Nuclear Instruments and Methods in Physics Research A279 (1989) 537–545
North-Holland, Amsterdam

TRACK FINDING WITH NEURAL NETWORKS

Carsten PETERSON

Department of Theoretical Physics, University of Lund, Sölvegatan 14A, S-22362 Lund, Sweden

Received 6 December 1988 and in revised form 6 March 1989

A neural network algorithm for finding tracks in high energy physics experiments is presented. The performance of the algorithm is explored on modest size samples with encouraging results. It is inherently parallel and thus suitable for execution on a conventional SIMD architecture. More important, it naturally lends itself to direct implementations in custom made hardware, which would permit real time operations and hence facilitate fast triggers. Both VLSI and optical technology implementations are briefly discussed.

1. Introduction

Experimental particle physics contain many pattern recognition problems ranging from track finding to feature extraction. Track finding is a combinatorial optimization problem; given a set of signals in space, reconstruct particle trajectories subject to smoothness constraints. When exploring all possibilities the computing requirements grow like N!. The problem is NP-complete *. Various heuristics algorithms are therefore used (see ref. [2] and references therein).

With the next generation of accelerators, e.g. LEP and SSC, one has to deal with very high multiplicity events appearing at an extremely rapid rate. For example, at SSC one expects ≈ 4 events per 100–250 ns [3] with charged multiplicities in the 100–500 range. The majority of these events are of "minimum bias" type and are likely to be ignored after a low level trigger (absence of jets, fast leptons etc.). However, a substantial fraction of the events ($O(1/10)$) contain high p_T jets and could be potentially interesting enough to record and analyze. Even after this data reduction one is faced with an enormous data processing task. With $O(100)$ signals per track one has to store $O(10^5)$ real numbers per μs. It is clear that if the tracks could be reconstructed at this level a lot is gained. First, the amount of data is compressed by a factor ≈ 10. Second, and maybe more important, this information (particle identification, etc.) could be used for higher level triggers. As an example, it would be possible to trigger on events with large neutral energy. To make this happen one needs a reliable track finding algorithm that naturally lends itself to parallel processing and custom hardware.

There has been an upsurge of interest in neutral network computational models in the last two years. These models have convincingly demonstrated a remarkable performance in their ability to learn pattern classifications (see e.g. ref. [4]). However, there exists another promising but less publicized application area of this technology: NP-complete problems [5,6]. Given the great VLSI potential of neural network models we find it worthwhile to explore the possibility of using neural network algorithms for the track finding problem. As mentioned above, the ultimate goal would be a custom hardware implementation. One should mention, though, that even a lower ambition level could be very rewarding; the intrinsic parallellism in neural network models can be fully exploited in commercially available SIMD * architectures like CRAY and the Connection machine. Hence off-line analysis could be speeded up substantially.

This paper is organized as follows: In section 2 we briefly review the basic ingredients and concepts in neural networks for the benefit of the newcomer to this field. Using neural networks for solving constraint satisfaction problems is discussed in section 3 and illustrated by a transparant case study; the graph bisectioning problem [6]. In section 4 we map the track finding problem onto a neural network and section 5 contains numerical explorations of this approach. Possible hardware implementations, VLSI and optical, are briefly discussed in section 6. Finally a brief outlook and summary can be found in section 7.

* An optimization problem is NP-complete if there is no known algorithm whose worst-case complexity (= time consumption) is bounded by a polynomial in the size of the input. For a review of these problems see ref. [1].

* SIMD = Single Instruction Multiple Data.

2. Neural networks basics

The basic ingredients in a neural network are N binary neurons $S_i = \pm 1$ and the synaptic strengths T_{ij} (i, $j = 1, \ldots, N$) that connect the neurons. These connection strengths can take both positive and negative real values and are in most models assumed to be symmetric, $T_{ij} = T_{ji}$. The dynamics of almost all neural network models is given by a local updating rule

$$S_i = \text{sign}\Big(\sum_j T_{ij} S_j\Big). \tag{1}$$

With eq. (1) each neuron evaluates the sign of the "upstream activity" (see fig. 1) and updates accordingly. Positive T_{ij} elements promote signals whereas negative ones inhibit them. At a given time the state of the system is given by the vector $\boldsymbol{S} = (S_1, S_2, \ldots, S_N)$. The dynamics of the system is determined by the **T**-matrix. It is easy to show that the updating rule of eq. (1) corresponds to gradient descent of the energy function [7]

$$E(\boldsymbol{S}) = -\tfrac{1}{2}\sum_{ij} T_{ij} S_i S_j. \tag{2}$$

In other words, for a given initial condition, eq. (1) takes us to the closest local minimum of the energy function given by eq. (2). There is a close analogy between this system and certain statistical mechanics systems; eq. (2) is the Ising spin glass Hamiltonian and the local updating rule of eq. (1) corresponds to Glauber dynamics [8] at zero temperature.

In some applications it is desirable to reach the global minimum rather than the closest local one. One then has to employ a stochastic "hill-climbing" procedure. One way to do this is by exposing the system defined by eq. (2) to a noisy environment, in which the state vector $\boldsymbol{S}$ obeys the Boltzmann distribution

$$P(\boldsymbol{S}) = \frac{1}{Z} e^{-E(\boldsymbol{S})/kT}, \tag{3}$$

where the partition function Z is given by

$$Z = \sum_{\boldsymbol{S}} e^{-E(\boldsymbol{S})/kT}. \tag{4}$$

In eqs. (3) and (4) temperature T represents the noise and k is the Boltzmann constant (in what follows we put $k = 1$). Ensemble members of this distribution can be generated through standard Monte Carlo procedures (Metropolis, Heat Bath, etc.) at a substantial CPU expense. However, it turns out that for spin systems with many degrees of freedom and large connectivity (T_{ij} connects not only locally) such time consuming simulations can be replaced by a set of deterministic equations by using the so called mean field theory (MFT) approximation [5,6,9]. The key ingredients of this trick are the following. Rewrite the discrete sum of eq. (4) as a multidimensional integral over the continuous variables $U_i = (U_1, \ldots, U_N)$ and $V_i = (V_1, \ldots, V_N)$:

$$Z = C \prod_{j=1}^{N} \int dV_j \int dU_j \, e^{-E'(V,U,T)}, \tag{5}$$

where C is a constant, $\Pi\int\int$ denotes multiple integrals, and

$$E'(V, U, T) = E(V)/T + \sum_{i=1}^{N} \left[U_i V_i + \log(\cosh U_i)\right]. \tag{6}$$

The MFT approach consists of approximating Z with the value of the integrand at the saddlepoint given by ($\partial E'/\partial U_i = 0$ and $\partial E'/\partial V_i = 0$), which yields

$$V_i - \tanh U_i = 0 \tag{7}$$

and

$$\frac{1}{T}\frac{\partial E(V)}{\partial V_i} + U_i = 0. \tag{8}$$

Combining eqs. (2), (7) and (8), we get

$$V_i = \tanh\Big(\sum_j T_{ij} V_j / T\Big), \tag{9}$$

where the new variables V_i denote the thermal average of S_i:

$$V_i = \langle S_i \rangle_T. \tag{10}$$

Thus the behaviour of the neural network at some temperature T can be emulated by the sigmoid updating rule of eq. (9) (see fig. 2). The step function (1) is the $T \to 0$ limit of eq. (9). Eqs. (9) are solved by iterative techniques. Improved convergence is often obtained by approaching the steady state with

$$\frac{dU_i}{dt} = -U_i + \sum_j T_{ij} V_j, \tag{11}$$

where

$$V_i = \tanh(U_i/T). \tag{12}$$

In eqs. (11) and (12) we have rescaled U_i of eqs. (7) and (8) with $U_i \to U_i/T$ in order to facilitate comparisons with VLSI implementations in section 6.

What are these neural network systems good for? There are two distinct application areas: feature recognition and optimization problems.

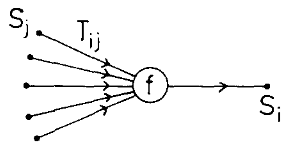

Fig. 1. A generic neural network updating.

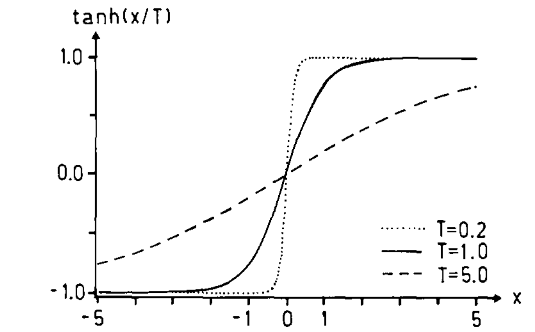

Fig. 2. Sigmoid gain functions (eq. (9)) for different temperatures T. The step function updating rule of eq. (1) corresponds to $T \to 0$.

- *Feature recognition*. Here one teaches the system a set of patterns by means of an appropriate choice of T_{ij}. In the production phase one then initiates the network with an incomplete pattern and the network completes it through eq. (1).
- *Optimization*. By appropriate "programming" of T_{ij} the network relaxes to a stable state that represents the solution of the problem.

Needless to say eqs. (1) and (9) are inherently parallel, which facilitates simulations of neural networks on parallel processors. In section 6 we discuss how eqs. (1) and (9) can be directly implemented in VLSI and optical technologies.

3. Graph bisectioning revisited

Let us here review the results from ref. [6] on using a neural network approach to obtain approximate solutions to the graph bisection problem. This problem, which constitutes an illustrative application, is defined as follows (see fig. 3):

Consider a set of N nodes, which are either connected with each other or not. Partition them into two halves such that there are $N/2$ nodes in each half. This problem can easily be mapped onto the Hopfield discrete neural network of eq. (2), by the following representation: For each node, assign a neuron $S_i = 1$ and for each pair of vertices S_iS_j, $i \neq j$, we assign a value $T_{ij} = 1$ if they are connected, and $T_{ij} = 0$ if they are not connected. In terms of fig. 3, we let $S_i = \pm 1$ represent whether node i is in the left or in the right position.

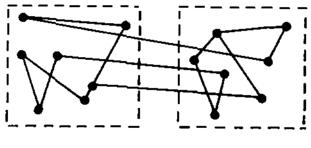

Fig. 3. A graph bisection problem.

With this notation, the product $T_{ij}S_iS_j$ is zero whenever nodes i and j are not connected at all, positive whenever connected nodes i and j are in the same partition, and negative when they are in separate partitions. With this representation, minimization of eq. (2) will favour the partitions within a partition while minimizing the connections between partitions. However, the net result will be that all nodes are forced into one partition. Hence we must add a "constraint term" to the right-hand side of eq. (2) that penalizes situations where the nodes are not equally partitioned. We note that $\Sigma S_i = 0$ when the partitions are balanced. Hence, a term proportional to $(\Sigma S_i)^2$ will increase the energy whenever the partition is unbalanced. Our neural network energy function for graph bisection then takes the form:

$$E = -\tfrac{1}{2}\sum_{ij} T_{ij}S_iS_j + \frac{\alpha}{2}\Big(\sum_i S_i\Big)^2, \tag{13}$$

where the imbalance parameter α sets the relative strength between the cut-size and the balancing term.

Again gradient descent on the energy surface defined by eq. (2) can be performed by making local updates with a step-function updating rule (see eq. (1)):

$$S_i = \operatorname{sign}\Big[\sum_j (T_{ij} - \alpha) S_j\Big]. \tag{14}$$

This procedure takes us to the closest local minimum rather than the global minimum that we desire. As discussed in section 2 this situation can be remedied by using MFT equations, which in this case have the form

$$V_i = \tanh\Big[\sum_j (T_{ij} - \alpha) V_j/T\Big]. \tag{15}$$

Before discussing the performance of eq. (15) we should make an important comment. The generic form of the energy function in eq. (13) is

$$E = \text{"cost"} + \text{"constraint"}. \tag{16}$$

This is very different from a more standard heuristics treatment of the optimization problem. For example, in the case of graph bisection one typically starts in a configuration where the nodes are equally partitioned and then proceeds by swapping pairs subject to some acceptance criteria. In other words, the constraint of equal partition is respected throughout the updating process. This is in sharp contrast to neural network techniques (eqs. (13), (16)), where the constraints are implemented in a "soft" manner by the Lagrange multiplier α.

In ref. [6] extensive studies of the performance of eq. (15) were made on graph bisection problems with sizes ranging from $N = 20$ to 2000 with very encouraging results. In fig. 4 we show typical results from ref. [6], where the neural network MFT algorithm is benchmarked against the more conventional simulated

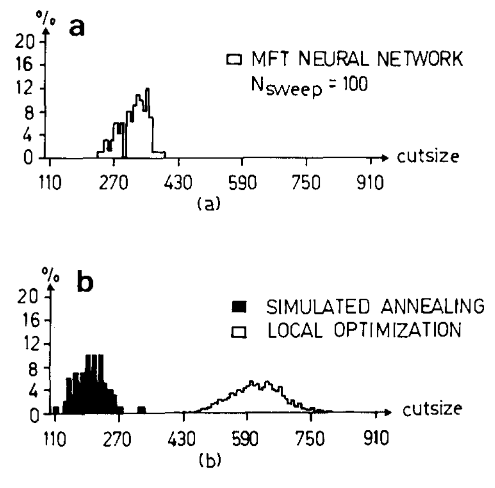

Fig. 4. Comparison of the *neural network* (a) performance of eq. (15) for an $N = 2000$ geometrical graph bisection problem with the more conventional approaches *local optimization*, and *simulated annealing* (b). The relative serial CPU consumption is 1:10:91. (From ref. [6].)

annealing and local optimization algorithms. Each histogram in fig. 4 represents 100 different experiments performed on a $N = 2000$ geometric graph. The results from ref. [6] can be summarized as follows:

- *Insensitivity to choice of α and T.* Two parameters enter into eq. (15). How sensitive is the result to our choice of these? Do these parameters have to be fine-tuned? It turns out that very good quality solutions are obtained for a substantial area of the (α, T)-plane.
- *Quality of the solutions.* The performance of the algorithm outperforms local optimization and falls on the tail of the very time consuming simulated annealing (see fig. 4).
- *Imbalance of final solution.* Since the constraint of bisectioning is soft, most MFT solutions have a small imbalance, i.e. $\Sigma S_i \neq 0$ *. This is easily remedied by a *Greedy Heuristics*, in which one searches in the larger set for a neuron that can swap sign with the least increase of the cut size.
- *Convergence times.* As is clear from the caption of fig. 4 the MFT algorithm is very competitive even when executed serially. The algorithm scales linearly with problem size.

4. Mapping the track finding problem onto a neural network

Consider a set of N space points or signals. For simplicity we limit the discussion to 2 dimensions. The track finding problem consists of drawing a number of continuous, nonbifurcating and smooth tracks through these points. In order to cast this problem onto a neural network algorithm we define $N(N-1)$ directed line segments S_{ij} between the N signals (see fig. 5). As will be discussed in the next section, the number of signals that needs to be connected is considerably less than $N(N-1)$ in most realistic examples.

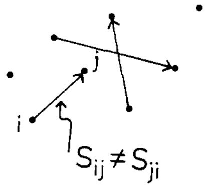

Fig. 5. N signals and directed line segments (neurons).

Let these segments S_{ij} be represented by binary neurons such that $S_{ij} = 1$ if the segment $i \to j$ is part of the track and $S_{ij} = 0$ if this is not the case. In order to define an appropriate energy function for the network we need two measurements: length of segments and angles between adjacent line segments (see fig. 6). Note that the definition of the angles θ_{ijl} takes the directions of S_{ij} and S_{jl} into account. Since our neurons in this case are 2-dimensional, the synaptic strengths have 4 dimensions. Again the energy function takes the generic form of eq. (16):

$$E = -\tfrac{1}{2}\sum_{ijkl}\left(T_{ijkl}^{(\mathrm{cost})} + T_{ijkl}^{(\mathrm{constraint})}\right)S_{ij}S_{kl}. \tag{17}$$

We want the "cost" part of T_{ijkl} to be such that short adjacent segments with small relative angles are favoured. There are many ways to accomplish this. We have made the choice

$$T_{ijkl}^{(\mathrm{cost})} = \delta_{jk}\frac{\cos^m\theta_{ijl}}{r_{ij} + r_{jl}}, \tag{18}$$

where m is an odd exponent. With this choice the energy of eq. (17) will blow up for long segments and large angles (in particular for $\theta > \pi/2$, when the contribution to E becomes positive). An example of $T_{ijkl}^{(\mathrm{cost})}$ is shown in fig. 7a. We next turn to the "constraint" term.

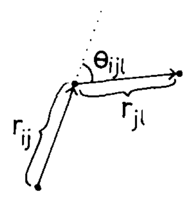

Fig. 6. Definition of segment lengths r_{ij} and angles θ_{ijl} between segments.

* In many engineering applications this is of no significance.

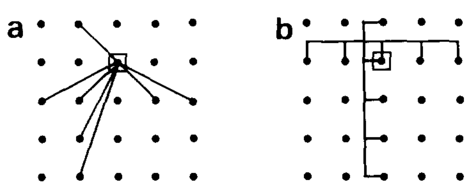

Fig. 7. Example of excitory (a) and inhibitory (b) connections to a neuron (S_{23}) corresponding to eqs. (18) and (20) respectively.

For this problem it consists of two parts:

$$T_{ijkl}^{(\text{constraint})} = T_{ijkl}^{(1)} + T_{ijkl}^{(2)}. \tag{19}$$

The first term in eq. (19) takes care of the requirement that there should be no bifurcating tracks; two segments with one identical index should not be "on" at the same time. In our language this requires that such configurations should give positive contributions to the energy function of eq. (17). This is done with the following choice for $T_{ijkl}^{(1)}$:

$$T_{ijkl}^{(1)} = -\tfrac{1}{2}\alpha\left[\delta_{ik}(1-\delta_{jl}) + \delta_{jl}(1-\delta_{ik})\right], \tag{20}$$

where α is a Lagrange multipier. In other words, connections between neurons in the same row or column are negative (see fig. 7b). The other term, $T_{ijkl}^{(2)}$, ensures that the number of neurons that are "on" is roughly equal to the number of signals N. It has the generic form

$$T_{ijkl}^{(2)} = -\tfrac{1}{2}\beta[\text{"global inhibition"}], \tag{21}$$

where β is another Lagrange multiplier. Its algebraic structure is more transparent in the expression for the total energy,

$$E = -\tfrac{1}{2}\sum \delta_{jk}\frac{\cos^m\theta_{ijl}}{r_{ij}+r_{jl}} S_{ij}S_{kl} + \tfrac{1}{2}\alpha \sum_{l\neq j} S_{ij}S_{il} + \tfrac{1}{2}\alpha \sum_{k\neq i} S_{ij}S_{kj} + \tfrac{1}{2}\beta\left(\sum S_{kl} - N\right)^2. \tag{22}$$

As in the graph bisection case we want to avoid local minima. Hence we employ the MFT equations analogous to eqs. (9) or (11) and (12).

$$V_{ij} = \tanh\left(-\frac{\partial E}{\partial V_{ij}}\frac{1}{T}\right), \tag{23}$$

which with eq. (22) yields

$$V_{ij} = \tanh\left[\left(\sum_l \frac{\cos^m\theta_{ijl}}{r_{ij}+r_{jl}} - \alpha\left(\sum_{l\neq j} V_{il} + \sum_{k\neq i} V_{kj}\right) - \beta\left(\sum_{k\neq l} S_{kl} - N\right)\right)\Big/T\right], \tag{24}$$

where again mean field variables $V_{ij} = \langle S_{ij}\rangle_T$ have been introduced. Eq. (24) contains three parameters, α, β and the temperature T. In the next section we explore eq. (24) numerically on modest size test problems.

5. Numerical explorations

We will now solve eq. (24) iteratively for test problems. In section 6 we will briefly outline how this can be done with dedicated VLSI or optical hardware.

With our representation, N signals require $N(N-1)$ neurons and equations. This number can be reduced in most applications. It is very unlikely that two signals, which are very far apart, are directly connected. Hence, one may define a radius R_{cut} around each signal beyond which no segments connect. There are many possible ways to define R_{cut}. We have chosen the following: A histogram of segment lengths (see fig. 8) typically peaks around small lengths. Assuming the peak to be approximately symmetric the cut is given by its high end slope. In our samples this procedure reduces the number of active neurons by a factor $\approx \frac{1}{3}-\frac{1}{2}$. Our results are not very sensitive to R_{cut}.

As testbeds we have chosen different images with 5 tracks based on approximately 10 signals per track (see fig. 9). In addition a few runs were made on smaller and larger problems (3 and 10 tracks) in order to get a feeling for the scaling properties. For the parameters in eq. (24) we used $T = 1$ and $\alpha = \beta = 1$. The exponent in the "cost" term in eq. (18) was chosen to be $m = 7$. The system was initialized at

$$V_{ij} = V_{ij}^0 + V_{ij}^{00}, \tag{25}$$

where V_{ij}^0 is given by the global constraint requirement (see the last term in eq. (22))

$$\sum_{ij} V_{ij}^0 = N, \tag{26}$$

and V_{ij}^{00} are random numbers in the interval

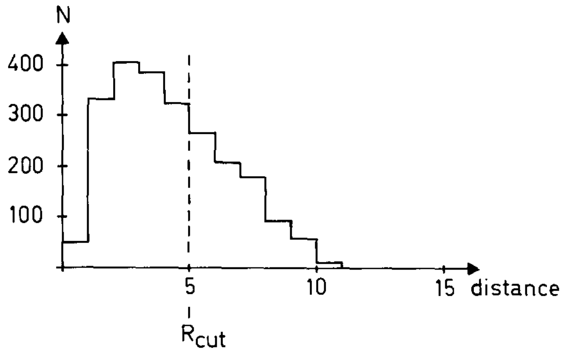

Fig. 8. Typical distribution of segment lengths for a 5-track problem with 50 signal points.

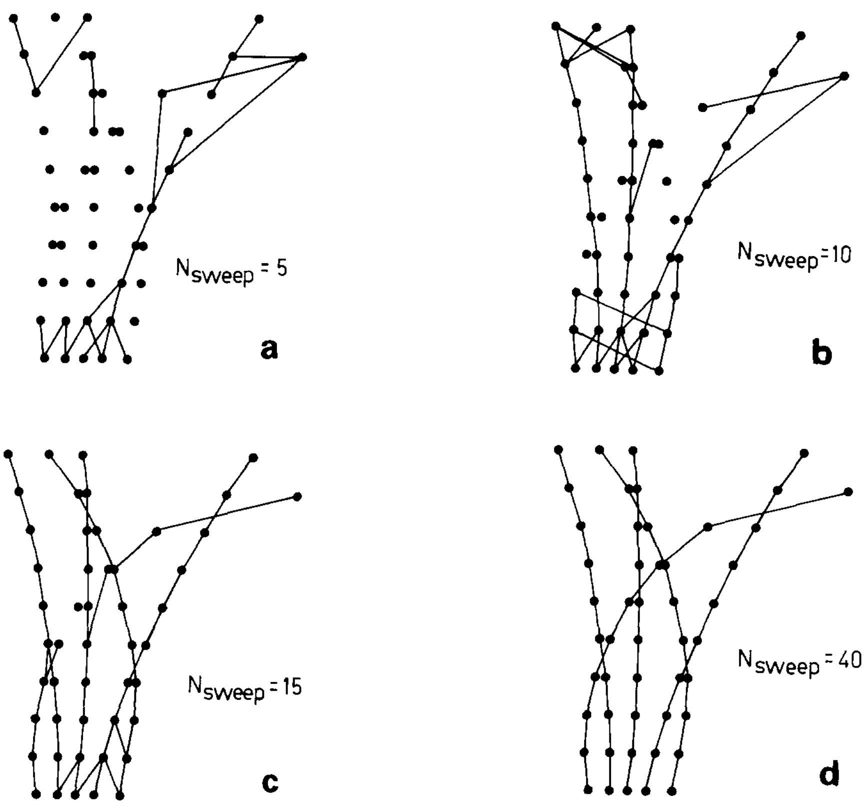

Fig. 9. Segments with $V_{ij} > 0.1$ at different evolution stages of the MFT equations.

$[-10^{-4}, 10^{-4}]$. As a convergence criterion we used

$$\sum_{ij}\left|\frac{V_{ij}(t+1)-V_{ij}(t)}{V_{ij}}\right|\frac{1}{\tilde{N}} \le 10^{-3}, \tag{27}$$

where $\tilde{N}$ is the number of "active" neurons subject to the cut $R < R_{\text{cut}}$.

For the iterative updating of eq. (24) there exist two extreme alternatives; asynchronous and synchronous updating. Asynchronous updating means that a neuron is randomly picked and updated and the new value is used in the r.h.s. of eq. (24) for the next updating. In our serial execution we have used a slightly weaker version, sequential updating, where the r.h.s. is evaluated with most recent values. This is in contrast to the synchronous method, where all neurons are swept over once and then updated. It is well known that nonlinear systems like eq. (24) in general perform better (= shorter convergence times and no cycle behaviour) when updated asynchronously. This is relevant when using SIMD architectures like CRAY and the Connection Machine, which only allow for synchronous updating. Degradation factors of order 2 have been observed already for relatively small systems [10] when using synchronous updating.

After convergence we set $S_{ij} = \text{sign}(V_{ij})$. In some cases the final solutions are plagued with violations of the constraints of eq. (20). These are easily removed by a greedy heuristic procedure [6] as follows: After convergence of the MFT equations, find all signals with more than one line segment entering or leaving. Remove the "extra" segments ($S_{ij} = 1$) subject to the condition of minimal increase of the "cost" part of the energy function. (In most cases it turns out that the "cost" actually decreases with this procedure.)

In fig. 9 we show segments at different stages of the evolution ($N_{\text{sweep}} = 5$, 10, 15 and 40 respectively) for a 5-track problem. The development of the individual neurons is shown in fig. 10 and the corresponding energy behaviour is shown in fig. 11.

We have also performed exploratory studies on the scaling properties of the algorithm for this problem. In fig. 12 we show results from limited statistics runs on three different problem sizes. The problem size behaviour is consistent with a linear convergence time dependence on $\tilde{N}$ as in ref. [6]. The vertical error bars in fig.12 reflect the fact that different problems of given size have different $\tilde{N}$.

All in all, our simulation results look very encouraging. As in the graph bisection case we observe good

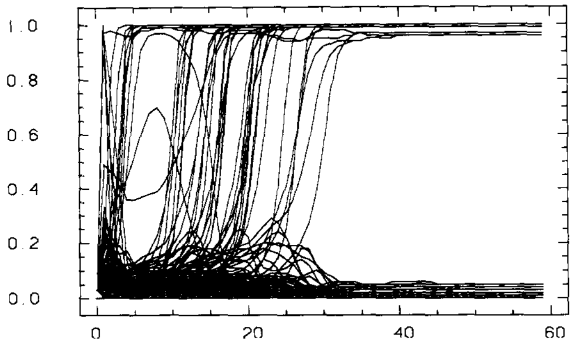

Fig. 10. The neuron variables V_{ij} as a function of N_{sweep} for the 5-track problem of fig. 9.

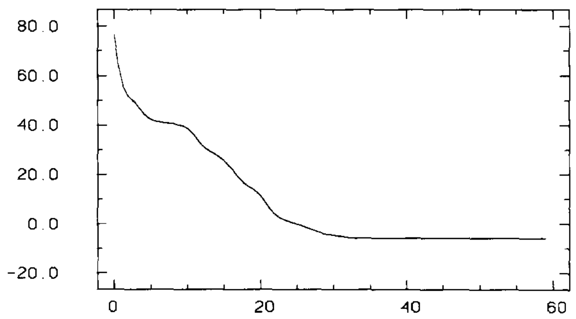

Fig. 11. Energy as a function of N_{sweep} for the 5-track problem of fig. 9.

solution quality, parameter stability and again possible imbalance of the solutions are easily removed with a *greedy heuristic*.

In contrast to the graph bisection results, our serial computations are not competitive with conventional methods for the track finding problem. The reason is that N scales approximately with $N(N-1)$, which should be compared with N for more conventional approaches [2] *. The real power in the neural network approach lies in the possibilities of implementing it in parallel hardware.

6. Hardware implementations – VLSI and optical

A neural network can either be fully implemented in hardware or virtually implemented in software. A full implementation requires a dedicated processor for each neuron, and a dedicated information channel for each interconnect. Virtual implementations, on the other hand, simulate the functions of the neurons using a fewer number of processors, by time multiplexing several neurons on each processor $N > M$. The states of the neurons, connection weights and other parameters are stored in local memories. Different mesh and hypercube architectures (e.g. the Connection Machine) are suitable for such a virtual implementation. Depending on the magnitude of M, substantial speedup of solving our MFT equations can be achieved.

However, for real-time processing full hardware implementation is desirable. Two different scenarios exist for this: VLSI and optics. Let us review these possibilities very briefly. Nothing in this section is original.

6.1. VLSI

This is the most mature discipline of the two. In this technology it is natural to represent the neurons as

* The same increase in the number of degrees of freedom occurs when generalizing graph bisection to graph partition [11].

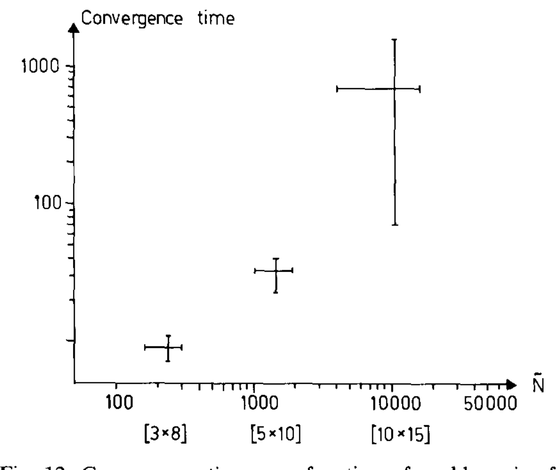

Fig. 12. Convergence time as a function of problem size for three different $N_{\text{tracks}} \times N_{\text{signals/track}}$.

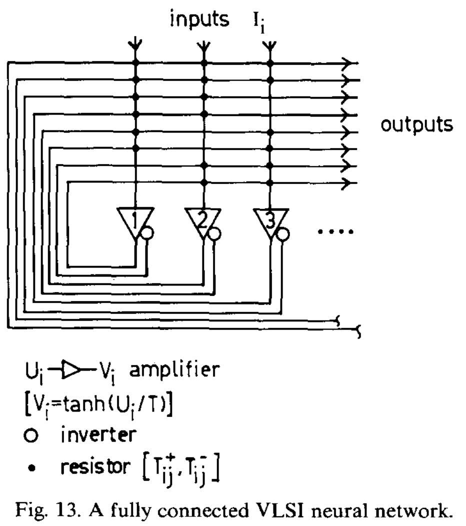

Fig. 13. A fully connected VLSI neural network.

amplifiers with a feedback circuit consisting of wires, resistors and capacitors [12,13]. The amplifiers have sigmoid input–output relations (cf. eq. (12) and figs. 2 and 13). Each amplifier also has an input capacitor (C_i) that defines the time constant and facilitates the integrative analog summation of the input signals from the other neurons. The connection strengths are represented by resistors with values $R_{ij} = 1/|T_{ij}|$. The resistors are of course always positive. On the other hand, most neural network modelling (at least the one we are dealing with here) requires that the T_{ij} elements can take both positive and negative values. This can be solved by having a pair of "rails" of connections between the neurons, T_{ij}^+ and T_{ij}^-. Each amplifier is given two outputs, a normal (> 0) and an inverted one (< 0). T_{ij}^+ and T_{ij}^- are both positive but are interpreted with different signals depending on the sign of the outgoing voltage from the neuron. In fig. 13 we show such a fully connected VLSI neural network, where we also have included externally provided currents I_i that set the overall and individual threshold levels for the amplifiers.

The time evolution of the circuit in fig. 13 is given by

$$C_i \frac{dU_i}{dt} = \sum_j T_{ij}^+ V_j + \sum_j T_{ij}^-(-V_j) + I_i, \quad (28)$$

with

$$V_i = \tanh(U_i/T). \quad (29)$$

These RC-equations are identical to eqs. (11) and (12) with $C_i = 1$ and $I_i = 0$. In other words, the RC-equations of a fully connected network of neurons have the MFT solutions as fixed points! This isomorfism constitutes a strong merger of theory and practical implementations. The circuit in fig. 13 could be fabricated with a chip consisting of CCD technology based resistors surrounded by off-chip amplifiers [14,15]. With 0.5 μm design rules a 10000 neuron network (100 signals in the track-finding case) would then occupy an area of 25 mm^2. The relaxation time for such a circuit is given by $\tau = 1/RC$. Realistic values for C and R are 0.1 fF and 20 MΩ respectively, which gives a very rapid convergence time, $\tau = O$(ns).

6.2. Optical computing

This novel technology is rapidly gaining momentum. It is well suited for parallel processing and many demanding applications involve image processing where optics is natural. Our MFT equations involve matrix multiplications [$U_i = \Sigma T_{ij} V_j$], which are natural to implement optically. To see this, consider a set of light emitting diodes (LEDs) and a set of photodetectors (PDs) with a spatial light modulator (SLM) in between (see fig. 14). With appropriate lenses these devices can perform a matrix multiplication when the SLM is used

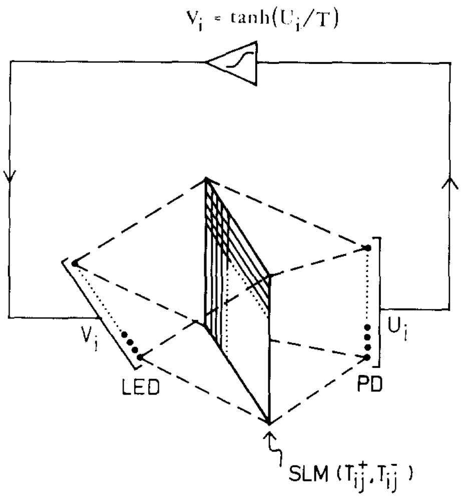

Fig. 14. A generic optical neural network.

to store the matrix elements (T_{ij}). Again the connection strengths need to be multiplexed (T_{ij}^+ and T_{ij}^-) in order to account for negative weights. In fig. 14 it is also indicated how to solve eq. (9) by closing a circuit with an amplifier (tanh). As a spatial light modulator one can either use a liquid crystal, which is loaded electronically, or a photorefractive crystal, which is set optically. With the latter technology one can store approximately 10^9 matrix elements per cm^3. This number is certainly smaller than the corresponding numbers for VLSI, but one should keep in mind that with this technology there are no scaling limitations. At present such a system is slower, O(ms), than the VLSI alternative but the field is young and the very large scale interconnect capacity is hard to beat!

7. Summary and outlook

We have mapped the track finding problem onto a neural network. Our encouraging results can be summarized as follows.

- With mean field theory techniques the neural network algorithm produces solutions with high quality for the relatively modest size problems we have explored; $N_{\text{tracks}} \times N_{\text{signals/track}} = 3 \times 8$, 5×10 and 10×15 respectively.
- The calculations were performed on a serial computer. In the neuronic representation of the problem, N signals require $O(N(N-1))$ neurons. The algorithm is therefore not competitive with conventional approaches when executed serially.
- The real power lies in the inherent parallelism. This can be exploited in two stages, through virtual implementation on general purpose parallel architectures and with custom made hardware.

- The virtual implementation will presumably turn out to be very competitive with conventional approaches and will be useful for off-line analysis.
- The MFT equations naturally lend themselves to direct implementations in VLSI and optics. In the case of VLSI a strong isomorfism exists. With these technologies it will be possible to perform on-line analysis on the μs scale with $O(10^4–10^5)$ degrees of freedom. This could make real time triggers at, e.g., the SSC feasible.
- One should keep in mind that our way of mapping the problem onto a neural network is by no means unique. Recently more efficient ways of mapping the TSP problem onto a neural network have been developed [17], where S_{ij} are replaced by vectors $\mathbf{S}_i$ subject to normalization constraints. Substantial performance improvements are obtained with this method. It will be very interesting to see if these improvements carry over to the track finding problem.

After the completion of this work we received a paper by Denby [18] with a similar approach to the track finding problem.

References

[1] M.R. Carey and D.S. Johnson, Computer and Intractability: A Guide to the Theory of NP-Completeness (W.H. Freeman, San Fransisco, 1979).
[2] H. Grote, Rep. Prog. Phys. 50 (1987) 473.
[3] See, e.g., M.G.D. Gilchries, in: Proc. 1984 Summer Study of the Design and Utilization of the Superconducting Supercollider, Snowmass, Colorado, ed. J.G. Morfin (1984).
[4] See, e.g., D. Rumelhart and J.L. McCelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1 (MIT, 1986).
[5] J.J. Hopfield and D.W. Tank, Biol. Cybernetics 52 (1985) 141.
[6] C. Peterson and J.R. Anderson, Complex Systems 2 (1988) 59.
[7] J.J. Hopfield, Proc. Nat. Acad. Sci. USA 79 (1982) 2554.
[8] R.J. Glauber, J. Math. Phys. 4 (1963) 294.
[9] C. Peterson and J.R. Anderson, Complex Systems 1 (1987) 995.
[10] C. Peterson and E. Hartman, MCC-ACA-ST/HI-065-88, to appear in Neural Networks.
[11] C. Peterson and J.R. Anderson, MCC-ACA-ST-064-88, to appear in Physica D.
[12] J.J. Hopfield, Proc. Nat. Acad. Sci. USA 81 (1984) 3088.
[13] C.A. Mead and M.A. Mahowald, Neural Networks 1 (1988) 91.
[14] L.D. Jackel, R.E. Howard, h.P. Graf, B. Straughn and J.S. Denker, J. Vac. Sci. Technol. B4 (1986) 61.
[15] J.P. Sage, K. Thompson and R.S. Withers, Neural Networks for Computing, Snowbird, UT 1986, ed. J.S. Denker.
[16] See, e.g., C. Peterson and S. Redfield, Proc. 3rd Int. Conf. on Optical Computing, Toulon, France (1988).
[17] C. Peterson and B. Söderberg, LU TP 89-1, to appear in Int. J. Neural Systems.
[18] B. Denby, Comp. Phys. Comm. 49 (1988) 429.