Journal of Intelligent & Fuzzy Systems 13 (2002/2003) 85–98
IOS Press

Turing's analysis of computation and artificial neural networks*

Wilson R. de Oliveira[a,**], Marcílio C. P. de Souto[b] and Teresa B. Ludermir[c]
[a]*Departamento de Física e Matemática, R. Dom Manoel de Medeiros, s/n, 52171-030, Recife (PE), Brazil*
E-mail: wrdo@cin.ufpe.br
[b]*ICMC, Universidade de São Paulo at São Carlos, Caixa Postal 668, 13560-970, São Carlos, SP, Brazil*
E-mail: marcilio@icmc.usp.br
[c]*Centro de Informática, UFPE, Av.Professor Luis Freire, s/n, 50740-540, Recife, PE, Brazil*
E-mail: tbl@cin.ufpe.br

Abstract. A novel way to simulate Turing Machines (TMs) by Artificial Neural Networks (ANNs) is proposed. We claim that the proposed simulation is in agreement with the correct interpretation of Turing's analysis of computation; compatible with the current approaches to analyze cognition as an interactive agent-environment process; and physically realizable since it does not use connection weights with unbounded precision. A full description of an implementation of a universal TM into a recurrent sigmoid ANN focusing on the TM finite state control is given, leaving the tape, an infinite resource, as an external non-intrinsic feature. Also, motivated by the results on the limit of what can actually be computed by ANNs when noise is taken into account, we introduce the notion of Definite Turing Machine and investigate some of its properties.

Keywords: Turing machine, recurrent sigmoid neural networks, mealy automata, definite automata

1. Introduction

Warren McCulloch and Walter Pitts published their paper "A Logical Calculus of the Ideas Immanent in Nervous Activity" [27] back in 1943. This paper is seminal to both the field of Artificial Neural Network (ANN) and to that of Automata Theory. Their logical model of the behavior of the nervous system turned out to be model of a Finite State Machine (FSM). The two fields, despite their common origin, grew apart from each other. The "finite automata" branch focused on computability, whereas the "neural network" branch focused on learning. The former was influenced by the work in [20,28] which gave a more mathematical layout, culminating in its current form [37] first introduced in [33]. The latter was initially influenced by the works in [30,35] and then by studies in [15,22,36,48]. Only recently the two branches interacted again through the use of ANNs as sequence processors [7,17,32] in the late 80 s; and in the early 90 s, with the theoretical analysis of the computational capabilities of ANNs [1, 11,40,41].

Regarding universal computation – the relationship between ANNs and Turing Machines – an odd situation arise. While computer scientists (automata theoreticians) agree that a TM is a finite state automata with an external auxiliary memory realized as a read-write tape and free-moving head [2,29], the neural network researchers insist in viewing the ANN "computer" as a system closed in itself in which all the memory/storage is kept within it. Hence, in their proposed simulation of TM, the infinite tape of the TM is represented as an infinite number of neurons or unbounded weights. This approach somehow mimics the conventional architecture of modern days computers where storage, memory and processors come in a closed box. For a very good collection of arguments and critics in favor

*This work is partially supported by the national research council CNPq, CAPES and FINEP and the state council FAPESP (Proc: 02/03049-6).
**Corresponding author.

of approaches that regard cognition as an interactive agent-environment processe see [47].

A different approach is taken here which is in agreement with the original work of McCulloch and Pitts [27]. Back in the 40's they already claimed that recurrent ANNs are computationally universal. They suggested a neural implementation of a TM, in which the network would both simulate the finite-state control part of the machine and have a read/write head and an infinite tape tape attached to it. Therefore, our *Neural Turing Machine* is similar to a Turing Machine only that the next state control function is not realized as finite state automata (or finite state machines) but rather as ANNs.

However, it should be said, researchers in the ANN field claim that such a simulation is not "neural" in that the tape should be a intrinsic part of the network, not an external device [9,13,31,40,44]. But looking at the process of human calculation[1] one can abstract two main agents: the brain and the environment. These can be further abstracted to brain plus sheet(s) of papers(s). Theoretical computer scientists agree to view them respectively as a finite-state automaton plus an infinite tape, i.e., a Turing Machine.

Thus, this work proposes to substitute the classical model of computation

$$\text{Turing Machine} = \text{Tape} + \text{Finite State Automaton (FSA)}$$

by the neural model of computation

$$\text{Neural Turing Machine (NTM)} = \text{Tape} + \text{Artificial Neural Networks (ANN)}$$

Cognitive and Physical arguments for this approach are given and the mathematical consequences of it are investigated. Among the main consequence of the approach is the definition of a novel Turing machine model, *Definite Turing Machine*(DTM), which arises when noise is taken into account.

It is widely known in the neurocomputing theory community the results in [26] which states that not every arbitrary FSA can be implemented in ANNs when noise or limited precision is considered. ANNs are just *Definite Automata* [21] under these conditions. This result reflects on the second equality above giving

$$\text{(noisy) NTM} = \text{Tape} + \text{(noisy) ANN}$$

with a corresponding

$$\text{Definite Turing Machine} = \text{Tape} + \text{Definite Finite State Automaton (DFSA)}$$

The computational abilities of these Definite Turing Machines are investigated. It is proved that they compute the class of elementary functions from Recursion Theory [4]. Another kind of limited memory automata, the finite memory automata [21], leads to *Finite Memory Turing Machine*. The latter is shown to be equivalent to (unlimited) Turing machines.

The remainder of this paper is divided into 6 sections. Section 2 presents the main definitions used in this work. In Section 3, in order to put this work in perspective, the main approaches for TM simulation in ANN are presented. Then, in Section 4, a critique to these current approaches for neural Turing simulation is presented. Based on the claims stated in the previous sections, in Section 5 we show how the finite control of a TM can be considered as a Mealy machine. Then, using the procedure presented in [5], we explain how to implement the control of a TM in a ANN. Some examples are presented – in particular a minimal universal TM is implemented as an ANN. All this provides the intuition for the proof of the universality of our model. Next, motivated by the results on the limit of what can actually be computed by ANNs when noise is taken into account, in Section 6 we introduce the notion of Definite Turing Machine and investigate some of its properties. We also show that the corresponding notion of Finite-Memory Turing Machine is computationally equivalent to Turing machines. Finally, in Section 7 we present some final remarks and possible extension of this work.

2. Main definitions

Below we start by formalizing the intuitive notion of a Turing machine depicted above. Then, we define the notion of Mealy machine, which is a finite state automaton with an output function. Finally, we define the kind of neural networks we use, the first-order recurrent neural network, where the units have the logistic function as activation function [5].

Definition 2.1. *A Turing Machine* T over *a finite alphabet* Σ *is a quintuple* $T = (\Sigma_T, Q_T, \delta_T, s_I, D_T)$, (*the subscripts are used when necessary*) *where:* (*1*) $Q = \{q_1, \ldots, q_{|Q|}\}$ *is a finite set of states with a distinguished* (initial) *state,* q_I (*usually* q_1)*;* (*2*) $\delta : Q \times \Sigma \rightarrow$

[1] Actually, the process of number theoretic calculations by a mathematician. This really was the original motivation of people such as Gödel, Turing, Church, Kleene, Herbrand, Post, Markov, etc. when attacking the *Entscheidungsproblem*.

$Q \times \Sigma \times D$ is a function, where $D = \{-1, 0, 1\}$ is the set of possible head movements. δ is to be thought as a finite set of instructions and $\delta(q_i, \sigma) = (q_j, \sigma', m)$ means that the machine being in the state $q_i \in Q$ and reading the symbol $\sigma \in \Sigma$ from the current cell in the tape will take the following actions: erase σ from the cell and write σ' in its place; change the internal state from q_i to q_j and move the head position one cell to the left ($m = -1$), one to the right ($m = 1$) or does not move ($m = 0$). □

Definition 2.2 *Mealy machine is a sixtuple $M = (\Sigma, \Gamma, Q, \delta, \lambda, q_I)$, where: (1) $Q = \{q_1, q_2, \cdots, q_{|Q|}\}$ is a finite set of states; (2) $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_{|\Sigma|}\}$ is a finite input alphabet; (3) $\Gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_{|\Gamma|}\}$ is a finite output alphabet; (4) $\delta : Q \times \Sigma \to Q$ is the next state function. $\delta(q_j, \sigma_k) = q_i$ should be interpreted as "the machine being in state q_j and reading symbol σ_k goes to state q_i"; (5) $\lambda : Q \times \Sigma \to \Gamma$. $\lambda(q_j, \sigma_k) = \lambda_i$ should be interpreted as "the machine being in state q_j and reading symbol σ_k writes symbol λ_i; and (6) $q_I \in Q$ is the initial state where the machine is before the first symbol is read.* □

Remark 2.1. *A pair (Q, δ) is usually called a transition system (over Σ). We can sometimes refer to the transition system of the Mealy machine.* □

For the class of ANNs used, in what follows in the definition below, superscripts in the weights indicate the computation involved: for example the xu in W^{xu} indicates that the weight is used to compute a state (x) from an input (u); the x in W^x (a bias) indicates that it is used to compute a state. Thus, u, x, y and z in the subscript means respectively input, state, output and hidden output layer. The definition of ANN that we use is basically the *augmented Robinson-Fallside network* of the Mealy kind in [5]. Throughout let $g : \Re \to [0, 1]$ be the *logistic map*: $g(x) = \frac{1}{1+\exp(-x)}$

Definition 2.3. *A Deterministic Time Recurrent Neural Network (DTRNN) is a sixtuple $N = (X, U, Y, \mathbf{f}, \mathbf{h}, \mathbf{x}_0)$ where: (1) $X = [0, 1]^{n_X}$, the n_X-dimensional unit cube, is the state space of the network and n_X is the number of units or neurons; (2) $U = \Re^{n_U}$ is the set of possible input vector, with $\Re$ the set of real numbers and n_U the number of input lines; (3) $Y = [0, 1]^{n_Y}$ is the set of outputs of the network, with n_Y the number of output units; (4) $\mathbf{f} : X \times U \to X$ is the next state function which computes a new state $\mathbf{x}[t]$ from the previous state $\mathbf{x}[t-1]$ and the input just read $\mathbf{u}[t]$. The $i-$th coordinate of $\mathbf{f}$ is given by*

$$f_i(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_X} W_{ij}^{xx} x_j[t-1] + \sum_{j=1}^{n_U} W_{ij}^{xu} u_j[t] + W_i^x\right)$$

□

(5) $\mathbf{h} : X \times U \to Y$ is the *output function* which computes a new output $\mathbf{y}[t]$ from the previous state $\mathbf{x}[t-1]$ and the input just read $\mathbf{u}[t]$. The $i-$th coordinate of $\mathbf{h}$ is given by

$$h_i(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_Z} W_{ij}^{yz} z_j[t] + W_i^y\right)$$

where

$$z_i[t] = g\left(\sum_{j=1}^{n_X} W_{ij}^{zx} x_j[t-1] + \sum_{j=1}^{n_U} W_{ij}^{zu} u_j[t] + W_i^z\right)$$

(6) and finally, $\mathbf{x}_0$ is the *initial state* of the network which simply is the value that will be used for $\mathbf{x}(0)$.

Remark 2.2. *As shown in [11], a network as previously defined can represent the output function of all Mealy machine as long as a two-layer scheme is used. Hence, we assume $Z = [0, 1]^{n_Z}$ be the hidden output layer where n_Z is the number of such units.* □

3. Related work

Most researchers in the ANNs field claim that in neural Turing simulation the tape should be a intrinsic part of the network, not an external device [9,13,31, 40,44]. Thus, in order to "internalize" the tape of a TM, these researchers have come up with the following approaches:

- Networks with an infinite number of nodes [9,13, 44]; and
- Networks with finite number of nodes but with unbounded weights [31,40,38].

3.1. Networks with infinite number of neurons

A recurrent ANN with an infinite amount of nodes can simulate a TM by using this potentially infinite amount to build the TM's machine tape [9,13,44].

Franklin and Garzon in [9] defend the study of these infinite networks for they claim that there exists a relation between finite and infinite networks such as, for example, that of Finite State Machines (FSMs) and TMs. That is, although each physically implemented machine is, in principle, a FSM, the concept of TM is needed for a deeper understanding on the complete computational power of sequential computers. For example, an asymptotic analysis is seen as providing a good understanding of the behavior of algorithms – when the size of the input scales up. Therefore, the TMs, with their infinite tape, provide a convenient way to study these scaling up of individual solutions.

Franklin and Garzon [9] also claims that infinite recurrent ANNs are strictly more powerful than TMs. Thus, for them, an understanding of the computational power of infinite networks would provide a better understanding of the neurocomputing. On the other hand, one can argue that, because these models contain an unbounded number of neurons, they cannot really explain the true power of their networks. They provide not only infinite memory (a fair source), but they become infinite automata. Some researchers, thus, argue that it would be more appropriate to consider computational models which include a finite number of neurons, but still allow for growing memory by allowing, for example, for growing precision in the neurons.

3.2. Networks with a finite number of neurons

The first neural computation model that combined a finite number of nodes and infinite precision was introduced by Pollack [31]; he proved it to be universal in power. Each node computed a second-order polynomial or threshold function. Pollack's networks did not allow for first polynomials, as it is common in ANNs, nor did it allow for any continuous activation function. Continuity, as already mentioned, is an important requirement when one wants to model physical systems with ANNs.

Thus, in this section, the attention will be focus on the findings of Sigelmann and Sontag [38,40] who studied the computability of analog recurrent networks. So far, their study has triggered much other work – e.g. [18, 23,39]. More specifically, Sigelmann and Sontag [40] showed that a single-input, single-output, first-order, recurrent ANN employing the saturated-linear activation function can simulate arbitrary TMs step per step (the network is similar to the one in Definition 2.3). Hence, by implementing universal TM, any function computable by a TM in time $T(n)$ can be computed by a fixed recurrent network with only 886 units in time $O(T(n))$. The size of this universal network can further be reduced even to 25 neurons at the cost of increasing the simulation time to $(n^2T(n))$ [16]. Their architecture, as mentioned before, is a simple first-order network:

$$\begin{aligned} x_i[t] &= g_\sigma\left(\sum_{j=1}^{n_X} W_{ij}x_j[t-1] + W_i u[t] + W_i\right) \\ y[t] &= x_1[t] \end{aligned} \tag{1}$$

where $x_i[t]$ are rational numbers, $u[t]$ is either 0 or 1, and g_σ is 0 if $x < 0$, 1 if $x > 1$ and x otherwise (saturated-linear function). The input tape is the sequence $u[1]u[2]\ldots$; the output tape is the sequence $y[1]y[2]\ldots$.

The proof presented in [40] stands on the following assumptions [8]:

- A TM can always be simulated by a pushdown automaton with three (in fact two) unary stacks [14];
- An unary stack can be encoded as a rational number: $q_s = 0.1111 = 15/16$ represents four items. Popping an item is the same as performing a subtraction $q_s^{'} = \sigma(2q_s - 1)$; whereas pushing an item is the same as doing an addition $q_s^{'} = \sigma(1/2 + q_s/2)$;
- All stack operations and all state transitions triggered by the states of the control unit and the symbols at the top of the stacks can be computed in at most two time steps of the network.

In addition, Turing universality has been shown for a more general class of activation function, including the standard sigmoid function [19,24], although in this case the simulation requires exponential time overhead per each computational step. Recently, the class of NARX networks has also been shown to be equivalent to TMs [39].

Another important point with respect to the simulation of TMs by first-order recurrent ANNs with saturated-linear activation function is that their power depends on the descriptive complexity of their weights. For integer weights, these networks coincide with recurrent networks of threshold gates as activation function, that is, they are equivalent to FSMs [40]. If rational weights are employed, this class of networks becomes equivalent to TMs.

On the other hand, when these networks are used with real weights, they can present super-Turing computational capabilities [38]. In particular, the computational power of such networks working within time

$T(n)$ is exactly the same as that of *non uniform* Turing machines[2]. That is, non realizable TMs that receive in their input tape, in addition to the input w, another string $W(|w|)$ called the "advice" to assist the computation. This means that polynomial-time computations by such networks correspond to the non uniform complexity class P/poly, and within exponential time any input-output mapping can be computed.

4. A critique to current implementations of TMs in neural networks

In this section, we present arguments against the claim that the infinite tape should be considered as an intrinsic feature of a TM, as it has often been done in the neurocomputing literature [9,13,40,44]. First, we argue that there is a misunderstanding in the way these works on neural simulation of TMs interpret the Turing's analysis of computation – this argument was first presented in [6]. Then, using as support the work by [47], we claim that an external view of the TM's tape is compatible with current studies of cognition based on the agent-environment paradigm. Next, we argue that, by considering the TM tape as an external device to the ANN, problems related to physical implementation of analog networks are avoided [25]. Finally, we present some objections for super-Turing computation in ANNs.

4.1. Turing's analysis of computation

One can argue that the "mind" of a Turing Machine is its control: a finite state automaton (more specifically a Mealy Machine). The access to the tape is only for helping the computation storing partial results. What makes a TM computationally powerful enough as to model effective computations (Church-Turing Thesis) is the set of permissible operations on the tape. The various classical kinds of finite automata in the Computer Science literature can then be roughly classified according to the set of permissible operations on their tape(s).

A finite state automaton (regular languages recognizers) have one input and one output tape, but it is allowed to just move the heads to the right and as a consequence is not able to reuse past computations. A pushdown automaton (context-free languages recognizers) has its access to the tape restricted to *stack* operations. A linear bounded automaton (context sensitive languages recognizers) is a TM with the restrictions that the head cannot leave those cells on which the input was placed. In contrast, a Turing machine have unrestricted access to the tape. For all types of machines previously described, the tapes are assumed to be infinite and the "mind" of each type of machine[3] is indeed a finite state automaton.

The point is that the fact that this memory is *potentially* infinite should not be considered as an intrinsic feature of a Turing machine, as it has been indicated in some neural computing literature [9,13,31,40,44]. This memory is there for use, however at any time step the TM is not allowed to use the whole infinite resource. Only finite portions of the tape can be used at any stage of a computation. It is the finitistic, repetitive and uninspired execution of *instructions* done by the finite state control that matters. Because of this, one could regard this finite state control as the "mind" of the TM.

Of course, as seen above, the set of permissible operations is also of fundamental significance. Indeed, these aspects of a TM capture reasonably well the notion of *effective computability*. The access to an infinite tape has also obvious technical advantages. One can speak, for example, of computations of number theoretic functions, $f : N \to N$, where N is the set of natural numbers, even if it is not possible for any existing device or any human being to actually compute the whole of a such function. Take as an example the addition of natural numbers; we know how to perform the addition of two given numbers, but it would not be possible for us to actually compute the function $+ : N^2 \to N$ for its whole domain of definition: just imagine sufficiently large numbers whose addition would take us, say, a billion of years to perform; however that is not what matters for the study of effective computations but rather the fact that there is something mechanical, repetitive, effective, *algorithmic* about addition that one can capture. And once one has *learned* how to add a finite set of pairs of *small* natural numbers it can be said that this person can (in principle) add any two given natural numbers when the person actually knows how to perform an algorithm for addition.

[2] Non uniform TMs are unrealizable because the length of w is not bounded. Thus, the number of possible advice strings is infinite and cannot be stored in finite memory previous to any computation

[3] In some cases it is important to distinguish between deterministic and non deterministic machines but this is out of the scope of the present work and the interested reader should consult the literature (e.g., the classic [14])

4.2. Turing machines and the study of cognition

In the past years there have been many debates on two distinct approaches to the study of human cognition (see articles in [42]). One approach, the tradition upon which cognitive science was founded, is that of symbolic processing [45]. The other more recent approach, emphasizing the role of the environment, the context, the social and cultural setting, and situations in which actors find themselves, is often called *situated action* or *situation cognition* [12].

In the first approach, symbol systems theorists argue that the human brain functions in the way as a TM. Thus, by hypothesis, there must be mechanisms in the brain that are functionally equivalent to the finite control and to the tape [45]. As pointed by [47], symbol systems theorists pay particular attention to memory/tape mechanisms because they are hypothesized to contain structured representation of the world. The manipulation and transformation of these representations constitute cognitive activity. Since this approach assumes that both the principal parts of the TM are instantiated in the brain, it has been referred to as the *internalist* interpretation of the Turing Machine architecture [47]. The internalist interpretation is essentially what has been supported by many theorists [10].

In the context of the neurocomputing, the internalist interpretation has also been often accepted. For example, representative works on the neural simulation of TMs such as [9,13,31,40,44] simulate both parts of the TM within the neural network. On the other hand, in this work we share similar views such as those presented by Wells [47]. He argues for an alternative approach for the internalist interpretation in which only one of the principal parts, the finite state control/central processor, is instantiated in the brain. The other principal parts, the tape/memory mechanism, is hypothesized to exist in the external environment. Consequently, cognitive computation is a process of organism-environment interaction. This approach is based on what Wells calls the *interactive* interpretation of the Turing machine architecture [47].

Under this interactive conception, the brain instantiates not the full task architecture of a Turing machine, but only the control architecture. The input architecture is found in the external environment. Therefore, according to Wells [47], this leads to a view of cognitive computation processes of structured interaction between the control architecture of the brain and the input architecture of the environment. In other words, the cognizer is embedded in an environment which constitutes part of the cognitive architecture. This approach is compatible with one of the central claims of situation theories: cognitive activities should be primarily as interactions between agents and physical systems and with other people [12].

As seen above, the interactive interpretation of TMs results in a view of cognitive architectures which is incompatible with architectures of symbols systems, as well as most neural simulation of TMs such as the ones in [40]. However, by re-interpreting Turing's analysis of computation (see previous section) and rescuing the original work by McCulloch and Pitts [27], our neural simulation of TMs proposed here shows that connectionist and interactive approaches to cognitive architectures can be theoretically compatible in that both can see the organism as a finite controller embedded in a wider environment that might form part of the architecture.

4.3. Analog noise and ANNs

The results on the Turing capabilities of analog neural networks presented in Section 3 hold only for the noise-free case. For instance, Maass and Orponen [25] showed that, in the presence of any reasonable type of analog noise, these networks can deal only with regular languages. This is true even if unlimited computation time is allowed and if arbitrary real-valued parameters are employed. Furthermore, they showed that if this analog noise is bounded, then all regular languages can be recognized by such systems. But if the noise is unbounded, then the neural network will no longer be able to recognize arbitrary regular languages.

In fact, Maass and Sontag [26] have shown that when the output gates of these analog networks are subjected to unbounded Gaussian noise, or any other common noise distribution, they can recognize only a small subclass of the regular languages called definite languages. Thus, in this context, and even for the bounded noise case, the computability of analog neural networks is limited to that of FSMs, despite the potential "infinite" number of states. This result implies severe constraints on the possibility for constructing recurrent analog networks that are robust against realistic types of analog noise.

Therefore, from a theoretical point of view, although the neural implementations of TM that internalize the tape as the ones in [13,40,44] could simulate the behavior of TMs; from a practical point of view, if these networks are subject to – let say – bounded analog noise, they will be nothing more than finite state automaton.

Hence, this is another argument to support our claim that the TM's tape should not be encoded within the neural network.

4.4. Objections for super-turing computation in ANNs

From a mathematical perspective, allowing neural networks to have real weights leads them to a richer representational ability than a machine that uses a finite number of discrete symbols on an infinite tape. However, as it is widely acknowledged, if a TM is to process in a finite time, and each square or digit on the tape takes a finite time to be read, then only a finite portion of the infinite tape can actually be used. This limits the device to dealing with a subset of rational numbers (those rational numbers that can be represented by the particular number of symbols or bits that can be read in the allotted time). Thus, in order to represent a continuous-valued quantity, the set of real numbers, R, must be mapped on to (at most) the set of rational numbers Q.

There are more irrational than rational numbers, though both sets are infinite, the former has cardinality N_1, whereas the latter has cardinality N_0. As a consequence, many states in the original continuous function map to a single number within the Turing machine. This loss of information between the continuous-valued world and the TMs is the source of the proposed non Turing ability of recurrent ANNs with real weights.

As pointed in [3], in order to avoid the problem above, one can argue that for any given process, the TM can be designed to use enough bits to produce the correct answer. However, while this could be possible for many engineering applications, it is not practical for intelligent systems designed to function in an unforeseeable universe.

5. Our model of simulation

Based on the arguments presented in the previous section, we show a proof for the Turing universality of a class of ANNs (DTRNN), in which the tape is seen as a non-intrinsic part of the network. But, before presenting the formal proof, we give some intuition. First, we show how the finite control of a TM can be simulated by the network. Then, we show an implementation of a universal Turing machine as an ANN. Finally, we show the formal proof.

5.1. Simulating the finite control of a TM in an ANN

The simulation can easily be grasped by first observing that computations in a Turing machine is actually controlled by a Mealy machine:

$$\delta_T(q, \sigma) = (q', \sigma', d)$$

can be seen as

$$\delta_M(q, \sigma) = q'$$

and

$$\lambda_M(q, \sigma) = (\sigma', d)$$

i.e., to a given TM $T = (\Sigma_T, Q_T, \delta_T, s_I, D_T)$ it is associated a Mealy machine $M = (\Sigma_T, \Gamma, Q_T, \delta_M, \lambda_M, s_I)$, where $\Gamma = \Sigma_T \times D_T$, and δ_M and λ_M are defined as above. The Mealy machine M can be seen as the "brain" of the Turing machine T.

Remark 5.1 *The number of output symbols is $3 * n$, where n is the number of symbols in Σ.* □

Closely following the procedure for stable implementation of a Mealy machine in DTRNNs presented in [5], from the Mealy machine M above it is constructed the *split-state split-output* Mealy machine

$$M' = (\Sigma_T, \Gamma', Q', \delta', \lambda', q'_I)$$

where

- $\Gamma' = \Gamma \times \Sigma$, $Q' = Q \times \Sigma$, and s'_I is any of the $(s_I, \sigma), \sigma \in \Sigma$;
- $\delta'((q, \sigma), \sigma') = (\delta_M(q, \sigma'), \sigma')$ and
- $\lambda'((q, \sigma), \sigma') = (\lambda(q, \sigma'), \sigma')$.

Remark 5.2 *The number of output symbols of M' is $2 * n^2$ and the number of states is $m * n$, where m is the number of states we start with, i.e., $|Q_T| = m$.* □

Remark 5.3 *The procedure in [5] gives a DTRNN with the number of state space units, input lines and output units, respectively, $n_X = |Q'|$, $n_U = |\Sigma|$ and $n_Y = |\Gamma'|$, which amounts to $n + 3n^2 + mn$ units.* □

Example 5.1 *Below in Fig. 1(a) is given an example of a Turing Machine T with 6 states and 2 input symbols which multiplies by two a given input in unary base. The corresponding Mealy machine M implementing the control of T is shown in Fig. 1(b). In Fig. 2(a) the split-state split-output machine S from M is illustrated. S has states which are unreachable from the initial state 0. Then, when these states are removed the machine in Fig. 2(b) is produced. Using the results in [5] one finds $H = 9.14211$, $\epsilon_0 = 0.0415964$ and $\epsilon_1 = 0.988652$.* □

δ_T (0 , 0)	=	(0 , 0, 1)		δ_M (0 , 0)	=	0	and	λ_M (0 , 0)	=	1
δ_T (0 , 1)	=	(1 , 0, 1)		δ_M (0 , 1)	=	1	and	λ_M (0 , 1)	=	1
δ_T (1 , 0)	=	(2 , 1, 2)		δ_M (1 , 0)	=	2	and	λ_M (1 , 0)	=	5
δ_T (1 , 1)	=	(1 , 1, 1)		δ_M (1 , 1)	=	1	and	λ_M (1 , 1)	=	4
δ_T (2 , 0)	=	(3 , 0, 1)		δ_M (2 , 0)	=	3	and	λ_M (2 , 0)	=	1
δ_T (2 , 1)	=	(4 , 0, 1)		δ_M (2 , 1)	=	4	and	λ_M (2 , 1)	=	1
δ_T (3 , 0)	=	(0 , 1, 0)		δ_M (3 , 0)	=	0	and	λ_M (3 , 0)	=	3
δ_T (3 , 1)	=	(3 , 1, 1)		δ_M (3 , 1)	=	3	and	λ_M (3 , 1)	=	4
δ_T (4 , 0)	=	(5 , 1, 2)		δ_M (4 , 0)	=	5	and	λ_M (4 , 0)	=	5
δ_T (4 , 1)	=	(4 , 1, 1)		δ_M (4 , 1)	=	4	and	λ_M (4 , 1)	=	4
δ_T (5 , 0)	=	(2 , 1, 2)		δ_M (5 , 0)	=	2	and	λ_M (5 , 0)	=	5
δ_T (5 , 1)	=	(5 , 1, 2)		δ_M (5 , 1)	=	5	and	λ_M (5 , 1)	=	5

(a) (b)

Fig. 1. (a)A Turing machine T that multiplies by two a given number in unary base (b)The control of the TM T as a Mealy machine M, as in the text. The composed output pairs $(\sigma, d), \sigma \in \Sigma, d \in D$, have been coded into the integer $\sigma|M| + d$, i.e., it is seen as a number in base $|D|$ (which is 3 in this case).

δ_S (0 , 0)	=	0	and	λ_S (0 , 0)	=	1		δ_R (0 , 0)	=	0	and	λ_R (0 , 0)	=	1
δ_S (0 , 1)	=	7	and	λ_S (0 , 1)	=	7		δ_R (0 , 1)	=	1	and	λ_R (0 , 1)	=	7
δ_S (1 , 0)	=	2	and	λ_S (1 , 0)	=	5		δ_R (1 , 0)	=	2	and	λ_R (1 , 0)	=	5
δ_S (1 , 1)	=	7	and	λ_S (1 , 1)	=	10		δ_R (1 , 1)	=	1	and	λ_R (1 , 1)	=	10
δ_S (2 , 0)	=	3	and	λ_S (2 , 0)	=	1		δ_R (2 , 0)	=	3	and	λ_R (2 , 0)	=	1
δ_S (2 , 1)	=	10	and	λ_S (2 , 1)	=	7		δ_R (2 , 1)	=	5	and	λ_R (2 , 1)	=	7
δ_S (3 , 0)	=	0	and	λ_S (3 , 0)	=	3		δ_R (3 , 0)	=	0	and	λ_R (3 , 0)	=	3
δ_S (3 , 1)	=	9	and	λ_S (3 , 1)	=	10		δ_R (3 , 1)	=	4	and	λ_R (3 , 1)	=	10
δ_S (4 , 0)	=	5	and	λ_S (4 , 0)	=	5		δ_R (4 , 0)	=	0	and	λ_R (4 , 0)	=	3
δ_S (4 , 1)	=	10	and	λ_S (4 , 1)	=	10		δ_R (4 , 1)	=	4	and	λ_R (4 , 1)	=	10
δ_S (5 , 0)	=	2	and	λ_S (5 , 0)	=	5		δ_R (5 , 0)	=	6	and	λ_R (5 , 0)	=	5
δ_S (5 , 1)	=	11	and	λ_S (5 , 1)	=	11		δ_R (5 , 1)	=	5	and	λ_R (5 , 1)	=	10
δ_S (6 , 0)	=	0	and	λ_S (6 , 0)	=	1		δ_R (6 , 0)	=	2	and	λ_R (6 , 0)	=	5
δ_S (6 , 1)	=	7	and	λ_S (6 , 1)	=	7		δ_R (6 , 1)	=	7	and	λ_R (6 , 1)	=	11
δ_S (7 , 0)	=	2	and	λ_S (7 , 0)	=	5		δ_R (7 , 0)	=	2	and	λ_R (7 , 0)	=	5
δ_S (7 , 1)	=	7	and	λ_S (7 , 1)	=	10		δ_R (7 , 1)	=	7	and	λ_R (7 , 1)	=	11
δ_S (8 , 0)	=	3	and	λ_S (8 , 0)	=	1								
δ_S (8 , 1)	=	10	and	λ_S (8 , 1)	=	7								
δ_S (9 , 0)	=	0	and	λ_S (9 , 0)	=	3								
δ_S (9 , 1)	=	9	and	λ_S (9 , 1)	=	10								
δ_S (10 , 0)	=	5	and	λ_S (10 , 0)	=	5								
δ_S (10 , 1)	=	10	and	λ_S (10 , 1)	=	10								
δ_S (11 , 0)	=	2	and	λ_S (11 , 0)	=	5								
δ_S (11 , 1)	=	11	and	λ_S (11 , 1)	=	11								

(a) (b)

Fig. 2. (a)The split-state split-output Mealy machine S, from M, above. Again we have encoded pairs into integers. The encoding takes (q, σ) into $q + |\Sigma| * \sigma$ (b)The Mealy machine R obtained from S above removing the unreachable states.

5.2. Implementing a universal TM

Example 5.2 *Repeating the steps in the previous section by starting now with a (minimal) Universal Turing Machine [34] with 24 states and 2 input symbols (UTM(24,2)); one finds* $H = 9.862505$, $\epsilon_0 = 0.0200119$ *and* $\epsilon_1 = 0.991678$. *The number of units depends on the amount of unreachable states being removed and cannot obviously be generally predicted. But it can easily be found the upper bound as follows. The numbers of units for a Turing machine with* m *states and* n *symbols is* $3n^2 + mn + n$ *which in our case (two symbol TM) becomes* $2(m + 7)$. *The whole process is illustrated in Fig. 3 to Fig. 6.* □

6. A proof of universality of ANN

Theorem 6.1 *Any Turing machine over* Σ *with* m *states can be simulated by a sigmoid Turing neural network*

δ_T (0 , 0)	=	(4 , 0, 1)	δ_T (12 , 0)	=	(9 , 0, 1)
δ_T (0 , 1)	=	(1 , 1, 1)	δ_T (12 , 1)	=	(23 , 1, 1)
δ_T (1 , 0)	=	(0 , 1, 1)	δ_T (13 , 0)	=	(14 , 0, 2)
δ_T (1 , 1)	=	(2 , 1, 2)	δ_T (13 , 1)	=	(10 , 1, 2)
δ_T (2 , 0)	=	(3 , 0, 2)	δ_T (14 , 0)	=	(15 , 0, 1)
δ_T (2 , 1)	=	(1 , 0, 2)	δ_T (14 , 1)	=	(16 , 1, 1)
δ_T (3 , 0)	=	(11 , 1, 2)	δ_T (15 , 0)	=	(14 , 0, 1)
δ_T (3 , 1)	=	(8 , 0, 2)	δ_T (15 , 1)	=	(9 , 1, 1)
δ_T (4 , 0)	=	(1 , 1, 1)	δ_T (16 , 0)	=	(15 , 0, 1)
δ_T (4 , 1)	=	(5 , 0, 2)	δ_T (16 , 1)	=	(20 , 1, 1)
δ_T (5 , 0)	=	(6 , 0, 2)	δ_T (17 , 0)	=	(18 , 0, 1)
δ_T (5 , 1)	=	(6 , 1, 2)	δ_T (17 , 1)	=	(19 , 1, 1)
δ_T (6 , 0)	=	(7 , 0, 2)	δ_T (18 , 0)	=	(2 , 1, 2)
δ_T (6 , 1)	=	(5 , 0, 2)	δ_T (18 , 1)	=	(17 , 1, 1)
δ_T (7 , 0)	=	(6 , 0, 2)	δ_T (19 , 0)	=	(17 , 1, 1)
δ_T (7 , 1)	=	(1 , 1, 1)	δ_T (19 , 1)	=	(17 , 0, 1)
δ_T (8 , 0)	=	(18 , 0, 1)	δ_T (20 , 0)	=	(21 , 0, 1)
δ_T (8 , 1)	=	(3 , 1, 2)	δ_T (20 , 1)	=	(22 , 1, 1)
δ_T (9 , 0)	=	(3 , 1, 2)	δ_T (21 , 0)	=	(9 , 1, 2)
δ_T (9 , 1)	=	(12 , 0, 1)	δ_T (21 , 1)	=	(20 , 1, 1)
δ_T (10 , 0)	=	(3 , 0, 2)	δ_T (22 , 0)	=	(20 , 1, 1)
δ_T (10 , 1)	=	(0 , 1, 0)	δ_T (22 , 1)	=	(20 , 0, 1)
δ_T (11 , 0)	=	(18 , 0, 1)	δ_T (23 , 0)	=	(12 , 0, 1)
δ_T (11 , 1)	=	(13 , 1, 2)	δ_T (23 , 1)	=	(2 , 0, 2)

Fig. 3. A minimal universal Turing machines on 24 states and two symbols.

δ_M (0 , 0)	=	4	and	λ_M (0 , 0)	=	1	δ_M (12 , 0)	=	9	and	λ_M (12 , 0)	=	1
δ_M (0 , 1)	=	1	and	λ_M (0 , 1)	=	4	δ_M (12 , 1)	=	23	and	λ_M (12 , 1)	=	4
δ_M (1 , 0)	=	0	and	λ_M (1 , 0)	=	4	δ_M (13 , 0)	=	14	and	λ_M (13 , 0)	=	2
δ_M (1 , 1)	=	2	and	λ_M (1 , 1)	=	5	δ_M (13 , 1)	=	10	and	λ_M (13 , 1)	=	5
δ_M (2 , 0)	=	3	and	λ_M (2 , 0)	=	2	δ_M (14 , 0)	=	15	and	λ_M (14 , 0)	=	1
δ_M (2 , 1)	=	1	and	λ_M (2 , 1)	=	2	δ_M (14 , 1)	=	16	and	λ_M (14 , 1)	=	4
δ_M (3 , 0)	=	11	and	λ_M (3 , 0)	=	5	δ_M (15 , 0)	=	14	and	λ_M (15 , 0)	=	1
δ_M (3 , 1)	=	8	and	λ_M (3 , 1)	=	2	δ_M (15 , 1)	=	9	and	λ_M (15 , 1)	=	4
δ_M (4 , 0)	=	1	and	λ_M (4 , 0)	=	4	δ_M (16 , 0)	=	15	and	λ_M (16 , 0)	=	1
δ_M (4 , 1)	=	5	and	λ_M (4 , 1)	=	2	δ_M (16 , 1)	=	20	and	λ_M (16 , 1)	=	4
δ_M (5 , 0)	=	6	and	λ_M (5 , 0)	=	2	δ_M (17 , 0)	=	18	and	λ_M (17 , 0)	=	1
δ_M (5 , 1)	=	6	and	λ_M (5 , 1)	=	5	δ_M (17 , 1)	=	19	and	λ_M (17 , 1)	=	4
δ_M (6 , 0)	=	7	and	λ_M (6 , 0)	=	2	δ_M (18 , 0)	=	2	and	λ_M (18 , 0)	=	5
δ_M (6 , 1)	=	5	and	λ_M (6 , 1)	=	2	δ_M (18 , 1)	=	17	and	λ_M (18 , 1)	=	4
δ_M (7 , 0)	=	6	and	λ_M (7 , 0)	=	2	δ_M (19 , 0)	=	17	and	λ_M (19 , 0)	=	4
δ_M (7 , 1)	=	1	and	λ_M (7 , 1)	=	4	δ_M (19 , 1)	=	17	and	λ_M (19 , 1)	=	1
δ_M (8 , 0)	=	18	and	λ_M (8 , 0)	=	1	δ_M (20 , 0)	=	21	and	λ_M (20 , 0)	=	1
δ_M (8 , 1)	=	3	and	λ_M (8 , 1)	=	5	δ_M (20 , 1)	=	22	and	λ_M (20 , 1)	=	4
δ_M (9 , 0)	=	3	and	λ_M (9 , 0)	=	5	δ_M (21 , 0)	=	9	and	λ_M (21 , 0)	=	5
δ_M (9 , 1)	=	12	and	λ_M (9 , 1)	=	1	δ_M (21 , 1)	=	20	and	λ_M (21 , 1)	=	4
δ_M (10 , 0)	=	3	and	λ_M (10 , 0)	=	2	δ_M (22 , 0)	=	20	and	λ_M (22 , 0)	=	4
δ_M (10 , 1)	=	0	and	λ_M (10 , 1)	=	3	δ_M (22 , 1)	=	20	and	λ_M (22 , 1)	=	1
δ_M (11 , 0)	=	18	and	λ_M (11 , 0)	=	1	δ_M (23 , 0)	=	12	and	λ_M (23 , 0)	=	1
δ_M (11 , 1)	=	13	and	λ_M (11 , 1)	=	5	δ_M (23 , 1)	=	2	and	λ_M (23 , 1)	=	2

Fig. 4. The control of the TM T as a Mealy machine M, as in the text. The composed output pairs $(\sigma, d), \sigma \in \Sigma, d \in D$, have been coded into the integer $\sigma|M| + d$, i.e., it is seen as a number in base $|D|$ (which is 3 in this case).

with $3n^2 + n + nm$ *units.* □

Proof: Given a TM $T = (\Sigma_T, Q_T, \delta_T, s_I, D_T)$ its associated Mealy machine M=(Σ_T,Γ,Q_T,δ_M,λ_M,s_I) (as shown above), where $\Gamma = \{0, \ldots, k-1\}$ with $k = |\Sigma_T \times D_T|$, and δ_M and λ_M are defined as:

if

$$\delta_T(q, \sigma) = (q', \sigma', d)$$

δ_S (0 , 0)	=	4	and	λ_S (0 , 0)	=	1		δ_S (24 , 0)	=	4	and	λ_S (24 , 0)	=	1
δ_S (0 , 1)	=	25	and	λ_S (0 , 1)	=	10		δ_S (24 , 1)	=	25	and	λ_S (24 , 1)	=	10
δ_S (1 , 0)	=	0	and	λ_S (1 , 0)	=	4		δ_S (25 , 0)	=	0	and	λ_S (25 , 0)	=	4
δ_S (1 , 1)	=	26	and	λ_S (1 , 1)	=	11		δ_S (25 , 1)	=	26	and	λ_S (25 , 1)	=	11
δ_S (2 , 0)	=	3	and	λ_S (2 , 0)	=	2		δ_S (26 , 0)	=	3	and	λ_S (26 , 0)	=	2
δ_S (2 , 1)	=	25	and	λ_S (2 , 1)	=	8		δ_S (26 , 1)	=	25	and	λ_S (26 , 1)	=	8
δ_S (3 , 0)	=	11	and	λ_S (3 , 0)	=	5		δ_S (27 , 0)	=	11	and	λ_S (27 , 0)	=	5
δ_S (3 , 1)	=	32	and	λ_S (3 , 1)	=	8		δ_S (27 , 1)	=	32	and	λ_S (27 , 1)	=	8
δ_S (4 , 0)	=	1	and	λ_S (4 , 0)	=	4		δ_S (28 , 0)	=	1	and	λ_S (28 , 0)	=	4
δ_S (4 , 1)	=	29	and	λ_S (4 , 1)	=	8		δ_S (28 , 1)	=	29	and	λ_S (28 , 1)	=	8
δ_S (5 , 0)	=	6	and	λ_S (5 , 0)	=	2		δ_S (29 , 0)	=	6	and	λ_S (29 , 0)	=	2
δ_S (5 , 1)	=	30	and	λ_S (5 , 1)	=	11		δ_S (29 , 1)	=	30	and	λ_S (29 , 1)	=	11
δ_S (6 , 0)	=	7	and	λ_S (6 , 0)	=	2		δ_S (30 , 0)	=	7	and	λ_S (30 , 0)	=	2
δ_S (6 , 1)	=	29	and	λ_S (6 , 1)	=	8		δ_S (30 , 1)	=	29	and	λ_S (30 , 1)	=	8
δ_S (7 , 0)	=	6	and	λ_S (7 , 0)	=	2		δ_S (31 , 0)	=	6	and	λ_S (31 , 0)	=	2
δ_S (7 , 1)	=	25	and	λ_S (7 , 1)	=	10		δ_S (31 , 1)	=	25	and	λ_S (31 , 1)	=	10
δ_S (8 , 0)	=	18	and	λ_S (8 , 0)	=	1		δ_S (32 , 0)	=	18	and	λ_S (32 , 0)	=	1
δ_S (8 , 1)	=	27	and	λ_S (8 , 1)	=	11		δ_S (32 , 1)	=	27	and	λ_S (32 , 1)	=	11
δ_S (9 , 0)	=	3	and	λ_S (9 , 0)	=	5		δ_S (33 , 0)	=	3	and	λ_S (33 , 0)	=	5
δ_S (9 , 1)	=	36	and	λ_S (9 , 1)	=	7		δ_S (33 , 1)	=	36	and	λ_S (33 , 1)	=	7
δ_S (10 , 0)	=	3	and	λ_S (10 , 0)	=	2		δ_S (34 , 0)	=	3	and	λ_S (34 , 0)	=	2
δ_S (10 , 1)	=	24	and	λ_S (10 , 1)	=	9		δ_S (34 , 1)	=	24	and	λ_S (34 , 1)	=	9
δ_S (11 , 0)	=	18	and	λ_S (11 , 0)	=	1		δ_S (35 , 0)	=	18	and	λ_S (35 , 0)	=	1
δ_S (11 , 1)	=	37	and	λ_S (11 , 1)	=	11		δ_S (35 , 1)	=	37	and	λ_S (35 , 1)	=	11
δ_S (12 , 0)	=	9	and	λ_S (12 , 0)	=	1		δ_S (36 , 0)	=	9	and	λ_S (36 , 0)	=	1
δ_S (12 , 1)	=	47	and	λ_S (12 , 1)	=	10		δ_S (36 , 1)	=	47	and	λ_S (36 , 1)	=	10
δ_S (13 , 0)	=	14	and	λ_S (13 , 0)	=	2		δ_S (37 , 0)	=	14	and	λ_S (37 , 0)	=	2
δ_S (13 , 1)	=	34	and	λ_S (13 , 1)	=	11		δ_S (37 , 1)	=	34	and	λ_S (37 , 1)	=	11
δ_S (14 , 0)	=	15	and	λ_S (14 , 0)	=	1		δ_S (38 , 0)	=	15	and	λ_S (38 , 0)	=	1
δ_S (14 , 1)	=	40	and	λ_S (14 , 1)	=	10		δ_S (38 , 1)	=	40	and	λ_S (38 , 1)	=	10
δ_S (15 , 0)	=	14	and	λ_S (15 , 0)	=	1		δ_S (39 , 0)	=	14	and	λ_S (39 , 0)	=	1
δ_S (15 , 1)	=	33	and	λ_S (15 , 1)	=	10		δ_S (39 , 1)	=	33	and	λ_S (39 , 1)	=	10
δ_S (16 , 0)	=	15	and	λ_S (16 , 0)	=	1		δ_S (40 , 0)	=	15	and	λ_S (40 , 0)	=	1
δ_S (16 , 1)	=	44	and	λ_S (16 , 1)	=	10		δ_S (40 , 1)	=	44	and	λ_S (40 , 1)	=	10
δ_S (17 , 0)	=	18	and	λ_S (17 , 0)	=	1		δ_S (41 , 0)	=	18	and	λ_S (41 , 0)	=	1
δ_S (17 , 1)	=	43	and	λ_S (17 , 1)	=	10		δ_S (41 , 1)	=	43	and	λ_S (41 , 1)	=	10
δ_S (18 , 0)	=	2	and	λ_S (18 , 0)	=	5		δ_S (42 , 0)	=	2	and	λ_S (42 , 0)	=	5
δ_S (18 , 1)	=	41	and	λ_S (18 , 1)	=	10		δ_S (42 , 1)	=	41	and	λ_S (42 , 1)	=	10
δ_S (19 , 0)	=	17	and	λ_S (19 , 0)	=	4		δ_S (43 , 0)	=	17	and	λ_S (43 , 0)	=	4
δ_S (19 , 1)	=	41	and	λ_S (19 , 1)	=	7		δ_S (43 , 1)	=	41	and	λ_S (43 , 1)	=	7
δ_S (20 , 0)	=	21	and	λ_S (20 , 0)	=	1		δ_S (44 , 0)	=	21	and	λ_S (44 , 0)	=	1
δ_S (20 , 1)	=	46	and	λ_S (20 , 1)	=	10		δ_S (44 , 1)	=	46	and	λ_S (44 , 1)	=	10
δ_S (21 , 0)	=	9	and	λ_S (21 , 0)	=	5		δ_S (45 , 0)	=	9	and	λ_S (45 , 0)	=	5
δ_S (21 , 1)	=	44	and	λ_S (21 , 1)	=	10		δ_S (45 , 1)	=	44	and	λ_S (45 , 1)	=	10
δ_S (22 , 0)	=	20	and	λ_S (22 , 0)	=	4		δ_S (46 , 0)	=	20	and	λ_S (46 , 0)	=	4
δ_S (22 , 1)	=	44	and	λ_S (22 , 1)	=	7		δ_S (46 , 1)	=	44	and	λ_S (46 , 1)	=	7
δ_S (23 , 0)	=	12	and	λ_S (23 , 0)	=	1		δ_S (47 , 0)	=	12	and	λ_S (47 , 0)	=	1
δ_S (23 , 1)	=	26	and	λ_S (23 , 1)	=	8		δ_S (47 , 1)	=	26	and	λ_S (47 , 1)	=	8

Fig. 5. The split-state split-output Mealy machine S, from M, above. Again we have encoded pairs into integers. The encoding takes (q, σ) into $q + |\Sigma| * \sigma$.

then

$$\delta_M(q, \sigma) = q'$$

and

$$\lambda_M(q, \sigma) = \nu(\sigma', d)$$

where

$$\nu(x, y) = x|M| + y,$$

i.e., the pair (x, y) is seen as a number in base $|D|$ (which is 3 in this case). Next the *split-state split-output* Mealy machine

$$M' = (\Sigma_T, \Gamma', Q', \delta', \lambda', q'_I)$$

from M is obtained, where

- $\Gamma' = \Gamma \times \Sigma$, $Q' = Q \times \Sigma$, and s'_I is any of the $(s_I, \sigma), \sigma \in \Sigma$;
- $\delta'((q, \sigma), \sigma') = (\delta_M(q, \sigma'), \sigma')$ and

δ_R (0 , 0)	=	1	and	λ_R (0 , 0)	=	1	δ_R (16 , 1)	=	18	and	λ_R (16 , 1)	=	10
δ_R (0 , 1)	=	8	and	λ_R (0 , 1)	=	10	δ_R (17 , 0)	=	4	and	λ_R (17 , 0)	=	5
δ_R (1 , 0)	=	2	and	λ_R (1 , 0)	=	4	δ_R (17 , 1)	=	16	and	λ_R (17 , 1)	=	7
δ_R (1 , 1)	=	29	and	λ_R (1 , 1)	=	8	δ_R (18 , 0)	=	19	and	λ_R (18 , 0)	=	1
δ_R (2 , 0)	=	0	and	λ_R (2 , 0)	=	4	δ_R (18 , 1)	=	3	and	λ_R (18 , 1)	=	8
δ_R (2 , 1)	=	3	and	λ_R (2 , 1)	=	11	δ_R (19 , 0)	=	17	and	λ_R (19 , 0)	=	1
δ_R (3 , 0)	=	4	and	λ_R (3 , 0)	=	2	δ_R (19 , 1)	=	18	and	λ_R (19 , 1)	=	10
δ_R (3 , 1)	=	8	and	λ_R (3 , 1)	=	8	δ_R (20 , 0)	=	14	and	λ_R (20 , 0)	=	1
δ_R (4 , 0)	=	5	and	λ_R (4 , 0)	=	5	δ_R (20 , 1)	=	21	and	λ_R (20 , 1)	=	10
δ_R (4 , 1)	=	27	and	λ_R (4 , 1)	=	8	δ_R (21 , 0)	=	22	and	λ_R (21 , 0)	=	1
δ_R (5 , 0)	=	6	and	λ_R (5 , 0)	=	1	δ_R (21 , 1)	=	23	and	λ_R (21 , 1)	=	10
δ_R (5 , 1)	=	12	and	λ_R (5 , 1)	=	11	δ_R (22 , 0)	=	17	and	λ_R (22 , 0)	=	5
δ_R (6 , 0)	=	7	and	λ_R (6 , 0)	=	5	δ_R (22 , 1)	=	21	and	λ_R (22 , 1)	=	10
δ_R (6 , 1)	=	9	and	λ_R (6 , 1)	=	10	δ_R (23 , 0)	=	24	and	λ_R (23 , 0)	=	4
δ_R (7 , 0)	=	4	and	λ_R (7 , 0)	=	2	δ_R (23 , 1)	=	21	and	λ_R (23 , 1)	=	7
δ_R (7 , 1)	=	8	and	λ_R (7 , 1)	=	8	δ_R (24 , 0)	=	22	and	λ_R (24 , 0)	=	1
δ_R (8 , 0)	=	0	and	λ_R (8 , 0)	=	4	δ_R (24 , 1)	=	23	and	λ_R (24 , 1)	=	10
δ_R (8 , 1)	=	3	and	λ_R (8 , 1)	=	11	δ_R (25 , 0)	=	4	and	λ_R (25 , 0)	=	2
δ_R (9 , 0)	=	6	and	λ_R (9 , 0)	=	1	δ_R (25 , 1)	=	26	and	λ_R (25 , 1)	=	9
δ_R (9 , 1)	=	10	and	λ_R (9 , 1)	=	10	δ_R (26 , 0)	=	1	and	λ_R (26 , 0)	=	1
δ_R (10 , 0)	=	11	and	λ_R (10 , 0)	=	4	δ_R (26 , 1)	=	8	and	λ_R (26 , 1)	=	10
δ_R (10 , 1)	=	9	and	λ_R (10 , 1)	=	7	δ_R (27 , 0)	=	6	and	λ_R (27 , 0)	=	1
δ_R (11 , 0)	=	6	and	λ_R (11 , 0)	=	1	δ_R (27 , 1)	=	28	and	λ_R (27 , 1)	=	11
δ_R (11 , 1)	=	10	and	λ_R (11 , 1)	=	10	δ_R (28 , 0)	=	5	and	λ_R (28 , 0)	=	5
δ_R (12 , 0)	=	13	and	λ_R (12 , 0)	=	2	δ_R (28 , 1)	=	27	and	λ_R (28 , 1)	=	8
δ_R (12 , 1)	=	25	and	λ_R (12 , 1)	=	11	δ_R (29 , 0)	=	30	and	λ_R (29 , 0)	=	2
δ_R (13 , 0)	=	14	and	λ_R (13 , 0)	=	1	δ_R (29 , 1)	=	32	and	λ_R (29 , 1)	=	11
δ_R (13 , 1)	=	20	and	λ_R (13 , 1)	=	10	δ_R (30 , 0)	=	31	and	λ_R (30 , 0)	=	2
δ_R (14 , 0)	=	13	and	λ_R (14 , 0)	=	1	δ_R (30 , 1)	=	29	and	λ_R (30 , 1)	=	8
δ_R (14 , 1)	=	15	and	λ_R (14 , 1)	=	10	δ_R (31 , 0)	=	30	and	λ_R (31 , 0)	=	2
δ_R (15 , 0)	=	4	and	λ_R (15 , 0)	=	5	δ_R (31 , 1)	=	8	and	λ_R (31 , 1)	=	10
δ_R (15 , 1)	=	16	and	λ_R (15 , 1)	=	7	δ_R (32 , 0)	=	31	and	λ_R (32 , 0)	=	2
δ_R (16 , 0)	=	17	and	λ_R (16 , 0)	=	1	δ_R (32 , 1)	=	29	and	λ_R (32 , 1)	=	8

Fig. 6. The Mealy machine R obtained from S above removing the unreachable states.

– $\lambda'((q,\sigma),\sigma') = (\lambda(q,\sigma'),\sigma')$.

M' is then implemented using the procedure shown in [5]. The number of units is calculated as in Remark 5.3.

7. TMs with finite memory

Motivated by our previous results on the relationship between classical and neural computation, and by the results in [25,26] on the limitation of the capability of analog neural computation, in this section we propose two novel classes of Turing machines by restricting the type of finite control of the machine. The Definite Turing Machines and Finite-Memory Turing Machines. As will be seen in the next sections, the latter is equivalent to the classical Turing machine. For the former, we show that such a class of TM is capable of computing the class of simple function, giving the first steps in order to characterize its actual computational power.

7.1. Definite TMs and simple functions

Definition 7.1 *A Turing machine is $k-$definite if, and only if, its transition system is.* □

There are various algorithms for deciding if a given transition system is definite or not. We use one based on *testing table* and *testing graph* presented in [21]. The testing table has $p = |\Sigma|$ columns, one for each symbol in the input alphabet. Its rows are divided into two parts, the upper part correspond to the states of the machine, and the table entries are the state transitions. The row headings in the lower part of the table are all unordered pairs of non equal sates, while the table entries are the corresponding pair of state transitions. The testing graph is a directed graph which has as vertices the row headings in the lower part of the testing table and there is an edge from (p,s), $p,s \in Q$, to (p',s'), $p' \neq s'$, if, and only if, there is an entry (p',s') in row (p,s) column $\sigma \in \Sigma$. The edge is labelled σ. No edge if (p,s) implies (p',p'). The transition system

δ(0 , 0) = (0 , 0, 1)	δ(0 , 1) = (1 , 1, 2)
δ(1 , 0) = (2 , 1, 1)	δ(1 , 1) = (1 , 1, 2)
δ(2 , 0) = (10 , 0, 1)	δ(2 , 1) = (3 , 0, 1)
δ(3 , 0) = (4 , 0, 1)	δ(3 , 1) = (3 , 1, 1)
δ(4 , 0) = (4 , 0, 1)	δ(4 , 1) = (5 , 0, 1)
δ(5 , 0) = (7 , 0, 2)	δ(5 , 1) = (6 , 1, 2)
δ(6 , 0) = (6 , 0, 2)	δ(6 , 1) = (1 , 1, 2)
δ(7 , 0) = (7 , 0, 2)	δ(7 , 1) = (8 , 1, 2)
δ(8 , 0) = (9 , 0, 2)	δ(8 , 1) = (8 , 1, 2)
δ(9 , 0) = (2 , 0, 1)	δ(9 , 1) = (1 , 1, 2)
δ(10 , 0) = (11 , 0, 1)	δ(10 , 1) = (10 , 1, 1)
δ(11 , 0) = (11 , 0, 2)	δ(11 , 1) = (11 , 1, 2)

Fig. 7. A non definite Turing machine T that finds the gcd of two number in unary notation using the Euclid's algorithm.

δ(0 , 0) = (9 , 0, 0)	δ(0 , 1) = (1 , 1, 1)
δ(1 , 0) = (7 , 0, 2)	δ(1 , 1) = (2 , 0, 1)
δ(2 , 0) = (7 , 0, 2)	δ(2 , 1) = (3 , 0, 1)
δ(3 , 0) = (9 , 0, 0)	δ(3 , 1) = (4 , 1, 2)
δ(4 , 0) = (4 , 0, 2)	δ(4 , 1) = (5 , 1, 1)
δ(5 , 0) = (8 , 1, 1)	δ(5 , 1) = (6 , 1, 0)
δ(6 , 0) = (6 , 0, 1)	δ(6 , 1) = (6 , 1, 1)
δ(7 , 0) = (7 , 0, 2)	δ(7 , 1) = (9 , 0, 0)
δ(8 , 0) = (8 , 0, 1)	δ(8 , 1) = (1 , 0, 1)
δ(9 , 0) = (9 , 0, 0)	δ(9 , 1) = (9 , 1, 0)

Fig. 8. A definite Turing machine T that divides a number in unary notation by 3.

is $k-$definite if, and only if, its testing graph G is loop free and the length of the longest path in G is $k-1$.

A non-definite TM is shown in Fig. 7, while in Fig. 8 a definite one is presented.

In Computing Theory, on the class of primitive recursive functions we can define a hierarchy of classes of functions $\mathcal{L}_0 \subset \mathcal{L}_1 \subset \mathcal{L}_2 \cdots x_0 \subset x_1 \subset x_2$ of increasing complexity (e.g., see [4]). They are related to programs having a maximum depth of nesting FOR statements in a toy programming language. $\mathcal{L}_i$ is the class of functions computed by programs having a maximum depth of nesting i FOR statements. One should note that the whole of the primitive recursive functions can be computed by programs with no GOTO statements (see Theorem 3.3 page 53 in [4]). Two classes in this hierarchy are particularly interesting. $\mathcal{L}_1$, the class of *simple functions*, and $\mathcal{L}_2$, the *elementary functions*. For $\mathcal{L}_1$, the equivalence of programs is decidable while for all $i \geqslant 2$ the equivalence is undecidable. $\mathcal{L}_2$ is claimed to contain all real problems since a program for a not elementary function say $g(x)$ must, for any k, use more than

$$f_2^k(x) = 2^{2^{\cdot^{\cdot^{\cdot^{2^x}}}}} \Bigg\} k$$

steps on infinitely many inputs. For this reason the elementary functions are also called *practical computable functions*

In order to show that definite TM computes the class of simple functions, we need the following characterization of the class $\mathcal{L}_1$ (see [4]).

Theorem 7.1 *The class of simple function is the smallest class which contains the basic simple functions*

$$\begin{aligned}
\mathsf{d}(x) &= (x, x) \\
\mathsf{e}(x, y) &= (y, x) \\
\mathsf{i}(x) &= x \\
\mathsf{p}(x) &= () \\
\mathsf{z}() &= (x) \\
\mathsf{s}(x) &= x + 1 \\
& x + y \\
& x \dot{-} 1 \\
& x/k && \text{for each constant } k \geqslant 1 \\
& \mathsf{mod}(x, k) && \text{for each constant } k \geqslant 1 \\
\mathsf{a}(x, y) = \neg x \to y &= \begin{cases} y \text{ if } x = 0 \\ 0 \text{ if } x > 0 \end{cases}
\end{aligned}$$

and which is closed under composition and combination. □

The Euclidean algorithm shows that the $\gcd(x, y)$ is an elementary function and our Fig. 7 shows a non definite TM for it. It remains to be shown that it is impossible to find a definite TM for the Euclidean algorithm in particular and the elementary functions in general.

In contrast, for the simple functions the situation is straightforward.

Theorem 7.2 *The class $\mathcal{L}_1$ of simple function is computable by definite Turing machines* □

proof: We use the characterization in Theorem 7.1. The less trivial case in the list of basic simple function are those which involves division: x/k and $\mathsf{mod}(x, k)$. But already in Fig. 8 we show a definite TM for division by 3, which can obviously be extended for any fixed $k \geqslant 1$. The TM for $\mathsf{mod}(x, k)$ is a submachine (in the graph theoretical sense) of the TM for x/k and a definite transition system does not have non definite subsystem.

The operations of composition and combinations are nothing but the serial and parallel operations, respectively, in [43], where is proved that they preserve definiteness.

7.2. Finite memory and TMs

For Mealy machines there is another notion of finite memory which are also related to the output and not only to the input as in the definite case. In a $k-$definite automaton the present state is completely determined by the last k inputs.

Definition 7.2 *A Mealy machine M is finite-memory machine of order k if k is the least integer, so that the present state of M can be determined uniquely from the knowledge of the last k inputs and the corresponding k outputs.* □

Definition 7.3 *A Turing machine is finite-memory if and only if, its associated Mealy machine if finite-memory.* □

In contrast to the definite machines, which does not recognize all regular language, finite-memory machines are computationally capable of computing all regular languages.

Theorem 7.3 *For any regular language L there exists a finite-memory machine M which recognizes L.* □

Proof: Let $N = (\Sigma, Q, \delta, q_I, F)$ be the finite state automaton recognizing L. Define the machine $M = (\Sigma, \Gamma, \delta, q_I)$, where $\Gamma = Q$. M is obviously finite-memory of order 1 and the language represented by the subset $F \subseteq \Gamma$ of output letters is equal to L.

Corollary 7.1 *Turing machines and finite-memory Turing machines are computationally equivalent.* □

8. Conclusions

Proofs of equivalence between Turing machines and artificial neural networks are important in that they allow, at least from a theoretical point of view, for the use of neural networks in cognitive tasks such as natural language processing and logical or common reasoning. However, such simulations often assume an unbounded number of weights or nodes and are not realistically implementable. Furthermore, Maass and Sontag [26] have proved that any analog neural network whose computational units are subject to unbounded Gaussian noise or other common noise distributions cannot recognize arbitrary regular languages. In fact, these networks will be able to recognize only definite regular languages.

We have proposed an alternative point of view in how to simulate a Turing machine by actually implementing the finite control of the machine and leaving the machine's tape out as an external non-intrinsic feature. As a result, the simulation is conceptually simpler and has more cognitive/biological appeal than the previous works. Besides, from results on the actual limits on the measurement of analogical data and on bounds on the information capacity of any physical system our proposal is physical realizable in contrast to the other approach.

By focusing on the implementation of the finite control of a TM we are thus led to the huge amount of work on finite state automata implementation on neural networks and *learning*, a subject not tackled in this work but which obviously is a next step.

Another contribution of this work is the presentation of two novel kind of Turing Machines: definite and finite memory Turing machine. We proved that the latter is computationally equivalent to (unrestricted) Turing machines. With respect to the former, we showed that they can compute simple functions (in the class of primitive recursive functions), but the complete characterization of their computational power is left as further work.

References

[1] N. Alon, A.K. Dewdney and T.J. Ott, Efficient simulation of finite automata by neural nets, *Journal of the Association for Computing Machinery* **38**(2) (1991), 495–514.

[2] M. Arbib, *Brains, Machines and Mathematics*, New York: McGraw-Hill. 152 + xiv pages, 1964.

[3] S. Bains, J. Johnson, Noise, Physics, and non-Turing computation, Technical report, Open University, Dep. of Design and Innovations, 2000.

[4] W. Brainerd and L. Landweber, *Theory of Computation*. Jonh Wiley and Sons, 1974.

[5] R.C. Carrasco, M.L. Forcada, M.A. Valdés-Muñoz and R.P. Neco, Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. *Neural Computation* **12**(9) (2000), 2129–2174.

[6] W.R. de Oliveira and T.B. Ludermir, Turing machine simulation by logical neural networks, in: *Artificial Neural Networks,2*, I. Aleksander and J. Taylor, eds, Netherlands: North-Holland, 2000, pp. 663–667.

[7] J.L. Elman, Finding structure in time, *Cognitive Science* **14** (1990), 179–211.

[8] M. Forcada, Neural Networks: Automata and Formal Models of Computation. Eletronic Manuscript, http://www.dlsi.ua.es/mlf/nnafmc/, 2001.

[9] S. Franklin and M. Garzon, Neural computability. *Progress in Neural Networks* **1** (1990), 128,144. Ablex, Norwood.

[10] H. Gardner, *The minds new science: a history of the cognition revolution*, Basic Books, New York, 1985.

[11] M.W. Goudreau, C.L. Giles, S.T. Chakradhar and D. Chen, First-order vs. second-order single layer recurrent neural networks. *IEEE Transactions on Neural Networks* **5**(3) (1994), 511–513.

[12] J. Greeno and J. Moore, Situativity and symbols: response to Vera and Simon, *Cognitive Science* **17**(1) (1993), 7–48.

[13] R. Hartley and H. Szu, A comparison of the computational power of neural network models, in: *IEEE First International Conference on Neural Networks*, M. Caudill and C. Butler, eds, San Diego, June, 1987, pp. 15–22.

[14] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Reading, MA: Addison-Wesley Publishing Company, Inc, 1979.

[15] J.J. Hopfield, Neural networks and physical systems with emergent collective properties, *Proc. Nat. Acad. Sci.* **79** (1982), 2554–2558.

[16] P. Indyk, *Optimal simulation of automata by neural nets*, in Proceedings of 12th STACS'95 Annual Symposioum on Theoretical Aspects of Computer Science, Vol. LNCS 900, Berlin, Springer-Verlag. 1995, pp. 337–348.

[17] M. Jordan, A parallel distributed processing approach. Technical Report 8604, La Jolla, CA: Institute for Cognitive Science, University of California, San Diego. Technical Report, 1986.

[18] J. Kilian and H. Siegelmann, *On the power of sigmoidal neural networks*, in Proc. Sixth ACM Workshop on Computational Learning Theory, Santa Cruz, July, 1993.

[19] J. Kilian and H. Siegelmann, The dynamic universality of sigmoidal neural networks, *Information and Computation* **37**(1) (1996), 48–56.

[20] S.C. Kleene, Representation of events in nerve nets and finite automata, in: *Automata Studies*, C. Shannon and J. McCarthy, eds, Number 34 in Annals of Mathematics Studies, Princeton, New Jersey: Princeton University Press, 1956.

[21] Z. Kohavi, *Switching and Finite Automata Theory* (second ed.), New York, NY: McGraw-Hill, Inc, 1978.

[22] T. Kohonen, Correlation matrix memories, *IEEE Transactions on Computers* **c-21** (1972), 353–359.

[23] P. Koiran, Dynamics of discrete time, continuous state Hopfield networks, *Neural Computation 6*(3) (1994), 459–468.

[24] P. Koiran, A family of universal recurrent networks *Theoretical Computer Science* **168**(2) (1996), 473–480.

[25] W. Maass and P. Orponen, On the efect of analog noise on discrete-time analog computations, *Neural Computation* **10**(5) (1998), 1071–1095.

[26] W. Maass and E.D. Sontag, Analog neural nets with Gaussian or other common noise distributions cannot recognize arbitrary regular languages, *Neural Computation* **11** (1997), 771–782.

[27] W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics* **5** (1943), 115–137.

[28] M. Minsky, Some universal elements for finite automata, in: *Automata Studies*, C. Shannon and J. McCarthy, eds, Number 34 in Annals of Mathematics Studies, Princeton, New Jersey: Princeton University Press, 1956.

[29] M. Minsky, *Computation: Finite and Infinite Machines*, Englewood Cliffs, NJ: Prentice-Hall Inc, 1967.

[30] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, Massachusetts: MIT Press, 1969.

[31] J.B. Pollack, *On Connectionist Models of Natural Language Processing*, Tese de Doutorado, Computer Science Department, University of Illinois, Urbana. Available as TR MCCS-87-100, Computing Research Laboratory, New Mexico State University, Las Cruces, NM, 1987.

[32] J.B. Pollack, Recursive distributed representations. *Artificial Intelligence* **46** (1990), 77–105.

[33] M. Rabin and D. Scott, Finite automata and their decision problems. Technical report, *IBM J. Res. Devel* (1959).

[34] Y. Rogozhin, Small universal Turing machines, *Theoretical Computer Science* **168** (1996), 215–240.

[35] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, New York: Spartan Books, 1962.

[36] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, in: *Parallel Distributed Processing*, Vol. 1, Foundations, Cambridge, MA: The MIT Press, 1986, pp. 318–362.

[37] D. Scott, Some definitional suggestions for automata theory. *J. Computer and System Sciences* **1**(2) (1967), 187–212.

[38] H. Siegelmann and E. Sontag, Analog computation via neural networks. *Theoretical Computer Science* **131** (1994), 331–360.

[39] H.T. Siegelmann, B.G. Horne and C.L. Giles, Computational capabilities of recurrent NARX neural networks, *IEEE Trans. on Systems, Man and Cybernetics – PART B – Cybernetics* **27**(2) (1997), 208–215.

[40] H.T. Siegelmann and E.D. Sontag, Turing computability with neural nets. *Applied Mathematics Letters* **4**(6) (1991), 77–80.

[41] H.T. Siegelmann and E.D. Sontag, On the computational power of neural nets. *Journal of Computer and System Sciences* **50**(1) (1995), 132–150.

[42] Special Issue: Situated Action, Cognitive science, vol. 17, 1993.

[43] M. Steinby, On definite automata and related systems, *Ann. Acad. Sci. Fenn.* **Series A I** (1969).

[44] G.Z. Sun, H.H. Chen, Y.C. Lee and C.L. Giles, Turing equivalence of neural networks with second order connection weights, in 1991 IEEE INNS International Joint Conference on Neural Networks – Seattle, Vol II, Piscataway, NJ, IEEE Press, 1991, pp. 357–362.

[45] A. Vera and H. Simon, Situated action: a symbolic interpretation, *Cognitive Science* **17**(1) (1993), 7–48.

[46] J. Šíma, Hopfield languages, in: Proc. 22nd SOFSEM95 Seminar on Current Trends in Theory and Practice of Informatics, Vol. LNCS 1012, Milovy, Czech Republic, Berlin:Springer-Verlag, 1995, pp. 461–468.

[47] A. Wells, Turing's analysis of computation and theories of cognit-ve architecture, *Cognitive Science* **22**(3) (1998), 269–294.

[48] P.J. Werbos, *Beyond Regression: New Tools for Prediction and Analkysis in the Behavioural Sciences*, Doctoral dissertation, Applied Mathematics, Havard University, Boston, MA, 1974.