Theoretical Computer Science 499 (2013) 2–22

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

The *ARNN* model relativises $P = NP$ and $P \neq NP$

José Félix Costa [a,b,*], Raimundo Leong [a]

[a] Department of Mathematics, Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal
[b] Centro de Matemática e Aplicações Fundamentais do Complexo Interdisciplinar, Universidade de Lisboa, Lisbon, Portugal

ARTICLE INFO

Article history:
Received 2 October 2011
Received in revised form 27 April 2013
Accepted 13 May 2013
Communicated by C.S. Calude

Keywords:
Analog recurrent neural nets (*ARNN* model)
Non-deterministic neural nets
Dynamical systems
Cost of an oracle
Simulation of a real weighted neural net by an oracle Turing machine
Positive relativisation of $P = NP$

ABSTRACT

In this paper we prove that the relations $P = NP$ and $P \neq NP$ relativise to the deterministic/non-deterministic artificial recurrent neural net (*ARNN*) with real weights (informally considered as *oracles* in Martin Davis (2006) [10,11]). Although, in the nineties, a dozen of papers were written on the *ARNN* model, some introducing computation via neural nets with real weights and some introducing non-deterministic and stochastic neural nets, it seems that no one noticed such a relativisation, which makes the *ARNN* an interesting but restricted model of computation.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

The computability analysis of some classes of analog recurrent neural networks can be found in, inter alia, [23,22,19,21]. In [21], lower and upper bounds on their computational power are established under diverse limitations of resources.

These nets satisfy the classical constraints of computation theory, namely, (a) input is discrete (binary) and finite, (b) output is discrete (binary) and finite, and (c) the system is itself finite (control is finite). However, the neurons may hold values in $[0, 1]$ with unbounded precision. The infiniteness arises from two different sources: real valued weights such like physical constants or real valued probabilities of outcome.

In such analog systems, the binary inputs are encoded into rational numbers in the unit interval $]0, 1[$, and the output (supposed to be a rational number) is decoded into a binary sequence too. The technique used in [23,22] consists of an encoding of the inputs into the Cantor set of either base 4 or base 9.

We may then identify the class of sets decidable by analog recurrent neural nets, once provided the type of the weights.

The first level is the class of nets $ARNN[\mathbb{Z}]$ (see [20,21]).[1] These nets are historically related with the work of Warren McCulloch and Walter Pitts. Since the weights are integer numbers, each processor can only compute a linear integer combination of zeros and ones. The resulting values are always zero or one. In this case the nets "degenerate" into classical devices called *finite automata*. It was Kleene who first proved that McCulloch and Pitts nets are equivalent to finite automata and, therefore, they are able to decide exactly the regular languages.

* Corresponding author at: Department of Mathematics, Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal.
E-mail addresses: fgc@math.ist.utl.pt (J.F. Costa), rcoleong@gmail.com (R. Leong).

[1] In what follows, we denote by *ARNN*[.], possibly with more suffixes, both the class of nets satisfying some structural property and the corresponding class of sets.

0304-3975/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.tcs.2013.05.022

The second relevant class is $ARNN[\mathbb{Q}]$ (see [23,21]): it is equivalent to the Turing machines. Twofold equivalent: rational nets decide the same sets as Turing machines and, under appropriate encoding of input and output, they are able to decide the same sets in exactly the same time. The class $ARNN[\mathbb{Q}]$ coincide with the class of recursive enumerable sets.

The third relevant class is $ARNN[\mathbb{R}]$ (see [22,21]). Reals are in general non-computable. As shown in [22,21], all sets over finite alphabets can be encoded as real weights. Under polynomial time computation, these networks simulate not only all efficient Turing computations, but also sparse oracle Turing machines (their power is exactly $P/poly$).

The way the last result mentioned above was proved raised controversy in the nineties, since the *ARNN* model displays the so-called hypercomputation effect. Here, we provide a few clues that might help the reader to go through our paper with a clear mind.

If we assume synaptic plasticity and a more physical realizable activation function for the neurons (system units), then the *ARNN* model turns to be the recurrent neural net used in engineering applications and usually trained (e.g., to learn grammar) using a modified method of *backpropagation* (see [12]). This (steepest descent) method works only with real weighted networks. Thus there is nothing special about neural nets with real weights, at least theoretically — it is a common model. But the focus of the papers cited above was digital computation. For that purpose, the authors considered a piecewise linear activation function rather than the regular sigmoid used in advanced learning theory. (A further paper (see [14]) attempts to prove that the computational power of the piecewise linear activation function does not collapse when it is replaced by the standard sigmoid.)

The classification of the computational power of real-weighted systems described in [23,22,19,21] is surely a meritorious work. From the point of view of models of the real world, these studies indicate that such systems tuned with real-valued parameters (if such an assignment exists) may display behaviour not simulable by a Turing machine. However, it does not mean that we can set the weights of the net in a programmable fashion, in order to compute above the Turing limit. Such a pretension was the subject of a paper in Science (the journal, see [19]), that attracted criticism. Professor Davis, in [10,11], addressed this issue by saying that the *author has put in the system exactly the information she wants to extract*, and by looking at the real number as an oracle, he reduced the *ARNN* model to that of a Turing machine equipped with an oracle.

In this paper we prove that the real number, inbuilt in the *ARNN* model in [22], can be seen as a conventional oracle; moreover, the nature of this oracle is such that the relation $P = NP$ relativises, i.e., the following result holds. If $ARNN[\mathbb{Q} \cup \{r\}]P$ denotes the class of sets decidable with deterministic rational nets in polynomial time with an extra single real weight r and $ARNN[\mathbb{Q} \cup \{r\}]NP$ denotes the class of languages decidable with non-deterministic rational nets in polynomial time with an extra single real weight r, then:

Theorem 1. *If $P = NP$, then, for every real weight r, $ARNN[\mathbb{Q} \cup \{r\}]P = ARNN[\mathbb{Q} \cup \{r\}]NP$.*[2]

Such a positive relativisation shows that the *ARNN* model is a restricted model of computation. We then explore this fact as a clear separation between physical and mathematical oracles, being the *ARNN* model closer to physical theories than the conventional Turing machine with oracle (Turing's o-machine). In fact, a more plausible *ARNN* model causes the full relativisation of $P = NP$:

Theorem 2. *If $P \neq NP$, then, for every real weight r, $ARNN[\mathbb{Q} \cup \{r\}]P \neq ARNN[\mathbb{Q} \cup \{r\}]NP$.*[3]

We adopted the definition of non-deterministic neural net provided in [21]. As far as we know, such nets are not yet characterized in the literature and the definition on a non-deterministic neural net seems not to be unique.

The idea of the proof of positive relativisation is as follows.

Once we bound the number of oracle calls of a Turing machine to a polynomial on the size of the input we get a relativisation of $P = NP$. Such a counting cannot be done by the oracle Turing machine itself since it is a counting over all possible non-deterministic branches of the tree of computations. However, the counting can be avoided if we restrict the oracles to the sparse or tally ones. We prove that the *ARNN* model with the saturated sigmoid activation function is simulable by Turing machines with sparse oracles.

Once we bound the number of oracle calls of a Turing machine to the logarithm on the size of the input we get the relativisation of $P \neq NP$. If the activation function becomes the analytic one, then in polynomial time only a logarithmic number of queries can be effective all over the branches of the non-deterministic trees. In this case, the counting cannot be avoided by "known/common" oracles (such like the tally oracles). We prove that the *ARNN* model with the analytic sigmoid activation function is simulable by the oracle Turing machine consulting its oracle a logarithmic number of times. It turns out that the double relativisation is true for the *ARNN* model with the analytic sigmoid.

In the classical positive relativisation condition, the logarithmic bound on the number of queries cannot be controlled by the Turing machine itself, since it is an overall counting over all the branches. It is a meta-machine counting. But in the *ARNN* case, that counting is not needed since the system cannot do more, and such result seems to be quite interesting for natural computation.

[2] This result is true for a finite number of real weights.

[3] Idem.

As described above, the method of proof of the two theorems consists of simulating the *ARNN* with real weights by the oracle Turing machine. To prove these statements, we need to recall knowledge about the *ARNN* model. We tried to reduce to a minimum the concepts needed, but the reader can find within this paper a survey of the main features of the *ARNN* model. Proofs done prior to this paper are fully referenced; all the others are done for the purpose of proving the above theorems.

2. The artificial recurrent neural net

2.1. The ARNN model

We recall the concept of Analogue Recurrent Neural Net — the *ARNN* model — as in [22,23,21]. The state space is $\mathbb{R}^n$, for some specified n (dimension of the system), although we will consider subspaces, namely $\mathbb{Q}^n$ and $\mathbb{Z}^n$ over $\mathbb{Q}$ and $\mathbb{Z}$, respectively. The state will be denoted by $\vec{x}$ of n components $x_1(t), \ldots, x_n(t)$. Inputs will be total functions of signature $\mathbb{N} \to \{0,1\}$, i.e., streams of Boolean values. The state of the input is given at any moment of time t by a vector $\vec{u}(t)$ of m components $u_1(t), \ldots, u_m(t)$ (for some m, the number of input stream lines). To specify the dynamical map we will consider matrices A of dimension $n \times n$, and B of dimension $n \times m$, both composed of real numbers. Sometimes we will restrict those values to the rationals or to the integers. We can always consider a state variable with fixed value 1 and workout the dynamic map to write it as follows[4]:

$$x_1(t+1) = \sigma_1\big(a_{11}x_1(t) + \cdots + a_{1n}x_n(t) + b_{11}u_1(t) + \cdots + b_{1m}u_m(t) + c_1\big)$$
$$\ldots$$
$$x_n(t+1) = \sigma_n\big(a_{n1}x_1(t) + \cdots + a_{nn}x_n(t) + b_{n1}u_1(t) + \cdots + b_{nm}u_m(t) + c_n\big)$$

This system can be presented in abbreviated form by

$$\vec{x}(t+1) = \vec{\sigma}\left(A\vec{x}(t) + B\vec{u}(t) + \vec{c}\right)$$

The most common functions used, $\sigma_1, \ldots, \sigma_n : \mathbb{R} \to \mathbb{R}$, belong to the following classes:

(a) The McCulloch–Pitts sigmoid (see [16,12]),

$$\sigma_d(x) = \begin{cases} 1 & \text{if } x \geqslant 0 \\ 0 & \text{if } x < 0 \end{cases}$$

(b) The saturated sigmoid (see [22,23]),

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

(c) The analytic sigmoid of parameter k (see [12]),

$$\sigma_{a,k}(x) = \frac{1}{1 + e^{-kx}}$$

Definition 1. Given a system $(\Sigma)\ \vec{x}(t+1) = \vec{\sigma}(A\vec{x}(t) + B\vec{u}(t) + \vec{c})$, with initial condition $\vec{x}(0)$ and $\vec{u}(0)$, a *finite computation* of Σ is a sequence of state transitions $\vec{x}(0)\vec{x}(1)\ldots\vec{x}(t)$ such that, for every $1 \leqslant \tau \leqslant t$, $\vec{x}(\tau) = \vec{\sigma}(A\vec{x}(\tau-1) + B\vec{u}(\tau-1) + \vec{c})$.

Definition 2. Given a system $(\Sigma)\ \vec{x}(t+1) = \vec{\sigma}(A\vec{x}(t) + B\vec{u}(t) + \vec{c})$, with initial condition $\vec{x}(0)$ and $\vec{u}(0)$, a *computation* of Σ is an infinite sequence of state transitions $\vec{x}(0)\vec{x}(1)\ldots\vec{x}(t)\ldots$ such that, for every $\tau > 0$, $\vec{x}(\tau) = \vec{\sigma}(A\vec{x}(\tau-1) + B\vec{u}(\tau-1) + \vec{c})$.

We choose a collection of state components, within the n components, to denote the output of the system. Those variables are called *effectors*, provided that they are always Boolean valued. For those effectors we can define an *output stream*, i.e., a map $v : \mathbb{N} \to \{0,1\}$, such that, if x_k is an effector, then $v(t) = x_k(t)$.

Definition 3. We say that a system Σ is in equilibrium at time t if its state is $\vec{0}$.

We will only consider systems with all $\vec{c}$-components less or equal to zero, which are in equilibrium at $t = 0$, and such that $\vec{u}(0) = \vec{0}$. Consequently, at time $t = 1$, the state is $\vec{x}(1) = \vec{\sigma}(A\vec{x}(0) + B\vec{u}(0) + \vec{c}) = \vec{0}$.

We will be working with the saturated sigmoid, but in Section 9 the analytic sigmoid will be the case.

[4] In this case the dimension n of the system does not include the state component holding the value 1.

2.2. Hard-wiring a system

Let us give two examples to show that such a system can compute as any other abstract machine. For the purpose, we will consider only saturated sigmoids as activation functions of the state components of the dynamical system. Some conventions on how to input data and extract the result from this systems have to be established. Let $\phi : \{0,1\}^+ \to \{0,1\}^+$ be a function and $w \in \{0,1\}^+$ given as conventional input: we will consider two input streams, one is $0w0^\omega$ and the other, to validate the sequence of time steps that the conventional input takes, is $01^{|w|}0^\omega$. Analogously, we use two streams for the system to output the result: the first one is the validation line $00^{t-1}1^{|\phi(w)|}0^\omega$, where $t > 1$ is the time step of the first bit of the output, and the output stream $00^{t-1}\phi(w)0^\omega$, where $\phi(w)$ is the expected answer.

We will consider hardwired structures, that is, the weights of neural nets (the entries of the matrices A, B and vector $\vec{c}$) are fixed at start. This means that we will be dealing with time invariant systems. The dynamics can then be expressed as $\vec{x}^+ = \sigma(A\vec{x}(t) + B\vec{u}(t) + \vec{c})$ where A and B are linear maps, which can be represented by matrices and c is a vector.

The first example is a rather simple but clarifying one — the *unary successor*. There are many simple networks simulating this operation.[5] Consider the system:

$$\begin{cases} y_1^+ = \sigma(a) \\ y_a^+ = \sigma(a + y_1) \end{cases}$$

with a as input stream and y_a providing the output stream. Note that in this case the validation streams are not necessary since the output is in unary. The reader can easily check that this system in fact computes the successor of the input in constant time.

Now, we will define one system that compute addition in unary in linear time. The streams a and b denote the summands, v their validation, and y_{a+b} the result, respectively.

The first part

$$\begin{cases} y_1^+ = \sigma(v) \\ y_2^+ = \sigma(y_1) \end{cases}$$

simply copies the input of the bigger number, digit by digit (being it delayed to match the timing of the next procedure). The second part

$$\begin{cases} y_3^+ = \sigma(a + b - 1) \\ y_4^+ = \sigma\left(\dfrac{1}{2}(y_3 + y_4) - (1 - y_3)\right) \end{cases}$$

codifies the term with smaller value in *2-Cantor set*. Next, this result will be "saved" in the next neuron

$$\{\, y_5^+ = \sigma\big(y_4 + y_5 - (1 - y_1) - y_3\big)$$

while the result from the first part is being exported. When it ends, the following part decodes the "saved" information[6]:

$$\begin{cases} y_6^+ = \sigma\big(2(y_5 + y_6 + y_4) - 1 - y_1\big) \\ y_7^+ = \sigma\big(2(y_5 + y_6 + y_4) - 2y_1\big) \end{cases}$$

Finally, the output neuron prints out the result $y_{a+b}^+ = \sigma(y_2 + y_7)$.

This example shows that the systems not only can perform computations, but also have the capacity of memory. This is due to the use of the saturated sigmoid, which allows to uniquely encode inputs into rational numbers in the interval $[0, 1]$. These systems can perform computations as other abstract machines such as finite automata or Turing machines.

2.3. Basic logical predicates

All quantifier-free Boolean formulas can be expressed with 0-ary predicates 0 and 1 (meaning *false* and *true*, respectively), unary predicate *NOT* and binary predicates *AND* and *OR*. The 0-ary predicates can be expressed by a single unit with the dynamics $z_0^+ = \sigma(0)$ and $z_1^+ = \sigma(1)$.[7] The unary function *NOT* is computed by $z_{NOT}^+ = \sigma(-u + 1)$. The predicates *AND* and *OR* given u_1 and u_2 as input can be simulated by $z_{AND}^+ = \sigma(u_1 + u_2 - 1)$ and $z_{OR}^+ = \sigma(u_1 + u_2)$.

By composing these units, we can compute any Boolean formula. To be able to distinguish between an output unit at rest and the one that sends the signal *false*, we use an extra unit called the validation of the output. When this unit holds

[5] The examples that follow are not published elsewhere.

[6] Note that if the terms are equal, there is nothing to be saved.

[7] Since positive bias is not permitted, such like 1, this value is injected in the proper neuron from the input or input validation streams.

the value 1, the value of the output unit holds the result of the computation. Similarly, we add an extra input validation line.

Let $\varphi(u_1, u_2, u_3) = u_1$ *AND* (u_2 *OR NOT* u_3). Clearly, it can be done in three layers. The first one to compute *NOT* u_3 and to hold the values of u_1 and u_2:

$$\begin{cases} x^+_{1,1} = \sigma(u_1) \\ x^+_{1,2} = \sigma(u_2) \\ x^+_{1,3} = \sigma(-u_3 + 1) \end{cases}$$

A second layer to compute the disjunction and to hold the value of u_1:

$$\begin{cases} x^+_{2,1} = \sigma(x_{1,1}) \\ x^+_{2,2} = \sigma(x_{1,2} + x_{1,3}) \end{cases}$$

The last one is to compute the function φ:

$$x^+_{\varphi} = \sigma(x_{2,1} + x_{2,2} - 1)$$

In three steps, the solution is computed. Denoting the input validation line by v, we add some extra units to compute the output validation $x^+_{\varphi,v}$:

$$\begin{cases} x^+_{0,1} = \sigma(v) \\ x^+_{0,2} = \sigma(x_{0,1}) \\ x^+_{\varphi,v} = \sigma(x_{0,2}) \end{cases}$$

The composite output is then given by the values of the state variables x_φ and $x_{\varphi,v}$.

2.4. Memory and local inhibition

Since cycles in single processors are allowed, one can easily understand that a single unit has the capacity to hold a value (forever, if needed). Given input u, a neuron x with the dynamics $x^+ = \sigma(x+u)$ can save the value of u. Once u feeds in the value 1, x will hold the first 1 of u forever. To build a more complex unit x that saves the last value introduced, consider one input line u and input validation line v with the dynamics $x^+ = \sigma(x + 2u - v)$.

Suppose now that we have another unit, y, and we want it to download the value of x when it receives an input 1 from input line u. This downloading unit can follow, e.g., the dynamics $y^+ = \sigma(x+u-1)$. The input line u can be seen as a *switch*. This idea can be generalised. Suppose that a unit y is defined as $y^+ = \sigma(\pi(x_{i_1}, \ldots, x_{i_k}) + bu - b)$, where $\pi : \{0,1\}^k \to \mathbb{Z}$ is a linear function with integer coefficients $a_1, \ldots, a_k$. Let a_l be the coefficient with greatest absolute value among them and $c = |a_l|$. Then the values held by π are within $[-ck, ck]$. By setting $b = ck + 1$ we guarantee that: if $u(t) = 0$, then $y(t+1) = 0$; if $u(t) = 1$, then $y(t+1) = \sigma(\pi(x_{i_1}(t), \ldots, x_{i_k}(t)))$.

Proposition 1 *(Switch Lemma). Let y be a unit of a system with dynamics $y^+ = \sigma(\pi(\vec{x}))$. Then, there exists a new system with a unit $\tilde{y}$ with the dynamics $\tilde{y}^+ = \sigma(\tilde{\pi}(\vec{x}, x_{switch}))$ such that, if $x_{switch}(t) = 0$, then $\tilde{y}(t+1) = 0$, and if $x_{switch}(t) = 1$, then $\tilde{y}(t+1) = y(t+1)$.*

2.5. Sequential composition and synchronization

Components of an ARNN might not have binary inputs and outputs. We will refer to them as subsystems or subnetworks. Note that an *ARNN* is a particular kind of subsystem while the reverse is not always true.

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two subsystems, where $\mathcal{N}_1$ has two output units x_{out} and $x_{out,v}$ (denoting output and output validation, respectively) and $\mathcal{N}_2$ has two input lines u_{in} and $u_{in,v}$ (denoting input and input validation, respectively) connected to units $x_1, \ldots, x_n$.

To connect $\mathcal{N}_1$ to $\mathcal{N}_2$, we add two extra units, x_{in} and $x_{in,v}$, to $\mathcal{N}_2$ such that $x^+_{in} = \sigma(x_{out})$ and $x^+_{in,v} = \sigma(x_{out,v})$. Then, we simply change $x_1, \ldots, x_n$ by substituting in their dynamics u_{in} and $u_{in,v}$ by x_{in} and $x_{in,v}$, respectively.[8]

After the download performed by $\mathcal{N}_2$ on the computation result of $\mathcal{N}_1$, $x_{out,v}(t) = 0$ and $x_{in,v}(t+1) = 0$, but the input unit x_{in} still needs to be switched off at $t+1$. We can add a switch as in Proposition 1 to x_{in}, that is, we can change its dynamics from $x^+_{in} = \sigma(x_{out})$ to $x^+_{in} = \sigma(x_{out} + x_{out,v} - 1)$. Suppose that $x_{out,v} = 1$ at $t = T$ and, for $t < T$, $x_{out,v} = 0$. At time $T+1$, $\mathcal{N}_2$ downloads the value that x_{out} holds, and, at time $T+2$, $\mathcal{N}_2$ starts its computation.

[8] The units x_{in} and $x_{in,v}$ are needed because $x_1, \ldots, x_n$ can depend also on other units of $\mathcal{N}_2$ and we do not want to change the dynamics of units that make part of the computations of $\mathcal{N}_2$.

If we want to shut $\mathcal{N}_1$ down after the download, we can add a switch unit x_{switch}. Let $x^+_{out,v} = \sigma(\pi(\vec{x}))$ and b be the constant in proof of Proposition 1. We set $x^+_{switch} = \sigma(-\pi(\vec{x}) + bx_{out,v} - b(1 - x_{out,v}) + bx_{switch})$ and we add to the dynamics of every unit x_i in $\mathcal{N}_1$ (including $x_{out,v}$) an extra term $-b_i x_{switch}$, where b_i is the constant of Proposition 1 for the respective unit.

Proposition 2. *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two subsystems. Then we can connect them sequentially into a new subsystem working as follows: $\mathcal{N}_1$ starts its computation while $\mathcal{N}_2$ is in equilibrium until the output of $\mathcal{N}_1$ is fed into $\mathcal{N}_2$. If needed, $\mathcal{N}_1$ can be shut down after $\mathcal{N}_2$ has downloaded its output.*

Now, consider two subsystems $\mathcal{N}_1$ and $\mathcal{N}_2$ working in different times and a third subsystem $\mathcal{N}$ waiting to receive as inputs the outputs from output units $x_{out,1}$ and $x_{out,2}$ of $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. We want to feed their outputs at the same time into $\mathcal{N}$. This is the synchronization problem in combining subsystems *in parallel*.

Without loss of generality, suppose $\mathcal{N}$ is composed by only one unit, x, with two input lines, u_1 and u_2, receiving signals from $\mathcal{N}_1$ and $\mathcal{N}_2$.

We add two units o_i and v_i to $\mathcal{N}_i$, where $o_i^+ = \sigma(o_i + x_{out,i})$ and $v_i^+ = \sigma(v_i + x_{out,v,i})$. These units hold the outputs and output validations. A switch unit x_{switch} works as follows[9]: $x^+_{switch} = \sigma(-x_{switch} + v_1 + v_2 - 1)$. We add two extra units, $x_{in,1}$ and $x_{in,2}$, to the system $\mathcal{N}$, with dynamics $x_{in,i} = \sigma(o_i)$ and replace u_i by $x_{in,i}$ in $\mathcal{N}$. To finish, we modify v_i, o_i, and $x_{in,i}$ according to Proposition 1.

Proposition 3. *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two subsystems with one output unit and $\mathcal{N}$ with two input lines. Then we can build a subsystem that works as follows: $\mathcal{N}_1$ and $\mathcal{N}_2$ start computing at $t = 1$ while $\mathcal{N}$ is in equilibrium. The output of one of the subsystems is saved until the other has finished its computation. When both have finished their computations, the outputs are fed into $\mathcal{N}$.*

We can turn off the subsystems $\mathcal{N}_1$ and $\mathcal{N}_2$ being synchronized after the download into $\mathcal{N}$.

These results are of extreme importance for the rest of this work, allowing description of neural networks by separated subsystems. Despite their direct application in what follows, this is just an unfolded corner of the discussion about the hard-wiring of neural nets and logical description of realizable logical propositions. For further results, see [16], [18], and [13].

2.6. Characterization of computational model

We ask now whether a system can recognize or decide a set:

Definition 4. A word $w \in \{0,1\}^+$ is said to be classified in time ν by a system $\mathcal{N}$ if the input streams are $\langle A, V\rangle$, with $A = 0w0^\omega$ and $V = 01^{|w|}0^\omega$, and the output streams are $\langle U, R\rangle$ with $R(t) \equiv (t = \nu)$. If $U(\nu) = 1$, then the word is said to be accepted, otherwise (if $U(\nu) = 0$) rejected.

The classes of sets decided by integral, rational, and real *ARNNs* will be denoted, respectively, by *ARNN*$[\mathbb{Z}]$, *ARNN*$[\mathbb{Q}]$, and *ARNN*$[\mathbb{R}]$.

2.6.1. Integral ARNNs

Neural nets with integer weights are an equivalent variant of those introduced by McCulloch and Pitts [16]. Since the state variables only hold linear combinations of 0 and 1, the activation function "degenerates" into a step function:

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leqslant 0 \end{cases}$$

Kleene proved the equivalence between finite automata and integral nets (see [17,20]).

Proposition 4. *A is a regular language if and only if $A \in$ ARNN$[\mathbb{Z}]$.*

2.6.2. Rational ARNN

The saturated sigmoid allows a trivial encoding of information of arbitrarily large size. For words $w \in \{0,1\}^+$, we can use $[\cdot]_4 : \{0,1\}^+ \to [0,1]$ given by:

$$[z]_4 = \sum_{i=1}^{|z|} \frac{2z_i + 1}{4^i}$$

[9] The constant term is $-(n-1)$ when we need to combine n systems. The term $-x_{switch}$ is a switch to turn itself off.

The set $\{0,1\}^+$ is mapped to a subset of the rational numbers (actually, the image of this function is a Cantor subset of the rationals). The functions $top(q) = \sigma(4q-2)$, $pop(q) = \sigma(4q-(2top(q)+1))$, and $nonempty(q) = \sigma(4q)$ can retrieve bit by bit the encoded word. The encoding is done by the functions $push_0(q) = \sigma(q/4+1/4)$ and $push_1(q) = \sigma(q/4+3/4)$. All these functions can be implemented with simple *ARNNs*. The two following propositions are proved in [23].

Proposition 5. *If $A \subseteq \{0,1\}^+$ is decidable (in the sense of Turing) in time t, then there exists a rational system $\mathcal{N}$ such that, for every word $w \in \{0,1\}^+$, the system classifies w in time $O(t(|w|)+|w|)$.*

We also have:

Proposition 6. *A set A is recognizable if and only if $A \in ARNN[\mathbb{Q}]$.*

Since the simulation of a Turing machine can be done in linear time and the other way around in polynomial time, the polynomial class P is preserved, i.e., denoting by $ARNN[\mathbb{Q}]P$ the class of sets decidable by rational nets in polynomial time, we have:

Proposition 7. *$P = ARNN[\mathbb{Q}]P$.*

Until now, we have been analysing deterministic neural net models. In the next section, we will define a non-deterministic version of these nets and explore their properties. We recall that our final goal is to exhibit relativisation results.

3. Non-determinism in the *ARNN* model

In this section, we will consider nets with rational or real weights, equipped with the saturated sigmoid. First, a definition of non-deterministic net:

Definition 5. A non-deterministic analog recurrent neural net (*NARNN*) $\mathcal{N}$ (of dimension n) consists of an analog neural net with three input units, receiving streams $V = 01^{|w|}0^\omega$, $U = 0w0^\omega$, and γ, a guess stream, with dynamics defined by

$$\vec{x}(t+1) = \sigma\left(A\vec{x}(t) + \vec{a}V(t) + \vec{b}U(t) + \vec{c}\gamma(t) + \vec{d}\right)$$

where $\vec{x}$ is the state vector of dimension n, A an $n \times n$ matrix, $\vec{a}$, $\vec{a}$, and $\vec{a}$ are vectors of dimension n.

Two special units are chosen for the output validation and the output, sending out streams $z = 0^{T_{\mathcal{N}}(|w|)-1}1^{|\phi(w)|}0^\omega$, $y(t) = 0$, for $t < T_{\mathcal{N}}(|w|)$, and $y(T_{\mathcal{N}}(|w|)-1+i) = (\phi(w))_i$, for $i = 1,\ldots,|\phi(w)|$, where $T_{\mathcal{N}}$ is the computation time and $\phi : \{0,1\}^+ \to \{0,1\}^+$ the function computed by $\mathcal{N}$.[10]

Note that if we impose a time bound $T_{\mathcal{N}}$, then only the first $T_{\mathcal{N}}(|w|)$ digits of the guess stream are needed. We can regard the guess stream (which only admits binary values as another input stream) as a path in the binary tree for possible sets of states of the given net. Each branch corresponds to a choice in $\{0,1\}$. The values that the γ unit takes decide the path along the computation such as a guess in a non-deterministic Turing machine.

A function ϕ computed by a *NARNN* receives as argument a word w. If the word w is in its domain, then value of ϕ can vary for different streams γ that lead to acceptance, so that ϕ is multi-valued: a partial function $\Phi : \{0,1\}^+ \times \{0,1\}^+ \to \{0,1\}^+$ can be defined, where $\Phi(w,\gamma_1)$ has value $\phi(w)$ given γ as guess stream, γ_1 being a prefix of γ. We are interested in a more restricted definition of *NARNN*, those that compute functions $\phi : \{0,1\}^+ \to \{0,1\}^+$, that is, given $w \in \{0,1\}^+$, there is a γ such that $\Phi(w,\gamma_1)$ is defined and, for all such γ, this value is the same whenever $\Phi(w,\gamma_1)$ is defined. In this case, we write $\phi(w) = \Phi(w,\gamma_1)$. Compare this restricted definition to the following:

Definition 6. A function ϕ is in *NPF* if it is computed in polynomial time by a non-deterministic Turing machine $\mathcal{M}$ in the following sense:

(a) $\mathcal{M}$ accepts the domain of ϕ in polynomial time;
(b) if $\langle x_1,\ldots,x_n\rangle \in dom(\phi)$, then any accepting computation writes in the output tape the value $\phi(x_1,\ldots,x_n)$ in polynomial time.

The notion of acceptance of a language is similar to the one for non-deterministic Turing machine:

[10] In particular, the characteristic function of a set.

Definition 7. A set A is said to be decided by a non-deterministic net $\mathcal{N}$ if (a) for all $w \in A$, there exists a guess stream γ, such that $\mathcal{N}$ accepts w (using some prefix γ_1 of γ) and (b) for all $w \notin A$, all computations reject no matter the guess γ.

This particular case coincides with the definition of a *NARNN* given in [23] more or less informally.[11] In the end of Chapter 4 in [21], it is written (1) *The above result suggests that a theory of computation similar to that of Turing machines is possible for our model of analogue computation.* (2) *Despite the very different powers of the two models, at the core of both theories is the question of whether the verification of solutions to problems is strictly faster than the process of solving them, or in other words whether P and NP are different.* (3) *While our model clearly does not provide an answer, it leads us to conjecture that it is quite likely that ARNN*$[\mathbb{R}]$*NP is strictly more powerful then ARNN*$[\mathbb{R}]$*P.*

We will see that the first statement above is not exactly true because deterministic and non-deterministic *ARNNs* necessarily relativise the relation $P = NP$ (see Section 8). The third statement points to the fact that, curiously, the author thinks that the relation $P = NP$ might be independent of the relation $ARNN[\mathbb{R}]NP = ARNN[\mathbb{R}]P$. We will see, in Section 8 that it is not.

The construction of the simulation of the non-deterministic Turing machine by a non-deterministic *ARNN* with rational coefficients is identical to that one of deterministic Turing machines in [23]. The simulation can be handled by injecting random bits to an auxiliary unit that is used by the control part of the dynamic system to make the option between the left or the right transition of the Turing machine being simulated (see the switch models in Section 2.4).

4. Polynomial resources

Let us assume in this section the special case of polynomial time computation via rational *NARNNs*. It is rather intuitive from [23] that these nets are equivalent to non-deterministic Turing machines clocked in polynomial time. We provide a short justification not found elsewhere.

Proposition 8. *If ϕ is a function computed by a rational NARNN clocked in polynomial time (denoted by $\phi \in ARNN[\mathbb{Q}]NPF$), then $\phi \in NPF$.*[12]

Proof. Let ϕ be a function of $ARNN[\mathbb{Q}]NPF$, computed by a *NARNN* $\mathcal{N}$, working in polynomial time $p_{\mathcal{N}}$, as in Definition 5.

Let $\mathcal{M}_{\mathcal{N}}$ be a deterministic Turing machine which simulates $\mathcal{N}$, according with Proposition 5, on input w and given the prefix γ_1 of guess stream γ, such that $|\gamma_1| = p_{\mathcal{N}}(|w|)$. We specify the following non-deterministic Turing machine $\mathcal{M}$:

Procedure:
Begin
input w;
guess z such that $|z| = p_{\mathcal{N}}(|w|)$;
simulate $\mathcal{M}_{\mathcal{N}}$ on $\langle w, z\rangle$;
if $\mathcal{M}_{\mathcal{N}}$ is in accepting state, output its result
End

Machine $\mathcal{M}$ witnesses the fact that $\phi \in NPF$, since polynomials are closed under addition, multiplication, and composition. □

Once proven the other inclusion, we will have the equivalence:

Proposition 9. $ARNN[\mathbb{Q}]NPF = NPF$.

The following definition and proposition provide an alternative definition of *NPF*, which consists on separating a non-deterministic Turing machine clocked in polynomial time in a polynomial long guess and a deterministic Turing machine clocked in polynomial time.

Definition 8. The class $\exists PF$ consists of functions $\phi : \{0,1\}^+ \to \{0,1\}^+$, such that there exist a function $\Phi : \{0,1\}^+ \times \{0,1\}^+ \to \{0,1\}^+$ in *PF* and a polynomial p with the properties:

(a) $w \in dom(\phi)$ if and only if there exists z such that $|z| \leqslant p(|w|)$ and $\langle w, z\rangle \in dom(\Phi)$;
(b) $\phi(w)$ is defined and its value is y if and only if there exists a z such that $|z| \leqslant p(|w|)$, $\langle w, z\rangle \in dom(\Phi)$, $\Phi(w, z)$ is defined and, for all such z, $\Phi(w, z) = y$.

[11] This model was not explored in the nineties.
[12] See Definition 6.

Proposition 10. *NPF* $= \exists PF$.

Proof. Let $\phi \in NPF$ witnessed by the non-deterministic Turing machine $\mathcal{M}$ clocked in polynomial time p. The following machine computes the function Φ with the properties of Definition 8:

Procedure:
Begin
input w and z;
simulate $\mathcal{M}(w)$ using z as guess;
if $\mathcal{M}(w)$ is led to the accepting state, then output $\phi(w)$,
else reject
End

Conversely, let $\phi \in \exists PF$ and $\Phi \in PF$ be witnessed by the Turing machine $\mathcal{M}$ clocked in polynomial time p. We specify a non-deterministic Turing machine clocked in polynomial time that computes ϕ:

Procedure:
Begin
input w;
guess z such that $|z| \leqslant p(|w|)$;
if $\mathcal{M}(w, z)$ is led to the acceptance state,
then output $\Phi(w, z)$
End

This ends the proof. □

As a corollary of Proposition 10, we have the well known result $NP = \exists P$, as expected, corresponding to the characteristic functions of sets: $A \in NP$ if and only if there is a set $B \in P$ and a polynomial p such that $w \in A$ if and only if there exists z, $|z| \leqslant p(|w|)$, such that $\langle w, z\rangle \in B$.

Now, we are ready to prove Theorem 9.

Proof. Suppose $\phi \in NPF = \exists PF$. Let $\mathcal{M}$ be the Turing machine that computes the corresponding function Φ. Note that $\mathcal{M}$ is a deterministic Turing machine clocked in polynomial time with two inputs. We can then simulate $\mathcal{M}$ in linear time by a deterministic neural net $\mathcal{N}$ with one pair input and input validation and a guess stream. In the computation of $\mathcal{N}$, no matter the guess, we end up always with same result. □

Hence, the class of languages accepted by non-deterministic rational nets is exactly *NP*. Joining this with the fact that deterministic neural nets with polynomial time bound decides exactly *P*, one is induced naturally to the positive relativisation of the hypothesis $P = NP$. The problem is now the lifting from rational to real valued weights.

5. Oracles and advices

By inserting an additional real weight in a rational *ARNN*, one obtains an effect similar to that of an oracle. We will introduce the notion of Davis' oracle that will allow to simulate such device by an oracle Turing machine and vice versa.

5.1. Oracles and advices in standard computation

The oracle Turing machine has also been seen as a way to reach the "uncomputable", called sometimes "hypercomputation" by some computer scientists such as in [9]. We emphasise this well known definition for we are about to consider a generalisation of it in Section 6.

Definition 9. An oracle Turing machine $\mathcal{M}$ is a Turing machine with a special tape called the query tape, three special states q_{query}, q_{yes}, and q_{no}, equipped with a set $\mathcal{O}$, called the oracle set, following the conditions:

1. When the machine is in the query state q_{query}, the machine interrupts its computation and in a single step verifies if the word in the query tape, say z, is in $\mathcal{O}$;
2. If $z \in \mathcal{O}$, then $\mathcal{M}$ makes a transition to q_{yes};
3. If $z \notin \mathcal{O}$, then $\mathcal{M}$ makes a transition to q_{no};
4. After the oracle's answer, $\mathcal{M}$ continues its computation.

The main idea of this oracle Turing machine is to enrich a standard Turing machine with a black box. This black box answers the membership question, that is, decides '$w \in \mathcal{O}$', in one step. Note that the oracle can be an arbitrary set, either decidable or not decidable.

We will also refer to the concept of advice.

Definition 10. Let $\mathcal{A}$ be a class of sets over the alphabet Σ and $\mathcal{F}$ a class of total functions of signature $\mathbb{N} \to \Sigma^*$. The non-uniform class $\mathcal{A}/\mathcal{F}$ is the class of sets B over Σ such that there exist $A \in \mathcal{A}$ and $f \in \mathcal{F}$ such that $w \in B$ if and only if $\langle w, f(|w|)\rangle \in A$. The function f is said to be an advice function.

$P/poly$ is the class of sets decidable by deterministic Turing machines working in polynomial time with advice of a size bounded by a polynomial. Note that f may not be computable. A similar example is P/log.

We will also need the following concept:

Definition 11. We denote by $\mathcal{F}\star$ the set of prefix functions with size limited by a function in $\mathcal{F}$. That is, if $f \in \mathcal{F}\star$, then for all n, $f(n)$ is a prefix of $f(n+1)$ and $|f(n)| \leqslant g(n)$ for some function $g \in \mathcal{F}$.

5.2. Oracles in non-standard computation

Families of circuits of polynomial size have been known to be in correspondence with sets in $P/poly$ (see [1], Chapter 5). In the article [22], it is proved that each circuit family can be simulated by a rational *ARNN* with an appropriate real weight.

Definition 12. A circuit is a directed acyclic graph, where nodes of in-degree 0 are called input nodes and the others gates, labelled by one of the Boolean functions *AND*, *OR*, or *NOT*, computing the corresponding function. The first two types are of many variables and the third is a unary function. A special node with no outgoing edge is designated as the output node. The size of a circuit is the total number of gates. The circuit is given by levels $0, \ldots, d$ so that the input nodes are in level 0, the output node in level d, and each level has gates only receiving input from gates of the previous level. The depth is then d. A family of circuits is a set of circuits $\{c_n\colon\ n \in \mathbb{N}\}$.

Proposition 11.

(a) *There is an injective encoding from the set of families of circuits to the* 9*-Cantor subset of* $[0, 1]$;
(b) *if r encodes a family $(c_k)_{k\in\mathbb{N}}$ of polynomial size circuits, then the code of the nth circuit can be found among the first $p(n)$ digits of the decimal expansion of r, where p is a polynomial depending on $(c_k)_{k\in\mathbb{N}}$, and furthermore*
(c) *we can construct an ARNN — call it $\mathcal{N}_r$ — with weights in $\mathbb{Q} \cup \{r\}$ to extract in polynomial time, given input w of size n, the code of c_n.*

The proof of this result relies on a subsystem, herein called *BAM*, which recovers, bit by bit, such encodings (see Section 5.3 for a proof). This shows that the class of languages decided by rational *ARNNs* working in polynomial time with one real weight is as powerful as $P/poly$.

Proposition 12. *$P/poly \subseteq ARNN[\mathbb{R}]P$, where $ARNN[\mathbb{R}]P$ denotes the class of sets decided by neural nets with real weights clocked in polynomial time.*

Proof. Suppose A is a set in $P/poly$ and let w be a word of size n. From Proposition 11, we can build $\mathcal{N}_r$, having a real weight r coding for the circuit family $\mathcal{A}$, that extracts the code c_n of the circuit deciding $A_n = A \cap \{0, 1\}^n$. Feeding $\langle w, c_n\rangle$ into $\mathcal{N}_{CVP}$, a rational *ARNN* that simulates the Turing machine deciding the set *CVP*[13] in polynomial time, we can decide in polynomial time if $w \in A$. □

In fact, *ARNNs* working in polynomial time with real weights decide exactly the sets in $P/poly$. (Moreover, real *ARNNs* working in exponential time can decide any set!) And here arises the claim of hypercomputational power of recurrent neural nets made in [19]: *If we can implement such a real weight in a neural net*, a computer with hypercomputational properties can be built, exceeding the computational power of Turing machines! Others proposed the use of physical constructs as oracles to access, in a natural way, real numbers encoded in the Universe. That the impossibility of achieving such devices does not derive from the necessity of adjustment of physical parameters with infinite precision is addressed in [15]. However, despite the advocated impossibility, this misconception can be refuted by the following result:

[13] Circuit Value Problem.

Table 1
Computational power of *ARNN* under various restrictions.

Set of weights	Time restriction	Computational power
$\mathbb{Z}$	none	Regular languages
$\mathbb{Q}$	none	Recursive languages
$\mathbb{Q}$	t	$DTIME(t)$
$\mathbb{R}$	polynomial	$P/poly$
$\mathbb{R}$	none	All languages

Proposition 13. *The output of an ARNN after t steps is affected only by the first $O(t)$ digits in the expansion of the weights.*

From the proof of this proposition (see [22]), it was showed that one can simulate a real weighted *ARNN* working in time t by a circuit of size $O(t^3)$. As a corollary, we can fully classify $ARNN[\mathbb{R}]P$ and $ARNN[\mathbb{R}]$ in the two following propositions.

Proposition 14. $ARNN[\mathbb{R}]P = P/poly$.

And, more trivially, we get:

Proposition 15. *For any binary language there is a neural network with real weights which decides it. Conversely, an exponential time restriction is sufficient for real nets to decide any set. Summing up, $ARNN[\mathbb{R}]EXP = ARNN[\mathbb{R}] = \mathcal{P}(\{0,1\}^+)$.*

The last result is a consequence of the fact that, to decide a given language, it is enough a family of circuits of exponential size. This uniform classification of languages decided by *ARNNs* (by changing the type of the weights) is resumed as in Table 1.

In [10], Professor Davis criticised the point of view expressed in [19], and also other attempts of the hypercomputation claims: *Since the non-computability that Siegelmann gets from her neural nets is nothing more than the non-computability she has built into them, it is difficult to see in what sense she can claim to have gone "beyond the Turing limit".*

As real numbers can "boost up" the computational power of rational *ARNN*, we will present real numbers in the remaining of this work as oracles and show that they are a restricted class of oracles — we will refer to them as *Davis' oracles*.

To simulate a neural net with real weights, one real weight suffices (e.g., resulting from the shuffling of the expansions of a finite number of reals). This will imply that a finite number of Davis' oracles can be reduced just to one. Therefore, it is enough to prove relativisation results in *ARNN* for nets with only one Davis' oracle, that is, only one real weight.

5.3. BAM and the prefix retrieval process

In linear time, the prefix of r of length n can be extracted by running an appropriate *ARNN*. In [22], it is presented for the first time a subsystem that simulates this procedure to retrieve the prefix of a given weight encoded in the 9-Cantor set.

Let $\mathcal{C}_b$ be the b-Cantor subset of $[0,1]$ and $r \in \mathcal{C}_b$ a real number with digits in $\{0,2,\ldots,b-l\}$, where l is 1 if b is odd and 2 if b is even.[14] Let us assume that b is odd. To extract the digits of r, we can first compare br with $k \in \{0,1,\ldots,b-1\}$ through the family of functions[15] $\Lambda_k(r) = \sigma(br-k)$ and then shift the encoding of r one digit to the left by the shift map $\Xi(r) = \sigma(\sum_{k=0}^{b-1}(-1)^k\Lambda_k(r))$. When the word is not trivial, for at most one k even, the value of $\Lambda_k(r)$ is in $(0,1)$. If such k exists, for $0 \leqslant j < k$, we have $\Lambda_j(r) = 1$ and, for $j > k$, $\Lambda_j(r) = 0$. The following map recovers the prefix of r along the extraction procedure and saves it in the reverse order as $\tilde{r}$:

$$\Psi(r,\tilde{r}) = \sigma\left(\tilde{r}/b + 2/b\sum_{j=0}^{\frac{b-1}{2}}\Lambda_{2j}(r)\right) \tag{1}$$

[14] A "good" encoding is one isomorphic to a subset of a b-Cantor encoding.

[15] In fact, for b even, verifying this property for $k \in \{0,1,\ldots,b-2\}$ is enough. For reasons of uniformity of notation, we can include $b-1$, since $\Lambda_{b-1}(r) = 0$ for all r.

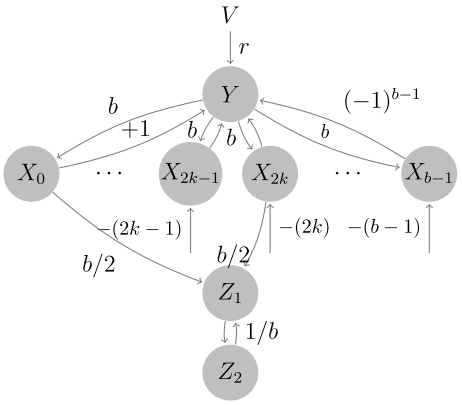

Fig. 1. Architecture of subnet *BAM*. The arrows without labels have weights alternating between -1 and 1; arrows without origin nodes denote the bias; finally, the input V injects the code r into the unit Y.

This dynamical system can be simulated by a four-layered net:

$$\begin{cases} y^+ = \sigma\left(V + \sum_{k=0}^{b-1} (-1)^k x_k\right) \\ x_k^+ = \sigma(by - k) \qquad\qquad \text{for } k \in \{0, \ldots, b-1\} \\ z_1^+ = \sigma\left(z_2/b + 2/b \sum_{j=0}^{\frac{b-1}{2}} x_{2j}\right) \\ z_2^+ = \sigma(z_1) \end{cases} \tag{2}$$

where V is the unit that uploads the encoding of r into the *BAM* (see Fig. 1). Once V sends r, the units x_k extracts one digit of r at a time, while z_1 keeps $\tilde{r}$.[16] By joining a clock to this subsystem (and doing some more work on synchronization), we can control the number of digits to be extracted. By choosing a 9-Cantor encoding, we have the *BAM* as in [22].

Let us prove Proposition 11.

Proof. (a) Let c_k be one of the circuits from the family $C = (c_k)_{k\in\mathbb{N}}$. We enumerate the gates by layer and we fix an order, say g_{ij} is the jth gate of level i. The encoding of the c_k, denoted by $\widehat{c}_k$, is performed as follows.

The encoding of a level starts with 6. Each level is encoded successively from bottom to the top level. In each level i, the gates are encoded by order, each g_{ij} starting with 0, followed by the code of the gate

$$\begin{aligned} \mathit{AND} &\mapsto 42 \\ \mathit{OR} &\mapsto 44 \\ \mathit{NOT} &\mapsto 22 \end{aligned}$$

and then, by order, 4 if a gate $g_{i-1,j'}$ from the previous level feeds into g_{ij}, and 2 otherwise.

We denote by $\overline{w}$ the reverse of $\widehat{w}$. The encoding of a circuit family C, denoted by $\widehat{C}$, is given by $\widehat{C} = 8\bar{c}_1 8\bar{c}_2 8\bar{c}_3 \ldots$. This infinite sequence can then be finally encoded into r, the corresponding real number in the 9-Cantor set $\mathcal{C}_9$.

(b) Follows from the fact that the circuits of the family $(c_k)_{k\in\mathbb{N}}$ are of polynomial size.

(c) We have two input streams, $A = 0w0^\omega$ and $V = 01^{|w|}0^\omega$. The validation line V feeds into the system the size of the input in unary. The unit $x^+_{|w|} = \sigma(\frac{1}{2}x_{|w|} + \frac{3}{2}V - 1)$ encodes $|w|$ into a rational number $q_{|w|}$. This value is saved in the unit x_q:

$$\begin{cases} x^+_{\mathit{delay}} = \sigma(x_{|w|}) \\ x^+_q = \sigma(x_{\mathit{delay}} - V) \end{cases}$$

[16] Unit z_2 is only for synchronization purposes.

We then build the net $\mathcal{N}$, the *BAM* system to extract r. Each time $\mathcal{N}$ encounters an 8 in r, it subtracts 1 from $|w|$, that is, $2q_{|w|}-1$. When $q_{|w|}=0$, $\mathcal{N}$ extracts the digits of r until the next 8, outputting the code sequence of the corresponding circuit in the correct order. The dynamics is as follows (from [22]):

$$\begin{cases} x_0^+ = \sigma(9x_{10}) \\ x_1^+ = \sigma(9x_{10}-1) \\ x_2^+ = \sigma(9x_{10}-2) \\ x_3^+ = \sigma(9x_{10}-3) \\ x_4^+ = \sigma(9x_{10}-4) \\ x_5^+ = \sigma(9x_{10}-5) \\ x_6^+ = \sigma(9x_{10}-6) \\ x_7^+ = \sigma(9x_{10}-7) \\ x_8^+ = \sigma(9x_{10}-8) \\ x_9^+ = \sigma(2A) \\ x_{10}^+ = \sigma(rx_9 + x_0 - x_1 + x_2 - x_3 + x_4 - x_5 + x_6 - x_7 + x_8) \end{cases}$$

To count the number of 8s encountered we add 3 more units:

$$\begin{cases} x_{13}^+ = \sigma(A + x_{15} + x_{14}) \\ x_{14}^+ = \sigma(x_7 + 2x_{13} - 2) \\ x_{15}^+ = \sigma(-x_7 + x_{13}) \end{cases}$$

where the unit x_{13} holds initially the value $(|w|)|_2$ and its value is decremented each time a 8 is detected in unit x_{14}. When the value held by x_{13} reaches 0, the circuit has been downloaded. The encoding of the circuit is done by the dynamics:

$$\begin{cases} x_{11}^+ = \sigma(2/9x_1 + 2/9x_3 + 2/9x_5 + 2/9x_7 + 1/9x_{12}) \\ x_{12}^+ = \sigma(x_{11}) \end{cases}$$

The result is provided by the state component $x_{16}^+ = \sigma(x_7 + x_{12} - 1)$. The units $x_0, \ldots, x_{12}$ constitute the *BAM*. The unit x_{11} keeps the key real number r. The control of retrieval is done by the units $x_{13}, \ldots, x_{16}$ (note that in the dynamic map of x_{14}, $2x_{13}-1$ counts $|w|$ downwards each time $x_7 = 1$, working as a switch). □

The y in Eq. (2) can be regarded as the *query unit* and z_1 (or z_2) as the *answer unit*. The time of extraction is linear on the numbers of digits required since each digit can be obtained in constant time. We say that this "oracle" is of *linear access time or linear cost*. In the next sections, we will show how to simulate a rational net with one real weight by a Turing machine with a special type of oracle and then define access time protocols for oracles.

5.4. Davis' oracles

The embedding of real numbers in machines or in neural nets has been seen as a path to achieve "non-computability". In this subsection we provide a way of regarding real numbers as oracles in the context of the *ARNN* model.

As stated in Proposition 13, only the first digits of the expansion of the weights are needed in a computation. The same output can be computed by replacing the weights by their first $O(t)$ bits if the computation can be achieved in time t, i.e., the computations performed by *ARNNs* equipped with real weights can only decide sets by means of (finite) prefixes of the expansion of the weights in some base. This gives us the intuition that the oracles embedded in *ARNNs* are a restricted class of (tally) oracles.

Definition 13. The Davis' oracle $\mathcal{O}_r$, for some $r \in \{0,1\}^\omega$, is the set *Prefix*(r) of the prefixes of r.[17] (The first n digits of r will be denoted $r|_n$.)

We will also call Davis' oracles to sets of the kind $\mathcal{O}_{r,f} = \{\langle 0^n, \tilde{r}\rangle\colon\ n \in \mathbb{N},\ \tilde{r} \text{ is a prefix of } r|_{f(n)}\}$, for some time constructible function $f : \mathbb{N} \to \mathbb{N}$. These two definitions are equivalent when implemented in a Turing machine without time restrictions.

[17] The stream r should be regarded as a real number in $[0,1]$.

Proposition 16.

(a) *A net $\mathcal{N}$ with weights in $\mathbb{Q} \cup \{r\}$, $r \in [0, 1]$, clocked in constructible time $T_{\mathcal{N}}(n)$, where n is the size of the input, can be simulated by a Turing machine with a Davis' oracle $\mathcal{O}_r$ in time $O(T_{\mathcal{N}}(n)^k)$, for some positive integer k;*
(b) *A Turing machine $\mathcal{M}$ with a Davis' oracle $\mathcal{O}_r$ [$\mathcal{O}_{r,f}$] working in constructible time $T_{\mathcal{N}}(n)$, where n is the size of the input, can be simulated by a neural net $\mathcal{N}$ with weights in $\mathbb{Q} \cup \{r\}$, $r \in [0, 1]$, working in time $O(T_{\mathcal{N}}(n))$ [$O(f(n) + T_{\mathcal{N}}(n))$].*

Proof. (a) Let $\mathcal{O} = \{\langle 0^n, \tilde{r}\rangle\colon\ n \in \mathbb{N},\ \tilde{r} \text{ is a prefix of } r|_{T_{\mathcal{N}}(n)}\}$ be the required Davis' oracle for the simulation of $\mathcal{N}$. Let $cT_{\mathcal{N}}(n)$ be the number of digits of r needed for the computation on words of size n. The following Turing machine can simulate $\mathcal{N}$ with at most a delay polynomial in $T_{\mathcal{N}}(n)$:

Procedure:
Begin
input w;
$n := cT_{\mathcal{N}}(|w|)$;
$\tilde{r} := \lambda$;
for $i := 1$ to n do
if $\langle 0^n, \tilde{r}0\rangle \in \mathcal{O}$, then $\tilde{r} := \tilde{r}0$,
else if $\langle 0^n, \tilde{r}1\rangle \in O$, then $\tilde{r} := \tilde{r}1$,
else exit for
end for;
simulate $\mathcal{N}$ replacing r by $\tilde{r}$ with input w;
output the result of $\mathcal{N}$
End

(b) This statement can be proved by separating oracle calls of $\mathcal{M}$ from the other computations. Since $\mathcal{M}$ works in time $T_{\mathcal{M}}$, for an input of size n, $T_{\mathcal{M}}(n)$ is an upper bound of the size of the query words. Let w be an input of size n, $\tilde{r}$ be the first $T_{\mathcal{M}}(n)$ bits of r, and $\tilde{\mathcal{M}}$ the Turing machine that receives $\langle z, \tilde{r}\rangle$ as input and simulates the computations of $\mathcal{M}$, replacing the oracle calls by the following procedure:

Procedure:
Begin
input query word z;
if z is a prefix of $\tilde{r}$, then switch to state q_{yes},
else switch to state q_{no}
End

Let $\mathcal{N}_2$ be an *ARNN* that simulates $\tilde{\mathcal{M}}$. Then we construct a net $\mathcal{N}_1$ receiving information from the input stream $(0w0^\omega)$ and the input validation stream $(01^{|w|}0^\omega)$: it consists on a binary *BAM* for the real $r = \lim \mathcal{O}_r$ (in binary) plus a counter that counts $T_{\mathcal{M}}(n)$ *BAM* steps (by linear time simulation of the Turing machine that witnesses the time constructibility of $T_{\mathcal{M}}$), to extract $\tilde{r}$, and a unit that saves the value of w; this net $\mathcal{N}_1$ exports w together with $\tilde{r}$. The sequential composition of $\mathcal{N}_2$ after $\mathcal{N}_1$, provided by Proposition 2 produces a rational net $\mathcal{N}$ with a real-valued weight r, working in time $O(T_{\mathcal{M}}(n) + 1 + T_{\mathcal{M}}(n))$, that is, $O(T_{\mathcal{M}}(n))$.

The case of $\mathcal{O}_{r,f}$ can be proved similarly, by changing the number of bits to be retrieved by the *BAM* to $f(n)$. This net will work in time $O(f(n) + T_{\mathcal{M}}(n))$. □

The proof of Proposition 16(b) is essential for the last sections of this paper. To sum up: the *BAM* system "queries" the oracle and implements the oracle consultation process of a Turing machine. On the other hand, the oracle Turing machine $\mathcal{M}$ composed of two parts: (a) one performing all the oracle calls first for an arbitrary input w and (b) $\tilde{\mathcal{M}}$ such that, upon receiving a prefix of a real number and the input w, behaves like $\mathcal{M}$ with the oracle calls replaced by comparisons, can be simulated by a neural net with two interconnected subsystems: (a) $\mathcal{N}_1$ that extracts a prefix of a real number (a *BAM*) and (b) $\mathcal{N}_2$ that simulates the original Turing machine with the oracle consultation process in real time.

Proposition 17. *For all $r \in [0, 1]$, $ARNN[\mathbb{Q} \cup \{r\}]P = P(\mathcal{O}_r)$.*

We will be writing $ARNN[\mathbb{Q}]P(r)$ instead of $ARNN[\mathbb{Q} \cup \{r\}]P$ when we want to emphasise that r is to be seen as an oracle. By induction, we can conclude the following:

Proposition 18. $ARNN[\mathbb{Q}]P(r_1)(r_2)\ldots(r_n) = P(\mathcal{O}_{r_1})(\mathcal{O}_{r_2})\ldots(\mathcal{O}_{r_n})$.

Proposition 19. *The following classes of sets coincide*:

1. $P/poly$
2. $\bigcup_{sparse\ S} P(S)$
3. $\bigcup_{tally\ T} P(T)$
4. $\bigcup_{r\in[0,1]} ARNN[\mathbb{Q}]P(r)$
5. $\bigcup_{r\in[0,1]} ARNN[\mathbb{Q}\cup\{r\}]P$
6. $\bigcup_{Davis'\ oracle\ \mathcal{O}} P(\mathcal{O})$

This is also true for the corresponding non-deterministic classes.

Proof. (1) = (2) = (3) is known from structural complexity (see [1], Chapter 5). Proposition 14 states that (5) = (1) and, from the last proposition, we have that (4) = (6). (1) $\subseteq$ (4) is guaranteed by Proposition 11 and, by definition, (4) = (5). □

6. Time protocols

The working time of a standard oracle is constant by definition. This means that, when it is called, no matter how long a query word is, the time taken to answer the query is the same, which is an unrealistic view when a physical process takes the place of an oracle. Certainly, we can "solve" the problem by replacing every call of a given oracle $\mathcal{O}$ by the following subsystem:

Procedure:
Begin
input w;
count $T(|w|)-1$ steps;
if $w\in\mathcal{O}$, then 'yes' else 'no'
End

where w is the query word and T a time constructible function. Instead, we include "naturally" an internal clock into the oracle system that works in the required time units.

Definition 14. An oracle $\mathcal{O}$ works in time T when, in each call, the oracle costs $T(|z|)$ time steps to answer to the query '$z\in\mathcal{O}$'.[18]

When T is a polynomial or an exponential, we say that the oracle has a *polynomial* or an *exponential* cost, respectively. We will be interested in Davis' oracles of polynomial and exponential cost. When no protocol is referred, the answers to queries are done in one time step.

The work on physical oracles to take into account the time taken by the physical process is introduced in [4,6,5,8,7,3].

7. Generalised nets

The retrieval process works in linear time by means of the use of the saturated sigmoid. When other activation functions are adopted, the time to retrieve the digits from the weight might well be longer.

Definition 15. A neural net is said to be a generalised net (of dimension n with m input lines) if the activation function is given by $f=\sigma\circ\pi$ where $\pi:\mathbb{R}^{n+m}\to\mathbb{R}^n$ is an affine map and $\sigma:\mathbb{R}^n\to\mathbb{R}^n$, called the activation function, has a bounded range and is locally Lipschitz.[19] The output units are chosen within the n processors and two decision thresholds $\alpha<\beta$ are set to be interpreted as 0 if an output is less than or equal to α and 1 if an output is greater than or equal to β.

Definition 16. A function $f:\mathbb{R}\to\mathbb{R}$ is s-approximable in time t_s if there is a Turing machine that computes $f|_{s(n)}$ in time $t_s(n)$ given an input of size n. When the function s is not explicitly mentioned, we are considering $s(n)=n$.

Let us consider now a neural net with the architecture of Proposition 16, replacing the activation function σ of the critical unit holding the real weight r by g, having the right inverse g approximable in constructible time t, giving rise to a generalised processor net $\mathcal{D}$. Simulating the Turing machine that computes g in order to retrieve from $f(r)$ the first n

[18] T may not be time constructible.
[19] That is, for each $\rho>0$, there is a constant K such that, for all x_1 and x_2 in the domain of σ, if $|x_2-x_1|<\rho$, then $|\sigma(x_2)-\sigma(x_1)|\leqslant K|x_2-x_1|$, where $|\cdot|$ is the Euclidean norm.

bits of r costs $t(n)$ steps. Suppose that the value $f(r)$ is given for a function f with right inverse, say g, approximable in time t. The cost of retrieving the first k bits of $f(r)$ (by the *BAM*) is $O(k)$, but the cost of retrieving n bits of $r = f(g(r))$ is $O(t(n))$.[20] This is a Davis' oracle with access time $O(t(n))$. This oracle can be easily implemented in an *ARNN*, done as follows: (a) a subsystem simulates $\mathcal{M}_g$ in real time and (b) a *BAM* subsystem extracts the bits of $f(r)$. It costs $O(t(n))$ to compute the first n bits of r. Languages decided by these nets will be denoted by $ARNN[\mathbb{Q}](f, r)$.

Definition 17. A generalised Davis' oracle is an oracle $\mathcal{O}_{f,r} \subseteq Prefix(f(r))$, where $r \in [0, 1]$ and f is a function with a right inverse approximable in some time t.[21]

The trivial case is the saturated sigmoid. For $x \in [0, 1]$, we have that $\sigma(x) = x$. In fact, $\mathcal{O}_{\sigma,r} = \mathcal{O}_r$ is just a Davis' oracle with linear cost. Another example is the family of analytic sigmoids

$$\sigma_{a,k}(x) = \frac{1}{1 + e^{-kx}}.$$

The computation of the inverse function takes exponential time. That is $\mathcal{O}_{\sigma_{a,k},r}$ is a Davis' oracle with exponential cost.

The generalised oracles do not fit into the definition of a standard oracle. As stated before, an internal clock is *naturally* implemented into them, so we can control the time taken to answer a given query. By choosing a function f approximable in polynomial or exponential time, we obtain a Davis' oracle with polynomial or exponential cost, respectively, and also their simulations in *ARNNs*.

8. Relativisation in *ARNNs*. I. Polynomial cost

We prove in this section the positive relativisation result for the rational *ARNNs* equipped with one real weight.[22]

The deterministic or non-deterministic *ARNN* $\mathcal{N}$ has access to the oracle only by performing the query in constant time — that we can take as a few computation time steps — as seen in Section 5.3. It means that to read n bits of the oracle r, no more than a polynomial number of time steps is needed. Let us suppose that p is the polynomial bounding the time of non-deterministic system $\mathcal{N}$. Then $\mathcal{N}$ can query an exponential number of words ($2^{O(p(|w|))}$) to the physical oracle, for input word w. However, that is equivalent to query just a polynomial number of words in the totality of the branches of the non-deterministic branching tree of computations, as we will prove in the propositions that follows.

The results arise naturally, since no restrictions are put on the underlying dynamic system. As we see, the number of queries are not limited by polynomial bounds, but only a polynomial number of queries are effective in determining the membership of a word to a set. This is the reason why we consider the *ARNN* model as a *natural model of machine that can effectively query in polynomial time no more than a polynomial number of words to its oracle*.

Definition 18. Let $Q(\mathcal{M}, w, \mathcal{O})$ denote the set of queries produced by the machine $\mathcal{M}$ with oracle $\mathcal{O}$ when the input word is w. If the system is non-deterministic, then the set $Q(\mathcal{M}, w, \mathcal{O})$ is the union of sets of queries referring to all the computations of the branching tree.

Although trivial, the following proposition helps to understand the limitations of Davis' oracles, now seen as sparse oracles to Turing machines:

Proposition 20. *Let A be a set decided by a non-deterministic Turing machine $\mathcal{M}$ clocked in polynomial time p, equipped with a Davis' oracle $\mathcal{O}$ of polynomial time access. Then, there is an equivalent non-deterministic Turing machine $\tilde{\mathcal{M}}$, clocked in polynomial time, querying the same oracle at most a polynomial number of times in a computation tree. In fact, $|Q(\tilde{\mathcal{M}}, w, \mathcal{O})| \leqslant p(|w|)$.*

Proof. Suppose without loss of generality that the query language is binary. Let $\mathcal{O} = Prefix(r)$, for an $r \in \{0, 1\}^\omega$, where the extraction of the nth bit takes time $q(n)$. Since $\mathcal{M}$ is working in polynomial time p, given an input w, the size of the queries does not exceed $p(|w|)$. Thus, only a polynomial number of bits of r are needed. Let $\mathcal{M}_1$ be the machine that extracts the first $p(|w|) + 1$ bits of r given the input w:

Procedure:
Begin
input w;

[20] Note that r is a weight of the net. In a first step, the net activates a unit, sending $r \times 1$ to another unit which evaluates in a single step $f(r)$, where f is the activation function. Thus, retrieving n bits of r corresponds to evaluate the first n bits of $g(f(r))$.

[21] The cost of the oracle is t.

[22] As described before, the same result applies to a finite number of real-valued weights. Since *ARNN* systems are finite, the result applies to arbitrary *ARNNs*.

$\tilde{s} := \lambda$;
for $i := 1$ to $p(|w|) + 1$ do
if $\tilde{s}0 \in \mathcal{O}$, then $\tilde{s} := \tilde{s}0$ else if $\tilde{s}1 \in \mathcal{O}$, then $\tilde{s} := \tilde{s}1$
return $\tilde{s}$
End

This subsystem takes polynomial time to perform the computations.[23] Let $\mathcal{M}_2$ be the Turing machine with the same description as $\mathcal{M}$, replacing each oracle call by the following procedure:

Procedure:
Begin
input z;
for $i := 1$ to $|z|$ do if $z_i \neq r_i$, then proceed to 'no';
proceed to 'yes'
End

where z is the query word. The machine $\mathcal{M}_2$ works in polynomial time and so does $\tilde{\mathcal{M}}$ which is the sequential composition of $\mathcal{M}_2$ after $\mathcal{M}_1$. The total number of oracle calls is indeed polynomial in the size of the input. □

Note that it is trivial that $P(\mathcal{O}_r) = \bigcup_f P(\mathcal{O}_{f,r})$, when f has a right inverse approximable in polynomial time (implying $P/poly = \bigcup_{\text{p.c.}\mathcal{O}_r} P(\mathcal{O}_r)$, where p.c. abbreviates polynomial cost and $\mathcal{O}_r$ is a Davis' oracle). Note that $\mathcal{M}$ results from a machine that exclusively does the oracle calls, $\mathcal{M}_1$, and a machine that does not query the oracle, $\mathcal{M}_2$. This was used in the proof of Proposition 16. By simulating $\mathcal{M}_1$ with a *BAM* and $\mathcal{M}_2$ by the linear time simulation of Turing machines by neural nets, we have the following corollary, resulting from Propositions 16 and 20:

Proposition 21. *Let A be a set decided by a non-deterministic rational neural net $\mathcal{N}$, with a real weight r, clocked in polynomial time p. Then, there is an equivalent non-deterministic neural net $\tilde{\mathcal{N}}$, clocked in polynomial time, querying r at most a polynomial number of times. In fact, $|Q(\tilde{\mathcal{N}}, w, r)| \leqslant p(|w|)$.*

To proceed to analyse the combined power of non-determinism and the oracles, we need to introduce more concepts. In the non-deterministic branching computation tree of depth t, each computation, of size t, corresponds to a binary word of size t.

Definition 19. Let $\mathcal{K}$ be the set of tuples of the form $\langle \mathcal{N}, w, 1^t, \tilde{r} \rangle$, where $\mathcal{N}$ is a non-deterministic *ARNN* system having as oracle (*BAM*) the dyadic rational $\tilde{r}$ that accepts w in at most t steps.[24]

We know that the set $\mathcal{K}$ is decidable, since the *ARNN* with rational weights is simulable by a Turing machine in polynomial time by Proposition 5. But we prove a stronger result, namely:

Proposition 22. $\mathcal{K} \in NP$.

Proof. Consider the following non-deterministic Turing machine $\tilde{\mathcal{M}}$:

Procedure:
Begin
(1) input $\Sigma = \langle \mathcal{N}, w, 1^t, \tilde{r} \rangle$;
(2) check if $\mathcal{N}$ encodes a non-deterministic
ARNN description;
(3) guess $z \in \{0,1\}^t$;
(4) simulate Σ on w guided by $z0^\omega$ as input random stream
to $\mathcal{N}$, with oracle value $\tilde{r}0^\omega$;
(5) if, given that the validation output stream outputs 1,
the decision is *accept*, then *accept* else *reject*
End

[23] This is indeed a Turing machine simulation of the bit extraction process done by the *BAM* subsystem of an *ARNN*.
[24] Note that, even in the case that $\mathcal{N}$ does not behave accordingly to the definition of a *NARNN*, provided after Definition 3, it does not affect the use of this definition in Proposition 23.

We now prove that the non-deterministic Turing machine $\tilde{M}$ can be clocked in polynomial time. The system $\mathcal{N}$ has to provide an answer in time t. In time t, each unit of $\mathcal{N}$, decoded in the appropriate base, can be rewritten t times during the computation guided by z, task that can be achieved in time polynomial in t, i.e., polynomial in the size of the input ($O(|\mathcal{N}| + |w| + t + |\tilde{r}|)$).

To conclude the proof we have to discuss some detail about (4) in the above algorithm. When the machine queries the oracle with query z, $\tilde{\mathcal{M}}$ runs the following deterministic procedure[25]:

Procedure:
Begin
input z;
for $i := 1$ to $|z|$
if $z_i > \tilde{r}_i$ then return *no*
else if $z_i < \tilde{r}_i$ then return *yes*;
return *yes*
End

We conclude that $\mathcal{K} \in NP$. □

Proposition 23 *(Positive relativisation* PART I*). $P = NP$ iff, for all oracle $r \in [0, 1]$, $ARNN[\mathbb{Q} \cup \{r\}]P = ARNN[\mathbb{Q} \cup \{r\}]NP$.*

Proof. Assume that, for every oracle r, we have $ARNN[\mathbb{Q} \cup \{r\}]P = ARNN[\mathbb{Q} \cup \{r\}]NP$. As particular case, $ARNN[\mathbb{Q}]P(\frac{1}{3})$ contains all sets decidable by deterministic *ARNN* systems clocked in polynomial time that do not take profit of any oracle, that is all sets decidable by deterministic Turing machines clocked in polynomial time, that is P; on the other side, $ARNN[\mathbb{Q}]NP(\frac{1}{3})$ contains all sets decidable by non-deterministic *ARNN* systems clocked in polynomial time that do not take profit of any oracle, that is all sets decidable by non-deterministic Turing machines clocked in polynomial time, that is NP. Thus $P = NP$.

For the converse, it is clear that $ARNN[\mathbb{Q} \cup \{r\}]P \subseteq ARNN[\mathbb{Q} \cup \{r\}]NP$. We will prove that $P = NP$ implies $ARNN[\mathbb{Q} \cup \{r\}]NP \subseteq ARNN[\mathbb{Q} \cup \{r\}]P$.

Let A be a set decided by a non-deterministic *ARNN* $\mathcal{N}$ with an oracle r clocked in polynomial time p, and consider the following algorithm:

Procedure:
Begin
input w;
if $\Sigma = \langle \mathcal{N}, w, 1^{p(|w|)}, r|_{cp(|w|)} \rangle \in \mathcal{K}$,
then *accept* else *reject*
End

A dynamical system that reads $\tilde{r} = r|_{cp(|w|)}$, for some constant c, can be executed in linear time and, according with Proposition 13, $\tilde{r}$ contains the bits needed for Σ to complete its computation; if $P = NP$ (by hypothesis), the test of membership to $\mathcal{K}$ can be decided in deterministic polynomial time too. This algorithm accepts the same set as the original non-deterministic *ARNN* system with oracle r, clocked in polynomial time, as we wished to show. Then we can simulate this procedure by an *ARNN* system in linear time and connect it with the previous system which reads the bits of $\tilde{r}$, obtaining a composite system that solves deterministically the decision process for A in polynomial time. □

By repeating the proof for particular restrictions we have:

Proposition 24. *$P = NP$ if and only if, for any $r \in [0, 1]$ and for any f having a right inverse approximable in polynomial time, $P(\mathcal{O}_{f,r}) = NP(\mathcal{O}_{f,r})$.*

As particular cases we have what was wanted:

Proposition 25. *$P = NP$ if and only if, for any $r \in [0, 1]$, we have*

$$ARNN[\mathbb{Q} \cup \{r\}]P = ARNN[\mathbb{Q} \cup \{r\}]NP.$$

Note that the left-hand side of the equivalence of all these results claims that, not only the whole class will collapse, but also each deterministic subclass will coincide with its non-deterministic counterpart. The difference from the trivial result $P = NP$ if and only if $P/poly = NP/poly$ should be emphasised.

[25] The word $\tilde{r}$ is padded with 0s to the right up to $|w| + 1$ if $|w| \geqslant |\tilde{r}|$.

Proposition 26. *The following propositions are equivalent*:

1. $P = NP$
2. $ARNN[\mathbb{Q}]P = ARNN[\mathbb{Q}]NP$
3. $ARNN[\mathbb{R}]P = ARNN[\mathbb{R}]NP$

9. Relativisation in *ARNNs*. II. Exponential cost

The analytic oracles are members of a larger class of oracles: the Davis' oracles with exponential cost. As in the polynomial case, we have the following results that can be proved using arguments similar to the proof of Proposition 20.

Proposition 27. *Let A be a set decided by a non-deterministic Turing machine M clocked in polynomial time equipped with a Davis' oracle with exponential cost. Then, there is an equivalent non-deterministic Turing machine $\tilde{M}$, clocked in polynomial time, querying the same oracle in a way such that the total number of calls in all branches of a computation tree is logarithmic in the size of the input.*

Let $P_\ell(\mathcal{O})$ the class of sets decided by Turing machines clocked in polynomial time and equipped with an oracle such that, for inputs of size n, only a number $O(\log(n))$ of calls are allowed. From the last proposition we have that:

Proposition 28. *For all Davis' oracles O_r with exponential cost, $P(O_r) = P_\ell(O_r)$.*

In particular,

Proposition 29. *For all functions f with a right inverse approximable in exponential time, $P(\mathcal{O}_{f,r}) = P_\ell(\mathcal{O}_{f,r})$.*

Proposition 30. *For all Davis' oracles $\mathcal{O}$ with exponential cost, $P(\mathcal{O}) \subseteq P/log\star \subseteq P/log$.*

The first inclusion can be easily extended to $P/log\star = \bigcup_{\text{e.c. } \mathcal{O}_r} P(\mathcal{O}_r)$ where e.c. abbreviates exponential cost and $\mathcal{O}_r$ are Davis' oracles. For this class of oracles we have also the positive relativisation of the negative case.

Proposition 31. *$P = NP$ if and only if, for all Davis' oracles O with exponential time access, $P(\mathcal{O}) = NP(\mathcal{O})$.*

Proposition 32. *$P \neq NP$ if and only if, for all Davis' oracles O with exponential time access, $P(O) \neq NP(O)$.*

The proof of Proposition 31 is omitted since it applies the same techniques as in the proof of Theorem 23. We shall prove the second one. For this we will need the following well known proposition from structural complexity (see [2], Chapter 5):

Proposition 33. *If $SAT \in P/log$, then $P = NP$.*

Now we proceed to the proof of Proposition 32.

Proof. If the statement on the right-hand side is true, then for the particular case of $\mathcal{O}_{\frac{1}{3}}$, we have that $P \neq NP$ (see the corresponding case for the polynomial cost). Suppose now that $P \neq NP$. Let $\mathcal{O}$ be a Davis' oracle with exponential cost. By Proposition 33, $SAT \notin P/log$. In particular, $SAT \notin P(\mathcal{O})$ by Proposition 30. As known from structural complexity $SAT \in NP$, which implies that $SAT \in NP(\mathcal{O})$. This proves that $P(\mathcal{O}) \neq NP(\mathcal{O})$. □

As particular cases, we have:

Proposition 34. *$P = NP$ if and only if, for all $r \in [0, 1]$ and f with a right inverse approximable in exponential time, $P(\mathcal{O}_{f,r}) = NP(\mathcal{O}_{f,r})$.*

Proposition 35. *$P \neq NP$ if and only if, for all $r \in [0, 1]$ and f with a right inverse approximable in exponential time, $P(\mathcal{O}_{f,r}) \neq NP(\mathcal{O}_{f,r})$.*

Applying the previous results to the *ARNN* model, we get:

Proposition 36. *$P = NP$ if and only if, for all $r \in [0, 1]$ and f with a right inverse approximable in exponential time, we have $ARNN[\mathbb{Q}](f, r)P = ARNN[\mathbb{Q}](f, r)NP$.*

Proposition 37. *$P \neq NP$ if and only if, for all $r \in [0, 1]$ and f with a right inverse approximable in exponential time, we have $ARNN[\mathbb{Q}](f, r)P \neq ARNN[\mathbb{Q}](f, r)NP$.*

Table 2
A more refined classification of the computational power of the *ARNN*.

Set of weights	Time restriction	Protocol	Computational power
$\mathbb{Z}$	none	none	Regular languages
$\mathbb{Q}$	none	none	r.e. languages
$\mathbb{Q}$	t	none	$DTIME(t)$
$\mathbb{Q} \cup \{r_1\}$	polynomial	exponential	$P_\ell(O_{r_1})$
⋮			⋮
$\mathbb{Q} \cup \{r_1, \ldots, r_n\}$	polynomial	exponential	$P_\ell(O_{r_1}) \ldots (O_{r_n})$
⋮			⋮
$\mathbb{R}$	polynomial	exponential	$P/log*$[a]
$\mathbb{Q} \cup \{r_1\}$	polynomial	polynomial	$P(O_{r_1})$
⋮			⋮
$\mathbb{Q} \cup \{r_1, \ldots, r_n\}$	polynomial	polynomial	$P(O_{r_1}) \ldots (O_{r_n})$
⋮			⋮
$\mathbb{R}$	polynomial	polynomial	$P/poly$[b]
$\mathbb{R}$	none	none	all languages

[a] Note that all the classes in this compartment coincide for arbitrary reals.
[b] Idem.

10. Conclusion

We formalised the idea of a real weight of a *ARNN* as an oracle, up to the point of Martin Davis criticism in [10].

The simulation of a rational neural net with one real weight by a Turing machine with a Davis' oracle shows that, not only the class of $ARNN[\mathbb{R}]P$ coincides with $P/poly$, but furthermore, the structural relation $P = NP$ positively relativises (making these oracles a subclass of sparse sets). *ARNNs* clocked in polynomial time are, in fact, a restricted model of computation. Table 2 summarises the classification. For the case of exponential cost, we also have the positive relativisation of $P \neq NP$. (This result is an open problem for the polynomial case, as in classical structural complexity.)

Returning to more "grounded" issues, there are still simple questions no yet answered. For instance, a very rewarding work would be a classification of functional properties of neural nets. Within the system of [16] it is possible to implement a given logical function. However, giving a description of *how* one *ARNN* performs its computation is not trivial. Most of the work published refers to layered neural nets (for example, a *BAM* structure).

Acknowledgements

The research of José Félix Costa is supported by Fundação para a Ciência e Tecnologia, PEst – OE/MAT/UI0209/2011. We thank John V. Tucker for suggesting the study of the relativisation of $P = NP$ in different models of computation, and Edwin Beggs for working with John V. Tucker and José Félix Costa on the subject of relativisation in quite different framework.

References

[1] José Luis Balcázar, Josep Días, Joaquim Gabarró, Structural Complexity I, 2nd edition, Springer-Verlag, 1988, 1995.
[2] José Luis Balcázar, Josep Días, Joaquim Gabarró, Structural Complexity II, Springer-Verlag, 1990.
[3] Edwin Beggs, José Félix Costa, Diogo Poças, John V. Tucker, On the power of threshold measurements as oracles, in: Manindra Agrawal, Dingzhu Du, Zhenhua Duan, Angsheng Li (Eds.), Unconventional Computation and Natural Computation (UCNC 2013), in: Lecture Notes in Computer Science, Springer, 2013.
[4] Edwin Beggs, José Félix Costa, John V. Tucker, Physical experiments as oracles, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 97 (2009) 137–151. An invited paper for the "Natural Computing Column".
[5] Edwin Beggs, José Félix Costa, John V. Tucker, Limits to measurement in experiments governed by algorithms, in: Salvador Elías Venegas-Andraca (Ed.), Special Issue on Quantum Algorithms, Math. Structures Comput. Sci. 20 (06) (2010) 1019–1050.
[6] Edwin Beggs, José Félix Costa, John V. Tucker, Physical oracles: The Turing machine and the Wheatstone bridge, in: D. Aerts, S. Smets, J.P. Van Bendegem (Eds.), Special Issue on Contributions of Logic to the Foundations of Physics, Studia Logica 95 (1–2) (2010) 279–300.
[7] Edwin Beggs, José Félix Costa, John V. Tucker, Axiomatising physical experiments as oracles to algorithms, Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 370 (12) (2012) 3359–3384.
[8] Edwin Beggs, José Félix Costa, John V. Tucker, The impact of models of a physical oracle on computational power, in: Cristian S. Calude, S. Barry Cooper (Eds.), Special Issue on Computability of the Physical, Math. Structures Comput. Sci. 22 (5) (2012) 853–879.
[9] Jack Copeland, Diane Proudfoot, Alan Turing's forgotten ideas in Computer Science, Sci. Am. 280 (1999) 99–103.
[10] Martin Davis, The myth of hypercomputation, in: Christof Teuscher (Ed.), Alan Turing: The Life and Legacy of a Great Thinker, Springer, 2006, pp. 195–212.
[11] Martin Davis, Why there is no such discipline as hypercomputation, Appl. Math. Comput. 178 (1) (2006) 4–7.
[12] S. Haykin, Neural Networks: A Comprehensive Foundation, MacMillan College Publishing, 1994.

[13] Hava T. Siegelmann, João Pedro Neto, José Félix Costa, Symbolic processing in neural networks, J. Braz. Comput. Soc. 8 (3) (2003) 58–70.
[14] Joe Kilian, Hava T. Siegelmann, The dynamic universality of sigmoidal neural networks, Inform. and Comput. 128 (1) (1996) 48–56.
[15] Bruno Loff, José Félix Costa, Five views of hypercomputation, in: Mike Stannett (Ed.), Special Issue on Future Trends in Hypercomputation, Int. J. Unconv. Comput. 5 (3–4) (2009) 193–207.
[16] Warren McCulloch, Walter Pitts, A logical calculus of ideas immanent in nervous activity, J. Math. Anal. Appl. 5 (1943) 115–133.
[17] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1967.
[18] Mark Schlatter, Ken Aizawa, Walter Pitts and "A Logical Calculus", Synthese 162 (2008) 235–250.
[19] Hava T. Siegelmann, Computation beyond the Turing limit, Science (1995) 545–548.
[20] Hava T. Siegelmann, Neural networks and finite automata, J. Comput. Intell. 12 (4) (1996).
[21] Hava T. Siegelmann, Neural Networks and Analog Computation: Beyond the Turing Limit, Birkhäuser, 1999.
[22] Hava T. Siegelmann, Eduardo D. Sontag, Analog computation via neural networks, Theoret. Comput. Sci. 131 (2) (1994) 331–360.
[23] Hava T. Siegelmann, Eduardo D. Sontag, On the computational power of neural networks, J. Comput. System Sci. 50 (1) (1995) 132–150.