Appl. Math. Lett. Vol. 0, No. 0, pp. 1–15, 1900
Printed in Great Britain.

0893-9659/00 $3.00 + 0.00

TURING COMPUTABILITY WITH NEURAL NETS*

Hava T. Siegelmann
Department of Computer Science
Rutgers University, New Brunswick, NJ 08903

Eduardo D. Sontag
Department of Mathematics
Rutgers University, New Brunswick, NJ 08903

(Received June 17, 1991)

Abstract.

This paper shows the existence of a finite neural network, made up of sigmoidal neurons, which simulates a universal Turing machine. It is composed of less than 10^5 synchronously evolving processors, interconnected linearly. High-order connections are *not* required.

1. Introduction

This paper addresses the question: What ultimate limitations, if any, are imposed by the use of neural nets as computing devices? In particular, and ignoring issues of training and practicality of implementation, one would like to know if every problem that can be solved by a digital computer is also solvable –in principle– using a net. This question has been asked before in the literature. Indeed, Jordan Pollack ([7]) showed that a certain recurrent net model –which he called a "neuring machine," for "neural Turing"– is universal. In his model, all neurons synchronously update their states according to a quadratic combination of past activation values. In general, one calls *high-order* nets those in which activations are combined using multiplications; see [11] for related work and many other references to such nets. Pollack left open the question of establishing if high-order connections are really necessary in order to achive universality; the feeling among people working in the area has been that they are. In contrast, we point out here that *standard linear connections* are indeed enough to construct networks that are computationally as powerful as any Turing Machine.

Note that at least since the classical work of McCulloch and Pitts in the 1940s, it has been clear how to simulate logic gates by networks of *threshold* (binary-valued) neurons, and hence how to obtain finite automata using such nets (see e.g. [1] for more recent work on that problem). One can simulate Turing machines if one allows a potentially unbounded number of neurons; see e.g. [4] for variations on this theme and relations to cellular automata. Since we insist on a fixed number of neurons, which does not increase during the computation, our problem is different.

*Supported in part by US Air Force Grant AFOSR-880235 and by Siemens Corporate Research.

2. Statement of Result

A (recursive) net is an arbitrary interconnection of N synchronously evolving processors. One of the processors, say the first, is singled out as the "output node" of the net, and there is an external input signal that feeds into every processor. Since finitely many threshold neurons cannot simulate more than finite automata behavior, continuous-valued neurons are required.

We model such a net as a dynamical system (with scalar inputs). At each instant, the state of this system is a vector $x(t) \in \mathbb{Q}^N$ of rational numbers, where the ith coordinate keeps track of the activation value of the ith processor. More precisely, we define a *processor net* $\mathcal{N}$ as having equations are of the form

$$x(t+1) \;=\; \vec{\sigma}(Ax(t) + bu(t) + c)\;, \quad t = 0, 1, 2 \ldots \tag{1}$$

(or simply $x^+ = \vec{\sigma}(Ax + bu + c)$ in shorthand notation). Here N is some positive integer, $A \in \mathbb{Q}^{N \times N}$, and $b, c \in \mathbb{Q}^N$, while $\vec{\sigma} : \mathbb{Q}^N \to \mathbb{Q}^N : (q_1, \ldots, q_N) \mapsto (\sigma(q_1), \ldots, \sigma(q_N))$, where σ is a simple "sigmoid," namely the saturated-linear function $\sigma(x) := 0$ if $x < 0$, $\sigma(x) := x$ if $0 \leq x \leq 1$, and $\sigma(x) := 1$ if $x > 1$.

Processor nets as above appear frequently in neural network studies, and their dynamic properties are of interest (see for instance [6]); the continuous-time analogue is also studied in the literature (see some comments in the concluding section below).

Given any infinite sequence of rational numbers $\omega = u(0), u(1), u(2), \ldots$ (thought of as external inputs), one defines the state at time t, for each integer $t \geq 0$, as the value obtained by recursively solving the equations (1) with initial condition $x(0) := 0$.

It is obvious that one can simulate a processor net with a Turing machine, as we took all values to be rational. Our main remark is that, conversely, any function computable by a Turing machine can be computed by a processor net. To state the precise result, we need to introduce the notion of a *unary input signal*: this is by definition any sequence $u(0), u(1), u(2), \ldots$ which consists of a string of n 1's (where $0 \leq n < \infty$) followed by an infinite string of 0's:

$$\omega[n] \;=\; \underbrace{11\ldots1}_{n}00\ldots\;.$$

Theorem. *Let $\phi : \mathbb{N} \to \mathbb{N}$ be any recursively computable partial function. Then, there exists a processor net $\mathcal{N}$ so that the following property holds. Pick any $n \in \mathbb{N}$, and consider the unary input signal $\omega[n]$. Starting from the zero (inactive) initial state, the first coordinates $x(0)_1, x(1)_1, x(2)_1, \ldots$ of the resulting states form a sequence of the following form:*

$$0\ldots0\,\underbrace{11\ldots1}_{m}00\ldots$$

where m may be zero or positive. Iff $\phi(n)$ is undefined, $m = 0$; otherwise $\phi(n) = m - 1$.

We next describe the main ideas of the proof; details can be found in the technical report [8]. As in the textbook approach of "counter machines," one can encode an infinite tape into real-valued activations; the only problem is how to do this while preserving a purely linear-interconnection architecture.

First of all, one starts with a realization of ϕ through a push-down automaton with three unary stacks; these are known to be sufficient to simulate all Turing machines. (It is equally possible to start with binary stacks, which is more satisfactory from a computational complexity point of view, but the construction becomes slightly more involved in that case.)

The control unit can be easily simulated by a net; this is basically the old automata result, but care must be taken in seeing that it is possible to let inputs enter additively rather than multiplicatively. The contents of each stack can be summarized by a natural number s, which can in turn be represented by the rational number with binary expansion $q_s = 0.1\ldots1$ (use s one's to represent the integer s). In this last representation, affine operations are sufficient: The stack "push" operation (increment counter) corresponds to $q_s \mapsto \frac{1}{2}q_s + \frac{1}{2}$, while "pop" (decrement) corresponds to $q_s \mapsto 2q_s - 1$. Reading a stack is straighforward: if q_s encodes the stack value, then $\sigma(2q_s) = 0$ if and only if the stack is empty, and $\sigma(2q_s) = 1$ otherwise. (A different encoding of a stack, as in [7], cannot be read in this simple manner.)

The critical point is to show that the whole design can be integrated (stack operations and state transitions gated by states of control unit and symbols at tops of stacks) without introducing high-order connections, that is, products. This is achieved basically by using negative values that act as "inhibitors" when fed into the activation function σ. As a preliminary step, one proves the following easy fact, which allows the expression of any function of the control-unit binary state variables x, the Boolean functions β obtained by reading stacks, and the actual stack values q, in terms only of sigmoids:

Lemma. *For each function $\beta : \{0,1\} \times \{0,1\} \times \{0,1\} \to \{0,1\}$, there exist eight vectors $v_1, v_2, \ldots, v_8 \in \mathbb{Q}^5$ and scalars $c_1, c_2, \ldots, c_8 \in \mathbb{Q}$ such that, for each $(a,b,d,x) \in \{0,1\}^4$ and each $q \in [0,1]$,*

$$\beta(a,b,d)xq = \sigma\left(\sum_{i=1}^{8} c_i\sigma(v_i \cdot \mu)\right) + \sigma\left(q - \sum_{i=1}^{8} c_i\sigma(v_i \cdot \mu) + 1\right) - 1\ ,$$

where $\mu = (1,a,b,d,x)$ and "$\cdot$" = dot product in $\mathbb{Q}^5$.

As an illustration, consider just the "no-op" and "pop" actions, and assume that there is given a binary control signal c (which is computed from the current states and stacks) so that the required effect is, on a stack having value q_s:

$$q_s^+ \ = \ \begin{cases} q_s & \text{if } c = 0 \\ 2q_s - 1 & \text{if } c = 1 \end{cases}$$

(and it is guaranteed that $q_s \neq 0$ in the second case, that is, one doesn't attempt to pop an empty stack). Then one may use the update:

$$q_s^+ \ := \ \sigma\left(\sigma(q_s) + \sigma(q_s + c - 1) - \sigma(c)\right)$$

(some of the σ's are redundant, but are needed in order to obtain the desired form).

Note that in particular it follows that one can obtain the behavior of a universal Turing machine via some net. A rough bound from the constructions shows that $N = 10^5$ processors are (far more than) sufficient for computing such a function.

3. Remarks

Note that the simulation result has many interesting consequences regarding the decidability, or more generally the complexity, of questions about recursive nets of the type we consider. For instance, determining if a given neuron ever assumes the value "1" is effectively undecidable (as the halting problem can be easily reduced to it; details are given in the full paper); on the other hand, the problem appears to become decidable

if a linear activation is used (halting in that case is equivalent to a fact that is widely conjectured to follow from classical results due to Skolem and others on rational functions; see [2], page 75), and is also decidable in the pure threshold case (there are only finitely many states). As our function σ is in a sense a combination of thresholds and linear functions, this gap in decidability is perhaps remarkable.

One obvious question deals with the use of other activation functions. Using the "standard sigmoid" $1/(1+e^{-x})$ presents some technical difficulties, because rational numbers are harder to deal with. For instance, requiring an ouput sequence of exact "1's" is too stringent, but there are obvious modifications that can be done. On the other hand, an equation of the type $x^+ = \tau(Ax+bu+c)$, where τ is a hard threshold (Heaviside) function, can only simulate a finite automaton, as all states are essentially binary.

Many other types of "machines" may be used for universality (see [9], especially Chapter 2, for general definitions of continuous machines). For instance, with a similar proof we can show that systems evolving according to equations $x^+ = x+\tau(Ax+bu+c)$, where τ takes the sign in each coordinate, again are universal in a precise sense. It is interesting to note also that such equations represent an Euler approximation of a differential equation; this suggests the existence of continuous-time simulations of Turing machines.

In closing, we note that the idea of using continuous-valued neurons in order to attain gains in computational capabilities compared with threshold gates had been explored in other work, for the special case of feedforward nets –see for instance [10] for questions of approximation and function interpolation, and [5] for questions of Boolean circuit complexity. See also [3] for other work on continuous-valued models of computation.

REFERENCES

1. Alon, N., A.K. Dewdney, and T.J. Ott, "Efficient simulation of finite automata by neural nets," *J. A.C.M.* (1991): to appear.
2. Berstel, J. and C. Reutenauer, *Rational Series and their Languages*, Sringer-Verlag, Berlin, 1988.
3. Blum, L., M. Shub, and S. Smale, "On a theory of computation and complexity over the real numbers: NP completeness, recursive functions, and universal machines," *Bull. A.M.S.* **21**(1989): 1-46.
4. Franklin, S., and M. Garzon, "Neural computability," in *Progress In Neural Networks, Vol 1*(O. M. Omidvar, ed.), Ablex, Norwood, NJ, 1990, pp. 128-144.
5. Maass, W., G. Schnitger, and E.D. Sontag, "On the computational power of sigmoid versus boolean threshold circuits," *Proc. of the 32nd Annual Symp. on Foundations of Computer Science*, San Juan, PR, October 1991.
6. Marcus, C.M., and R.M. Westervelt, "Dynamics of iterated-map neural networks," *Phys. Rev. Ser. A* **40**(1989): 3355-3364.
7. Pollack, J.B., *On Connectionist Models of Natural Language Processing*, Ph.D. Dissertation, Computer Science Dept, Univ. of Illinois, Urbana, 1987. (Available as MCCS-87-100, Computing Research Laboratory, NMSU, Las Cruces, NM.)
8. Siegelmann, H.T., and E.D. Sontag, "Neural nets are universal computing devices," Report SYCON-91-08, Rutgers Center for Systems and Control, Rutgers University, May 1991.
9. Sontag, E.D., *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, Springer, New York, 1990.
10. Sontag, E.D., "Remarks on interpolation and recognition using neural nets," in *Advances in Neural Information Processing Systems 3* (R.P. Lippmann, J. Moody, and D.S. Touretzky, eds), Morgan Kaufmann, San Mateo, CA, 1991, pp. 939-945.
11. Sun, G.Z., H.-H. Chen, Y.-C. Lee, and C.L. Giles, "Turing equivalence of neural networks with second order connection weights," in *Proc. Int. Joint Conf. Neural Networks*, Seattle, July 1991.