ELSEVIER

Theoretical Computer Science 168 (1996) 473–480

Theoretical Computer Science

A family of universal recurrent networks

Pascal Koiran *

LIP, ENS Lyon – CNRS, 46, allée d'Italie, 69364 Lyon Cedex 07, France

Abstract

Hava Siegelmann and Eduardo Sontag have shown that recurrent neural networks using the linear-bounded sigmoid are computationally universal. We show that this remains true if the linear-bounded sigmoid is replaced by any function in a fairly large class.

1. Introduction

Siegelmann and Sontag have shown that recurrent neural networks are computationally universal, i.e., can simulate an arbitrary Turing machine [6]. Their result holds only for networks of "neurons" using the so-called *linear-bounded sigmoid* as output function. In this paper, we generalize it to a large class of output functions.

Our class $\mathcal{U}$ is made of those functions ϕ that are equal to σ for inputs of sufficiently large magnitude, and are non-constant (and sufficiently smooth) on some arbitrary interval I. (Recall that $\sigma(x) = 0$ if $x \leqslant 0$, $\sigma(x) = x$ if $0 \leqslant x \leqslant 1$, and $\sigma(x) = 1$ for $x \geqslant 1$.) The idea of the proof is to approximate the linear part of σ by $\phi_{|I}$, which makes it possible to simulate finite automata with two unary stacks. It is well-known that these machines are universal [1]. The main drawback of this unary encoding (which was used by Siegelmann and Sontag in an older paper [4]) is its exponential slowdown with respect to the Turing machine model. In the final paper [6] these authors used binary stacks, which made it possible to simulate Turing machines in linear time. Unfortunately, binary stacks seem to be ill-suited to our simulation technique.

The linear-bounded sigmoid σ is very convenient for simulations of pushdown automata since the three operations push – pop – no-op can be implemented exactly, without any "rounding error". With an arbitrary ϕ, these operations can be implemented only approximately, and the approximation error is fixed for any given network. The main

* Tel.: +33 72 72 80 00; fax: +33 72 72 80 80; e-mail: koiran@lip.ens-lyon.fr.

PII S0304-3975(96)00088-6

technical difficulty is that errors accumulate as the computation progresses. This may endanger the correctness of the simulation. In order to solve this problem, we duplicate each stack, and the memory is constantly moved from one stack to its sibling. This helps because one can suppress noise from an empty stack.

It is perhaps interesting to note that a somewhat similar technique is used in everyday computers: certain types of RAM chips have to be read and written constantly because of electric "leaks" that would otherwise corrupt their content.

In contrast, implementing the control part of stack automata is not difficult: state transitions can be realized exactly since $\phi = \sigma$ at infinity. The details can be found in Section 4.2.

The simulation result is stated precisely in Section 2 and proved in the following sections.

2. Universal networks

We consider discrete-time recurrent networks with external inputs. All units (or *neurons*) use the same output function ϕ. The dynamics of a ϕ-network is described by the following equation:

$$x_i(t+1) = \phi\left(\sum_{j=1}^{n} w_{ij}x_j(t) + a_iE(t) - \theta_i\right).$$

Here $x_i(t)$ is the state of unit i at time t. The parameters w_{ij} and a_i are called *weights*, θ_i is a *threshold*. The input line E carries binary inputs: $\forall t \in \mathbb{N}$, $E(t) \in \{0,1\}$. We shall see that computationally universal ϕ-networks exist when ϕ belongs to a class $\mathscr{U}$ defined as follows.

Definition 1. A real function ϕ is in $\mathscr{U}$ if the two following conditions are satisfied:
- $\phi(x) = 0$ when $x \leqslant 0$, and $\phi(x) = 1$ when $x \geqslant 1$;
- there exists an open interval I such that ϕ is C^2 on I, and $\forall x \in I$, $\phi'(x) \neq 0$.

With these networks one can compute partial functions from $\mathbb{N}$ into $\{0,1\}$ as follows. The initial state of all units is 0, except for one designated unit whose initial state is 1. The input $n \in \mathbb{N}$ is written on the input line E as follows: $E(t) = 1$ if $0 \leqslant t \leqslant 4n-1$, $E(t) = 0$ if $t \geqslant 4n$. The output can be read on two designated units, say, x_0 (the validation unit) and x_1 (the output unit). It is required that $x_0(4t) = 1$ for at most one t. Then the output is by definition $x_1(4t)$, and it is required to be in $\{0,1\}$. If there is no such t, the output is undefined.

The occurrence of the term $4t$ is due to the fact that we construct networks with 4 layers, so that 4 units of time are needed to simulate one transition of a pushdown automaton. For this reason, the only data carried by the input line that actually matter are those that occur at times of the form $4t$.

Theorem 1. *For every partial recursive function $f : \mathbb{N} \to \{0,1\}$ and every $\phi \in \mathcal{U}$ there exists a ϕ-network $\mathcal{N}_{\phi,f}$ which computes f with the input–output conventions defined above.*

It is not hard to see that the first requirement on ϕ can be somewhat relaxed: it is sufficient to assume that for some $A > 0$, $\phi(x) = 0$ when $x \leqslant -A$ and $\phi(x) = 1$ when $x \geqslant A$. Similarly, the values 0 and 1 can be replaced by any two distinct numbers, if the input-output conventions are changed accordingly.

Theorem 1 is not merely an existence result: the proof is based on a constructive, step by step simulation of a Turing machine computing f. In particular, we do *not* encode function values in the digits of a single real number as in [5] (anyway, it is not clear whether the same trick is possible with an arbitrary $\phi \in \mathcal{U}$). The weights and thresholds of $\mathcal{N}_{\phi,f}$ can be taken to be rational numbers, or algebraic expressions involving rational numbers, ϕ and its derivatives. This property rules out a similar trick in Theorem 1.

3. The stacks

In this section we show how stack operations can be implemented. Sufficient conditions for implementing these operations by function iteration are established in Section 3.1. We show in Section 3.2 that these constraints can be satisfied by sigmoidal functions. As a matter of fact, we are not going to simulate arbitrary sequences of stack operations, but only sequences of n pushes followed by n pops. It is shown in Section 4 that this is enough to simulate arbitrary Turing machines.

3.1. Pushing and popping

In the following, f should be viewed as an approximate pushing or popping operation, and l as the exact operation.

Lemma 1. *Let l and f be two real functions such that the following condition holds on an interval $[x_\infty, x_m]$ where $x_m > x_\infty$:*

1. *$l(x) = a(x - x_\infty) + x_\infty$ with $a > 0$, $a \neq 1$.*
2. *$l(x) \geqslant f(x) \geqslant l(x) - M(x - x_\infty)^2$, with $M \geqslant 0$.*

Let $s_n = f^n(s_0)$, $y_n = l^n(y_0)$, where $s_0 \leqslant y_0$. Assume also that $s_p, y_p \in [x_\infty, x_m]$ for $0 \leqslant p \leqslant n-1$. The following relation holds:

$$|s_n - y_n| \leqslant (|s_0 - y_0| - E)\, a^n + Ea^{2n} \tag{1}$$

where $E = M(y_0 - x_\infty)^2/(a^2 - a)$.

Proof. By induction. The base case $n = 0$ is clear. By the triangle inequality,

$$|s_{n+1} - y_{n+1}| = |f(s_n) - l(y_n)| \leqslant |f(s_n) - l(s_n)| + |l(s_n) - l(y_n)|.$$

Hence by condition 2 in the lemma,

$$|s_{n+1} - y_{n+1}| \leqslant M(s_n - x_\infty)^2 + a|s_n - y_n| \leqslant M(y_n - x_\infty)^2 + a|s_n - y_n|.$$

The latter inequality holds because $x_\infty \leqslant s_n \leqslant y_n$ by condition 2. Assume the result true at step n. Since $|y_n - x_\infty| = |y_0 - x_\infty| a^n$,

$$|s_{n+1} - y_{n+1}| \leqslant a[(|s_0 - y_0| - E)\, a^n + Ea^{2n}] + M(y_0 - x_\infty)^2 a^{2n}.$$

The result at step $n+1$ follows from the relation $Ea + M(y_0 - x_\infty)^2 = Ea^2$. □

This result will now be applied twice in a row, first to push n elements with $(f, l) = (f_1, l_1)$, then pop them with $(f, l) = (f_2, l_2)$. Let $l_1(y) = (y + x_\infty)/2$, $l_2(y) = l_1^{-1}(y) = 2y - x_\infty$, and $x_0 \in]x_\infty, x_m]$. The states corresponding to n approximate or exact push operations are $s_{1,n} = f_1^n(x_0)$ or $y_{1,n} = l_1^n(x_0)$, respectively. The corresponding sequences of states when these n elements are popped are $s_{2,p} = f_2^p(s_{1,n})$ and $y_{2,p} = l_2^p(y_{1,n})$, respectively $(0 \leqslant p \leqslant n)$.

According to (1),

$$\begin{cases} |s_{2,p} - y_{2,p}| \leqslant |s_{1,n} - y_{1,n}| 2^p + \dfrac{M}{2}(y_{1,n} - x_\infty)^2 2^{2p}, \\ |s_{1,n} - y_{1,n}| \leqslant \dfrac{4M}{2^n}(x_0 - x_\infty)^2. \end{cases}$$

Since $y_{1,n} - x_\infty = (x_0 - x_\infty)/2^n$,

$$|s_{2,p} - y_{2,p}| \leqslant 4M(x_0 - x_\infty)^2 2^{p-n} + \tfrac{1}{2}M(x_0 - x_\infty)^2 2^{2(p-n)}.$$

Since $p \leqslant n$, it follows that

$$|s_{2,p} - y_{2,p}| \leqslant \tfrac{9}{2}M(x_0 - x_\infty)^2.$$

Note that $y_{2,n-1} = y_{1,1} = (x_0 + x_\infty)/2$ and $y_{2,n} = y_{1,0} = x_0$. Hence, if M is fixed and x_0 is sufficiently close to x_∞, the following separation property holds:

$$\begin{cases} s_{2,p} \leqslant b_i = (2x_0 + x_\infty)/3 \quad \text{for } p < n, \\ s_{2,n} \geqslant b_s = (3x_0 + x_\infty)/4. \end{cases} \tag{2}$$

It is therefore easy to find out whether all elements pushed by applying f_1 have been popped by applying f_2 (case $p = n$), or if only some of them have (case $p < n$).

3.2. *Implementation with sigmoidal functions*

We make the following choices.

- The pushing and popping functions are of the form

$$f_i(x) = \phi(\alpha_i \phi(\beta_i \psi(x) + \gamma_i) + \delta_i) \quad (i = 1, 2)$$

where

$$\psi(x) = \phi(\alpha\phi(\beta x + \gamma) + \delta);$$

- $x_\infty = \phi(b)$, where $b \in I$;
- $x_0 = \phi(c)$, where $c \in I$.

It will become clear in Section 4 why the term $\psi(x)$ is useful: the f_i's will be computed by depth-4 circuits. Without this synchronization constraint one could just take $\psi(x) = x$.

Given b and the f_i's, one can take c so that $x_0 = \phi(c) > x_\infty$ and x_0 is close enough to x_∞ for (2) to hold (here we use the property $\phi'(b) \neq 0$). We now explain how b, x_m and the f_i's can be chosen.

Let us assume first that ϕ'' is not identically 0 on I, and let $b \in I$ be such that $\phi''(b) \neq 0$. For the conditions of Lemma 1 to be satisfied with the exact pushing and popping functions l_1 and l_2 defined in Section 3.1, it suffices that:

- $f_i(x_\infty) = x_\infty$.
- $f_i'(x_\infty) = d_i$, where $d_1 = \frac{1}{2}$ and $d_2 = 2$.
- $f_i''(x_\infty) < 0$.

The latter condition implies that if $x_m > x_\infty$ is sufficiently close to x_∞, condition 2 in Lemma 1 will hold for some $M > 0$.

Let us show first how the conditions $\psi(x_\infty) = x_\infty$, $\psi'(x_\infty) = 1$ and $\psi''(x_\infty) = 0$ can be enforced. For the first condition to be satisfied, we just have to choose α, β, γ, and δ so that $\beta x_\infty + \gamma = b$ and $\alpha x_\infty + \delta = b$. Hence,

$$\psi'(x_\infty) = \alpha\beta\phi'(b)^2$$

and

$$\psi''(x_\infty) = \alpha\beta^2\phi'(b)\phi''(b)[\alpha\phi'(b) + 1].$$

We thus take $\alpha = -1/\phi'(b)$ so that $\psi''(x_\infty) = 0$, then $\beta = 1/[\alpha\phi'(b)^2]$ so that $\psi'(x_\infty) = 1$ and finally $\gamma = b - \beta x_\infty$, $\delta = b - \alpha x_\infty$.

We also choose α_i, β_i, γ_i, and δ_i so that $\beta_i x_\infty + \gamma_i = b$ and $\alpha_i x_\infty + \delta_i = b$. Then $f_i(x_\infty) = x_\infty$, and it follows from the properties of ψ established above that

$$f_i'(x_\infty) = \alpha_i\beta_i\phi'(b)^2$$

and

$$f_i''(x_\infty) = \alpha_i\beta_i\phi'(b)\phi''(b)[\alpha_i\beta_i\phi'(b) + \beta_i].$$

We need $\alpha_i\beta_i = d_i/\phi'(b)^2$ in order to have $f_i'(x_\infty) = d_i$. One can obtain an arbitrary value for $f_i''(x_\infty)$, e.g., -1, by varying β_i and keeping the relation $\alpha_i\beta_i = d_i/\phi'(b)^2$. Finally, we take $\gamma_i = b - \beta_i x_\infty$ and $\delta_i = b - \alpha_i x_\infty$.

If $\phi'' = 0$ on I, ϕ is linear and non-constant on this interval. We leave it to the reader to check that the same setting of parameters works, with $M = 0$. In fact, binary stacks can be used in this case, resulting in a linear-time simulation of Turing machines as in [6].

4. Control and network structure

It is well known that finite automata with two unary stacks are universal [1] (simpler and more efficient simulations are possible if more stacks are available). We shall see that any automaton M with two (unary) stacks can be simulated by a 4-stack automaton M' of a special type, and that M' can be simulated by a recurrent network. This network begins its computation by pushing on a stack the data read on the input line E. The simulation of M' proper starts when the input has been completely read. Switching from the input reading phase to the simulation phase, and then to the output production phase (following the rules stated in Section 2) is rather straightforward, since we know how to simulate finite automata (if you are not convinced of this, wait until Section 4.2). Hence, we will only describe the simulation phase.

Our pushdown automata are slightly unusual because they are not equipped with a no-op instruction. Instead, there is a reset operation which empties the stack to which it is applied.

4.1. Control

The transition functions of M' are denoted θ_i, $i = 0,\ldots,4$ where

- $\theta_0 : S \times \{0,1\}^4 \to S$ is the state transition function. The state set of M' is S and in the last four components 0 stands for an empty stack, 1 for a non-empty stack. In the following these four components are denoted $e = (e_1,\ldots,e_4)$.
- Stack operations are specified by $\theta_i : S \times \{0,1\}^4 \to \{u,d,r\}$, $i = 1,\ldots,4$ where u stands for push, d for pop, and r for reset. The implementation of these operations on a recurrent network is described in Section 4.2.

Each stack P of M is represented by a pair (P_1,P_2) of stacks of M'. This automaton constantly move the content of P back and forth between P_1 and P_2: P is transferred from P_1 to P_2, then from P_2 to P_1 when P_1 is empty, then again from P_1 to P_2, etc. The point of this manipulation is that M' has to be simulated by a recurrent network, and by the separation property (2) a sequence of pushes followed by the same number of pops can be realized without errors piling up too much. Errors that have accumulated on a stack must be erased when it is empty. This is done with the reset operation r. From the point of view of M' this operation is useless, but it is essential for the correctness of the simulation of M' by a recurrent network.

We now go into the inner workings of M', without formalizing too much. In order to apply an operation to, e.g., P_1, we wait until this stack has been emptied (i.e., the content of P has been transferred completely on P_2). For a reset operation, we leave the stack empty for one more time step, and meanwhile an additional element is pushed on P_2. One can get rid of this element by popping it at the next time step. In order to simulate a push operation of M we can just push this element on P_1 instead of throwing it away. For a pop operation, we erase the first two elements of P_2 (meanwhile reset operations are applied to P_1). Finally, M' must maintain a set of stack operations of M (at most one per stack of M) to be carried out. The current transition of M is

completed only when this set is empty. The strategy described above can be applied in parallel to both stacks of M.

4.2. Network structure

We simulate M' by a 4-layer network. The output of the last layer is fed back to the first one. This first layer is the "visible" part of the network, i.e., it encodes the state and stacks of M'. It is essential that the network structure be strictly layered, i.e., without connections between distant layers (i.e., between layers 1 and 3). Otherwise, there would be a synchronization problem when the output of the last layer is fed back to the first one. The first layer can be broken up into two groups of units:

- a set $\{s_1, s_2, s_3, s_4\}$ of four units encoding the four stacks of M';
- a set U_C indexed by C of "state units" encoding the state of M': if M' is in state i at time t, $x_i(4t) = 1$ and $x_j(4t) = 0$ for $j \in C$, $j \neq i$.

The transition function of a stack unit is defined as follows:

$$s_i(t+4) = \phi(\alpha_1 \phi(\beta_1 \psi(s_i(t)) + B_{1,i}) + \alpha_2 \phi(\beta_2 \psi(s_i(t)) + B_{2,i}) + B_i).$$

The stack operation is determined by the inputs $B_{1,i}$, $B_{2,i}$ and B_i. By construction $s_i(t)$ belongs to a bounded interval, so there exists a constant $C > 0$ such that $\forall i, t$, $|\psi(s_i(t))| \leqslant C$. Recall also that $\phi(x) = 0$ when $x \leqslant 0$. In order to select the correct operation when M' is in state j (and thus $x_j = 1$ in the network), it is therefore sufficient for the inputs to satisfy the following conditions:

- if $\theta_i(j, e) = u$ then $B_{1,i} = \gamma_1$, $B_{2,i} = -C|\beta_2|$, $B_i = \delta_1$;
- if $\theta_i(j, e) = d$ then $B_{1,i} = -C|\beta_1|$, $B_{2,i} = \gamma_2$, $B_i = \delta_2$;
- if $\theta_i(j, e) = r$ then $B_{1,i} = -C|\beta_1|$, $B_{2,i} = -C|\beta_2|$, $B_i = c$.

The input $B_{1,i}$ can be implemented as follows:

$$B_{1,i} = \sum_{j \in C,\ e \in \{0,1\}^4} c_{i,j,e}\, \mathrm{AND}(x_j(t), \delta(s_1(t), e_1), \ldots, \delta(s_4(t), e_4)),$$

where $c_{i,j,e} = \gamma_1$ if $\theta_i(j, e) = u$, $c_{i,j,e} = -C|\beta_1|$ otherwise. The empty-stack test δ is defined by

$$\begin{cases} \delta(x, 0) = 1 & \text{if } x \geqslant b_s, \ 0 \quad \text{if } x \leqslant b_i, \\ \delta(x, 1) = 1 & \text{if } x \leqslant b_i, \ 0 \quad \text{if } x \geqslant b_s. \end{cases}$$

These tests can be implemented as follows:

$$\delta(x, 0) = \phi\left(\frac{x - b_i}{b_s - b_i}\right)$$

and

$$\delta(x, 1) = \phi\left(\frac{b_s - x}{b_s - b_i}\right).$$

A term $\mathrm{AND}(x, \delta_1, \ldots, \delta_4)$ can be implemented by

$$\phi(\phi(x) + \delta_1 + \cdots + \delta_4 - 5).$$

We use the same construction for $B_{2,i}$; For B_i, it is necessary to add redundant ϕ functions in order to have a depth-3 network.

The transition function of a state unit $i \in U_C$ is defined as follows:

$$x_i(t+4) = \phi\left(\sum_{(j,e)\in\theta_0^{-1}(i)} \mathrm{AND}(x_j(t), \delta(s_1(t), e_1), \ldots, \delta(s_4(t), e_4))\right).$$

This term can be transformed into a depth-4 network using the same techniques as for the stack units.

5. Final remarks

We leave the following question as an open problem: are the networks considered in this paper capable of simulating Turing machines with a polynomial (rather than exponential) slowdown?

One can also ask whether Turing machines can be simulated by iterations of analytic functions on $[0,1]^n$, even if we drop the requirement that this function be the transition function of a neural network[1] (this can be done on $\mathbb{R}$, see [3, Ch. 6]).

References

[1] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, Reading, MA, 1979).

[2] J. Killian and H.T. Siegelmann, On the power of sigmoid neural networks, in: *Proc. 6th ACM Workshop on Computational Learning Theory*, 1993.

[3] P. Koiran, Puissance de calcul des réseaux de neurones artificiels, Ph.D. Thesis, Ecole Normale Supérieure de Lyon, June 1993.

[4] H.T. Siegelmann and E.D. Sontag, Turing computation with neural nets, *Appl. Math. Lett.* **4** (1991) 77–80.

[5] H.T. Siegelmann and E.D. Sontag, Analog computation via neural networks, *Theoret. Comput. Sci.* **131** (1994) 331–360.

[6] H.T. Siegelmann and E.D. Sontag, On the computational power of neural nets, *J. Comput. System Sci.* **50** (1995) 132–150.

[1] In an extended abstract [2], Killian and Siegelmann claimed that networks using the hyperbolic tangent as output function are universal. However, the complete proof is not available at the time of writing.