output the same result and halt;
      otherwise, continue with the next t.

To see that the output is correct, note that after Step 3, M has the weights of $\mathcal{N}$ with enough precision to correctly simulate t steps of the computation. The rational parts of the weights are encoded inside the machine M, being a finite, fixed amount of information.

To conclude, note that the time overhead is polynomial in $B(|x|)$. Indeed, the simulation in Step 4 takes time polynomial in t, and the time required to simulate with $t = 1, 2, \cdots, B(|x|)$ is only quadratic in the time to simulate with $t = B(|x|)$. In fact, this quadratic overhead can be reduced to linear by trying only t's that are powers of 2.

2) **by** 3). This proof is based on a change on the model of oracle Turing machine. For an infinite sequence α, an α-TM is a multitape Turing machine equipped with a read-only input tape, a finite number of read–write tapes, and a semi-infinite read-only "reference tape" that permanently contains the infinite word α. The initial configuration has an input in the input tape, α on the reference tape, and the other tapes are blank. We sketch the (easy) proof of the following:

Lemma 5.3: For any time bound t at least linear, the following are equivalent:

- A is accepted by an oracle Turing machine in time polynomial in $t(n)$ with the tally oracle set T;
- A is accepted by a χ_T-TM in time polynomial in $t(n)$.

Proof: Recall that χ_T is relative to the single-letter alphabet. Essentially, the oracle Turing machine is simulated by the χ_T-TM as follows: nonoracle steps are mimicked; when a query is to be made, the machine scans the oracle tape, checking that only 0's appear there, and simultaneously advances the reference tape head. After the scan, the reference tape provides the oracle answer. If the query had some nonzero symbol, then the answer must be NO. The reference head is reset, and the simulation continues. For the converse, the simulation of each move of the χ_T-TM involves a query. The oracle Turing machine keeps the count, on a separate tape, to know the position of the reference tape head. This is used to construct a tally query, whose answer is the bit read by the reference head. Both time overheads are easily seen to be polynomial. ∎

Now, the appropriate combination of simulations in [26] and [27] allows us to argue the inclusion. Fix an oracle Turing machine with oracle set $T \in \mathcal{T}$, and let M be the corresponding χ_T-TM given by Lemma 5.3. Let ω be an arbitrary input to M. The required net acts as follows: first, a small net reads the input ω and stores its δ_4 code as the activation value of a neuron, as in [26, Sec. 4.4]; when the end of the input is reached, it loads $\delta_4(\chi_T)$ as the state of another neuron, through a connection with exactly that weight; finally, it triggers another subnet that, started with $\delta_4(\omega)$ and $\delta_4(\chi_T)$ in two specific neurons, simulates M on input ω and oracle T. This net exists by Theorem 3.2 and its proof, and it takes only linear time for the simulation. Note that the only nonrational weight is $\delta_4(\chi_T) \in \delta_4(S)$.

3) **by** 1). No simulation is needed here. We only show that $\delta_4(S) \subseteq \delta_2(S)$, and then the stated relationship holds *a fortiori*. Consider the real $\delta_4(s)$ for $s \in S$ arbitrary. Its base-four expansion only uses the digits 1 and 3, and therefore its binary expansion consists of a concatenation of pairs 01 or 11. Moreover, a pair 01 appears where s has a 0, and a pair 11 appears where s has a 1. Hence this binary expansion, seen as an infinite sequence s', is exactly the mix of $s \in S$ and 1^∞. Since, by hypothesis, 1^∞ is in S and S is closed under mixing, s' is also in S. Thus $\delta_4(s) = \delta_2(s')$, and $\delta_4(S) \subseteq \delta_2(S)$ follows. ∎

We close this section with the following interesting consequence. In many situations, we can only hope to implement nets with weights that can be effectively computed (or, communicated) using a constant amount of initial information. These are the real numbers in $K[O(1)]$ and, as explained in Section IV, they coincide with the recursive reals, those whose binary expansion is the characteristic sequence of a recursive set. Serving also as an introduction to next section, we can prove now:

Corollary 5.4: In polynomial time, neural networks with recursive weights compute exactly P/poly ∩ REC, the recursive part of P/poly.

Proof: Take the set S of recursive infinite sequences. It is easy to see that it is closed under mixing, and of course it contains 1^∞. Then $\delta_2(S) + \mathbb{Q}$ is exactly the set of recursive reals. By the theorem, the power of polynomial-time neural nets with recursive weights becomes characterized by polynomial-time Turing machines with recursive tally sets as oracles.

Recall that we denote by $\mathcal{P}(B)$ the class of languages that can be decided in polynomial time by an oracle Turing machine querying the oracle B. Now assume that a set A is in $\mathcal{P}(T)$, for T a recursive tally set. Then A is in $\mathcal{P}(T)$ for a recursive T, so it is recursive, and is in $\mathcal{P}(T)$ for T tally, so it is in P/poly by the characterization in Section II (see, e.g., [1]).

Conversely, it is known that if $A \in \mathcal{P}(T)$ for some tally set T, and A is recursive, there is also some recursive tally set T' for which $A \in \mathcal{P}(T')$. Here we sketch one possible construction for T'.

Assume that $A \in \mathcal{P}(T)$ is witnessed by a Turing machine M running in time $p(n)$. Let us say that a string w of length $p(n)$ is a good advice for length n if, for every x of length n, $M(x)$ correctly decides whether $x \in A$ when using any tally set whose characteristic sequence has w as first $p(n)$ bits. Note that $p(n)$ bits are enough for the simulation, and that to decide this property on w it is enough to apply 2^n times the decision procedure for A. Now define w^n to be the lexicographically first good advice for length n. One such string exists, namely, the prefix of χ_T, but any one will do for deciding A. The tally set T' is defined as

$$T' = \{\langle 0^n, 0^i\rangle | n \geq 0, 1 \leq i \leq p(n), \text{ and the } i\text{th bit of } w^n \text{ is } 1\}$$

where the pairing function $\langle \cdot, \cdot \rangle$ is taken to map $0^* \times 0^*$ to 0^*. A straightforward modification of M accepts A relative to T'. It is also easy to prove that T' is recursive, and in fact has complexity at most exponentially higher than A. (It is possible to obtain better bounds on the complexity of T', see [23].) ■

VI. Kolmogorov Weights and Advice Classes

The main contribution here is to show that the Kolmogorov complexity of the weights of nets relates to a structural notion: the amount of advice for nonuniform classes.

We prove it in a general setting, so as to infer at once several particular cases of interest. For this, we introduce some technical conditions that we expect all our advice bounds to have, namely: we say that a class of functions $\mathcal{F}$ forms *reasonable advice bounds* if: 1) $\mathcal{F} \subseteq$ poly, 2) it is closed under $O(\cdot)$, and 3) for every polynomial p and every $f \in \mathcal{F}$ there is another $g \in \mathcal{F}$ such that $f \circ p \leq g$ and $g(n)$ is computable in time polynomial in n. A number of reasonable advice bounds are described below, and all advice bounds used in the literature are immediately seen to be reasonable.

In order to apply Theorem 5.2 to classes of the form $K[\mathcal{F}$, poly], we first show that they are closed under mixing. Here we use that our definition of Kolmogorov complexity is relative to the length.

Lemma 6.1: Let $\mathcal{F}$ be a function class that is closed under $O(\cdot)$. Then, $K[\mathcal{F}$, poly] is closed under mixing.

Proof: Let $k \in \mathbb{N}$ and $\alpha^1, \cdots, \alpha^k \in K[\mathcal{F}$, poly]. That is, for all $i = 1, \cdots k, \alpha^i_{1:n}$ is computed in time $g_i(n) \in$ P using the input $\beta^i_{1:f_i(n)}$. Thus the shuffled string of the α^i's $(\tilde{\alpha})$ up to length n can be computed in time

$$g(n) = O(n) + \sum_{i=1}^{k} g_i(\lceil n/k \rceil)$$

from an input that is equal to the prefix of length

$$f(n) = \sum_{i=1}^{k} f_i(\lceil n/k \rceil)$$

of the shuffled string of the β^i's. It is easy to verify that as both $\mathcal{F}$ and P are closed under $O(\cdot), g \in$ P and $f \in \mathcal{F}$. Thus $\tilde{\alpha} \in K[\mathcal{F}$, poly]. ■

Now we are ready to prove the relation between advice classes and languages accepted by neural nets.

Theorem 6.2: Let $\mathcal{F}$ be a class of reasonable advice bounds. Then the class Pref-P/$\mathcal{F}$ is exactly the class of languages accepted by polynomial-time neural networks with weights in $K[\mathcal{F}$, poly].

Proof: Note that, since all constants are in $O(\mathcal{F}) = \mathcal{F}$ and every rational has constant complexity, $\mathbb{Q} \subseteq K[\mathcal{F}$, poly]. For the same reason, 1^∞ is in $K[\mathcal{F}$, poly]. Furthermore, by Lemma 6.1, the class $K[\mathcal{F}$, poly] is closed under mixing. By Theorem 5.2, it suffices to show that Pref-P/$\mathcal{F}$ coincides with the class of sets decidable in polynomial time with tally oracle sets T such that $\chi_T \in K[\mathcal{F}$, poly].

Thus assume that $A \in \mathcal{P}(T)$ with such a T, and consider an infinite string α such that the first m bits of χ_T can be recovered from the first $f(m)$ bits of α in time polynomial in m, with $f \in \mathcal{F}$. Let p be the polynomial bounding the time necessary to recognize A, and let $g \in \mathcal{F}$ be a bound on $f \circ p$ such that $g(n)$ is computable in time polynomial in n. This g exists by the assumption that $\mathcal{F}$ forms reasonable advice bounds. Then A is in Pref-P/$\mathcal{F}$ by choosing as advice for length n the first $g(n)$ bits of α, from which up to $p(n)$ bits of χ_T can be reconstructed in polynomial time and then used to decide A instead of querying the oracle set T.

Conversely, let $A \in$ Pref-P/$\mathcal{F}$ via the polynomial-time machine M and the infinite word α whose prefix of length $f(m)$ can be used as advice string for length m, with $f \in \mathcal{F}$. Since $\mathcal{F}$ is reasonable, we can take $g \in \mathcal{F}$ that is always greater than f, is easily computable from m, and bounded by a polynomial p.

Let $q(t) = (t+1)p(t)$, so that $p(t+1) \leq q(t+1) - q(t)$. Note that, then

$$(g(t+1) - g(t)) \leq g(t+1) \leq p(t+1) \leq (q(t+1) - q(t)).$$

Consider the oracle tally set T whose characteristic function is defined as follows: bits 1 to $g(1)$ of χ_T are bits 1 to $g(1)$ of α; bits from $g(1)+1$ to $q(1)$ of χ_T are 0; when

$$1 \leq \ell \leq (g(t+1) - g(t))$$

bit $q(t) + \ell$ of χ_T is bit $g(t) + \ell$ of α; and when

$$(g(t+1) - g(t)) < \ell \leq (q(t+1) - q(t))$$

bit $q(t)+\ell$ of χ_T is 0. It is easy to see that, given m and $g(m)$ bits of α, we can print out $q(m) \geq m$ bits of χ_T in polynomial time. So, $\chi_T \in K[\mathcal{F}$, poly], and in polynomial time a Turing machine with oracle T can obtain the necessary advice words to simulate M and, thus, decide A. This completes the proof. ■

Some interesting special cases arise when considering various natural bounds for the Kolmogorov complexity.

- $S = K$[poly, poly], that is, arbitrary strings. The class of languages accepted in this case is P/poly; this is the main result of [27].
- $S = K$[log, poly]. In this case, the class of languages accepted is Pref-P/log, which equals P/log (with our definition) as shown in [3], [7].

- $S = K[O(1), \text{poly}]$, that is, characteristic strings of sets computable in polynomial time. Here it is important that we use Kobayashi's definition, since with other variants the class $K[O(1), \text{poly}]$ may contain characteristic functions of sets requiring arbitrarily high time to decide. The class of languages accepted in this case is P, since, by the results of [26], polynomial-time computations are exactly those of polynomial-time neural nets with rational weights.

VII. The Hierarchy Theorem

In this section we show our main result, namely, the existence of a proper hierarchy of complexity classes of networks. In order to prove this, we first define an ordering between classes of tally sets. Then we show that oracle Turing machines consulting these classes of oracles result in an infinite hierarchy of computational classes. Finally, we apply again the characterization in Theorem 5.2 and obtain the desired hierarchy of neural nets. The advantage of this approach is that Kolmogorov complexity allows for a simpler argument to prove that the classes defined by oracle Turing machines are indeed different.

The partial order on function classes is as follows. Let $\mathcal{F}, \mathcal{G}$ be function classes. We say that $\mathcal{F} \prec \mathcal{G}$ if there is some nondecreasing $s(n) \in \mathcal{G}$, computable in time polynomial in n, so that $s(n) = o(n)$ and, for every polynomial p and every $r \in \mathcal{F}$, $r(p(n)) = o(s(n))$. (Then $\mathcal{F}$ is sublinear by transitivity.)

Note that this partial order defines an infinite hierarchy. For instance, one may consider the function classes $\theta_i = \{q_1, \cdots, q_i\}$, where $q_i = \log^{(i)}$ is defined inductively by $q_1 = \log$ and $q_i = \log(q_{i-1})$ for $i > 1$.

For the next theorem, we denote by $\mathcal{T}_{\mathcal{F}}$ the family of all tally sets T with the property that $\chi_T \in K[\mathcal{F}, \text{poly}]$, and by $\mathcal{P}(\mathcal{T}_{\mathcal{F}})$ the class of all sets decidable in polynomial time by an oracle Turing machine with an oracle in $\mathcal{T}_{\mathcal{F}}$.

Theorem 7.1: Let $\mathcal{F}, \mathcal{G}$ be nonempty function classes, such that $\mathcal{F} \prec \mathcal{G}$. Then, $\mathcal{P}(\mathcal{T}_{\mathcal{F}})$ is properly included in $\mathcal{P}(\mathcal{T}_{\mathcal{G}})$.

Proof: Let $s(n) \in \mathcal{G}$ be a bound on $\mathcal{F}$ as in the definition of the partial order. Note that, since $\mathcal{F}$ is nonempty, s must be unbounded. Trivially, $\mathcal{P}(\mathcal{T}_{\mathcal{F}}) \subseteq \mathcal{P}(\mathcal{T}_{\{s\}}) \subseteq \mathcal{P}(\mathcal{T}_{\mathcal{G}})$. We define a set A that is in $\mathcal{P}(\mathcal{T}_{\mathcal{G}})$ but not in $\mathcal{P}(\mathcal{T}_{\mathcal{F}})$. Choose an infinite sequence $\gamma \notin K[n/2]$. (Such sequences exist as discussed in Section IV.) For each n define the string β_n as

$$\beta_n = \gamma_{1:s(n)/2} \cdot 0^{n - s(n)/2}$$

if $n \geq s(n)/2$, and $\beta_n = 0^n$ otherwise. The dot stands for concatenation.

Let A be the tally set with characteristic string $\beta_1 \cdot \beta_2 \cdot \beta_3 \cdots$. Given $\gamma_{1:s(n)/2}$ and recalling that $s \in$ P, it is easy to build $\chi_{A_{1:n}}$, so that

$$\chi_A \in K[s(n)/2 + c, \text{ poly}] \subseteq K[s(n), \text{ poly}]$$

for some constant c. Hence, $A \in \mathcal{T}_{\mathcal{G}} \subseteq \mathcal{P}(\mathcal{T}_{\mathcal{G}})$.

However, $A \notin \mathcal{P}(\mathcal{T}_{\mathcal{F}})$. Assume, otherwise, that there is some machine that accepts A in time $p_1(n)$ with a tally set T as oracle, where $\chi_T \in K[r(n), p_2(n)]$, p_1 and p_2 are polynomials, and $r \in \mathcal{G}$. In time $p_1(n)$, the machine can query at most the first $p_1(n)$ elements of T. Using this machine, we can easily print out $\beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_n$, and hence $\gamma_{1:s(n)/2}$, given inputs $s(n)/2$ and the first $r(p_1(n)) + O(1) < s(n)/4$ bits of the seed for χ_T. This contradicts the choice of γ. ∎

The requirement that s is nondecreasing can be removed at the expense of some technical complication. For instance, the upper bound on the complexity χ_A only holds infinitely often. To reach a contradiction, it is enough to take γ highly complex almost everywhere.

Let us note that there is nothing special about P classes in the previous theorem. Similar separation results can be proved for other well-behaved families of run-time functions.

As we have seen, Theorem 5.2 establishes a connection between the set of weights of a family of networks and the oracle Turing machines that consult related tally sets. Theorem 7.1 displays a hierarchy of oracle Turing machines querying oracles that belong to different Kolmogorov complexity classes. From both theorems we conclude immediately:

Theorem 7.2 (Hierarchy Theorem): Let $\mathcal{F}, \mathcal{G}$ be two function classes closed under $O(\cdot)$, with $\mathcal{F} \prec \mathcal{G}$. Let $\mathcal{N}_{K[\mathcal{F}, \text{poly}]}$ be the class of languages accepted by networks that compute in polynomial time, and each of which uses weights from $K[\mathcal{F}, \text{poly}]$, and similarly for $\mathcal{G}$. Then

$$\mathcal{N}_{K[\mathcal{F}, \text{poly}]} \subsetneq \mathcal{N}_{K[\mathcal{G}, \text{poly}]}.$$ ∎

This paper introduces a Kolmogorov-like measure of complexity for numbers and applies it to reveal the computational power of recurrent neural networks. We leave the possible applicability of our measure in bridging between information theory and computational complexity for future research.

Acknowledgment

The authors wish to thank E. Sontag for his encouragement and support via many discussions and D. Harel for asking the question that led them to Corollary 5.4.

References

[1] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity I* (Springer-Verlag EATCS Monographs, vol. 11). Berlin, Germany: Springer-Verlag, 1988.

[2] J. L. Balcázar, R. Gavaldà, H. T. Siegelmann, and E. D. Sontag, "Some structural complexity aspects of neural computation," in *Proc. 8th Annu. IEEE Conf. on Structure in Complexity Theory*, 1993, pp. 253–265.

[3] J. L.Balcázar, M. Hermo, and E. Mayordomo, "Characterizations of logarithmic advice complexity classes," in *Information Processing 92*, vol. I, (IFIP Transactions A-12). Amsterdam, The Netherlands: North-Holland, 1992, pp. 315–321.

[4] G. Chaitin, "A theory of program size formally identical to information theory," IMB Tech. Rep. RC 4805, Yorktown Heights, NY, Apr. 1974.

[5] ——, "Information theoretic limitations of formal systems," *J. Assoc. Comput. Mach.*, vol. 21, pp. 403–424, 1974.

[6] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New York: IEEE Press, 1994.

[7] M. Hermo, "On Kobayashi's compressibility of infinite sequences," Res. Rep. LSI–93–36–R, Universitat Politècnica de Catalunya, 1993.

[8] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley, 1991.

[9] T. Hong, "On connectionist models," *Commun. Pure Appl. Math.*, vol. 41, pp. 1039–1050, 1988.

[10] R. M. Karp and R. J. Lipton, "Some connections between uniform and nonuniform complexity classes," in *Proc. 12th ACM Symp. on the Theory of Computing*, 1980, pp. 302–309.

[11] K. Ko, "On helping by robust oracle machines," *Theor. Comput. Sci.*, vol. 52, pp. 15–36, 1987.
[12] K. Kobayashi, "On compressibility of infinite sequences," Res. Rep. C-34, Dept. Inform. Sci., Tokyo Inst. Technol., Tokyo, Japan, 1981.
[13] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Probl. Inform. Transm.* vol. 1, pp. 1–7, 1965.
[14] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Trans. Inform. Theory* vol. IT-22, pp. 75–81, Jan. 1976.
[15] M. Li and P. M. B. Vitányi, "Kolmogorov complexity and its applications," in *Handbook of Theoretical Computer Science*, vol. A. Boston, MA: MIT Press/Elsevier, 1990, pp. 187–254.
[16] _____, *An Introduction to Kolmogorov Complexity and Its Applications*, (Texts and Monographs in Computer Science Series). Berlin, Germany: Springer-Verlag, 1993.
[17] D. W. Loveland, "A variant of the Kolmogorov concept of complexity," *Inform. Contr.*, vol. 15, pp. 510–526, 1969.
[18] P. Martin-Löf, "The definition of random sequences," *Inform. Contr.*, vol. 9, pp. 602–619, 1966.
[19] _____, "Complexity oscillations in infinite binary sequences," *Z. Wahrscheinlichkeit verw. Geb.*, vol. 19, pp. 225–230, 1971.
[20] P. Orponen, "On the computational power of discrete Hopfield nets," *Neural Comput.*, vol. 8, pp. 403–415, 1996.
[21] I. Parberry, *Circuit Complexity and Neural Networks*. Boston, MA: MIT Press, 1994.
[22] C. P. Schnorr, "The process complexity and effective random tests," *J. Comput. Syst. Sci.*, vol. 7, pp. 376–388, 1973.
[23] U. Schöning, *Complexity and Structure* (Lecture Notes in Computer Science, vol. 211). Berlin, Germany: Springer-Verlag, 1986.
[24] H. T. Siegelmann, "Computation beyond the Turing limit," *Science*, vol. 268, no. 5210, pp. 545–548, Apr. 28, 1995.
[25] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," *Appl. Math. Lett.*, vol. 4, no. 6, pp. 77–80, 1991.
[26] _____, "On the computational power of neural nets," *J. Comput. Syst. Sci.*, vol. 50, pp. 132–150, 1995.
[27] _____, "Analog computation via neural networks," *Theor. Comput. Sci.*, vol. 131, pp. 331–360, 1994.