Computer Physics Communications 49 (1988) 429–448
North-Holland, Amsterdam

NEURAL NETWORKS AND CELLULAR AUTOMATA IN EXPERIMENTAL HIGH ENERGY PHYSICS

B. DENBY

Laboratoire de l'Accélérateur Linéaire, Orsay, France

Received 20 September 1987; in revised form 28 December 1987

Within the past few years, two novel computing techniques, cellular automata and neural networks, have shown considerable promise in the solution of problems of a very high degree of complexity, such as turbulent fluid flow, image processing, and pattern recognition. Many of the problems faced in experimental high energy physics are also of this nature. Track reconstruction in wire chambers and cluster finding in cellular calorimeters, for instance, involve pattern recognition and high combinatorial complexity since many combinations of hits or cells must be considered in order to arrive at the final tracks or clusters. Here we examine in what way connective network methods can be applied to some of the problems of experimental high energy physics. It is found that such problems as track and cluster finding adapt naturally to these approaches. When large scale hard-wired connective networks become available, it will be possible to realize solutions to such problems in a fraction of the time required by traditional methods. For certain types of problems, faster solutions are already possible using model networks implemented on vector or other massively parallel machines. It should also be possible, using existing technology, to build simplified networks that will allow detailed reconstructed event information to be used in fast trigger decisions.

1. Parallel processing

The basic tenet of parallel processing, that independent tasks can be performed at the same time, is of course not a new idea, but it is only recently, with the ever-decreasing cost of computer hardware, that one has dared to think of actually putting two or more computers to work on the same problem. Parallel processing has made an impact on high energy physics in a variety of applications.

Three parameters can be used to specify the architecture of any parallel processing system: 1) the number of processors, 2) the power of each processor, and 3) the degree of connectivity between processors. In what follows we shall see that difficult problems do not necessarily require powerful processors. Depending upon the problem to be solved, a large array of simple but tightly coupled processors may be more appropriate than a small array of powerful, loosely coupled processors. Processing power of the individual processors can be traded off against more processors, or higher connectivity, or both. The overall power of a particular architecture is specified not by any one of the parameters, but, in some sense, by the product of the three.

To date, in high energy physics, most applications have involved a modest number, a few tens, or at most hundreds, of powerful processors which are loosely coupled, i.e., have low connectivity. Examples of this approach are the 168/E and 3081/E emulator 'farms', and the Fermilab ACP system (see ref. [1] for a review of these and other related systems). In these applications, individual data events are distributed to the processors, each of which completely analyzes an entire event, passes on the results, indicates its readiness to accept another event, and so on. Because the processors independently operate on separate events, the amount of communication required between processors is minimal.

In another type of application, a group of processors work on different parts of the same event: one on the tracking, one on the calorimetry, and one on the Cherenkov analysis, for instance.

In this case the processors can be somewhat less sophisticated (smaller individual memory requirement, e.g.) but the degree of connectivity is necessarily larger since the results must be reassembled at the end of the calculations to build a complete event or to make a trigger decision (the NIKHEF FAMP system is an example of the latter [2]).

Recently, powerful, commercially produced vector computers have begun to become available. These machines are an example of large arrays of very primitive processors (each can only add, subtract, multiply, divide, and perhaps extract square roots) which are very highly coupled. Consider for example forming the dot product of two 100-component vectors. In a vector computer, each of the 100 multiplications of the individual components can be done simultaneously (assuming the maximum machine vector length is at least 100), but afterwards the results must all be brought together again to be summed into the dot product. By recognizing those areas of high energy physics data analysis programs which contain large numbers of manipulations of independent quantities, and by recoding these sections for application on a vector machine, significant gains in processing speed can be had. Examples are the Fermilab E711 and SLAC Mark III vectorized track finding programs in which the drift chamber hit pattern is compared to a bank of stored templates of all possible tracks, [3,4] and the CUSB calorimeter clustering algorithm, which uses stored neighbor information to group pulse heights together [5].

2. High connectivity parallel processing

Two of the most recent developments in parallel processing, and ones to which as yet little attention has been given by experimental high energy physicists, are the use of cellular automata and neural networks, which are in fact closely related to each other. Both consist of arrays of very simple individual processing cells, with multiple inputs and outputs, which are connected to each other according to a permanent, fixed pattern which depends on the problem to be solved. Thus, these arrays are a bit like a vector machine in which the pattern of connections between processors for a given problem has been uniquely fixed, being established either in the hardware, or through an imposed addressing scheme.

Associated with each cell is a variable describing its current output value; it is the set of all output values which ultimately will encode the ‘answer’ to the problem being solved. Each cell’s output value is uniquely determined from the ensemble of its inputs, i.e., the ensemble of information received from the cells to which it is connected, according to a simple rule which is normally the same for all cells. Each cell’s new output value is then transmitted to the input of each cell to which it is connected, and the process continues. The system is allowed to evolve in this way until a steady state is reached, i.e., until the pattern of output values stops changing. These steady states are called *attractors*.

The ‘Game of Life’ of Conway [6] is a popular computer game using cellular automata in which the cells are represented by squares in a grid on a display screen. The player assigns random output values to the cells, which are represented by the cell being dark (0), or bright (1), and defines a transition rule. Each cell, responding to its neighbors, changes its state according to the transition rule, making interesting patterns of bright and dark dots as the system evolves toward an attractor state (fig. 1).

High-connectivity methods are most effective on so-called ‘intractable’ problems in which, due to combinatorics, or inherently large numbers of variables, solutions by more traditional means would require collossal amounts of computer time. Cellular automata are currently being used to study, for example, detailed solutions to turbulence problems in fluid mechanics, and the growth of snowflakes [7], two areas long considered too complex to be solved in detail by computational methods. Recently, they have even been applied to simulations of the distribution of galaxies in the universe [8].

One of the more famous successes of neural networks is in their application to the Traveling Salesman Problem, or TSP [9]. In this problem a list of cities and their coordinates is presented, and the computer has to pick the shortest trip that visits each city exactly once. The number of possi-

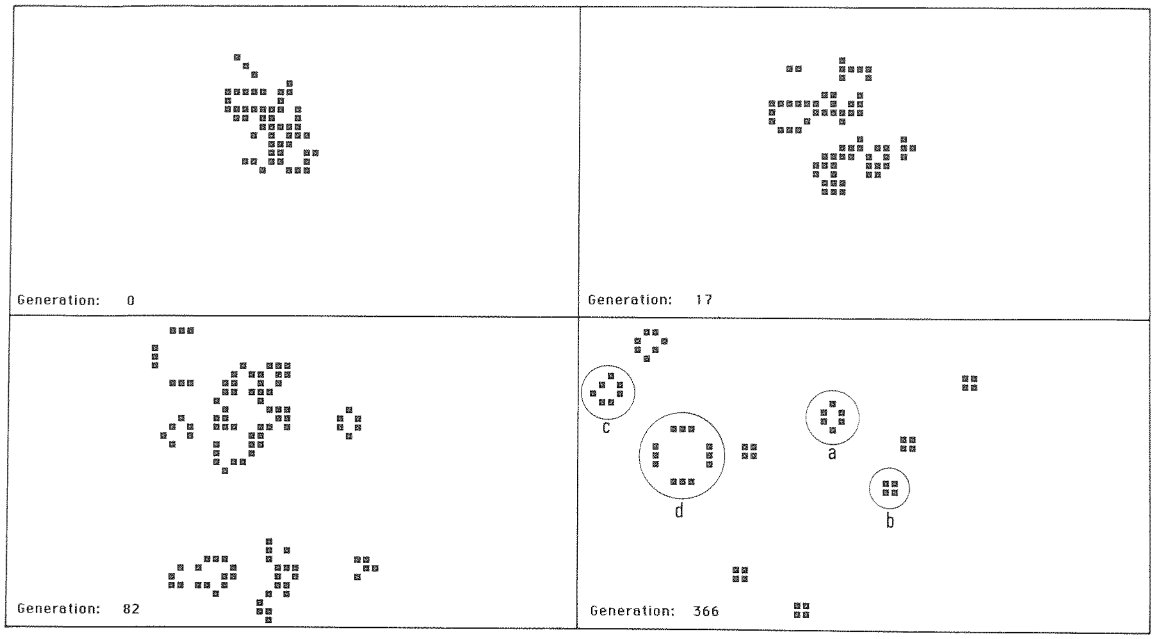

Fig. 1. Four stages in the evolution of a Game of Life. The initial state is assigned at random. The transition rule states that, in each succeeding step, a cell will take on the value of the majority of its four neighbors. If their is no majority, no change takes place. The state of the system is shown after 17, 82, and 366 generations, after which no further change occurs. Four types of stable or "attractor" configurations, a), b), c) and d) are seen in the final frame: these are invariant under the application of the transition rule (d) is actually a 2-state, i.e., invariant under two consecutive applications of the rule). These are simple examples of the spontaneous creation of large scale structure by the application of simple local rules.

bilities grows as $N!$ so that for more than about 12 cities the problem becomes intractable on a traditional, even vector, computer. A neural network of N^2 neurons, on the other hand, can provide near-optimal solutions in a few network time constants. Time constants of currently available networks are of the order of 1 microsecond [10]. (These prototype networks are still rather small, but the time constant should not be a function of network size.) Another current application of neural networks is the NETtalk project developed at Johns Hopkins University [11]. This is a simulated neural network with some 10 000 interconnections which can be 'taught' to translate printed text into spoken words by using an iterative 'learning' procedure.

Neural networks and cellular automata thus are at the opposite end of the parallel processing spectrum from, say, emulator farms: the former consist of large numbers of simple processors that are heavily interconnected, while the latter consist of moderate numbers of powerful processors that are loosely interconnected.

The following section defines more precisely the properties of the two types of networks; subsequently, most of the discussion will deal with neural networks, with a brief return to cellular automata in the section on calorimeter clustering. Associative memory, an important property of neural networks, is discussed in section 11.

3. Characteristics of cellular automata and neural networks

As mentioned earlier, cellular automata and neural networks are closely related, but four qualities distinguish a cellular automata system (see ref. [7] for a more detailed treatment). First, the output values of the cells are discrete (usually 0 and 1, although multistate automata are also being studied). Second, each automaton is connected only to a limited number of nearby cells, often just its nearest neighbors. Third, a cellular automata system is clocked; with each clock tick, each automaton examines its inputs, decides on the basis of a 'transition rule' whether or not to change its state, and sends its new output value to the inputs of its neighbors. On the next tick these new inputs are examined, and so on. Fourth, the transition rule used, which can have an arbitrary functional form, depends upon the problem to be solved.

A neural network differs on these four points. First, the output values of the neurons are continuous variables, though usually bounded between 0 and 1. Second, a given neuron can in principle be connected to any or all of the other neurons in the network, including those which are relatively distant. Third, neural networks are asynchronous, not clocked: the outputs vary constantly according to an instantaneously evaluated function of the inputs. In addition, the inputs are integrative over time, with an integration time constant to be specified. Finally, the function which determines the output from the inputs has a specific form, and is the same for any problem: the output is a sigmoid function (see fig. 2) of a linear combination of the inputs. Only the coefficients in this linear combination (and of course, the pattern of connectivity) must be specified for a particular problem.

The use of the sigmoid response function is a relatively new idea [9], motivated by studies of the response functions of real neurons in the cortex, which have a similar form. It was not until the addition of this feature that the calculational properties of neural networks were realized; that is, neurons with this type of response function seem to exhibit computational properties not shared by neurons with simple step-function response [9]. The exact shape of the sigmoid curve is not important, but what appear to be the essential elements are the existence of a central, nearly linear region, and plateaux as the input approaches large positive or negative values. The reasons for the importance of these features will be discussed further in section 5. It is because of the biological origins of the response function, and of course the high degree of interconnectivity, that the mathematical networks we study here have come to be called 'neural'.

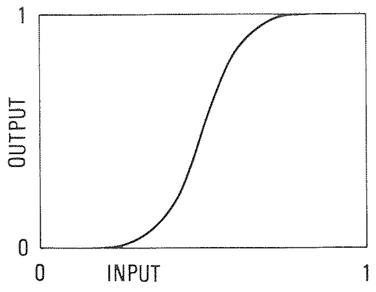

Fig. 2. Sigmoid response function.

4. Implementation of high-connectivity systems

True hard-wired networks in which the signal processing is done with analog circuitry are just beginning to appear. Examples are a 256-neuron, fully connected, silicon-based device developed at AT&T Bell Labs [10] and an experimental 10 000 cell optical device, developed at Caltech, that uses a hologram to map the output of each cell onto its neighbors [12]. It should be possible, however, with conventional optical and electron-beam lithography, to build networks with a connection density of 0.5×10^9 per cm^2 [10]. Such densities are possible because the *connections* between neurons, by far the element required in the greatest numbers, can be implemented by simple resistors, which take up considerably less area on a silicon chip than a transistor. The basic circuit which needs to be realized is shown schematically in fig. 3. This figure shows an array of neurons, represented by simple amplifiers, which have normal and inverted outputs in order to supply either reinforcing or inhibitory input to other neurons. All of the outputs cross all of the inputs in a grid, and resistances, with values proportional to the desired coefficients, are attached at the interstices in order to form the required connection network. The inputs to a given neuron are wire-summed directly on its input line. The network shown is a fully connected one, with the signs of the connections chosen at random.

As large scale hard-wired networks do not yet

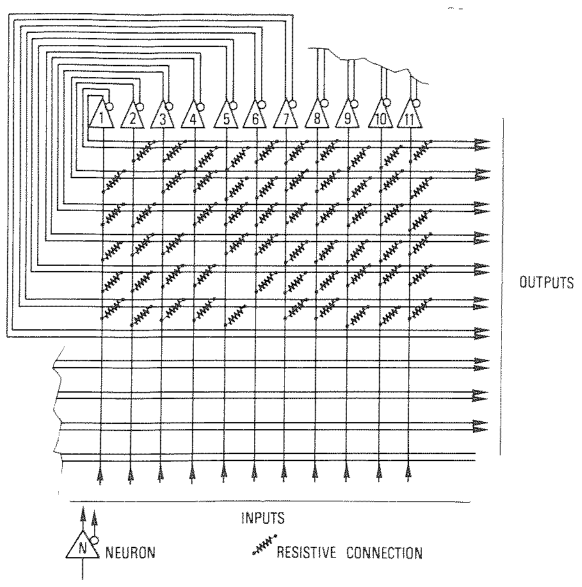

Fig. 3. Schematic of a fully connected network. Neurons, represented as amplifiers, have normal and inverted outputs, for making reinforcing and inhibitive connections, respectively. The network shown is fully connected, with signs of connections assigned at random. Note the absence of connections along the diagonal.

exist, it is necessary to use modelling to study their properties. Because of the structure of a network, these model calculations are almost totally vectorizable, and thus can fully exploit the speed of vector machines. Some of the fluid flow simulations have also been done using the Connection Machine^{TM} [13]. This is a commercially produced machine consisting of an array of 65 536 individual processors each of which can explicitly address any of the other processors in the array, through the intermediary of a Boolean 12-cube routing scheme. This routing scheme is necessarily slower than a true hard-wired connection network (which would require billions of wires), but is nonetheless very efficient since each word of information sent is never more than 12 steps away from its destination. Because of this architecture, the Connection Machine is ideal for modelling neural and cellular automata networks. It was designed to be able to simulate any size array of processors (even arrays bigger than 65 536 are handled transparently), with the number of processors and their interconnections specified by the user.

It turns out, perhaps somewhat surprisingly, that for extremely complicated problems, *even the model* of a neural network is fast, faster than a more conventional approach. That this is so will be demonstrated in the discussion, in section 6, of the following example.

5. Example. The Travelling Salesman Problem (TSP)

We follow here, loosely, the treatment of Hopfield and Tank [9]. Consider, for a trip of N cities, an N by N array of neurons. The horizontal position of a neuron in the array determines the city number, and the vertical position determines the order in which the city is visited (see fig. 4). Thus a valid trip is represented by a *different* neutron being on in each row. Now consider the connections between neurons. Let every neuron be connected to each neuron in the rows immediately below and above with a coefficient proportional to $A - d_{ij}$, where d_{ij} is the distance between cities i and j, and A is a positive constant. That is, if neuron i has an output value of f_i, it will present a signal proportional to $(A - d_{ij})f_i$ at the input of

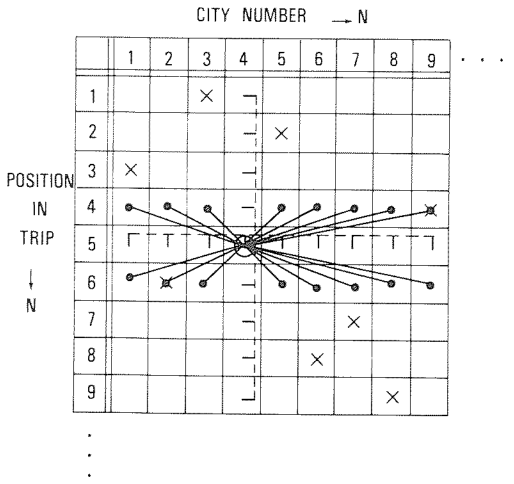

Fig. 4. Travelling salesman problem (TSP). Solid line = reinforcing coefficients $A - d_{ij}$, broken line = inhibitive coefficients $-B$. Connections for neuron (4, 5), represented by ○, are shown, other 'on' neurons are represented by ×.

neuron j. The reverse relation is also true since the matrix of coefficients is normally taken to be symmetric. Thus, if d_{ij} is large, neurons i and j will tend to inhibit each other, with the effect that patterns of neurons with small d_{ij} values between neighboring rows are favored. Inhibitive connections, of value $-B$, say are also imposed, between each neuron and those in its own row and column. This has the result that if one neuron is on in a particular row and column, it will tend to turn off other neurons in the same row and column, thus ensuring that *valid* tours exclusively will be considered.

Random starting values are assigned to the neuron outputs, and the system is allowed to evolve. Throughout the evolution of the network, each neuron will have an output value between 0 and 1, which is equivalent to saying that the state of the system is described by a point in an N-dimensional space, with one dimension for each neuron. In the final solution, each neuron must have a value of either 0 or 1, i.e., the solution lies at a corner of an N-dimensional hypercube. During the calculation, however, the state of the system moves around *within* the volume of the hypercube. The neurons at this stage are operating in the linear region of their response functions. The state of the system during the calculation, which does not yet represent a valid tour, can be thought of as superposition of several possible tours which are being considered *simultaneously*.

Neurons with step function response lack this linear region. Simulations have shown [9] that the tour minimization ability of such neurons is far inferior to that of continuous valued neurons, presumably because the ability to simultaneously consider a number of tentative solutions is not present in that case. The interior volume of the calculational hypercube is forbidden, and the system is restricted to jumping from one corner of the hypercube to another.

In the TSP, pairs of neurons representing close-together cities reinforce each other through positive feed-back, i.e., a positive change in the output of one neuron will elicit positive changes in the outputs of those neurons with which it has mutually reinforcing connections. These neurons will in turn re-stimulate the initial neuron, so that, after several cycles, the outputs will be pushed into the $f=1$ plateau region of the sigmoid response function. Pairs of neurons whose connections are predominantly inhibitive (d_{ij} large), will, similarly, force each other to zero after several cycles. It is in this way that the system evolves to an attractor state of neurons with saturated output values of 1 and 0.

Hopfield and Tank have shown that the neural system obeys a set of coupled non-linear differential equations, which for the case of a homogeneous network, and in the absence of input currents from outside the network itself, can be written as:

$$\tau \frac{\mathrm{d}v_i}{\mathrm{d}t} = \sum_{j=1}^{N} T_{ij} f(v_j) - v_i,$$

where f is a sigmoid output function v_i is the input value of neuron i, T_{ij} is the coupling strength or coefficient between neurons i and j, and τ is the integration time constant of the array. The equations tell us that the change in the input value of a particular neuron in a small interval $\mathrm{d}t$ is given by two terms, the first being the sum of the output values of its neighbors multiplied by their coefficients, and the second being simply a time decay term.

It can be shown [9,14] that for such an array, there exists an 'energy' function E given by

$$E = -\tfrac{1}{2} \sum_{i,j=1}^{N} T_{ij} f(v_i) f(v_j)$$

which, as long as $T_{ij} = T_{ji}$ and $T_{ii} = 0$, will be continuously minimized as the system approaches an attractor state. For the TSP, with $T_{ij} = A - d_{ij}$, it is clear that, for valid tours, E is minimized when

$$\sum_{i,j}^{\text{valid trip}} d_{ij} = \text{minimum}$$

i.e., when the total trip length is shortest. *

* Much of the mathematics used in describing neural networks is identical to that found in the treatment of a set of problems called *spin glasses* [15], named for a type of magnetic alloys containing both ferromagnetic and antiferromagnetic spin orderings.

6. Philosophy of neural networks

A peculiarity of neural networks is that they do not always pick the absolutely shortest tour, but rather pick one of a set of nearly degenerate short tours whose lengths vary only slightly from that of the shortest tour. It is not yet understood whether this is a fundamental limitation of the method or whether it will ultimately be possible to ensure perfect performance, but, as Hopfield and Tank point out, bad solutions in their 30-city test network were rejected over good ones by a factor of 10^{23}. For many applications this kind of performance is good deal more than adequate. On a trip totalling many thousands of kilometers, differences of a few kilometers are just not important. Similarly, a very fast track finding algorithm which misses a few points or splits a track occasionally is probably preferable over a very slow one which is perfect. In experimental high energy physics, perhaps more so than other fields, fast, approximate solutions are at a premium, whether it be for triggering purposes, on-line event reconstruction, or to provide approximate starting points to off-line reconstruction programs. At the same time, of course, it is necessary to understand possible biases which may be introduced by the use of approximate solutions.

There are no known 'fast' algorithms for finding *optimal* solutions to the TSP. Of the approximate methods (i.e., those giving near-optimal solutions), model neural networks are not at present the best or the fastest. Neural network methods are still of considerable interest however, for the TSP as well for other problems, because of their conceptual simplicity, wide range of applicability, intrinsic parallelism, and promise of eventual hardwire implemention.

It is also possible to handle 'intractable' minimization problems like the TSP using the formalism of *simulated annealing* [16]. In this method, a cost function first is evaluated using some starting values of the state variables of the system. The state variables are then changed by small random amounts, and the cost function reevaluated. A Metropolis algorithm is used to decide whether to accept this new configuration of the system, i.e., if the new value of the cost function is smaller than the previous one, the new configuration is accepted, but if the new value is larger than the old by an amount ΔE, it is accepted only with probability $e^{-\Delta E/T}$ where T is a control parameter analogous to temperature. This method has two advantages: 1) the cost function need not necessarily be expressible as a sum of terms bilinear in the state variables, as in the case of neural networks, and 2) the system is less likely to become trapped in local minima, since the Metropolis function allows the system to escape from them by permitting temporary increases in the cost function. In typical applications, an 'annealing schedule' is followed in which several minimisations are made, each time lowering the control parameter T. Simulated annealing has, however, two important disadvantages from the point of view of high energy physics. The first is that the high quality global minima are computationally extremely expensive: simulated annealing does not offer the promise of speed that neural networks do [17]. The second is that there is no apparent way to implement simulated annealing in parallel hardware.

For problems with high combinatorial complexity, vector computers are faster than serial computers because they can calculate large numbers of combinations simultaneously, continuing until all combinations have been exhausted. Neural networks are even faster since they effectively employ a *directed* search in which the solutions are *weighted* by their quality. Bad solutions damp themselves out very quickly through negative feedback. The network effectively considers many solutions simultaneously via the system of interconnections, continuously evolving toward configurations of lower energy.

The TSP is an example of a system for which the model of the network, implemented on a vector machine, would be faster than an exhaustive search. To demonstrate this we estimate the number of calculations necessary in the two cases. Consider first a vector computer. An N city tour has $\frac{1}{2}(N-1)!$ possible solutions. Each solution involves the summing of $N-1$ terms to calculate the tour length. At the end of this, $\frac{1}{2}(N-1)!$ comparisons must be made to find which combination gave the shortest tour. The total number of calculations is thus

$$N_T = (N-1)\frac{(N-1)!}{2} + \frac{(N-1)!}{2} = \frac{N!}{2}.$$

Now consider the neural network. There are N^2 neurons. Each one is connected to $4(N-1)$ others. In each iteration, each connection must be multiplied by its coefficient, and the inputs of each neuron must be summed. This implies $2 \times 4(N-1)$ operations per iteration. In addition, there will be some 8 or so other operations per iteration associated with the function calculation, etc. Thus,

$$N_T \sim N^2[8(N-1)+8]\,N_{\text{iter}} = 80N^3,$$

where we have assumed N_{iter}, the number of iterations, is about 10, a typical number of iterations required for convergence. For a 30 city tour, we find

$$N_{T,\text{exhaustive search}} \sim 1 \times 10^{32},$$

$$N_{T,\text{neural net}} \sim 2 \times 10^{6}.$$

The neural net is 26 orders of magnitude faster, ignoring possible effects due to differences in average vector length in the two cases, in spite of the fact that the coefficients have to be explicitly multiplied and inputs explicitly summed on each iteration [18].

7. Combinatorics in high multiplicity events

The energy function minimised by a neural network can of course be any function expressible as the sum of pairwise terms between the neurons. We consider here a method of finding the set of n tracks in an event of multiplicity N, supposed large, which has the smallest value of some function $F = \sum_{i \neq j}^{n} f(p_i, p_j)$ where f is any function of the four-vectors p_i and p_j. Suppose, for example, we wish to find the set of n tracks which have the largest overall invariant mass M. First note that

$$M^2 = \left| \sum_{i=1}^{n} p_i \right|^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} p_i p_j$$

$$= \sum_{i=1}^{n} m_i^2 + \sum_{i \neq j}^{n} p_i p_j,$$

where m_i is the mass of the ith track. We assume that the m_i are all roughly equal and that thus maximizing the second term above will maximize M^2 (an alternate method would be to incorporate the m_i into externally applied bias currents to the neurons).

We construct the N by n neural network shown in fig. 5. The horizontal position of a neuron indicates the track number, and the vertical position indicates the order of that track in the group of n tracks being considered. Valid solutions to the problem have one neuron in each row and column, as in the TSP. To this end, inhibitive connections are made between each neuron and the others in its row and column. The coefficients between two neurons not in the same row or column will be of the form $T_{ij} \propto p_i p_j$. The system thus will evolve toward the configuration in which the value of M^2 for the n chosen tracks is largest. Events for which this largest value is above some threshold could be flagged for further study.

We can again compare the number of calculations needed to perform this task through an exhaustive search to that required for a *model* neural network. We assume that all the $f(p_i, p_j)$ have been calculated separately in advance. For the exhaustive search, the total number of combinations is $N!/n!(N-n)!$ while the number of sums to be performed for each combination is $n!/2!(n-2)!$. The number of compares to be made at the end is also $N!/n!(N-n)!$, so that the

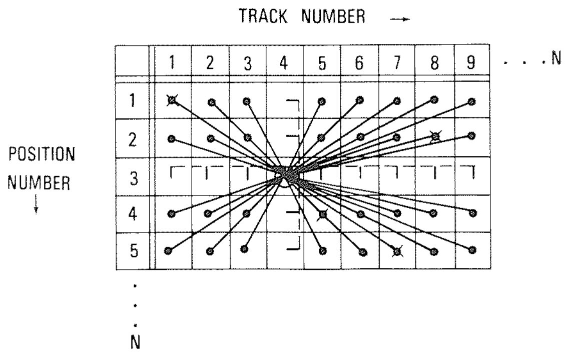

Fig. 5. Combinatoric function minimization network. Connections for neuron (4, 3), represented by ○, are shown, other 'on' neurons are represented by ×. Solid lines are reinforcing coefficients, broken lines are inhibitive coefficients. Each neuron is connected to every other neuron.

Table 1
Comparison of approximate total numbers of calculations necessary to find, among a set of N tracks, those n which have the smallest value of a function $F = \Sigma_{i \neq j}^{n} f(p_i, p_j)$, for an exhaustive search and for a model neural network

N	n	$N_{T,\text{exhaustive search}}$	$N_{T,\text{neural network}}$
35	4	3.7×10^5	3.9×10^5
50	4	1.6×10^6	8×10^5
50	10	4.7×10^{11}	5×10^6
50	25	3.8×10^{16}	3.1×10^7
64	6	1.2×10^9	2.9×10^6

total number of calculations is

$$N_{T,\text{exhaustive search}} = \frac{N!}{n!(N-n)!}\left[\frac{n!}{2!(n-2)!} + 1\right].$$

For the neural network, the number of neurons is Nn, the number of connections for each neuron also Nn, and the number of calculations per iteration thus roughly $2N^2n^2$. For 10 iterations the total number of calculations is thus

$$N_{T,\text{neural network}} \sim 20N^2n^2.$$

The totals for a few particular cases are presented in table 1. For $N = 35$, $n = 4$, the two methods are roughly equivalent. For $N = 50$, $n = 4$ the model neural net is better, but only by about a factor of 2. For $N = 64$, $n = 6$, an improvement of 10^3 is predicted, and for $N = 50$, $n = 25$, an improvement of 10^9 (although admittedly a resonance with 25 tracks is not a very sensible thing to want to look for). It is thus the exact nature of the problem which will determine whether a model neural net is faster than an exhaustive search; nevertheless, a hardwired network (with all possible tracks precalculated so that the T_{ij} do not have to be changed for each event) should always be fast, for any N and n, and could be used, for instance, as part of an experimental trigger (see also ref. [18]).

8. Track finding with a neural network

Most track finding programs use a considerable amount of computing time in trying numerous combinations of points, most of which are immediately rejected, in an attempt to find track segments which will then be used to build tracks. Thus, track finding looks like a perfect candidate for treatment on a neural network. For a complementary review of more conventional track finding methods, see the recent article of Grote [19].

Consider a set of N space points measured in a TPC. From these points we define a set of legal *segments*, defined as segments joining two points in the set with lengths less than some cutoff R_c, where R_c has a value of several times the mean distance between space points. Thus each point can be thought of as interacting only with those points within radius R_c, to form possible track segments. Each legal segment will be represented by a neuron, as in the array in fig. 6. The circle in the figure represents the presence of the directed segment $3 \to 2$. The solution to the problem will consist of an array in which all legal segments which belong to tracks are 'on'. Each final track

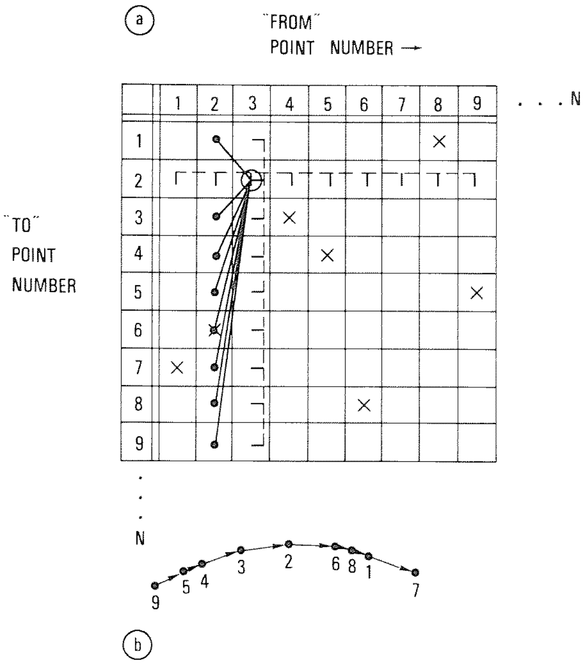

Fig. 6. Neural track finding. ⓐ Connections are shown for neuron $3 \to 2$, represented by ○. The reinforcing coefficients are represented by solid lines, the inhibitive coefficients by broken lines. ⓑ The track segment $9 \to 5 \to 4 \to 3 \to 2 \to 6 \to 8 \to 1 \to 7$ is represented in ⓐ by ○ and the other 'on' neurons, ×.

will consist of a non-bifurcating, unbroken chain of segments, which can be read out by choosing an arbitrary point on the chain and following the sequence in both directions until the ends of the track are reached. On a valid track, no point should have more than one directed segment entering or leaving it, and no point should appear more than once. For this reason, inhibitive connections are imposed, as in the previous examples, between each neuron and those in its row and column. The reinforcing coefficients will be set up such that the joining together, end-to-end, of short segments of similar direction will be favored, in order to ensure a smooth track. For the neuron $3 \rightarrow 2$ in the figure, for instance, the reinforcing coefficients will be imposed between itself and all the neurons in row 2, i.e., neurons $2 \rightarrow k$ for $k = 1$, N. This chaining together of segments with similar direction is similar in some ways to the *minimal spanning tree* approach to track finding [20].

To see if this approach might actually work, a test network was set up on the VAX 11/785 at L.A.L. Orsay. The reinforcing coefficients were chosen to have the form

$$T_{ij} \propto \frac{\cos^n \theta_{ij}}{r_{ij}},$$

where θ_{ij} is the angle between segments i and j, r_{ij} is the length of the vector sum of segments i and j, and n is a small integer. We shall call these *type 1* coefficients. The θ dependence favors pairs of neurons with similar slopes, while the r dependence assures that the shorter neurons are favored. With these coefficients, the lowest energy configurations will consist of chains of short segments which follow smooth curves. We can liken the track finding to threading a piece of flexible spring wire through the eyes of a set of fixed needles: it is the total stored energy in the wire that is being minimised by the network. Clearly solutions in which the wire bends sharply or doubles back on itself will have higher stored energy.

The track recognition capability that we seek is similar to some of the processes studied in early vision research. One of the goals of this field is to understand the mechanisms by which visual field data in the retina are processed before being sent to the brain. There, as in our case, the basic task is to provide algorithms which can extract information about objects in the three dimensional environment from sparsely sampled, often noisy, two dimensional data fields. The use of computational neural networks has been investigated for such early vision problems as edge detection, surface reconstruction [21], and velocity field estimation [22].

The network was tried on a set of simulated events with from 1 to 6 tracks. The sigmoid function used was a piecewise linear approximation as shown in fig. 7, and the parameters used in these tests were: $n = 5$, $\Delta t = 0.5\tau$ per time step, and $R_c = 4.5\langle r \rangle$, where $\langle r \rangle$ is the mean distance between adjacent points on the same track. In order to keep the computing time and memory requirements low, the number of points per track was kept below 20. Some sample results are shown in figs. 8a and b, which show 4 stages in the evolution of a 4 and a 5 track event, respectively. In fig. 8a the network has faultlessly reconstructed a 4 track event with two crossing points. Fig. 8b shows a somewhat more typical result: confusion in regions where tracks are very close together, leading to incorrect or illegal (e.g., three neurons at same point) solutions in these regions. In the last frame of both figures, evolution has stopped, i.e., conver-

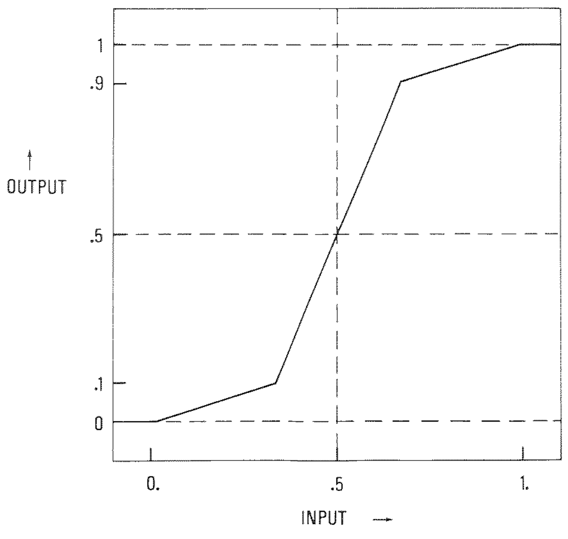

Fig. 7. Piecewise linear sigmoid function used in track and cluster finding simulations.

gence of the system has occurred. Convergence usually occurred in less than 10 iterations, i.e., less than 5 time constants. It is believed that the overall performance can be improved by a more judicious choice of coefficients, as a rigorous optimization has not been done.

In an attempt to improve the performance for close together tracks, another type of coefficient, which we shall call *type 2*, was tried. These represent connections between neurons which do not share an endpoint, but nevertheless are relatively near to each other. The value of the type 2 coefficient between two neurons is proportional to the $1-\chi^2$ of a fit to a circle through the four points that determine the two neurons. No type 2 coefficient is assigned if the neurons in question are more than a few R_c away from each other, or if the radius of the fitted circle is less than several times the length of the longer of the two neurons. Because the radius of the fitted circle is determined by the neurons themselves, only local smoothness is imposed, there is no restriction on the global functional form of the resulting path, (although the requirement of a constant radius of curvature is an option that could be of interest in triggering, see section 9). The presence of type 2

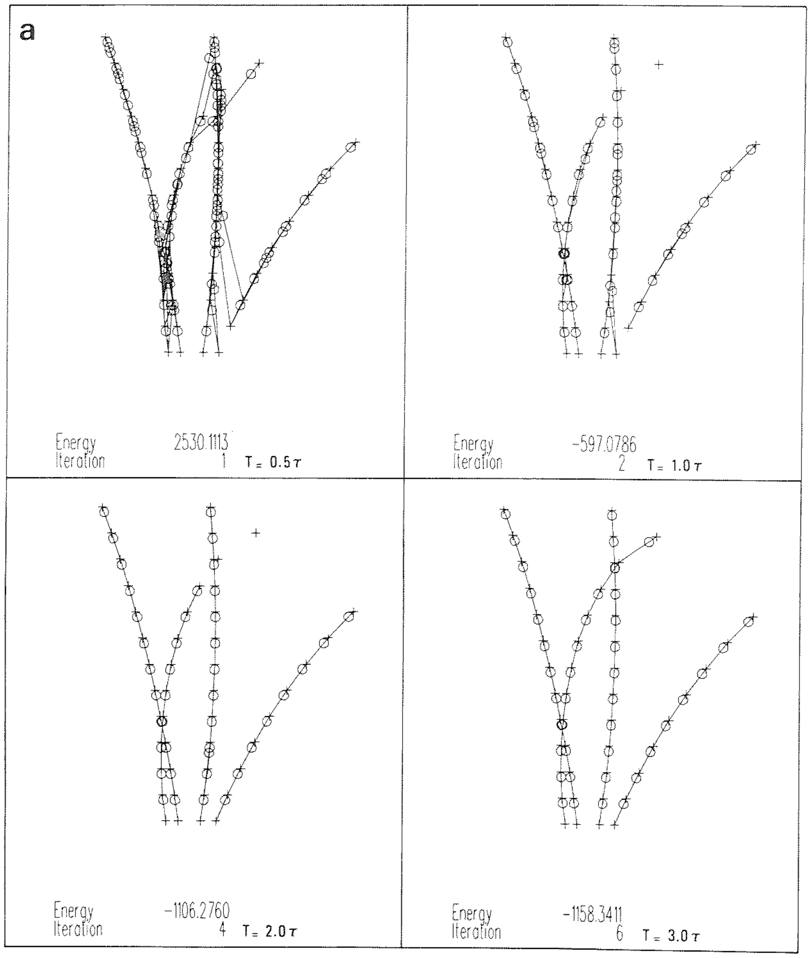

Fig. 8. (a) Track finding with a model neural network. Total network energy, iteration number, and total elapsed time, T, are given. Measured space points are represented by crosses, neurons by segments joining points, with a circle at the neuron head indicating direction. Only neurons with output values greater than 0.1 are drawn. In practice, most neurons are found to have values near either 0 or 1. This 4 track event is perfectly reconstructed.

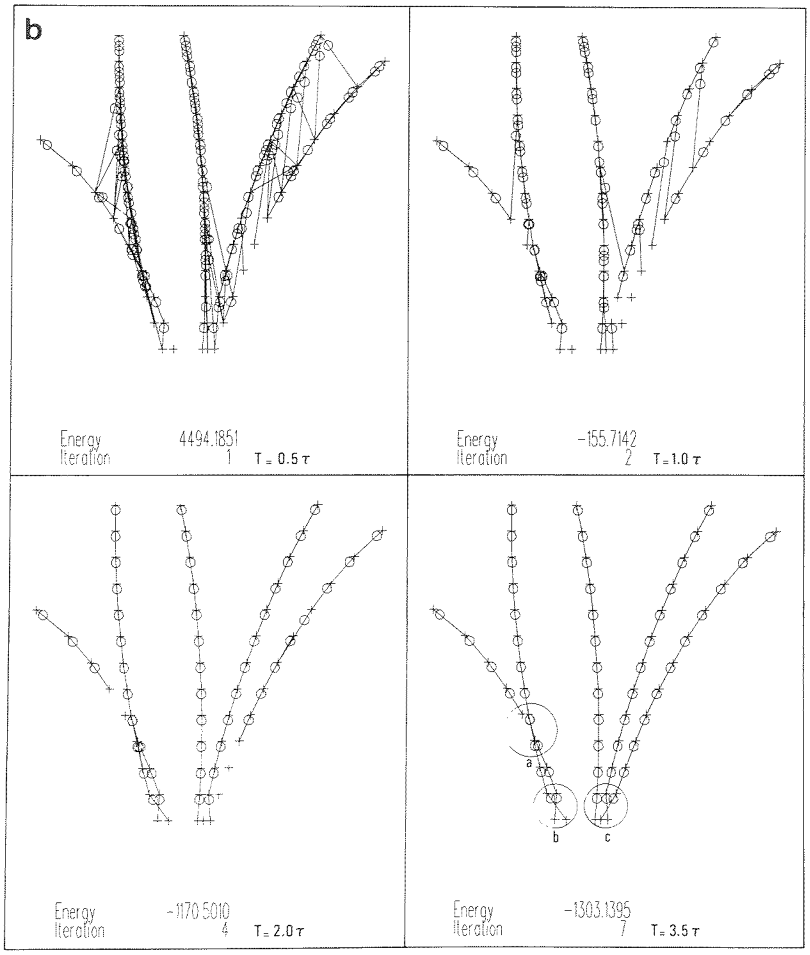

Fig. 8. (b) same notation as (a). At convergence (final frame), the reconstruction is not perfect. Examples of missing or illegal neurons (a), and incorrect choice of neurons (b), (c) are observed.

connections, if they prove essential, increases the total number of connections, but not dramatically, since, like the type 1 connections, they are local. With both type 1 and type 2 coefficients in use, it was possible to bring two concentric arcs quite close together (e.g., radii of 1.0 and 0.98) without failure. Tests with both type 1 and type 2 coefficients on higher multiplicity events are currently in progress.

The choice of coefficients seems to be rather delicate, and a number of attempts were made before arriving at forms which gave reasonable results. There is much to be learned about how to choose the form of the coefficients and the relative normalizations of reinforcing and inhibitory coefficients. The effect of the gain of the response function, the time step per iteration, and the initial conditions on the evolution of the network must also be better understood. There will surely be some difficulties to be resolved in passing from these simple tests to a full fledged track reconstruction program for a real high energy physics experiments, but the results so far are promising.

An interesting feature of this method is that the tracks do not need to come from the origin, nor do they need to be helices. The network will find and associate together groups of points that follow any reasonably smooth curve at any location in the volume being examined. Thus it is ideal for cases where the magnetic field is not uniform, and for finding tracks from secondary vertices. Noise points will in general be far from real tracks, and segments including them would make unacceptably large angles with other segments in a track. This in turn means that such a configuration is unfavored 'energetically', with the (desirable) result that noise points will be simply 'ignored'.

It would be interesting to determine if track finding with a model neural network is fast. To do this properly, a comparison should be made, using realistic data, between a model neural program and a 'conventional' program, both developed on a vector computer, or other massively parallel devices, such as the Connection Machine. This will be an undertaking of some magnitude, but in the meantime it is possible to make some rough approximations.

First we make two simplifying assumptions. The first is that the coordinates of the points are *binned*, so that points can occur only at discrete locations. In a hardwired network it is the granularity of the network which would ensure this characteristic. If a given point can have a maximum of m possible neighbors within R_c, and each of these in turn $m-1$ neighbors, there will then be a set of only $m(m-1)$ possible *different* values of coefficients for legal segments (for simplicity, we consider here only type 1 coefficients), and this set will be sufficient to describe the entire network. Thus it will not be necessary to calculate new coefficients for each event. The second assumption is that, in the model, we can ignore points that are not 'on', thus eliminating a large number of essentially irrelevant calculations.

Let m now be the mean number of 'on' space points that are within R_c of any given space point. This number will of course depend strongly on the density of measurements along each track and the spatial density of tracks, but we interpret m here to be the average over the data set chosen. This means that for each of the N points, there are on average m legal segments that can connect it to another nearby point. This second point then will have $m-1$ possible new legal segments that can contain it. Thus the total number of reinforcing connections in the network is $Nm(m-1)$ (since neurons that do not share an end point are not connected to each other in the network). A similar argument shows that there are roughly $2Nm$ inhibitory connections for a grand total (10 iterations) of

$$N_{T,\text{model network}} \sim 10N(m^2+m).$$

Suppose we choose 'histogramming' as the 'conventional' method to which to compare the model network. In one application of this procedure [23] the quantity

$$\sin(\lambda) = \frac{z}{\sqrt{x^2+y^2+z^2}}$$

is histogrammed for all measured space points. A peak finding program then is applied to the resulting histogram. Points that contribute to the same peak in the sin(λ) distribution will have come from the same track. To calculate and histogram sin(λ) for N points should require of the order of $10N$ calculations. The number of calculations in the peak finding is proportional to the number of bins, which we presume to be much smaller than the number of points, and thus negligible, although as pointed out in ref. [20], in some cases the peak finding can become prohibitively complicated. Thus,

$$N_{T,\text{histogramming}} \sim 10N.$$

This suggests that the histogram method will be considerably faster than the model network, in those cases where histogramming is feasible, since m will typically be of the order of a few tens or so. It must be remembered of course, that histogramming will only work on helical tracks through the origin, while the neural approach will work for any kind of track. Clearly, a detailed simulation is required to determine which method is better overall for a given case.

9. A neural trigger

Hardware neural networks are capable of providing detailed reconstructed event information, on tracks, calorimeter clusters, etc., on a time scale of microseconds, and thus are of interest from the standpoint of triggering.

Darbo and Heck [24] have developed a contiguity trigger for the Delphi TPC at CERN which is similar in many ways to a neural network. Ionization from a track in the TPC drifts to the wire planes where it is collected. Avalanches at the wires induce signals on the pad arrays behind the wires. These signals give r and ϕ information, from pad position and sharing, and θ information, from drift time. For each of 10 θ bins, the r, ϕ information is mapped onto a rectangular array of nodes, called an Image Memory (IM), of dimensions 144 bins in ϕ and 16 bins in r. Each node in the IM can be connected to its nearest neighbors by means of addressable latches. A particular pattern of connections for a given node is called a *contiguity mask*. The contiguity mask pictured at the top of fig. 9 can be described as follows: each 'on' node is connected to its neighbors above and below, and in addition, the right-hand neighbor of each 'on' mode is also connected to its neighbors above and below. Tracks are found by applying the same contiguity mask to all 'on' nodes and looking for a continuous electrical path across the IM. Fig. 9 shows an IM which contains 2 tracks, one of which (the one on the right) is found using the mask chosen, and the other, not found. The efficiency of this trigger can be studied as a function of the type of contiguity mask (fig. 10). This trigger, because of its array of simple nodes and high connectivity, resembles a neural network.

Consider now a 'true' neural approach to the same problem. Let each IM node be connected to each of its nearest, second-nearest, and third-nearest neighbors for a total of 48 connections per node (fig. 11). Following the treatment of the previous section, but with N now being the number of *possible* points (i.e., the number of nodes) and with $m = 48$ being the total number of *possible* neighbors, not 'on' neighbors, we find

$$N_T \sim N(m^2 + m)$$
$$\sim 144 \times 16 \times (48^2 + 48)$$
$$N_T \sim 5 \times 10^6$$

for the total number of connections. Based on the circuit densities given in section 3, a trigger of this design should fit quite handily on a single chip, and be capable of finding tracks in an IM on a

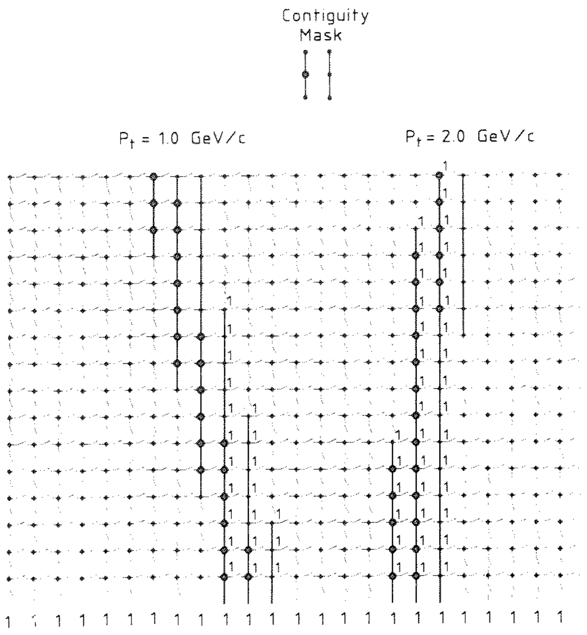

Fig. 9. Using the contiguity mask shown at the top, the trigger finds the track on the right, which has a continuous path across the IM, but does not find the track on the left. From ref. [24].

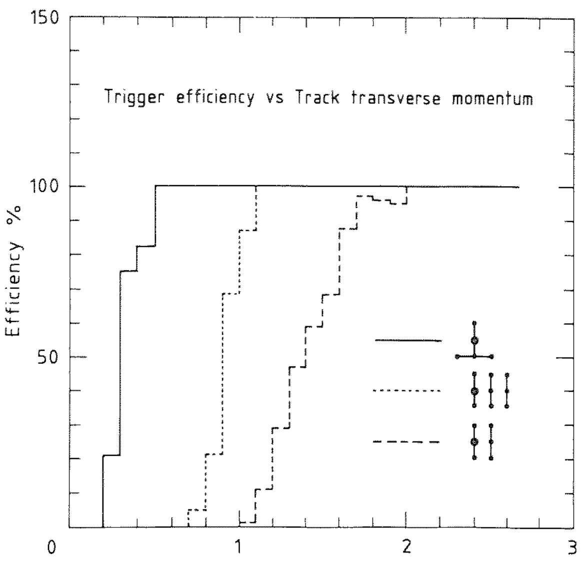

Fig. 10. Efficiency of Delphi TPC contiguity trigger as a function of p_t for 3 contiguity masks. From ref. [24].

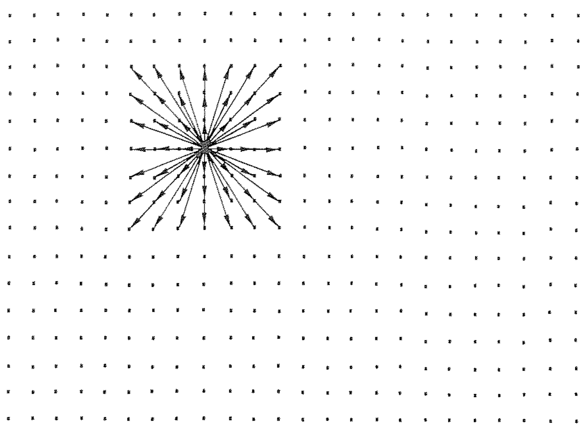

Fig. 11. IM with one node mapped onto its nearest, second nearest, and third nearest neighbors. Each of the 48 arrows constitutes a neuron. Each node has exactly the same pattern of connections. Based on the IM definition of ref. [24].

time scale of microseconds. The exact implementation of the trigger needs to be specified before the question of rejection power can be properly addressed, but presumably rejection of noise events can be based on lack of convergence of the network, or convergence only to very shallow minima. The neural net will be somewhat more tolerant of chamber inefficiencies than the contiguity trigger, since it considers a wider range of possible connections. Also, the neural work network should be sensitive to tracks of any momentum (i.e., curvature), while a given contiguity mask does not (by definition) efficiently find tracks whose momenta lie outside the range for which the mask was chosen. Thus, the track information is more detailed, which opens the possibility of making more sophisticated trigger decisions.

An alternative approach to the neural trigger would be to load in type 1 and type 2 coefficients which correspond to tracks of a specified constant radius of curvature. The network would then only converge, i.e., give a trigger, when a track with this curvature is present. In this configuration, the loading of the coefficients in the neural trigger is analogous to the loading of a contiguity mask in the contiguity trigger. If the tracks are all roughly the same length, they will each contribute roughly equally to the energy function, so that the energy function could perhaps be used to determine the number of tracks found.

10. Cluster finding with a neural network or cellular automata

Consider a homogeneous multicell calorimeter in which N cells are above some threshold. We wish to partition these N cells into subgroups of contiguous cells, i.e., into clusters of energy. Form an N by N array of neurons as in fig. 12. In each column, reinforcing coefficients are set up between each neuron and those corresponding to its nearest neighbors in the calorimeter. There are no connections between neurons in different columns, nor are there any inhibitory connections.

As starting values, each neuron on the diagonal is set to 1 and all others are set to 0. In each column, the 'on' neuron will proceed to turn on the neurons corresponding to its neighbors in the calorimeter, which in turn start to turn on their neighbors, etc. Finally, in each column, all neurons that are in the same cluster as the neuron that was initially on in that column will be turned on. The number of clusters will be equal to the number of different kinds of columns, and the content of each cluster can be read off from any of the identical rows associated with that cluster.

This method was used in a test on the L.A.L.

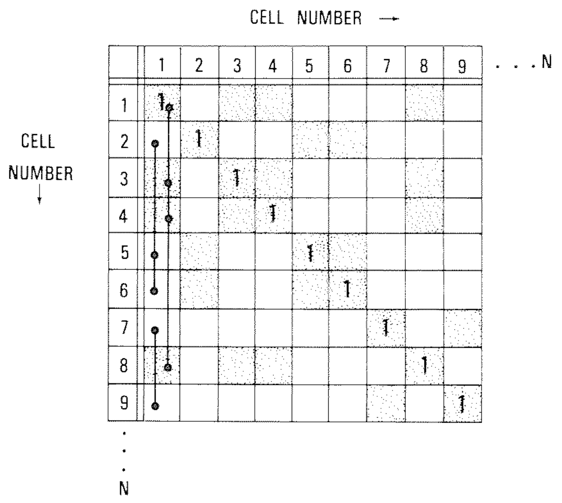

Fig. 12. Cluster finding with N^2 neurons or cellular automata. The reinforcing connections are the same for all columns and are only shown in column 1. Starting values of 1 are put along the diagonal. The hashed areas indicate cells that will be 'on' in the resulting attractor. The clusters found are (1, 3, 4, 8), (2, 5, 6), (7, 9).

VAX. Four energy clusters, of from 2 to 10 cells each, were found simultaneously in a 50 by 50 cell calorimeter. With steps of 0.5 time constants per iteration, convergence took 3 iterations (though this will clearly depend on cluster size). The sigmoid response function used was the same as in the track finding test mentioned earlier.

The clustering can clearly also be done with cellular automata. In one approach, the array of automata is exactly the same as that of the neurons, but instead of using a sigmoid response, we use the transition rule that a neuron will turn itself on in a particular iteration if at least one of its calorimeter neighbors is on. This system will evolve exactly as did the neurons.

These two methods will have roughly the same speed since they are actually almost identical. * In both cases there are N^2 cells. Let m be the average number of cells to which a given cell is connected. There are thus roughly N^2m total connections for either of these two methods, and it is easily seen that the numbers of calculations per connection is roughly one for each case.

The number of iterations needed can be estimated in the following way. We turn on one cell at random. In the cycle that follows, its nearest neighbors will turn on, in the next cycle, its next nearest neighbors, and so on. The cells will begin to turn on in concentric 'rings' which expand away from the chosen cell like ripples in a pond. In each cycle the radius of the ring will increase by roughly one cell, and the expansion continues until the edge of the cluster is reached. (For simplicity we have taken the case of an 'average' cell located near the middle of the cluster.) The number of cycles required is thus proportional to the characteristic linear dimension of the cluster. Assuming a 'globular' cluster of N_i cells, this will give, $N_{\text{iter},i} \sim \frac{1}{2}\sqrt{N_i}$. Clearly the largest cluster will be the determining one, so that,

$$N_{T,N^2\text{cells}} \sim \tfrac{1}{2}N^2m\sqrt{N_{\max}},$$

where $N_{\max}$ is the number of cells in the biggest cluster.

A faster cellular automata approach involves using N, multistate automata, with one automaton per calorimeter cell (see fig. 13). Let each automa-

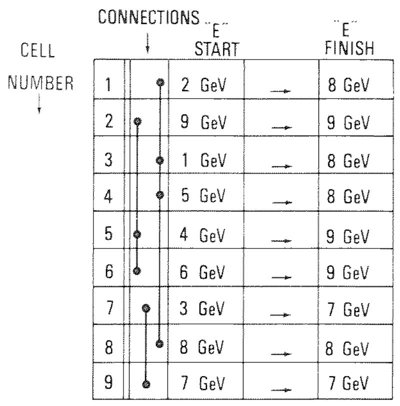

Fig. 13. Cluster finding with N multivalued cellular automata. Clusters are the same as in the example of fig. 12.

ton's 'state variable' be equal to the energy contained in that cell (we could equally well use the cell number instead of the energy). The transition rule this time will be that each automaton takes on as the new value of its state variable the largest state variable value of any of its 'on' neighbors. The system will evolve to a configuration in which all cells belonging to the same cluster will have taken on a value equal to the energy of the highest energy cell in that cluster. Assuming that no two peak cells had exactly the same energy at the beginning of the calculation, this classification will be unique. This approach might well be called the 'contagion' approach: the identity of the unique element in each cluster (the one with largest energy, e.g) is a 'germ' which is passed from one automaton to the next until all in the cluster have been 'infected'. The disease cannot spread to other clusters since elements in different clusters by definition do not touch each other.

With this method, each of the N cells compares its own value to those of its m neighbors, for roughly Nm calculations per cycle. The number of iterations will go as $\sqrt{N_{\max}}$ as before so that

$$N_{T,N\,\text{cells}} \sim \tfrac{1}{2}Nm\sqrt{N_{\max}}.$$

For comparison we consider a more 'traditional', vector approach. We form 3, N by N arrays, A, B and C, all initially set to zero. In each row i of A we set to 1 the ith bit, as well as the bits corresponding to the nearest 'on' neighbors of i. Each row i can now be thought of as a vector A_i which represents a 'minicluster' centered on the cell i. We initialize by setting $C_i = A_i$ and leave B all zeroes.

In each iteration, we form for each i a vector Q_i by *or*-ing together B_i and all vectors A_j where j is the index of a non-zero element of C_i. We then set $C_i = B_i \oplus Q_i$ and finally $B_i = Q_i$, where $\oplus$ is the *exclusive or* operator. Thus B_i contains the sum total of elements known at this stage to belong to the same cluster as i, while C_i contains just the new elements of the cluster found in this iteration. The process continues until the C_i become zero, indicating that all the elements of the clusters have been found.

For an 'average' initial cell near the center of a 'globular' cluster, the number of new elements found will increase roughly linearly each iteration, in steps of m, the mean number of neighbors, until the last iteration (again determined by the largest cluster) where some $4\sqrt{N_{\max}}$ new elements will be found. The number of calculations required in each iteration is roughly m per new element so that

$$N_{T,\text{vector}} \sim Nm\left[m + 2m + 3m + \cdots + 4\sqrt{N_{\max}}\right],$$
$$N_{T,\text{vector}} \sim 8NN_{\max}.$$

These calculations are rather crude but it appears that, for typical numbers, the N-automata approach may be advantageous.

11. Associative memory in neural networks

11.1. Feedback network

An exciting aspect of neural networks is their potential for information storage. Suppose we use an array of pixels to represent an image, a face, for instance. We first set up reinforcing coefficients between all the 'on' pixels, then extinguish the array. Subsequently, if we turn on any set of pixels that were previously on, the system will evolve, because of the reinforcing coefficients, to the state in which the image is on once again. The image can be thought of as being 'stored', and can be 'recalled' by specifying a subset of the 'on' pixels in the image. If we wish to store a second image in this same network, in order to ensure 'noiseless' recall, it will be necessary to modify the coefficients in the regions of overlap of the two images, by including inhibitory coefficients, for instance, so that the two images can not turn each other on. Clearly, the subset of a given image that must be specified in order to solicit its recall will have to be larger now: in particular, it must contain more than the set of points common to both images, in order to uniquely specify the image desired.

Similarly, as more and more images are added to the 'memory', the coefficients of all images stored thus far will have to be modified slightly in order to keep the 'recollections' distinct. (Also, the subsets of points needed for recall will have to be more specific. This is equivalent to saying that the set of points applied as input to the network must be sufficient to place the system within the basin of attraction of the target response.) This iterative process of 'learning' in order to store memories was first discussed in detail by Hebb [25]. The coefficients are established by the iterative learning procedure: initial values are chosen at random, and then the coefficients are adjusted slightly after each image is presented, in such a way as to decrease the difference between the response of the network and the desired response. (This difference is often expressed as the *Hamming distance*, defined as the number of pixels which are not identical in the desired and obtained responses.) After several iterations through the set of images, the coefficients will be effectively optimised for the set of images chosen. It has been shown [14] that the 'Hebbian' coefficients for the storage of p memories in an array of N neurons can be expressed as

$$T_{ij} = \frac{1}{N}\sum_{\mu=1}^{p} \xi_i^{\mu}\xi_j^{\mu}$$

where ξ_i^{μ} $(= \pm 1)$ is the value of neuron i in memory configuration μ.

We have here, then, a means of recalling the entirety of a stored pattern merely by specifying a unique subset of that pattern. This type of information storage is referred to as *associative* or *content addressable* memory. It is believed that human (or animal) memory of, say, familiar faces or objects, may operate along similar lines, e.g., being able to recognize a friend's face even though it is partially obscured. An optical device capable of carrying out exactly this task is described in ref. [12]. In another application, a model network was used to identify fingerprints from fragmentary or noisy versions of the prints [26].

Intuitively, it would seem that at some point the network would become saturated, i.e., that the number of coefficients would become insufficient to uniquely specify all the memory configurations. For coefficients as defined above, it has been shown [14] that in fact perfect or nearly perfect recall of memories is only possible if $p < \alpha N$, with $\alpha \sim 0.1$–0.2.

In high energy physics, the most common form of data storage is the mass storage of thousands to millions of events on magnetic tape or in large disk files. The binary information contained in a data event can, in principle at least, be represented as a pattern of pixels. One can imagine, then, a scenario in which a subset of an event, for example "a stiff track, an electromagnetic cluster, and 40 GeV missing energy", is specified, in pixel form, to a neural network. The network, if such an event exists, then displays the event.

In practice, there are a number of problems that will have to be addressed before associative memory becomes of use, in this form, at least, in high energy physics. One problem has to do with the relationship between p and N. To store a data sample of 1 million events, say, requires ($\alpha = 0.1$) 10 million neurons. The degree of connectivity required can not be a priori specified, but in the limiting case of a fully connected network, 10^{14} connections are required, which seems extremely high. Secondly, each event, in this scenario, should apparently be represented by 10 million bits of information, which in most high energy physics applications would be quite a bit more than required or desired. Finally, in high energy physics, it is usually a subset of events whose characteristics fall within certain limits that is desired, rather than a particular event that exactly fulfills a given set of criteria. Thus, in order that associative memory be useful in high energy physics, it will be necessary to a) arrive at a more appropriate relationship between the number of events, the sizes of the events, and the number of neurons, and b), to find a way to express the 'answer' as a set of events rather than a single event.

A number of methods have been put forward which address the first of these requirements, although it is not yet apparent how they might be implemented. One approach [27] is the generalization of the energy function to include trilinear or higher terms. This does in fact have the effect of increasing the storage capacity of the network, but the considerable increase in computational complexity makes this technique unattractive. Other approaches center upon the use of different types of learning rules. In one implementation, the network is able to continuously store incoming images by 'forgetting' the earlier images, the storage capacity at any given time being constant [28]. The use of non-local learning rules can also enhance storage capacity. The 'Hebbian' rule is a local one: the coefficient between two neurons depends only upon the values of those two neurons in the patterns stored. By allowing non-local terms in the learning rule, values up to $\alpha = 2$ have been achieved [29]. For random patterns, this represents the maximum storage capacity, but it has been shown that arbitrarily large numbers of *correlated* patterns can be stored, even though the total amount of information stored remains limited [30]. Non-local learning rules are frowned upon by some since they are improbable for biological systems, which are after all the model upon which neural networks are supposed to be based, but they may nonetheless prove to be useful. Another interesting approach to increasing storage capacity of neural networks is the use of sparsification techniques [31]. If z patterns of x bits are to be stored, a network of $n = 2\sqrt{xz}$ neurons is sufficient providing that the patterns can be encoded into n 0–1 strings of sparsity $\sim \sqrt{x}$.

11.2. Filter network

An associative memory can also be constructed using linear networks without feedback. The circuit will then be simply fig. 3 with the amplifiers and feedback loops removed. On the input line i

we apply the value, V_i, of pixel i in the input image, which we wish to compare to a set of p reference images. We have arranged that the resistance j in column i by equal to the reciprocal of ξ_i^j, the value of pixel i in image j. With this prescription, the output currents in row j are given by $\sum_{i=1}^{p} V_i \xi_i^j$. The output lines thus provide a list of the 'dot products' of the input image with each of the stored reference images. Additional circuitry can then be used to pick the image which best matches the input (i.e., has the largest dot product). A hardware implementation along these lines is described in ref. [32]. In high energy physics, this method might prove useful in 'template matching' applications, such as the track finding method of refs. [3,4] mentioned earlier, and is worthy of further exploration.

12. Conclusion

Pattern recognition has always been an important part of high energy physics data analysis. But, as one authors has put it [1], "This observational, pattern recognizing task... is an activity better matched to the capability of a brain than to that of a computer. We have applied the brute force numerical capability of computers to very non-numerical problems." We have seen, though, that these types of problems adapt themselves quite naturally to treatment with neural networks or cellular automata, which at the same time are an extension of the trend toward less intelligent but more highly connected processors in high energy physics that we have remarked upon earlier.

Encouraging progress has been made in the study of highly complex phenomena such as fluid flow through the use of cellular automata. In high physics, some of the most complex problems, as we have seen, can also benefit from the use of model neural networks. It has been shown that track finding can be implemented in a connective network approach in which segments between measured space points are represented by neurons. When a simple conventional approach, such as histogramming, is feasible, *model* networks, due to the high computational overhead that modelling entails, may not offer gains in speed, though more rigorous tests will be necessary to be certain of this. The high speed of existing hardwired networks suggests the investigation of scaled-down, coarse-grained networks that could rapidly calculate detailed tracking or calorimeter quantities for use in triggering. It is appropriate that these investigations, in addition to further studies of model networks, (using, e.g., a Connection Machine) be undertaken in the field of experimental high energy physics, traditionally a leader in the development and application of state-of-the-art technology and computing techniques.

Acknowledgements

The author wishes to acknowledge the computing and programming support of the *Service Informatique* at L.A.L. This work was supported by funds from the *Centre National de la Recherche Scientifique* of France.

References

[1] I. Gaines and T. Nash, Use of New Computer Technologies in Elementary Particle Physics, FERMILAB-Pub-87/38, Fermilab, 1987.

[2] D. Gosman et al., in: Proc. Topical Conf. on the Application of Microprocessors to HEP Experiments, CERN 81-07, eds. A. Michelini et al., CERN (1981) pp. 70–82, 83–90.

[3] C. Georgiopoulos et al., A Vectorized Track Finding and Fitting Algorithm in Experimental HEP using a CYBER-205, FSU-SCRI-87-08, Florida State University, 1987.

[4] J.J. Becker et al., Nucl. Instr. and Meth. A235 (1985) 502.

[5] D. Levinthal, H. Goldman, C. Georgiopoulos, J.L. DeKeyser, S. Linn, S. Youssef and M.F. Hodous, Comput. Phys. Commun. 45 (1987) 137.

[6] See, e.g., A.K. Dewdney, Scientific American (May 1985) pp. 10–16.

[7] S. Wolfram, Theory and Applications of Cellular Automata (World Scientific, Singapore, 1986); Scientific American (Sept. 1984) p. 140.

[8] T. Vicsek and A. Szalay, Phys. Rev. Lett. 58 (1987) 2818.

[9] J.J. Hopfield and D.W. Tank, Science 233 (1986) 625.
J.J. Hopfield and D.W. Tank, Biological Cybernetics 52 (1985) 141.

[10] L.D. Jackel et al., Electronic Neural Computing, AT&T Bell Labs, Holmdel, NJ 07733, presented at les Houches, April 1986.

[11] T.J. Sejnowski and C.R. Rosenberg, NETtalk: A Parallel Network that Learns to Read Aloud, Technical Report

JHU/EECS-86/01, Johns Hopkins University, Baltimore, 1986.
[12] Y. Mostafa and D. Psaltis, Scientific American (March 1987).
[13] W. Hillis, The Connection Machine (MIT Press, Cambridge, MA, 1985).
[14] W.A. Little, Math. Biosci. 18 (1974) 101.
J.J. Hopfield, Proc. Nat'l. Acad. Sci. USA 79 (1982) 2554.
[15] Reviewed in: Castellani, Di Castro and Peliti, eds., Disordered Systems and Localization (Springer, New York, 1981).
[16] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Science 220 (1983) 671.
[17] See, e.g., R.D. Williams, Optimization by a Computational Neural Net, Caltech note C^3 P-371, CALT-68-1409, 20 November 1986.
[18] There are some indications that neural network performance does not necessarily remain constant as the network size grows. See for example G.V. Wilson and G.S. Pawley, On the Stability of Travelling Salesman Problem Algorithm of Hopfield and Tank, Univ. of Edinburgh preprint, Department of Physics, 1987, submitted to Biol. Cybernetics.
[19] H. Grote, Rep. Prog. Phys. 50 (1987) 473.
[20] Ibid., ref. [19] and D. Cassell and H. Kowalski, Nucl. Instr. and Meth. 185 (1981) 235.
[21] C. Koch, J. Marroquin and A. Yuille, Proc. Nat'l. Acad. Sci. U.S.A. 83 (1986) 4263.
[22] J. Tanner, Ph.D. thesis, Caltech (1986).
[23] N.J. Hadley, Charged Hadron Production in e^+e^- Collisions at PEP with the TPC, Ph.D. thesis, U.C. Berkeley, 1983, LBL-16116.
CERN Delphi Collaboration Data Analysis Group, Report on Local Pattern Recognition Methods for the Individual Detectors in Delphi, Delphi 86-56 (May 1986).
[24] G. Darbo and B. Heck, The TPC Trigger for the Delphi Experiment, CERN/EF 86-22, 1986.
[25] D.O. Hebb, The Organization of Behaviour (Wiley, New York, 1949).
[26] E. Mjolsness, Neural Networks, Pattern Recognition, and Fingerprint Hallucination, Ph.D. Thesis, California Institute of Technology, Pasadena, CA, 1985.
[27] L.F. Abott and Yair Arian, Storage Capacity of Generalized Networks, Boston University preprint BUHEP-87-6.
E. Gardner, Multiconnected Neural Network Models, University of Edinburgh preprint 86/375.
[28] J.P. Nadal, G. Toulouse, J.P. Changeux and S. Dehaene, Europhys. Lett. 1 (1986) 535.
[29] L. Personnaz, I. Guyon, G. Dreyfus, J. de Phys. Lett. 16 (1986) L359,
I. Kanter and H. Sompolinsky, Phys. Rev. A 35 (1987) 380,
E. Gardner and B. Derrida, Optimal Storage Properties of Neural Network Models, University of Edinburgh preprint 87/397,
T.M. Cover, IEEE Transactions EC 14-3 (1986) 326,
S. Venkatesh (1986), in: Proc. of the Conf. on Neural Networks for Computing (Snowbird Utah), also Ph.D. Thesis, California Institute of Technology (1986).
[30] E. Gardner, Maximum Storage Capacity in Neural Networks, University of Edinburgh preprint 87/395,
[31] Technical Comment of Gunter Palm in Science 235 (1987) 1227.
[32] H.P. Graf and P. de Vegvar, Proc. Conf. Advan. Research VLSI, Stanford, ed. P. Losleban (MIT Press, Cambridge, MA, 1987) pp. 351–367.