HOW TO CLUSTER IN PARALLEL WITH NEURAL NETWORKS

Behzad Kamgar-Parsi
Center for Automation Research
University of Maryland
College Park, MD 20742

J. A. Gualtieri
Code 635
NASA GSFC
Greenbelt, MD 20771

Judy E. Devaney
Science Applications Research
4400 Forbes Blvd.
Lanham, MD 20706

Behrooz Kamgar-Parsi
Dept. of Computer Science
George Mason University
Fairfax, VA 22030

ABSTRACT

Partitioning a set of N patterns in a d-dimensional metric space into K clusters – in a way that those in a given cluster are more similar to each other than the rest – is a problem of interest in astrophysics, image analysis and other fields. As there are approximately $\frac{K^N}{K!}$ possible ways of partitioning the patterns among K clusters, finding the best solution is beyond exhaustive search when N is large. We show that this problem in spite of its exponential complexity can be formulated as an optimization problem for which very good, but not necessarily optimal, solutions can be found by using a neural network. To do this the network must start from many randomly selected initial states. The network is simulated on the MPP (a 128×128 SIMD array machine), where we use the massive parallelism not only in solving the differential equations that govern the evolution of the network, but also by starting the network from many initial states at once thus obtaining many solutions in one run. We obtain speedups of two to three orders of magnitude over serial implementations and the promise through Analog VLSI implementations of speedups comensurate with human perceptual abilities.

Keywords: Combinatorial Optimization, Synchronous Analog Network, Parallel Simulation, SIMD.

INTRODUCTION

Problems that involve data analysis are becoming increasingly severe in that data sets are becoming very large and their rate of acquisition is growing rapidly. It is clear that humans possess immense computational power for solving certain problems through visualization and that what is needed is the development of algorithms that have some of these capabilities.

The value of neural networks – whose development has been motivated by human beings' computational capabilities – as a computational device is yet to be explored. In fact, little is known about the reliability and complexity of these algorithms, and how they scale with the size of the problem. The work we present in this paper is an attempt to answer some of these questions. For this, we will concentrate on the problem of data clustering – a problem of interest in astrophysics, image analysis and other fields. The conjecture is that because of the many connections among neurons, neural networks should be particularly useful for the class of problems that involve collective decision making, of which one example is unsupervised clustering. Here the patterns must decide together how to partition themselves into subsets according to a given criterion. The problem considered here, as in all partitioning problems, is a *discrete* optimization with a goodness-of-fit criterion. By embedding this discrete problem in the continuous space of an analog network one can perform a downhill search on the energy surface which is more purposeful and effective than the search in the discrete space. Until hardware implementation of analog neural networks in VLSI become available – which is expected in the next few years [1] – simulation is going to be an indispensible tool in the study and design of these systems. Analog networks are intrinsically synchronous and hence well suited for simulation on massively parallel SIMD machines.

In this paper, we simulate the neural net we propose for solving the clustering problem on the MPP [a 128×128 SIMD array machine with 1024 bits of local memory per processor]. The issue of performance of neural net algorithms on parallel machines is also addressed. Before we proceed, however, we will discuss the clustering problem in some detail.

THE CLUSTERING PROBLEM

By clustering we mean partitioning a set of N patterns (the patterns are represented as points in a d-dimensional metric space) into K clusters in a way that those in a given cluster are more similar to each other than the rest. As there are approximately $\frac{K^N}{K!}$ possible ways of partitioning the patterns among K clusters [2], the problem has exponential complexity and finding the best solution is beyond exhaustive search. As is often employed, we let our criterion for *best solution* be the minimum square-error. That is, representing the patterns by d-dimensional points $\{\vec{r}_i | i = 1, \ldots, N\}$, the best solution is the one minimizing $\chi^2 = \sum_{i=1}^{N} (\vec{r}_i^{(p)} - \vec{R}_p)^2$ with respect to $\{\vec{R}_p | p = 1, \ldots, K\}$. Here cluster p contains the subset of the points, $\{\vec{r}_i^{(p)}\}$, and its centroid is given by $\vec{R}_p = \sum_{i=1}^{N_p} \vec{r}_i^{(p)}$, where N_p is the number of points in the cluster. A partitioning based on such a criterion is also known as minimum variance partioning. Because of the complexity of the problem, finding the best solution may not be possible. This, however, is not a major concern, because in practice usually only *a good* solution is sufficient.

Due to the importance of this problem many methods have been proposed by various researchers. (See Jain and Dubes [3] for a survey of the literature.) Many of these approaches are based on iterative schemes and often the differences between the suggested algorithms are quite subtle. The number of clusters K may or may not be fixed. For a given value of K, the essence of iterative algorithms is as follows.

After the initial partioning of the patterns into K clusters, their centroids, i.e. seed points in the d-dimensional metric space of the patterns, are computed. Each pattern is then assigned to the cluster with the nearest seed point and new centroids are computed. The process is repeated until the partitioning ceases to change. However, the process of the computation of new centroids can be carried out in two ways: (i) Keep the centroids fixed until the distances of *all* patterns to the K centroids are computed [4]; (ii) Update centroids as frequently as one pattern is found to be closer to the centroid of a cluster other than the one it is assigned to. In this case, the pattern is immediately reassigned and the centroids of the winning and the losing clusters are updated [5]. This method is sometimes referred to as K-means. Note that for a parallel machine, where the distances of the patterns from cluster centroids can be computed simultaneously, the first approach appears to be more efficient.

The neural net approach that we propose has many similarities with the iterative scheme described above. As will be explained later in more details, the major difference, however, is that the neural net allows a given pattern to belong to several clusters until the final iteration. That is, at least during the execution of the algorithm, a given pattern belongs to all clusters, though with different weights. The closest conventional method to this is the one proposed by Gordon and Henderson [6]. In their method, however, the sum of the weights for every pattern is restricted to one at any given iteration; thus, it dose not possess the full flexibility of neural networks.

As for the initial cluster centroids, one may take the first K points of the input data, which is very simple and inexpensive; or if one suspects the input points are prearranged in some special way, one may choose at random any K points of the input data [7]. More elaborate and expensive methods for choosing more promising initial centroids have been proposed in the literature (see Ref. [8] and [3]). Such methods, however, are not of interest to us.

OPTIMIZATION WITH NEURAL NETS

It has been recognized in recent years that artificial neural networks have computational properties [9,10]. The Hopfield model of neural network, which we use in this work, is particularly suitable for solving certain optimization problems. A neuron is a simple nonlinear processor that is connected to many (possibly all) other neurons in the network; it adds up the signals it receives from other neurons and fires a signal accordingly. The state of the network, that is the firing rates or activities of the neurons, through interactions with each other, change with time but eventually the network settles into a steady state where the neuronal activities remain constant. The energy of the Hopfield network is Lyapunov (i.e. it does not increase with time) and its minima are the steady states of the network. It is this property of neural networks that is used in optimization. The approach is to cast the problem in terms of an energy function that is then minimized by the corresponding network as it evolves spontaneously from some randomly selected initial state to states of lower energy. The energy function has typically many minima that represent valid solutions to the problem; deeper minima correspond to good solutions and the deepest minimum to the best solution.

In this paper we use analog neural nets, because they outperform digital nets in solving optimization problems [9,11]. Many problems of interest, including the problem we address in this paper, can be cast in terms of an energy function, E, that is quadratic in the neuronal activities and has the form [9],

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij} V_i V_j - \sum_{i=1}^{n} I_i V_i + \frac{1}{\tau}\sum_{i=1}^{n} \int^{V_i} dx\, g^{-1}(x). \tag{1}$$

Here n is the number of neurons in the network, and

V_i $(0 \le V_i \le 1)$ is the activity or firing rate of neuron i. The first term in (1) is the interaction energy among neurons, and the elements of the connection matrix, $T_{ij} = T_{ji} = -\frac{\partial^2 E}{\partial V_i \partial V_j}$, are completely determined from E. In the second term I_i is the bias or activity threshold of neuron i. The third term encourages the network to operate in the interior of the n dimensional unit cube $\{0 \le V_i \le 1\}$ that forms the state space of the system. In this term τ is the self-decay time of the neurons, and $g(u)$, a sigmoid function, is the gain or transfer function of the neurons that relates the input u_i to the output V_i. A standard form for g, which we will also use, is

$$V_i = g(u_i) = \frac{1}{2}(1 + \tanh\frac{u_i}{u_0}) = \frac{1}{1 + e^{-2u_i/u_0}}, \qquad (2)$$

where u_0 determines the steepness of gain. The neuronal activities, V_i, as well as the input signals, u_i, depend on time t. The evolution of the network is determined by the n coupled ordinary differential equations, $du_i/dt = -\partial E/\partial V_i$, which are

$$\frac{du_i}{dt} = -\frac{u_i}{\tau} + \sum_{j=1}^{n} T_{ij}V_j + I_i. \qquad (3)$$

We will set $\tau = 1$, so that time is measured in units of τ. Note that the *bias*-term can be eliminated from the energy and instead incorporated into the gain function if we define $V_i = g(u_i - \tau I_i)$.

To find a solution (i.e. a minimum), we start the network from a randomly selected state and let it evolve freely until it reaches a minimum of the function E and stops. As is usual in dealing with computationally intractable problems, we find not just one but several solutions by starting the network from different initial states, and then take the best one as *the solution* which may or may not be the optimum. Since a neural network converges rapidly to a minimum we can afford to run it many times thus ensuring that we find at least a very good solution. Below, we discuss how to construct an appropriate network for solving this problem.

CONSTRUCTION OF THE ENERGY FUNCTION

We want to partition a set of N points in a 2-D plane into the *best* K clusters (generalization to arbitrary dimensions is trivial) – best in the sense that sum of the squares of the distances of the points from their respective cluster centroids (i.e. sum of "within cluster variances") is minimized. We formulate the problem in a manner that can be solved by a neural network; that is we cast the problem in terms of an energy function that can be minimized by the network.

The energy function will consist of two parts: (i) constraint terms which make certain a point, at the end of the search, belongs to one and only one cluster; (ii) the cost term which is the sum of the residuals and is the function we actually wish to minimise. The formulation can best be illustrated through an example. Let us consider the case where we wish to partition $N = 10$ points into $K = 3$ clusters. A *possible* solution (not necessarily the best one) would be that, say, points 1, 2, 6 and 9 belong to cluster A, points 4 and 5 belong to cluster B, and points 3, 7, 8 and 10 belong to cluster C. This particular solution can be represented by the 3×10 rectangular array given in Table 1, where the rows are labeled by the clusters and the columns are labeled by the points. The elements of this matrix are 0 or 1 with the interpretation that "element A1=1" indicates that point 1 belongs to cluster A, "element B1=0" indicates that point 1 does not belong to cluster B, and so on.

Table 1: A possible solution for partitioning 10 points into 3 clusters.

Cluster	Points									
	1	2	3	4	5	6	7	8	9	10
A	1	1	0	0	0	1	0	0	1	0
B	0	0	0	1	1	0	0	0	0	0
C	0	0	1	0	0	0	1	1	0	1

If we think of the elements of this matrix as the activities of neurons ($n = K \times N$ neurons altogether), and denote them by V_{pi}, where p and i refer to the cluster and the point, respectively, then the constraint part of the energy function, E, can be expressed as

$$E = \frac{A}{2}\sum_{i=1}^{N}\sum_{p=1}^{K}\sum_{q\neq p}^{K} V_{pi}V_{qi} + \frac{B}{2}\sum_{i=1}^{N}(\sum_{p=1}^{K} V_{pi} - 1)^2, \qquad (4)$$

where the coefficients A and B are positive constants. The *A-term* has its minimum value (i.e. zero) if in each column (representing a point) at most one neuron is active and the rest are off. The *B-term* has its minimum value (also zero) if the sum of activities in each column equals 1. The two terms together enforce the *syntax* of the solution given in Table 1.

There is an additional constraint that we should, in principle, include in the energy function: that each cluster should contain at least one point. In terms of the solution matrix of Table 1 it means that in each row there should be at least one fully active neuron. Such a constraint can be imposed by $\sum_{p=1}^{K}\Theta(1-\sum_{i=1}^{N}V_{pi})$, where $\Theta(x) = 0$ for $x \le 0$ and $\Theta(x) = 1$ for $x > 0$ is the step function. However, since this term is nonanalytic its inclusion in the energy function creates problems and a better strategy appears to be to leave out this term and rather reject

those solutions that violate this constraint. In our simulations of neural networks (several thousand trials) the solutions never violated this constraint. Therefore, it appears that the absence of this constraint from the energy function is of little consequence.

To complete the energy function we must also formulate the *cost* term. We denote the square of the distance of point i from the centroid of cluster p (i.e. the residual) with R_{pi} which is given by

$$R_{pi} = (x_i - X_p)^2 + (y_i - Y_p)^2, \tag{5}$$

where (x_i, y_i) are the coordinates of point i, and (X_p, Y_p) are the coordinates of the centroid of cluster p. Here we have chosen the Euclidean distance as the metric; but one can define any metric one wants. Let us consider again the solution represented by Table 1. The sum of residuals or the cost for this solution is

$$(R_{A1} + R_{A2} + R_{A6} + R_{A9}) + (R_{B4} + R_{B5}) + (R_{C3} + R_{C7} + R_{C8} + R_{C10}), \tag{6}$$

which can be written as

$$\sum_{p=1}^{K}\sum_{i=1}^{N} R_{pi}V_{pi}^2. \tag{7}$$

Hence the energy function E, including cost and constraint, for this problem can be expressed in the final form

$$E = \frac{A}{2}\sum_{i=1}^{N}\sum_{p=1}^{K}\sum_{q\neq p}^{K} V_{pi}V_{qi} + \frac{B}{2}\sum_{i=1}^{N}(\sum_{p=1}^{K} V_{pi} - 1)^2 + \frac{C}{2}\sum_{p=1}^{K}\sum_{i=1}^{N} R_{pi}V_{pi}^2, \tag{8}$$

where C is also a positive constant. When the constraints (or the syntax) are satisfied the A-term and the B-term vanish and the energy function, E, reduces to just the cost term, therefore *deep minima* of E correspond to *good* solutions, and the deepest minimum to the best solution.

The network dynamics, obtained from $-\partial E/\partial V_{pi}$, are

$$\frac{du_{pi}}{dt} = -u_{pi} - A\sum_{q\neq p}^{K} V_{qi} - B(\sum_{q=1}^{K} V_{qi} - 1) - CR_{pi}V_{pi} + I_{pi}. \tag{9}$$

Note that (8) is only the quadratic part of the energy function corresponding to the first term in (1), and that the two terms I_{pi} and $-u_{pi}$ in (9) come from the second and third terms in (1), respectively.

To find a solution we assign random values between 0 and 1 to all the $n = K \times N$ neuronal activities, V_{pi}. Thus the N points are partitioned into K clusters. Note that the partitioning is not done in the proper sense that a point belongs to a particular cluster and to no others; rather, point i is partitioned among all the K clusters with varying strengths that are the magnitudes of V_{pi}, that is, we interpret V_{pi} as the strength of hypothesis that point i belongs to cluster p. Hence the centroid of cluster p is obtained from the weighted average

$$X_p = \sum_{i=1}^{N} x_iV_{pi} / \sum_{i=1}^{N} V_{pi}, \qquad Y_p = \sum_{i=1}^{N} y_iV_{pi} / \sum_{i=1}^{N} V_{pi}. \tag{10}$$

As the state of the network changes with time the centroids, as well as the residuals R_{pi}, also change. Starting from this randomly selected initial state the network evolves toward states of lower energy according to the equations of motion (9), until it reaches a minimum energy state and stops. The downhill motion of the network on the energy surface is guided toward a proper solution (one that satisfies the constraints) by the A- and B-terms and toward solutions of good quality by the C-term. As the network is searching for a solution the constraints are most surely violated since most neurons are partially active. Only at the end of the search when a solution is found the clustering becomes unambiguous. Note that the energy E also contains other minima that do not correspond to solutions (i.e. violate the syntax); such minima when found by the network are of course rejected as meaningless.

We remark that the cost term (7) can be written as a linear function of activities such as $R_{pi}V_{pi}$ which is *bias*-like rather than *interaction*-like. However, bias-like terms are not as effective in breaking the symmetry among the states that satisfy the syntax, and leave the energy landscape more flat. Hence it will not be as easy for the network to find valid solutions as it frequently becomes stuck in the middle of the n-dimensional unit cube. This is confirmed in our simulations, where the rate of success for finding valid solutions drops significantly when we use the linear form for the cost.

For simulations we have chosen the following values for the parameters of the energy function: $A = B = 1$, $C = 0.9/R_{avg}$, all $I_{pi} = 1$, and the gain function parameter $u_0 = 0.1$. Scaling parameter C with the average residual R_{avg} is necessary to ensure good solutions, because as the network evolves, the residuals become generally smaller and the cost term becomes less effective in driving the network toward good solutions; this rescaling of parameter C keeps the cost term of the same order of magnitude as the syntax terms.

PARALLEL IMPLEMENTATION

We have simulated the behavior of the neural net on the MPP. To do this we first generate a random initial state $\{V_{pi}(t = 0)\}$ and then solve the equations of motion (9) to find which of the minima (or solutions) it converges to. Solutions of ordinary differential equations, such as

the equations of motion, lend themselves very nicely to a massively parallel computational approach. In addition, since we want to find several solutions starting from different initial states – as is usual in computationally intractable problems – we run several trials at once on the MPP. Thus the speedup comes from parallel solution of the differential equations as well as running several trials at the same time.

We use the Euler method [12] with a fixed time step δt to solve the differential equations (9), i.e. we iterate the set of $n = K \times N$ equations,

$$u_{pi}(t+\delta t) = u_{pi}(t) + \delta t\{-u_{pi}(t) - A\sum_{q\neq p}^{K} V_{qi}(t)$$

$$-B[\sum_{q=1}^{K} V_{qi}(t) - 1] - CR_{pi}V_{pi}(t) + I_{pi}\}, \quad (11)$$

until the system converges to a stationary state. The only stopping criterion we use is when the changes in the firing rates become insignificant, i.e. when all $|V_{pi}(t+\delta t) - V_{pi}(t)| < \epsilon$, where $\epsilon \ll 1$. After the network converges to a solution, we must check if it is a valid solution that satisfies the syntax, i.e. for every point i we must have one $V_{pi} = 1$ and all the rest $V_{qi} = 0$ for $q \neq p$. In analog networks the activity of a neuron can never become exactly 0 or 1 and can only reach close to the limits. Therefore, if $V_{pi} < \eta_0$ we take $V_{pi} = 0$, and if $V_{pi} > 1 - \eta_1$ we take $V_{pi} = 1$, where η_0 and η_1 are small positive numbers. In the simulations we have chosen the following parameter values: time step $\delta t = 10^{-3}$, convergence parameter $\epsilon = 10^{-4}$, and the syntax parameters $\eta_0 = \eta_1 = 0.2$.

Mapping onto a SIMD parallel processor was accomplished by assigning a unique processing element to each data point. With this requirement, all of the necessary operations reduce to simple array arithmetic, parallel sums, row and column broadcasts, and global boolean tests. All of these are the strong points of a massively parallel processor such as the MPP. Since the MPP has 16384 processors, fewer data points allow more separate trials to be run in parallel. Thus, for example, the 128 point case allowed for 128 trials with different starting conditions to be run at the same time. The overhead to the program to keep track of the different trials is trivial since the data movement required is straightforward and controlled by the programmer. The set of data points is replicated for each trial run in parallel.

Each processor has stored in its memory its coordinate values x_i and y_i, the neuronal activities V_{pi}, input signals u_{pi}, residues R_{pi} for $p = 1, \cdots, K$, convergence indicators for each neuron, and other ancillary information. The processing begins with the calculation of the centroids of each cluster according to (10). This involves a simple array multiplication of the x_i and y_i by V_{pi} for each cluster $p = 1, \cdots, K$. This result is summed using the cascading sum technique [13] and divided by the sum of V_{pi} for each cluster. These centroids are broadcast in parallel over the remainder of the array using the MPP micrcoded broadcast primitive. This primitive, designed by Rudi Feiss (described in [14]) is very fast using only 231 cycles to broadcast a row or column – 128 32 bit numbers – to the remainder of the rows or columns of the 128×128 array. Then we calculate the residues from (5) which involves more array arithmetic. The new input signals $u_{pi}(t+\delta t)$ are calculated from (11) and the new activities $V_{pi}(t+\delta t)$ are calculated from the sigmoid function (2). These are all array arithmetic operations. A boolean mask for each cluster is created in parallel to record where the new activities are different from the old activities by more than the convergence parameter ϵ. A logical 'or' (implemented as the ANY function in MPP Pascal) on the masks determines whether the convergence criteria has been met for all activities. This logical 'or' directly translates into a hardware instruction on the MPP and thus allows simultaneous checking of conditions which on a serial processor would have to be done individually. Updating of all neurons for each trial was continued, regardless of whether a particular trial had converged, until all trials had converged. Thus unnecessary bookkeeping time is eliminated.

Thus the speed on the MPP is obtained from, (i) the mapping which allows most operations to be formulated in terms of array arithmetic, (ii) the movement of data among the processing elements which can be done with parallel algorithms, and (iii) the global boolean tests which are done by the machine hardware. For the case of 128 points to be clustered into 5 clusters, 128 trials were run simultaneously. This required 19 seconds per 500 iterations. The corresponding CPU time on a VAX 8800 was 2940 seconds (a speedup of over 150 times), and 21100 seconds on a VAX 11/780 (a speedup of about 1100 times).

EXAMPLES

To study the performance of the neural net we have tested it on some examples. In the first data set, there are 128 points divided among 5 clusters with within-cluster Gaussian distributions (Fig. 1a). Here the 5 clusters are rather well defined and out of the 128 trials the neural net found the optimum clusters 128 times. The average number of iterations for convergence was 4263; since $\delta t = 10^{-3}\tau$, the average convergence time is about 4.3τ, where τ is the decay time of a neuron. In VLSI implementations of neural networks that are currently in progress [1], the decay time of neurons, τ, is in the range $10^{-6} - 10^{-3}$ second, hence the convergence time of the network should be in the range of a few micro-seconds to a few milli-seconds. Note that from numerical solution of

differential equations one can only obtain an estimate of the actual convergence time, because the number of iterations for convergence depends on the value of the convergence parameter as well as the time step. Obviously if the convergence parameter is made smaller it will take more iterations for the network to meet the convergence criterion, resulting in a higher estimate for the convergence time. On the other hand if the time step is made smaller by, say, a factor of 10, it will take fewer than 10 times the number of iterations to converge, thus resulting in a lower estimate for the convergence time. Fig. 2 shows in more detail the number of iterations for the convergence of all the 128 trials.

The conventional method of Forgy [4] in 128 trials found the best clusters only 46 times and various other solutions 82 times. The average number of iterations for convergence was 7. Clearly, in this example, the neural net outperforms the conventional method, in that it finds the best solution much more frequently. On the other hand, the conventional method takes far fewer iterations to converge than the neural net. But we should bear in mind that these are *simulations* of the neural net, and that the number of iterations needed for convergence is not the true measure of the processing time of the network. The convergence time of an actual analog VLSI network must be measured in τ, the characteristic time of a neuron, which is in the micro to milli-second range.

To test the performance of the network in cases where clusters are fuzzy, we started from the data points of Fig. 1a, randomly selected 10% of the points and distributed them uniformly throughout the unit square (Fig. 1b). Thus we obtained 5 clusters with uniform background noise. The neural net in 128 trials found the best clusters 28 times. It failed to find valid solutions statisfying the syntax 46 times. This large number of failed solutions can be interpreted as an indication that the clusters are fuzzy, that there are outliers, and that perhaps the specified number of clusters, $K = 5$, is too few. However, even when the syntax is not satisfied we can extract a valid solution with the following scheme. For each point i set the largest V_{pi} to 1 and all the other V_{qi} with $q \neq p$ to 0, and interpret this solution as the one favored by the network, thus we obtain 128 solutions. Conventional algorithms always find valid solutions and cannot give an objective indication of the fuzziness of clusters.

Similarly to Fig. 1b, we generated other data sets by increasing the background noise to 25%, 50%, 75%, and 100% (i.e. no clusters). These data are shown in Fig. 1c-f. The results of partitioning the data among 5 clusters obtained, in 128 trials, with the neural net and with Forgy's method are listed in Table 2. The average estimated convergence times for the network are given in units of τ. Two points of note in this table are: (i) As the 5 clusters become less discernible the network increasingly fails to satisfy the syntax indicating that clusters are fuzzy and that 5 clusters are not sufficient. The conventional method, on the other hand, always finds valid solutions, and although the variety of solutions that it finds increases (this is true in both methods) which may be taken as a clue to the fuzziness of clusters it is not as objective an indicator as the failure to satisfy the syntax; (ii) When there are well defined clusters the neural net performs better than the conventional techniques which is reflected in the lower average χ^2 (χ^2 is the sum of within-cluster variances) for solutions found by the neural net. And as clusters become fuzzier the quality of solutions found by both methods become comparable.

Table 2: In this table the results obtained by Forgy's conventional algorithm are compared with those by the neural network. The Data refer to data points of Fig. 1a-f. These are based on 128 trials.

Data	Conventional			
	Iter	Best Var	Best%	Avg Var
a	7	0.62	36	1.23
b	8	1.06	34	1.57
c	8	1.95	12	2.27
d	10	2.94	2	3.14
e	10	3.88	10	4.11
f	10	4.13	2	4.64

Data	Neural Net				
	Time	Best Var	Best%	Avg Var	Synt%
a	4	0.62	100	0.62	100
b	7	1.06	22	1.24	64
c	7	1.95	19	2.03	9
d	8	3.00	15	3.04	0
e	6	3.89	1	4.11	1
f	8	4.46	2	4.71	0

Iter: is the average number of iterations for convergence.
Best Var: is the variance of the best solution found.
Best%: is the percentage of trials that found the best solution.
Avg Var: is the average variance of the solutions found.
Time: is the average estimated time of convergence in units of τ.
Synt%: is the percentage of trials that found solutions satisfying the syntax.

In Fig. 3, we have plotted the trajectories of the centroids of the 5 clusters as a function of time for all the 128 trials for the data of Fig. 1a. It can be seen that although the centroids start from different places in different trials, they all eventually converge to the same 5 points which are the true centroids of the 5 clusters. This clearly shows

that the network succeeds, in every trial, in finding the structure in the data. In Fig. 4, we have plotted the centroid trajectories for the data of Fig. 1f. The spreading of trajetories (as contrasted to the contraction of trajectories in Fig. 3) of different trials, shows that where there is no underlying structure in the data, the network does not prefer any particular clustering and hence finds many different solutions.

CONCLUDING REMARKS

Preliminary results for clustering with neural networks are promising. The neural net appears to outperform conventional iterative techniques, when there are well defined clusters since it finds better solutions more frequently. And when clusters are fuzzy, or when the number of clusters we specify is not compatible with the structure of data, the neural net indicates that it cannot find valid solutions easily, and that something may be wrong. This indicator is an objective measure and hence more reliable than the user supplied bounds and tolerances for conventional techniques. Work on larger data sets is in progress.

The clustering criterion we have used in this paper, that is minimum sum of within-cluster variances, results in convex compact clusters. Often clusters are not round or compact. By adding to the energy function, appropriate terms that favor closeness of a point to its neighbors (and not just to the cluster centroid), one can design a network that finds non-convex elongated clusters of various shapes.

Simulations of the neural net on the MPP for the clustering problem are two to three orders of magnitude faster than simulations on serial machines such as the VAX 8800 and VAX 11/780. The speedup is due to parallel solution of the differential equations that govern the behavior of the network, as well as running several trials at the same time. However, the real benefit of neural nets may lie in the future when they can be mapped on analog chips. There are forecasts that analog VLSI neural nets will become available in several years [1]. These devices will have processing times in the micro to milli-second range, making their performance comensurate with human perceptual abilities.

References

[1] C. Mead, "Real-time analog computation in VLSI neural networks", in the First Annual International Neural Networks Society Meeting (Boston, 1988).

[2] W. Feller, *An Introduction to Probability Theory and Its Applications*, 2nd edition (John Wiley, 1959) Vol. 1, p. 58.

[3] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data* (Prentice Hall, 1988).

[4] E.W. Forgy, "Cluster analysis of multivariate data: efficiency versus interpretability of classifications", Biometric Soc. Meetings, Riverside, California. Abstract in *Biometrics*, **21**, 768 (1965).

[5] J.B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations", Proceedings of Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, p. 281 (1967).

[6] A.D. Gordon and J.T. Henderson, "Algorithm for Euclidean sum of squares classification", *Biometrics*, **33**, 355 (1977).

[7] D.J. McRae, "MIKCA: A FORTRAN IV iterative k-means cluster analysis program", *Behavioral Science*, **16**, 423 (1971).

[8] M.R. Anderberg, *Cluster Analysis for Applications* (Academic Press, 1973).

[9] J.J. Hopfield and D.W. Tank, "Neural computation of decisions in optimization problems", *Biological Cybernetics*, **52**, 141 (1985).

[10] *Neural Networks for Computing*, edited by J.S. Denker (American Institute of Physics, 1986).

[11] B. Kamgar-Parsi and B. Kamgar-Parsi, "An efficient model of neural networks for optimization", in Proceedings of the IEEE First International Conference on Neural Networks, edited by M. Caudill and C. Butler, Vol.3, p. 785 (1987).

[12] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations* (Prentice-Hall, 1971).

[13] H.S. Stone, "Problems of Parallel Computation", in *Complexity of Sequential and Parallel Numerical Algorithms*, edited by J.F. Traub (Academic Press, 1973).

[14] J.E. Devaney, "The MPP – a Totally Different Approach to Programming", presented at the IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management (1985).

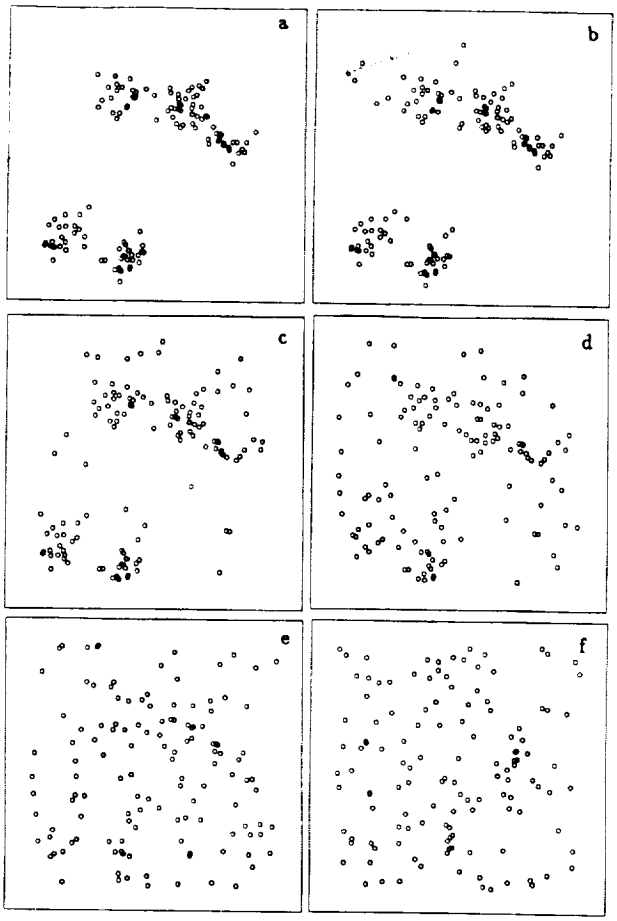

Fig. 1. 128 points divided among 5 clusters and respectively 0,10,25,50,75,100 % uniform background in a,b,c,d,e,f.

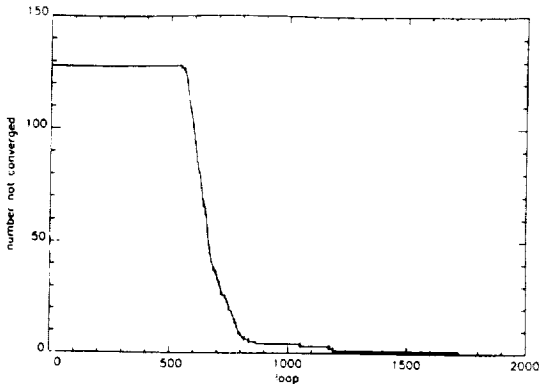

Fig. 2. Number of trials not converged versus iteration for the data in Fig. 1a. (0 % background) (loop is the iteration number).

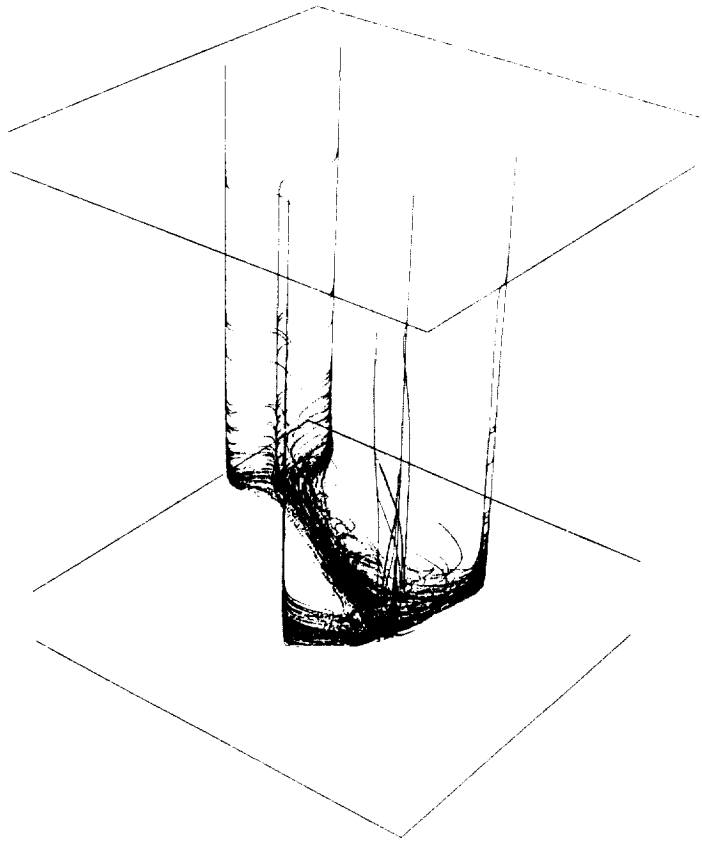

Fig. 3. Trajectories of the five cluster centroids for all 128 trials for the data in Fig. 1a. (0 % background). Lower left corner of Fig. 1a. corresponds to back top corner in this figure

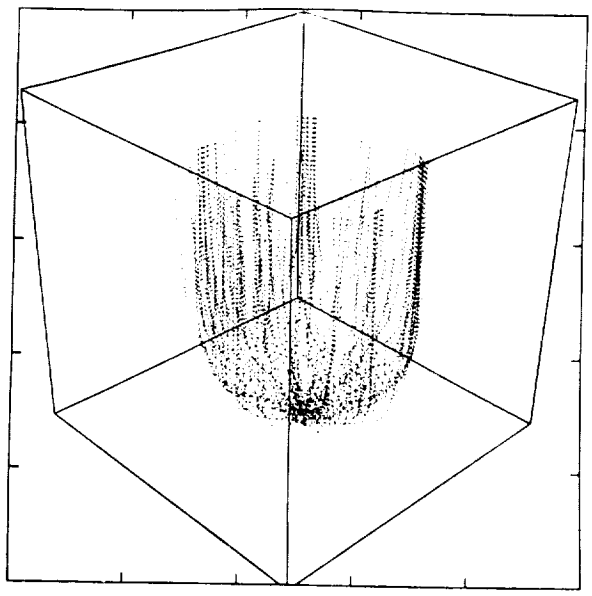

Fig. 4. Trajectories of the five cluster centroids for the data in Fig. 1f. (uniform distribution - 100% background).