Computational Intelligence, Volume 12, Number 4, 1996

RECURRENT NEURAL NETWORKS AND FINITE AUTOMATA

HAVA T. SIEGELMANN

Information Systems Engineering, Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel

This article studies finite size networks that consist of interconnections of synchronously evolving processors. Each processor updates its state by applying an activation function to a linear combination of the previous states of all units. We prove that *any* function for which the left and right limits exist and are different can be applied to the neurons to yield a network which is at least as strong computationally as a finite automaton. We conclude that if this is the power required, one may choose any of the aforementioned neurons, according to the hardware available or the learning software preferred for the particular application.

Key words: recurrent neural networks, finite state automata, computational power.

1. INTRODUCTION

Recurrent neural networks are capable of approximating rather arbitrary dynamical systems, and this is of use in adaptive control and signal processing applications (Sontag 1992, Matthews 1992, Polycarpou and Ioannou 1991). They also constitute a powerful model of computation (Siegelmann 1993). In speech processing applications and language induction, recurrent net models are used as identification models, and they are fit to experimental data by means of a gradient descent optimization (the so-called "backpropagation" technique) of some cost criterion (Cleeremans *et al.* 1989, Elman 1990, Giles *et al.* 1992, Pollack 1990, Williams and Zipser 1989).

In these networks, the activation of each processor is updated according to a certain type of function of the activations (x_j) and inputs (u_j) at the previous instant, with real coefficients—also called *weights*—(a_{ij}, b_{ij}, c_i). Each processor's state is updated by an equation of the type

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}x_j(t) + \sum_{j=1}^{M} b_{ij}u_j(t) + c_i\right), \qquad i = 1, \ldots, N, \tag{1}$$

where N is the number of processors and M is the number of external input signals. The function σ is called the activation function. The computational and general dynamical properties of recurrent neural networks depend intimately upon the choice of the activation function. For example, if σ is a linear function, then the system is essentially computing repeated matrix multiplications on an initial vector. If σ is the Heaviside function 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. In particular, we wish to know whether various (popular) neurons constitute a strong enough machine. For this aim, we consider a mathematical tool: an automaton.

1.1. Automata

An automaton, or sequential machine, is a device which evolves in time, reacting to external stimuli and in turn affecting its environment through its own actions. In computer science and logic, *automata theory* deals with various formalizations of this concept. In this formal sense, neural networks constitute a (very) particular type of automata. It is therefore natural to analyze the information processing and computational power of neural networks through their comparison with the more abstract general models of automata classically studied in

computer science. This permits a characterization of neural capabilities in unambiguous mathematical terms.

The components of actual automata may take many physical forms, such as gears in mechanical devices, relays in electromechanical ones, integrated circuits in modern digital computers, or neurons. The behavior of such an object will depend on the applicable physical principles. From the point of view of automata theory, however, all that is relevant is the identification of a set of *internal states* which characterize the status of the device at a given moment in time, together with the specification of rules of operation which predict the next state on the basis of the current state and the inputs from the environment. Rules for producing output signals may be incorporated into the model as well.

Although the mathematical formalization of automata took place prior to the advent of digital computers, it is useful to think of computers as a paradigm for automata, in order to explain the basic principles. In this paradigm, the state of an automaton, at a given time t, corresponds to the specification of the complete contents of all RAM memory locations as well as of all other variables that can affect the operation of the computer, such as registers and instruction decoders. The symbol $x(t)$ will be used to indicate the state at the time t. At each instant (clock cycle), the state is updated, leading to $x(t+1)$. This update depends on the previous state, as instructed by the program being executed, as well as on external inputs like keyboard strokes and pointing-device clicks. The notation $u(t)$ will be used to summarize the contents of these inputs. It is mathematically convenient to consider "no input" as a particular type. Thus one postulates an update equation of the type

$$x(t+1) = f(x(t), u(t)) \tag{2}$$

for some mapping f. Also at each instant, certain outputs are produced: update of video display, characters sent to printer, and so forth; $y(t)$ symbolizes the total output at time t. (Again, it is convenient to think of no output as a particular type of output.) A mapping

$$y(t) = h(x(t)) \tag{3}$$

(which depends only on the present state) provides the output at time t associated to the internal state at that instant.

A finite state automaton is any automaton for which the sets of internal states $x(t)$, input letters $u(t)$, and output letters $y(t)$ are finite.

1.2. Neural Networks and Finite Automata

It has been known at least since the work of McCulloch and Pitts (1943) that finite size recurrent networks consisting of threshold neurons can simulate finite automata. Motivated by successful applications in learning and adapting continuous-type networks and the biological adequacy of analog computation, the computational power of continuous-type networks has become of growing interest. Siegelmann and Sontag proved that networks consisting of a particular type of analog neuron which computes a piecewise linear function may simulate finite automata (Siegelmann 1993) (and even a various infinite automata, i.e., Turing machines and super-Turing models). Kilian and Siegelmann (1993) proved that networks with sigmoidal neurons may simulate finite automata (and Turing machines).

Much speculation was raised about the general characteristics of the activation functions that are associated with regular functions. Here, we provide such a characteristic.

Let ϱ be any function that satisfies the following property:

$$\textit{Both } \lim_{x\to\infty} \varrho(x) = t_+ \textit{ and } \lim_{x\to-\infty} \varrho(x) = t_- \textit{ exist and } t_+ \neq t_- \,. \tag{$\star$}$$

We show that any network of ϱ-neurons of the type

$$x_i(t+1) = \varrho\left(\sum_{j=1}^{N} a_{ij}x_j(t) + \sum_{j=1}^{M} b_{ij}u_j(t) + c_i\right), \qquad i = 1, \ldots, N, \tag{4}$$

can simulate a finite automaton. Thus, any such network can be chosen for implementation, depending only on the cost, availability, and the learnability properties of the particular application.

2. PREVIOUS EFFORT

Some of the previous work on recurrent neural networks has focused on networks of infinite size. As each neuron is itself a processor, such models of infinite power are less interesting for the investigation of computational power, compared to our model, which consists of a finite number of neurons.

There has been previous work concerned with computability by finite networks, however. The classical result of McCulloch and Pitts (1943) (and Kleene (1956)) showed how to implement logic gates by threshold networks, and therefore how to simulate finite automata by such nets.

Another related result was due to Pollack (1987), who argued that a certain recurrent net model, which he called a "neuring machine," is Turing universal. The model in Pollack (1987) consisted of a finite number of neurons of two different kinds, having identity and threshold responses, respectively. The machine was *high order*; that is, the activations were combined using multiplications as opposed to just linear combinations only. Hartley and Szu (1987) discovered a similar result.

Siegelmann and Sontag considered a network of neurons computing linear combinations and a piecewise sigmoidal-like activation function. They proved that the computational power of their network depends on the numbers utilized as weights: If the weights are integers, the network computes as a finite automaton. If the weights are rational numbers, the network is equivalent in power to a Turing machine (Siegelmann and Sontag 1995). When weights are general real numbers, the network turns out to have super-Turing capabilities. However, it is sensitive to resource constraints and thus is not a tautology (Siegelmann and Sontag 1994).

Kilian and Siegelmann (1993) constructed a network of sigmoidal neurons which is Turing universal, although it is exponentially slower than the Siegelmann–Sontag net. They generalized their result to other sigmoidal-like nets.

3. SIMULATION

Here, we use the general definition of finite automaton with no initial state and with sequential output. As a mathematical object, an *automaton* is a quintuple

$$M = (S, U, Y, f, h)$$

consisting of sets S, U, and Y (called respectively the state, input, and output spaces), as well as two functions

$$f\colon S \times U \to S, \qquad h\colon S \to Y$$

(called the next-state and the output maps, respectively). A *finite automaton* is one for which each of the sets S, U, and Y is finite.

We start by introducing the notion of simulation. In general, given an automaton $M = (S, U, Y, f, h)$, the map f can be extended by induction to arbitrary input sequences. That is, for any sequence $u_1, \ldots, u_k$ of values in U, and $q \in S$

$$f_*(q, u_1, \ldots, u_k)$$

is defined as the iterated composition $f(f(\ldots f(f(q, u_1), u_2), \ldots, u_{k-1}), u_k)$. Suppose now given two automata $M = (S, U, Y, f, h)$ and $\overline{M} = (\overline{S}, U, Y, \overline{f}, \overline{h})$ which have the same input and output sets. The automaton $\overline{M}$ *simulates* M if there exist two maps

$$\text{ENC}: S \to \overline{S} \qquad \text{and} \qquad \text{DEC}: \overline{S} \to S,$$

called the *encoding* and *decoding* maps respectively, such that, for each $q \in S$ and each sequence $\omega = u_1, \ldots, u_k$ of elements of U,

$$f_*(q, \omega) = \text{DEC}\,[\overline{f}_*(\text{ENC}\,[q], \omega)], \qquad h(q) = \overline{h}(\text{ENC}\,[q]).$$

Assume that for some integer m the input value set U consists of the vectors $e_1, \ldots, e_m$ in R^m, where e_i is the ith canonical basis vector, that is, the vector having a 1 in the ith position and zero in all other entries. Similarly, suppose that Y consists of the vectors $e_1, \ldots, e_p$ in R^p. (The assumption that U and Y are of this special "unary" form is not very restrictive, as one may always encode inputs and outputs in this fashion.) The 0 vector denotes the no-information in both input and output ends.

4. MAIN RESULT

The following interpolation fact holds for any function ϱ which satisfies property $(\star)$:

Lemma 1. For any activation function ϱ that satisfies property $(\star)$, there exist constants $\omega_0, \omega_i, b_i, c_i \in R, i = 1, 2, 3$ so that the function

$$f(y) = \omega_0 + \sum_{i=1}^{3} \omega_i \varrho(b_i y + c_i)$$

satisfies $f(-1) = f(0) = 0$ and $f(1) = 1$.

Before proving the above lemma, we show how the theorem follows from it:

Theorem 1. Every finite automaton can be simulated by a neural net with any activation function that satisfies property $(\star)$.

In particular, we show that any finite automaton with s states and m input values can be simulated by a network of $N = 3sm$ neurons whose activation function ϱ satisfies property $(\star)$.

Proof of the Theorem. Assume that the states of the finite automaton M to be simulated are $\{\xi_1, \ldots, \xi_s\}$. A neural network that simulates M has $N = 3sm$ neurons and is built as follows. Denote the coordinates of the state vector $x \in R^N$ by $x_{ijk}, i = 1, \ldots s, j = 1, \ldots, m$, $k = 1, 2, 3$.

Consider the Boolean variables g_{rv} for $r = 1, \ldots, s$, $v = 1, \ldots, m$, that indicate if the current state of M is r and the last input read was v. We will express these variables as

$$g_{rv} \equiv \omega_{rv} + \sum_{k=1}^{3} \omega_{rvk} x_{rvk}, \tag{5}$$

where the weights ω_{rv}, ω_{rvk} will be described later. We write (g_{rv}) for a matrix indexed by r and v. In terms of these quantities, the update equations for $r = 1, \ldots, s$, $v = 1, \ldots, m$, $k = 1, 2, 3$, can be expressed as:

$$x^{+}_{rvk} = \varrho\left(b_{rvk}\left(\sum_{j=1}^{m}\sum_{i\in S_{rv}} g_{ij} + u_v - 1\right) + c_{rvk}\right), \tag{6}$$

where

$$S_{rv} := \{l \mid f(q_l, e_v) = q_r\}$$

and where b_{rvk} as well as c_{rvk} will also be specified below. Finally, for each $l = 1, \ldots, p$, the lth coordinate of the output is defined as

$$y_l = \varrho\left(c\sum_{j=1}^{m}\sum_{i\in T_l} g_{ij}\right),$$

where $T_l := \{i | h_l(q_i) = 1\}$ for the coordinate h_l of the function h, and c is any constant so that $\varrho(0) \neq \varrho(c)$.

The proof that this is indeed a simulation is as follows:

1. We first prove inductively on the steps of the algorithm that the expressions (g_{rv}) are always of the type E_{ij}, where E_{ij} denotes the binary matrix that its ijth entry has the value 1 and all the rest have the value 0. Furthermore, except for the starting time, E_{rv} indicates that the simulated finite automata is in state r and its last read input was v.
   - We start the network with an initial state $x_0 \in R^N$ so that the starting (g_{ij}) has the form E_{r1}, where q_r is the corresponding state of the original automaton. It is easy to verify that such x's are possible, since in Eq. (5) the different equations are uncoupled for different r and v, and not all weights ω_{rvk} for $k = 1, 2, 3$ can vanish; otherwise in Lemma 5, the function f (which is used as g_{rv}) is constant.
   - Assume (g_{ij}) has indeed the Boolean values as stated at any time; that is, only one of the g_{rv} has the value 1 and the rest are in 0, the associated state of the finite automata is q_r and the last read input is u_v. Then, the expression

$$y_{rv} = \left(\sum_{j=1}^{m}\sum_{i\in S_{rv}} g_{ij} + u_v - 1\right)$$

     can only take the values -1, 0, or 1. The value 1 can only be achieved for this sum if both $u_v = 1$ and there is some $i \in S_{rv}$ so that $g_{ij} = 1$, that is, if the current state of the original machine is q_i and $f(q_i, e_v) = q_r$.

     By Lemma 1, there exist values ω_{rv}, ω_{rvk}, b_{rvk}, c_{rvk} $(k = 1 \ldots 3)$ so that $f(y_{rv})$ (which is here the value of g_{rv} at time $t+1$) assumes the value 1 only if y_{rv} was 1 and is 0 for the other two cases. This proves the correctness in terms of the expressions g_{ij}. The vectors x_{rvk} take the values $\varrho(b_{rvk}y_{rv} + c_{rvk})$, as described by Eq. (6).

2. The encoding and decoding functions are defined as follows: The encoding map ENC $[q_r]$ maps q_r into any fixed vector x so that Eq. (5) gives $(g_{ij}) = E_{r1}$. The decoding map DEC $[x]$ maps those vectors x that result in $(g_{ij}) = E_{rv}$ $(r = 1 \ldots s,\ v = 1 \ldots m)$ into q_r and is arbitrary on all other elements of R^N. ■

In the next section, we prove the lemma.

5. PROOF OF THE ABOVE LEMMA

Proof. We prove more than required; namely, we show that for each choice of three numbers $r_{-1}, r_0, r_1 \in R$ there exist $\omega_0, \omega_i, b_i, c_i \in R$, $i = 1, 2, 3$ so that, denoting

$$f(y) = \omega_0 + \sum_{i=1}^{3} \omega_i \varrho(b_i y + c_i),$$

it holds that $f(-1) = r_{-1}$, $f(0) = r_0$, and $f(1) = r_1$. To prove this, it suffices to show that there are b_i, c_i $(i = 1, 2, 3)$, so that the matrix

$$\Sigma_{bc} = \begin{pmatrix} \varrho[b_1(-1) + c_1] & \varrho[b_2(-1) + c_2] & \varrho[b_3(-1) + c_3] \\ \varrho[b_1(0) + c_1] & \varrho[b_2(0) + c_2] & \varrho[b_3(0) + c_3] \\ \varrho[b_1(1) + c_1] & \varrho[b_2(1) + c_2] & \varrho[b_3(1) + c_3] \end{pmatrix}$$

is nonsingular. Hence, for all $R = \text{COL}\,(r_{-1}, r_0, r_1)$ there exists a vector $A = \text{COL}\,(\omega_1, \omega_2, \omega_3)$ so that

$$\Sigma A = R,$$

and this solves our problem with $\omega_0 = 0$. We will prove this latter property for a certain function $\tilde{\varrho}$ of the form $a\varrho(x) + b$, and this will imply the result for ϱ.

Let m_i, $i = 1, 2, 3$ be maps on the real numbers so that $(m_i(-1), m_i(0), m_i(1)) = e_i$ ($e_i \in R^3$ is the ith canonical vector). Let $U = \{-1, 0, 1\}$. We say that k ϱ-neurons g_j *linearly interpolate[U] the map* m_i if there exist constants $\omega_1^i, \ldots, \omega_k^i$ so that $f_i(u) = \sum_{j=1}^{k} \omega_j^i g_j(u)$ and

$$f_i(u) = m_i(u)$$

for all $u \in U$. These neurons are said to ϵ-*approximate[U]* m_i if

$$|f_i(u) - m_i(u)| < \epsilon$$

for all $u \in U$.

Proposition 1. There are three $\mathcal{H}$-neurons (i.e., threshold neurons, also called Heaviside neurons) that interpolate[U] the maps m_i, $i = 1, 2, 3$.

Proof. Let

$$h_1(x) = \mathcal{H}\left(x + \frac{1}{2}\right)$$

$$h_2(x) = \mathcal{H}\left(x - \frac{1}{2}\right)$$

$$h_3(x) = \mathcal{H}\left(-x - \frac{1}{2}\right).$$

The interpolation is by: $m_1 = h_2$ ($w_1^1 = 0, w_2^1 = 1, w_3^1 = 0$), $m_2 = h_1 - h_2$ ($w_1^2 = 1, w_2^2 = -1, w_3^2 = 0$), and $m_3 = h_3$ ($w_1^3 = 0, w_2^3 = 0, w_3^3 = 1$). ■

Proposition 2. For all $\epsilon > 0$ there are three ϱ-neurons that ϵ-interpolate[U] the maps m_i, $i = 1, 2, 3$.

Proof. As in property $(\star)$ $t_+ \neq t_-$, we can impose $t_+ = 1$, and $t_- = 0$ on ϱ without restricting the affine span of the neurons. This is possible by defining the function

$$\tilde{\varrho} = \frac{\varrho(x) - t_-}{t^+ - t_-}.$$

Without loss of generality, we assume $t^+ = 1$ and $t_- = 0$ from now on. So, for each $\epsilon > 0$ there is some $\eta > 0$ such that for all y, $|y - \frac{1}{2}| > \eta$

$$\left|\varrho\left(y - \frac{1}{2}\right) - \mathcal{H}\left(y - \frac{1}{2}\right)\right| < \epsilon.$$

In particular $\forall y$, $|y - \frac{1}{2}| > \frac{1}{4}$ we can choose $\lambda > 4\eta$, and using that $\mathcal{H}(\lambda(y - \frac{1}{2})) = \mathcal{H}(y - \frac{1}{2})$,

$$\left|\varrho\left(\lambda\left(y - \frac{1}{2}\right)\right) - \mathcal{H}\left(y - \frac{1}{2}\right)\right| < \epsilon.$$

A similar argument can be applied to $(y + \frac{1}{2})$ and $(-y - \frac{1}{2})$. We conclude that for all $\epsilon > 0$ there exists some λ so that

$$\begin{aligned} g_1(x) &= \varrho\left(\lambda\left(x + \frac{1}{2}\right)\right) \\ g_2(x) &= \varrho\left(\lambda\left(x - \frac{1}{2}\right)\right) \\ g_3(x) &= \varrho\left(\lambda\left(-x - \frac{1}{2}\right)\right) \end{aligned}$$

satisfy $|g_i(x) - h_i(x)| < \epsilon/3$ for all $u \in U$. The result is now clear. ■

Now, define the three matrices:

$$\begin{aligned} \Sigma &= [g_j(u_i)]_{ij} \\ M &= [m_j(u_i)]_{ij} \\ W &= [\omega_j(u_i)]_{ij}. \end{aligned}$$

From Proposition 2 we conclude that for all $\epsilon > 0$ there are g_1, g_2, g_3 so that

$$\left|\begin{pmatrix} \varrho[b_1(-1) + c_1] & \varrho[b_2(-1) + c_2] & \varrho[b_3(-1) + c_3] \\ \varrho[b_1(0) + c_1] & \varrho[b_2(0) + c_2] & \varrho[b_3(0) + c_3] \\ \varrho[b_1(1) + c_1] & \varrho[b_2(1) + c_2] & \varrho[b_3(1) + c_3] \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right| < \epsilon.$$

Choosing ϵ so that Σ is nonsingular, the lemma results. ■

ACKNOWLEDGMENT

The author thanks Professor Eduardo Sontag for providing much useful advice.

REFERENCES

CLEEREMANS, A., SERVAN-SCHREIBER, D., and MCCLELLAND, J. 1989. Finite state automata and simple recurrent networks. Neural Computation **1**(3):372.

ELMAN, J. 1990. Finding structure in time. Cognitive Science **14**:179–211.

GILES, C. L., MILLER, C., CHEN, D., CHEN, H., SUN, G., and LEE, Y. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. Neural Computation **4**(3).

HARTLEY, R., and SZU, H. 1987. A comparison of the computational power of neural network models. *In* Proceedings of IEEE Conf. Neural Networks, pp. 17–22.

KILIAN, J., and SIEGELMANN, H. 1993. On the power of sigmoid neural networks. *In* Proceedings, 6th ACM Workshop on Computational Learning Theory.

KLEENE, S. C. 1956. Representation of events in nerve nets and finite automata. *In* Automata studies. Edited by C. Shannon and J. McCarthy. Princeton Univ. Press, Princeton, NJ, pp. 3–41.

MATTHEWS, M. 1992. On the uniform approximation of nonlinear discrete-time fading-memory systems using neural network models. Technical report Ph.D. thesis, ETH No. 9635, E.T.H. Zurich.

MCCULLOCH, W. S., and PITTS, W. 1943. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**:115–133.

POLLACK, J. B. 1987. On connectionist models of natural language processing. Ph.D. dissertation, Computer Science Dept. Univ. of Illinois, Urbana, IL.

POLLACK, J. 1990. The induction of dynamical recognizers. Technical report 90-JP-Automata, Dept. of Computer and Information Science, Ohio State Univ.

POLYCARPOU, M. M., and IOANNOU, P. 1991. Identification and control of nonlinear systems using neural network models: design and stability analysis. Technical report 91-09-01, Department of EE/Systems, UCS, Los Angeles.

SIEGELMANN, H. 1993. Foundations of recurrent neural networks. Ph.D. dissertation, Rutgers University.

SIEGELMANN, H. T., and SONTAG, E. D. 1991. Turing computability with neural nets. Appl. Math. Lett. **4**(6):77–80.

SIEGELMANN, H. T., and SONTAG, E. D. 1994. Analog computation via neural networks. Theoretical Computer Science **131**:331–360.

SIEGELMANN, H. T., and SONTAG, E. D. 1995. On the computational power of neural networks. J. Comput. Syst. Sci. **50**(1):132–150.

SONTAG, E. 1992. Neural nets as systems models and controllers. *In* Proceedings, 7th Yale Workshop on Adaptive and Learning Systems, pp. 73–79.

WILLIAMS, R., and ZIPSER, D. 1989. A learning algorithm for continually running fully recurrent neural networks. Neural Computation **1**(2).