AN OVERVIEW OF THE COMPUTATIONAL POWER OF RECURRENT NEURAL NETWORKS

Pekka Orponen

University of Jyväskylä, Department of Mathematics
FIN-40351 Jyväskylä, Finland

We present an overview of some of the work done in the past ten years concerning the power of recurrent neural network models as general computational devices.

1. INTRODUCTION

The two main streams of neural networks research consider neural networks either as a powerful family of nonlinear statistical models, to be used in for example pattern recognition applications [6], or as formal models to help develop a computational understanding of the brain [10]. Historically, the brain theory interest was primary [32], but with the advances in computer technology, the application potential of the statistical modeling techniques has shifted the balance.[1]

The study of neural networks as general computational devices does not strictly follow this division of interests: rather, it provides a general framework outlining the limitations and possibilities affecting both research domains. The prime historic example here is obviously Minsky's and Papert's 1969 study of the computational limitations of single-layer perceptrons [34], which was a major influence in turning away interest from neural network learning to symbolic AI techniques for more than a decade. A less dramatic, but at least as significant example is Kleene's 1956 paper [25] presenting an algebraic characterization of the computations feasible in finite McCulloch–Pitts neural networks — and thereby introducing the notion of regular expressions and their connection to finite automata. An interesting recent instance of an ambitious research programme aspiring to connect computation theory to real biology is Wolfgang Maass's work on the computational capabilities of pulse coded neural networks [29, 30].

In this overview, we focus on the computational power of *recurrent* (cyclic) neural networks. The computational study of *feedforward* (acyclic) networks has intimate connections to the classical theory of Boolean circuit complexity [53], and is surveyed briefly in the articles [36, 40], and at greater depth in the books [41, 43, 52].

2. BASIC NOTIONS AND RESULTS

With the brief exception of Section 6 on continuous-time models, we shall be concerned with *finite discrete-time recurrent networks*. Such a network consists of n computational units or *neurons*, indexed as $1, \ldots, n$, with interconnections described by an *architecture graph*. Each edge in the architecture graph, leading from neuron i to j, is labeled with the corresponding *interconnection weight* w_{ji}. An important special case are the *symmetric* or *Hopfield networks*, whose architecture is given by an undirected graph with symmetric

[1] Of course, these research orientations are not mutually exclusive, but coexist and interact in productive ways.

weights $w_{ij} = w_{ji}$ for every pair of neurons i, j. At each discrete time instant $t = 0, 1, \ldots$, every neuron j in the network has a well-defined *state* $y_j^{(t)}$, which in a *binary-state* network comes from the set $\{0, 1\}$, and in an *analog-state* network from the interval $[0, 1]$.

We shall mostly be concerned with the *synchronous fully parallel* dynamics, under which the evolution of the global network state $\mathbf{y}^{(t)} = (y_1^{(t)}, \ldots, y_n^{(t)}) \in [0, 1]^n$ is determined for all discrete time instants $t = 0, 1, \ldots$ as follows. At the beginning of the computation, the network is placed in an initial state $\mathbf{y}^{(0)}$ which may include an external input. At discrete time $t \geq 0$, each neuron $j = 1, \ldots, n$ collects its inputs from the states $y_i^{(t)} \in [0, 1]$ of incident neurons i. Then its *excitation* ξ_j is computed as a weighted sum of its inputs, $\xi_j^{(t)} = \sum_{i=0}^{n} w_{ji} y_i^{(t)}$ $(j = 1, \ldots, n)$. This sum may include a local *bias* term w_{j0}, which can formally be viewed as a weighted input derived from a constant-state unit $y_0^{(t)} = 1$, $t \geq 0$. At the next instant $t + 1$, an *activation function* σ is applied to $\xi_j^{(t)}$ for all neurons $j = 1, \ldots, n$ in order to determine the new network state $\mathbf{y}^{(t+1)}$ as follows:

$$y_j^{(t+1)} = \sigma\left(\xi_j^{(t)}\right) \quad j = 1, \ldots, n.$$

In binary neural networks the activation function is the *step* or *Heaviside* function

$$\sigma(\xi) = \begin{cases} 0, & \text{for } \xi < 0, \\ 1, & \text{for } \xi \geq 0, \end{cases}$$

and in analog networks the activation function is some continuous sigmoid, such as $\sigma = \tanh$, or the *saturated-linear* function

$$\sigma(\xi) = \begin{cases} 0, & \text{for } \xi \leq 0, \\ \xi, & \text{for } 0 < \xi < 1, \\ 1, & \text{for } \xi \geq 1. \end{cases}$$

Alternative computational dynamics are also possible. For example, under *sequential mode* only one neuron updates its state at each time instant. Also various block-parallel dynamics can be considered, but we do not discuss them here (see [12, 15]).

A fundamental property of symmetric networks is that their dynamics are constrained by Liapunov, or "energy" functions. A *Liapunov function* E is a bounded real-valued function defined on the state space of a network, whose values $E\left(\mathbf{y}^{(t)}\right)$ are properly decreasing along any non-constant (*productive*) computation path $\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots$. It follows from the existence of such a function E that the network state must in any computation converge towards some simple attractor described by the respective minimum value of E.

Consider, for example, a sequentially updated binary symmetric network with zero biases $w_{j0} = 0$ and feedbacks $w_{jj} = 0$. (Nonnegative feedbacks are actually necessary in symmetric sequential nets to guarantee the Liapunov property.) Without loss of generality [41], one may also assume non-zero excitations $\xi_j^{(t)} \neq 0$, $j = 1, \ldots, n$. It was then observed by Hopfield [20] (see also [11, 16]) that the following energy function

$$E\left(\mathbf{y}^{(t)}\right) = E(t) = -\frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} w_{ji} y_i^{(t)} y_j^{(t)},$$

has the property that $E(t) \leq E(t-1) - 1$ for every update step $t \geq 1$ of a productive computation. Moreover, the energy function is bounded, i.e. $|E(t)| \leq W$, where

$$W = \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} |w_{ji}|$$

is the *weight* of the network. Hence, the computation must converge to a stable state where no more neuron state changes occur, within $O(W)$ update steps. An analogous result can be shown for synchronous dynamics in binary nets; however in this case the network may also converge to a limit cycle of two alternating states [8, 13, 42].

It also follows from these considerations that if the weights w_{ij} in a symmetric network are polynomially bounded in the number of units n, then the network converges in polynomial time. Conversely, it can be shown that networks with exponentially large weights may indeed require an exponential time to converge. This was first shown in [17] for synchronous updates (a simplified construction appears in [14]), and in [19] for a particular sequential update rule. (For a different sequential rule the result actually follows already from [17] or [14] by a fairly simple construction.) Finally, a network requiring exponential time for convergence under *any* sequential update rule was demonstrated in [18].[2]

3. FINITE BINARY-STATE NETWORKS

It has been known since the early work of McCulloch and Pitts [32] and Kleene [25] that finite binary-state neural networks are equivalent to finite automata for processing sequentially given inputs. A somewhat interesting question here is how efficient are neural nets as representations of finite automata. The elementary constructions [33] yield a network of about $2m$ binary-state neurons for simulating an m-state automaton, and it was proved only relatively recently in [1] that at least $\Omega((m \log m)^{1/3})$ neurons are really required in the worst case.t The upper bound was further improved to $O(m^{1/2})$ neurons in [22, 23], and it was shown that under some additional constraints this upper bound is tight.

The case of sequence processing by *symmetric* binary-state networks was considered in [48, 51], where it was shown that this model is properly weaker than finite automata, and the respective subclass of the regular languages, so called *Hopfield languages* was characterized. More precisely, it was shown in [48, 51] that a regular set of sequences $S \in \{0,1\}^*$ can be recognized by a symmetric binary-state neural network, if and only if for every pair of sequences $u, v \in \{0,1\}^*$, and for every two-symbol sequence $ab \in \{0,1\}^2$, either $u(ab)^k v \in S$ or $u(ab)^k v \notin S$ holds uniformly for all sufficiently large values of k.

4. SEQUENCES OF BINARY-STATE NETWORKS

In finite automaton -type models, the input is presented to the network as a sequence of pulses, so that a single net is capable of processing arbitrary long input sequences. Another convention, closer to many applications' point of view, is to load the input initially to a set of designated input neurons, and then let the network run unintervened until it (possibly) converges, at which point the output is read from a set of designated output neurons (in the case of Boolean function computations, a single output neuron). To be able to process arbitrary input sizes, this formulation requires that the network grows with increasing input size, i.e., that we actually consider nonuniform *sequences* of networks,

[2]The manuscript [18] remains unpublished, but the construction is reviewed in [37]. The result can also be shown to follow, although in a somewhat convoluted way, from the more general theory of "PLS-completeness" for local optimization problems [44].

one for each input size. If arbitrary changes to the network structure are possible, then this model is fundamentally different from finite automata.

Since the computations of a binary recurrent net of n neurons converging in t update steps may be "unwound" into a binary feedforward net containing $n \cdot t$ neurons, polynomial time computations by polynomial-size binary recurrent nets can be simulated by polynomial-size binary feedforward nets. It then follows from standard results [5, 41, 53] that the class of Boolean functions computable by such networks coincides with the class P/poly of functions computable by "nonuniform" Turing machines in polynomial time. This is basically the standard complexity class P, with a certain technical proviso introduced in [24] to account for changing the machine structure for different input sizes.

More interesting results can be obtained concerning *unbounded* time computations by binary recurrent networks. A folklore result, apparently first formulated in print in [28], states that polynomially space-bounded Turing machines can be simulated by polynomial-size binary recurrent nets. The idea of the construction is as follows: one starts with the standard simulation of a polynomially space-bounded Turing machine computation by a feedforward net of polynomial width and exponential depth [5]. A moment of thought shows that all the exponentially many layers of neurons in the resulting net are in fact similar, and so the net can be "folded" upon itself to create a recurrent net of polynomial size and exponential convergence time.[3] More precisely, one can show that the class of Boolean functions computed by polynomial size binary recurrent nets coincides with the nonuniform complexity class PSPACE/poly considered in [4, 24].

One might think that because of the Liapunov property discussed in Section 2, symmetric recurrent networks would be weaker computational devices than general asymmetric ones. For instance, symmetric networks cannot produce arbitrary oscillatory behavior, which seems to be an essential characteristic of general computation, and is also trivially created in asymmetric networks.

However, it was shown in [38] that infinite oscillations are in a sense the *only* feature of general-purpose (digital) computation that cannot be reproduced in symmetric recurrent networks. Even polynomial-size symmetric binary networks compute exactly the functions in the class PSPACE/poly, and when restricted to polynomially-bounded weights (implying polynomial-time convergence as discussed in Section 2), they compute exactly the functions in the class P/poly.

The proof in [38] proceeds by showing that an arbitrary *converging* computation of an asymmetric binary network of n neurons can be simulated by a symmetric binary network of $O(n^2)$ neurons. (The overhead was later reduced to $O(n)$ in [50].) The crucial observation is that because the simulated network is binary and deterministic, any converging computation on it must terminate within 2^n steps. (Otherwise the network repeats a configuration and goes into a nonterminating cycle.)

The core of the simulation is then the construction of an n-bit symmetric *clock* network (a simulated binary counter) that, using $O(n)$ neurons, produces a sequence of 2^n well-controlled oscillations before it converges. This sequence of clock pulses is used to drive the rest of the network, where each asymmetric edge is simulated by a symmetric, clock-latched constant-size subnetwork. (In the improved construction in [50], individual neurons, instead of edges, are simulated directly, leading to the indicated improvement in the size overhead.)

[3] An even more elementary way to see this result is to appeal to the intuition that a polynomially space-bounded Turing machine can be simulated by a polynomial amount of digital "hardware", and this hardware can be implemented using a polynomial number of binary neurons.

The crucial clock construction in [38, 50] is the same as was used in [14] to establish the exponentially slow convergence of symmetric binary networks under synchronous fully parallel updates. The analogous, but more complicated counter network for arbitrary sequential updates [18] was exploited in [38] to prove that even under sequential updates, symmetric binary networks have the same computational capabilities as the other models considered.

5. DISCRETE-TIME ANALOG NETWORKS

An interesting result proved by Siegelmann and Sontag in [46, 47] shows that if one moves from binary-state to analog-state neurons, then arbitrary Turing machines may be simulated by single, finite recurrent networks. The original construction in [46] required 1058 saturated-linear neurons to simulate a universal Turing machine, but this has later been improved to at least 114 neurons [27], and even to 25 neurons [23].

The starting point in these constructions, and also in many other recent simulations of Turing machines by finite-dimensional dynamical systems (e.g. [3, 7, 27, 35]), is the well-known correspondence of Turing machines and two-stack pushdown automata. The Turing machine tape is first represented as two opposing stacks, and then the contents of these stacks are encoded in some manner as two real numbers, further implemented as the states of two analog neurons. The other units in the network are then used to implement the stack operations and the finite-state control of the simulated Turing machine.

A natural question arises again concerning the computational power of *symmetric* analog networks. A Liapunov-function argument applies here too to show that under fully parallel updates such networks converge to a limit cycle of length at most two [26]. In this case the only possibility for a general Turing machine simulation on a single network would be to exploit finer and finer distinctions among a sequence of network states converging to a limit cycle. Such a simulation seems to be tricky at best, if possible at all.

A more reasonable approach is to augment the network with an external clock that produces an infinite sequence of binary pulses, thus providing it with an "energy source" for e.g. simulating an asymmetric analog network similarly as in [37]. Indeed, it is shown in [50] that the computational power of analog Hopfield nets with an external clock is the same as that of asymmetric analog networks.

The computational power of discrete-time analog networks is further discussed in the book [45]. However, the theoretical fascination of this topic is tempered by the practical observation that simulating arbitrarily long computations on a single finite network requires arbitrary-precision real number calculations, and these are in practice unavoidably corrupted by noise. In fact, it is shown in [9, 31] that any small amount of noise reduces the computational power of analog recurrent networks back to that of finite automata.

6. CONTINUOUS-TIME NETWORKS

In a continuous-time network, discussed for instance by Hopfield in [21], the dynamics of the network state $\mathbf{y}^{(t)} = (y_1^{(t)}, \ldots, y_n^{(t)}) \in [0,1]^n$ is determined for every real $t > 0$ by the following system of differential equations, with the initial network state $\mathbf{y}(0)$ providing

the initial conditions:

$$\frac{dy_j}{dt}(t) \;=\; -y_j(t) + \sigma(\xi_j(t)) \;=\; -y_j(t) + \sigma\left(\sum_{i=0}^{n} w_{ji} y_i(t)\right),\; j = 1, \ldots, n.$$

Similar Liapunov-function arguments as in the case of the other network types can be applied to prove the convergence of symmetric continuous-time networks [21].

Because of the simultaneous evolution of the neuron states, the dynamics of continuous-time networks are quite difficult to control. Nevertheless, it was shown in [39] that asymmetric continuous-time networks based on the saturated-linear activation function can simulate asymmetric binary-state networks with a linear size overhead, and in [49] that the same holds also for symmetric continuous-time networks with respect to converging computations. These results establish that also polynomial-size continuous-time networks have the full computational power of PSPACE/poly for arbitrary interconnection weights and P/poly for polynomially bounded weights. However, so far no *upper* bounds are known for the power of continuous-time nets, so conceivably they could be even *more* powerful than the corresponding binary models.

REFERENCES

[1] Alon, N., Dewdney, A. K., Ott, T. J. Efficient simulation of finite automata by neural nets. *J. Assoc. Comp. Mach. 38* (1991), 495–514.

[2] Anderson, J. A., Rosenfeld, E. (eds.) *Neurocomputing: Foundations of Research.* The MIT Press, Cambridge, MA, 1988.

[3] Asarin, E., Maler, O. On some relations between dynamical systems and transition systems. *Proc. 21st Internat. Colloq. on Automata, Languages, and Programming, Lecture Notes in Computer Science Vol. 820*, 59–72. Springer-Verlag, Berlin, 1994.

[4] Balcázar, J. L., Díaz, J., Gabarró, J. On characterizations of the class PSPACE/poly. *Theoret. Comput. Sci. 52* (1987), 251–267.

[5] Balcázar, J. L., Díaz, J., Gabarró, J. *Structural Complexity I, 2nd Ed.* Springer-Verlag, Berlin, 1995.

[6] Bishop, C. M. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[7] Branicky, M. Analog computation with continuous ODEs. *Proc. Workshop on Physics and Computation 1994,* 265–274. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[8] Bruck, J. On the convergence properties of the Hopfield model. *Proc. of the IEEE 78* (1990), 1579–1585.

[9] Casey, M. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation 8* (1996), 1135–1178.

[10] Churchland, P. S., Sejnowski, T. J. *The Computational Brain.* The MIT Press, Cambridge, MA, 1992.

[11] Fogelman, F., Goles, E., Weisbuch, G. Transient length in sequential iterations of threshold functions. *Discr. Appl. Math. 6* (1983), 95–98.

[12] Fogelman, F., Robert, Y., Tchuente, M. *Automata Networks in Computer Science: Theory and Applications.* Manchester University Press, 1987.

[13] Goles, E., Fogelman, F., Pellegrin, D. Decreasing energy functions as a tool for studying threshold networks. *Discr. Appl. Math. 12* (1985), 261–277.

[14] Goles, E., Martínez, S. Exponential transient classes of symmetric neural networks for synchronous and sequential updating. *Complex Systems 3* (1989), 589–597.

[15] Goles, E., Martínez, S. *Neural and Automata Networks.* Kluwer Academic, Dordrecht, 1990.

[16] Goles, E., Olivos, J. Comportement périodique des fonctions à seuil binaires et applications. *Discr. Appl. Math. 3* (1981), 93–105.

[17] Goles, E., Olivos, J. The convergence of symmetric threshold automata. *Information and Control 51* (1981), 98–104.

[18] Haken, A. Connectionist networks that need exponential time to stabilize. Manuscript, 10 pp., January 1989.

[19] Haken, A., Luby, M. Steepest descent can take exponential time for symmetric connection networks. *Complex Systems 2* (1988), 191–196.

[20] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. USA 79* (1982), 2554–2558.

[21] Hopfield, J. J. Neurons with graded response have collective computational properties like those of two-state neurons. In *Proc. Nat. Acad. Sci. USA 81* (1984), 3088–3092.

[22] Horne, B. G., Hush, D. R. Bounds on the complexity of recurrent neural network implementations of finite state machines. *Neural Networks 9* (1996), 243–252.

[23] Indyk, P. Optimal simulation of automata by neural nets. In *Proc. of the 12th Ann. Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science Vol. 900*, 337-348, Springer-Verlag, 1995.

[24] Karp, R. M., Lipton, R. J. Turing machines that take advice. *L'Enseignement Mathématique 28* (1982), 191–209. (Preliminary version in *Proc. of the 12th Ann. ACM Symp. on Theory of Computing.* ACM, New York, 1980. Pp. 302–309.)

[25] Kleene, S. C. Representation of events in nerve nets and finite automata. In: *Automata Studies* (ed. C. E. Shannon and J. McCarthy). Annals of Mathematics Studies n:o 34. Princeton Univ. Press, Princeton, NJ, 1956. Pp. 3–41.

[26] Koiran, P. Dynamics of discrete time, continuous state Hopfield networks. *Neural Computation 6* (1994), 459–468.

[27] Koiran, P., Cosnard, M., Garzon, M. Computability with low-dimensional dynamical systems. *Theoretical Computer Science 132* (1994), 113–128.

[28] Lepley, M., Miller, G. Computational power for networks of threshold devices in an asynchronous environment. Unpublished manuscript, Dept. of Mathematics, Massachusetts Inst. of Technology, 1983.

[29] Maass, W. Computation with spiking neurons. In *The Handbook of Brain Theory and Neural Networks, 2nd ed.* (ed. M. A. Arbib). The MIT Press, Cambridge, MA, to appear.

[30] Maass, W., Bishop, C. M. (Eds.) *Pulsed Neural Networks.* The MIT Press, Cambridge, MA, 1999.

[31] Maass, W., Orponen, P. On the effect of analog noise in discrete-time analog computations. *Neural Computation 10* (1998), 1071–1095.

[32] McCulloch, W. S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys. 5* (1943), 115–133. Reprinted in [2], pp. 18–27.

[33] Minsky, M. L. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, NJ, 1967.

[34] Minsky, M. L., Papert, S. A. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, Cambridge, MA, 1969 (expanded edition 1988).

[35] Moore, C. Unpredictability and undecidability in physical systems. *Phys. Review Letters 64* (1990), 2354–2357.

[36] Orponen, P. Computational complexity of neural networks: A survey. *Nordic Journal of Computing 1* (1994), 94–110.

[37] Orponen, P. Computing with truly asynchronous threshold logic networks. *Theoretical Computer Science 174* (1997), 123–136.

[38] Orponen, P. The computational power of discrete Hopfield nets with hidden units. *Neural Computation 8* (1996), 403–415.

[39] Orponen, P. The computational power of continuous time neural networks. In *Proc. of the SOFSEM Seminar on Current Trends in Theory and Practice of Informatics, Lecture Notes in Computer Science Vol. 1338*, 86–103, Springer-Verlag, Berlin, 1997.

[40] Parberry, I. A primer on the complexity theory of neural networks. In *Formal Techniques in Artificial Intelligence: A Sourcebook* (ed. R. B. Banerji). Elsevier – North-Holland, Amsterdam, 1990. Pp. 217–268.

[41] Parberry, I. *Circuit Complexity and Neural Networks.* The MIT Press, Cambridge, MA, 1994.

[42] Poljak, S., Sůra, M. On periodical behaviour in societies with symmetric influences. *Combinatorica 3* (1983), 119–121.

[43] Roychowdhury, V., Siu, K.-Y., Orlitsky, A. (Eds.) *Theoretical Advances in Neural Computation and Learning.* Kluwer Academic, Dordrecht, 1994.

[44] Schffer, A. A., Yannakakis, M. Simple local search problems that are hard to solve. *SIAM J. Computing 20* (1991), 56–87.

[45] Siegelmann, H. T. *Neural Networks and Analog Computation: Beyond the Turing Limit.* Birkhuser, Boston, MA, 1999.

[46] Siegelmann, H. T., Sontag, E. D. Analog computation via neural networks. *Theoretical Computer Science 131* (1994), 331–360.

[47] Siegelmann, H. T., Sontag, E. D. Computational power of neural networks. *Journal of Computer System Science 50*, 132–150, 1995.

[48] Šíma, J. Hopfield languages. In *Proc. of the SOFSEM Seminar on Current Trends in Theory and Practice of Informatics, Lecture Notes in Computer Science Vol. 1012*, 461–468, Springer-Verlag, 1995.

[49] Šíma, J., Orponen, P. A continuous-time Hopfield net simulation of discrete neural networks. In *Proc. NC'2000: Second International ICSC Symposium on Neural Computing*, 36-42. ICSC Academic Press, Wetaskiwin, Canada, 2000.

[50] Šíma, J., Orponen, P., Antti-Poika, T. On the computational complexity of binary and analog symmetric Hopfield nets. *Neural Computation*, to appear.

[51] Šíma, J., Wiedermann, J. Theory of neuromata. *J. Assoc. Comput. Mach. 45* (1998), 155–178.

[52] Siu, K.-Y., Roychowdhury, V., Kailath, T. *Discrete Neural Computation.* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[53] Wegener, I. *The Complexity of Boolean Functions.* John Wiley & Sons, Chichester, and B. G. Teubner, Stuttgart, 1987.