Computational Power of Neural Networks: A Kolmogorov Complexity Characterization *

José L. Balcázar† Ricard Gavaldà†
Department of Software (LSI)
Universitat Politècnica de Catalunya, Barcelona 08028, Spain
E-mail: balqui@lsi.upc.es, gavalda@lsi.upc.es

Hava T. Siegelmann‡
Department of Mathematics and Computer Science
Bar-Ilan University, Ramat-Gan 52900, Israel
E-mail: hava@bimacs.cs.biu.ac.il

December 24th, 1993

Abstract

The computational power of neural networks depends on properties of the real numbers used as weights. We focus on networks restricted to compute in polynomial time, operating on boolean inputs. Previous work has demonstrated that their computational power happens to coincide with the complexity classes P and P/poly, respectively, for networks with rational and arbitrary real weights. Here we prove that the crucial concept that characterizes this computational power is the Kolmogorov complexity of the weights, in the sense that, for each bound on this complexity, the networks can solve exactly the problems in a related nonuniform complexity class located between P and P/poly. By proving that the family of such nonuniform classes is infinite, we show that neural networks can be classified into an infinite hierarchy of different computing capabilities.

1 Introduction

Consider briefly the task of implementing, for some practical purpose, a neural net with real weights. Be it on digital simulator, or on exceedingly precise analogic equipment, knowledge of many decimal positions in the mantissas is necessary for a correct long-term implementation. It may be argued, therefore, that there must exist reasonably efficient ("short") descriptions of the real-valued weights.

Thus, it is interesting to have characterizations of the computational power of the neural models in terms of the amount of information required to specify the reals involved, taking also into account

*A preliminary version of the results described here appears as a part of [2].

†Research supported in part by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II) and by DGICYT, project number PB92–0709.

‡Research supported in part by US Air Force Grant AFOSR-91-0343.

the amount of resources necessary to compute their actual values. Complexity Theory offers several variants of a concept that formalizes precisely such issues: Kolmogorov complexity.

The purpose of this note is to show that this issue is closely linked to a natural question regarding nonuniform complexity classes, namely, the possibility of bounding the amount of nonuniform information allowed for machines in the class. As a consequence, we can prove that, by using as weights slightly more complex reals, one indeed increases strictly the computational power attainable.

More precisely, we set bounds on the resource-bounded Kolmogorov complexity of the reals used as weights in neural nets, and then prove that such bounds correspond precisely to the amount of advice, or nonuniform information, allowed to a family of classes lying between P and P/poly. Some of these classes were studied previously in [3], [6], and other references.

As a technical tool, we start from the known fact that P/poly, as well as some of its subclasses, can be characterized by polynomial time computation relative to tally oracles, i.e., oracle languages defined over a single-letter alphabet. This motivates us to compare reduction classes to tally oracles of diverse complexities, in order to infer properties about real-weight neural models. Using Kolmogorov complexity arguments, we prove that there exists a proper hierarchy of complexity classes defined by neural nets whose weights have increasing Kolmogorov complexity.

Summarizing in an informal but clearer language, the results that we describe here are

1. that polynomial time neural nets, with boolean inputs, solve exactly the problems decidable in polynomial time relative to a certain tally oracle set directly connected to the real weights;
2. that by bounding the Kolmogorov complexity of the weights, such neural nets characterize so-called prefix-advice nonuniform classes with the same bounds on the advice length;
3. and that a proper hierarchy of such complexity classes exists, obtaining, as a consequence, a hierarchy of neural nets: using more complex weights really adds power to the nets.

2 Preliminaries

2.1 The Neural Network Model

We focus on recurrent neural networks that consist of a finite number of neurons. In these networks, the *activation value* or *state* of each processor is updated at times $t = 1, 2, 3, \ldots$, 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

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

where N is the number of processors and M is the number of external input signals and σ is a "sigmoid-like" function. In this paper, σ is a very simple sigmoid, namely the saturated-linear function:

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

As part of the description, we assume that we have singled out a subset of the N processors, say $x_{i_1}, \ldots, x_{i_p}$; these are the p *output processors*, and they are used to communicate the outputs of the network to the environment. Thus a net is specified by the data (a_{ij}, b_{ij}, c_i) together with a subset of its nodes.

2.2 Related Previous Work

Recently, foundational research on the computational power of the recurrent neural network model has been developed [12, 13, 2, 14]. The main results of this line of research are characterizations of classes of formal languages that networks can accept. (A formal language is a set of words over the alphabet $\{0,1\}$. By standard encoding methods, any other finite, fixed alphabet could be assumed if necessary provided that it has at least two different symbols.) For this aim, 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 the recurrent network (with a finite number of neurons) depends of the 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 [12]. Two different I/O conventions are suggested in [13]: 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 1 *Let $\phi : \{0,1\}^* \to \{0,1\}^*$ 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 $O\left(T(\omega) + |\omega|\right)$.* ∎

Theorem 2 *Let M be a Turing machine computing $\phi : \{0,1\}^* \to \{0,1\}^*$ 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*

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

where $\delta : \{0,1\}^* \mapsto [0,1]$ is defined by $\delta(\omega_1\omega_2\cdots\omega_n) = \sum_{i=1}^{n} \frac{2\omega_i+1}{4^i}$. 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

$$\xi(\omega)_2^t \;=\; 0\,, \;\; \text{for } t = 0, \ldots, T(w)-1, \quad \xi(\omega)_2^{T(w)} \;=\; 1 \quad \text{and} \quad \xi(\omega)_1^{T(w)} = \delta[\phi(\omega)].$$

Furthermore, all weights in $\mathcal{N}$ are simple, non-periodic, 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 (may assume infinite precision) the network turns out to have *super-Turing capabilities.* However, it is sensitive to resource constraints and thus is not a tautology. In [14] 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 class P/poly includes all P and, moreover, some non-recursive sets, e.g. a unary encoding of the halting problem. Still, this class contains only a very small fraction of all languages.

  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 [14] 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 2.1 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.

2.3 Complexity Theory

Other than the prefix-advice nonuniform complexity classes, defined below, all the concepts from Complexity Theory mentioned throughout are all standard; see [1], volume 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 $w_{1:k}$ the word consisting of the first k symbols of w, and by w_k the kth symbol of w; this notation is used when w is a finite or an infinite sequence. Symbol ϵ denotes the empty string. The length of a finite word w is denoted by $|w|$, 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 i^{th} bit of the sequence is 1 if and only if the i^{th} 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 $\langle .,. \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ 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* $O(.)$ if for every f, g, if $g \in O(f)$ and $f \in \mathcal{F}$, then $g \in \mathcal{F}$.

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

Our notation for nonuniform complexity classes was introduced in [5]. 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

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

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 $O(.)$.

String w^n in the definition is called "advice for length n". Observe that w^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 [5], and use the classes termed "strong" in [6] 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 $n \leq m$, w^n is a prefix of w^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 $O(.)$) Pref-P$/H$ coincides with P$/H$ according to our definition [4].

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 $\mathrm{P}(T)$ for some tally set T.

3 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.

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

Figure 1: Our Universal Turing Machine.

it has found to several fields of Computer Science are explained in detail in the survey by Li and Vitányi [8] and their recent book [9]. 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 [7].

Definition 3.1 Fix a universal Turing machine U. Let f and g be any two functions between naturals 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 $\alpha \in K[f(n)]$.

Figure 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 $K[\log, \mathrm{poly}]$ 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 $K[n + O(1), \mathrm{poly}]$; 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 for our definition, since it follows Kobayashi's variant, but is also true (although harder to show) for the standard definition [10]. It is also known [7, 10] 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 provably 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

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

by the formulas

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

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:

$$\left\{ \sum_{i=1}^{n} \frac{\beta_i}{4^i} \ \middle| \ \beta \in \{1,3\}^{\infty} \right\}_{n \in \mathbb{N} \cup \{\infty\}} .$$

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,

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

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 $\{ \ a + b \mid a \in A \wedge b \in B \ \}$, for two sets $A, B \subseteq \mathbb{R}$.

4 Equivalence of Turing Machines with Tally Oracles and NNs

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

Definition 4.1 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,

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

the shuffled string

$$\alpha_1^1\alpha_1^2\alpha_1^3\cdots\alpha_1^k\,\alpha_2^1\alpha_2^2\alpha_2^3\cdots\alpha_2^k\,\alpha_3^1\alpha_3^2\cdots$$

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 3 *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*

$$\mathcal{T} = \{\; T \mid \chi_T \in S \;\} .$$

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

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

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

$$\alpha = \frac{\omega_1^1-1}{2}\frac{\omega_1^2-1}{2}\frac{\omega_1^3-1}{2}\cdots\frac{\omega_1^k-1}{2}\frac{\omega_2^1-1}{2}\frac{\omega_2^2-1}{2}\frac{\omega_2^3-1}{2}\cdots\frac{\omega_2^k-1}{2}\frac{\omega_3^1-1}{2}\frac{\omega_3^2-1}{2}\frac{\omega_3^3-1}{2}\cdots$$

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 2.1 such that precision $c \cdot B(n)$ in the weights guarantees correct results in the computation. Machine M performs as follows:

1. M receives the input string x.
2. For $t := 1, 2, 3, \ldots$ 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 $t = 1, 2, \ldots, B(|x|)$ 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 4.2 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: non-oracle 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 non-zero 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 [13] and [14] 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 4.2. Let w be an arbitrary input to M. The required net acts as follows: first, a small net reads the input w and stores its δ_4 code as the activation value of a neuron, as in Section 4.4 of [13]; 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 $\delta_4(w)$ and $\delta_4(\chi_T)$ in two specific neurons, simulates M on input w and oracle T. This net exists by Theorem 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 $K[1]$ and, as explained in Section 3, 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 4.3 In polynomial time, neural networks with recursive weights compute exactly P/poly $\cap$ 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 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 P(T), for T a recursive tally set. Then A is in P(T) for a recursive T, so it is recursive, and is in P(T) for T tally, so it is in P/poly by the characterization in Subsection 2.3 (see e.g. [1]).

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

Assume that $A \in P(T)$ 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

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

where the pairing function $\langle .,. \rangle$ is taken to map 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 [11].) ∎

5 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) $\mathcal{F} \subseteq \text{poly}$, 2) it is closed under $O(.)$, 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 3 to classes of the form $K[\mathcal{F}, \text{poly}]$, we first show that they are closed under mixing. Here we use that our definition of Kolmogorov complexity is relative to the length.

Lemma 5.1 Let $\mathcal{F}$ be a function class that is closed under $O(.)$. Then, $K[\mathcal{F}, \text{poly}]$ is closed under mixing.

Proof. Let $k \in \mathbb{N}$ and $\alpha^1, \ldots, \alpha^k \in K[\mathcal{F}, \text{poly}]$. That is, for all $i = 1, \ldots k$, $\alpha^i_{1:n}$ is computed in time $g_i(n) \in \text{P}$ 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 $g(n) = O(n) + \sum_{i=1}^{k} g_i(\frac{n}{k})$ from an input that is equal to the prefix of length $f(n) = \sum_{i=1}^{k} f_i(\frac{n}{k})$ of the shuffled string of the β^i's. It is easy to verify that as both $\mathcal{F}$ and P are closed under $O(.)$, $g \in \text{P}$ and $f \in \mathcal{F}$. Thus, $\tilde{\alpha} \in K[\mathcal{F}, \text{poly}]$. ∎

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

Theorem 4 *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 $K[\mathcal{F}, \mathit{poly}]$.*

Proof. Note that, since all constants are in $O(\mathcal{F}) = \mathcal{F}$ and every rational has constant complexity, $\mathbb{Q} \subseteq K[\mathcal{F}, \text{poly}]$. For the same reason, 1^∞ is in $K[\mathcal{F}, \text{poly}]$. Furthermore, by Lemma 5.1, the class $K[\mathcal{F}, \text{poly}]$ is closed under mixing. By Theorem 3, 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 $\chi_T \in K[\mathcal{F}, \text{poly}]$.

Thus, assume that $A \in \text{P}(T)$ 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 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 $q(1)$ of χ_T are zero; 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, and vice versa. So, $\chi_T \in K[\mathcal{F}, \text{poly}]$, 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:

- $S = K[\text{poly}, \text{poly}]$, that is, arbitrary strings. The class of languages accepted in this case is P/poly; this is the main result of [14].
- $S = K[\log, \text{poly}]$. In this case, the class of languages accepted is Pref-P/log, which equals P/log (with our definition) as shown in [3, 4].
- $S = K[O(1), \text{poly}]$, 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 [13], polynomial-time computations are exactly those of polynomial-time neural nets with rational weights.

6 A 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 3 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 all 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 polynomial-time computable $s \in \mathcal{G}$ so that $s = o(n)$ 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 $\theta_i = \{q_1, \ldots, q_i\}$, 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 $\text{P}(\mathcal{T}_\mathcal{F})$ the class of all sets decidable in polynomial time by an oracle Turing machine with an oracle in $\mathcal{T}_\mathcal{F}$.

Theorem 5 *Let $\mathcal{F},\mathcal{G}$ be function classes, such that $\mathcal{F} \prec \mathcal{G}$. Then, $P(\mathcal{T}_\mathcal{F})$ is properly included in $P(\mathcal{T}_\mathcal{G})$.*

Proof. Let $s(n) \in \mathcal{G}$ be a bound on $\mathcal{F}$ as in the definition of the partial order. Trivially, $\mathrm{P}(\mathcal{T}_\mathcal{F}) \subseteq \mathrm{P}(\mathcal{T}_{\{s\}}) \subseteq \mathrm{P}(\mathcal{T}_\mathcal{G})$. We define a set $\mathcal{A}$ that is in $\mathrm{P}(\mathcal{T}_\mathcal{G})$ but not in $\mathrm{P}(\mathcal{T}_\mathcal{F})$. Choose an infinite sequence $\gamma \notin K[n/2]$. (Such sequences exist as discussed in Section 3.) 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 $\mathcal{A}$ be the tally set with characteristic string $\beta_1 \cdot \beta_2 \cdot \beta_3 \ldots$ Given $\gamma_{1:s(n)/2}$ and recalling that $s \in \mathrm{P}$, it is easy to build $\chi_{\mathcal{A}_{1:n}}$, so that

$$\chi_\mathcal{A} \in K[s(n)/2 + c, \mathrm{poly}] \subseteq K[s(n), \mathrm{poly}] \ ,$$

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

However, $\mathcal{A} \notin \mathrm{P}(\mathcal{T}_\mathcal{F})$. Assume otherwise, then there is some machine that accepts $\mathcal{A}$ 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. ∎

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 3 establishes a connection between the set of weights of a family of networks and the oracle Turing machines that consult related tally sets. Theorem 5 displays a hierarchy of oracle Turing machines querying oracles that belong to different Kolmogorov complexity classes. From both theorems we conclude immediately:

Theorem 6 *(*HIERARCHY THEOREM*) Let $\mathcal{F},\mathcal{G}$ be two function classes closed under $O(.)$, with $\mathcal{F} \prec \mathcal{G}$. Let $\mathcal{N}_{K[\mathcal{F},poly]}$ be the class of languages accepted by networks that compute in polynomial time, and each of which uses weights from $K[\mathcal{F}, poly]$, and similarly for $\mathcal{G}$. Then:*

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

∎

7 Acknowledgements

We are grateful to Eduardo Sontag for his encouragement and support via many discussions. We also thank David Harel for asking the question that led us to Corollary 4.3.

References

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

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

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

[4] Hermo M., "On Kobayashi's Compressibility of Infinite Sequences", Research Report LSI–93–36–R, Universitat Politècnica de Catalunya, 1993.

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

[6] Ko K., "On helping by robust oracle machines", *Theoretical Computer Science,* 52, 1987, 15–36.

[7] Kobayashi K., "On compressibility of infinite sequences", Research Report C-34, Department of Information Sciences, Tokyo Institute of Technology, 1981.

[8] Li M. and P.M.B. Vitányi: "Kolmogorov complexity and its applications", in *Handbook of Theoretical Computer Science,* vol. A, MIT Press/Elsevier, 1990, 187–254.

[9] Li M. and P.M.B. Vitányi: *An Introduction to Kolmogorov Complexity and Its Applications,* Texts and Monographs in Computer Science Series, Springer-Verlag, 1993.

[10] Loveland D.W., "A variant of the Kolmogorov concept of complexity", *Inform. and Control* 15, 1969, 510–526.

[11] Schöning U.: *Complexity and Structure,* Lecture Notes in Computer Science 211, Springer-Verlag, 1986.

[12] Siegelmann H.T. and E.D. Sontag, "Turing computability with neural nets," *Appl. Math. Lett.* 4(6), 1991: 77-80.

[13] Siegelmann H.T. and E.D. Sontag, "On the computational power of neural nets," in *Proc. Fifth ACM Workshop on Computational Learning Theory*, Pittsburgh, July 1992, 440-449. A detailed version will appear in *J. Comp. Syst. Sciences.*

[14] Siegelmann H.T. and E.D. Sontag, "Analog Computation Via Neural Networks," in *Second Israel Symposium on Theory of Computing and Systems,* Natanya, Israel, June 1993. A detailed version will appear in *Theoretical Computer Science.*