TOWARDS A SIMPLE AND FAST LEARNING AND CLASSIFICATION SYSTEM

Thesis by

Yiu-fai Isaac Wong

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1992

(Defended January 8, 1992)

To my parents and my brothers

The unconquerable will.
JOHN MILTON, *Paradise Lost*

Genius is one per cent inspiration and ninty-nine per cent perspiration.
THOMAS EDISON

Acknowledgments

It is my greatest pleasure to thank my thesis advisor Edward C. Posner for the good things he has done for me. Without his encouragement, advice and guidance, this work would have been impossible. His willingness to put students' interests first is deeply appreciated.

I also thank in chronological order Professors James Bower, Christof Koch, Geoffrey Fox and Athanasios Sideris who had in the past served as my advisors. The broad exposure has been very valuable. My gratitude also goes to Eitan Gurewitz who, during his sabbatical leave at Caltech, had given me much advice. Professor Sunny Chan has been a source of inspiration for me. I admire, among other things, his ability in administration and his care and devotion to the students. Professor John Doyle has provided mathematical software which made simulations easy. I am also grateful to have Joel Burdick, Charles Elachi, Carver Mead and Steve Wiggins on my dissertation committee.

Among the friends, it is very difficult to acknowledge each individually for the friendship and the support. I can only sample at a very low rate and hopefully the aliasing effect is not visually disturbing. During my undergraduate years at Caltech, a friend named Bao Nguyen and I had spent numerous hours together struggling with the homework and sharing the frustration about lost dreams for greatness. I thank my friends Ming-chung Chu and Guo-fu Tan for demonstrating to me that working hard can be fun sometimes. Benson Chan and the group of Hong Kong students here at Caltech have always provided a good source of entertainment. The experience of being an officer for the Caltech C had served a nice channel for tunneling my energy during the trial times in my academic life and the tragic events in China. I had the pleasure to meet friends like Weimin Lu, Allen Su and Liping Yan. Amir Atiya has been helpful in getting needed references throughout my graduate studies. Within the Systems Group, many friends have been quite supportive. In particular, Petros Mouchtaris and I have shared, among other things, the excitement of hitting a home run in research. Chats with Andy Moore have always been a pleasure. Bette, our secretary, has been very helpful on things from decoding Posner's handwriting to faxing. Thanks should also go to Magic Johnson and the Lakers for providing Showtime entertainment on the court, which has inspired numerous late-night basketball workouts on a dimly lighted court with my friends.

Finally, I am deeply indebted to my parents who sacrificed their good careers in China so that the children were able to go to Hong Kong. As the youngest son in a Chinese family, I have certainly enjoyed the best of everything from my parents, and from my two elder brothers as well. For that I am grateful. My niece Lin-lin who has been staying with me since last July has provided numerous occasions for laughter and joyful moments by being the cutest toddler in the world. My wife-to-be Kit Li has been very supportive during this last period as I am finishing up my work.

Abstract

This work consists of two parts which can be read independently.

The first part contains a novel proof of the learning convergence in the Cerebral Model Articulation Controller (CMAC) proposed by Albus in 1976. That CMAC can learn any discrete input-output mapping was not known. Our work presents two ways of looking at the learning algorithm in CMAC. The learning algorithm is formulated as a matrix iteration scheme, the convergence of which can be proved by a) standard matrix theory and b) Fourier analysis. Each approach offers unique insights about the nature of the learning mechanism in CMAC. The analysis provides mathematical rigor and structure for a neural network learning model with simple and intuitive mechanisms.

The second part presents a new clustering algorithm derived from an interdisciplinary approach. The original motivation came from studies in Part I. The new algorithm departs from traditional approaches in many ways. It is the only algorithm which incorporates scale, though scale has been recognized by other researchers. It also introduces a new concept into clustering: cluster independence, which proves essential. The new framework allows us to derive a formulation based on information theory and statistical mechanics. The cluster centers correspond to the local minima of the thermodynamical free energy, which are identified as the fixed points of a one-parameter nonlinear map. Bifurcation techniques are used to obtain a complete picture of the dynamics of the map. A new clustering algorithm based on the melting process is obtained, which is hierarchical and unsupervised. Melting produces a tree of clusters in the scale space, analogous to a dendrogram. A characterization of "cluster" is given. Robustness considerations in scale space lead to a natural way of determining the optimal number of clusters. The algorithm is also insensitive to variability in cluster densities, cluster sizes and ellipsoidal shapes and orientations. We tested the algorithm successfully on both simulated data and a multi-dimensional Synthetic Aperture Radar image of an agricultural site for crop identification, and found that it beat the competition. Our clustering algorithm may also provide new and important insights for neural network research and optimization theory.

Contents

List of Figures

Chapter 1

Analysis of Learning in a Simple Neural Network (CMAC)

1.1 Introduction

An important application of neural networks lies in system identification. The goal here is to construct a neural network which can closely reproduce the input-output mapping of the real "plant" one is trying to model. We investigate a class of neural network models which utilize neurons with local responses. Such networks have been called *radial basis function networks* (RBF networks) [1, 2] and they are quite popular as a viable alternative to the traditional multilayer sigmoidal networks [3]. We will only present the analysis on the cerebellar model articulation controller (CMAC), proposed by J.S. Albus [5, 4] in 1975. CMAC is the precursor to the present-day RBF networks.

CMAC is a three-layer feedforward network for learning input-output mappings. It is capable of very fast learning and shares some common features with interpolation and approximation. However, this model has often been overlooked by the neural network research community; and it has been traditional to regard it as a look-up-table. But the ideas behind CMAC can be found in Kanerva associative memory [6] and some learning schemes using neurons with local receptive fields [7, 8]. Recently, we do see some growing interest in CMAC. Several researchers have applied this model of neural network to solve different problems. For example, John Moody [8] has used a modified form of CMAC for predicting chaotic time series. Michael Hormel [9] has applied Kohonen-type algorithms to adapting the storage mechanisms of CMAC to match the input distribution. M. J. Carter and others [10] have investigated the fault-tolerance of CMAC networks. Kraft and Campagna [11] have done a comparison between CMAC-based controller and two traditional adaptive controllers. Thomas Miller has had extensive experience in using CMAC for real-time control of robots with vision [12, 13].

In this work we will present a brief review of CMAC. We then show the nature of learning in the system and prove the key result that the CMAC learning algorithm always converges, using both matrix theory and Fourier analysis. We show that CMAC is capable of learning any discrete input-output mapping and the learning is governed by a single parameter.

1.2 Mechanisms of CMAC

A schematic sketch of CMAC is shown in Figure 1.1. We will next describe briefly the building blocks of CMAC. Although we can cast the model in the framework of connectionist multi-layer feedforward networks, the description we adopt here is closer to the original version because conceptually both versions are similarly easy to visualize.

For simplicity, let us start with CMAC for one-dimensional mapping, say $R \mapsto R$. Suppose the input is bounded between $[a, b]$ and the interval is discretized into a fixed number of levels in natural order indexed by integers. Each input x can then be identified by its discretization level x_d, an integer. There is a mapping between the discretized input space and a set S of association cells which is also indexed by the same set of integers defined by the discretization levels. The output of an association cell is 1 if excited and 0 otherwise. For each input x, we first compute its

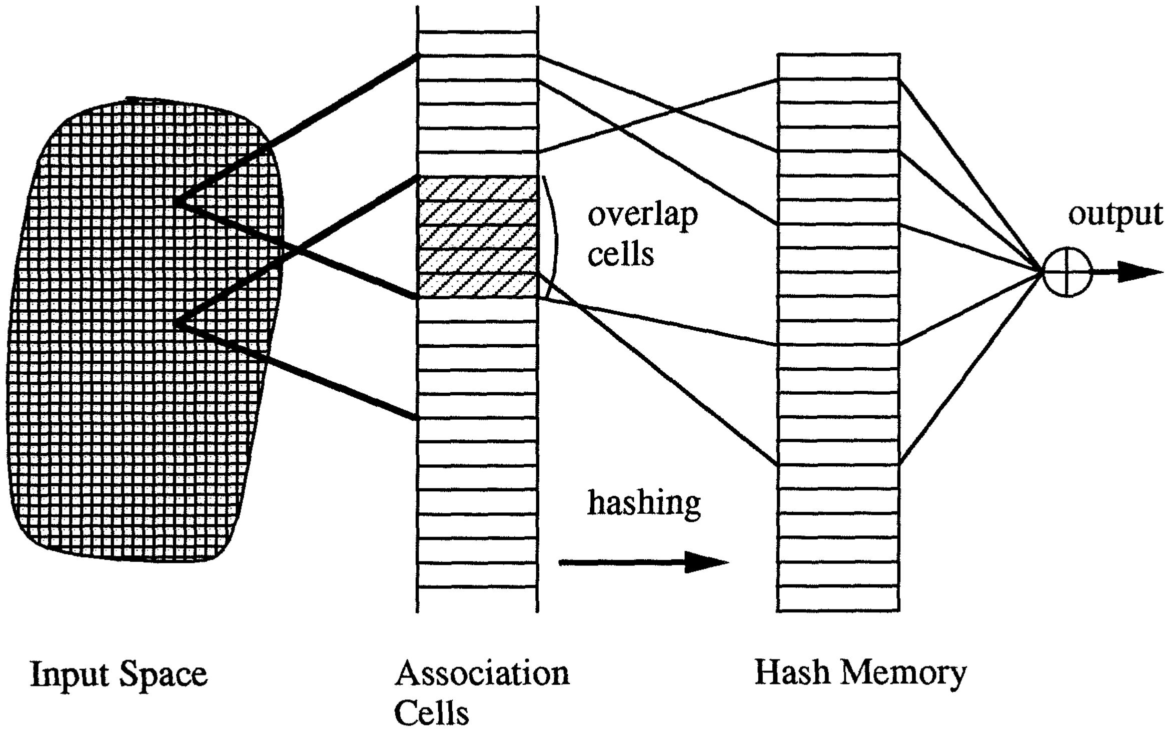


Figure 1.1: Basic skeleton of CMAC: the input excites M association cells which are coded to produce the output.

discretization level x_d. Then M consecutive association cells are excited starting at x_d. (If, instead of describing which association cells are excited by a particular input, we can also describe which inputs excite a particular association cell. That is, we can use receptive fields of the association cells to describe this mapping.) We then sum the weighted outputs of these cells to get the CMAC output. Note that this map induces a metric on $\mathcal{S}$ such that nearby inputs excite some common association cells.

If the discretization is very fine, there will be too many association cells so that it is physically impossible to implement them (this is especially true for multi-dimensional inputs). Albus solved this problem by hash coding the set of association cells into a smaller, manageable memory $\mathcal{A}_p$ with A_p memory locations. Each location is associated with a weight. Therefore, the M association cells are mapped to M addresses in $\mathcal{A}_p$ by hash coding. We then sum the weights from these addresses to produce the output. Because of the hashing procedure, in the following, we will sometimes

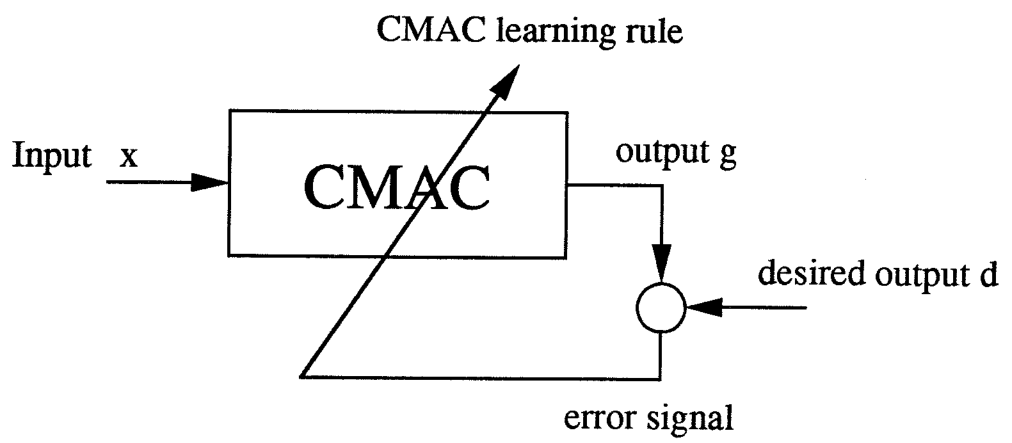


Figure 1.2: System diagram for training CMAC. The error signal is distributed evenly among the excited neurons.

identify association cells with addresses and cell outputs with weights.

Collision occurs when two different association cells are mapped to the same address. The famous "birthday" problem [14] reminds us that collisions are bound to occur. For example, suppose we want to learn a function of one variable on some bounded interval which is discretized such that 300 input-output pairs are selected for training. Suppose further that each input x excites $M = 31$ cells, and the average overlap between the association cells that the neighboring inputs excite is 27. This is because we train CMAC at some sampled points along the graph and hope or assume that CMAC will generalize to the untrained ones. Then $1227 = 31 + 4 \times 299$ distinct cells will be excited. To calculate the probability of no collision if we have a memory of 10^6 addresses, we consider the problem of putting r balls into n cells. The probability that each cell has at most one ball is

$$(n)_r/n^r = (1 - \frac{1}{n}) \cdots (1 - \frac{r-1}{n}). \tag{1.1}$$

Therefore, for the example above, the probability of no collisions is only 0.0005.

Having described the mapping from input to output, we next describe the learning algorithm for CMAC. Given the i^{th} training sample, the desired output is d_i and the network output is $g_i = \sum w_j$ where w_j is the weight stored in memory location j and j indexes the cells which are excited. The error signal is $\delta_i = d_i - g_i$. The CMAC scheme calls for evenly distributing the error among the weights. That is, for each address excited by the input,

$$\delta w_i = \delta_i / M. \tag{1.2}$$

Note that this is one-shot error-correction. This rule can be derived from back-propagation [3] learning rule with learning rate $1/M$. This is the basic skeleton of CMAC. The reader is referred to the reference [4] for details and the case of multi-dimensional inputs.

1.3 Understanding CMAC

Although CMAC has a simple learning rule, it is capable of fast learning. This has been reported by Albus [4] and Moody [8]. It is also confirmed in the example shown in Section 1.7 where CMAC tries to learn the inverse dynamics of a two-link robot arm along a trajectory.[1] The training scheme is shown in Figure 1.2.

As pointed out by Albus, CMAC also has a proper generalization property. This can be intuitively understood from the overlapping association cells which nearby inputs excite, providing some degree of interpolation. For a new input x' which is close to some learned inputs $x_1, \ldots, x_k$, the association cells that x' excites will have some overlap with the association cells these learned inputs excite. Therefore, a natural interpolation occurs. The more the overlap, i.e. the larger M, the better generalization. But there is a price to pay for larger M. Specifically, the computational complexity will be higher and a larger $\mathcal{A}_p$ would be required so that hashing would not pose serious problems for learning convergence as we will see later on.

One natural question to ask is: Is CMAC capable of learning any mapping? Miller [12] showed that one can identify the CMAC learning rule with the least-mean-square (LMS) rule [15]. This ensures the convergence of CMAC learning to a minimum. However, proof of convergence by LMS rule requires very slow adaptation of the weights and the learning only converges in the mean-square sense. Furthermore, it is not guaranteed that CMAC will produce zero error on the training set. If

[1]This problem was picked for simulation due to some interest in applying CMAC to neural control.

the output of CMAC is not correct for any training sample, one can see that CMAC learning will never converge. To ensure convergence, one should actually make the learning rate go to zero. This is not too satisfactory because simulation results seem to suggest that CMAC learns very well using a fixed learning rate $1/M$. In fact, no one has previously done a rigorous analysis of CMAC. However, for precise analysis of CMAC, we found it useful to assume that we have a big enough memory for the association cells so that there is no need for hashing. If we do that, we can actually prove that CMAC is capable of learning any mapping. However, we will get back to the hash-coding in Section 1.5 to address the observation that the problem associated with collision in hash-coding does not seem to hinder the accuracy of the learning process. Here is the statement of our main result and its proof.

Theorem 1 *Given a set of training samples composed of input-output pairs from* $\mathcal{R}^n \mapsto \mathcal{R}^m$*, CMAC always learns the training set with arbitrary accuracy if the input space is discretized such that no two training input samples excite the same set of association cells.*

Proof: For better exposition of CMAC, we will assume $n = m = 1$ until near the end of the proof since the proof for the case of $\mathcal{R} \mapsto \mathcal{R}$ easily extends to the more general case stated in the theorem.

The i^{th} training sample has its input excite M association cells. Let x_{ij} be its j^{th} address, $j = 1, \cdots, M$. (Here we identify address with association cell.) Let $g(x_{ij})$ be the weight at that address. Then the output for the i^{th} sample is $g_i = \sum_j g(x_{ij})$. If d_i is the desired output, the error is $\delta_i = d_i - g_i$. According to the CMAC learning rule, these M weights are changed by an amount $\frac{\delta_i}{M}$, as shown in equation (1.2).

Generally there are two ways to update the weights. One can update after each epoch. One can also update after each presentation. First, we look at the latter case. But the conclusion is the same for both cases, as will be seen later. After the presentation of the k^{th} input, how is the update going to affect the outputs for the other training samples? Due to the update on the k^{th} input, the new output g_i for the i^{th} training sample is $\sum_j g(x_{ij}) + c_{ik}\delta_k/M$, where c_{ik} is the number of association cells both sample i and k address, as shown in Figure 1.1.

But let us look at this procedure in another way. We have n training samples and each of them addresses M weights. When we correct for the k^{th} training sample at the l^{th} iteration, the associated weights are changed by the same amount, namely $\delta_k^{(l)}/M$ where $\delta_k^{(l)}$ is the output error for the k^{th} training sample at the l^{th} iteration. Let us consider the accumulated output error for the k^{th} training sample, $E_k = \sum_l \delta_k^{(l)}$. The total change in the individual weights associated with k^{th} training sample due to updates on itself is $\Delta_k = E_k/M$. Let us call this Δ_k the *accumulated weight error* for k^{th} input sample. The contribution to the CMAC output of the i^{th} training sample will then be $c_{ik}\Delta_k$. When the learning converges, i.e. $\delta_k^{(l)}$ goes to zero, Δ_k will converge to a constant. If we can recover the weights w_i from the accumulated weight errors Δ_i, we see that, instead of treating the weights as variables to be learned, we can also treat the accumulated weight errors Δ_i as variables to be learned.

This is the key step in obtaining the new understanding. Indeed, from equation (1.2) we can obtain a procedure to recover the weights. For each training sample i, compute the set of association cells this input excites and add Δ_i to each of these association cells. At the end of this procedure, the correct weights have been recovered. Therefore, the convergence of the CMAC learning procedure is equivalent to the convergence of the Δ_i's. When the Δ_i's converge, the error goes to zero. The learning is then complete.

It is quite obvious that for each training sample i, if the initial weights are set to zero (see Section 1.6 for the case of nonzero initial weights), the CMAC output g_i is

$$g_i = \sum_k c_{ik}\Delta_k. \tag{1.3}$$

The goal of the learning rule is to make $g_i = d_i$. If we change g_i to d_i, the above expression is a linear system

$$C\Delta = D \tag{1.4}$$

where the ij^{th} element of C is c_{ij} and the i^{th} element of D is d_i.

A key observation now is that the CMAC updating rule is nothing more than a Gauss-Seidel iterative scheme [16] to solve the linear system, when the learning rule corrects the output error at each presentation. If, however, we instead update only after each epoch, that is, we save the changes in the weights for each training sample and update only after all the training samples have been presented, then the learning scheme is instead equivalent to Jacobi iteration [17]. We remark that the Gauss-Seidel iteration is easy to implement both in software and hardware.

It is also clear that by using the scheme proposed by Albus that, once the training set and the required resolution, i.e. the discretization level, are chosen, the matrix C is determined and symmetric; all elements are non-negative, too. So we ask whether the Gauss-Seidel method, when applied to this matrix C, will converge to the desired solution. In other words, will CMAC learning converge?

If we can prove that C is positive definite, then we are sure that the Gauss-Seidel method will converge [16]. In the following arguments we will show that an inherent property of C makes it a positive-definite matrix. Hence convergence is guaranteed.

So, what is this C matrix? As pointed out above, c_{ij} is the number of association cells both samples i and j address. We represent the addresses each input sample excites by a vector of characteristic function $\Theta_i(t)$ where

$$\Theta_i(t) = \begin{cases} 1 & \text{if } t^{th} \text{ address is excited by input sample } x_i \\ 0 & \text{otherwise.} \end{cases}$$

Let us call this vector the *indicator*.

It follows then that c_{ij} is just the correlation between the indicators for i^{th} and j^{th} training sample.

$$c_{ij} = \Theta_i * \Theta_j = \sum_t \Theta_i(t)\Theta_j(t) \tag{1.5}$$

Hence given any vector $y \neq 0$,

$$\begin{aligned} y^T C y &= \sum_{ij} y_i c_{ij} y_j \\ &= \sum_{ij} y_i \sum_t \Theta_i(t)\Theta_j(t) y_j \end{aligned}$$

$$\begin{aligned}
&= \sum_{ijt} y_i \Theta_i(t) y_j \Theta_j(t) \\
&= \sum_t \sum_i y_i \Theta_i(t) \sum_j y_j \Theta_j(t) \\
&= \sum_t \Big(\sum_i y_i \Theta_i(t) \Big)^2 \qquad (1.6) \\
&\geq 0
\end{aligned}$$

which is just the "energy" of the indicators weighted by the vector y. Hence C is positive semi-definite.

Lastly, it is easy to see that, by the construction inherent in CMAC, weighted indicators cannot vanish. For note that each indicator $\Theta_i(t)$ has a minimum t^i_{min} for which $\Theta_i(t^i_{min}) = 1$, and t^i_{min} is the discretization level of x_i. Among all these training samples, there is only one sample which has a t^j_{min} which is the minimum among all the t^i_{min}'s, namely the sample with the minimum discretization level. We have called this sample j. It follows that $\Theta_j(t)$ cannot be expressed as a linear combination of the other indicators, i.e. $y_j = 0$. Applying this argument recursively, we see that $y = 0$. This says that the sum of squares in equation (1.6) cannot vanish for any nonzero vector y. Hence C is strictly positive definite. This completes the proof that CMAC always converges for one-dimensional inputs.

For multi-dimensional inputs, i.e. n arbitrary, we have found that the proof for the positive semi-definiteness still holds. Since we do not describe CMAC with multi-dimensional input in this paper (the reader can get the details from Albus's paper [4]), we simply point out that the mapping between the input space and the set of association cells $\mathcal{S}$ induces an n-dimensional Euclidean structure on $\mathcal{S}$. Thus we can index the association cells in lexicographical order. The indicator for each input is formed by concatenating the indicators for each coordinate. The proof for strict positive definiteness easily carries over. To get multi-dimensional outputs, we simply construct m set of association cells or memories. This is the plan for the proof for the mapping $\mathcal{R}^n \mapsto \mathcal{R}^m$. □

Now we can explain the fast convergence and proper generalization based on the correlation matrix C. Let Δ^* be the solution of $C\Delta = D$ and $\gamma^{(l)} = \Delta^{(l)} - \Delta^*$. Write

C as $C = L+R$ where L is the lower diagonal part of C and R is the upper offdiagonal part of C. From matrix theory [16], Gauss-Seidel iteration gives

$$\gamma^{(l)} = (-L^{-1}R)^l \gamma^{(0)}. \tag{1.7}$$

If we further expand $\gamma^{(0)}$ in terms of the eigenvectors u_j's of $(-L^{-1}R)$, say $\gamma^{(0)} = \sum_j \rho_j u_j$, we obtain

$$\Delta^{(l)} = \sum_{j=1}^{N} \lambda_j^l \rho_j u_j \tag{1.8}$$

where λ_j are eigenvalues of $(-L^{-1}R)$, with $|\lambda_i| \leq |\lambda_j|$ for $i < j$. From [16], the positive-definiteness of matrix C guarantees that $\lambda_j < 1$ for all j. Thus, initially the training error decreases very fast due to the first few terms that are associated with the small eigenvalues. But the overall convergence is asymptotically proportional to λ_N^l as l becomes large. Figure 1.6 shows the mean-square-error versus iterations for the example in Section 1.7. Indeed, a slower decrease in mean-square-error sets in after an initial steep decrease. Therefore, the theory is that we have convergence of the training error exponential in the number of iterations even though it can be very slow if the spectrum of $L^{-1}R$ is close to 1.

How about generalization? In one dimension, the matrix elements decrease linearly from the diagonal element and they are non-negative. Hence every row of C can be interpreted as an interpolation filter. Larger M gives more interpolation. But the price we pay is slightly more computational complexity. In Section 1.4, we will see that the convergence is slower. At worst, the marginal positive-definiteness of the correlation matrix C, when subject to hash coding, might lead to singularity as explained in Section 1.5.

Before we move on to apply the Fourier analysis to the Gauss-Seidel iteration of the linear system $C\Delta = D$, let us summarize the results obtained so far. Form the matrix $A = [\Theta_1(t)\ \Theta_2(t) \cdots \Theta_N(t)]^t$ where N is the number of training samples, $\Theta_i(t)$ is the indicator for input sample x_i and the superscript $(\cdot)^t$ denotes the transpose operation. We see that the matrix C can be written as $C = AA^t$. Let W be a vector denoting the output of the association cells, or the weights in the smaller memory

$\mathcal{A}_p$. The goal of CMAC learning is to find a set of weights such that

$$AW = D \tag{1.9}$$

where D is the vector of the desired outputs. The linear system $C\Delta = D$ is just $AA^t\Delta = D$. After Δ has converged, the procedure to restore the weights W corresponds to the matrix operation $A^t\Delta$. Thus, putting all the equations together, the weights are given by $W = A^t(AA^t)^{-1}D$. Note that $A^t(AA^t)^{-1}$ is the pseudo-inverse $A^+$of A [19]. It is quite remarkable that Albus's intuitive scheme is actually a numerical implementation of this inversion procedure.

We should also note that the linear system in equation (1.9) is underspecified. That is, it has more unknowns (weights) than equations (one equation for each training sample). Therefore, there are an infinite number of solutions. Any vector of weights which lies in the null space $\mathcal{N} = ker(A)$ of A will produce zero output. Therefore, we expect that a different set of initial weights may lead to a different set of weights W. We will now try to characterize this null space.

1.4 Fourier Analysis of the Convergence

To make the Fourier analysis approach more manageable, we restrict the input space to be one-dimensional. Let N be a number such that all the training samples have their discretization levels between 0 and N. Since we only present training samples x_i for some selected points on the interval, we have the known values of d_i and the unknown values of Δ_i^* for these i's. The rest of Δ_i^* and d_i are don't cares. Hence it does not matter what their true values are[2]. The number of association cells needed is therefore $N + M - 1$ and the dimension of matrix A is $N \times (N + M - 1)$. From the proof of the positive-definiteness of the C matrix, we see that the dimension of the null space $\mathcal{N}$ is $M - 1$. It is not difficult to see that $\mathcal{N}$ consists of all sinusoids of frequency $2\pi k/M$ where $k = 1, \ldots, M - 1$. This is because a sum of M consecutive

[2]It matters when we consider the generalization of CMAC to untrained points, which we do not consider here.

weights will produce a zero output. Note that these frequencies are also the zeros of the spectrum of a rectangular box of length M.

The Gauss-Seidel relaxation takes the form

$$\Delta_i^{(l+1)} = \Delta_i^{(l)} + \frac{1}{M}(d_i - \sum_{j<i} C_{ij}\Delta_j^{(l+1)} - \sum_{j\geq i} C_{ij}\Delta_j^{(l)}) \tag{1.10}$$

where

$$C_{ij} = \begin{cases} M - |i-j| & \text{if } |i-j| < M \\ 0 & \text{otherwise.} \end{cases}$$

This system can be viewed as a linear shift-invariant system on an infinite grid. It can therefore be analyzed in the Fourier domain. That is, techniques from digital signal processing can be used to analyze this numerical problem [23], if we assume that we present the inputs in a sequential order.

To take care of the boundary conditions when we replicate the interval over the infinite grid, the index i now runs from $-(M-1)$ to $N+M-2$. As we said earlier, d_i and Δ_i^* are don't cares if the input sample for this i is missing. Let $e_i^{(l)} = \Delta_i^{(l)} - \Delta_i^*$ and $e^{(l)}$ be the vector of $e_i^{(l)}$, where Δ_i^* is one of the desired values. (Note that because of the don't-cares in Δ_i, Δ^* is not unique.) From equation (1.10), we find that the errors evolve according to

$$e_i^{(l+1)} = e_i^{(l)} - \frac{1}{M}\Big(\sum_{j<i} C_{ij}e_j^{(l+1)} + \sum_{j\geq i} C_{ij}e_j^{(l)}\Big). \tag{1.11}$$

Let $N_1 = N + 2M - 2$ and expand e in a Fourier series of the form

$$e_n = \sum_{k=0}^{N_1-1} \tilde{e}_k \exp(j\frac{2\pi}{N_1}kn). \tag{1.12}$$

Then in the Fourier domain, equation (1.11) becomes [23]

$$\tilde{e}_k^{(l+1)} = H(e^{j\frac{2\pi}{N_1}k})\tilde{e}_k^{(l)} \tag{1.13}$$

where

$$H(z) = \frac{-\sum_{n=1}^{M}(M-n)z^n}{M + \sum_{n=1}^{M}(M-n)z^{-n}}. \tag{1.14}$$

The weights are obtained by the operation $W = A^t\Delta$. Thus if we denote $W^{l+1} - W^*$ by $\delta W^{(l)}$, then $\delta W^{(l)} = A^t e^{(l)}$. It is clear that in the frequency domain, $\delta\tilde{W}^{(l+1)} = H(e^{j\omega})\delta\tilde{W}^{(l)}$. For convergence, we desire that $|H(e^{j\omega})| < 1$.

Note that $H(z)$ can be written as:

$$H(z) = \frac{-h(z)}{1 + h(-z)} \tag{1.15}$$

where $h(z) = \frac{(M-1)z - Mz^2 + z^{M+1}}{M(1-z)^2}$. With $h(z) = a(z) + jb(z)$, the magnitude of $H(z)$ is given by

$$|H(z)|^2 = \frac{1}{1 + \frac{1+2a(z)}{a(z)^2 + b(z)^2}}. \tag{1.16}$$

It is obvious that $|H(z)| < 1$ if and only if

$$1 + 2a(z) > 0. \tag{1.17}$$

Equating the real and imaginary parts of $H(e^{j\omega})$, we find that $a(e^{j\omega}) = \frac{\sin^2 \frac{M\omega}{2}}{2M\sin^2\frac{\omega}{2}} - \frac{1}{2}$ and $b(e^{j\omega}) = \frac{M\sin\omega - \sin M\omega}{4M\sin^2\frac{\omega}{2}}$. Thus,

$$1 + 2a(e^{j\omega}) = \frac{\sin^2(\frac{M\omega}{2})}{M\sin^2(\frac{\omega}{2})}. \tag{1.18}$$

It is seen that $1 + 2a(e^{j\omega}) > 0$ except at $\omega = 2\pi k/M, k = 1, \ldots, M-1$, where it vanishes. Therefore, the errors cannot be reduced at those $M - 1$ frequencies. But these errors have no effect on the CMAC output. Thus W and Δ do converge. This proves that the Gauss-Seidel iteration converges in the frequency domain. It should come as no surprise that these errors have the same dimension as the null space of the matrix A. The Fourier analysis obviously generalizes to the CMAC with multi-dimensional inputs since the kernel $\exp(-jkx)$ separates.

Now, let us investigate the rate of convergence for the different frequencies by utilizing the properties of the filter $H(z)$. After considerable algebraic manipulations, we get

$$|H(e^{j\omega})|^2 = \left(1 + \frac{\frac{4}{M}\sin^2(\frac{M\omega}{2})}{(1-p(\omega))^2 + 2p(\omega)(1 - \cos(\frac{M-1}{2})\omega)}\right)^{-1} \tag{1.19}$$

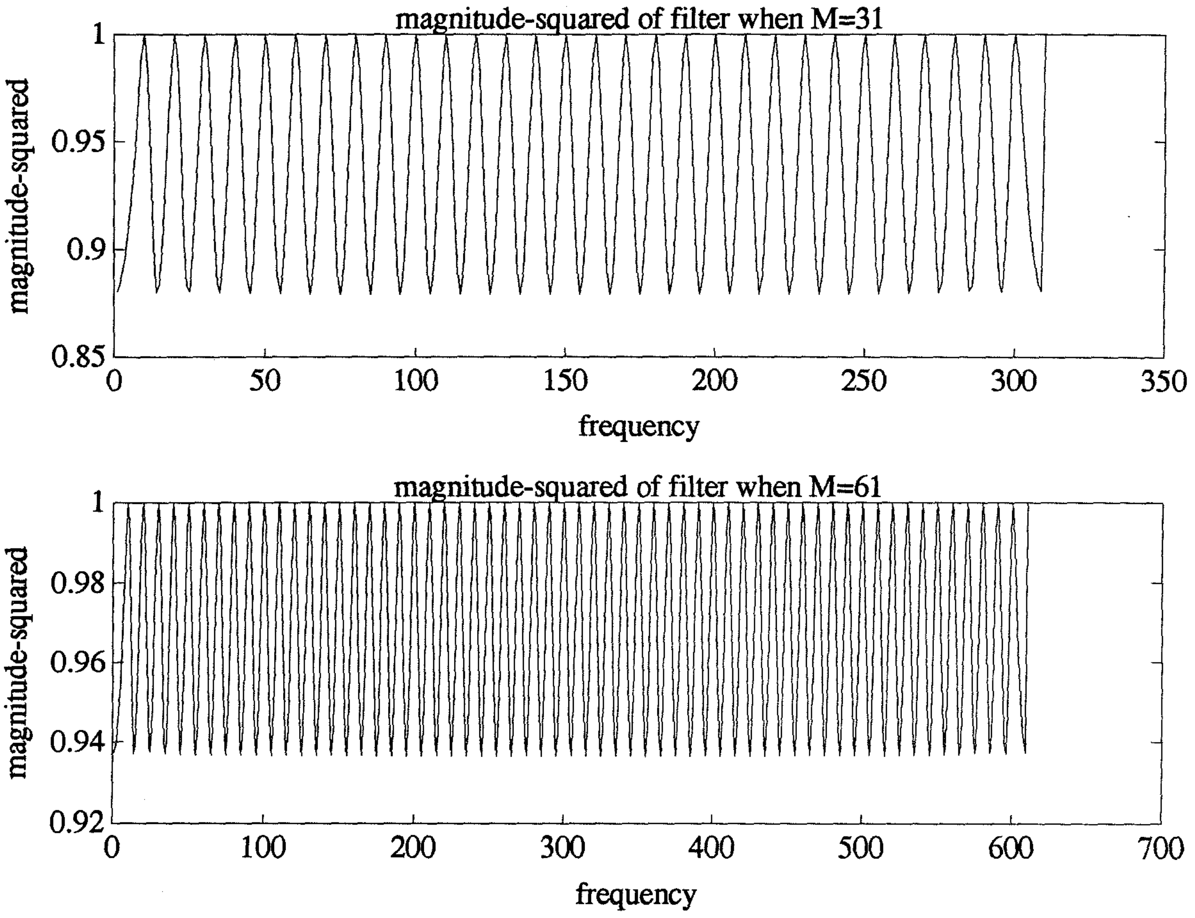


Figure 1.3: Comparison of the filters for two choices of M. It shows that the convergence is governed by M.

where $p(\omega) = \sin(\frac{M\omega}{2})/M \sin\frac{\omega}{2}$. In Figure 1.3, we plot the squared magnitude of $H(e^{j\omega})$ for two choices of M and we see that larger M means slower overall convergence, as observed in the last section. The magnitude is 1 at frequencies $2k\pi/M, k = 1, \ldots, M-1$. Hence, the convergence is slow for frequencies near the harmonics of $2\pi/M$. As M increases, there will be more of these harmonics. Note that $H(e^{j\omega})$ is periodic with period$=2\pi/M$ and its minimum value is $\frac{M/2-1}{M/2+1}$. We also note that the d-c gain of the filter is at the minimum. Thus CMAC can learn smooth mappings quite effortlessly. This phenomenon is observed in the simulation example in Section 1.8.

1.5 Hash Coding

Finally, we have to cope with hash coding since in higher dimensions, the number of association cells gets prohibitively large. If there is no collision, everything is fine. If collision does occur, it corresponds to perturbations to the matrix c_{ij}.

Let us pause for a minute to think about the operations here. First, it is very unlikely that collision occurs when in mapping the M association cells any particular input excites to the smaller memory $\mathcal{A}_p$, as noted from equation (1.1). One possible strategy is to divide the $\mathcal{A}_p$ into M submemories and hash the k^{th} component of the M addresses for all inputs into the k^{th} submemory, where $k = 1, \cdots, M$. So we can assume that this type of collision does not happen. What actually happens is that different association cells get mapped to the same address. Now, if $\Theta_i'(t)$ denotes the indicator for input x_i in the smaller memory $\mathcal{A}_p$, it is composed of M pulses of length 1 scattered around. Since the hash coding is certainly a deterministic algorithm, the same association cells are mapped to the same addresses. So the new correlation matrix C' is still symmetric, non-negative, and positive semi-definite, with $c_{ij}' \geq c_{ij}$.

To better illustrate the nature of this perturbation, we can consider the problem of putting s balls into n cells where r cells are already occupied. The probability of a ball hitting one of these r balls is $p = r/n$. Let $q = 1 - p$. Then the probability for k collisions is $\binom{s}{k} p^k q^{s-k}$. A simple numerical example will show that the probability for more than one collision is exceedingly small. For instance, if $n = 10^6, r = 31, s = 31$, Prob(0 collision)$= 0.99904$; Prob(1 collision)$= 0.00096$; Prob(2 collisions)$= 4 \times 10^{-7}$. For neighboring inputs, the number of balls s is small and the probability of having to add 1 to c_{ij} is extremely small; for inputs far apart, $s = M$, this probability is higher but still very small.

Therefore, we see that the perturbation can be regarded as a random but symmetric matrix with integer elements m_{ij} where m_{ij}'s are random variables with distribution $P(m_{ij} = k) = \binom{s}{k} p^k q^{s-k}$. Here s is a number between 0 and M, depending on how close the input samples x_i and x_j are to each other. Larger M means more perturbation to the original matrix. Consequently, we see that if we assume that the hashing function is random enough, in the hashing part only two parameters matter: A_p and M. We can always choose A_p and M such that the probability of zero collisions for any two non-overlapping subsets of excited association cells approaches 1. In that case, most m_{ij}'s are 0; rarely will m_{ij} be greater than 1. To a first order approximation, that is, if we assume that no collision occurs when in mapping the M cells any particular input excites to $\mathcal{A}_p$, then the perturbation is of the nature we described above.

What really causes trouble is that the new $\Theta_i'(t)$'s can become linearly dependent, in which case it is possible to choose y such that equation (1.6) vanishes. The matrix C' is no longer positive definite, which means that the Gauss-Seidel iterative scheme could not converge. The linear dependence of the indicators $\Theta_i'(t)$ is an example of generalized "ghosts," as noted by the authors in [18], which can create problems when each memory is encoded by a subpopulation of neurons. The analysis of the random matrix, we believe, will be closely related to the capacity of CMAC with hash coding. We have not actually conducted an analysis to address this question. But based on the simulation results with different A_p, CMAC is very resistant to collisions.

1.6 Robustness Against Noise in Learning

Lastly, we address the question of robustness of CMAC to noise added to the weights in the learning process. Recall, from the end of Section 1.3, the desired weight vector can be written as

$$W = A^t(AA^t)^{-1}D. \tag{1.20}$$

Let us write the initial weights as $W_0 = W_0^0 + W_0^1$ where $W_0^0 \in ker(A)$ and $W_0^1 \in ker(A)^\perp$ where $ker(A)^\perp$ is the subpace orthogonal to $ker(A)$ in the input space. Such decomposition is uniquely defined [19]. Let $G_0 = AW_0$ denote the vector of initial output g_i^0's for the input samples. Then, after the k^{th} presentation and weight correction, the CMAC output for the i^{th} input sample is $c_{ik}\Delta_k + g_i^0$. Therefore, we must replace each element d_i of D by $d_i - g_i^0$ to arrive at the system $C\Delta = \tilde{D}$ where $\tilde{D} = D - G_0$. After the learning converges, the weights are given by

$$\begin{aligned} W &= A^+(D - G^0) + W_0 \\ &= A^+D + (W_0 - A^+AW_0) \\ &= A^+D + W_0 - W_0^1 \\ &= A^+D + W_0^0. \end{aligned}$$

We see that the final weights are $W = A^+D + W_0^0$. The norm of the weights is $||W||^2 = ||A^+D||^2 + ||W_0^0||^2$. Therefore, to get a minimum-normed-weight vector, we should initialize the weights to zero.

Based on the above results, we see that CMAC is very robust to noise in the weights. This robustness against noise is a tremendous advantage for hardware implementation, where noise will always be present. Suppose that in evaluating the error signal $\delta_i = d_i - g_i$, a noise term n_i is added. Then, after a certain number of iterations, the error correction signal δ_i is basically due only to the noise term, which is then added to the weights. Such perturbation can be broken into two components. The component which lies in $ker(A)^\perp$ will change $\tilde{D}$ a little, while the one which lies in the null space of A has no effect on the CMAC outputs for inputs that belong to the training set. If the noise is random and not biased, then convergence will be achieved, but only in the sense that the weights will asymptotically fluctuate around their no-noise values; the fluctuations will not go to zero.

One method to force the weights to converge is to let the learning rate η decrease with iteration number after a certain number of iterations that had $\eta = 1/M$. Suppose we demand that $\lim_{m\to\infty} \sum_{k=1}^{m} \eta_k = +\infty$ and $\lim_{m\to\infty} \sum_{k=1}^{m} \eta_k^2 < \infty$. Then the weights converge to the correct values in mean square [20]. That is, if we write the weight vector after k iterations as $W_k = W_k^0 + W_k^1$, where $W_k^0 \in ker(A)$ and $W_k^1 \in ker(A)^\perp$, $\lim_{k\to\infty} E[||W_k^1 - A^+D||^2] = 0$.

In the above analysis, W_k^0, the component which lies in $ker(A)^\perp$, will increase the norm of the weights. There is thus no guarantee that W_k^0 will not affect, and to what extent, the CMAC outputs when presented with inputs which do not lie in the training set but still close to some of the training samples. A detailed analysis of this question will give a better understanding of the nature of generalization in CMAC. It would also be interesting to know if small weights in CMAC might lead to a better generalization property. In backpropagation, for example, it is argued that small weights lead to better generalization [21].

1.7 A New Implementation of CMAC

We can recapitulate the results obtained in the previous section as follows:

1. CMAC learning is essentially solving a linear system with known matrix algorithm which converges; this explains why
   - CMAC learning is highly accurate and
   - CMAC converges exponentially fast.
2. CMAC is doing some kind of interpolation for inputs it has not been trained on; this gives it a certain generalization ability;
3. CMAC learning is slow for frequencies near harmonics of $2\pi/M$ and fast otherwise;
4. CMAC learning is very robust to the noise added in the learning process.

Having said that, we are finally ready to propose a new implementation for CMAC based on the insights obtained in the proof:

1. given a set S of training samples;
2. discretize the input space with fine enough resolution so that no two different inputs occupy the same discretization level;
3. pick an appropriate memory size A_p and M;

4. for each sample i, determine the amount of overlap in the addresses with sample k;

5. repeat 3 for all training samples. This establishes the matrix $C = c_{ik}$;

6. use Gauss-Seidel or better matrix algorithm to solve the linear system

$$C\Delta = D$$

   where Δ is the accumulated weight error;

7. to recover the weights, use the procedure outlined in Section 1.3, which is summarized as the operation $A^t\Delta + W_0$.

In our simulations, typical values for A_p and M are 50000 and 31 respectively. These values are chosen based on the arguments in Section 1.5. We remark that it is relatively easy to compute the c_{ij}. The number of operations is at most of $O(n^2)$ where n is the number of training samples. The matrix C is of dimension n and is very sparse. It is not hard to see how to compute the output for any given input. This is given by equation (1.3) in Section 1.3. We should also point out that the original implementation proposed by Albus has advantages of simplicity and real-time processing while our proposed scheme is more suitable for off-line learning.

1.8 A Simulation of the New Scheme

To illustrate the proof derived above, we carried out a computer simulation of the new scheme and compared the result with the original scheme. The task we set out to learn is to determine the inverse dynamics of a two-link robot arm along a given trajectory in the joint space. The trajectory was discretized into 300 points, which is shown in Figure 1.4. The torque required to control the arm so that it moves along the trajectory is computed by the dynamics equation found in [22] and is shown in Figure 1.5. This is how the training set is obtained. Next we train the CMAC with the original implementation and the new implementation using the same training set

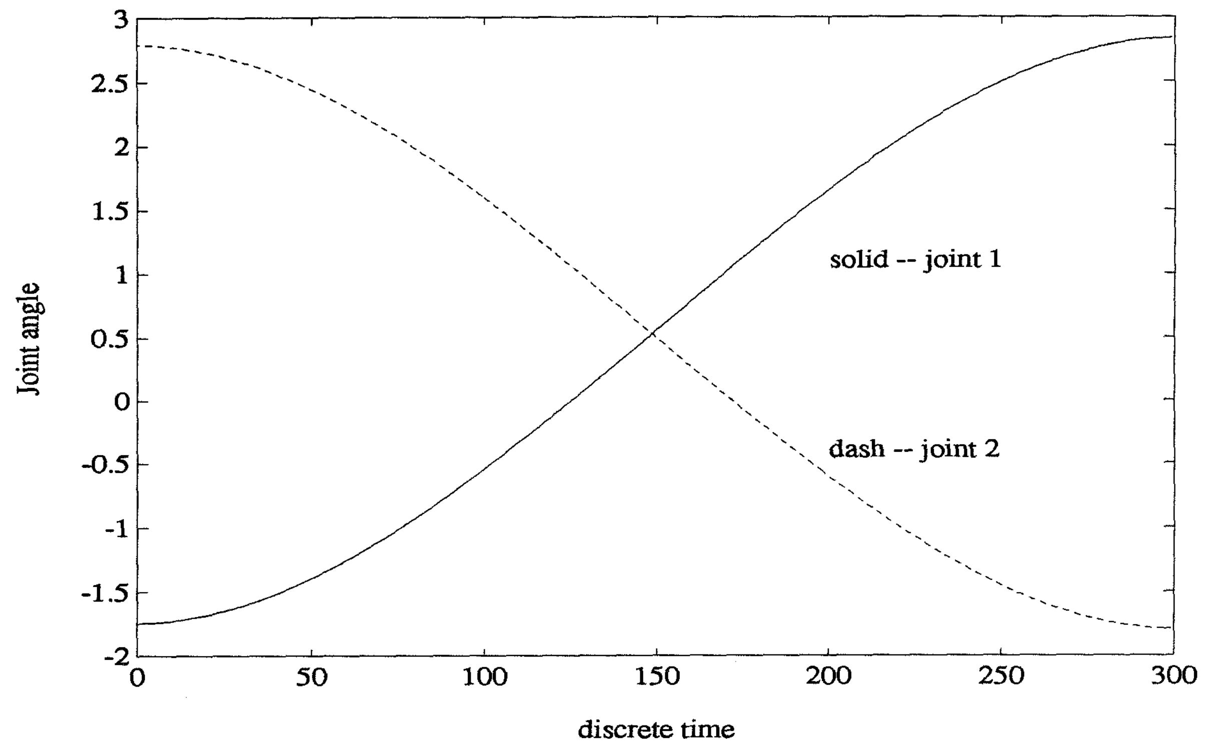


Figure 1.4: Trajectory in the joint space.

and system parameters. The only difference is in the choice of $\mathcal{A}_p$. For the original implementation, the hashing memory is chosen to be 50021 which is large enough to allow only slight collisions. In the new implementation, we assume an infinite memory, i.e. we calculate the c_{ij} in the space of association cells before hashing.

Figure 1.6 shows the mean-square-error versus each iteration (each iteration here corresponds to a full cycle over the training set). In both cases, the learning curves are indeed nearly exponential in iteration number. Note the fast decrease in the first few iterations. That is because the curve is quite smooth and convergence is very fast for these low frequencies. This confirms our theoretical predictions earlier.

Also, the collision problem due to hashing does not seem to make any difference in the learning curves. In fact we have used $A_p = 10000$ and again the curves (not shown here) are almost the same. This shows that our rather intuitive explanations in Section 1.5 are not far off from the reality and the matrix C is indeed very robust.

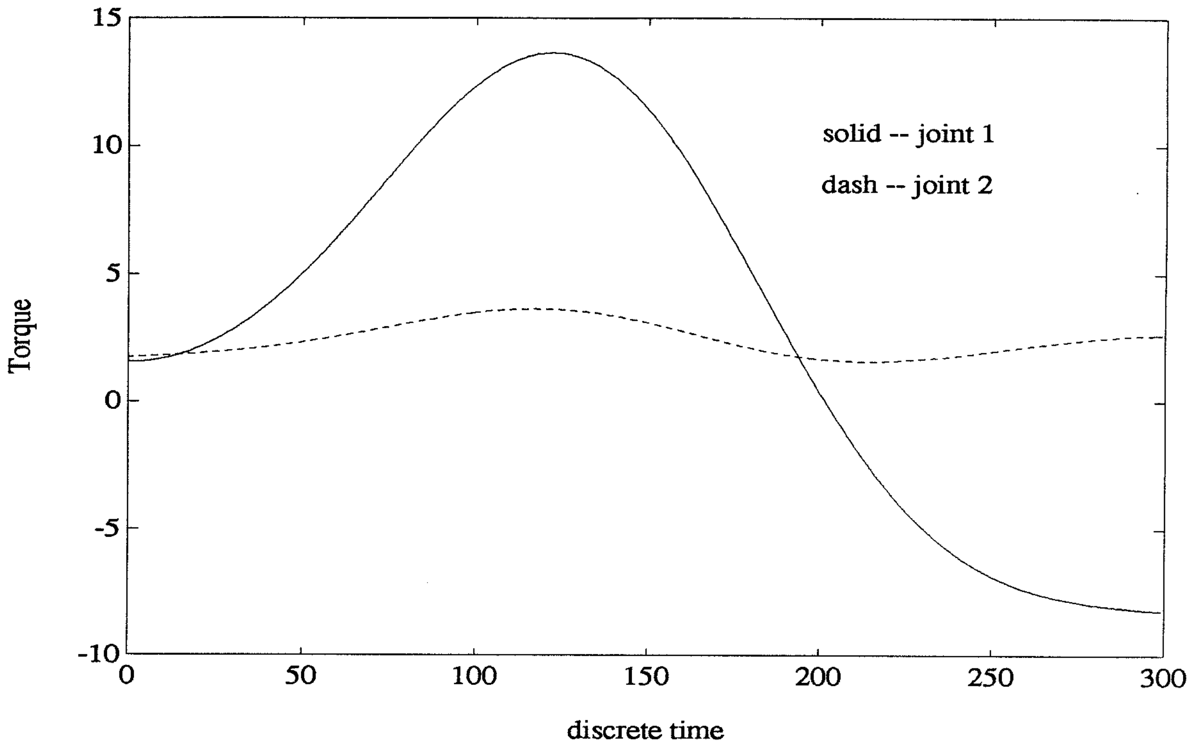


Figure 1.5: The two torques along the trajectory shown in Figure 1.4.

1.9 Summary

Using either of two new ways of looking at the learning algorithm in the original CMAC, we have presented a proof that CMAC learning always converges. We also proved that CMAC is equivalent to solving a linear system with a Gauss-Seidel iteration scheme. The good generalization property comes from the interpolation nature of the matrix. Fast learning comes from the exponentially fast convergence of the matrix algorithm. The approach using Fourier analysis gives additional insight into the learning process. It shows the convergence rates for the different frequency components of the desired solution. It also clearly shows the relationship between M, which is the size of the receptive field of the association cells, and the overall convergence rate. The results obtained give us a better understanding of CMAC. It is very satisfying that linear techniques can be utilized to rigorously analyze this simple neural network model. This is some encouragement for those who try to build mathematical

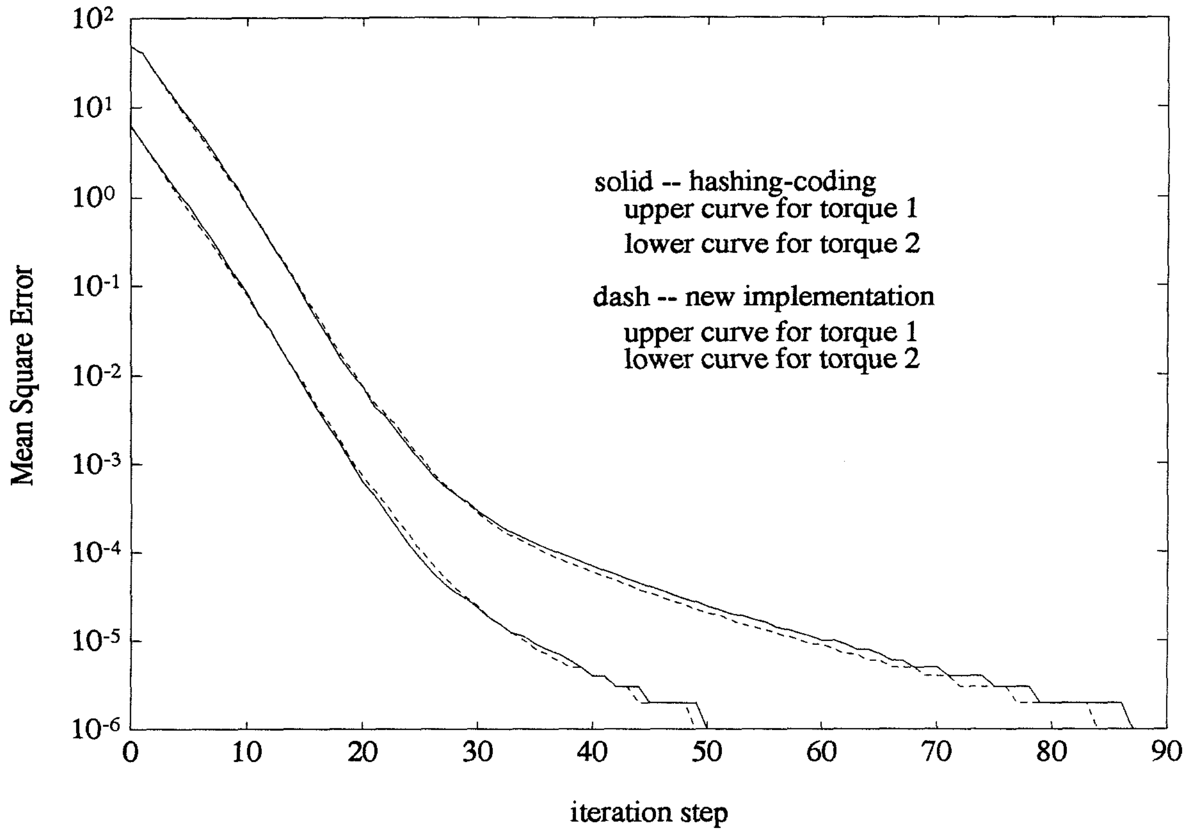


Figure 1.6: Convergence of CMAC learning and the comparison between the two schemes.

models of simple neural networks. But it remains to be seen if similar rigor can be brought to bear on more complicated neural networks.

Bibliography

[1] D.S. Broomhead and D. Lowe,"Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems*, 2, 321-355, 1988.

[2] Tomaso Poggio and Federico Girosi, "Networks for Approximation and Learning," *Proceedings of the IEEE*, vol 78, 1481-1497, 1990.

[3] D.E. Rumelhart, *et al.*, *Parallel Distributed Processing*, vol 1, MIT Press, Cambridge, Mass., 1986.

[4] J. S. Albus, "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," *Trans. ASME, J. Dynamic Syst. Meas. Contr.*, vol 97, 220-227, 1975.

[5] J. S. Albus, *Theoretical and Experimental Aspects of a Cerebellar Model*, Ph.D. dissertation, Univ. of Maryland, 1972.

[6] Pentti Kanerva, "Parallel Structures in Human and Computer Memory," in *Neural Networks for Computing*, J.S. Denker (Ed.), AIP Conf. Proc. *151*, Snowbird, 247-258, 1986.

[7] B.W. Mel and C. Koch, "Sigma-Pi Learning: On Radial Basis Functions and Cortical Associative Learning," in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (Ed.), Morgan Kaufmann Publishers, 474-481, 1989.

[8] John Moody, "Fast Learning in Multi-Resolution Hierarchies," in *Advances in Neural Information Processing Systems 1*, D.S. Touretzky (Ed.), Morgan Kaufmann Publishers, 29-39, 1989.

[9] Michael Hormel, "A Self-organizing Associative Memory System for Control Applications," in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (Ed.), Morgan Kaufmann Publishers, 332-339, 1990.

[10] M. J. Carter, F. J. Rudolph and A. J. Nucci, "Operational Fault Tolerance of CMAC Networks, " in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (Ed.), Morgan Kaufmann Publishers, 340-347, 1990.

[11] L. G. Kraft and D. P. Campagna, "A Comparison between CMAC Neural Network Control and Two Traditional Adaptive Control Systems," *IEEE Contr. Syst. Mag.*, 36-43, April 1990.

[12] W. Thomas Miller, "Real-Time Application of Neural Networks for Sensor-Based Control of Robots with Vision," *IEEE Trans. Syst. Man and Cyb.*, SMC-19, 825-831, 1989.

[13] W. Thomas Miller, F.H. Glanz and L.G. Kraft, "CMAC: An Associative Neural Network Alternative to Backpropagation," *Proc. IEEE*, vol 78, 1561-1567, 1990.

[14] William Feller, *An Introduction to Probability Theory and Its Applications*, vol. I, John Wiley & Sons, New York, 28-33, 1968.

[15] B. Widrow and S.D. Stearns, *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, N.J., 1984.

[16] Joel Franklin, *Matrix Theory*, Prentice-Hall, 221-227, 1968.

[17] Gene H. Golub and Charles F. van Loan, *Matrix Computations*, Johns-Hopkins University Press, 353-356, 1983.

[18] Ronald Rosenfeld and David Touretzky, "A Survey of Coarse-Coded Symbol Memories", *Proc. of the 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton, T. Sejnowski (Ed.), Morgan Kaufmann Publishers, Menlo Park, CA, 256-264.

[19] D.E. Catlin, *Estimation, Control, and the Discrete Kalman Filter*, Springer-Verlag, New York, 93-105, 1989.

[20] Richard O. Duda and Peter E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 155-159, 1973.

[21] Chuanyi Ji, Robert R. Snapp and Demetri Psaltis, "Generalizing Smoothness Constraints from Discrete Samples," *Neural Computation*, vol 2, 188-197, 1990.

[22] John J. Craig, *Introduction to Robotics—Mechanics and Control*, 2nd edition, Addison-Wesley Publishing Company, Reading, Massachusetts, 201-205, 1989.

[23] C.-C. Jay Kuo and Bernard C. Levy, "Discretization and Solution of Elliptic PDEs —A Digital Signal Processing Approach," *Proc. IEEE*, vol 78, 1808-1842, 1990.

Chapter 2

Clustering Data by Melting

2.1 Introduction

Clustering is an important problem which can be found in many applications where *a priori* knowledge about the distribution of the observed data is not available [1]. Simply stated, the goal is to partition a given data set into different groups such that the data points in each group are as compact as possible. Each group indicates the presence of a distinct category in the measurements. It is a tool widely used for analyzing multi-dimensional data in diverse disciplines such as biology, social sciences and astronomy. The literature is therefore spread among many different fields over many years. It is almost impossible to cite each contribution individually.

One of the early algorithms was invented by Lloyd [2] which was later extended by Linde *et al.* [3] for vector quantization. In pattern recognition, the ISODATA algorithm [4] and its sequential version, the k-means clustering algorithm, have been extensively used. These methods are called *partitioning* techniques because at any stage of the computation, each data point is associated with one and only one cluster. These algorithms suffer from two difficulties. First, one can always find a partition for any number of clusters. That is, there are no cluster validity criteria [7] to decide how many clusters are really present in the data set. Second, the final partition is highly sensitive to the initialization of the algorithms. In particular, these algorithms perform poorly if the data contains overlapping clusters.

The fuzzy ISODATA algorithm was first introduced by Ruspini [5] and later extended by Dunn and Bezdek [6, 7]. This approach allows a data point to be associated with all the clusters, but with different degrees of membership. However, fuzzy clustering still suffers the same difficulties as the partitioning algorithms do. Rose *et al.* presented a statistical mechanics formulation of clustering [10], which is a form of fuzzy clustering. Their formulation made an assumption which need not be true in general, namely, the probabilities of associating the data points to the clusters are independent.

Another class of clustering algorithms consists of the hierarchical techniques which include the agglomerative and the divisive methods. Starting with every data point as a singleton cluster, agglomerative algorithms build up a tree known as a *dendrogram* by merging the clusters according to some similarity measures until all the clusters are grouped into one. Divisive algorithms reverse the process. However, different similarity measures lead to qualitatively different results. This necessarily means that the user has to have some knowledge about the possible clustering of the data. It has also been reported that with random data, agglomerative methods tend to produce trees which show characteristics of clustered data. In other words, it is very difficult to interpret the tree, not to mention a tree obtained from multi-dimensional data.

In addition to the aforementioned difficulties, existing algorithms suffer from the inability to handle variabilities in cluster shapes, cluster densities and cluster sizes. Recently, Gath and Geva [8] proposed an unsupervised fuzzy clustering technique which is a modified form of fuzzy maximum-likelihood estimation [9]. Based on simulations, it seemed to demonstrate the ability to solve some of the above problems. However, it was said that "the algorithm requires starting from 'good' seed points" [8].

Among the dilemmas mentioned above, the single most urgent one which plagues all the algorithms is the lack of cluster validity criteria. All the algorithms tend to create clusters even when no natural clusters exist in the data. Yet to both a researcher and a layman, the notion of "cluster" seems easy to understand. It also seems easy to define what a clustering algorithm should do.

In this chapter, we present a new way of looking at the problem of clustering, derive a new approach based on information theory and statistical mechanics, and prove some of its properties. The original motivation behind this work came from research in neural networks[1]. The new formulation presented here allows us to identify clustering with a thermodynamic system. The local minima of the thermodynamical free energy correspond to the cluster centers. It turns out that the local minima can be treated as the fixed points of a one-parameter nonlinear map. The map undergoes local bifurcation as the temperature-like parameter reaches certain critical values. Using bifurcation techniques [13, 14], we rigorously obtain a complete picture of the dynamics of the map.

Finally, based on the insights gained, we devise a new clustering algorithm by reversing the process of annealing, i.e. by melting. Hierarchical clustering at different scales is achieved as temperature is gradually raised. We introduced a new definition of cluster which is demonstrated successfully on both simulated and real data. That definition plus robustness in the scale space give rise in a natural way in determining the optimal number of clusters in the given data in a completely unsupervised fashion, i.e. there exists a natural cluster validity criterion. Also, the initialization of the cluster centers is accomplished in a way which depends trivially on the given data. The algorithm can also account for variability in cluster densities, cluster sizes and cluster shapes (ellipsoids). Finally, the algorithm was tested on an actual multi-dimensional Synthetic Aperture Radar (SAR) image of an agricultural land for crop identification, where it did better than the existing algorithm.

A main contribution of this work is that this interdisciplinary approach from information theory, thermodynamics and nonlinear dynamics can provide the proper formulation for effective clustering. We will try to convince the reader in the rest of the chapter.

[1]The problem was how to efficiently allocate neurons that have local responses [11].

2.2 Scale and Cluster Independence

Roughly speaking, given a set of data points, the goal of clustering is to partition the data into a number of subgroups, with the data points in each subgroup forming a "cloud" in the underlying multi-dimensional space. There are many ways to define a criterion for clustering [1, 19]. For example, maximum-likelihood estimation (MLE) [20] is a parametric estimation-based approach. A commonly used criterion in k-means or partitioning algorithms is the sum-of-squares-error function. This criterion minimizes the within-cluster distances and maximizes the between-cluster distances. The fact that many criteria for clustering exist merely points out that the concept of "cluster" is not well-defined: exactly what do we mean when we say a cluster is located at position y?

We consider a very fundamental way of looking at clustering and examine the existing algorithms in a critical way. First, let us do a thought exercise to illustrate the key ideas. A set of data points is given. Imagine that we are looking at the data through a lens of certain resolution. We see a cluster if there is a bunch of data points scattered inside the field of view while few data points are seen in the area bordering them. Since we group all the data points within view into a cluster, the center of the cluster can be taken to be a weighted average of all the data points, with large weights for those inside and small weights for those outside. The center should coincide with the center of the lens. If we are looking at the data at a very fine scale, it is perfectly legitimate to say that every datum is itself a cluster. On the other hand, if we are looking at the data at a very coarse scale, then we could also say that all the data points belong to a single cluster. This simple mind exercise tells us that the answer to the question "What is the number of clusters?" depends critically on the scale parameter. This fact is not new and has been recognized by other researchers. What is surprising is that the existing algorithms either have not incorporated "scale" at all or have not addressed it in a satisfactory manner.

It is obvious that partitioning techniques do not deal with scale. One might be tempted to view agglomerative or divisive algorithms as hierarchical techniques which perform clustering in the scale space. Unfortunately, the mechanisms used to merge the clusters fail to capture the essence of "scale" in the tree. It is also known that different distance measures tend to bias these clustering results. Moreover, the algorithms are not able to distinguish random data points from clustered data. Fuzzy clustering is another class of algorithms which attempts to address this issue. But fuzziness only measures how strongly the data points are associated with the clusters; it is not a measure of scale. No existing clustering algorithms can generate clusters at different scales. Scale, such an important ingredient in clustering, has indeed been missing in the existing clustering algorithms.

In the existing algorithms, the clusters interact. In updating the parameters for a cluster, one has to take into account the values of the parameters for the other clusters. For example, to update the cluster representatives in partitioning algorithms, one has to decide which data points are associated with it. To do so, one has to compute the distances of a data point to all the clusters. In maximum-likelihood estimation (MLE), although one assumes the functional independence of the parameters, the likelihood is a function of all the cluster parameters. Likewise, in neural networks where unsupervised competitive learning models often feature clustering elements, neurons (representing clusters) interact in a competitive manner. Cluster independence has in fact not been recognized as an issue by the previous researchers. We will show that it is an essential ingredient in clustering.

Consider the situation where several people are given the same kind of lens, the same rule about clusters and the same data set. Each person may find a different cluster depending on the initial choice of the search area. But the assignment of a particular cluster is independent of what the other persons are doing. That is, the determination of a cluster is independent of the assignment of the other clusters. It is possible for two individuals to find the same cluster. If the lenses are very small, each person may find a cluster very quickly and the clusters tend to be different. If the lenses are large, the probability of two or more individuals locating the same cluster increases.

The above analogy says that as scale increases, the clusters are expected to merge by some mechanism. In other words, clusters interact in the scale space by merging, not by telling the other clusters where to place their centers. The principal reason behind this interaction is the notion of robustness. Clusters differ in robustness, which is embodied in the attracting dynamics inherent in a cluster. As scale changes, some clusters may adjust their locations; some less robust ones may even disappear. That is, some clusters may not be stable with respect to changes in the scale space. Such scenarios are very likely, yet a discussion is absent from the other existing algorithms.

2.3 Statistical Mechanics Formulation

Based on the above reasoning, we introduce the postulates which form the basis for our formulation:

Postulate: *At any given scale, the choice of a cluster is*

- *dependent on* all *the data (on some strongly and on some weakly); and*
- independent *of the other clusters.*

If we treat the clusters as particles, the latter postulate says that they do not interact. We remark that these postulates have strong appeal to the intuition behind clustering. Our earlier analogy implies that the data points near the cluster centers should give more information about the clusters while the data points far away should give less. One way to implement this idea is to assign a cost of having a data point reveal the cluster locations. For example, the data points should incur little cost on a cluster nearby because it is natural that they should give us more information about that cluster. Conversely, the data points should incur a higher cost on a cluster far away. But, to make a cluster robust with respect to scale, we desire that the information about a cluster should not be conveyed by a single datum. Rather, the information should be spread among the data as evenly as possible. If we treat the contributions to the determination of a cluster from all the data points as a probability distribution, another way of phrasing our criterion is that we should choose this probability distribution such that its entropy is maximized subject to a linear cost constraint. Therefore, we see that the maximum entropy principle [21] applies naturally.

We are now ready to formalize our ideas. Let us denote the contribution of a data point x to a cluster y, or equivalently, the probability[2] that the cluster is influenced by datum x, by $P(x)$. Suppose there is a cost $e(x)$ for associating the data point x to the cluster centered at y. For example, we may choose $e(x) = (x - y)^2$. Then, for this choice of cluster center, we can define an average cost as follows:

$$C = \sum_x P(x)e(x). \tag{2.1}$$

Our criterion says that among all probability distributions which satisfy equation (2.1), pick the one which maximizes its entropy

$$S = -\sum_x P(x)\log P(x). \tag{2.2}$$

This is a simple yet very powerful way to transform the problem into a problem in statistical mechanics. Using Lagrangian multipliers, one can show that the probability distribution is the the "Gibbs distribution" [22]:

$$P(x) = \frac{e^{-\beta e(x)}}{Z} \tag{2.3}$$

where Z is the "partition function"

$$Z = \sum_x e^{-\beta e(x)}. \tag{2.4}$$

The "free energy" is

$$F = -\frac{1}{\beta}\log Z. \tag{2.5}$$

In the above formulations, all the quantities depend on y. But y has been deliberately left out in the notation because our postulates allow us to consider one cluster at a time. This greatly simplifies the notation.

[2]We do not attempt to describe $P(x)$ in terms of *a priori*, *a posteriori* or *state-conditional* probabilities of the clusters.

To make the connection with thermodynamics, let $1/\beta$ be the temperature. As will be seen later, temperature here is a measure of the resolution or scale. The higher the temperature, the coarser is the scale and vice versa. The temperature parameter is not a measure of fuzziness in our formulation, however.

Having established a link to the statistical mechanics formulation, we can now investigate properties of this system. Without loss of generality, we restrict the notation and the exposition to the case of one-dimensional data. The extension of the analysis to higher-dimensional data is straightforward though more complicated and is treated in the appendix. The good news is that the dynamics are essentially the same. For the time being, let us choose the cost function

$$e(x) = (x-y)^2 \tag{2.6}$$

where y is the center of the cluster. This means that we use the squared distance as a measure of the compactness of a cluster.

At equilibrium, it is known that a thermodynamical system settles into configurations which minimize its free energy. The free energy is minimized when $\partial F/\partial y = 0$. That is, we want

$$\sum_x (x-y)e^{-\beta(x-y)^2} = 0 \tag{2.7}$$

or equivalently

$$y = \sum_x \frac{xe^{-\beta(x-y)^2}}{\sum_x e^{-\beta(x-y)^2}}. \tag{2.8}$$

Let us introduce the following definition:

Definition 1 *A* nominal cluster *is* centered at y *if and only if* y *is a local minimum of the free energy of the thermodynamical system described above.*

Equation (2.8) shows that the nominal cluster center y is just a weighted average of the data, which is well known to minimize the squared distortion. Note two features of the equation: (1) The weight is an exponential decreasing function of the distance of the data points from its nominal cluster center; and (2) The localization of the weighting function is directly controlled by the scale or inverse temperature β. These two features turn out to be crucial in terms of nonlinear dynamics.

We have derived the nominal cluster center by assuming that y is a cluster center. Thus equation (2.8) is only a necessary condition for y to be a cluster. What are the sufficient conditions? We will defer the answer because we have to study the dynamics of equation (2.8) to get clues to the answer. Because of that, we will use "cluster" instead of "nominal cluster" until we precisely define the term "cluster." Without worrying about whether nominal clusters are real clusters, one can see that all the questions we want to ask have answers in the properties of the solution space of equation (2.8). For example,

1. Do clusters exist? The answer depends upon whether the equation has any solutions.

2. How many clusters are there? This depends on the number of solutions the equation has.

3. How do the clusters evolve? The answer is given by the trajectories of the solutions of equation (2.8) as β changes.

The above list could have been longer but it suffices to illustrate the importance of equation (2.8) in our formulation. Since we are only concerned with local minima, the question of global optimization is irrelevant here. This is a great advantage over other applications of physical optimization where global optima are sought.

2.4 Dynamics of the Nonlinear Map

We borrow some tools from nonlinear dynamics to investigate the nature of the solution space as β is varied. Clearly, the solutions of equation (2.8) are identical to the fixed points of the following one-parameter map[3]:

$$y \xmapsto{f} y + \sum_x \frac{(x-y)e^{-\beta(x-y)^2}}{\sum_x e^{-\beta(x-y)^2}}. \tag{2.9}$$

[3]We could have instead established a similar equivalence with the differential equation $dy/dt = \sum_x (x-y)e^{-\beta(x-y)^2}$, but the analysis is very similar and the results are the same.

This connects our problem with nonlinear dynamics. We first prove the following lemma:

Lemma 1 *A nominal cluster is centered at y if and only if equation (2.7) is satisfied and* $|\frac{\partial f}{\partial y}| < 1$.

Proof: For a fixed point of an iterative map to be attractive, the magnitude of its derivative clearly must be less than one. Thus, the above conditions are just the necessary and sufficient conditions for y to be a local minimum of the free energy. $\square$

Before we continue, we point out that the formulation is inherently "fuzzy" [7, 6, 8], although not strictly in the sense of fuzzy logic. Except in the case when every datum freezes into a cluster and contributes exclusively to that cluster, the contribution depends on the distance of that data point from the cluster center.

Let us study the fixed points at extreme values of β. When the temperature is very high, i.e. $\beta = 0$, $e^{-\beta(x-y)^2} = 1$. There is only one cluster and its center is the centroid of the data, as expected at a very coarse scale. At the other extreme when the temperature is very cold, i.e. $\beta = \infty$, we have $e^{-\beta(x-y)^2} = \delta_{xy}$. Thus every data point is a cluster since we are looking through a huge magnifying lens. We see that β is directly related to the scale: small β means coarse scale and vice versa.

For $\beta \neq 0$ but finite, and y sufficiently far away from the data points, let x_c be the data point that is closest to y; then $\frac{e^{-\beta(x_c-y)^2}}{\sum_x e^{-\beta(x-y)^2}} \approx 1$. Thus, the map f approaches x_c when y is sufficiently large. (If there are m data points sitting at x_c, the sum of m terms of $\frac{e^{-\beta(x_c-y)^2}}{\sum_x e^{-\beta(x-y)^2}} \approx 1$.) The slope of the map is given as

$$\frac{\partial f}{\partial y} = 2\beta\Big(E(x^2) - E(x)^2\Big) \geq 0 \tag{2.10}$$

where $E(x) = \sum_x \frac{xe^{-\beta(x-y)^2}}{\sum_x e^{-\beta(x-y)^2}}$ is the expectation operator.

We see that the map looks like the sketch in Figure 2.1. To create two or more nominal clusters, the graph of f has to cross the line $f = y$ at least as many times as the number of nominal clusters. As β varies from a huge value to a small value, the number of crossovers decreases. That is, the map must change behavior as β

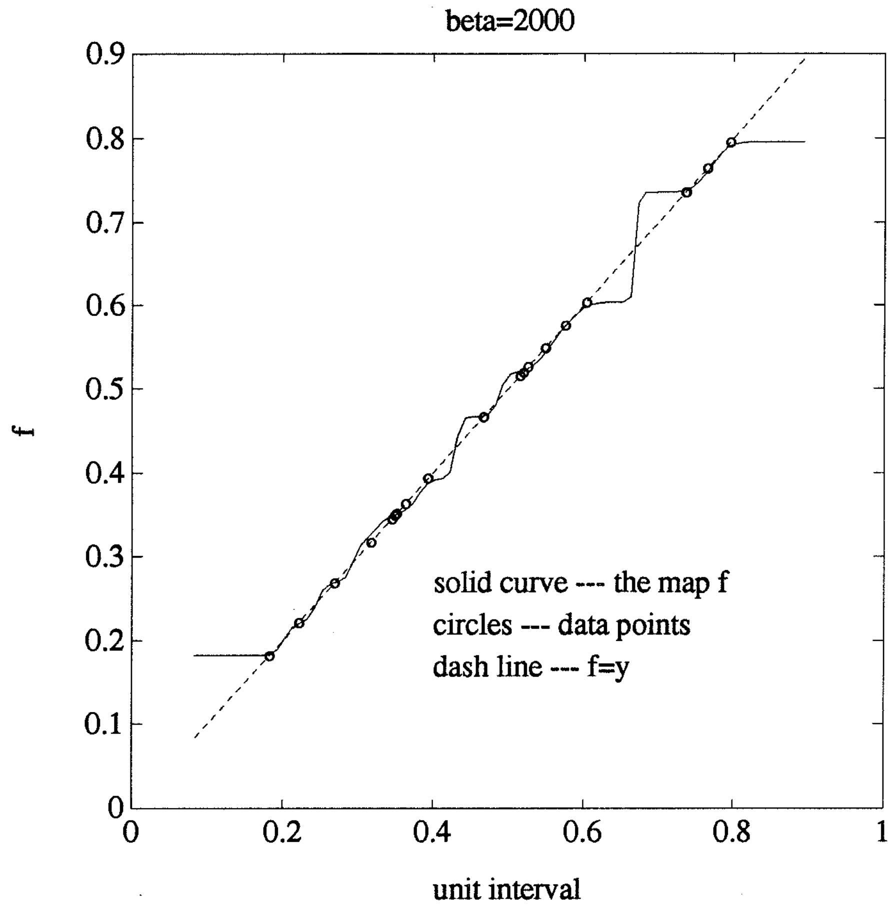


Figure 2.1: The map for 20 data points along the unit interval.

increases. The only way for this crossover to happen is that there are critical values of β when the map f undergoes bifurcation.

From local bifurcation theory [14], there are only four types of local bifurcations that can occur with one-dimensional maps. They are listed below together with the conditions (all the partial derivatives are taken at the bifurcation points).

1. *saddle-node*: $f(y,\beta)=y$, $\frac{\partial f}{\partial y}=1$, $\frac{\partial f}{\partial \beta}\neq 0$, $\frac{\partial^2 f}{\partial y^2}\neq 0$.

2. *pitchfork*: $f(y,\beta)=y$, $\frac{\partial f}{\partial y}=1$, $\frac{\partial f}{\partial \beta}=0$, $\frac{\partial^2 f}{\partial y^2}=0$, $\frac{\partial^2 f}{\partial y\partial \beta}\neq 0$, $\frac{\partial^3 f}{\partial y^3}\neq 0$.

3. *transcritical*: $f(y,\beta)=y$, $\frac{\partial f}{\partial y}=1$, $\frac{\partial f}{\partial \beta}=0$, $\frac{\partial^2 f}{\partial y\partial \beta}\neq 0$, $\frac{\partial^2 f}{\partial y^2}\neq 0$.

4. *period-doubling*: $f(y,\beta)=y$, $\frac{\partial f}{\partial y}=-1$, $\frac{\partial f^2}{\partial \beta}=0$, $\frac{\partial^2 f^2}{\partial y\partial \beta}\neq 0$, $\frac{\partial^2 f^2}{\partial y^2}=0$, $\frac{\partial^3 f^2}{\partial y^3}\neq 0$.

Immediately we see that period-doubling is ruled out because it would require that the slope of the graph of f be negative: there can be no chaos in our formulation. The following theorem shows that transcritical bifurcation cannot occur in this clustering problem either.

Theorem 2 *For the map as defined in equation (2.9), there is no transcritical bifurcation.*

Proof: Instead of writing $x-y$ all the time, let $\hat{x}=x-y$. Note that $d\hat{x}/dy=-1$. In this new notation, the map becomes

$$y \overset{f}{\longmapsto} y+\sum_{\hat{x}} \frac{\hat{x}e^{-\beta\hat{x}^2}}{\sum_{\hat{x}} e^{-\beta\hat{x}^2}}. \tag{2.11}$$

At fixed point y, this leads to

$$\sum_{\hat{x}} \frac{\hat{x}e^{-\beta\hat{x}^2}}{\sum_{\hat{x}} e^{-\beta\hat{x}^2}} = E(\hat{x}) = 0. \tag{2.12}$$

Let us rewrite equation (2.10) as

$$\frac{\partial f}{\partial y} = 2\beta\frac{\sum_{\hat{x}} \hat{x}^2 e^{-\beta\hat{x}^2}}{\sum_{\hat{x}} e^{-\beta\hat{x}^2}} = 2\beta E(\hat{x}^2) \geq 0. \tag{2.13}$$

The second derivative is

$$\frac{\partial^2 f}{\partial y^2} = 4\beta^2\frac{\sum_{\hat{x}} \hat{x}^3 e^{-\beta\hat{x}^2}}{\sum_{\hat{x}} e^{-\beta\hat{x}^2}}. \tag{2.14}$$

Similarly,

$$\frac{\partial f}{\partial \beta} = -\frac{\sum_{\hat{x}} \hat{x}^3 e^{-\beta\hat{x}^2}}{\sum_{\hat{x}} e^{-\beta\hat{x}^2}}. \tag{2.15}$$

Comparing equations (2.14) and (2.15), we see that $\frac{\partial f}{\partial \beta}$ and $\frac{\partial^2 f}{\partial y^2}$ have opposite signs. The proof is established as we observe that the conditions for transcritical bifurcation cannot now be satisfied. □

Therefore, there are only two candidates left: pitchfork and saddle-node bifurcations. Their bifurcation diagrams are sketched in Figures 2.2(a) and 2.2(b) respectively. A bifurcation diagram shows the trajectories of the fixed points as the parameter β is varied around its critical value.

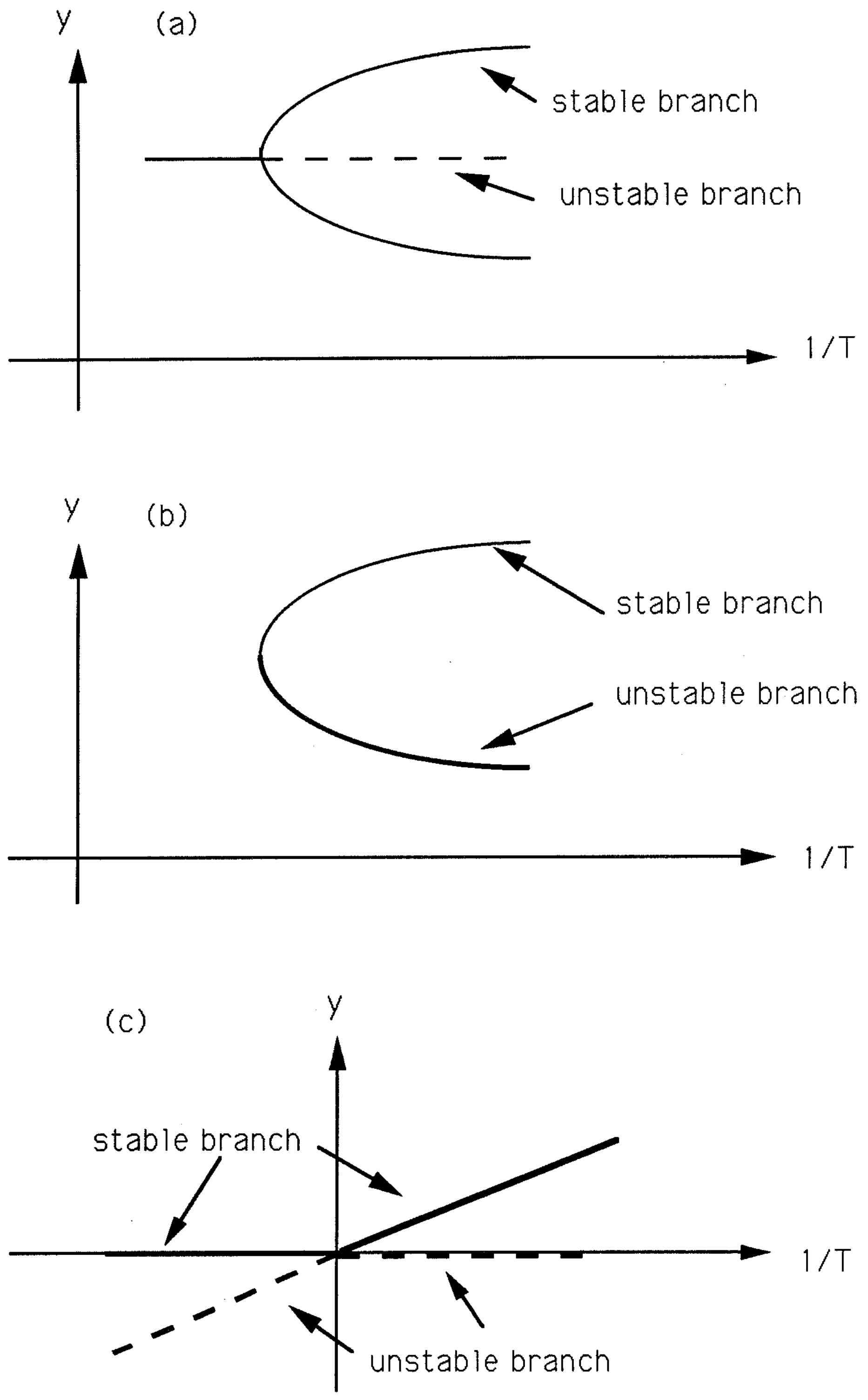


Figure 2.2: (a) A pitchfork bifurcation. (b) A saddle-node bifurcation. (c) A transcritical bifurcation.

In pitchfork bifurcation, a stable fixed point becomes unstable, giving rise to two neighboring stable fixed points. In clustering, this would correspond to the splitting of a large cluster into two smaller clusters as the scale becomes finer. Reversing the picture, it says that two clusters slowly merge into one cluster by bringing their cluster centers continuously together.

In a saddle-node bifurcation, a solution to equation (2.8) appears somewhere which then immediately bifurcates into two branches: one stable and the other unstable. In terms of clustering, this means that as the scale decreases, a cluster can appear somewhere. So as the scale increases, a cluster can thus disappear. The cluster center will be mapped to another cluster center, not necessarily infinitesimally close, under the repeated action of the iterative map in equation (2.9).

When two clusters merge, the respective trajectories of the fixed points will be identical thereafter. The clusters become indistinguishable. Therefore, the data points associated the merged clusters are merged as well. No matter what kinds of bifurcation occur, the number of clusters will become fewer as we increase the temperature. Eventually, there will be only one cluster left: every data point belongs to a single cluster. Hence we have the following melting procedure:

Melting Procedure

1. Choose β_{max}; β_{max} is a number related to the dynamic range and an assumed noise in the observations;
2. let every data point be a cluster;
3. set $i = 1$, $\beta_1 = \beta_{max}$;
4. iterate according to the mapping (2.9) N times or until the map converges by testing the difference between successive y's;
5. record the cluster centers;
6. if more than two clusters share the same center, merge the data associated with the original clusters. This will become the set of data associated with the new

cluster; call this cluster a *node* and denote it by (y, β);

7. $i = i + 1$, $\beta_i = \beta_{i-1}/1.05$ (here 1.05 has been chosen arbitrarily and works fine in our simulations);

8. if there is more than one cluster, go to 4; otherwise stop. Melting is complete.

An explanation of the choice of β_{max} is warranted. β_{max} should be chosen such that the number of clusters at $i = 2$ should not be significantly less than the number of data points to start with. Otherwise, the initial temperature is too high, which might cause partitioning of the data set prematurely. The initial choice can be obtained easily by simple pre-processing. Another note is that testing for convergence can be accomplished in several ways. In our work, we set a limit to the number of iterations at $N = 200$, but the iteration can stop anytime when the difference between successive y's is less than a threshold. Finally, we emphasize that the purpose of the melting schedule is to track the trajectories of the fixed points in the scale space. Instead of decreasing β by a constant factor, we can also utilize the numerical techniques such as continuation [15] and adaptive stepsize selection [16] to track the bifurcation points more accurately and speed up the melting procedure.

Whenever two clusters merge, we create a node labeled by the pair (y_m, β_l). This means (y_m, β_l) is the parent of (y_n, β_k) if the merging is the only one (y_n, β_k) encounters as the scale changes from β_k to β_l. At the finest level, when β is very large, we have all the data points as the leaves or initial nodes. The merging of the branches corresponds to the merging of the clusters. At the coarsest scale, we have the root of the tree. Therefore, it is clear that this procedure generates a strict tree structure, or what is also known as a *dendrogram*. If we cut the tree at a single β, the data sets associated with the nodes are non-overlapping and their union is the original data set. This is to be the basis for our hierarchical clustering algorithm.

We should make a comment about why we choose to "melt" the system starting from a low temperature, as contrasted with annealing [12]. In annealing, one starts with high temperature and throws in many clusters (particles), allowing the system to come to an equilibrium. One then gradually lowers the temperature according to some annealing schedule. In the case of clustering, this procedure can be repeated until every data point becomes a cluster. Since our formulation involves saddle-node bifurcation, it means that a new local minimum can appear some distance away from the existing minima. Annealing would fail since we do not know how much perturbation or hill-climbing is needed to move a particle to the other local minimum. Therefore, the melting process is our choice. One can see that this melting procedure has considerable advantages over the annealing procedure. Since the basins of attraction of the fixed points form a partition of the input space, the merging of two fixed points results in larger basins of attraction for the new fixed point. As temperature increases, both the weighting function and the map become smoother. If a cluster center has to adjust its position, its old location is equivalent to a perturbed position of the new cluster center. Thus, a kind of "noise" is introduced automatically by heating the system.

Figures 2.3a and 2.4a are examples of the trees generated by the melting procedure in the case of one-dimensional data. The graphs are obtained by plotting the trajectories of the fixed points versus scale. The horizontal axis indexes i in the melting procedure. (Since scale increases with the index i, in the later plots, we will merely identify scale with i.) The scale is, in fact, plotted logarithmically because of the exponential terms in equation (2.8). The original data is plotted as $\star$ at $i = 0$. As scale increases, we see that the number of distinct curves decreases due to merging, as we expect.

Now let us consider saddle-node versus pitchfork bifurcations and what they mean.

2.4.1 Saddle-node versus Pitchfork

A pitchfork bifurcation means that two clusters merge continuously into one cluster. Intuitively, this is possible. But the tricky part is that the trajectories of the fixed points are continuous near the bifurcation point. The situation of two clusters sitting infinitely close to each other is, we expect, hard to achieve for clustered data in our formulation.

Let us analyze this further by computing one more partial derivative:

$$\frac{\partial^2 f}{\partial y \partial \beta} = -2\beta E(\hat{x}^4) + 3/2\beta. \tag{2.16}$$

We see that for pitchfork bifurcation to occur, we would require $E(\hat{x}) = 0$, $2\beta E(\hat{x}^2) = 1$, $E(\hat{x}^3) = 0$ and $E(\hat{x}^4) \neq 3/4\beta^2$. In terms of moments, these conditions say that the first and third moments should vanish and the second/fourth moment should/should not be equal to some specific value. Such "balanced" data is hard to find. In general, these conditions are difficult to satisfy if the data is truly clustered and β is sufficiently small.

Let us examine three simple examples. Figure 2.3a shows the fixed points of the map (2.9) as β is varied when the data consists of uniformly spaced points over an interval. The symmetry observed in the diagram says that only pitchfork bifurcations are involved here. Figure 2.3c shows the fixed points using the same data with some noise added. Note that at fine scales, one does see clusters merge by a saddle-node bifurcation. In the figure, the last two clusters merge by a pitchfork bifurcation. At fine scales, the fluctuation due to noise is detectable and makes the pitchfork quite unlikely. But at coarse scales, the noise smooths out, and thus the underlying data appears symmetric.

To further illustrate the idea of "balanced" data that is necessary for a pitchfork bifurcation to occur, another set of uniformly spaced data was generated. The data was translated and then reflected to obtain a mirror image of the original data set. These two sets of data are then combined. Thus the data has two clusters. Random, but small, noise was added to the data. The trajectories of the fixed points are plotted in Figure 2.4a. Although global symmetry is broken, one still observes pitchfork bifurcation when the last two clusters combine into one. This is because the two sets of data are separated; at a coarse scale, the noise again smooths out and the two clusters are almost identical.

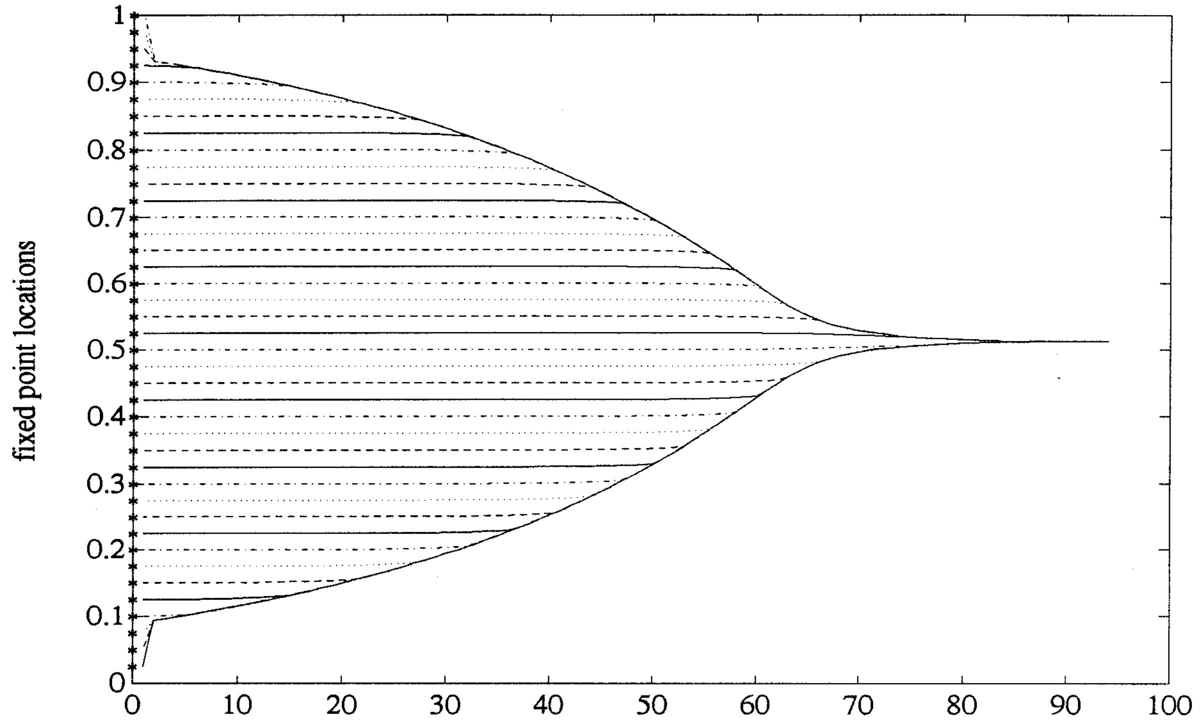


Figure 2.3a. The fixed points versus scale for uniformly spaced data

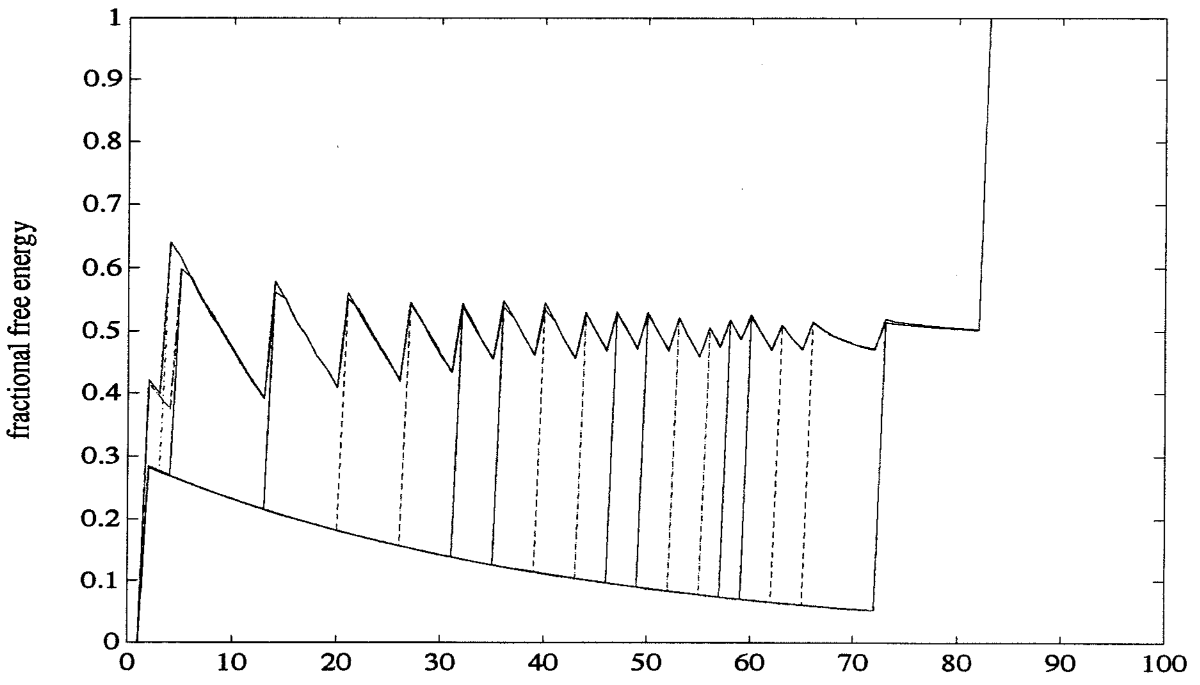


Figure 2.3b. The FFE's of the fixed points versus scale for uniformly spaced data

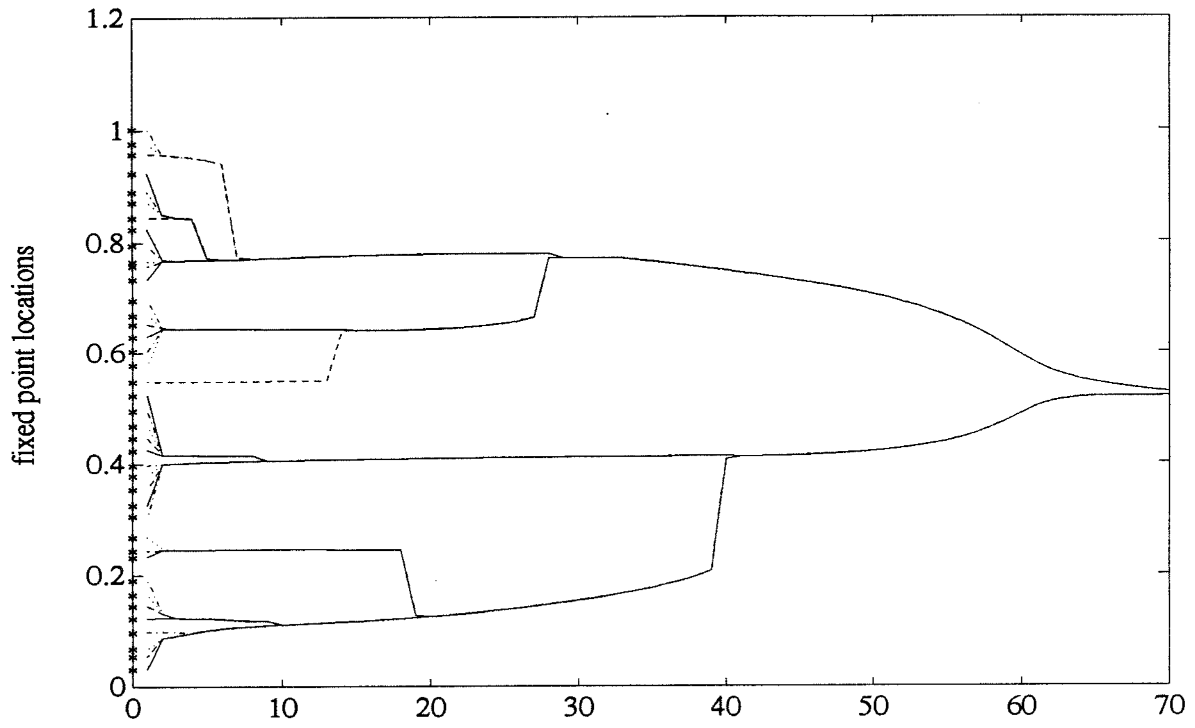


Figure 2.3c. The fixed points versus scale for uniformly spaced data corrupted with noise

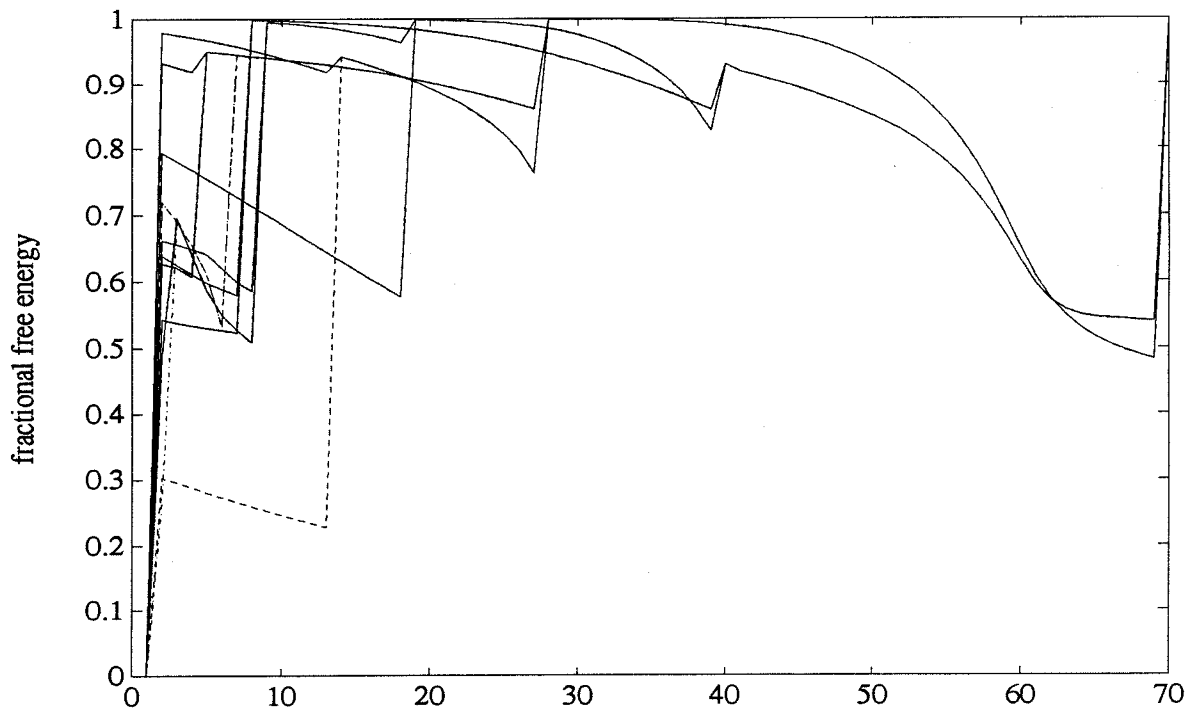


Figure 2.3d. The FFE's of the fixed points versus scale for the data shown in Fig. 2.3c

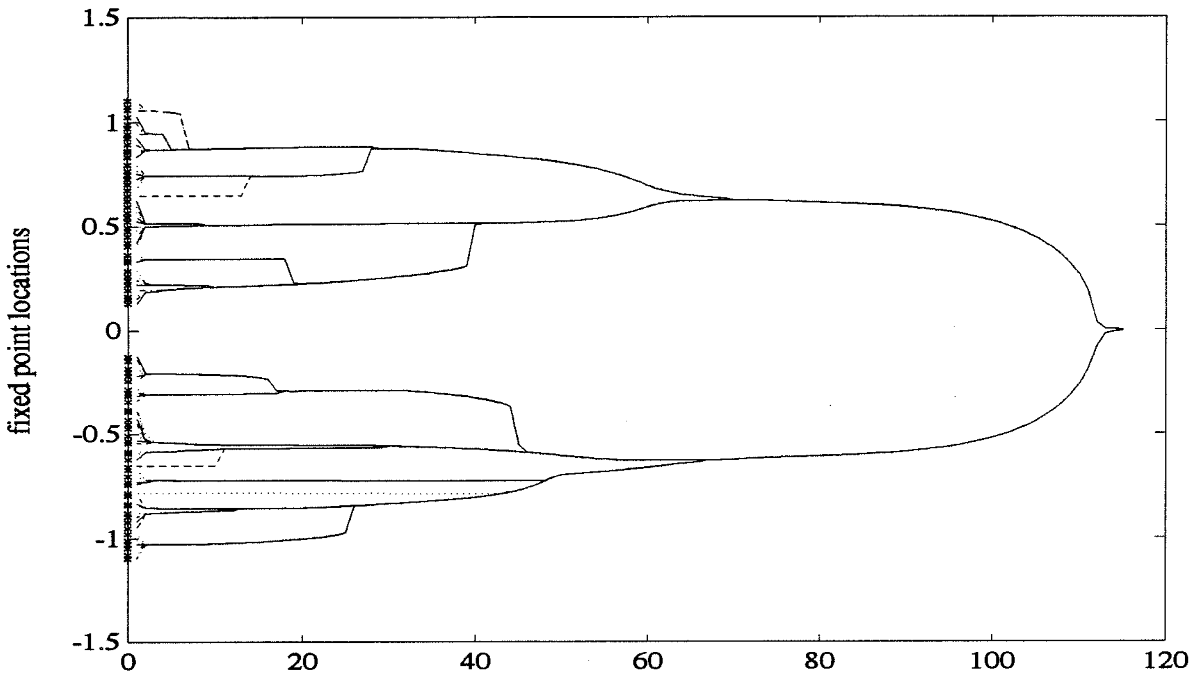


Figure 2.4a. The fixed points versus scale. The leftmost points are the data.

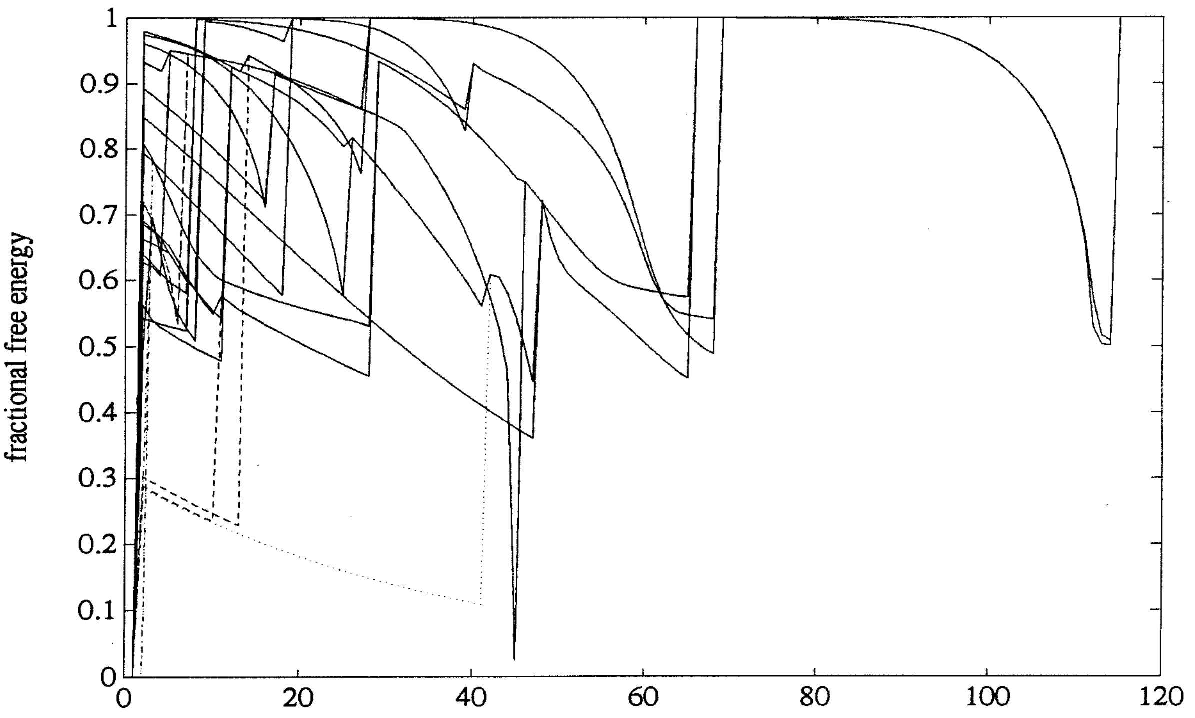


Figure 2.4b. The FFE's of the fixed points versus scale for the data shown in Fig. 2.4a

Hence two conclusions one can draw from these simulations and the accompanying analysis are that a pitchfork bifurcation indicates

1. uniformly spaced or non-clustered data; or

2. clustered data but with a high degree of symmetry in the data space with respect to scale.

What about a saddle-node bifurcation? Its key feature is that a cluster can disappear and its center gets mapped to another cluster center under the repeated iterations of the map (2.9). There is thus a jump in the location of the fixed point of the map (2.9). Here y is a weighted sum of the data centered about y. As one increases the scale, the balance about y can be lost due to the influence of the data from another cluster, which cannot be restored by tiny adjustments to y. The cluster just gets "siphoned away" by the attracting dynamics of the other fixed point. Thus, an inhomogeneous spatial distribution present in the data is captured by a saddle-node bifurcation. That is exactly what clustering is really all about. Even for the quite unstructured data in Figure 2.3c and the artificially generated clustered data in Figure 2.4a, one can see instances of the saddle-node bifurcation in action.

To distinguish the two clusters in a saddle-node bifurcation, here is one more definition which is needed in the later sections:

Definition 2 *In a saddle-node bifurcation, a* losing *cluster is the one which gets siphoned off by the other cluster called the* winning *cluster. In a pitchfork bifurcation, both clusters* win.

To recapitulate, we expect both types of bifurcations to occur in our formulation (see Figures 2.2a and 2.2b again). Intuitively, a pitchfork bifurcation happens in systems which have some underlying structure which is symmetric, while a saddle-node bifurcation happens in systems which show some degree of asymmetry. For clustering at fine scales, when relatively few data points are "seen" by a cluster center and the distances between neighboring clusters are small, a pitchfork bifurcation is possible.

However, as scale increases, i.e. as more and more data points are grouped into a given cluster and the total number of clusters decreases, saddle-node bifurcation will be the dominant form of bifurcation to occur in the system. This is indeed true in all the simulations presented in this chapter except the one shown in Figure 2.3b. Nevertheless, it is important to keep in mind that in a saddle-node bifurcation, the winning cluster center does not go through bifurcation. It just happens that its basin of attraction comes to include the losing cluster center. For the moment, we will denote the new fixed point of the losing cluster as another node at which merging occurs. This statement will be subject to minor modification later. To avoid repetitiveness, from now on we will use the word "merge" to describe the disappearance/merging of the clusters due to saddle-node/pitchfork bifurcation.

In other existing clustering algorithms, saddle-node bifurcations are not observed. In partitioning techniques, the rules on how to discard or split the clusters barely resemble saddle-node bifurcation. In agglomerative methods, the merging of the tree only gives a remote semblance to saddle-node type dynamics.

2.4.2 Bifurcation and Rate Distortion Theory

A fixed point y is stable if and only if

$$\sum_{\hat{x}}(1 - 2\beta\hat{x}^2)e^{-\beta(x-y)^2} > 0 \tag{2.17}$$

or equivalently,

$$2\beta\frac{\sum_{\hat{x}}\hat{x}^2 e^{-\beta\hat{x}^2}}{\sum_{\hat{x}} e^{-\beta\hat{x}^2}} < 1. \tag{2.18}$$

But $1/2\beta$ is just the variance of the Gaussian distribution in equation (2.7). Hence the cluster disappears (saddle-node) or merges (pitchfork) when the shifted data has variance exactly matching the assumed variance of the Gaussian distribution. This is intuitively pleasing. Thus, we can interpret condition (2.18) as the justification for y to be a cluster. Once this condition is violated, we can say a cluster loses its reason to be a cluster.

For higher-dimensional data, $\frac{\partial f}{\partial y}$ is a Jacobian matrix [14]. This is 2β multiplied by the covariance matrix whose element c_{ij} is

$$\frac{\sum_{\hat{x}_i} \hat{x}_i \hat{x}_j e^{-\beta \hat{x}^2}}{\sum_{\hat{x}} e^{-\beta \hat{x}^2}} \tag{2.19}$$

where $\hat{x}_i$ is the i^{th} component of the shifted datum $\hat{x}_i = x_i - y$ in the higher-dimensional space. Since a covariance matrix always has non-negative eigenvalues, it means that the mapping (2.9) is orientation-preserving. In addition, a fixed point is stable if and only if all eigenvalues of the Jacobian are less than $1/2\beta$. At the critical value of β, there is a direction (eigenvector) in the data space along which the Jacobian has an eigenvalue of $1/2\beta$. That is, projecting the data points onto w yields a set of data points which contribute a distortion of $1/2\beta$ to the cluster. In the appendix, we will show that one can observe the same kinds of bifurcation in multi-dimensional data as in the one-dimensional case. This, indeed, is very satisfying since it is consistent with our intuition.

We can also find an information-theoretic basis for condition (2.18). Rate-distortion theory deals with the question of the minimum number of bits needed to faithfully encode a source symbol subject to an expected distortion constraint [17]. The rate distortion function of a Gaussian source with variance σ^2, subject to the constraint that the average distortion $\leq \delta$, is

$$R(\delta) = \begin{cases} \frac{1}{2}\log(\sigma^2/\delta) & \text{if } \delta \leq \sigma^2 \\ 0 & \delta \geq \sigma^2. \end{cases} \tag{2.20}$$

Thus, when equation (2.18) becomes an equality, the distortion matches the variance of the Gaussian. $R(\delta) = 0$ signifies that there is no need to waste bits to encode the signal. That is, the cluster should either disappear or be merged. It is rather satisfying that two diverse disciplines—information theory and nonlinear dynamics—converge to the same result.

Having obtained a complete picture of the dynamics, what can we do with it?

2.5 Cluster Validity

In this section, we will try to address the issue of cluster validity. The cluster validity problem [7, 27] deals with the question of a utility criterion for determining the number of clusters. This question is closely related to the question of deciding how to associate the data points to the clusters. In some sense, this reminds one of the chicken and egg problem. How do you know where the clusters are without knowing which data points belong to the same cluster? But how do you know which data points belong to the same cluster without knowing where the clusters are? In the literature, a typical approach is to assume the number of clusters is known or to repeat the clustering procedure several times for each initial guess and pick the one which maximizes some utility function [19, 1]. This approach is far from satisfactory. The issue of cluster validity has not been fully addressed by the clustering schemes proposed by others.

Let us try to find out which comes first: chicken or egg? Even if we know the number of clusters, the question of partitioning is hard. In fact, the general problem of partitioning a data set into k groups such that the total distortion is minimized is NP-complete [18]. This means that knowing k may not help to make the partitioning task easier. But fortunately, clustering is not partitioning the data such that the total distortion is minimized. On the other hand, if we know which data points should be grouped together, the answer to the number of clusters and the cluster representatives follow naturally. This seems to indicate that the key difficulty which has set back past efforts is that the concept of "cluster" is not well specified. We shall now make use of our insights obtained from the new formulation to come up with a definition of cluster. We shall claim that we have a good solution to this chicken and egg problem.

2.5.1 What is a Cluster?

Recall that a nominal cluster center is just a local minimum of the free energy F. In thermodynamics, free energy is interpreted as the amount of energy which can be converted to external work if the temperature is held constant [22]. From a dendrogram, for example, one can backtrack and find the set of data points Q which is associated with any nominal cluster center y. One can compute how much of this free energy comes from the data points associated with y. The number is just

$$F_y = -1/\beta \sum_{x \in Q} P(x) \log(Z). \tag{2.21}$$

Hence the fraction of the free energy is just

$$M(\beta) = \sum_{x \in Q} P(x) = \frac{\sum_{x \in Q} e^{-\beta(x-y)^2}}{Z}. \tag{2.22}$$

Definition 3 *The* fractional free energy (FFE) *of a nominal cluster y is defined as in equation (2.22).*

Recall that $P(x)$ is interpreted as the contribution of a data point x to the cluster y. Thus, $M(\beta)$ is a measure of the goodness of the nominal cluster at a given scale β. For the range of scales over which no merging occurs, we ask, is $M(\beta)$ decreasing? Note that Z is increasing with temperature:

$$dZ = 2\beta \sum_{x} (x-y) e^{-\beta(x-y)^2} dy - \sum_{x} (x-y)^2 e^{-\beta(x-y)^2} d\beta. \tag{2.23}$$

At a fixed point, the first term on the right is zero. Hence $dZ = -E((x-y)^2) d\beta > 0$ (remember for melting, $d\beta < 0$). Therefore, Z always increases. This means that the spread of the weighting function cannot decrease. This is because as scale increases, the influence of the data points outside S on the cluster grows. Taking the adjustments of the fixed point into account, we can safely assume that the contribution from the data points associated with a cluster drops as β decreases. So $M(\beta)$ is indeed decreasing. Moreover, as the scale is further increased, $M(\beta)$ will drop quickly due to the onset of instability. This is confirmed in the simulations. Therefore, a large FFE indicates that most of the contributions come from the data belonging to the cluster itself. Likewise, a small FFE is a sign that the cluster is highly influenced by data outside of itself. What is small and what is large is set by a threshold, which will be useful in distinguishing good clusters among the nominal clusters. Based on our intuition and observations from the simulations, we have chosen $M_T = 0.95$ in our work[4]. This value of M_T

We now make the following two definitions:

Definition 4 *An FFE M is called* good *if $M > M_T$.*

Definition 5 *The* figure of merit *of a cluster is the FFE of that cluster at its recording as a node. A figure of merit M is called* good *if $M > M_T$.*

Note that $M(\beta)$ depends on the scale parameter. When the scale is very fine, each cluster contains at most a few data points. Then almost certainly $M(\beta)$ will be close to 1. In practice, we need to set a limit for the minimum number of data points in a cluster. This limit depends on the dimensionality of the data space. If the features are uncorrelated, then data points belonging to a cluster in the d-dimensional space will have rank d. Since we know that any set of d data points in a d-dimensional space lie in a $(d-1)$-dimensional hyperplane, we arbitrary set the limit to be $d+1$.

Based on the above discussion, let us finally present the following definition for a good cluster:

Definition 6 *A* good cluster *is a nominal cluster with the following properties:*

1. it has at least $d+1$ data points;

2. its figure of merit is good;

3. if it merges, the other cluster either

  (a) has a good figure of merit and at least $d+1$ data points, or

  (b) results from the merging of clusters with properties (1) and (2) or any of its subsequent mergings.

A *hierarchical* cluster is a cluster with the above property 3(b).

[4]This may seem *ad hoc*, but any statistic will require a threshold.

Let us now examine what happens when two clusters merge. There are several cases which we now consider. (a) This merging could result from outliers. These singleton or doublet clusters can remain stable for a long range of scales. When they lose stability, they are merged to a more stable cluster by a saddle-node bifurcation. The effect on $M(\beta)$ of the winning cluster will be almost unnoticeable. We should therefore not create a new node from this merging. (b) Two good clusters merge. A cluster with a good figure of merit may be attracted to the other cluster which also has a good figure of merit but may/may not have a good FFE at the time of merging. If the other cluster has a good FFE, the resulting FFE due to merging may be very similar to the original one. That is, we may not observe a noticeable change in the FFEs of the winning cluster. However, the losing cluster is a good cluster and hence a node should be created. (c) A good cluster merges with a hierarchical cluster which may/may not even have a good figure of merit. If the hierarchical cluster has a good FFE at the time of merging, it is usually close to 1 because it is carrying so many data points with it. In this case, we do not observe a noticeable change in the FFEs of the winning cluster. But a node is created anyway. (d) Both clusters have very few points. But when they merge, they do have more than d points and the resulting FFE is good. A node should of course be created due to such merging.

In the original version of the melting procedure in Section 2.4, a new node is created on every merging. This certainly needs to be modified in view of the above discussion. Specifically, step 6 is replaced by the following two steps:

1. Compute the $M(\beta)$ for the clusters;

2. if more than two clusters share the same center, merge the data associated with the original clusters. This will become the set of data associated with the winning cluster;

3. create a *node* denoted by (y, β) in any of the following cases:

   (a) if the difference in the winning cluster's FFE exceeds a threshold;

(b) if one cluster has a good figure of merit and the other cluster also has a good figure of merit and both clusters have more than d points;

(c) if one cluster has good figure of merit and the other cluster is hierarchical;

(d) if the winning cluster does not have a good figure of merit and is not hierarchical, but the resulting cluster has a good FFE and has more than d data points.

In our simulations, the threshold is chosen to be 2%. Also, the minimum number of data points each cluster contains can be more flexible that $d + 1$. This number can be varied according to the availability of collected data. We remark also that there are two ways to handle the outliers: (1) store them away as noise for later processing, or (2) assign them to the data set associated with the winning cluster, but create no new node. In our work here, we have chosen the second strategy.

Let us now see how we can apply our definition of good clusters in deciding the goodness of the nominal clusters. Suppose we plot the FFE's for all the fixed points according to equation (2.22). The vertical axis is the FFE and the horizontal axis is the scale parameter, as in the plots of the fixed points earlier. Since we are plotting many curves, we may not be able to identify each individual curve by inspection. But it is easy to write a computer program to do that.

Let us examine the examples presented earlier more closely. Figure 2.3b shows such a plot for the uniformly spaced data shown in Figure 2.3a. One notices that the FFE's are small for all the nominal clusters except for the last one, in which all the data points are grouped into one. This, according to our definition, means that there is only one cluster for the data. Figure 2.3d shows the plot for the data in Figure 2.3c which is basically random noise. The discontinuities on the curves are caused by the merging of the clusters. Note that there are quite a few nominal clusters with figures of merit greater than M_T, but they all merge with clusters with figures of merit less than 0.9 which is less than M_T. According to our definition, all those nominal clusters are no good. The given data set contains no clusters, which is in fact true from the way the data were generated. The key message from these two examples is: The other existing algorithms tend to claim clusters in random data, but our algorithm does not.

Figure 2.4b shows the plots of the FFE's for the data shown in Figure 2.4a. The data contains two clusters. At coarse scales, one does see that there are only two nominal clusters. Both clusters have figures of merit close to 1 and they merge through a pitchfork bifurcation. According to our definition, there are indeed two clusters.

At first sight, it may seem that our definition of good cluster is complicated and too elaborate. What the definition says is really simple, though. A cluster should contain a minimal number of points and should contain enough information about itself. When it is forced to give up its existence, it can only surrender to another one with a good figure of merit too, if not a better one. The other cluster can also be hierarchical.

Now let us attack the other part of the chicken and egg problem—the number of clusters.

2.5.2 How Many Clusters?

Having defined what a cluster means, we now need to find the number of clusters in a data set. If no good cluster is a subset of another good cluster, then the set of good clusters constitutes a partition of the data and we are done. However, since our clustering algorithm generates clusters at all scales, it is quite possible that a good cluster can be embedded in another good cluster. In this case, which one should we pick? How do we determine the optimal number of clusters? This is where "scale" comes into play.

Again, let us do a mind exercise by making use of the lens analogy. If there exist distinct good clusters, then we expect that if we track the fixed points, there must be some good clusters whose FFE's remain good over a large range of logarithmic scale in β even though the fixed points themselves may vary their positions. The trajectories of these fixed points should remain relatively stable over a long range of scales. In addition, the FFE's of these nominal clusters should start out with very high value, and remain there for a while. When a cluster is about to bifurcate, its FFE will drop quickly. Hence, given a good cluster, the longer its FFE remains high, the more robust the cluster is. When two clusters interact, certainly the more robust one wins. Therefore, we prefer the more robust cluster to the less robust one (see Example 7). Here is how we define the robustness of a good cluster:

Definition 7 *The* robustness *of a good cluster is defined as the range of logarithmic scales over which its FFE remains above M_T.*

Based on the above discussions, therefore, we have the following procedure for selecting the "better" clusters:

Sorting Out the Better Clusters

1. First decide upon all the good clusters among the nominal clusters; denote the set by $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$;
2. initialize $\mathcal{U}$ to an empty set;
3. While $\mathcal{T}$ is nonempty, do the following:
   (a) pick the element T_k in $\mathcal{T}$ with the largest robustness measure;
   (b) put this element into $\mathcal{U}$;
   (c) remove T_k itself, the elements in $\mathcal{T}$ which are subsets of T_k, and the elements in $\mathcal{T}$ which contain T_k;
   (d) collect all the data points which do not belong to one of the clusters in $\mathcal{U}$ into a set $\mathcal{N}$, which we hope is empty;
4. the cardinality of $\mathcal{U}$ equals the number of clusters.

This procedure is guaranteed to stop. The only drawback is that it is possible that not every data point belongs to one of the clusters in $\mathcal{U}$, i.e. $\mathcal{N}$ is non-empty. Is this necessarily undesirable? Why should we insist on partitioning the data set into so-called "clusters"? Is it not acceptable that some data points are so noisy that it is better to put them aside for further analysis or consider them "outliers"? Without going beyond the scope of this thesis, we simply point out that step 3(a-c) can be modified to further study the finer structure of the data, such as clusters within clusters.

Having stated our criterion for cluster validity, i.e. deciding on the number of clusters, we observe that our approach is rather different from previous approaches. Previously, one had to iterate between cluster centers and data partition, or continuously update the probability of associating the data points to the cluster centers by iterating the cluster centers. We have been able to achieve several goals at once in the new framework: finding the optimal number of clusters, the cluster centers and the corresponding partition of the data set.

2.6 Melting Algorithm

We have taken an elaborate and twisted exposition in explaining our formulation for clustering. To put everything together and without further repetition, here is the final version of our clustering algorithm:

Clustering by Melting

1. Choose β_{max}; β_{max} is a number related to the dynamic range and an assumed noise in the observations;

2. set $i = 1$, $\beta_1 = \beta_{max}$;

3. let every data point be a cluster;

4. iterate according to the mapping (2.9) N times or until the clusters converge;

5. record the cluster centers;

6. compute the FFE $M(\beta_i)$ of each cluster;

7. if more than two clusters share the same center, merge the data associated with the original clusters. This will become the set of data associated with the winning cluster;

8. create a *node* denoted by (y, β) in any of the following cases:

   - if the difference in the winning cluster's FFE exceeds a threshold;
   - if one cluster has a good figure of merit and the other cluster also has a good figure of merit and both clusters have more than d points;
   - if one cluster has good figure of merit and the other cluster is hierarchical;
   - if the winning cluster does not have a good figure of merit and is not hierarchical, but the resulting cluster has a good FFE and has more than d data points;

9. update the robustness of the winning clusters according to definition 7;

10. $i = i + 1$, $\beta_i = \beta_{i-1}/1.05$;

11. if there is more than one cluster, go to 4;

12. first decide upon all the good clusters among the nominal clusters; denote the set by $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$;

13. initialize $\mathcal{U}$ to an empty set;

14. while $\mathcal{T}$ is nonempty, do the following:

    (a) pick the element T_k in $\mathcal{T}$ with the biggest robustness measure;

    (b) put this element into $\mathcal{U}$;

    (c) remove T_k itself, the elements in $\mathcal{T}$ which are contained in T_k, and the elements in $\mathcal{T}$ which contain T_k;

15. collect all the data points which do not belong to one of the clusters in $\mathcal{U}$ into a set $\mathcal{N}$, which we hope is empty.

2.6.1 Computational Complexity and Other Observations

What is the computational complexity of this algorithm? One should note that theoretically the convergence to a fixed point takes forever. In real life, we always set a criterion for deciding convergence. We do know, however, that the convergence is exponentially fast because of the basins of attraction of the fixed points. At the initial stage, there are a huge number of clusters. But the dynamics is very fast when the temperature is low. Let us in particular look at the one-dimensional case as shown in Figure 2.1 which is plotted with a fairly low temperature. The map looks almost like a staircase with the data points sitting at the different levels. The gradients at the fixed points are almost zero. Thus we see that the convergence is extremely fast. Besides, most clusters will merge quickly because, at fine scales, the clusters are not "real." Many are there just because the temperature is too "cold." We remark that in our simulations, the schedule for heating is taken to be $\beta_i = \beta_{i-1}/1.05$. Melting is not sensitive to the exact schedule. One can certainly vary the schedule according to one's need. This is another advantage over annealing where this is a critical issue [23].

Before we move on to the next section, we collect and emphasize the properties of our proposed clustering algorithm:

1. At $\beta = \infty$, every data point becomes a cluster itself;

2. At $\beta = 0$, there is only one cluster;

3. The attracting dynamics of the fixed points implies that the clusters are either merged or separated. The tree partitions the set of data points into clusters. Therefore, there is some similarity between our algorithm and the agglomerative techniques. But the clusters merge in a very different manner;

4. The basins of attraction may change with scale. There may be a few data points associated with the current cluster which may not lie in the basin. But we never intend to partition our data according to the basins of attraction anyway;

5. The number of data points merged into a cluster is a measure of the *a priori* probabilities of that cluster. There is no need to compute the probabilities of the associations of the data points with the clusters;

6. There is a lot of local computation in the algorithm from the locality of the weighting function (one could then use tricks such as thresholding to reduce the amount of computation when the scale is fine);

7. The algorithm is ideal for parallel implementation because the calculations are local and the clusters are independent of one another.

We would like to briefly contrast this algorithm with the competitive learning paradigms [24, 25, 26] in neural network research. One central problem there is to find adaptive mechanisms whereby the neurons can tune to the features present in the inputs. Very often the models have competitive interactions among the neurons which try to learn the "clusters." As a result of this competition, we see that effectively the neurons will be partitioning the space into different categories. It is very likely that they will suffer the same fallacy which has troubled the other clustering algorithms. This may hinder the capture of the true category/concept one hopes the neurons can learn. An analogy with real-world politics is this: too much competition may lead to unnatural borders which can lead to ethnic unrest and war, but not to peace.

2.7 Versatility of the Clustering Algorithm

Now we will address the issues of variability of cluster densities, cluster sizes and cluster shapes. Cluster density refers to the number of data points in a cluster while cluster size refers to the volume of the smallest convex set which encloses the data points of a cluster, their convex hull. Cluster shape refers to the shape of this convex set.

If each cluster is of the same size but the number of data points in each cluster is different, our old analogy of looking at the data through a lens indicates that our formulation is insensitive to such variations. Suppose on the other hand the number of data points in each cluster is the same, but the sizes of the clusters are different; then by scaling the data, one can get the data down to the same size. This operation is naturally captured by performing clustering in the scale space. Thus each cluster will appear over its own range of scales. This is to be expected and can be handled by our formulation since we have the dendrogram at hand. We will demonstrate this in the simulations presented in the next section.

Now consider the case when the clusters have different shapes. We restrict ourselves to ellipsoidal shapes only. One approach is to include an adjustable metric into the formulation. For example, the cost function can be

$$e(x) = (x - y)^t A(x - y) \tag{2.24}$$

where A is a symmetric and positive definite matrix. Since A is not known *a priori*, one can force another constraint, $\det(A) = 1$, in the minimization formulation in Section 2.3. Going through the algebra, one obtains an equation similar to equation (2.8), except that one now has to update A as well.

There are three-fold difficulties with this approach. First, the computation is much more involved. Second, the interactions of the clusters in the scale space become less obvious. If each cluster has its own metric, it is seeing a different set of data from the others. Incorporating A is equivalent to rotating and rescaling the data. It is known [19] this will cause some data points, originally not clustered, to form a cluster. Third, the estimation of A is sensitive to scale and the shape of the clusters. At fine scales, the relatively smaller number of data points seen by a cluster means that A may not reflect the true A around the cluster. Undesirable distortion introduced at the early stages may alter a final partition obtained unfavorably. Similarly, if the features are correlated[5], A may be near-singular. But since we force $\det(A) = 1$, in some directions the data will be so stretched/shrunk that the distortion introduced to the data set will cause unreliable results. This observation also poses a difficulty

[5]Feature selection can be a tough problem. So it is not guaranteed that all features are uncorrelated.

in the FMLE formulation [8].

Fortunately, we find that, in practice, we do not need to resort to such complicated schemes. Let us review the fixed-point equation again:

$$y = \sum_x \frac{xe^{-\beta(x-y)^2}}{\sum_x e^{-\beta(x-y)^2}}. \tag{2.25}$$

Suppose each cluster consists of data coming from a source corrupted with noise which has a unimodal distribution with zero mean[6]. This means that if we sample enough data points, the density of the data points is a decreasing function of the distance from the true signal. But due to insufficient sampling or pure random fluctuation, this density will not be monotonically decreasing. However, if we smooth it enough, it will ultimately be monotone. What this implies for the above equilibrium equation is that as one increases scale, sooner or later a cluster will see that its center cannot be balanced about its original position due to the monotonicity. It will try to "swim" towards the gradient until balance is reached. Nowhere is the balance possible other than the neighborhood around the true signal.

Hence, even without a norm which is biased in the different directions, there is a built-in dynamics in the formulation to take care of such variations. This is demonstrated in the examples.

2.8 Simulation Results: Synthetic Data

In this section, we demonstrate the power of the proposed algorithm with examples. To make it easier to visualize the results, we mostly test the scheme on 2-dimensional data. In all the plots, the word "iteration" refers to the number of times the temperature has been increased. Temperature is plotted logarithmically. The inverse temperature β is lowered by a factor of 1.05 after each iteration.

When we plot the trajectories of the fixed points, the x and y components are plotted separately since the data is two-dimensional. All the data is generated from normal distributions, not necessarily with the same variances. A "cross" denotes the

[6]This is not an unreasonable assumption.

center of the distribution as seen by the computer. A "circle" denotes the representative of a cluster. Here, a cluster representative is just the arithmetic mean of the data in a given cluster, which is in some sense the best estimate if the statistics of the data are unknown. Note that this is slightly different from the cluster center as computed by the algorithm, which varies with β. We make the distinction just for the sake of visualization and comparison. For all the plots which have the iteration as the horizontal axis, only the relevant information when the number of clusters is less than 10 is shown (to avoid too many curves for the plotter).

Example 1. Ability to handle variable cluster densities: Figure 2.5a shows the data which consists of two clusters of the same variance. One cluster has 80 points and the other has twice as many. Figures 2.5b-c show the trajectories of the fixed points as β decreases. The number of clusters stays at two for the largest number of iterations. Figure 2.5d shows the clustering when the most robust clusters are chosen as shown in Figure 2.5b. Figure 2.5e shows the plot of the relative free energies. One can indeed see two robust branches which correspond to the two clusters.

From Figures 2.5b-c, we see that cluster 1 disappears first instead of cluster 2 even though the data comes from the same normal distribution. This is because as density goes up, equation (2.18) can be satisfied at a higher temperature. We also note that the cluster centers are discontinuous whenever two clusters merge, agreeing with our claim that the dominant form of bifurcation is of the saddle-node type. Moreover, we see some "outliers" which remain singletons for a large number of iterations.

Example 2. Ability to handle variable cluster sizes: In Figure 2.6a, we have two clusters, each containing 80 data points, but the variance of the second cluster is 3 times that of the first one. Hence the sizes of the clusters are different. Figures 2.6b-c show the trajectories of the fixed points as β decreases. We see that cluster 2 appears at a coarser scale. This is natural because its size is larger. This result agrees with our predictions. Figure 2.6d shows the clustering when the most robust clusters are chosen as shown in Figure 2.6b. The FFE's for this example are not plotted.

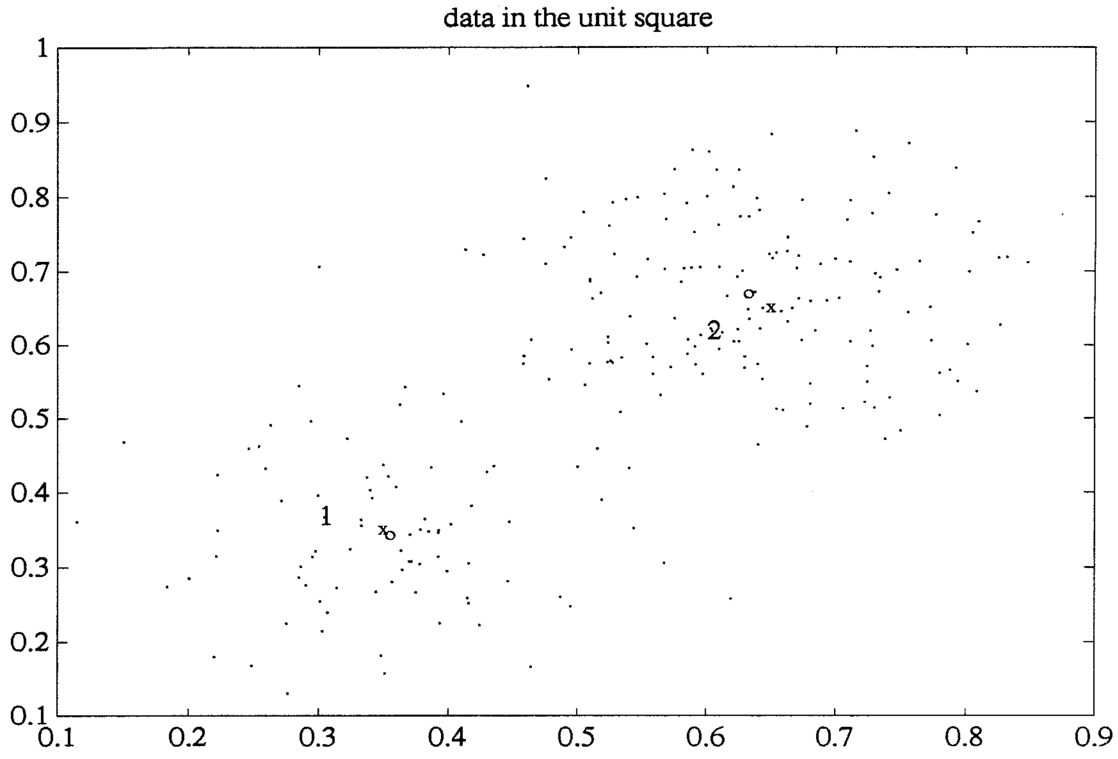


Figure 2.5a. Data and the computed clusters illustrating different cluster densities

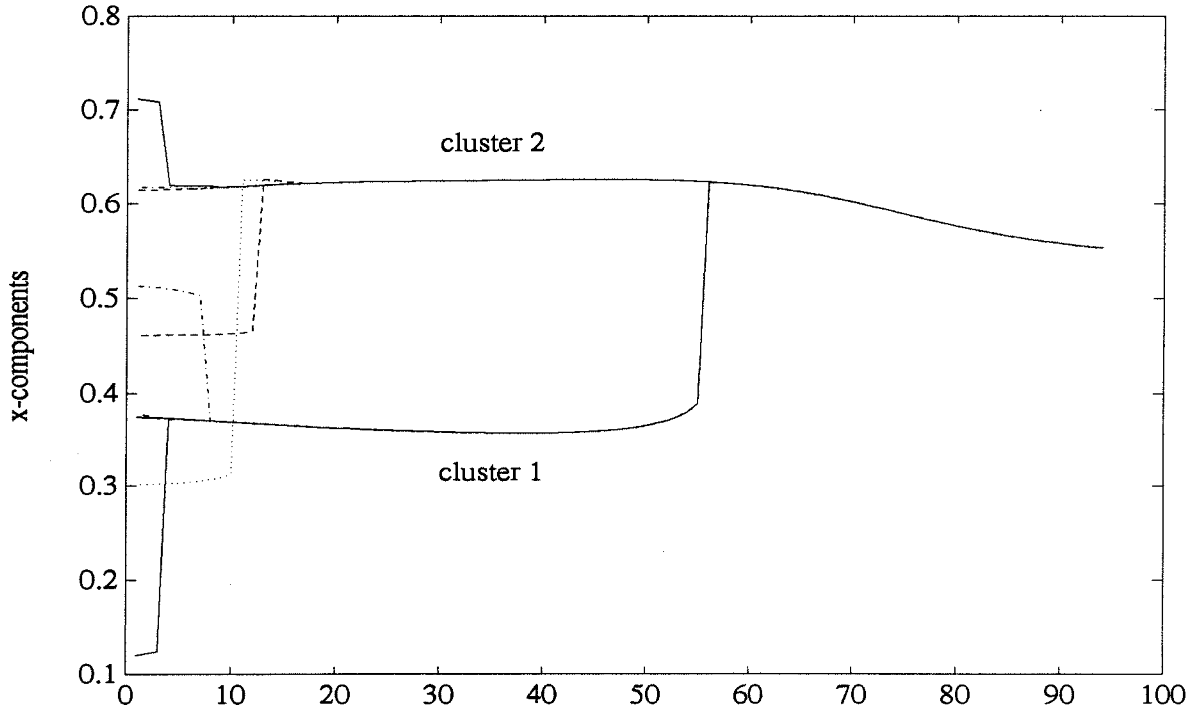


Figure 2.5b. x-components of the trajectories of the cluster centers versus iteration(T)

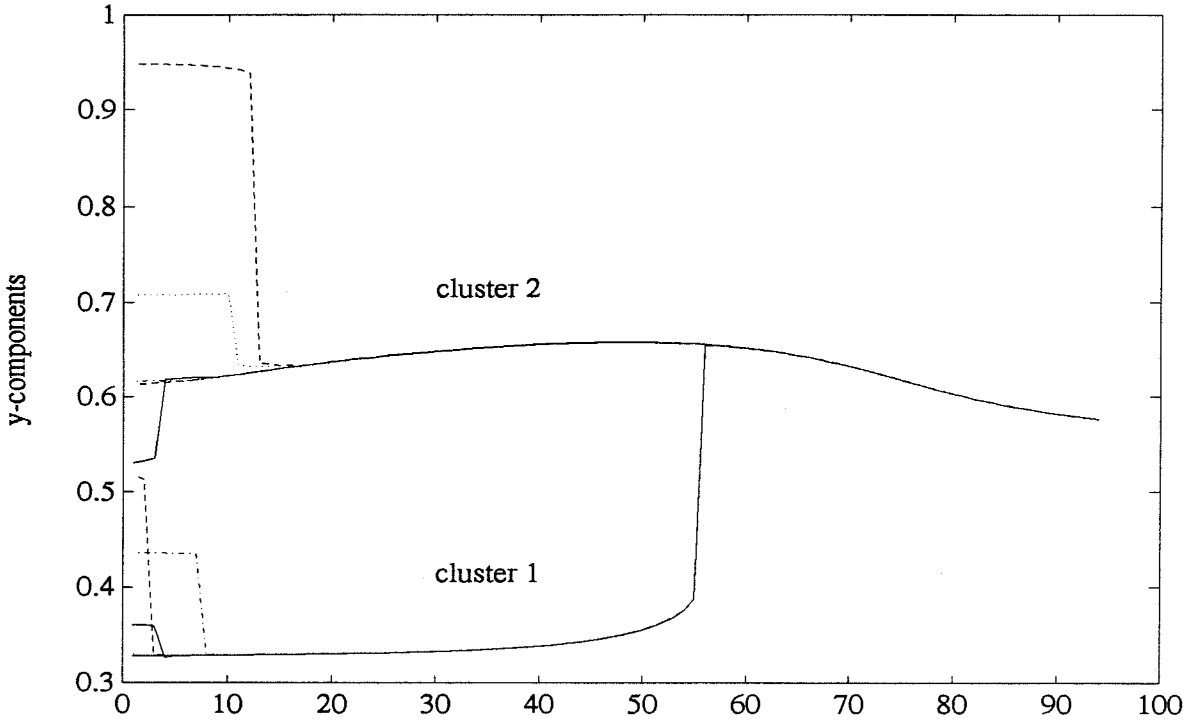


Figure 2.5c. y-components of the trajectories of the cluster centers versus iteration(T)

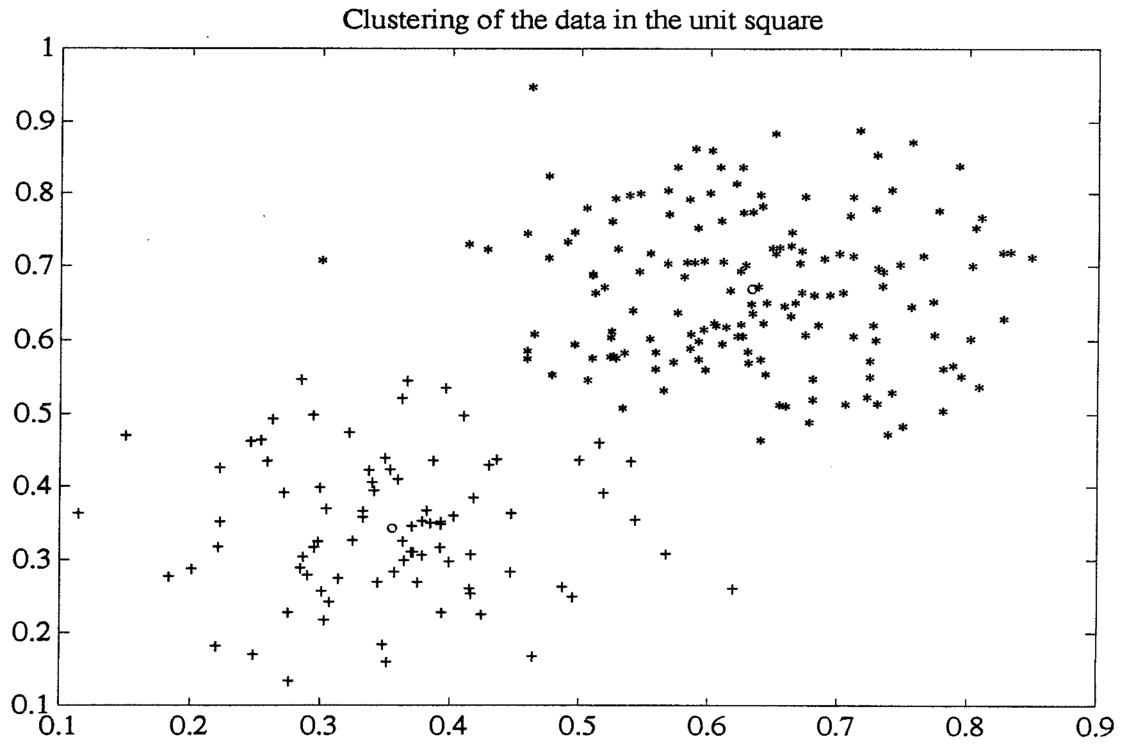


Figure 2.5d. Clustering with the choices of the clusters as shown in Figure 2.5b-c

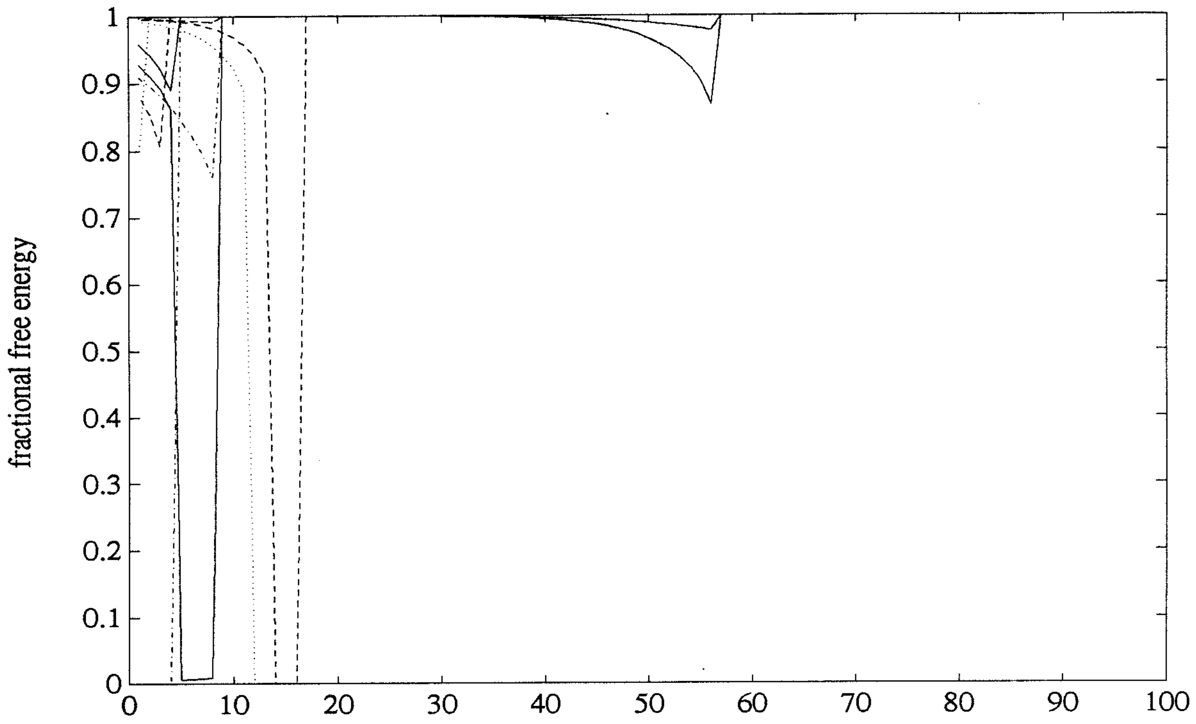


Figure 2.5e. Plots of FFE's of the fixed points for the data shown in Figure 2.5a

Example 3. Ability to handle many clusters: We generated four clusters with noise added from the same Gaussian distribution. Each cluster has 80 points. The data plotted in Figure 2.7a shows considerable overlap among the first two clusters. In fact, it is difficult for human eyes to tell if there are three or four clusters. Figures 2.7b-c show the trajectories of the fixed points as β decreases. Note that there is a range of scales (iterations $\approx$ 45 to 65) over which there are only three clusters in the data because of the merging of the first two clusters. But our algorithm can indeed separate the data into four clusters; Figure 2.7d shows the clustering thus obtained. Figure 2.7e shows the FFE's for the clusters. Note that as scale becomes coarser, it is the third and the fourth clusters which lose their stability to the cluster obtained by combining the first two.

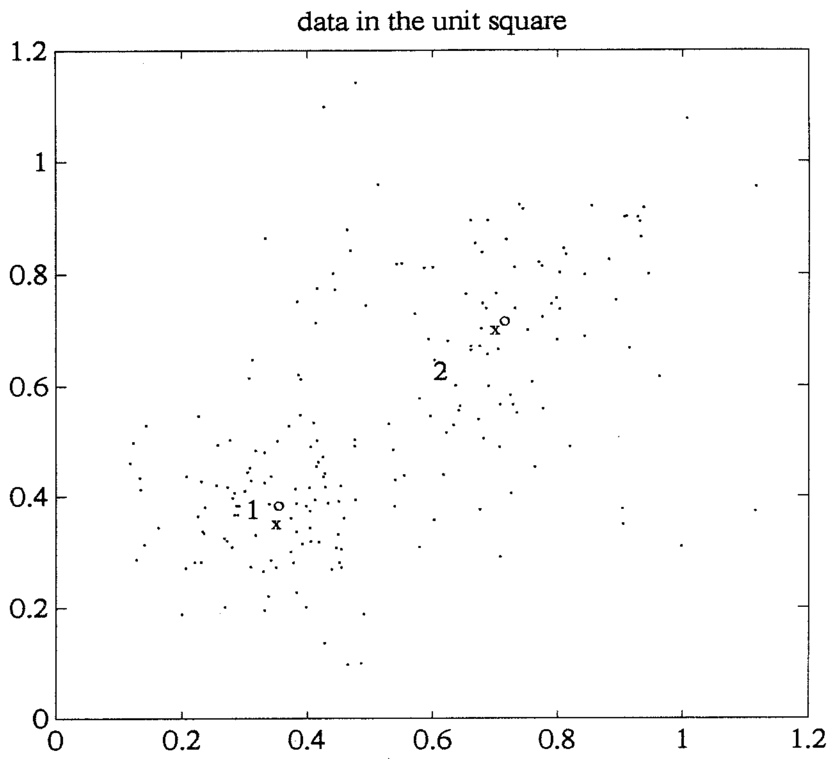


Figure 2.6a. Data and the computed clusters illustrating different cluster sizes

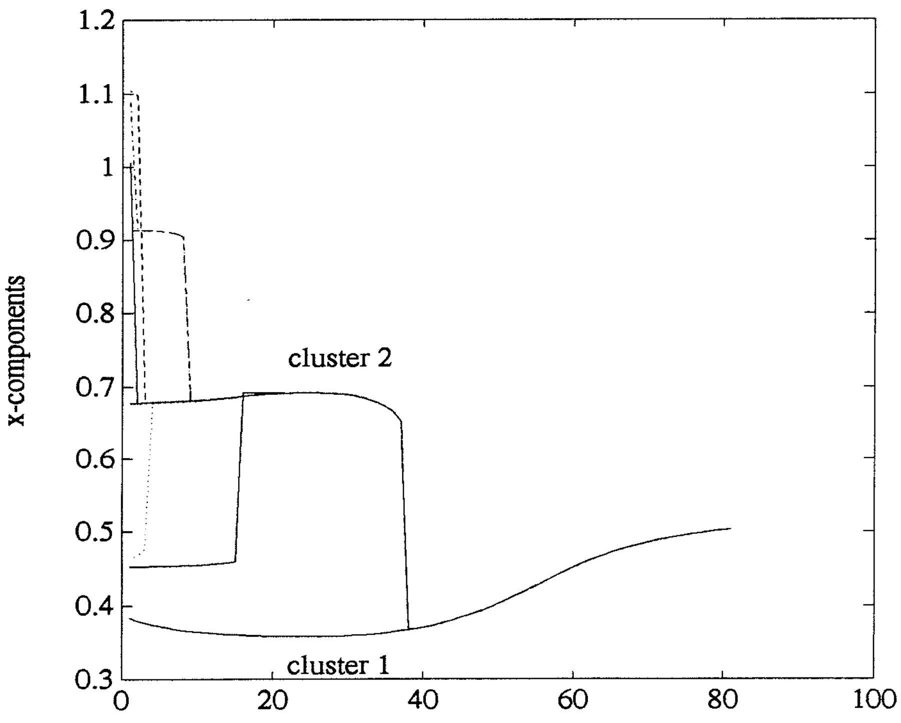


Figure 2.6b. x-components of the trajectories of the cluster centers versus iteration(T)

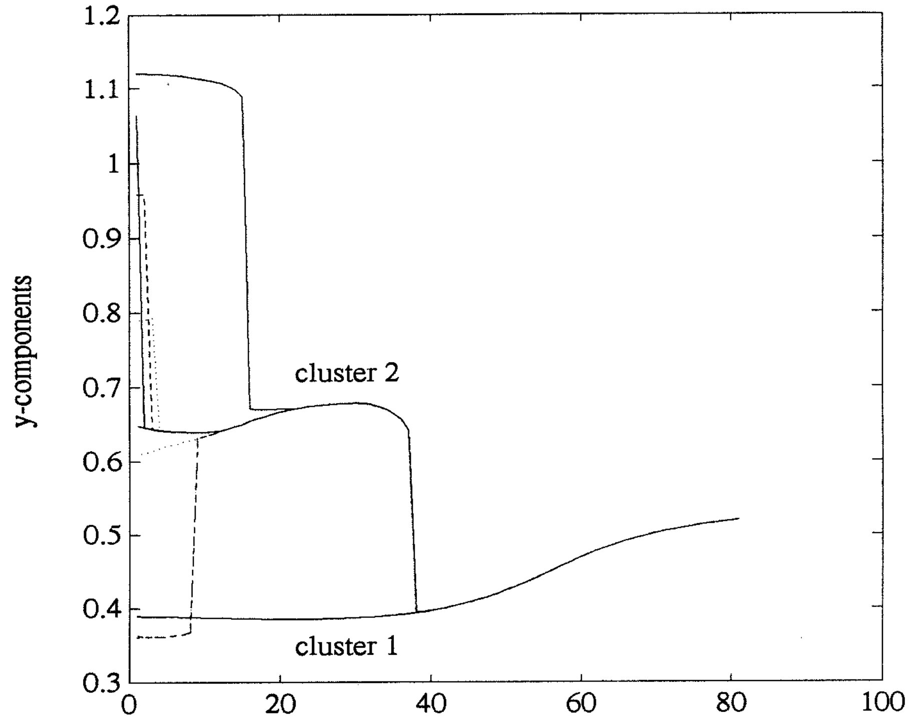


Figure 2.6c. y-components of the trajectories of the cluster centers versus iteration(T)

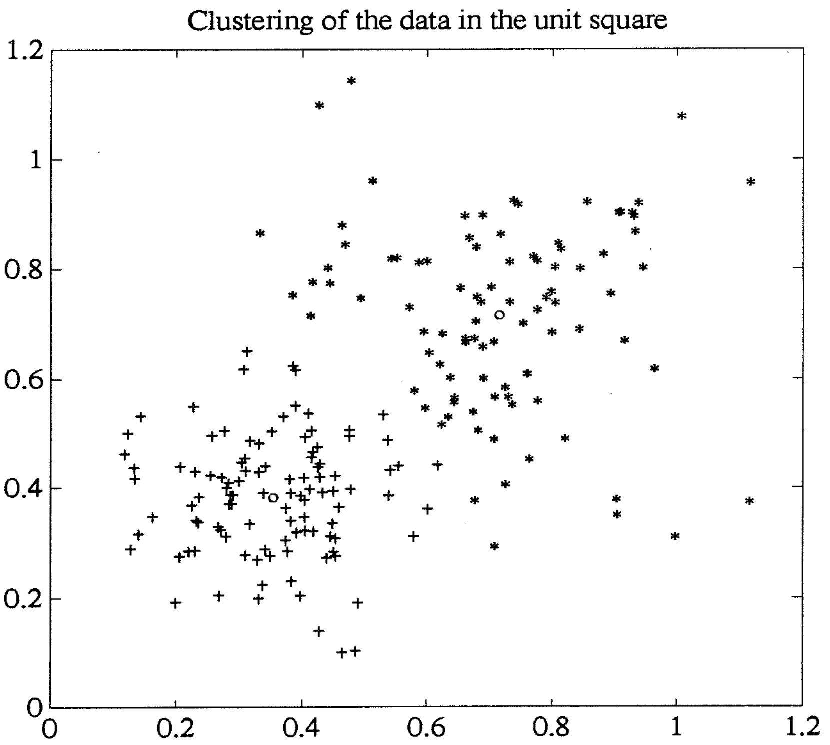


Figure 2.6d. Clustering with the choices of the clusters as shown in Figures 2.6b-c

Example 4. Ability to handle clusters of different shapes: Figure 2.8a shows a data set consisting of two sources with different noise levels, i.e., clusters of different shapes. Figure 2.8b plots the trajectories of the fixed points as β decreases. Again, we see two distinct clusters, which is also evident from the figures of merit shown in Figure 2.8c. This shows that our algorithm is at least capable of handling clusters of different shapes.

Example 5. Ability to handle overlapping clusters of different shapes: Figure 2.9a shows a data set consisting of four sources with various levels of noise. The third cluster is oriented at 45 degrees. Figure 2.9b shows the trajectories of the fixed points. Figure 2.9c is the plots of the FFEs for the clusters; it clearly shows that there are four clusters. Figure 2.9d shows the partition obtained by the algorithm. Note the few data points marked by 'O's. These are the data points which are grouped into clusters different from those generated by the computer. The few data points on the upper center are clearly far away from the oriented cluster. So it is acceptable that they are grouped to a different cluster closer to them.

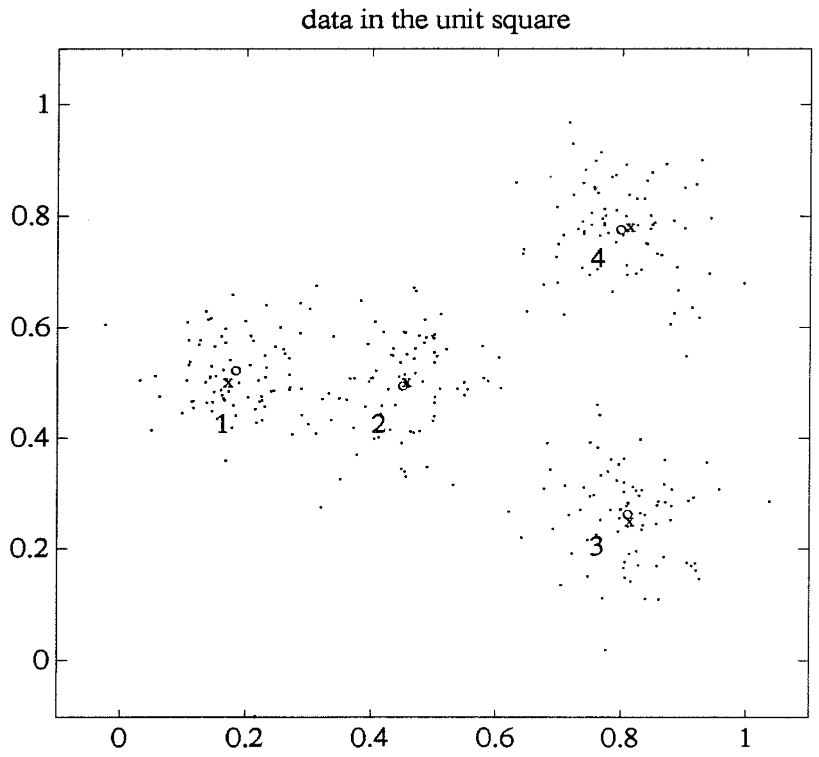


Figure 2.7a. Data and the computed clusters illustrating ability to handle many cluster sizes

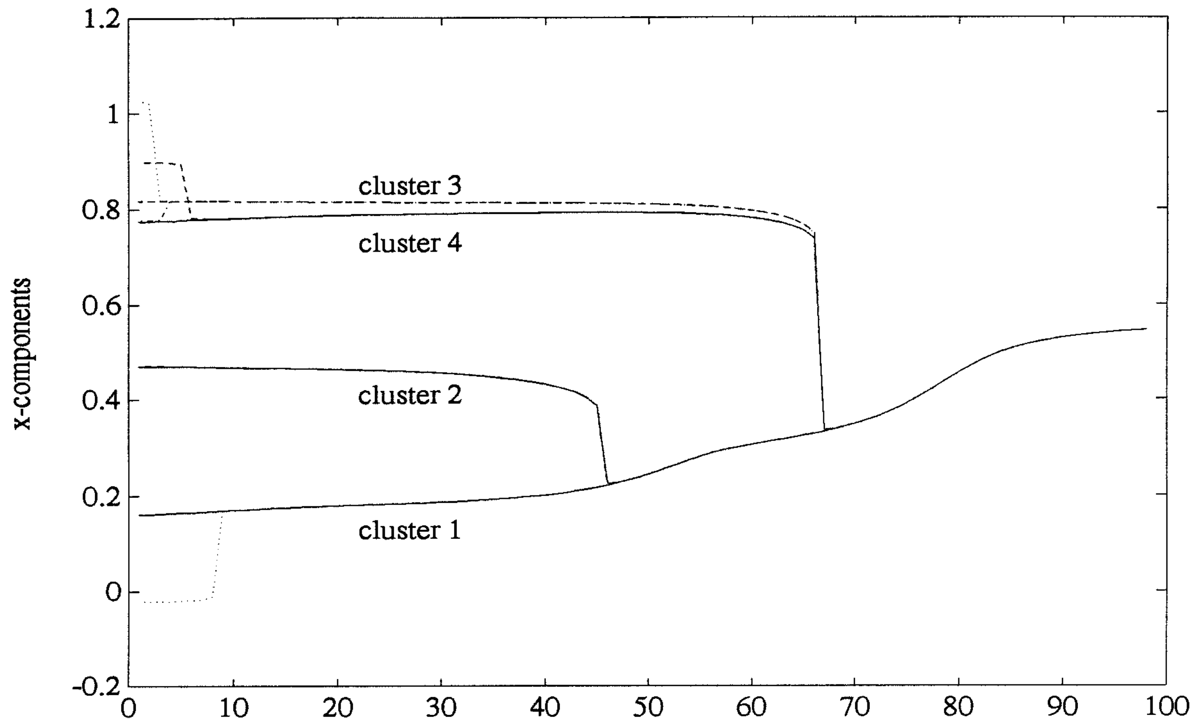


Figure 2.7b. x-components of the trajectories of the cluster centers versus iteration(T)

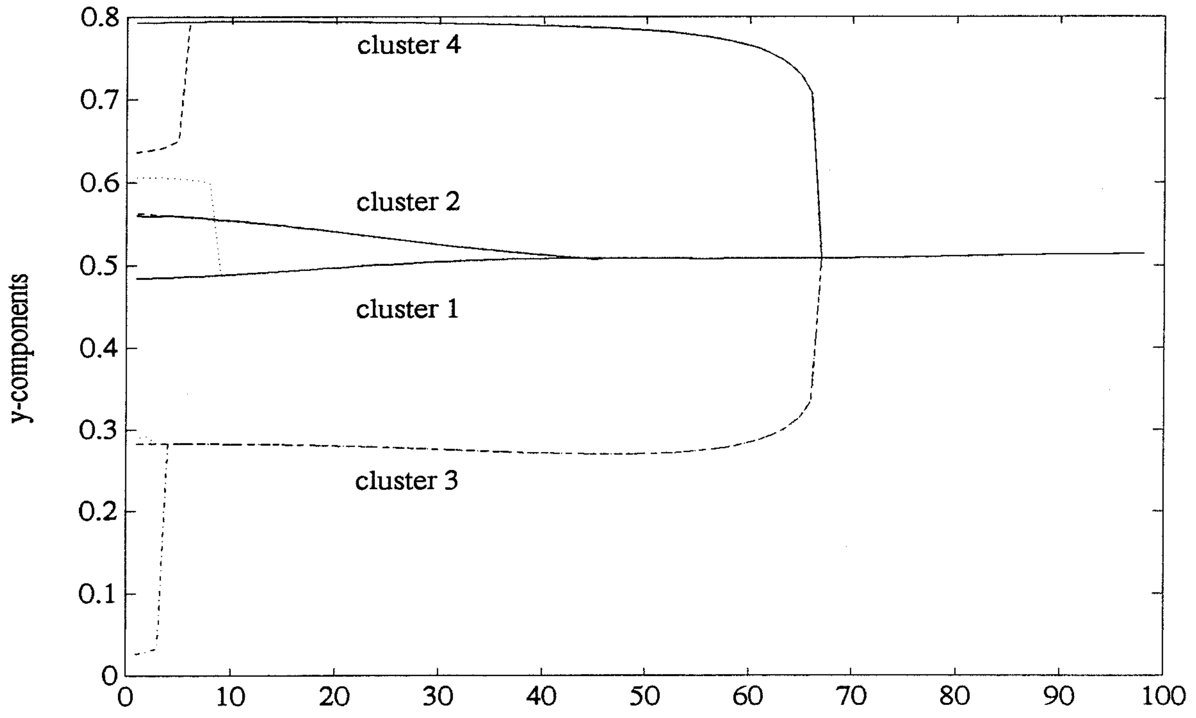


Figure 2.7c. y-components of the trajectories of the cluster centers versus iteration(T)

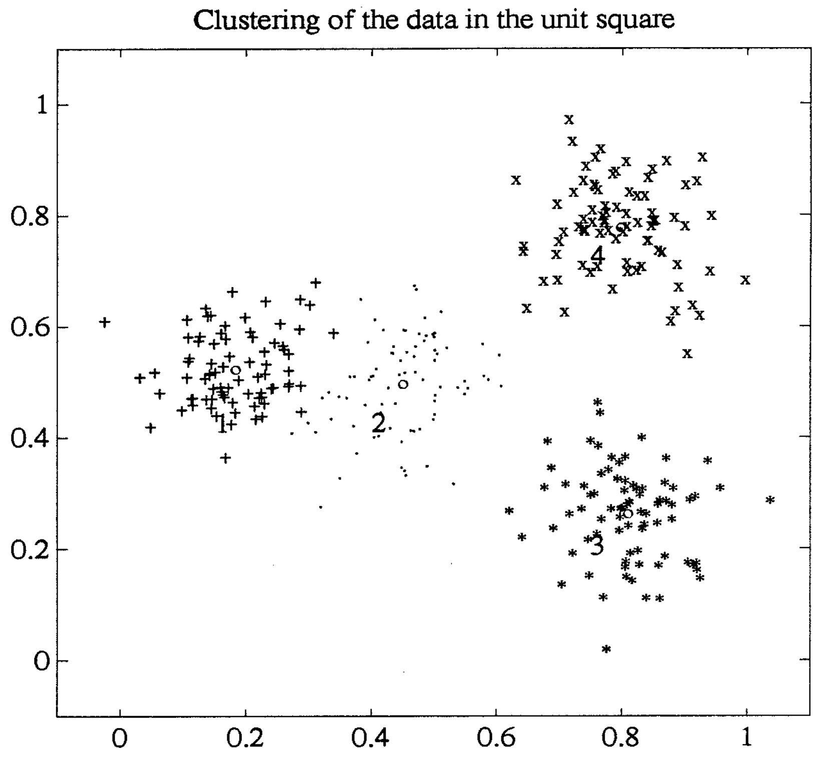


Figure 2.7d. Clustering with the choices of the clusters as shown in Figures 2.7b-c

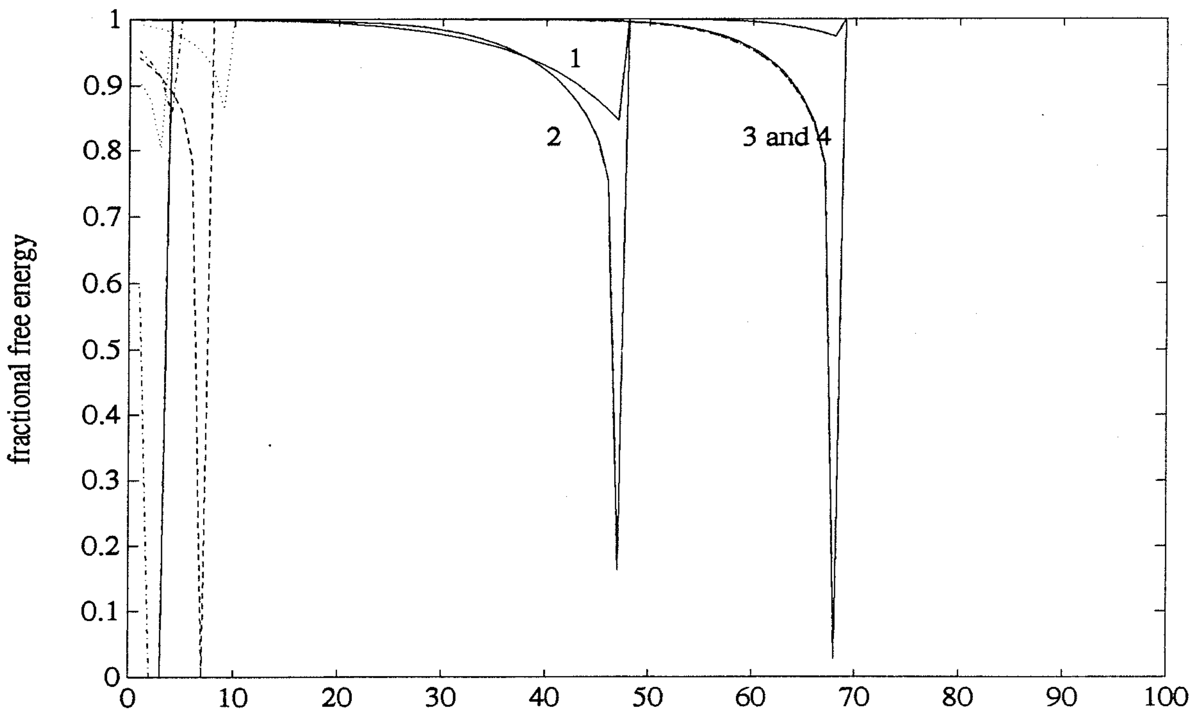


Figure 2.7e. Plots of fractional free energies for the data points in Figure 2.7a

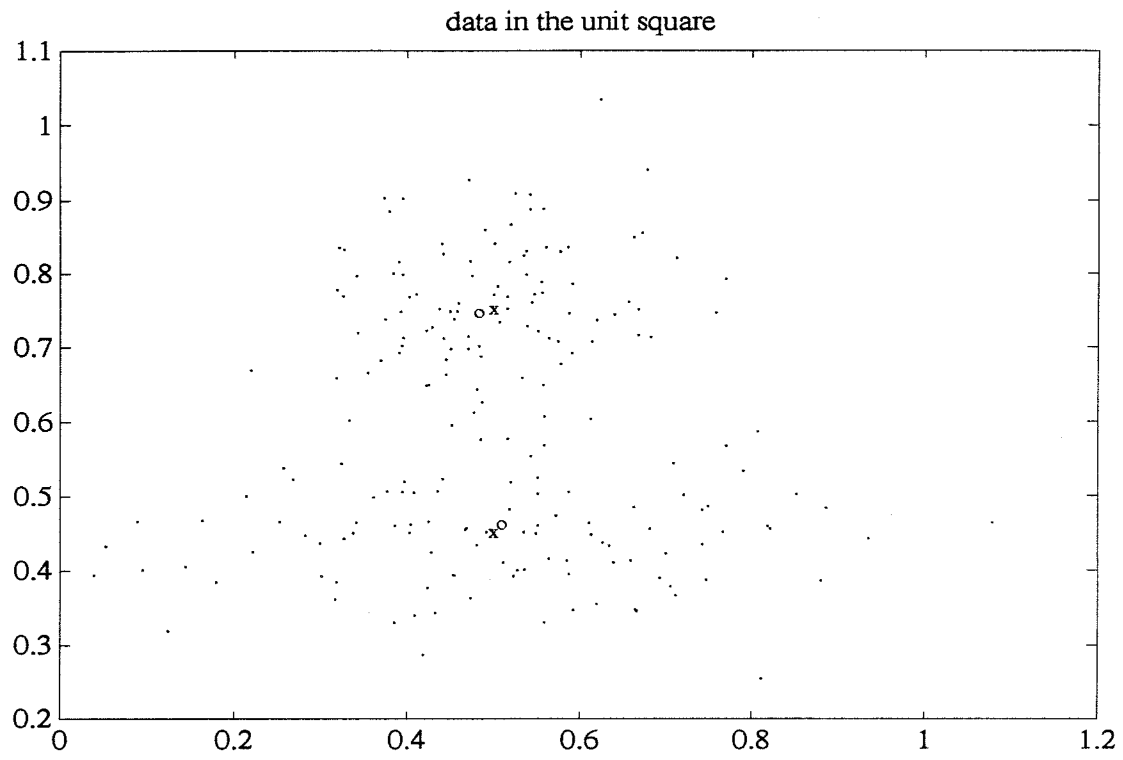


Figure 2.8a. Data and the computed clusters illustrating ability to handle many cluster sizes

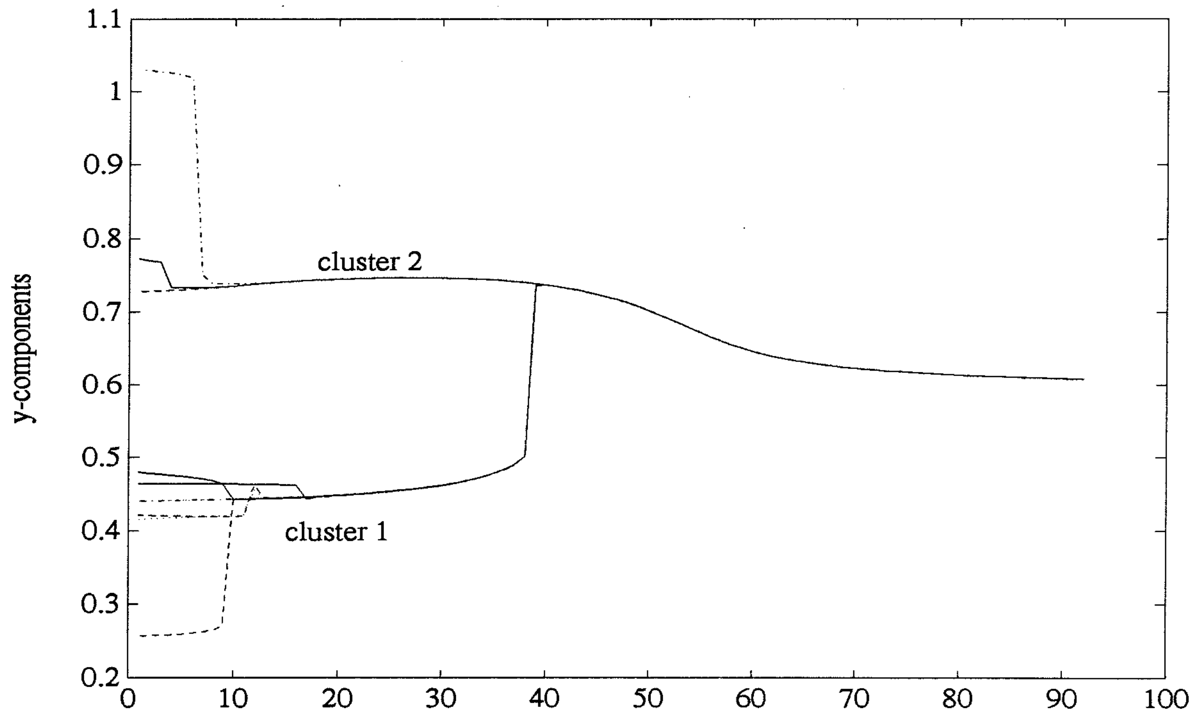


Figure 2.8b. y-components of the trajectories of the cluster centers versus scale

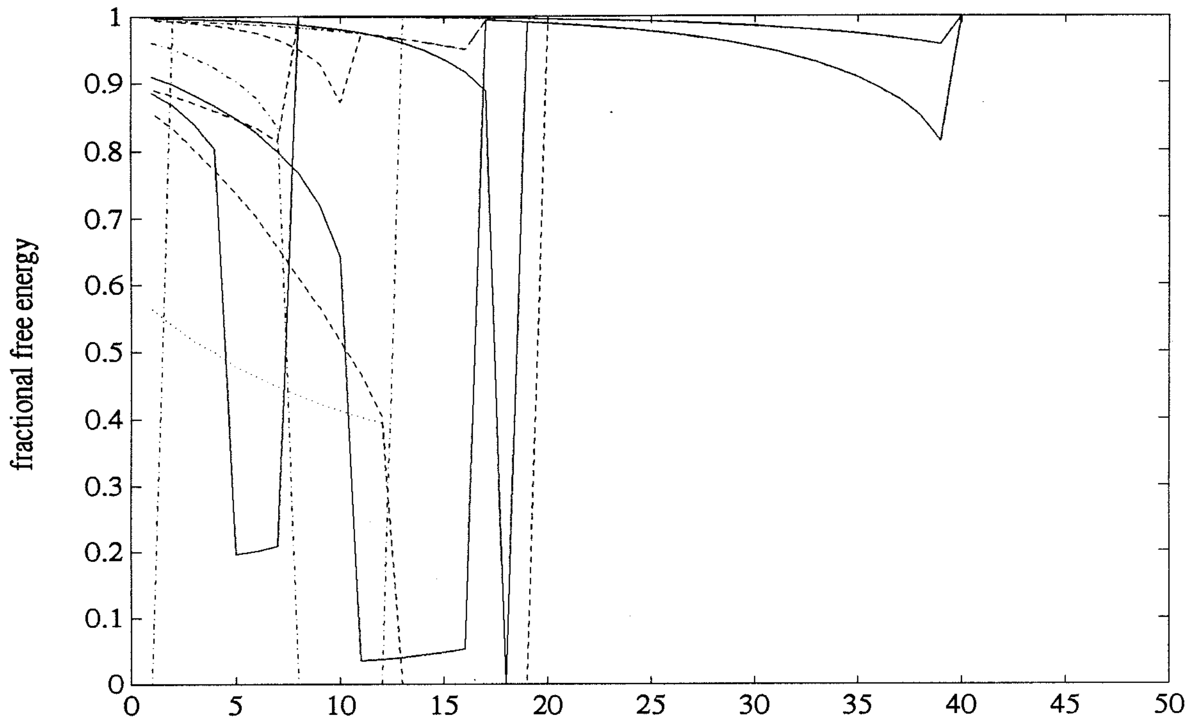


Figure 2.8c. Plots of FFE's of the clusters for the data shown in Figure 2.8a

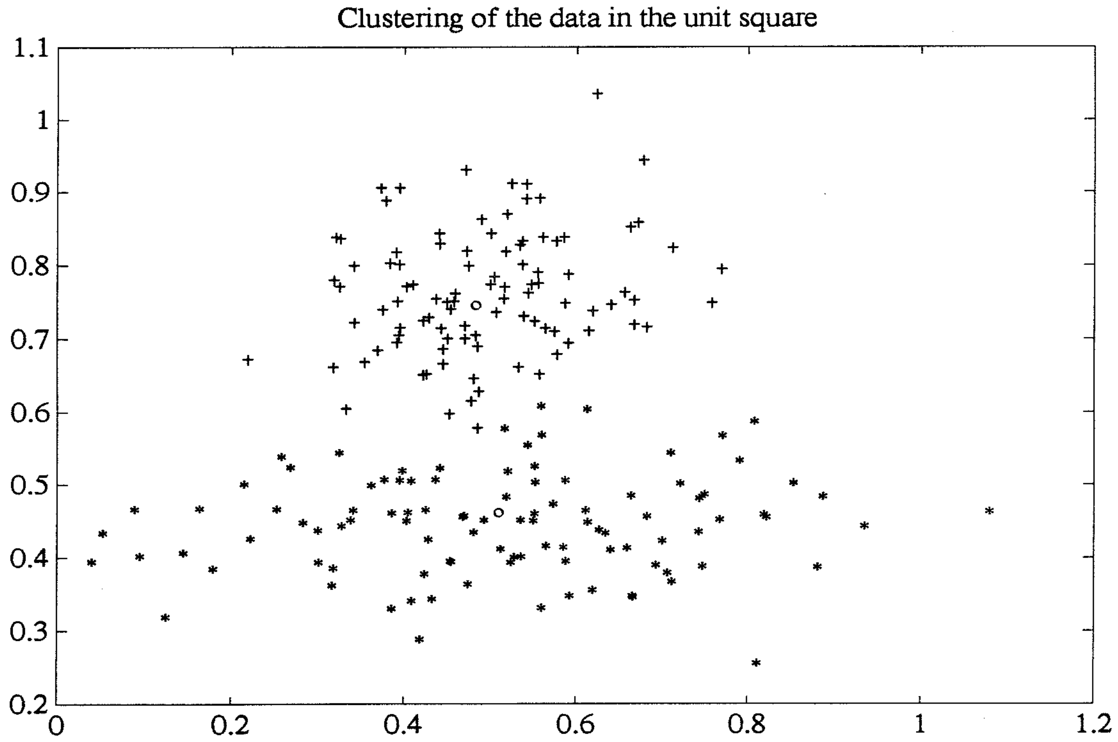


Figure 2.8d. Clustering with the choices of the clusters as shown in Figure 2.8b

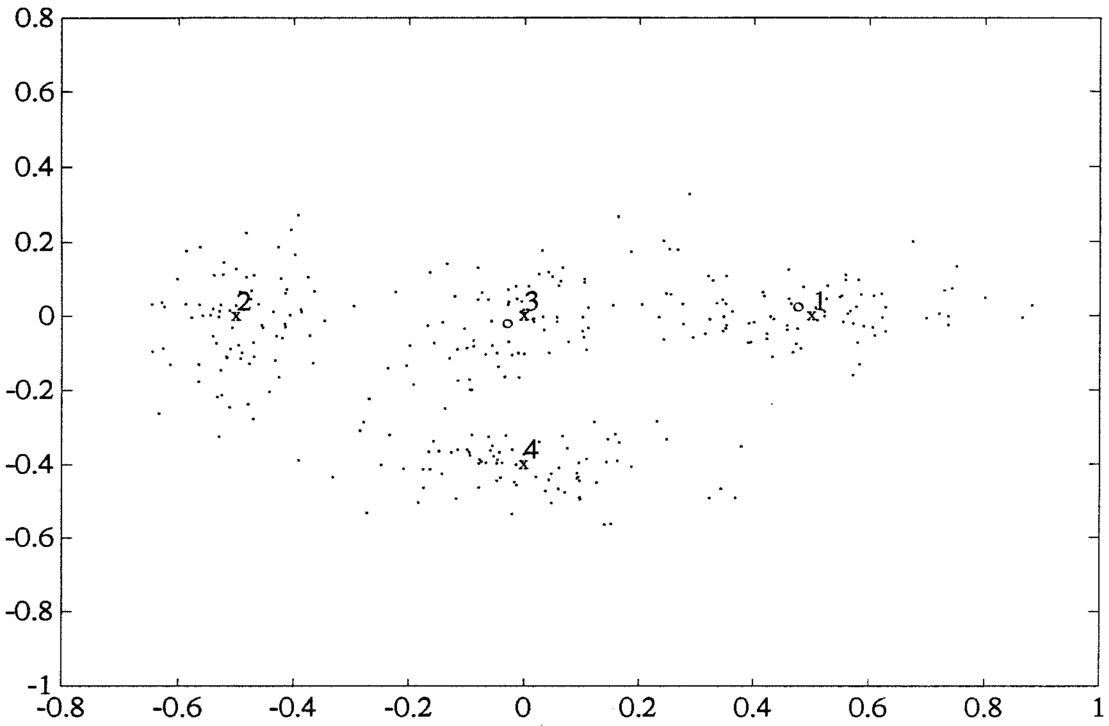


Figure 2.9a. Data illustrating ability to handle many clusters of different shapes and sizes

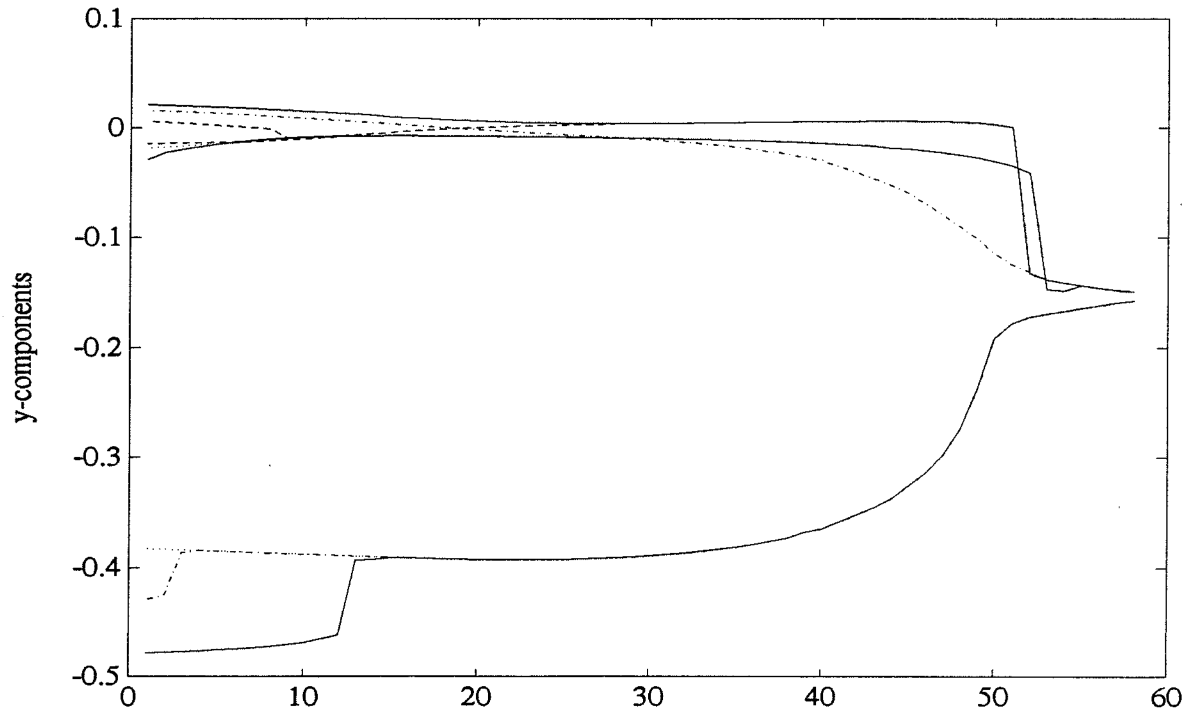


Figure 2.9b. y-components of the trajectories of the cluster centers versus iteration(T)

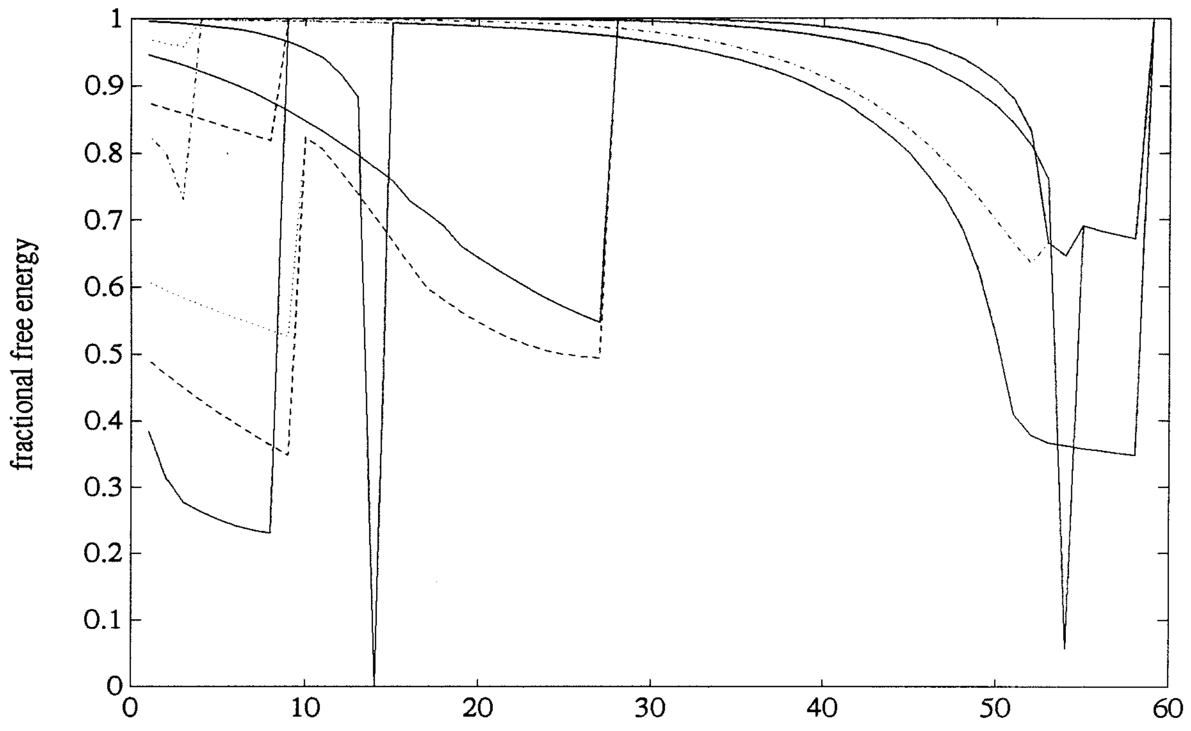


Figure 2.9c. Plots of fractional free energies for the data points in Figure 2.9a

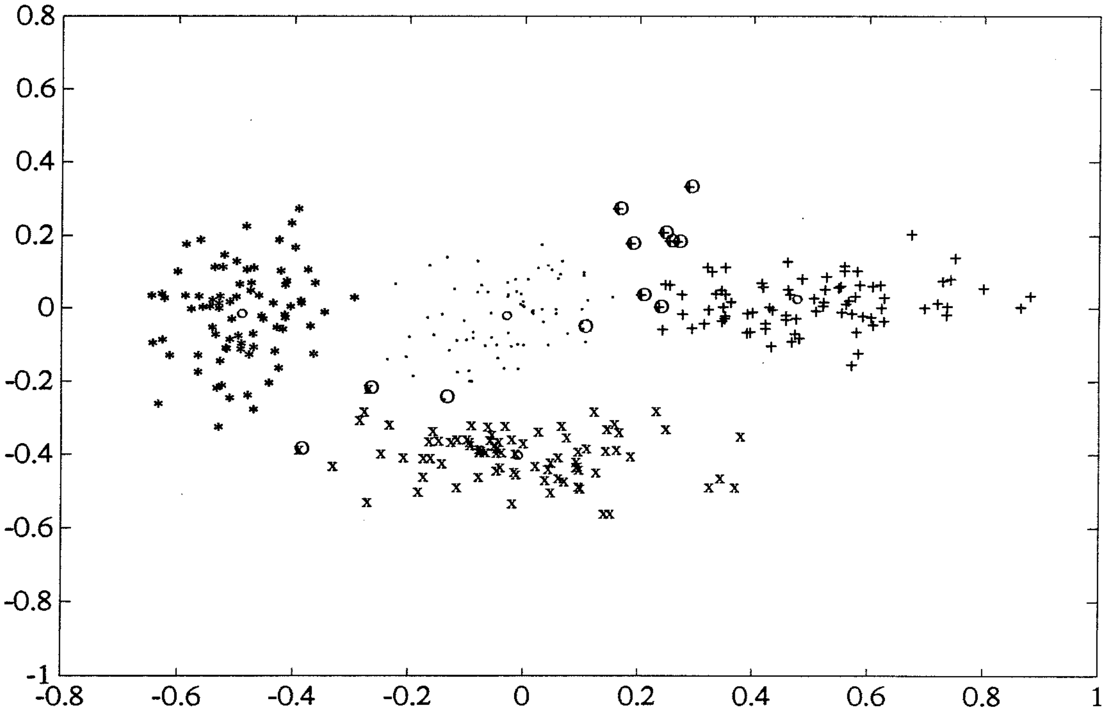


Figure 2.9d. Clustering of the data shown in Figure 2.9a. 'O' -- misclassified data points

2.9 Application to Clustering and Classification of Synthetic Aperture Radar Images

We briefly describe this real application. The details can be found in [28]. Radar imaging is gaining importance as a tool for earth and planetary studies [29]. In recent years, Synthetic Aperture Radar is becoming more and more popular due to its high resolution [30]. Often, SAR is multispectral and in any case gives a multi-dimensional image. In this section, we describe an application of our clustering algorithm to the clustering and classification of a SAR image of an agricultural area.

The particular image we use was collected from a DC-8 SAR [31] over an agricultural site named "Flevoland" in the Netherlands. Initially, polarimetric and radiometric calibrations were carried out using the procedures outlined in [32, 33]. Then the noise levels in the polarimetric data channels were estimated using techniques in [31]. After noise subtraction, one obtains the scattering matrix of the targets. One then extracts a set of parameters from this matrix which hopefully will be able to separate the different crops. We call this set of parameters the *features* for the pixels. The number of features is 12 or more in our applications.

If we plot the feature vectors for the pixels in the multi-dimensional space, we will observe cluster-like structures. This is because the pixels corresponding to the same crop should have similar backscatter characteristics and should form a cluster. The clusters are not expected to be of the same shape. They also have different sizes. If one can identify all the clusters, each cluster will give valuable information about the signature of the corresponding crop. For example, the average of the data points in a cluster gives the average backscatter characteristics of the crop. The shape of the cluster tells us the correlation between the different features. This can further yield the underlying chemical and physical properties of the crop or the medium over which the radar is transmitted and received. The ability to obtain the backscatter characteristics of different crops is clearly important for remote sensing for land use.

There is ground truth for some selected sites. In our simulation, we first obtained according to ground truth thirteen groups of 12-dimensional feature vectors, each corresponding to a different crop (one group is in fact water, but we call it a crop anyway). Each group has 36 vectors. Figure 2.10 shows the components of the set of feature vectors. In the plots, we concatenated the groups one after the other; each component is plotted separately. For each plot, we see that a feature is similar across the data points within each group while it varies from group to group. But there is not any clear separation between the different groups. In fact, the crops are all mixed up. Of course one reason to use multi-spectral representation is that hopefully in the higher dimensional space, the features separate.

In our simulation, we fed the data from the thirteen groups into the algorithm, but without the labels. Our algorithm was able to partition the data into thirteen clusters which correspond to the true partitioning in the data. Since it is difficult to read the curves of the FFE's for the clusters, we only plot the evolution of these thirteen clusters in Figure 2.11 (we only plot one component of the 12 features). One can see clearly how they merge. Note that one always observes a saddle-node bifurcation.

Having obtained the clusters, i.e. the labels for the data points, the next task is to design a classifier to classify the whole image. There are many methods to design the classifiers [19, 34]. We used a simple one. We computed the covariance matrices for the different clusters. The determinants of the matrices gave the noise (scale) of the clusters. We then normalized each covariance matrix such that its determinant was 1. For each new pixel, we computed the Mahalanobis distances [19]

$$d_i = \beta_i (x - y_i)^t A_i^{-1} (x - y_i) \tag{2.26}$$

where i indexes the crops, y_i is the mean vector, β_i is the determinant of the unnormalized covariance matrix and A_i is the normalized covariance matrix for the crop i. The pixel was then classified according to the minimum distance criterion. Figure 2.12 shows the image thus obtained. Note that A_i is different for different crops. Thus, there is no single metric for the multi-dimensional space. Each signal source generates data points according to its own noise levels and hence we can only estimate the metrics.

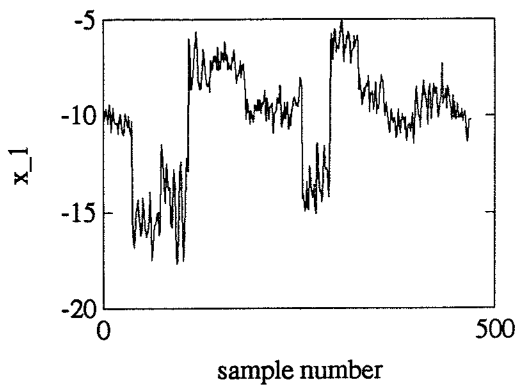
We compared our results to that obtained in [31]. There, the feature vectors for the known sites were plotted and certain salient feature and combinations were picked for each site according to their ability to separate the site from the others. A hierarchical set of rules was then constructed to classify the whole image. Figure 2.13 shows the classification obtained in [31]. Note that the image produced is very noisy. This suggests that the classifier is not able to classify all the pixels and possibly has classified many wrongly.

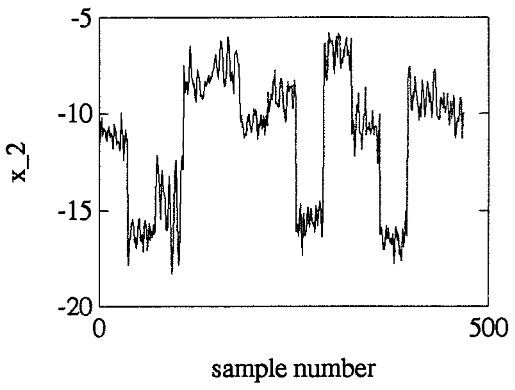
Without going into the details on the assessment of our results in [28] as compared to those obtained in [31], we just point out a few observations here:

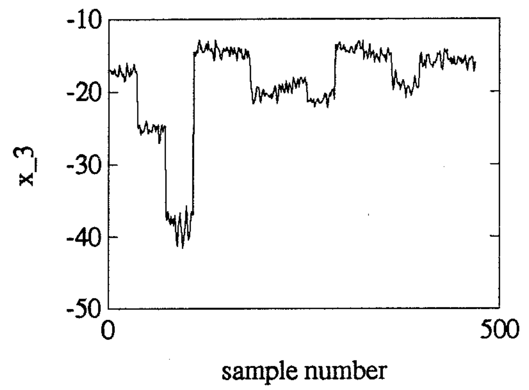
- Our classification produces an image which is much less noisy; it also produces many more homogeneous blocks which are more faithful to the original ground truth;

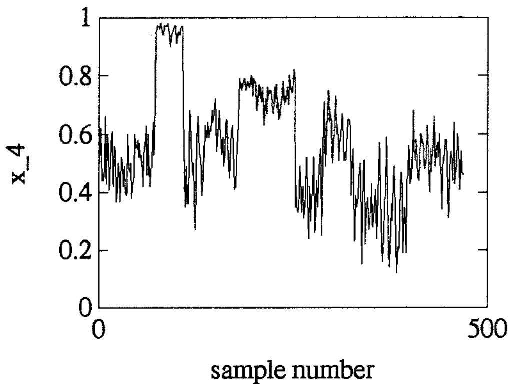
- Our classification produces inaccurate results for pixels corresponding to "water" because the features from water are very noisy and do not form clusters at all. Heuristic rules used in [31] perform better;

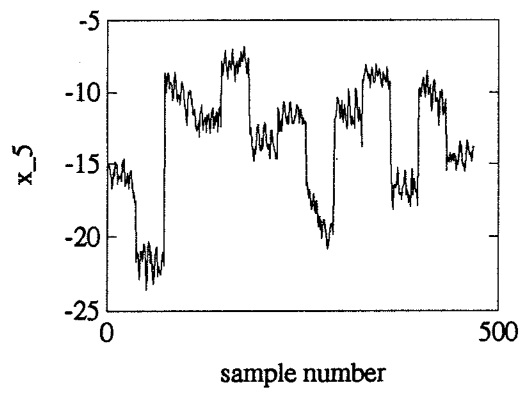
- Our algorithm is unsupervised; the only human knowledge required is the labeling of the clusters.

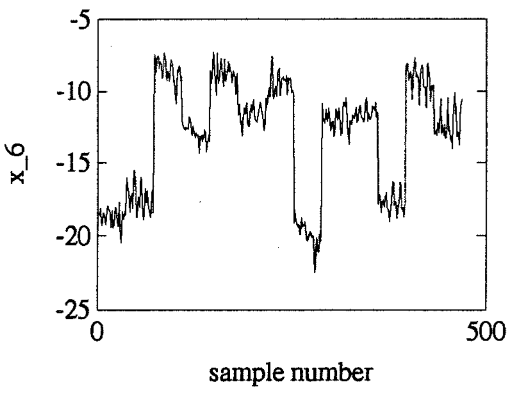

-5
-10
-15
-20
0
500
x_1
sample number

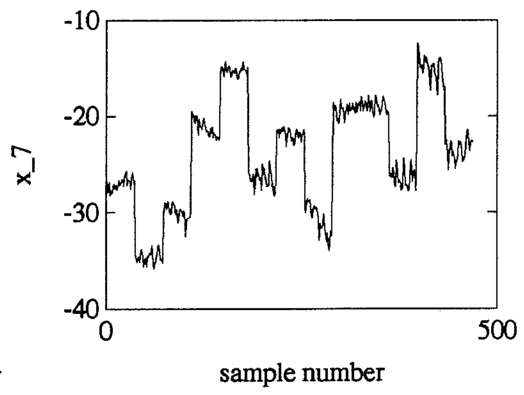

-5
-10
-15
-20
0
500
x_2
sample number

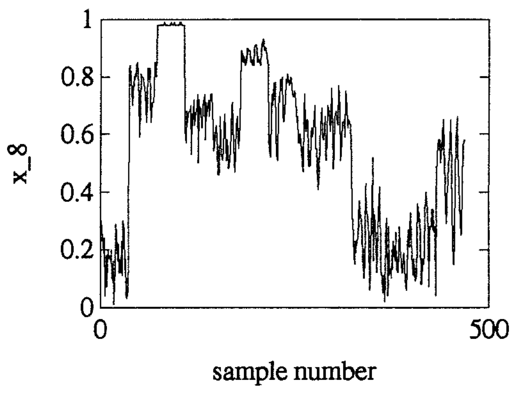

-10
-20
-30
-40
-50
0
500
x_3
sample number


1
0.8
0.6
0.4
0.2
0
0
500
x_4
sample number


-5
-10
-15
-20
-25
0
500
x_5
sample number


-5
-10
-15
-20
-25
0
500
x_6
sample number


-10
-20
-30
-40
0
500
x_7
sample number


1
0.8
0.6
0.4
0.2
0
0
500
x_8
sample number

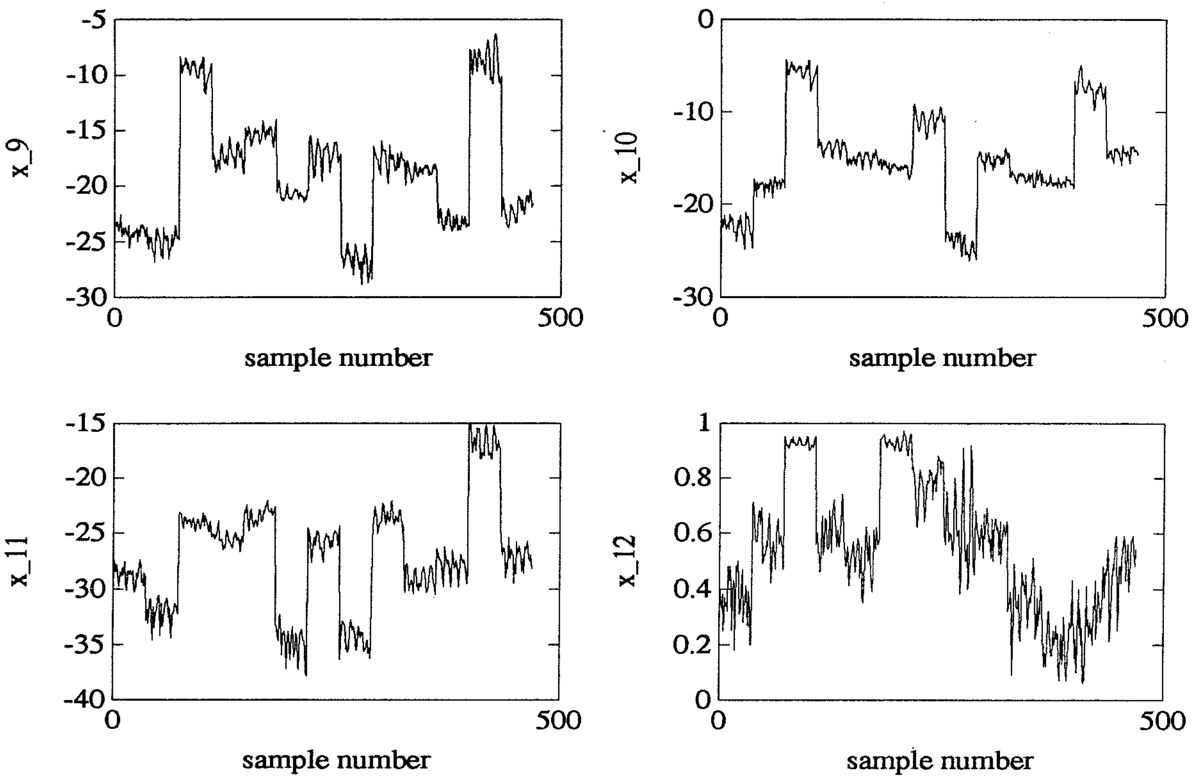


Figure 2.10. Each graph plots one feature for the whole data set.

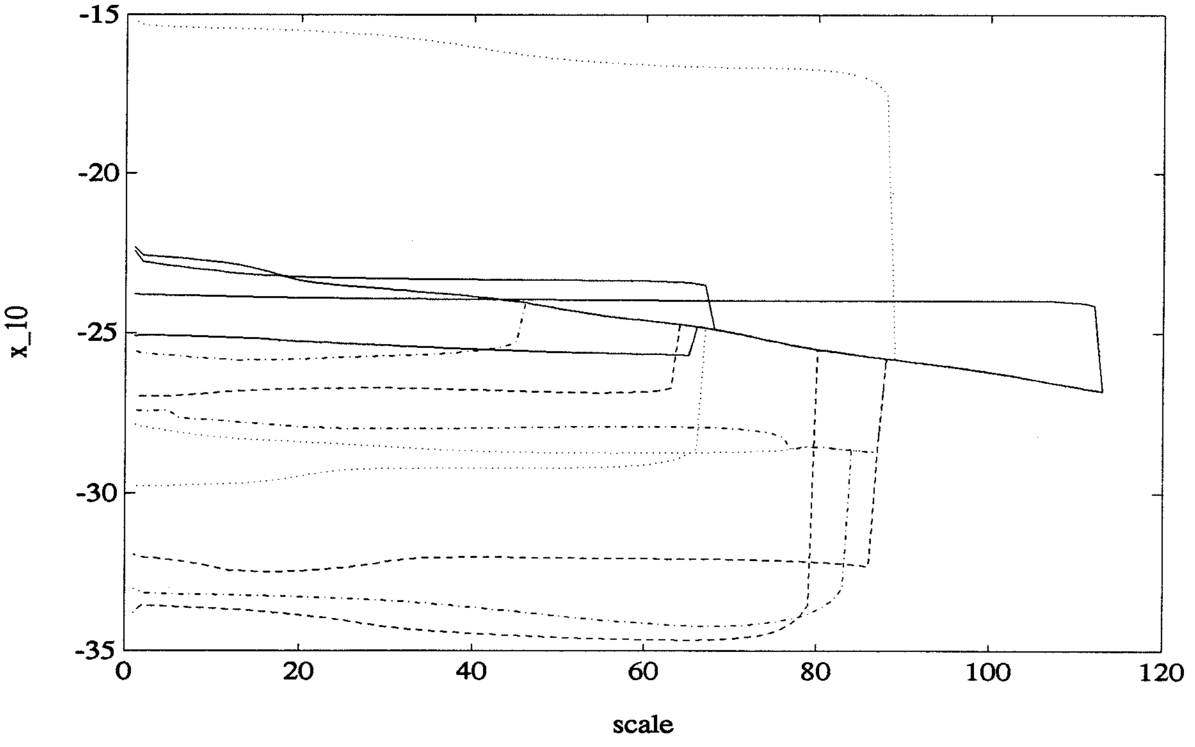


Figure 2.11. Trajectories of one component of the thirteen clusters in the scale space

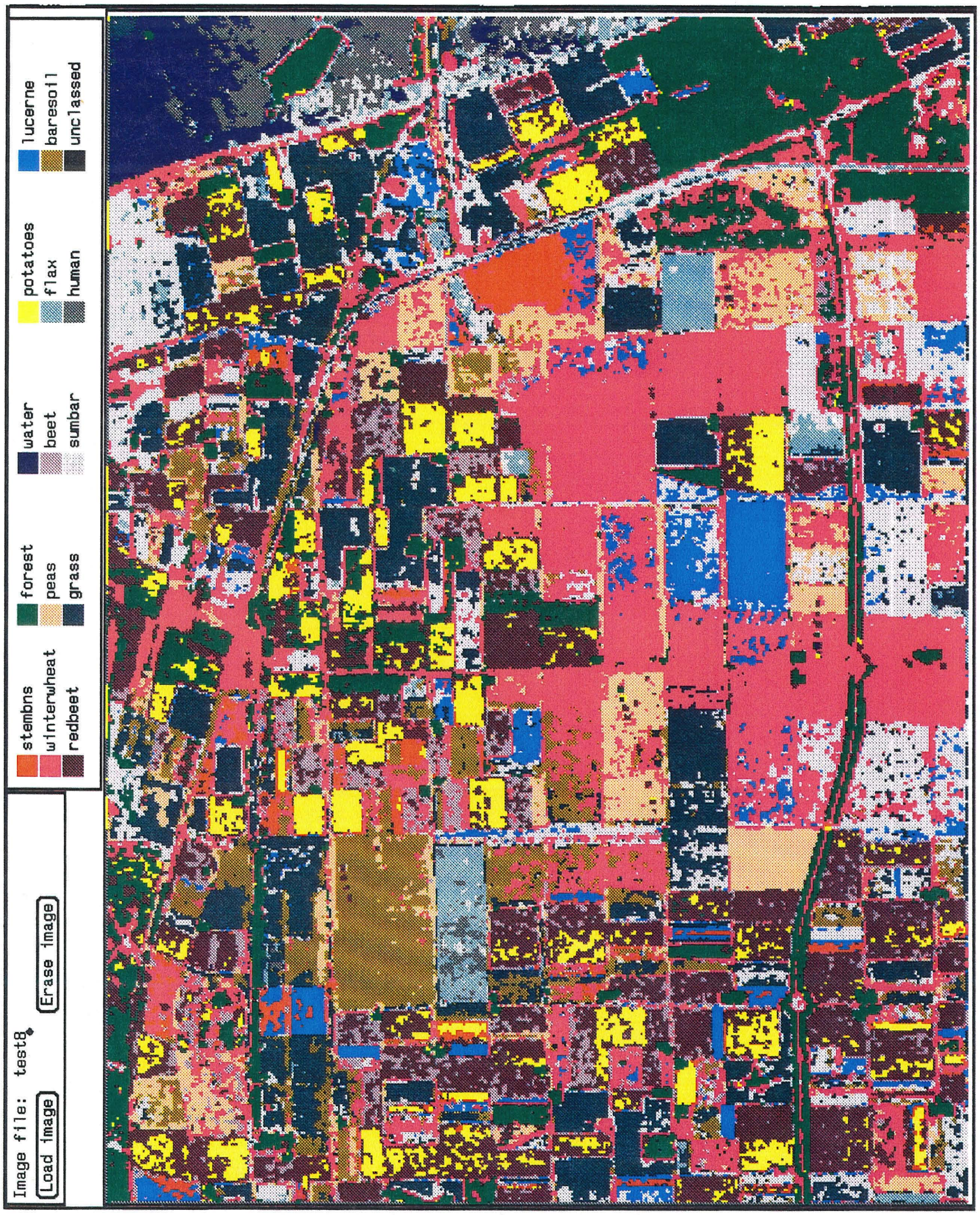


Figure 2.12: Classification obtained of the agricultural area by our algorithm.

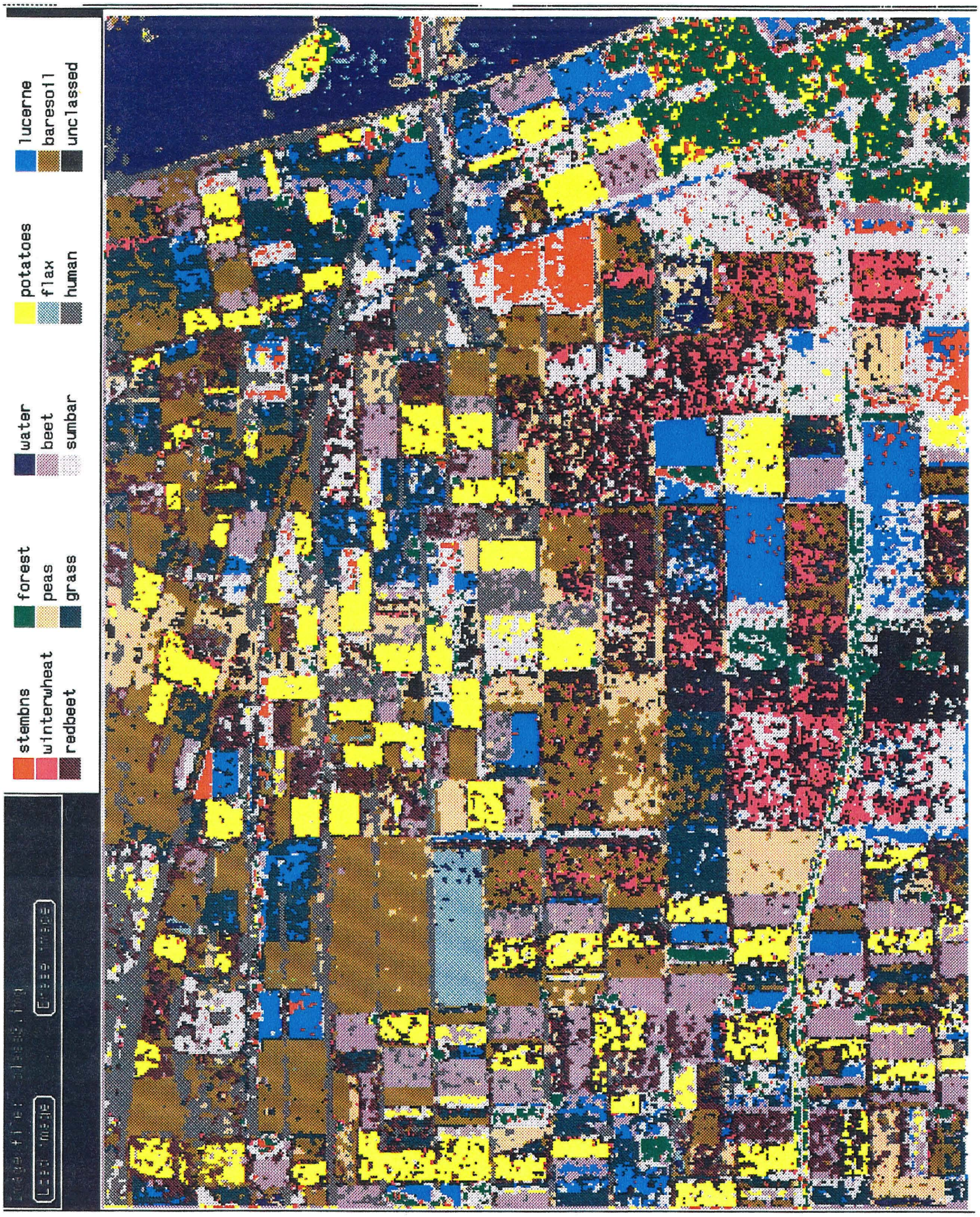


Figure 2.13: Classification obtained of the agricultural area by hierarchical rules.

2.10 Summary

Clustering is a hard problem. The traditional clustering algorithms suffer from several difficulties. First, they usually find only local minima of the functionals they are trying to minimize. Second, they cannot deal effectively with overlapping clusters, nor with clusters with various densities, sizes and shapes. Third, no prior algorithm knows if its solution is any good. We believe that these manifest difficulties are due to a lack of precise understanding about clusters. Such lack has prevented the recognition of random points among natural clusters. As a consequence, there have been no satisfactory utility functions proposed to measure how good the clustering results are. The willingness of existing algorithms to partition any set of data suggests that they are more suitably named "partitioning" algorithms rather than "clustering" algorithms.

In this work, we have proposed a new way of treating the clustering problem. We pointed out that the notion of scale has been recognized by other researchers, yet it has not been properly incorporated in any of the other algorithms. We also pointed out the notion of cluster independency which has not been formally recognized by prior researchers. Yet its importance is so crucial that it not only sheds new light on the clustering process, but also permits the natural application of the maximum entropy principle.

The maximum entropy principle allows us to connect our new formulation with statistical mechanics. Cluster centers correspond to the local minima of the thermodynamical free energy, which are just the fixed points of a one-parameter nonlinear map. By using bifurcation techniques, we are able to obtain a complete picture of the clustering process. We also derived a new clustering algorithm based on melting instead of annealing. Hierarchical clustering is achieved by merging the clusters via saddle-node or pitchfork bifurcations. We introduced a new characterization of cluster based on this formulation. We showed that it has the ability to determine the number of clusters in the data. The algorithm can also account for variabilities in cluster densities, cluster sizes and cluster shapes. We tested this algorithm with many examples, both 1-D and 2-D. We further tested this algorithm on a real application: crop identification of a SAR image of an agricultural site, and found great improvement over existing methods. The computational complexity is minimal and the algorithm is ideal for parallel implementation.

Since clustering is a form of unsupervised learning, we expect this work should provide some new insights for neural network research, too, but will not discuss that here. As for future work, there are many directions. We only mention one here. For example, we may ask, can we adapt our algorithm so that it can track the changes in the signals in a nonstationary environment? In the appendix, we show some preliminary work which suggests that the answer is positive.

2.11 Appendix A: Nonlinear Dynamics of the Formulation in Higher Dimensions

We study the dynamics of the following map:

$$y \overset{f}{\longmapsto} y + \sum_x \frac{(x-y)e^{-\beta\|x-y\|^2}}{\sum_x e^{-\beta\|x-y\|^2}} \tag{2.27}$$

in higher dimensions. The Jacobian of the map f at a fixed point y_0 when $\beta = \beta_0$ is

$$D_y f_{|y=y_0} = 2\beta_0 \sum_x (x-y_0)(x-y_0)^t P(x). \tag{2.28}$$

Here, x, y_0 are d-dimensional vectors; $y_0 = (y_1, \ldots, y_d)^t$ and $x = (x_1, \ldots, x_d)^t$. Since this matrix is real and symmetric, we can diagonalize it by a rotation matrix V. The resulting diagonal matrix Λ consists of the eigenvalues of the Jacobian. We have $D_y f_{|y=y_0} = V\Lambda V^t$. Now multiply equation (2.28) by V^t on the left and by V on the right; we have

$$\Lambda = 2\beta_0 \sum_x V^t(x-y_0)(x-y_0)^t V P(x). \tag{2.29}$$

Since $P(x) = P(V^t x)$, we see immediately that we can rotate the coordinate system by the rotation matrix V^t. Under the new coordinate system, the Jacobian will be diagonal. Hence we can now assume $D_y f_{|y=y_0}$ is diagonal.

Since the Jacobian is positive-semidefinite at every point, the mapping is orientation-preserving. The free energy F acts as the Lyapunov function [14] for the mapping. The difference between successive y's is $\frac{-1}{2}\partial F/\partial y$. Thus, the y's march down the surface of the free energy and settle down in some local minimum. We can conclude that the mapping (2.27) exhibits no chaotic behavior.

Expand the map around the fixed point (y_0, β_0); we obtain

$$y_1 \mapsto \lambda_1 y_1 + b_1\beta + f_1(y_1, y_2, \ldots, y_d, \beta) \tag{2.30}$$

$$y_2 \mapsto \lambda_2 y_2 + b_2\beta + f_2(y_1, y_2, \ldots, y_d, \beta) \tag{2.31}$$

$$\ldots$$

$$y_d \mapsto \lambda_d y_d + b_d\beta + f_d(y_1, y_2, \ldots, y_d, \beta) \tag{2.32}$$

$$\beta \mapsto \beta \tag{2.33}$$

where $\lambda_i \leq 1$ and f_i contains second and higher order terms. The variables y_i and β are all small perturbations about the fixed point. The last equation is included [13] because we want to study the behavior of the map as β is varied around β_0.

If all the eigenvalues are less than 1, the mapping is stable and no bifurcation occurs. However, when $\lambda_i = 1$, the mapping undergoes bifurcation. To characterize the nature of bifurcation and the dynamics, we now invoke the Center Manifold Theorem (CMT) [35]:

Theorem 3 *Let $T : \mathcal{R}^{n+m} \mapsto \mathcal{R}^{n+m}$ have the following form:*

$$T(x, y) = \big(Ax + f(x, y), By + g(x, y)\big) \tag{2.34}$$

where $x \in \mathcal{R}^n, y \in \mathcal{R}^m, A$ and B are square matrices such that all the eigenvalues of A have modulus 1 and all the eigenvalues of B are inside the unit circle, f and g are C^2 (continuous derivatives up to first order); and f, g and their first order derivatives are zero at the origin. There exists a center manifold $h : \mathcal{R} \mapsto \mathcal{R}^m$ for T. More precisely, for some $\epsilon > 0$ there exists a C^2-function $h : \mathcal{R}^n \mapsto \mathcal{R}^m$ with $h(0) = 0, h'(0) = 0$ such that $\mid x \mid x < \epsilon$ and $(x_1, y_1) = T(x, h(x))$ implies $y_1 = h(x_1)$.

Essentially the CMT says when the mapping is near bifurcation, its local behavior is governed by the behavior of the map in the center manifold which is described by the graph h. It is generally possible using a power series expansion to approximate h as accurately as one wants. This greatly reduces the dimensionality of the problem. Now, let us apply the CMT to study the behavior of the map (2.27) at the bifurcation point.

Without loss of generality, let the first eigenvalue λ_1 be 1. We perform a simple transformation $y_j = \tilde{y}_j + \frac{b_j}{1-\lambda_j}\beta$ for $j > 1$ so that the CMT can be applied. Thus we have

$$\begin{aligned}
y_1 &\mapsto \lambda_1 y_1 + b_1\beta + g_1(y_1, \tilde{y}_2, \ldots, \tilde{y}_d, \beta) \\
\tilde{y}_2 &\mapsto \lambda_2 \tilde{y}_2 + g_2(y_1, \tilde{y}_2, \ldots, \tilde{y}_d, \beta) \\
&\ldots \\
\tilde{y}_d &\mapsto \lambda_d \tilde{y}_d + g_d(y_1, \tilde{y}_2, \ldots, \tilde{y}_d, \beta) \\
\beta &\mapsto \beta
\end{aligned}$$

where g_i's contains second and higher order terms only. The CMT allows us to restrict our attention to the first equation, which is

$$\begin{aligned}
y_1 \longmapsto\ & y_1 + b_1\beta + c_1 y_1^2 + a_{11} y_1 \beta + d_1 \beta^2 \\
& + \sum_{j>1} c_j(\tilde{y}_j + \frac{b_j}{1-\lambda_j}\beta)^2 + \sum_{j>1} d_j y_1(\tilde{y}_j + \frac{b_j}{1-\lambda_j}\beta) + \sum_{j>1} e_j(\tilde{y}_j + \frac{b_j}{1-\lambda_j}\beta)\beta \\
& + \sum_{1<j<k} c_{jk}(\tilde{y}_j + \frac{b_j}{1-\lambda_j}\beta)(\tilde{y}_k + \frac{b_k}{1-\lambda_k}\beta) \\
& + \text{third and higher order terms.} \qquad (2.35)
\end{aligned}$$

Now, the CMT says that we can approximate $\tilde{y}_j (j > 1)$ by the second and higher order terms in y_1 and β. That is,

$$\tilde{y}_j = \alpha_j y_1^2 + \gamma_j y_1 \beta + \eta_j \beta^2 + \text{higher order terms.}$$

Substitute and expand; equation (2.35) is reduced to:

$$y_1 \longmapsto y_1 + b_1\beta + c_1 y_1^2 + \{a_{11} + \sum_{j>1} d_j \frac{b_j}{1-\lambda_j}\} y_1 \beta$$

$$+\Big\{d_1 + \sum_{j>1} c_j \frac{b_j^2}{(1-\lambda_j)^2} + \sum_{j>1} e_j \frac{b_j}{1-\lambda_j} + \sum_{1<j<k} c_{jk} \frac{b_j b_k}{(1-\lambda_j)(1-\lambda_k)}\Big\}\beta^2$$
$$+ \text{third and higher order terms.} \tag{2.36}$$

The coefficients in the above expression are found to be (we write $\tilde{y}_j$ as y_j):

$$\begin{aligned}
\lambda_j &= 2\beta_0 \sum_x (x_j - y_j)^2 P(x), \;\; j = 2, \ldots, d \\
c_1 &= 4\beta_0^2 \sum_x (x_1 - y_1)^3 P(x), \\
b_j &= -\sum_x (x_j - y_j) \|x - y\|_2^2 P(x), \;\; j = 1, \ldots, d \\
a_{11} &= -2\beta_0 \sum_x (x_1 - y_1)^2 \|x - y\|_2^2 P(x) + 1/\beta_0 + \sum_x \|x - y\|_2^2 P(x), \\
d_1 &= \sum_x (x_1 - y_1) \|x - y\|_2^4 P(x) - 2 \sum_x (x_1 - y_1) \|x - y\|_2^2 P(x) \sum_x \|x - y\|_2^2 P(x), \\
c_j &= 4\beta_0^2 \sum_x (x_1 - y_1)(x_j - y_j)^2 P(x), \;\; j = 2, \ldots, d \\
d_j &= 4\beta_0^2 \sum_x (x_1 - y_1)^2 (x_j - y_j) P(x), \;\; j = 2, \ldots, d \\
e_j &= -2\beta_0 \sum_x (x_1 - y_1)(x_j - y_j) \|x - y\|_2^2 P(x), \;\; j = 2, \ldots, d \\
c_{jk} &= 4\beta_0^2 \sum_x (x_1 - y_1)(x_j - y_j)(x_k - y_k) P(x), \;\; 1 < j < k \leq d.
\end{aligned}$$

Unlike the one-dimensional case in which one can rule out transcritical bifurcation, it is not clear what value b_1 assumes if $c_1 = 0$. If $b_1 = 0$ or $a_{11} = \sum_{j>1} d_j \frac{b_j}{1-\lambda_j}$, then we find that transcritical bifurcation is impossible. But this is not obvious from the above expression. So let us examine what transcritical bifurcation means in terms of computation. Its bifurcation diagram is shown in Figure 2.2c. It is seen that transcritical bifurcation implies continuous exchange of stability [14]. One cluster changes from being stable to unstable while the other cluster changes from being unstable to stable. However, at the point of exchange, the cluster is no longer attracting in all directions. It is instead a saddle. Thus, rate-distortion theory says that the cluster should disappear or be merged. If transcritical bifurcation really occurs, then a stable cluster re-appears. This would mean the system can recover lost information. This is quite unimaginable. If we accept the argument based on rate-distortion theory, we will conclude that transcritical bifurcation cannot occur. The mapping has to rely on pitchfork or saddle-node bifurcations to reduce the number of clusters to achieve hierarchical clustering. Consequently, the behavior of the map (2.27) is essentially similar to that in the one-dimensional case. Indeed, in simulations, the saddle-node bifurcation is the dominant type observed.

2.12 Appendix B: Ability to Track the Signals

Let us consider the following situation. Suppose that the i^{th} datum x_i changes its position by δx_i; how is this going to affect the location of the fixed point y? Take the derivative of equation (2.8) with respect to x_i while keeping β fixed; we obtain:

$$\delta y = \frac{\left(1 - 2\beta(x_i - y)^2\right)P(x)}{1 - 2\beta E((x-y)^2)}\delta x_i. \tag{2.37}$$

Let us examine the above equation carefully. The denominator is always positive. It is zero only at bifurcation. The numerator is positive if x_i falls within $1/2\beta$ of cluster center y, negative if x_i falls beyond $1/2\beta$ of the cluster center. This is very interesting. In fact, we have a center-surround[7] nonlinear adaptation mechanism built into the formulation. Furthermore, this effect is modulated by the denominator, depending on the proximity of the clusters to the bifurcation point. This is intuitively very appealing, since it is expected that near bifurcations, the clusters should be more sensitive to changes in the data.

Now let us do a mental visualization of the adaptation. Imagine that we pull a particular datum x_i along some direction. A cluster center y lying within $1/2\beta$ from x_i will adapt to the change in the same direction. However, for cluster centers y lying beyond $1/2\beta$ from x_i, they will adapt in the direction that is opposite to the datum movement. We believe that this mechanism will be closely related to the ability of our formulation to track the changes in the signals in a nonstationary environment. We will now examine this analysis for existing methods.

In partitioning algorithms, a given partition determines each cluster center as the arithmetic mean of the data contained in each group irrespective of the metric used.

[7]Center-surround is just a term used to describe a neuron which responds positively/negatively to inputs falling inside/outside its receptive field.

That is

$$y_i = 1/n_i \sum_{i \in S_i} x_i \tag{2.38}$$

where S_i denotes the subset of the data contained in the i^{th} cluster. Let us consider a slight perturbation to datum x_k, small enough that the final partition is not affected. We see that

$$\delta y_i = \begin{cases} 1/n_i & \text{if } x_k \text{ belongs to the } i^{th} \text{ cluster} \\ 0 & \text{otherwise.} \end{cases} \tag{2.39}$$

Note that the coefficient n_i is constant. What is the problem with such a mechanism? First, a cluster is equally sensitive to all the points in its domain. That is, the spatial resolution is very coarse. Second, with no adaptation in the other clusters, adaption in a given cluster can cause it to venture into the domains of the other clusters. Once that occurs, the final partition can be quite different from the original one. That is, the clusters do not adapt gracefully to changes. Consequently, the traditional partitioning techniques adapt to changes in the signals in a manner different from our algorithm.

For agglomerative methods, we have a tree consisting of the clusters obtained by merging the sub-clusters. Let the change in the datum x_i be small enough that the ordering of the merging process stays the same, the nodes which contain the particular datum x_i will be perturbed by a factor $1/n_i$ where n_i is the number of data points in the node i, while the other nodes have their centers unchanged. Again we see that the effects on the clusters are either positive or zero. Thus, it suffers the same problems as the partitioning techniques.

To recapitulate, in our formulation, the nominal clusters have the ability to track the changes in the signals. The adaptation mimics the mechanism found in the biological systems. The previous methods adapt in a different way and may not have this tracking ability.

Bibliography

[1] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[2] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Trans. Information Theory*, 28, 129:137, 1982 (reprint of the 1957 paper).

[3] Y. Linde, A. Buzo and R.M. Gray, "An Algorithm for Vector Quantization," *IEEE Trans. Communications*, COM-28, 84-95, 1980.

[4] G. Ball and D. Hall, "A Clustering Technique for Summarizing Multivariate Data," *Behavioral Science*, 12, 153-155, 1967.

[5] E. Ruspini, "A New Approach to Clustering," *Inform. Contr.*, 15, 22-32, 1969.

[6] J. C. Dunn, "A Fuzzy Relative of the ISODATA Process and its Use in Detecting Compact Well-separated Clusters," *J. Cybern.*, 3, 32-57, 1974.

[7] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum, New York, 1981.

[8] I. Gath and A. B. Geva, "Unsupervised Optimal Fuzzy Clustering, " *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-11, 773-781, 1989.

[9] E. E. Gustafson and W. C. Kessel, "Fuzzy Clustering with a Fuzzy Covariance Matrix," in *Proc. IEEE CDC*, 761-766, San Diego, CA 1979.

[10] K. Rose, E. Gurewitz and G.C. Fox, "A Deterministic Annealing Approach to Clustering," *Pattern Recog. Lett.*, 11, 589-594, 1990.

[11] Yiu-fai Wong and A. Sideris, "Learning Convergence in the Cerebellar Model Articulation Controller," *IEEE Trans. Neural Networks*, 3, 115-121, 1992.

[12] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, 220, 671-680, 1983.

[13] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Springer-Verlag, New York, 1983.

[14] S. Wiggins, *Introductions to Applied Nonlinear Dynamical Systems and Chaos,* Springer-Verlag, New York, 1990.

[15] Eusebius Doedel, *AUTO: Software for Continuation and Bifurcation Problems in Ordinary Differential Equations*, Technical Report, Applied Mathematics, Caltech, 1986.

[16] D.J. Perozzi, *Analysis of Optimal Step Size Selection in Homotopy and Continuation Methods*, Ph.D. Thesis, Part II, Caltech, 1980.

[17] John R. Pierce and Edward C. Posner, *Introduction to Communication Science and Systems*, Plenum Press, New York, 1980.

[18] R.J. Fowler, M.S. Paterson and S.L. Tanimoto, "Optimal Packing and Covering in the Plane are NP-complete," *Inf. Proc. Letters*, 12, 3, 133-137, 1981.

[19] Richard O. Duda and Peter E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.

[20] J. H. Wolfe, "Pattern Clustering by Multivariate Mixture Analysis," *Multivariate Behavioral Research*, 5, 329-350, 1970.

[21] E.T. Jaynes, "Information Theory and Statistical Mechanics I," *Phy. Rev.*, vol. 106, 620-630, 1957.

[22] David L. Goodstein, *States of Matter*, Dover Publications Inc., New York, 1985.

[23] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distribution, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6, 721–741, 1984.

[24] S. Grossberg, "Adaptive Pattern Classification and Universal Recording I: Parallel Development and Coding of Neural Feature Detectors," *Biol. Cybern.*, 23, 187-202, 1976.

[25] T. Kohonen, *Self-Organizing and Associative Memory*, Springer, Berlin, 1984.

[26] D.E. Rumelhart and D. Zipser, "Feature Discovery by Competitive Learning," *Cognitive Science*, 9, 75-112, 1985.

[27] R. Dubes and A.K. Jain, "Validity Studies in Clustering Methodologies," *Pattern Recognition*, 11, 235-254, 1979.

[28] Yiu-fai Wong, K.J. Peters and E.C. Posner, "Unsupervised and Hierarchical Cluster Analysis and Classification of SAR Images," contributed paper in *International Space Year Conference on Earth and Space Science Information Systems*, Pasadena, February 1992.

[29] Charles Elachi, *Introduction to Physics and Techniques of Remote Sensing*, John Wiley & Sons, Inc., New York, 1987.

[30] W.T.K. Johnson, "Magellan Imaging Radar Mission to Venus," *Proc IEEE*, 79, 777-790, 1991.

[31] A. Freeman, J Villasenor and J.D. Klein, "Multi-frequency and Polarimetric Radar Backscatter Signatures for Discrimination between Agricultural Crops at the Flevoland Experimental Test Site," in *1991 AIRSAR Calibration Workshop*, Jet Propulsion Laboratory, May 23-24, 1991.

[32] J.J vanZyl, "Calibration of Polarimetric Radar Images using only Image Parameters and Trihedral Corner Reflector Responses," *IEEE Trans. Geoscience and Remote Sensing*, GE-28, 337-348, 1990.

[33] J.D. Klein, "Calibration of Complex Polarimetric SAR Imagery using Backscatter Correlations," to appear in *IEEE Trans. Aerospace and Elec. Sys.*, January, 1992.

[34] William Y. Huang and R.P. Lippmann, "Neural Net and Traditional Classifiers," in *Neural Information Processing System*, D.Z. Anderson (Ed.), 387-396, AIP, New York, 1988.

[35] Jack Carr, *Applications of Centre Manifold Theory*, Springer-Verlag, New York, 1981.

Chapter 3

Epilogue

As usual, as one searches for truth, one encounters more questions than previously thought. Quite often, the original plan has to be modified.

Although we have obtained a much better understanding of a simple and fast neural network model called CMAC, we also ran into several new and unanswered questions. For example, in realistic situations, one often presents the input samples randomly. The convergence of CMAC learning corresponds to the convergence of randomly updating the Gauss-Seidel iteration. Surprisingly, this question has never been answered in the literature. One can prove that Gauss-Seidel iteration still converges [1] and likewise CMAC learning also converges. It is also clear that the idea of analyzing the model in the frequency domain has applicability to the radial basis function networks regarding the choice of different basis functions. Although some preliminary work [2] has been done, more complete work is left for the future. Given the tradeoff between training accuracy and generalization [4], one might look into the generalization properties of CMAC with unknown inputs. Finally, the question concerning the efficient allocation of neurons led to clustering. It is here that we modified our path and got involved with an area of data analysis, one of the major goals of neural network research.

We have presented a clustering algorithm rather different from and more effective than the previous ones. Since cluster analysis shares many things in common with other neural learning paradigms such as self-organization and competitive learning, we have gained a deeper and better understanding of self-organization [3]. Our work may provide new insights for neural network research. Other unanswered questions include: Is there an optimal scale for a given cluster? How does one exploit the structure of the tree generated by the melting procedure to further understand the data? How can we incorporate the labeled samples into our formulation? Is there a better way to pick out the clusters? Can we adapt the clustering algorithm so that it can track the changing signals? It seems that the clustering algorithm raises more questions than it originally set out to answer. The challenge is left for future work.

Bibliography

[1] Joel Franklin, Private Communication, 1991.

[2] Yiu-fai Wong, "How Radial Basis Functions Work," in *Intl. Joint Conf. Neural Networks*, II. 133-138, Seattle, July 1991.

[3] Yiu-fai Wong, "A Comparative Study of the Kohonen Self-Organizing Map and the Elastic Net," in *Workshop on Computational Learning Theory and 'Natural' Learning Systems: Constraints and Prospects*, 1991.

[4] E. Levin, N Tishby and S.A. Solla, "A Statistical Approach to Learning and Generalization in Layered Neural Networks," *Proc. IEEE*, 78, 1568-1574, 1990.