INFORMATION AND COMPUTATION **128**, 48–56 (1996)
ARTICLE NO. 0062

The Dynamic Universality of Sigmoidal Neural Networks

JOE KILIAN

NEC Research Institute, Princeton, New Jersey 08540
E-mail: joe@research.nj.nec.edu

AND

HAVA T. SIEGELMANN

Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel
E-mail: iehava@ie.technion.ac.il

We investigate the computational power of recurrent neural networks that apply the sigmoid activation function $\sigma(x) = [2/(1+e^{-x})] - 1$. These networks are extensively used in automatic learning of non-linear dynamical behavior. We show that in the noiseless model, there exists a universal architecture that can be used to compute any recursive (Turing) function. This is the first result of its kind for the sigmoid activation function; previous techniques only applied to linearized and truncated version of this function. The significance of our result, besides the proving technique itself, lies in the popularity of the sigmoidal function both in engineering applications of artificial neural networks and in biological modelling. Our techniques can be applied to a much more general class of "sigmoidal-like" activation functions, suggesting that Turing universality is a relatively common property of recurrent neural network models. © 1996 Academic Press, Inc.

1. INTRODUCTION

We consider the power of recurrent sigmoidal neural networks. In the simplest form, an N-state recurrent neural network is an N-dimensional dynamical system over a bounded subset of the reals (e.g., over the solid N-cube $[-1, 1]^N$), and can be expressed as a quadruple (N, W, Θ, f). Here N is the dimension of the network, $W = \{w_{i,j} \in \mathbb{R} \mid 1 \leqslant i, j \leqslant N\}$, $\Theta = \{\theta_1, ..., \theta_N\}$ are called the *weights* (or constants), and $f\colon \mathbb{R} \to [0, 1]$ is called the *activation function*. Each neuron i computes its next state, $x_i(t+1)$, by the formula

$$x_i(t+1) = f\left(\left(\sum_{j=1}^{N} w_{i,j} x_j(t)\right) - \theta_i\right). \qquad (1)$$

We emphasize the difference between acyclic (so called *feedforward*) architectures and the general (*recurrent*) ones, like those treated here. The feedforward nets cannot predict non-stationary time-series nor can they describe temporal processing, while the later demonstrate a much more flexible and rich dynamics. Hence, in the acyclic model, one is interested in approximation mainly, while the expectations from the recurrent structured networks are at the level of dynamical and computational properties.

The computational and general dynamical properties of recurrent neural networks depend intimately upon the choice of the activation function. For example, if f is a linear function, then this linear system is essentially computing repeated matrix multiplications on an initial vector. If f is the Heaviside function given by

$$f(x) = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{otherwise,} \end{cases}$$

then each neuron takes on a value in $\{0, 1\}$, and the system becomes finite-state. These qualitatively different behaviors motivate the study of the power of neural network models under different activation functions.

1.1. *Previous Work*

Pollack [8] proposed a recurrent net model that is Turing universal. His model consists of a finite number of neurons of two different kinds, having linear and Heaviside responses. (The unbounded precision of the neurons was used by him to implement the context of the tape, and the Heaviside neurons simulated the finite control.) A crucial characteristic of Pollack's machine was that the activations were combined using multiplications as opposed to just linear combinations. His model, thus, does not fall into the framework given above.

Siegelmann and Sontag first demonstrated the Turing universality of first-order neural nets for a specific activation function [10, 11]. Their activation function, known as the *saturated linear function*, is defined by

$$f(x) = \begin{cases} 0 & \text{for} \quad x \leqslant 0 \\ x & \text{for} \quad 0 < x < 1 \\ 1 & \text{for} \quad x \geqslant 1. \end{cases} \qquad (2)$$

0890-5401/96 $18.00

Copyright © 1996 by Academic Press, Inc.
All rights of reproduction in any form reserved.

They demonstrated the existence of a single choice of weights (which are simple rational numbers) and hence a constant number of nodes, such that by choosing the initial values of the neurons one could simulate the behavior of an arbitrary Turing machine. That is, once the initial values were chosen, then n steps of computation by a Turing machine could be simulated in $O(n)$ steps by this universal network. This result was generalized in [5] to other saturated functions (ones that eventually become constants in both ends), not necessarily saturated-linear ones.

Given these results, it is natural to ask whether one can prove Turing universality results for activation functions used in practice. One family of activation functions widely considered in the literature is this of the *sigmoid* functions, such as $\sigma(x)=1/(1+e^{-x})$ or

$$\sigma(x)=\frac{2}{1+e^{-x}}-1. \tag{3}$$

Much effort has been directed towards the practical implementations of sigmoidal neural networks applications [1, 2, 3, 9, 13]. However, almost no previous theoretical work was done on such networks, mainly because of technical difficulties encountered with the sigmoidal function.

1.2. *The Results of This Paper*

We show the existence of a finite dimensional universal sigmoidal neural network. That is, we show the existence of a network (N, W, Θ, σ), with a distinguished neuron x_1 and a recursive encoding function $\delta(M, \alpha)$, such that for every Turing machine M and string $\alpha \in \{0, 1\}^*$: M halts on input α iff x_1 ever exceeds $\frac{3}{4}$ when (N, W, Θ, σ) is run from initial configuration $\delta(M, \alpha)$. At any other time $x_1 \leqslant \frac{1}{4}$. In particular, if M does not halt on input α then x_1 is always less than $\frac{1}{4}$.

As a corollary of our result, there is no computable limit on the running time of a general sigmoidal neural network. (Clearly, there are particular networks having a well specified limit.) Also, if one wishes to emulate a sigmoidal neural network using fixed-precision arithmetic, one cannot fix in advance the number of bits of precision. Thus, our construction may be thought of as a negative result concerning real-life sigmoidal neural networks. One cannot automatically assume that a natural network converges or enters a detectable oscillatory state within any reasonable time bound. Also, one cannot *a priori* ignore the presence of even the slightest noise or roundoff error—since our construction is exquisitely sensitive to both effects. (In the construction given in this version of our paper, some of the weights are equal to a constant that is a solution to a transcendental equation. We conjecture that indeed only rational weights are required.)

The universality results hold not only for sigmoidal networks, but also for networks with activation functions "similar" to the sigmoid. This will be further described below.

1.3. *Techniques Used*

The proof we present here must contend with some substantial technical difficulties that arise when using σ networks. The primary difficulty we face is that we can no longer implement noise-free logical operations. Using saturated activation functions (i.e., ones that eventually become perfectly constant), one can keep a finite number of bits stored as discrete values (such as 1 and 0 or 1 and -1), and perfectly implement logical operations on these bits. These implementations will map slightly noisy boolean outputs to perfectly noise-free boolean outputs. This ability has proved crucial to all previous constructions. However, with the sigmoid activation function, we do not have this property. For instance, even if neurons x and y were guaranteed to have "ideal" 0/1 values, we still cannot exactly compute the logical AND of x and y in our model. In our implementation, not only will the answer not take on an "ideal" 0/1 value, but it will take on slightly different values depending on whether $(x, y)=(0, 0)$ or $(x, y)=(0, 1)$. We only guarantee that the two values will be reasonably close together.

In our construction, there is complete "crosstalk": deviations from the ideal in one part of the system will result in deviations from the ideal in every other part. We have to be careful, for instance, to keep one data-storing neuron from irrecoverably corrupting another data-storing neuron. A more difficult problem is that continuous fluctuations in the state of the finite control will send unmanageable amounts of noise throughout the entire system. Indeed, just the fact that it remembers state information will in subtle ways corrupt the data.

On a high level, we solve these problems by introducing a new type of automaton, called an *alarm clock machine*, that does not rely on remembering state information. An alarm clock machine consists of a restricted finite control that has access to a finite number of alarm clocks. Each alarm clock c_i has a variable period p_i. If the clock alarms at time t_i, then the clock will next alarm at time t_i+p_i (unless delayed, as described below). Until it is woken by one or more alarm clocks, the finite control is required to spend its time in a memoryless "sleep" state. When woken, the finite control is allowed to run for a *constant* number of steps, in which it may perform operations such as delaying a clock (so that it next alarms at t_i+1) or lengthening its period (setting $p_i=p_i+1$) before going back to sleep. Since the finite control remembers nothing from before it woke up, it must base its actions only on its knowledge of which clocks alarmed during this short waking phase; see

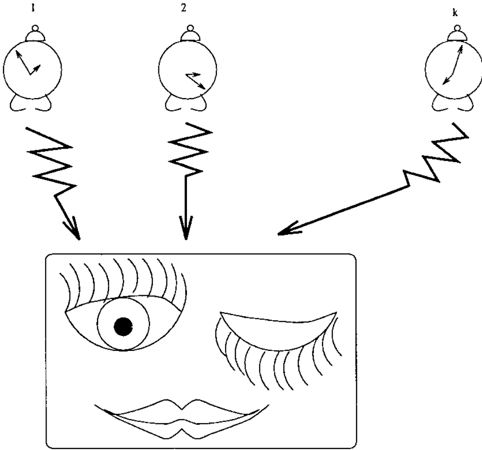


FIG. 1. An alarm clock machine: k alarm clocks and a controller that is in a sleeping state and when woken up is active for c steps only.

Fig. 1. We first show that alarm clock machines are Turing universal by a series of reductions to more conventional automata. We then show how to implement our alarm clock machines by sigmoidal neural networks. (To be more precise, this implementation is only guaranteed to work for the universal alarm clock machine running on a properly constructed input.) The finite control neurons in this implementation spend most of their time in a low-noise sleep state that prevents our data from being corrupted faster than it can be repaired.

The rest of the paper is organized as follows. In Section 2, we introduce alarm clock machines and prove that they are Turing universal. In the following section, we show how to substitute the alarm clocks with "dynamic counters" that behave in a restricted manner. In Section 4, we describe how to simulate alarm clock machines by sigmoidal first-order neural networks thus proving the universality of the networks. Section 5 generalizes the universality result to other "sigmoidal-like" networks.

The proof is conceptually hard. To make it relatively reader-friendly, we omit excessive notations and details.

2. ALARM CLOCK MACHINES

An *alarm clock machine* $\mathscr{A}$ is a triple (F, k, c) where $k, c \geqslant 1$ and F is a function from $\{0, 1\}^{kc}$ to a subset of ACTION, where

$$\text{ACTION} = \{delay(i), lengthen(i) \mid 1 \leqslant i \leqslant k\} \cup \{halt\}.$$

Here, k denotes the number of alarm clocks available to F, and F is a function that, based on the history of alarms from the last c time steps, halts and/or performs some simple operations on its clocks.

The input to (F, k, c) consists of $((p_1, t_1), ..., (p_k, t_k))$, where p_i denotes the period of clock i, and time t_i denotes the next time it is set to alarm.

The alarm clock machine operates as follows. For notational ease, we (conceptually) keep arrays $a_i(t)$, for $t \in Z$ and $1 \leqslant i \leqslant k$, with each entry initially set to 0. At time step T (initially 0), for $1 \leqslant i \leqslant k$, if $t_i = T$, then $a_i(T)$ is set to 1 and t_i is set to $t_i + p_i$. This event corresponds to clock i alarming. F then looks at $a_i(t)$ for $1 \leqslant i \leqslant k$ and $T - c < t \leqslant T$, and executes 0 or more actions. Action *delay*(i) sets t_i to $t_i + 1$, action *lengthen*(i) sets p_i to $p_i + 1$, and action *halt* halts the alarm clock machine.

We make two stipulations on a legal execution of an alarm clock machine. First, if its input consists of all 0's, then F outputs the null set of actions (the machine is "asleep" until woken). Second, we require that $|p_i/p_j| < O(1)$ for all $1 \leqslant i, j \leqslant k$. That is, there is a positive upper bound on the ratio between any two clock periods. This second restriction allows us to more easily simulate our machines. In fact, in our proof of Turing universality, we guarantee that p_i and p_j differ by at most 1. We assert that alarm clock machines are computationally (dynamically) universal.

THEOREM 1. *There exists an alarm clock machine (F, k, c) and a recursive encoding function enc(M) such that for all Turing machines M and binary inputs α, (F, k, c) halts on input enc(M, α) iff M halts on input α. Furthermore, if M halts in T steps, then (F, k, c) will halt in $2^{O(T)}$ steps.*

We prove this result by a series of reductions to counter machines, which are known to be Turing universal.

2.1. *Adder Machines*

We first introduce the adder machines.

DEFINITION 2.1. An adder machine $\mathscr{D}(k)$ is a machine consisting of a finite control and k adders.

The operations on the adders are

- Inc(adder) for adders $i = 1, ..., k$,
- Compare (Adder-i, Adder-j) is a function with the range $\{\leqslant, >\}$.

DEFINITION 2.2. An Adder machine is said to be *simply controlled* if its finite control consists of a combinational circuit only, with no loops (i.e., no internal memory).

LEMMA 2.3. *An adder machine $\mathscr{D}(k)$ with c control states can be simulated by a simply controlled adder machine $\mathscr{D}'(k + c)$ (proof omitted).*

Now, we show the equivalence of adder machines and counter machines, thus proving that adder machines compute all recursive functions.

DEFINITION 2.4. A counter machine $\mathscr{C}(k)$ consists of a finite control and k counters. The counters hold whole numbers; the operations on each counter are Test-for-0, Inc, Dec, and also No-change (Hopcroft and Ullman, 1979)

LEMMA 2.5. *Adder machines and counter machines are linear time equivalent (proof omitted).*

COROLLARY 2.6. *The class of functions computed by an adder machine is recursive. $\forall$ recursive functions ϕ which are computed by a TM M in time T, $\exists$, adder machines that compute ϕ in time $O(2^T)$.*

Proof. Counter machines with at least four counters are known to simulate TM's in exponential time slowdown [4, p. 171, Lemma 7.4]. ▮

2.2. *Alarm Clock and Adder Machines*

An alarm clock machine $\mathscr{A}$ is a special case of a counter machine, and hence $\{\mathscr{A}\} \subseteq \{\mathscr{D}\}$. Next, we show the other inclusion.

LEMMA 2.7. *Given a simply controlled adder machine $\mathscr{D}(k)$ that computes in time T, $\exists$ an alarm clock machine having $O(k^2)$ clocks that simulates $\mathscr{D}$ in time $O(T^3)$.*

The rest of this section is the proof of Lemma 2.7.

Given a simply controlled adder machine $\mathscr{D}$ with k adders, $1, ..., k$, we construct an alarm clock machine $\mathscr{A}$ which simulates $\mathscr{D}$.

The alarm clocks $1, ..., k$ of $\mathscr{A}$ simulate the adders. Alarm clock 0 is used as the "0" value to be compared against by the other k alarm clocks. An adder i is simulated by the alarm clock i, by its *temporal shift* from alarm clock 0. That is, if adder i is set to n, then clock i has the same period as clock 0, but it alarms n time units after clock 0 alarms. We always ensure that the period of the clocks is greater than their phase differences, thus avoiding wraparound problems. The correspondence between adders and the alarm clocks $1, ..., k$ is as follows:

Adder_i	Alarm clock_i
$\text{Inc}(A_i)$ $\text{Compare}(A_i, A_j)$	$\text{Delay}(i)$ Compare shift phase of clocks i and j from 0

One subtlety is how to implement the Compare operation. The alarm clock machine's finite control is only allowed to remember the alarm sequence for the last $O(1)$ time steps. However, after simulating the t th time step of the adder machine, any two alarm clocks may be phase shifted by $\Omega(t)$ time units. We need to perform the comparisons and represent this information in a way usable by the short-memory finite control. We accomplish this task by having a set of $O(k^2)$ auxiliary clocks used to collect this information:

For each pair of clocks (i, j), $i < j$, the auxiliary clock ij determines whether the phase shift of clock i is less than or equal to the phase shift of clock j. The auxiliary reference clock 00 is used to synchronize the auxiliary clocks.

We now describe how the finite control uses the auxiliary clocks to compare the phase shift of the adder clocks. The period of the auxiliary clocks is maintained to be one greater than the period of the adder clocks. Thus, they alarm one time-step later in each successive cycle of the adder clocks. Conceptually the finite control uses these auxiliary clocks to sweep through the adder clock cycle, and records the information it needs by delaying the auxiliary clocks.

Initially, we assume that all of the auxiliary clocks alarm in synchrony with clock 00, and that their phase shift with respect to clock 0 is less than that of any of the adder clocks (this is easily accomplished by suitably setting the initial conditions). The finite control works as follows:

- If clocks 00, ij, and i alarm simultaneously, but not clock j, then the finite control delays clock ij once. If 00 and j but not i alarm, it delays clock ij twice.
- If clocks 00 and 0 alarm simultaneously, then it means that the comparison is done and its result is stored by the auxiliary clocks. The finite control will then be woken up and receive the results during the next two steps: the alarm pattern of the auxiliary clock ij determines whether clock i's phase shift is less than, equal to, or greater than that of clock j, for all clocks i, j. The finite control then delays the auxiliary clocks so that they will again be synchronous.

It is easy to verify that each of these operations can be performed by remembering the alarm history of the last four time steps only.

Once the finite control has the comparison information, it determines if the original adder machine would have halted, and halts accordingly. Otherwise, it determines which adders of the original machine would have incremented, and delays their corresponding clocks. Finally, in order to ensure that the phase shift for the adder clocks do not wrap around, the finite control lengthens the period of all of the clocks by 1.

To simulate the t th step of the adder machine, the alarm clock machine performs the comparisons in $O(t^2)$ time (the period is $O(t)$) and in $O(1)$ time it performs the requisite delays and lengthens the clock periods. Thus, $O(t^3)$ steps are required to simulate t steps of the adder machine, and Lemma 2.7 is proven.

3. SIMULATING CLOCKS WITH DYNAMIC COUNTERS

We now show how to simulate the clocks in the universal alarm clock machine with simple restricted counters, which

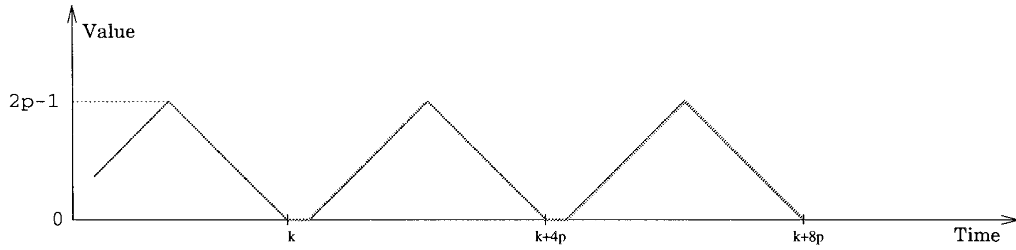
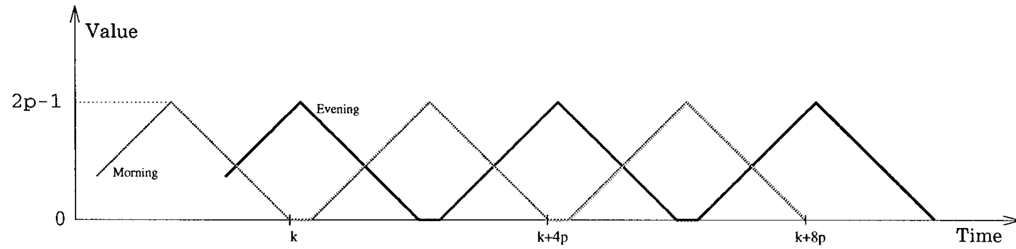
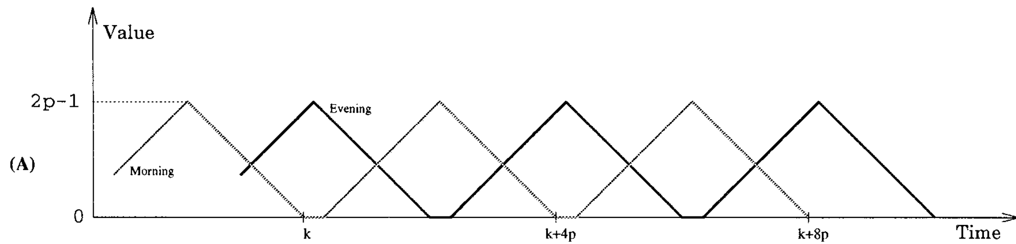


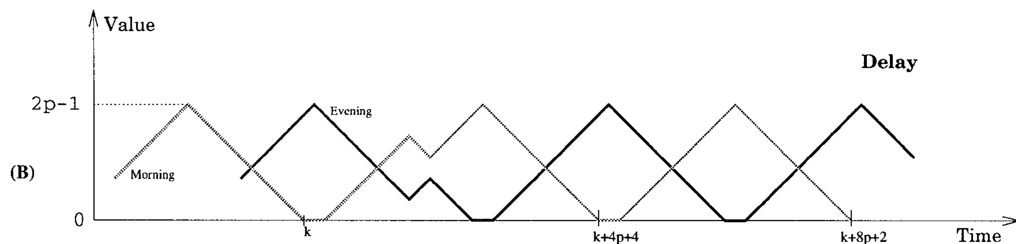
FIG. 2. The values of a dynamic counter that is associated with an alarm clock of period p.



M:	$1\to0$	$0\to0$	$0\to0$	$0\to1$	$1\to2$	$2\to3$	$3\to4$	$4\to5$	$5\to4$	$4\to3$	$3\to2$	$2\to1$	$1\to0$	$0\to0$
E:	$3\to4$	$4\to5$	$5\to4$	$4\to3$	$3\to2$	$2\to1$	$1\to0$	$0\to0$	$0\to0$	$0\to1$	$1\to2$	$2\to3$	$3\to4$	$4\to5$
		k												$k+12$

FIG. 3. A pair of dynamic counters simulating an alarm clock of period p: in steady state (top) after alarming at time k. (Bottom) A numerical example for $p=3$.





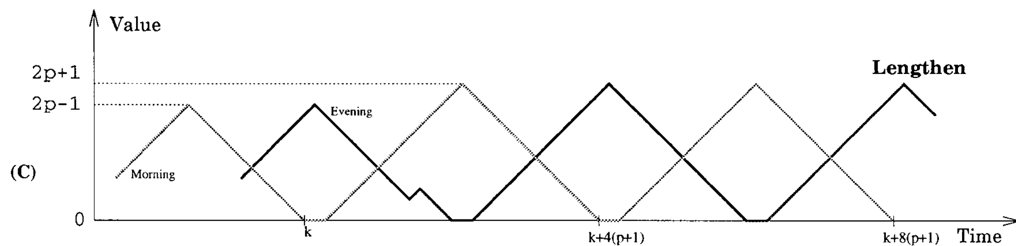


FIG. 4. A pair of dynamic counters (A) simulating an alarm clock in steady state (B) simulating delay in alarming (C) simulating lengthening the period of an alarm clock.

we call dynamic counters. This will make the simulation of alarm clock machines by sigmoidal networks—described in the next section—easier. We call the event that dynamic counter i is set to 0 a *zero event for i*, and the event that some dynamic counter is set to 0 a *zero event*. Our restrictions are as follows:

- Every dynamic counter must always be either incremented or decremented in each time step, unless it is at 0, in which case it must be incremented within $O(1)$ steps.
- Once the finite control starts incrementing {decrementing} a dynamic counter, it must continue to do so in each successive time step, until it has an opportunity to change its direction. (It can "sleep" after giving the command of what direction to go on with.)
- The finite control is only allowed to change the directions of the dynamic counters during a period of $O(1)$ time steps following a zero event.
- At any time step, for any i, there will be at most $O(1)$ zero events before the next zero event for i.
- A clock alarming must correspond to a zero event.

We next show our particular implementation. To simplify matters, we assume that the universal alarm clock machine runs a valid simulation of a simply controlled adder machine, and thus behaves as described in the previous section.

We implement each alarm clock i with a pair of dynamic counters, which we call *morning* (M_i) and *evening* (E_i). We refer to the period between two successive alarmings of a morning dynamic counter as a day. The operations *delay* and *lengthen* can be referred to as delaying the next day and lengthening the duration of all days from now on, respectively. When the clock is in its steady state (neither being delayed or lengthened) with period p, the value of each dynamic counter has the periodic behavior, described in Fig. 2. That is, it counts up to $2p-1$, then down to 0, stays 0 for two time steps, and starts counting up again:

$$\cdots 0\ 0\ 1\ 2\ 3\ 4 \cdots (2p-1) \cdots 4\ 3\ 2\ 1\ 0\ 0\ 1\ 2\ 3 \cdots.$$

To achieve this oscillatory effect, we put M_i and E_i $2p$ time steps out of phase. If M_i (resp. E_i) is decremented to 0 at time k, then E_i (resp. M_i) (which has been incrementing) starts decrementing at time $k+1$ and M_i (resp. E_i) starts incrementing at time $k+2$.

Thus, in its steady state, the system oscillates with a period of $4p$. We interpret a unit of clock time as four units of the dynamic counter time, and identify the event that M_i turns from 1 to 0 with the clock alarming; see Fig. 3. (This construction does not handle clocks with period 1. However, such clocks are not necessary for our alarm clock machine to be universal.)

We now show how to implement the *delay* and *lengthen* operations. For these operations, we assume that neither dynamic counter is equal to 0, and that it is known which dynamic counter is decrementing and which dynamic counter is incrementing. By inspection of our "program", one can verify that the finite control will always have this information within $O(1)$ time after it has woken up, and that it must wait only $O(1)$ steps before the nonzero condition is met. For example, when the finite control has received all of its comparison information, it can wait a few steps and ensure that the morning dynamic counters of all the comparison clocks and the 0 clock are incrementing, while the evening dynamic counters of all the adder clocks are decrementing.

To delay a clock, the finite control increments the dynamic counter it had previously been decrementing and decrements the dynamic counter it has previously been incrementing for two time steps, and then resumes its normal operations; see Fig. 4. To lengthen the day's period, the finite control increments, for one time step, the dynamic counter that it had previously been decrementing (and does not change the direction of the other counter) and then continues with normal operation. Note that this operation will also alter the phase shift of the dynamic counters. However, since it will be performed on all of the clocks in the simulation, the relative phase shifts will be preserved.

4. SIGMOIDAL NETWORKS ARE UNIVERSAL

In this section, we prove the main theorem:

THEOREM 2. *Given an alarm clock machine $\mathscr{A}$ (with no input) that simulates a simply controlled adder machine; there is a σ-network $\mathscr{N}$ that computes just like $\mathscr{A}$ and requires the same computation time. Furthermore, the size of this network is linear in the number of clocks and the size of the finite control of $\mathscr{A}$.*

The rest of this section is dedicated to the proof. Let us, first, demonstrate the four properties of our sigmoid that are useful in the proof:

- *Feature* 1. There exists a positive constant c such that $\forall\, |x| > c$, $\sigma(x)$ is monotone nondecreasing, and $\sigma(x) \in [-1, -1/2]$ or $\sigma(x) \in [\frac{1}{2}, 1]$, depending on the sign of x.

• *Feature* 2. For every constant b for which the slope of $\sigma(bx)$ at 0 is larger than 1, $\sigma(bx)$ has three fixed points. One is zero and the two others are denoted A and $-A$; they differ only in sign. (For example, for the sigmoid $\sigma(x)=2/(1+e^{-x})-1$, the slope at 0 is $\frac{1}{2}$ and the feature requires $b>2$.) The larger b is, the closer A gets to 1 $(0.5<A<1)$. For example, using 15 decimal digits in the precision:

$$b=5 \qquad A=0.98562369130483$$

$$b=10 \qquad A=0.999909121699349$$

$$b=30 \qquad A=0.999999999999812.$$

Let c be a constant. In the equation $\sigma(bx+c)$, the two external fixed points are not equal in size anymore, provided that $c\neq 0$, and thus are denoted by A_1 (≈ -1) and A_2 (≈ 1).

The fixed points A_1 and A_2 are exponential attractors (for all $x\neq 0$), and the middle fixed point is unstable. (In fact, one can achieve d^{-t} convergence for any d, $0<d<1$ by a suitable choice of the constant b.)

• *Feature* 3. The function σ is differentiable twice around its attractors.

• *Feature* 4. For every x, $\sigma(x)=x+O(x^3)$. (Hence, if x is a small number then $\sigma(x)\sim x$.) This is proved by considering the Taylor expansion around 0.

We will use the above four properties of our sigmoid to prove its universality. Given an alarm clock machine $\mathscr{A}$ (with dynamic counters), the network $\mathscr{N}$ that simulates $\mathscr{A}$ consists of three main components as shown in Fig. 5: a finite control, a set of dynamic counters, and a set of flip-flops. The finite control and the dynamic counter parts of the network simulate the corresponding components of the alarm clock machine. Since the finite control is memoryless, we need a third mechanism for controlling the dynamic counters. This is accomplished by implementing a set of bi-state flip-flop neurons which serve as intermediaries between the finite control and the dynamic counters.

Implementing the Finite Control. It has long been known how to simulate any finite control $FC_{\mathscr{A}}$ of $\mathscr{A}$ by a network of threshold devices [6, 7]. If the original finite control depends only on the last $O(1)$ time steps, the resulting threshold network can be made to be feed-forward.

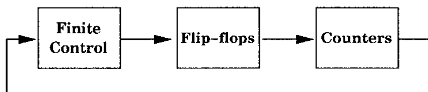


FIG. 5. Block diagram of our simulation.

We substitute each threshold device

$$x_i(t+1)=\mathscr{H}\left(\sum_{j=1}^{N}\omega_{ij}x_j+\theta_i\right)$$

with a sigmoidal device

$$x_i(t+1)=\sigma\left(\alpha_a\left(\sum_{j=1}^{N}\omega_{ij}x_j+\theta_i\right)\right),$$

for a large fixed constant α_a. As long as the summation in the above expression is guaranteed to be bounded away from 0, the output values of the neuron using the sigmoid activation function will closely approximate the output of the neurons using the Heaviside activation function. By choosing sufficiently large α_a, we can make this approximation as close as we desire (feature 1).

Note that the number of states in our "finite control" is in fact infinite, since every neuron can take on an infinite set of values. Since these values fall within a small neighborhood of either 1 or -1, we can conceptually discretize them; however the continuous nature of these values result in accuracy problems.

For each dynamic counter i, the finite control has two output lines (implemented as neurons) *Start-Inc*$_i$ and *Start-Dec*$_i$. When *Start-Inc*$_i$ is active (i.e., ≈ 1), this means that dynamic counter i should be continually incremented. Similarly, an active *Start-Dec*$_i$ means that dynamic counter i should be continually decremented. Most of the time both output lines are in an inactive state (i.e., ≈ 0). In this case dynamic counter i is treated according to the last issued command, allowing operations to be performed on the dynamic counter when the finite control is inactive. It will never be the case that both signals are simultaneously active.

Bi-directional Flip-flops. Recall that to avoid irrecoverable data corruption, we implement a "finite-control" that converges to a constant "ground state" during the long periods between interesting events. In order to maintain control of the dynamic counters during these quiet period, we introduce special flip-flop devices. These devices will have two stable states, and are guaranteed to exponentially converge to one of them during the quiet periods. While the finite control is active, it can set or reset the value of a flip-flop. Otherwise, the flip-flop maintains its current state.

The update equation of each flip-flop is

$$\mathrm{ff}_i=\sigma(\alpha_{f1}(\text{Start-Inc}_i-\text{Start-Dec}_i)+\alpha_{f2}\mathrm{ff}_i+\alpha_{f3}),$$

where α_{f1}, α_{f2}, and α_{f3} are suitably chosen constants (feature 2).

Counters. Each dynamic counter is implemented via three sigmoidal neurons: one, called the dc neuron, retains

the value of the dynamic counter, and the other two assist in executing the Inc/Dec operations. Let B be a constant $B>2$. A dynamic counter with the value $v \in \mathbb{N}$ is implemented in a dc neuron with a value "close" to B^{-v}. That is, a value 0 in a dynamic counter is implemented as a constant close to 1 in the neuron. When a dynamic counter increases, the associated dc neuron decreases by a factor of B.

Thus, at each step, the dc neuron is multiplied by either B or $1/B$. To do this, we use the approximation

$$\sigma(V+cx_i)-\sigma(V) \approx \sigma'(V)\, cx_i$$

for sufficiently small c and $|x_i|<1$ (feature 3). Let V be the direction input signal, coming from the i th flip-flop. That is, V converges to either A_1 or A_2. A dc neuron updates by the equation

$$\begin{aligned} x_i(t+1) &= \sigma[\alpha_{c1}\sigma(\alpha_{c2}V+\alpha_{c3}+\alpha_{c4}x_i(t)) \\ &\quad -\alpha_{c1}\sigma(\alpha_{c2}V+\alpha_{c3})+\alpha_{c5}x_i(t)] \\ &\approx \sigma[(\alpha_{c1}\sigma'(\alpha_{c2}V+\alpha_{c3}))\,\alpha_{c4}x_i(t)+\alpha_{c5}x_i(t)]. \end{aligned}$$

By a suitable choice of the constants $\alpha_{c1}, ..., \alpha_{c5}$, we have

$$\begin{aligned} \alpha_{c1}\alpha_{c4}\sigma'(\alpha_{c2}A_1+\alpha_{c3})+\alpha_{c5} &= B \\ \alpha_{c1}\alpha_{c4}\sigma'(\alpha_{c2}A_2+\alpha_{c3})+\alpha_{c5} &= 1/B. \end{aligned}$$

If the value of x_i is close enough to 0, we can approximate (using feature 4)

$$\begin{aligned} &\sigma[(\alpha_{c1}\sigma'(\alpha_{c2}V+\alpha_{c3})+\alpha_{c4})\,x_i] \\ &\quad\approx (\alpha_{c1}\sigma'(\alpha_{c2}V+\alpha_{c3})+\alpha_{c4})\,x_i. \end{aligned}$$

The above discussion provides the intuition for why the dc neuron computes either $\sim Bx_i$ or $(\sim 1/B)\,x_i$. Note also that when x is positive and "close to 1," and it is "multiplied by B" then it will in fact be drawn toward a fixed point of the above equation. This acts as a form of error correction.

Proof of Convergence: *Sketch*

To save much details and heavy notation, we sketch the proof to the point that we believe it is clear to complete the details. Ideally, our finite-state neurons would all have $\{0, 1\}$ values, our flip-flops would take on precisely two values (A_1, A_2) and the dc neuron would have the exact activation B^{-v}, where v is the value of the simulated dynamic counter. Unfortunately, it is inevitable that the neuron's values will deviate from their ideal values. To obtain our result, we show that these errors are controllable.

The proof of convergence is organized inductively on the serial number of the day (that is, the times of M_0 alarming). As the network $\mathcal{N}$ consists of three parts; finite automaton (FA), flip-flops (FF), and dynamic counters; for each part we assume a "well behaved input" in day d and prove a "well behaved output" for the same day. As on the first day, input to all parts is well behaved; the correctness follows inductively.

LEMMA 4.1. *We next provide three claims and prove that assuming that one claim is true, the next one* (*in cyclic order*) *results.*

(1) *On each day d,* FC *sends $O(1)$ signals* (*intentionally non-zero*) *to the flip-flops. Each signal has an error bounded by $\mu<0.01$. The sum of errors in the signals of the* FC *during the d th day is bounded by the constant $\beta<0.1$.*

(2) *On each day d, $O(1)$ of the signals sent by* FF *have an error of γ, where γ can be made arbitrarily small* (*as a function of μ and the constants of the flip-flops*). *The sum of error of all signals during the d th day is bounded by δ, where δ can also be made arbitrarily small.*

(3) *On each day d, a dynamic counter with a value y acquires total multiplicative error $\zeta<0.01$. That is, the ratio of the actual value with the ideal value will always be between* 0.99 *and* 1.01.

Proof. $1 \Rightarrow 2$. Assume the finite control sends *Start-Inc$_i$* and *Start-Dec$_i$* to ff$_i$ and these two values are never both active simultaneously. The update equation for each flip-flop is

$$\text{ff}_i = \sigma(\alpha_{f1}(\text{Start-Inc}_i - \text{Start-Dec}_i)+\alpha_{f2}\text{ff}_i+\alpha_{f3}).$$

- When either *Start-Inc$_i$* or *Start-Dec$_i$* is active, ff$_i$ is set to the new value. The error γ is bounded by

$$\gamma \leqslant |1-\sigma(\alpha_{f1}(1-\mu)-\alpha_{f2}+\alpha_{f3})|.$$

It is easy to see that when $|\alpha_{f1}|-|\alpha_{f2}|+|\alpha_{f3}|$ increases, γ decreases. That is, γ is controllable. For example, if $\alpha_{f1} \geqslant \mu^{-1}$ and $\alpha_{f2}, \alpha_{f3} \leqslant 20$ then $\gamma<0.01$.

- When both *Start-Inc$_i$* and *Start-Dec$_i$* are small, ff$_i$ converges to its closer fixed point. If (*Start-Inc$_i$* − *Start-Dec$_i$*) were exactly 0, then ff$_i$ would be attracted exponentially to its closest fixed point. If α_{f2} is large enough, the fixed points can be made arbitrarily close to -1 and 1. Furthermore, noise from α_{f1} (*Start-Inc$_i$* − *Start-Dec$_i$*) can be arbitrarily attenuated, since $|\alpha_{f1}\sigma'(\alpha_{f2}\text{ff}_i+\alpha_{f3})|$ can be made vanishingly small by a suitable choice of constants.

$2 \Rightarrow 3$. The update equation of a dynamic counter x_i is given by

$$\begin{aligned} x_i &= \sigma[\alpha_{c1}\sigma(\alpha_{c2}V+\alpha_{c3}+\alpha_{c4}x_i) \\ &\quad -\alpha_{c1}\sigma(\alpha_{c2}V+\alpha_{c3})+\alpha_{c5}x_i]. \end{aligned}$$

We next show that by using such an update equation, the dc neuron x_i multiplies itself each step by either $\sim B$ or $\sim 1/B$ allowing a small controllable error. Recall that for small y,

$$\sigma(\sigma(V+y)-\sigma(V)) \approx \sigma'(V)\, y.$$

We can choose constants $\alpha_{c1}, \alpha_{c2}, \alpha_{c3}, \alpha_{c4}, \alpha_{c5}$ such that

$$\alpha_{c1}\alpha_{c4}\sigma'(\alpha_{c2}A_1+\alpha_{c3})+\alpha_{c5}=B$$

$$\alpha_{c1}\alpha_{c4}\sigma'(\alpha_{c2}A_2+\alpha_{c3})+\alpha_{c5}=1/B.$$

The deviation from this ideal behavior is caused by three elements:

- the error caused by approximating the difference equation by the differential.
- the error in $\sigma'(\alpha_{c2}V+\alpha_{c3})$ relative to the desired $\sigma'(\alpha_{c2}A_i+\alpha_{c3})$,
- the error caused by using the approximation $\sigma(x)\approx x$ for x small.

In the first case, the multiplicative error is proportional to $\sigma''(A_i)(\alpha_{c4}x_i)^2$. However, x_i shrinks exponentially (and then grows back in a symmetric manner). Hence, in a given day, these terms form two exponentially decreasing sums. In the second case, we can bound the resulting multiplicative error by a function of α_{c1}, α_{c2}, α_{c4} and $\sigma''(a_{c2}A_i+\alpha_{c3})$ times the error in V relative to A_i. Finally, note that $\sigma(x)=x+O(x^3)=x(1+O(x^2))$. Since x_i exponentially vanishes (and reappears), the multiplicative error terms form two exponentially decreasing sums.

By "summing" these multiplicative errors, we get the desired bound. We can then use the identity that

$$(1+\delta_1)(1+\delta_2)\cdots(1+\delta_k)=1+O(\delta_1+\cdots\delta_k),$$

when the sum $\delta_1+\cdots+\delta_k$ is sufficiently small, to approximate the multiplicative error.

$3\Rightarrow 1$. Because the finite control is feed-forward, and since each dynamic counter alarms $O(1)$ times a day, the finite control will output (intentionally) non-zero signals only $O(1)$ times a day. By adjusting the constants in our implementation of the finite control, we can make them have arbitrarily small errors when they change their values. During the quiescent period, if the dynamic counters were actually at 0, then the *Start-Inc*$_i$ and *Start-Dec*$_i$ neurons would converge exponentially to some canonical value, and their difference would converge exponentially close to 0. We can bound the errors caused by the dynamic counters being non-zero as some constant c times the sum of the values of all the dynamic counters at every time in the day. By choosing the weights appropriately, we can in fact make c as small as desired. ∎

5. DISCUSSION

The proof of universality is not constrained to the particular sigmoid we used, but rather can be generalized. Let $\tilde{\sigma}$ be any function which adheres to features 1–4, defined in Section 4. We call all such functions sigmoidal-like functions. The proof of Theorem 2 can be generalized to any sigmoidal-like net.

COROLLARY 5.1. *Let $\tilde{\sigma}$ be any sigmoidal-like function. Given an alarm clock machine $\mathscr{A}$ (with no input) that computes the function ϕ in time T, there is a $\tilde{\sigma}$-network $\mathscr{N}$ that computes ϕ in time $O(T)$. Furthermore, the size of this network is linear in the number of clocks and the size of the finite control of $\mathscr{A}$.*

We conclude that universality may be a general feature for many neural networks.

Received December 4, 1994; final manuscript received April 1, 1996

REFERENCES

1. Cleeremans A., Servan-Schreiber, D., and McClelland, J. (1989), Finite state automata and simple recurrent networks, *Neural Computation* **1**, 372.
2. Elman, J. L. (1990), Finding structure in time, *Cognitive Sci.* **14**, 179.
3. Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., and Lee, Y. C. (1992), Learning and extracting finite state automata with second-order recurrent neural networks, *Neural Computation* **4**, 393–405.
4. Hopcroft, J., and Ullman, J. (1979), "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Readings, MA.
5. Koiran, P., Universal neural networks, manuscript.
6. McCulloch, W. S., and Pitts, W. (1943), A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.* **5**, 115–133.
7. Minsky, M. L. (1967), "Computation: Finite and Infinite Machines," Prentice–Hall, Engelwood Cliffs, NJ.
8. Pollack, J. B. (1987), "On Connectionist Models of Natural Language Processing," Ph.D. Dissertation, Compute Science Dept., Univ. of Illinois, Urbana.
9. Pollack, J. B. (1990), "The Induction of Dynamical Recognizers", Technical Report 90-JP-Automata, Dept. of Computer and Information Science, Ohio State University.
10. Siegelmann, H. T., and Sontag, E. D. (1991), Turing computability with neural networks, *Appl. Math. Lett.* **4**, 6.
11. Siegelmann, H. T., and Sontag, E. D. (1995), On the computational power of neural networks, *J. Comput. System Sci.* **50**, 132–150.
12. Siegelmann, H. T., and Sontag, E. D. (1994), Neural networks analog computation via real weights: Analog computational complexity, *Theoret. Comput. Sci.* **131**, 331–360.
13. Williams, R. J., and Zipser, D. (1989), A learning algorithm for continually running fully recurrent neural networks, *Neural Comput.* **1**, 270.