Entry **"Automata, Neural Networks as"**
for *Handbook of Brain Theory and Neural Networks*
(M.A. Arbib, ed, MIT Press, 1995)
This draft dated August 25, 1994

Eduardo D. Sontag

1 Introduction

An automaton, or sequential machine, is a device which evolves in time, reacting to external stimulii and in turn affecting its environment through its own actions. In computer science and logic, *Automata Theory* deals with various formalizations of this concept.

Artificial *Neural Networks*, as understood in this article, are systems obtained from a finite number of memory-free scalar elements or "neurons" by means of linear interconnections. The complete system is updated synchronously, in discrete time steps, and the transmission of information among neurons requires a unit time delay. The term *recurrent* or *feedback* network is sometimes used in order to emphasize the fact that possible loops in signal paths may allow neurons to mutually affect each other.

In this formal sense, neural networks constitute a (very) particular type of automaton. 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.

Several variants of the notion of automata are possible, depending on the type and availability of "external" secondary memory storage. Similarly, many different types of neural networks can be conceived of, depending on the time scales of operation and the types of signals transmitted among neurons. This article will present some basic background about automata and will briefly explain how such objects can be compared to, and simulated by, neural networks.

The area of relations between automata and neural nets is an old one, dating back at least to the work of the neurophysiologists McCulloch and Pitts ([McCulloch and Pitts, 1943]), but it is also one of active current research, so only a small fraction of the topic can be covered in this short article; pointers to relevant literature are given in the references. For the same reason, most ideas will be only discussed in an intuitive and sketchy fashion, with mathematical details omitted.

2 Automata

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 beginings of 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 random access 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 of input.) Thus one postulates an update equation of the type

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

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

$$y(t) = h(x(t)) \tag{2}$$

provides the output at time t associated to the internal state at that instant.

Abstractly, an automaton is defined by the above data. As a mathematical object, an *automaton* is simply a quintuple

$$\Sigma = (X, U, Y, f, h)$$

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

$$f : X \times U \to X , \quad h : X \to Y$$

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

The Finiteness Assumption

It would appear on first thought that it is sufficient in practice to restrict studies to finite automata. After all, only a finite amount of memory is available in any computer. However, even for digital computation, finiteness imposes theoretical constraints which are undesirable when one is interested in the understanding of ultimate computational capabilities. As a trivial illustration, assume that one wishes to design a program which reads an input string of 0's and 1's and, *after* this string ends, displays the same string in its output. A finite automaton cannot accomplish this task, obviously, since the task requires an unbounded amount of memory (unless one knows in advance that the strings to be memorized and repeated will be of no more than a certain predetermined length, in which case enough memory, represented by a certain number of states, can be preallocated for storage). On the other hand, one could certainly write a computer program, in any modern programming language, to perform this task. The program will instruct the computer to write the string into a file as it is being received, to be later retrieved when $u(t) = \#$ is encountered. This program will execute correctly as long as enough external storage (e.g., in the form of disk drives) is potentially available.

A mathematical model more general than finite automata allows for "external" storage in addition to the information represented by the current "internal" state of the system. This is the *Turing Machine* model, introduced by the English mathematician Alan Turing in 1936, and it forms the basis of most of modern computer science. In a Turing machine, a finite automaton is used as a "control" or main computing unit, but this unit has access to a potentially infinite read/write storage device. The entire system, consisting of the control unit and the storage device, together with the rules that specify access to the storage, can be seen as a particular type of infinite automaton, albeit one with a very special structure. It is widely accepted today in the computer science community that no possible digital computing device can be more poweful, except for relative speedups due to more complex instruction sets or parallel computation, than a Turing machine.

It has been known at least since [McCulloch and Pitts, 1943] that finite automata can be simulated by (recurrent) neural networks; this fact is stated formally later in this article. One may wish to ask as well about neural simulation of infinite automata, such as Turing machines. Such a simulation can be approached in at least two different ways. The first is to make a distinction between "RAM" and secondary storage for neural networks, in which case the simulation of a finite automata (the control part) is all that is needed, and details of the secondary storage implementation are not studied. An alternative possibility, perhaps more reasonable from a biological standpoint, is to blur this distinction and to allow the extra memory to be represented by quantities also associated to neurons, such as activation levels or concentrations of neurotransmitters at synapses. In this latter case, a more sophisticated mathematical analysis is required.

Types of Machines Not Covered in this Article

In this article it is assumed that all behavior is deterministic. The field of *probabilistic* automata (cf. [Paz, 1971]) deals with random effects, and will not be studied here.

Note that the definition of automata implicitely assumes that all attention is restricted to operation at discrete instants of time, or *epochs* $t = 0, 1, 2, \ldots$. On the other hand, many physical devices (and "analog computers") are most naturally described by means of continuous quantities such as voltages or forces, evolving perhaps in continuous time, and are subject to continuous-valued external signals. Such more general devices can be studied as well, leading to connections between automata theory and control systems theory; see [Padulo and Arbib, 1974], [Sontag, 1990]. Various areas of continuous mathematics, including differential equation theory, are relevant in that study.

3 Recurrent Neural Nets

As mentioned earlier, this article deals with recurrent neural networks. These are devices built by linearly combining a finite number n of memory-free scalar processing units, each of which performs the same nonlinear transformation $\sigma : \mathbb{R} \to \mathbb{R}$ on a linear combination of its inputs. The units are interconnected through unit delays. One may describe such a system by introducing n real-valued variables x_i, $i = 1, \ldots, n$ which represent the internal state of the ith processor respectively, subject to update equations such as:

$$x_i(t+1) \;=\; \sigma\left(\sum_{j=1}^{n} a_{ij}x_j(t) + \sum_{j=1}^{m} b_{ij}u_j(t)\right) \;, \tag{3}$$

where each u_i, $i = 1, \ldots, m$, is an external input signal. The coefficients a_{ij}, b_{ij} denote the weights, intensities, or "synaptic strengths" of the various connections. The function $\sigma : \mathbb{R} \to \mathbb{R}$ which appears in all the equations, is called the "activation function" and is often taken to be a sigmoidal-type map, as discussed below; it characterizes how each neuron responds to its aggregate input. One also assumes given a certain number p of probes, or measurement devices, whose outputs signal to the environment the collective response of the net. Each such device averages the activation values of many neurons. Mathematically, this is modelled by adding a set of functions

$$y_i(t) \;=\; \sum_{j=1}^{n} c_{ij}x_j(t)\,, \quad i = 1, \ldots, p\,. \tag{4}$$

The coefficient c_{ij} represents the effect of the jth neuron on the ith measurement.

As defined, a recurrent neural net is a particular type of automaton, with state space $X = \mathbb{R}^n$, input space $U = \mathbb{R}^m$, and output space $Y = \mathbb{R}^p$.

In many studies, starting with the classical work by McCulloch and Pitts ([McCulloch and Pitts, 1943]), the function σ is taken to be the hardlimiter, threshold, or *Heaviside* function $\mathcal{H}(x)$, which equals 1 if $x > 0$ and 0 for $x \leq 0$. Often one wants a differentiable saturation, and for this, especially in contemporary neural networks studies, it is customary to consider the *standard sigmoid* $\sigma(x) = (1 + e^{-x})^{-1}$ or equivalently, up to translations and change of coordinates, the hyperbolic tangent $\tanh(x)$. Also common in practice is a piecewise linear function, $\pi(x) := x$ if $|x| < 1$ and $\pi(x) = \operatorname{sign}(x)$ otherwise; this is sometimes called a "semilinear" or "saturated linearity" function. In this article, it will only be assumed that σ is a *sigmoid* (sometimes called a "squashing" function): σ is monotone (not necessarily continuous), bounded, and not constant. Note that necessarily both $\lim_{x\to-\infty}\sigma(x)$ and $\lim_{x\to+\infty}\sigma(x)$ exist and are distinct. The above activation functions are all sigmoids.

4 Simulations

In order to precisely state that neural nets can do everything that a finite automata can, one needs to introduce the notion of simulation.

In general, given an automaton $\Sigma = (X, U, Y, f, h)$, 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,

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

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

$$\text{ENC} : X \to \overline{X} \quad \text{and} \quad \text{DEC} : \overline{X} \to X ,$$

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

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

Assume that for some integer m the input value set U consists of the vectors $e_1, \ldots, e_m$ in $\mathbb{R}^m$, 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. Similary, suppose that Y consists of the vectors $e_1, \ldots, e_p$ in $\mathbb{R}^p$. (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.) There holds then the following "Simulation Theorem:"

Theorem. *Every finite automaton can be simulated by a neural network with activation function $\sigma = \mathcal{H}$.*

See [Alon, Dewdney, and Ott, 1991] for a study of the minimal number of neurons (the dimension n) required for the simulation; the theory of threshold functions, as covered in [Muroga, 1971], is useful in this context. The implementation of finite automata by networks has been suggested in some application areas; see for instance [Cleeremans, Servan-Schreiber, and McClelland, 1989]. For more on automata, their behaviors, and relations to neural networks, the reader may wish to consult [Kleene, 1956, Rosenblatt, 1962, Hopcroft and Ullman, 1979, Minsky, 1967, Arbib, 1987].

A sketch of a proof of the theorem is as follows. Assume that the states of the finite automaton Σ to be simulated are $\{\xi_1, \ldots, \xi_N\}$. A neural network that simulates Σ has $n = Nm + 1$ neurons and is built as follows. Denote the coordinates of the state vector $x \in \mathbb{R}^n$ by x_{ij}, $i = 1, \ldots, N$, $j = 1, \ldots, m$, and x_0. (This last coordinate will be identically equal to 1 in the simulation.) In terms of these coordinates, the update equations are $x_0^+ = \mathcal{H}(x_0)$ and, for $r = 1, \ldots, N$, $s = 1, \ldots, m$:

$$x_{rs}^+ = \mathcal{H}\left(\sum_{j=1}^{m} \sum_{i \in S_{rs}} x_{ij} - x_0 + u_s\right)$$

where

$$S_{rs} := \{l \mid f(q_l, e_s) = q_r\} .$$

For each $l = 1, \ldots, p$, the lth coordinate of the output is defined as

$$y_l = \sum_{j=1}^{m} \sum_{i \in T_l} x_{ij}$$

where $T_l := \{i | h_l(q_i) = 1\}$ and h_l is the lth coordinate of h. The encoding map is

$$\text{ENC}\,[q_r] := e_{r1} + e_0$$

where e_{ij} and e_0 denote the canonical basis vectors in the coordinates x_{ij} and x_0 respectively. The decoding map is

$$\text{DEC}\,[e_{ij} + e_0] = q_i$$

for all i, j, and is arbitrary on all other elements of $\mathbb{R}^n$. The proof that this is indeed a simulation is immediate, based on the observation that, provided that the starting state has the form $e_{lj} + e_0$, at each instant the state of the network has the same special form, where q_l is the corresponding state of the original automaton. Note that the expression $\sum_{j=1}^{m} \sum_{i \in S_r} x_{ij} - x_0 + u_s = \sum_{j=1}^{m} \sum_{i \in S_r} x_{ij} + u_s - 1$ can only take the values -1, 0, or 1, because only one of the x_{ij} can be equal to one at any given time. Moreover, the value 1 can only be achieved for this sum if both $u_s = 1$ and there is some i so that $x_{ij} = 1$, that is, if the current state of the original machine is q_i and $f(q_i, e_s) = q_r$. Thus the next state is $e_{rs} + e_0$ precisely if the next state of Σ is q_r and the input applied is e_s.

Unbounded Machines

As remarked earlier, it follows as a simple consequence of the theorem just stated that every possible Turing machine (i.e., any digital computer) can also be simulated by a neural network, provided that an external memory be assumed. It suffices to simulate the control part of the Turing machine by a net. However, this begs the question of what are the capabilities of neural networks when no such *additional* memory is allowed. This issue is discussed next.

It is not very difficult to establish that the above simulation theorem is true whatever sigmoid σ one wishes to use, not merely the Heaviside map $\mathcal{H}$. If this sigmoid happens to be $\mathcal{H}$, which is used in most of the classical neural networks literature, then no more than finite automata can be realized without additional memory. This is because after the first time step the states of the system are forced to lie in the finite set $\{0, 1\}^n$. Observe however that neural networks might be more powerful than finite automata provided that the next-state mapping be allowed to have an infinite range, that is, if the sigmoid σ itself can take infinitely many possible values.

This latter observation was exploited in the paper [Siegelmann and Sontag, 1992] (see also the paper [Cosnard, Garzon, and Koiran, 1993], which expanded on this work) to show that, if one uses the piecewise linear sigmoid $\sigma(x) = \pi(x)$ introduced earlier, a novel simulation result is possible. It turns out that all Turing machines can be simulated by neural networks, with no recourse to an artificial secondary storage device. The basic idea is to take advantage of the unbounded precision of the state variables x_i to store information. The construction results in a simulation of a universal Turing machine by a recurrent network which employs roughly 1000 neurons. The main mathematical difficulty in the proof lies in the fact that a continuous sigmoid does not allow, in principle, for discontinuous logical decisions of the "if $x \leq 0$ then do (...) else do (...) " type, and such decisions are required in general programs. This drawback was overcome by the use of a Cantor set representation for the storage of activation values. Cantor sets permit making binary decisions by means of finite precision devices, taking advantage of the fact that no values may ever appear in the "middle" range.

The simulation of Turing machines by sigmoidal networks in effect substitutes one assumption, namely that a potentially infinite external storage device exists, by another, namely the hypothesis that unlimited precision is possible in neural computation and in the storage of activations. Neither assumption (infinite finite precision storage units nor finitely many infinite precision cells) is of course valid in the real world, but the two models represent different idealizations of the potential unlimited availability of resources. By studying these mathematical idealizations. one gains a better understanding of the power and limitations of real devices.

While classical computation theory stops essentially with Turing machines, in the context of neural networks, seen as analog computing devices, there is no need to do so. If weights and activations are allowed to assume arbitrary real values, neural networks with continuous sigmoidal σ are able to "compute" more than digital computers. A precise study of such analog computations was carried out in the paper [Siegelmann and Sontag, 1993], which characterized the information processing power of neural networks under resource constraints, in particular in terms of computation time. One of the the main results in that reference is that the class of languages recognized in polynomial time using recurrent nets, called there "circuit P" or "analog P" is exactly the same as a class also studied in computer science, namely the class of languages recognized in polynomial time by Turing machines which consult oracles, where the oracles are sparse sets. This gives a precise characterization of the power of recurrent nets in terms of a known (non-computable in the classical sense) complexity class. In summary, even though networks, as analog devices, can "compute" far more than digital computers, they also give rise to a rich theory of computation. The ultimate implications of this characterization for analog implementations of neural computers are as yet unclear, but the result implies that many problems, namely those *not* in the class analog-P, cannot be expected to be solved efficiently even in analog computers such as neural networks.

The reference [Sontag, 1993] provides an approximation type of result. It shows (with a proof similar to the one sketched above) that arbitrary machines $\Sigma = (X, U, Y, f, h)$, even infinite, but supposing now that X as well as U and Y have a topological structure and the maps f and h are continuous, can be *approximated* arbitrarily well, on compact subsets, by neural networks with sigmoidal activations. Various theoretical aspects of neural network theory are explored in that paper as well.

References

[Alon, Dewdney, and Ott, 1991] Alon, N., A.K. Dewdney, and T.J. Ott, "Efficient simulation of finite automata by neural nets," *J. A.C.M.* 38 (1991): 495-514.

[Arbib, 1987] Arbib, M.A., *Brains, Machines, and Mathematics, Second Edition*, Springer, 1987.

[Cleeremans, Servan-Schreiber, and McClelland, 1989] Cleeremans, A., D. Servan-Schreiber, and J.L. McClelland, "Finite state automata and simple recurrent networks," *Neural Computation* **1** (1989): 372-381.

[Cosnard, Garzon, and Koiran, 1993] Cosnard, M., M. Garzon, and P. Koiran, "Computability properties of low-dimensional dynamical systems," in *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science*, Springer-Verlag, Berlin, 1993. Also *Theor. Comp. Sci.*, to appear.

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

[Kleene, 1956] Kleene, S.C., "Representation of events in nerve nets and finite automata," in Shannon, C.E., and J. McCarthy, eds., *Automata Studies*, Princeton Univ. Press 1956: 3-41.

[McCulloch and Pitts, 1943] McCulloch, W.S., and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.* **5**(1943): 115-133.

[Minsky, 1967] Minsky, M.L., *Computation: Finite and Infinite Machines*, Prentice Hall, Engelwood Cliffs, 1967.

[Muroga, 1971] Muroga, S., *Threshold Logic and its Applications*, Wiley, New York, 1971.

[Padulo and Arbib, 1974] Padulo, L., and M.A. Arbib, *System Theory. A Unified State-Space Approach to Continuous and Discrete Systems*, W.B. Saunders, Philadelphia, 1974. (Publishing continued by Hemisphere Publ., Washington, DC.)

[Paz, 1971] Paz A., *Introduction to Probabilistic Automata* Academic Press, New York, 1971.

[Rosenblatt, 1962] Rosenblatt, F., *Principles of Neurodynamics*, Spartan Books, New York, 1962.

[Siegelmann and Sontag, 1992] Siegelmann, H.T., and E.D. Sontag, "On the computational power of neural nets," in *Proc. Fifth ACM Workshop on Computational Learning Theory*, Pittsburgh, July 1992, 440-449. To appear in *J. Computer Sys. Sci.*.

[Siegelmann and Sontag, 1993] Siegelmann, H.T., and E.D. Sontag, "Analog computation via neural networks," in *Proc. 2nd Israel Symposium on Theory of Computing and Systems (ISTCS93)*, IEEE Computer Society Press, 1993. Also see "Analog computation, neural networks, and circuits," *Theor. Comp. Sci.*, to appear.

[Sontag, 1993] Sontag, E.D., "Neural networks for control," in *Essays on Control: Perspectives in the Theory and its Applications* (H.L. Trentelman and J.C. Willems, eds.), Birkhauser, Boston, 1993, pp. 339-380.

[Sontag, 1990] Sontag, E.D., *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, Springer, New York, 1990.