On the computational efficiency of symmetric neural networks*

Juraj Wiedermann

VUSEI-AR, Dúbravská 3, 842 21 Bratislava, Czechoslovakia

Abstract

Wiedermann, J., On the computational efficiency of symmetric neural networks, Theoretical Computer Science 80 (1991) 337-345.

An open problem concerning the computational power of neural networks with symmetric weights is solved. It is shown that these networks possess the same computational power as general networks with asymmetric weights; i.e., these networks can compute any recursive function. The computations of these networks can be described as a minimization process of a certain energy function; it is shown that for uninitialized symmetric neural networks this process presents a Σ_2-complete problem.

1. Introduction

With the advent of neural computers a new computational paradigm is emerging saying that certain collective spontaneous properties of physical systems can be used to immediately realize the computations. This gives rise to a brand-new class of computational devices in which the physics of the machine is intimately related to the algorithm of computations. The prominent representatives of such machines are Hopfield neural networks [8, 9], Boltzmann machines [1], and spin glasses [2].

So far these machines have been experimentally used for solving various isolated problems, such as associative memory realizations [8, 9], solving some combinatorial problems [1, 10] or simple models of learning [1].

Despite some promising experimental evidence of these machines, a satisfactory complexity theory that would answer the general questions concerning their computational power and efficiency is emerging only slowly.

In what follows we shall concentrate our attention on the neural networks with symmetric interconnections as represented by a so-called Hopfield model [8].

First, in Section 2, we briefly review the computational model used in developing our results. Then, in Section 3, we show that the computational power and efficiency of these networks is equivalent to that of neural circuits that are known to be

* A preliminary version of this paper appeared in: *Proc. 14th Symp. on Mathematical Foundations of Computer Science. MFCS'89*, Porabka-Kozubnik, Poland (1989) 545-552.

0304-3975/91/$03.50 © 1991—Elsevier Science Publishers B.V.

equivalent to unbounded fan-in linear threshold circuits [12]. This means that these networks present not only a universal tool but, at the same time, they present a computational tool as efficient as we can imagine (at least from the time complexity point of view). When restricted to bounded fan-in they belong to a so-called second machine class [14] while with unrestricted fan-in they can compute any boolean function in constant time.

The close connection between computations of neural networks with symmetric weights and certain physical processes is exemplified by a so-called energy function that can be associated with each neural network. Any computation of these networks can be seen as a minimization process of the corresponding energy function. In Section 4 we study the relation between non-deterministic computations and the energy function minimization problem. Here we show that the process of minimizing the energy function of an uninitialized neural network presents a Σ_2-complete problem (where Σ_2 denotes the complexity class in Stockmeyer's polynomial time hierarchy; see e.g. [3] or [7]) and we shall formulate some consequences of this result.

2. Neural networks definition

We shall consider a model similar to the original Hopfield model of neural networks [8] that uses two-state linear threshold "neurons". Each *neuron* u_i in this network can enter two different *states* 0 (inactive) or 1 (active) as characterized by its *output* x_i. There is a so-called *threshold value* t_i assigned to each neuron u_i. Each neuron has an arbitrary number of input and output *connections* that are labeled by *weights*. The *total input* to each neuron u_i at any moment is given by the sum $h_i = \sum_{j=1}^{n} a_{ij}x_j$, where a_{ij} is the weight of the u_i's input connection leading from u_j to u_i, x_j is the state of u_j at a given moment and n is the total number of neurons in the network.

The set of neurons in any neural network can be partitioned into two disjoint subsets: the set of all *initialized neurons* and the set of all *uninitialized neurons*; either of these two sets can be empty. There is a distinguished subset of initialized neurons, the set of *input neurons*.

The *computation* of such a system on a given input starts by initializing the states of input neurons to corresponding *input values* (0 or 1) and the states of remaining neurons (if any) in the set of initialized neurons to corresponding prescribed initial values which do not depend on input values. The neurons from the uninitialized set can be left in arbitrary states.

The description of states of all neurons in the network at any moment is called a *configuration* of that network at that moment.

The network works in an asynchronous way; each neuron u_i samples its inputs at random moments independently of other neurons and if $h_i > t_i$ the output x_i is set to 1, otherwise to 0. We shall suppose that this action takes an infinitely small amount of time and that within the entire network the actions of all neurons are accomplished within a bounded time interval, a so-called *computational cycle*.

The network then works as described above and the computation on a given input is *finished* when a *stable state* is achieved which is the situation in which the state of each neuron remains unchanged during one computational cycle. In that case we say that the computation was *convergent.*

The *result* of the convergent computation on a given input is given by the states of some selected *output neurons.* When the stable state is not reached the output of the computation is not defined.

Note that due to the fact that the computation on a given input can start with non-input neurons in arbitrary states and also due to the asynchronicity even on the same inputs, each computation can lead to different results or some can lead to no results at all. It will be our concern to design the network in such a way that the results will be unique if necessary.

The *time complexity* of a convergent computation on an input of length n will be given as the maximum number of computational cycles needed for achieving a stable state taken over all inputs of length n and over all possible initial configurations.

The *size* of the network will be given by the number of its neurons.

The networks for which $a_{ij} = a_{ji}$, $a_{ii} = 0$ holds will be called *symmetric networks*; otherwise we shall speak about *asymmetric* or *directed* networks. Note that in symmetric networks there is actually no difference between input and output connections of any neuron. An asymmetric acyclic neural network will be called a *neural circuit.* Networks with the empty set of uninitialized neurons will be called *initialized networks*; otherwise they will be called *uninitialized networks.* A special case of initialized networks in which all neurons except input ones are initialized to zero will be called *zero-initialized networks.*

3. Computational power of symmetric neural networks

From the viewpoint of computational complexity theory there is no substantial difference between asymmetric neural networks and unbounded fan-in linear threshold circuits (see e.g. [12]). The proof that the computational power of these machines is the same as that of Turing machines goes back to Minsky [11]. Further it is known that any boolean function can be realized by an unbounded fan-in linear threshold circuit of depth 3, but this means that the corresponding neural circuit computes this function in parallel constant time!

However the computational power of symmetric neural networks has not been known so far [5, 6] as it was conjectured that perhaps these networks need not be as powerful as symmetric ones since the former are but a special case of the latter.

We shall show that the computational power and efficiency of symmetric neural networks is the same as that of neural circuits. To prove this claim we shall need the following definition.

Definition 3.1. We shall say that a given neuron u (with symmetric weights) has the insensitivity range $\langle a, b)$, with $a \leq 0$, $b > 0$, if the addition of a further input with weight $w \in \langle a, b)$ will not affect the activity of u (i.e., its behavior will further depend only on the original inputs).

In the proof of the following lemma we shall see that the definition of the insensitivity range is correct, i.e. that the insensitivity range of any neuron always comprises an interval of form $\langle a, b)$, with $a \leq 0$ and $b > 0$. The lemma actually says more.

Lemma 3.2. *For any neuron u and any $\alpha \leq 0$ and $\beta > 0$ there is an equivalent neuron v that computes the same function as u does, and with insensitivity range $\langle \alpha, \beta)$.*

Proof (*Sketch*). Let $w_1, w_2, \ldots, w_k$ be the input weights of u and t its threshold. Define

$$a = \max\left\{ \sum_{i=1}^{k} w_i x_i \,\middle|\, \sum_{i=1}^{k} w_i x_i \leq t,\ x_i \in \{0, 1\} \right\},$$

$$b = \min\left\{ \sum_{i=1}^{k} w_i x_i \,\middle|\, \sum_{i=1}^{k} w_i x_i > t,\ x_i \in \{0, 1\} \right\}.$$

Clearly $a \leq t < b$ and $\langle a - t, b - t)$ is the insensitivity range of u, for any $t \in \langle a, b)$. Select now such a $t_0 \in \langle a, b)$ that splits the interval $\langle a, b)$ in the same ratio in which 0 splits the interval $\langle \alpha, \beta)$; i.e. $t_0 = (\alpha a - \beta b)/(\alpha - \beta)$. To obtain the weights and the thresholds of v multiply all weights of u and t_0 by $(\beta - \alpha)/(b - a)$. □

Now we are ready to formulate the main result of this section.

Theorem 3.3. *Any neural circuit C of size $S(n)$ and depth $D(n)$ can be simulated by a symmetric neural network N of size $S(n)$ in time $O(D(n))$.*

Proof (*Sketch*). The main idea in the construction of N is to adjust the weights and the thresholds of each neuron in C with the help of Lemma 3.2 so that the total minimal and maximal sum of its output weights would lie in the insensitivity range of each neuron. This will then enable us to introduce to each output connection the symmetric connection with the same weight; i.e., the transformation of C to N.

To do so, start with the set of neurons of C that have no successors in C and leave their weights and thresholds as they are and consider these neurons as being already adjusted. Now proceed recursively as follows: for each neuron v whose weights have already been adjusted compute the minimal sum α and the maximal sum β of its output weights. Then adjust the input weights and the threshold of v with the help of Lemma 3.2 so that the insensitivity range of v would be $\langle \alpha, \beta)$. The process will stop at input neurons that have no predecessors.

As a result we obtain a circuit C' equivalent to C. To obtain N introduce the backward connections to existing ones in C' with the same weights and note that these connections can by no means affect the behavior of the corresponding target neurons since their contribution lies always in the insensitivity range of target neurons.

Thus the neurons that are farther from the input neurons cannot affect those that are closer to them; hence in a sense the computation is directed from input neurons towards the output ones. Therefore the computation time will be $O(D(n))$. □

Corollary 3.4. *Any boolean function f can be realized by a symmetric neural network in constant time.*

Proof (*Sketch*). Apply the transformation from the previous theorem to a neural circuit that straightforwardly computes f represented by its conjunctive normal form. □

4. Non-deterministic computations and energy function minimization

Hopfield [8] has shown that the computation of any symmetric neural network can be thought of as a process of a minimization of a certain energy function which takes the form

$$E = -\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j + \sum_{i=1}^{n} t_ix_i$$

with $a_{ij} = a_{ji}$, $a_{ii} = 0$, and the meaning of individual symbols as described in Section 2. Hopfield proved in fact the following theorem that makes symmetric neural networks so attractive and which is mentioned here for consistency and completeness.

Theorem 4.1. *Starting from any initial configuration and providing that no two neurons will be in action at the same time any symmetric neural network with energy function E will achieve a stable state after at most* $O(p)$ *computational cycles, where* $p = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}| + \sum_{i=1}^{n}|t_i|$; *moreover this stable state represents a local minimum of E.*

Proof (*Sketch*). The change ΔE in E due to changing the state of the ith neuron by Δx_i is $\Delta E = -\sum_{j=1}^{n}[a_{ij}x_j - t_j]\Delta x_i$. According to the mechanism of neural network computation the change of x_i is positive if and only if the expression in the bracket is positive and similarly for the negative case. Thus any action of any neuron cannot cause the increase of the value of E and whenever some neuron changes its state the value of E will decrease. Since $|E|$ is bounded by p after at most p computational cycles the network must reach a stable state which is a local minimum of E. □

From the proof of Theorem 3.3, it follows that the computation of the corresponding symmetric neural network will always end in a unique final configuration that depends only on the initial states of input neurons. Hence for a given input the corresponding energy function will have exactly one (local or global) minimum irrespective of initial states of non-input neurons. In general, however, this need not be the case as seen also from the proof of the following theorem which shows that the minimization problem of energy function is a difficult one.

To formulate the theorem we shall make use of the following notion: the set of all initial configurations that differ only in the states of uninitialized neurons will be called the set of *compatible initial configurations.*

Theorem 4.2. *Let N be an uninitialized symmetric neural network with weights of at most polynomial size in the size of N. Then for any integer k the problem of deciding whether there exists a set of compatible initial configurations of N for which a stable state with energy not greater than k will be achieved is a Σ_2-complete problem.*

Proof (*Sketch*). First we shall show that the above problem is in Σ_2, i.e., in the class of polynomially time-bounded alternating Turing machine computations that use at most two alternations on each computational path, starting in an existential state (see [3] and [7] for the definition of the complexity class Σ_2).

Consider therefore an alternating Turing machine M that simulates N. M first guesses the input of N and then in parallel it creates the set of compatible input configurations compatible with that input. This takes time polynomial in the size of N since the size of each configuration is linear.

Then in parallel for each configuration, M simulates sequentially the computation of N. According to Theorem 4.1, this simulation will end in polynomial time due to our assumption concerning the size of weights of N.

The computation of M ends successfully if and only if for each configuration a stable state with energy $\leqslant k$ is achieved.

Thus the total running time of M's simulation is polynomial and since only two alternations have been used on any computational path of M, our problem belongs to Σ_2.

Next we shall construct a special uninitialized symmetric network N with energy function E that tests the validity of a given quantified boolean formula f in a conjunctive normal form with n variables, starting with existential quantifiers followed by universal ones. It is known that the validity problem of such formulae presents a Σ_2-complete problem [7]. Then we will show that there is a constant k such that f is valid if and only if there is a set of compatible initial configurations for which local minima of E with values $\leqslant k$ are always achieved.

The scheme of N is depicted in Fig. 1. In this figure the thresholds of only some neurons that will be important in the following explanation are given in circles representing the corresponding neurons; similarly important edges are labeled by their weights.

The states of neurons $i_1, i_2, \ldots, i_n$ play the role of boolean variables in f; neurons $n_1, n_2, \ldots, n_q$ are negating neurons that compute literals (they are present only

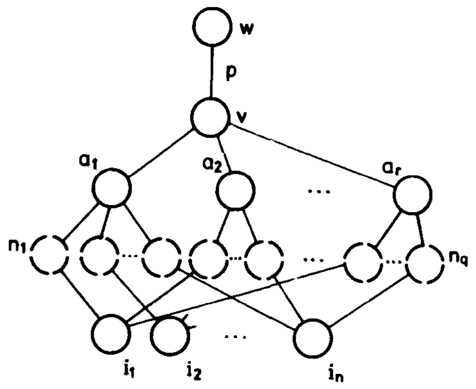

Fig. 1. A schema of an uninitialized symmetric network for validity testing.

when the respective variable has to be negated in the corresponding class of f). Neurons $a_1, a_2, \ldots, a_r$ compute multiple ORs; i.e. individual clauses of f and the neuron v computes the multiple AND of all clauses, i.e. the value of f on the input represented by states of $i_1, i_2, \ldots, i_n$.

The purpose of w is to decrease the value of E as low as we wish in the case that v is active; this is achieved by choosing p large enough. Note that when neurons v and w are both active they contribute with a value of $\Theta(p)$ to the energy function.

In the initial configuration of N the neurons i_j corresponding to those variables in f that are quantified by existential quantifiers represent the input neurons and those quantified by universal quantifiers represent the uninitialized neurons. The states of all other neurons are initialized to 0.

Under this arrangement it follows that for a set of compatible initial configurations v could be active in some stable state if and only if f is a valid formula.

Consider now the corresponding energy function E. It is clear by now that by a suitable choice of p we can achieve that the value of E is $\leqslant k$ for any computation that starts in the set of compatible initial configurations that satisfy f.

Finally note that the value of p need not be greater than the one used in Theorem 4.1, and that all weights in N and the size of N, is polynomial in the length of f. Therefore the reduction from f to N (and hence to E) takes polynomial time. □

Corollary 4.3. *Let N be a zero-initialized symmetric neural network with weights of at most polynomial size in the size of N. Then for any integer k the problem of deciding whether there exists an initial configuration of N for which a stable state with the energy $\leqslant k$ can be achieved is* NP-*complete.*

Proof (*Sketch*). The proof is analogous to that of Theorem 4.1, the main difference being that the satisfiability problem instead of that of validity of a simple non-quantified boolean formula in a conjunctive normal form, is considered. □

Corollary 4.4. *Let M be an arbitrary single-tape non-deterministic Turing machine of time complexity $T(n) \geqslant n \log n$. Then there is a zero-initialized symmetric neural network N of size $O(T(n))$ with energy function E and a constant k such that M accepts its input if and only if there is such an initial configuration of N for which a stable state with energy $\leqslant k$ is achieved.*

Proof (*Sketch*). There is a reduction [13] from a single-tape non-deterministic Turing machine with time complexity $T(n) \geqslant n \log n$ to a boolean formula f in conjunctive normal form of length $T(n)$ which is satisfiable if and only if the machine accepts its input. For this formula use the construction from the previous theorem. □

Observe the "cautious" formulation of the last corollary: it is not claimed here that N will always find a solution of the original problem. There is no doubt that the network will converge to some stable state but not necessarily to that in which the value of E is $\leqslant k$. The network will converge to that local minimum that is a so-called attractor of the initial configuration of the network. Hence the convergence can be "directed" by a suitable choice of initial states of certain neurons; but Corollary 4.3 and its proof show that exactly this presents an NP-complete problem by itself! This seems to be the bottleneck also of analog Hopfield networks when used for solving NP-complete problems (see e.g. [10]) where in order to obtain a good approximate solution it is necessary to set the initial values of analog variables so that they lie in the region of attraction of sufficiently low local minimum of the corresponding energy function.

5. Conclusions

In the paper we have presented a "missing link" between the standard complexity theory and the class of parallel computations that can be described as a minimization process of a certain energy function. On one hand the results show that as far as their computational power and efficiency is concerned, the neural models of computations are comparable with the existing models of parallel computations. On the other hand they enrich the classic repertoire of computational devices by a new class in which the physical aspects of computations as represented by the energy function are closely related to the computational algorithm itself. Last but not least these models provide new natural conceptual tools for solving some problems—especially those related to brain activities—that are hardly manageable by other techniques.

References

[1] D.N. Ackley, G.E. Hinton and T.I. Sejnowski, A learning algorithm for Boltzmann machines, *Cognitive Sci.* **9** (1985) 147–169.

[2] F. Barahona, On the computational complexity of Ising spin glass models, *J. Phys. A.* **15** (1982) 3241-3253.

[3] A.K. Chandra, D.C. Kozen and L.I. Stockmeyer, Alternation, *J. ACM* **28** (1981) 114-133.

[4] A.K. Chandra, L.I. Stockmeyer and U. Vishkin, Constant depth reducibility, *SIAM J. Comput.* **15** (3) (1984) 423-432.

[5] O. Egecioglu, T.R. Smith and I. Moody, Computable functions and complexity in neural networks, Tech. Rep. ITP-124, University of California, Santa Barbara, 1986.

[6] J.A. Feldman, Energy and the behavior of connectionist models, Tech. Rep. TR-155, University of Rochester, 1985.

[7] M.R. Garey and D.S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

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

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

[10] J.J. Hopfield and D.W. Tank, "Neural" computations of decisions in optimization problems, *Biol. Cybernet.* **52** (1985) 141-152.

[11] M. Minsky, *Computation. Finite and Infinite Machines* (Prentice Hall, Englewood Cliffs, NJ, 1967).

[12] I. Parberry and G. Schnitger, Parallel computation with threshold functions, *J. Comput. System Sci.* **36** (1988) 278-302.

[13] J.M. Robson, Linear size formulas for non-deterministic single tape computations, in: *Proc. 11th Australian Comp. Sci. Conf.*, Brisbane (1988).

[14] P. van Emde Boas, Machine models and simulations, ITLI Prepublication Series of Computation and Complexity Theory CT-88-95, University of Amsterdam, 1988.