Neural Networks, Pattern Recognition, and Fingerprint Hallucination

Thesis by

Eric Mjolsness

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

5198:TR:85

California Institute of Technology

Pasadena, California

1986

(Submitted September 2, 1985)

to my parents
Patricia and Raymond Mjolsness

Acknowledgments

I would like to acknowledge the great influence of my advisor, John Hopfield, on the direction of this thesis and of my interests. I wish to thank Carver Mead for many discussions on vision and for adopting me into his wide-ranging research group. I thank Richard Feynman for providing a stimulating intellectual environment at the intersection of physics and computation. Geoffrey Fox provided support for my early explorations of neural networks, and my collaboration with Dave Sharp and Alan Lapedes helped a great deal in the early phase of the present work. John Wawrzynek provided much selfless computer assistance over a long time. John Platt's enthusiasm, insights and proofreading are greatly appreciated. I have benefitted from discussions with Jack Cowan, Mike Douglas, David Feinstein, and Misha Mahowald. Gerry Sussman asked awkward questions. Shelley Mjolsness has been enormously helpful in all aspects of my life during the years of graduate study.

This work was supported by the System Development Foundation and by the Los Alamos National Laboratory.

Abstract

Many interesting and globally ordered patterns of behavior, such as solidification, arise in statistical physics and are generally referred to as collective phenomena. The obvious analogies to parallel computation can be extended quite far, so that simple computations may be endowed with the most desirable properties of collective phenomena: robustness against circuit defects, extreme parallelism, asynchronous operation and efficient implementation in silicon. To obtain these advantages for more complicated and useful computations, the relatively simple pattern recognition task of fingerprint identification has been selected. Simulations show that an intuitively understandable neural network can generate fingerprint-like patterns within a framework which should allow control of wire length and scale invariance. The purpose of generating such patterns is to create a network whose stable states are noiseless fingerprint patterns, so that noisy fingerprint patterns used as input to the network will evoke the corresponding noiseless patterns as output. There is a developing theory for predicting the behavior of such networks and thereby reducing the amount of simulation that must be done to design them.

Table of Contents

I Introduction

It is natural, when imagining very large computers and their construction, to compare present artificial computing machinery with the much less well-understood brains of animals such as ourselves. The comparison is biased in favor of hardware considerations, since neurobiologists have understandably found it easier and more direct to investigate neurons and their connections than algorithms or other programming principles that may be present in biological computation. (In the case of artificial computers, of course, we have the illusion of complete understanding of both hardware and software.) Nevertheless, such a comparison is striking.

Brains and computers differ first in size, there being $10^{10} - 10^{11}$ neurons and $10^3 - 10^4$ connections from one neuron to others, as against $\sim 10^7$ - 10^8 transistors with a fanout of 1-10, usually, for computers. There is no guarantee that these are the right numbers to compare, but the size difference appears to be immense: perhaps six orders of magnitude. On the other hand, computers are faster by 3-4 orders of magnitude depending on what speeds are compared. Brains apparently make extensive use of analog signals and analog computation unlike present general-purpose computers. Brains are capable of surprising feats of self-repair [Merzenich et al. 84] and learning, whereas present-day computers are vulnerable to even the smallest hardware malfunctions and must be quite tediously programmed. Until recently artificial computers could be distinguished from brains on the basis of inefficiencies associated with purely serial computation, but advances in parallel computation have largely removed this problem. All of these comparisons are well known, even cliches, and have been used to bolster many mutually inconsistent but optimistic-sounding recommendations for computer design [Brown 84; Conrad 85; Hillis 82a and 82b].

One of the most interesting normative models of biological computation is the content addressable memory (CAM) of Hopfield [Hopfield 82]. It used large fanout "neurons" to retrieve one of a number of stored bit strings, choosing one very similar to an input bit string – the computation implemented is an "inexact match" which tolerates many independent one-bit errors in the input. The neural network may be programmed (memories may be stored) by a simple formula or by an equally simple learning mechanism. Numerous hardware defects can occur before the network is broken, so that the CAM may be appropriate for use in Wafer Scale Integration where, for present technology, some hardware failures are inevitable and must be tolerated [Leighton and Leiserson 85]. This possibility leads to one of the more tantalizing similarities between the CAM and biological computation: that the CAM, by tolerating hardware errors, may be easily implemented in error-prone technologies and thus on a much larger scale than conventional circuits; this would decrease the size disparity between natural and artificial computing machinery. Finally, the CAM may use either discrete or analog neurons [Hopfield 84].

In the case of discrete neurons, the behavior of Hopfield's network is as follows. Each neuron has one of two possible values, for example +1 or -1, and when it is updated a neuron with value s_i changes that value to

$$s_i' = \mathrm{sgn}(\sum_{j=1}^{N} T_{ij} s_j - h_i) \tag{1}$$

where

$$\mathrm{sgn}(x) = \begin{cases} +1 & \text{if } x \geq 0; \\ -1 & \text{otherwise,} \end{cases}$$

and

T_{ij} = a numeric "synapse strength" from j to i

h_i = a numeric "threshold" for neuron i.

A variety of update schemes are acceptable. One standard scheme is to choose a random permutation of the neuron indices $\{1, \ldots, N\}$ and repeatedly update the neurons in that standard order. If $T_{ii} = 0$ and $T_{ij} = T_{ji}$, then each time a neuron is updated the energy function

$$E = \sum_{ij} T_{ij} s_i s_j \tag{2}$$

decreases. This time behavior corresponds to that commonly used for Monte Carlo simulations of thermal Ising models with temperature approaching zero, or being quenched to zero [Kirkpatrick 85].

In the case of analog neurons, one replaces the "sgn" transfer function with a continuous sigmoidal function g and one uses a more continuous updating scheme, such as simultaneous differential equations with a time constant τ for equilibration; an example is

$$\tau \dot{s}_i + s_i = g(\sum_j T_{ij} s_j - h_i).$$

The discrete neurons are much less expensive to simulate on general-purpose computers. Needless to say, any resemblance between the dynamical system just defined, with its neural terminology, and the real dynamics of neurons and synapses is a historical coincidence. The two systems may or may not have similar behavior; each is interesting independently.

The CAM model has a number of the properties associated with biological computations, and it can be efficiently implemented in silicon electronic technology [Sivilotti et al. 85]. Its principal drawback is the relatively uninteresting computation which it performs. Generally, attempts to use a "Hamming distance minimizer" as the heart of a practical computation fail due to the rigid assignment of meanings to bits in the stored and input bit strings; there is no natural mechanism for movement of information such as shifting. Still, the Hamming distance minimization computation is suggestive of pattern recognition problems and it may be possible to use some of Hopfield's ideas to design pattern recognition networks. Certainly, pattern recognition could use the kind of increased computational power which a large, robust, analog, parallel special-purpose computer (or one incorporating just some of these features) might provide. Visual pattern recognition, in particular, is widely thought to be in need of several orders of magnitude more computational power than is available now.

The retrieval of a memory in the CAM model involves all the neurons in the network, being an essentially collective behavior. The model is in fact related to collective models

from statistical physics known as spin-glasses, and works best in the limit of large numbers of neurons. Consequently, the behaviors of the CAM and of similar networks are often referred to (collectively, of course) as "collective computation."

The main problems with extending the CAM model to create a new paradigm for computing are how to find connection matrices T_{ij} that do more interesting computations, and how to minimize the physical cost of implementing T_{ij} in present-day technologies: silicon electronic, optical, or the poorly understood neural technology that occurs in nature. Usually the physical cost of a network is related to the total wire length or wire volume needed, and to the time required to finish a computation. In perception problems it is particularly tempting to introduce designs which are profligate in their use of wire length. The present work introduces methods of some (unknown) generality for solving both problems.

Several methods for "programming" networks, i.e., finding T_{ij} which produce a desired behavior, will be used. One is the method introduced by Hopfield with the CAM [Hopfield 82]. A second, the use of a few fixed points to control large basins of attraction, is introduced in Chapter III. Also novel is a geometrical network organization which minimizes wirelength by enforcing locality of most connections (most T_{ij} matrix elements are zero) while efficiently implementing nonlocal pattern recognition algorithms such as "multiple scales of resolution" methods. These network programming techniques will be introduced entirely through a set of simple computational tasks related to fingerprint recognition.

1.1. The Potential Virtues of Collective Computation

The neural CAM has a number of properties which make it worthy of imitation, each possibly contributing a factor of 10 in computing power. In addition, it is extremely parallel.

About 20% of the simulated network could be eliminated, and the rest would still work. This is a stronger form of robustness against hardware errors than would be required to solve an important technological problem: using Wafer Scale Integration despite the extreme difficulty of obtaining any error-free samples of such large circuits. Certainly, the network's large fanout helps here, but it is not yet clear that spatially local implementations of large fanout networks can share this robustness. There was also a form of robustness against errors in the relative timing of signals: the network operated asynchronously. This is good for networks so large that synchronization is difficult to achieve. Another form of robustness simply makes it easier to simulate collective circuits on a general-purpose machine: insensitivity to the exact dynamics of the elementary computing units allows one to simulate the cheapest reasonable dynamics rather than, for example, the full analog electrical equations.

A second potentially important virtue of collective computation is its compatibility with analog computation: Hopfield has simulated an analog CAM [Hopfield 84]. It's reasonable to encode eight bits per neuron, rather than one. The present work will not much explore this direction due to the higher expense of simulation.

The CAM and other networks to be introduced later tend to converge to fixed points (outputs) quite quickly unless there is an information-theoretic reason not to. This means they are efficient in their use of time, at least for tasks which needn't take much time.

A crucial property is efficient implementation in silicon. Sivilotti et al. have fabricated a VLSI CAM [Sivilotti et al. 85], and analog networks can be implemented with very low overhead indeed. The neurons can be implemented with just a few transistors [Platt 85]. The summation of inputs can be done by the simple intersection of wires if neural inputs are

represented by currents rather than voltages. This efficiency of implementation means that neural network theory is directly relevant to special-purpose VLSI design. This fact permits the simultaneous treatment of an algorithm and its implementation. Usually, in computer science these two are separated for ease of thinking and understanding, but if one can think about both in the same framework there is a significant advantage: a greater appreciation of cost issues. Quick appraisal of the wirelength cost of an algorithm encourages algorithms that are efficient in their use of long wires.

In all this, the extreme parallelism inherent in talking about very simple elementary units working simultaneously is taken for granted; the properties of fault-tolerance, analog signals and special-purpose design are good for computational power over and above that expected from a general-purpose parallel machine.

1.2. General Methods for Network Design

To obtain any of the desirable properties listed above for a new computation it is necessary somehow to specify the connections T_{ij} between the neurons in a network. This "programming" or "network design" problem occurs for each new computation so that general network design methods are desperately needed. Fortunately, several are available.

The first method is a way to compose simple networks to get a complicated one. The choice of T_{ij} for the CAM network and for the equally interesting Travelling Salesman Problem network of [Hopfield and Tank 85] are both of the form

$$T_{ij} = \sum_{\alpha=1}^{A} w^{\alpha} T_{ij}^{\alpha}$$

where T_{ij}^{α} is a more easily understood network and w^{α} is a real weight. This method of composing networks is very unusual in computer science but common in physics, where the corresponding addition of energy functions

$$E = \sum_{ij} T_{ij} s_i s_j + \sum_{i} h_i s_i = \sum_{\alpha=1}^{A} w^{\alpha} E^{\alpha}$$

is very common for A = several to several dozen terms, and the w^{α} are called coupling constants. The resulting physical systems are often simply understandable in terms of the elementary E^{α} – and often not. A is the number of stored memories. In the CAM network A can be proportional to the number of neurons in the network, and the nonlocal connections T_{ij} are

$$T_{ij} = \sum_{\alpha=1}^{A} T_{ij}^{\alpha} = \sum_{\alpha=1}^{A} s_i^{\alpha} s_j^{\alpha},$$

which is the sum of the outer products of A memory vectors s^{α} with themselves. This kind of network will be discussed further in Chapter II.

Another general method for finding desirable T_{ij} matrices consists of controlling a relatively small number of fixed-point configurations (configurations which are fixed under the neural update rule). Often a few fixed points can be used to control the entire behavior of the network in the high-dimensional space of possible configurations. In a CAM network, for example, one can try to arrange that the desired fixed points' basins of attraction cover

the space and have roughly equal volume. In local image processing networks, it may be sufficient that a neural configuration be always locally similar to one of the controlled fixed points, though which of the controlled fixed points it looks like may vary across the image.

The stability of each neuron within a configuration s^α may be expressed as an inequality in T_{ij}:

$$s_i^\alpha = \mathrm{sgn}(\sum_{i \neq j} T_{ij} s_j^\alpha - h_i) \Rightarrow$$

$$0 \leq (s_i^\alpha)^2 = \mathrm{sgn}(\sum_{i \neq j} T_{ij} s_i^\alpha s_j^\alpha - h_i s_i^\alpha)$$

$$\sum_{i \neq j} T_{ij} s_i^\alpha s_j^\alpha - h_i s_i^\alpha \geq 0. \tag{3}$$

Satisfying a number of such linear inequalities in T_{ij}, together with locality constraints and minimum wirelength constraints, is often a computationally tractable problem. If it isn't, one can decrease the number of fixed-point constraints and decrease the number of free parameters in T_{ij} by assuming a special form for the connection matrix; that way a large network can be specified with a few parameters. An example is a one-dimensional special form $T_{ij} = T(|i - j|)$, which reduces N^2 parameters to N while retaining the linearity of the inequality constraints (4). The control of fixed points through special forms and inequalities is practiced in Chapter III, where it is modified somewhat, and in Platt's work on simulating Petri nets [Platt 85].

An interesting and somewhat unusual network design technique is the expression of a network as the discretization of a continuous medium. For very large and strictly local analog networks one obtains partial differential equations this way; the analog network is a spatial discretization of the differential equation and the discrete-neurons network severely discretizes the time derivative as well.

However, it is not necessary to use strictly local connections, which correspond to low-order differential operators, and in vision one often wants to calculate quickly and use global information. Global information can be calculated and disseminated slowly by iterated local interactions, or quickly by long-range interactions which, in the electrical terminology, require expensive long wires. The usual compromise was suggested to the present author by the renormalization group formalism [Wilson 74] but is in fact standard in vision research: by connecting several local networks whose lattice spacings differ by a constant factor, one obtains a relatively few long wires (the local connections in the low-resolution networks) which appear to the high-resolution networks as global interactions. This idea is called "multiple levels of resolution" or simply the "multiresolution" method.

In Chapters II and especially III we will see how to express such a network as the discretization of an unusual continuous medium by the addition of a third "hierarchy" axis and a neural density function. The cost of this addition is a constant factor in wirelength. There is no reason that other, more complicated, neural architectures could not be expressed as continuous media with other new axes, such as measured object size.

The continuous medium treatment has a number of advantages. It makes the network look homogeneous, each neuron being surrounded by the same medium in its neighborhood. This encourages scale- and translation-invariant behavior (scale invariance being the relevant

invariance along the hierarchy axis). A neighborhood looks exactly the same after being translated or scaled. This fact has beneficial effects: first, the image processing is invariant. Translation invariance is obviously desirable, and in many images scale encodes depth so that scale invariance is also desirable. Also, network homogeneity means that the network connectivity is described by very few parameters (those adequate for describing just one neighborhood) so that linear constraint satisfaction and other programming techniques become computationally tractable. These advantages come at a price. The continuous medium approximation requires a high density of neurons, compared (perhaps) to the information density. Thus some constant efficiency factor in neuron number and total wirelength is sacrificed. In a hierarchical organization, one makes highly variable and nonuniform connectivity neighborhoods look uniform by sprinkling in extra neurons. An obvious corrective measure, not pursued here, would be to start with the continuous (infinite density) approximation and correct it by some sort of 1/density perturbation theory. The lowest order of such a theory would be a continuous homogeneous medium with infinitely many neurons per unit area, and higher orders would allow one to treat finite densities. The idea is to derive corrections to the idealized T_{ij} matrix which compensate for local density variations; such corrections would no longer be translation- and scale-invariant, but would exist for the purpose of preserving invariant behavior after a non-invariant network discretization. As the information/neuron is increased, interesting packing and tessellation problems arise; Misha Mahowald has done intriguing work on such problems. These considerations, however, are secondary for the present purposes.

Thus we have a continuous, geometric approach to multiresolution methods. One of the most interesting such methods is the work of Terzopoulos [Terzopoulos 84] in shape deduction. In this case a two-dimensional relaxation medium (a thin plate) is to be more quickly relaxed by coupling to coarser-grain versions of itself. The interplate coupling is done through interpolation and averaging interactions. The present method differs slightly: the design of the interpolation and smoothing interactions are unified with that of the intralevel interactions. Both are accomplished through the optimization of an invariant neighborhood connectivity function. Also, the coupling occurs here without doubling the number of degrees of freedom as in [Terzopoulos 84]. It remains to be seen whether the extra simplicity can be used to any good effect. It is to be hoped that the simple and general methods discussed here will prove useful for other computations, after having been developed and refined through experience with fingerprint recognition tasks.

In vision there are many computations for which a continuous, homogeneous hierarchy may be useful (see [Rosenfeld 84] for multiresolution examples). Since interesting shape-producing processes operate at all scales in nature, any general shape recognizer would need to work equally well for large and small objects, and would need to allow different sizes to interact to keep track of object parts and subparts. The automatic description of frequently encountered shapes may be a task well modelled by the much simpler problem of scale-invariant fingerprint recognition. There are also non-homogeneous hierarchies in known vision algorithms, where the nature of the computation changes with the level in the hierarchy. It is possible that the change could be made gradual, so that continuous computational media could again be useful.

Recently, the relationship between a continuous one-dimensional medium and a finite sequence of locally coupled analog circuits has been exploited by Platt and Mead [private communication] to design a VLSI "cochlea" and related sound-processing devices.

1.3 The Fingerprint Identification Task and Subtasks

One reason that people are drawn to work on vision is its apparent universality. There are so many things a computer could do if only it could "see." This is of course a comment on the versatility of our own visual systems. Certainly, there is no specially evolved neural mechanism for driving a car; the credit for this accomplishment must be divided between learning and the generality of mechanisms designed for other tasks. It's the generality that we're after; any particular task will likely not be the one we're "really" interested in, at least not for long.

The alternative to universality is the specific, once-only solution which is useless outside of its original context. In computing circles this is called a "kludge" or "hack," especially when more general alternatives are available. It is possible that a given vision task is best done with many kludges, but that a collection of tasks are best done with fewer kludges each and more sharing of methods. The chief danger of concentrating on a few example tasks in vision, as is done in the present work, is the accumulation of kludges.

A new research idea cannot at the outset be tested on many real tasks, but it is possible to use no real examples at all. For example, one can make up vision tasks which nobody needs done such as solving idealized jigsaw puzzles. This seems unhealthy since the properties and difficulties of real tasks are often surprising. It was therefore decided to try out the new neural programming techniques on one of the simplest of real pattern recognition tasks.

Fingerprints do not represent a missing third dimension as scenes do, and they may be recognized even after each pixel is forced to take one of two values (black or white). Previous work by Megdal [Megdal 83] made the problem a local favorite at Caltech. The idea is to locate the minutiae (mostly branch points and ridge endings) despite the presence of a much greater number of spurious branch points and ridge endings due to characteristic fingerprint noise such as holes in ridges caused by sweat pores and bridges across valleys or across ridges. For more information on the nature of fingerprints, see [L.S. Penrose and Ohara 73a; L.S. Penrose 73b; R. Penrose 79]. Figure 1 shows a fingerprint obtained by Megdal after being filtered through a 3x3 pixel Gaussian filter and thresholded to black and white. Notice that branchings and endings are dual under interchange of black and white.

The first thing to do in analyzing a fingerprint is to get rid of the characteristic noise, leaving only clean ridges, valleys and isolated minutiae. Minutiae should be isolated because sufficiently close pairs of minutiae are easily confused with bridges and holes. This processing greatly simplifies minutia detection by making that operation local on the scale of the prevailing ridge width. Also, simplifying the ridges and valleys allows simple algorithms for counting the number of ridges between nearby minutia to succeed. Then, from the cleaned up image one can calculate the minutiae locations and orientations. A complete print has 50 to 150 minutiae, of which about twelve are needed to establish a match with another print. So there ought to be considerable redundancy in the minutiae information. It is then possible to find relationships (such as ridge count and relative direction) between nearby minutiae; the relationships between distant minutiae are much less dependable. Finally, one can compare the deduced sparse information with the same representation of many other fingerprints to match a print with a library of prints.

A number of efforts fit this general description, including Megdal's and a successful commercial system by NEC. The $2.6 million NEC system is operating in San Francisco and has identified prints in 20 per cent of the cases where police recovered fingerprint evidence

[Johnston 85]. A larger ($22.5 million) version of the NEC system is also being purchased for the California Department of Justice's CAL-ID project. There are several competing commercial systems from other companies.

In the NEC machine, as described in [NEC, CG&A], a skeleton ridge pattern is detected and cleaned up first. Then the minutiae are detected, and for each minutia a local coordinate system is specified so that one axis travels along the local ridge direction and the other axis is perpendicular to the first. In each quadrant of the resulting coordinate system the nearest minutia is found and the number of intervening ridges is counted. This produces a sparse labelled graph which compactly summarizes the minutiae information, and may be matched against many stored graphs to produce matching scores. A list of the best candidate fingerprints and their matching scores is the output.

The matching procedure is assisted by a special-purpose chip which can compare 40 search minutiae with 100 file minutiae in 1.3 ms. 18 such chips [CG&A] are in the CAL-ID system. The matching algorithm involves scoring and ordering the possible file minutiae corresponding to a search minutia by using the local ridge count, position and orientation information. On this basis a global translation and rotation of one image with respect to the other is calculated to optimize the matching between them, and this step is followed by a recalculation of the pairing (or lack of it) between search and file minutiae. The proposed match is then assigned a score depending on the plausibility of the proposed minutiae matches and the number of minutiae not matched. More detail (but not much more) is available in [NEC,CG&A].

If such a system encounters N fingerprint images over its lifetime and matches each with every other then there are N fingerprint analyses and N^2 matchings, so transferring work from the matching to the analysis phase may be a very good idea. It would not be nearly so useful for repetitive identity verification for a small population $P \ll N$; there would only be PN matches performed. By looking for unusual "features" during the analysis phase, it may be possible to reduce the matching phase to a comparison of two bit strings, one for each fingerprint. Each bit in a string would record the presence or absence of one type of unusual feature. Retrieval of a stored fingerprint image would consist of finding the stored string whose unusual features coincided in type, as much as possible, with those of the input image. This is just a Hamming distance minimization of the sort performed by the Content Addressable Memory. If such a scheme were possible it would be an improvement over the present NEC machine.

If one looks at the amount of information involved in any of these fingerprint recognition schemes, there is a lot of image information which is mostly removed to leave a small description of the minutiae graph. This description is then compared with an enormous data base of stored fingerprint information. So there is a severe bottleneck between the image and the data base which naturally divides the recognition task into image processing and labelled graph matching tasks. It makes sense to try the first (image processing) task first since the statistical properties of fingerprint minutiae may be necessary to design the most efficient matcher.

Among the various image-processing operations needed (removal of fingerprint noise, branch detection, ridge counting, etc.) the one involving the most information is the removal of fingerprint noise since this operation works best on the original image data rather than on a simpler skeleton. So this computation is most in need of special-purpose hardware, and

we shall try to design a neural network which removes characteristic fingerprint noise and the false minutiae signals associated with such noise. It will be argued that branch and ridge end detection, at least, is then a straightforward local computation (see Chapter II).

One of the difficulties with noise removal is that ridges and valleys vary in width even within one fingerprint by a factor of 2.5. This fact, and the general interest in scale-invariant pattern recognition mentioned previously, suggested that the network be required to behave in a scale-invariant way. This constraint is stronger than what is strictly necessary for the fingerprint task but it may increase the applicability of the present results to other problems.

Finally, the approach used will be first to obtain a network which produces plausible clean fingerprint-like patterns from random input, and then modify it so that real fingerprint patterns as input produce "nearby" but clean (noiseless) fingerprint patterns as output. The idea is first to require the network to have the right "vocabulary" of outputs, and then to refine it to obtain an acceptable input-output map. The first stage, the production of clean fingerprint-like patterns from random input, is referred to as "hallucination."

In Chapter II we present a hierarchical fingerprint hallucinating network which produces stripes of various widths. In Chapter III the relationship between neural network and continuous medium is clarified, and an acceptable input/output map is obtained between idealized one-dimensional slices of fingerprint images. The map is correct for a wide range of ridge widths so that the network appears to be genuinely scale-invariant. This success is restricted to the easier one-dimensional case. Chapter IV reverses the perilous decline in dimensionality; the 2-d network of Chapter II is here tried out on windows from real fingerprint images and modified to perform part of the required computation. The satisfactory 2-d network is not achieved, but much is learned about neural programming and design methods along the way.

Figure 1

II Stripe Hallucination

In general, then, to control the behavior of a network of threshold elements we have at our disposal the connection weights T_{ij} and the thresholds h_i. There are several techniques of some considerable but unknown generality for "programming" networks to behave as desired robustly. These techniques include the synapse-by-synapse summation of previously understood networks, the control of a relatively few fixed points, and the use of appropriate continuous geometries as an approximation to the actual discrete network. A simple example of one of these techniques is the use of summation to construct a content addressable memory, or CAM.

II.1. The Content Addressable Memory

In the content addressable memory network as explained by Hopfield [82], one is presented with an input bit string $s_i^{(\text{in})}$, $1 \leq i \leq N$, and asked to retrieve that one of M stored bit strings $s_i^{(m)}$ which is closest to the input string in Hamming distance. If the individual bits are represented by $+1$ or -1 instead of the traditional 0 or 1, then the following energy function suffices for the easy case $M = 1$, in that it is minimal if and only if the current network configuration s_i is $s_i^{(1)}$ or its negation, $-s_i^{(1)}$:

$$E(s) = \left[\sum_{i=1}^{N} s_i s_i^{(1)}\right]^2.$$

The connection matrix is

$$T_{ij} = s_i^{(1)} s_j^{(1)}.$$

Thus, starting this network off in the input state $s_i^{(\text{in})}$ will result in the network's reaching the desired final state $s_i^{(1)}$, the "closest" and only stored memory. It is at first surprising that a network with M such memories, chosen at random, can be generated by adding up the M corresponding single-memory networks. The resulting matrix of connection weights is

$$T_{ij} = \sum_{m=1}^{M} s_i^{(m)} s_j^{(m)} \tag{1}$$

and its efficiency, robustness and other properties are introduced in [Hopfield 82].

The production of a complicated network by synapse-by-synapse summation of a set of more thoroughly understood and simpler networks may be expressed as

$$T_{ij} = \sum_{\alpha} T_{ij}^{\alpha} \tag{2}$$

and is very different from conventional composition of circuits to produce new ones. The principal difference is that no new active (threshold) elements are introduced; these elements

are shared by all of the simpler networks. The more conventional approach would be to collect a set of sub-networks with distinct threshold elements s_i^α, connected only to other elements within the the same subnet α, and then try to sparsely interconnect s^α and s^β.

II.2. A Fixed-Width Stripe Network

A simple image-processing problem may also be solved using the summation technique, together with some geometric ideas. We would like to construct a two-dimensional network whose stable states include the striped patterns of constant width. We would like to have two sets of neurons distributed about the plane: one, which contains just a pattern of black and white pixels, and a second set of neurons which in addition contains information about the local direction along a stripe. The second set of neurons is included to allow further constraints, which must be expressed in terms of the local stripe direction, to be imposed later on. An example of such a constraint would be that branch points should be separated by many stripe widths. Note that by introducing direction-sensitive neurons we diverge from other efforts, like those of [Hinton and Sejnowski 83] and the Perceptron research of the 1950's, which attempt to find some simple dynamics for changing the network's connections and thereby discovering the necessary network organization.

We imagine that the neurons are spread uniformly over the plane with great density, at first infinite. Instead of being indexed by a single index i, the neurons come in two types (as mentioned above) which are indexed differently. The first type are called $P_{\vec{x}}$, for pixel, where $\vec{x}$ is a spatial index. The second are called stripe detectors $S_{\vec{x},\phi,c}$ where ϕ is the local angle of stripe orientation and c is the local color, -1 for white and $+1$ for black. At a given spatial index $\vec{x}$ all but one $S_{\vec{x},\ldots}$ should be turned off.

The condition for the use of the sum of outer products formula

$$T_{ij} = \sum_m s_i^{(m)} s_j^{(m)}$$

is that the desired fixed points s_i^m be approximately orthogonal. Given that most of the S-type neurons at any point are turned off, this assumption cannot hold if the "off" value of a neuron is -1. The assumption stands a much better chance if "off" neurons do not contribute to inner products so that their value is 0, and Appendix I to this chapter shows that in the limit where A, the number of different values of ϕ, is large the orthogonality of our desired striped fixed points is maximized when P neurons have the two possible states $+1$ and -1, as in the content addressable memory, and the S neurons can assume the values 0 or 1. These two cases are already standard so we will not bother to improve the large A approximation.

A particularly interesting aspect of this network is the synaptic weights from P-type neurons to S-type neurons, as these determine which pictures a given stripe-detector responds to. The $S \rightarrow S$ and $S \rightarrow P$ connections may then be thought of as consistency checks. The $P \rightarrow S$ connections receive contributions only from stripe patterns which turn on the given stripe-detector, since the other patterns in P are multiplied by an $S = 0$ neuron value in the outer product synapse formula. Thus

$$T_{(\vec{x}_0,\phi,c),\vec{x}} = \sum_{\text{configs } m} S^m_{\vec{x}_0,\phi,c} P^m_{\vec{x}} = \sum_{\text{valid configs } m} P^m_{\vec{x}}$$

and obtaining the desired synapse weights is simply a matter of superimposing all the stripe configurations which should turn on a given stripe detector. Since these configurations are of

infinite extent, one will eventually want to truncate the stripe detector's innervation pattern by removing all connections between very distant points. The infinite-range case will provide some guidance as to how far away two neurons may get before their connection may be safely truncated to zero.

The desired fixed point configurations are all stripes and may be characterized by an angle of deviation, ψ, from the stripe detectors' ideal angle, and by the phase θ, by which a stripe pattern is shifted in the direction perpendicular to the stripe direction. The conditions for the stripe detector to respond are

$$-\Delta\psi \leq \psi \leq \Delta\psi$$

and

$$-\frac{1}{2} \leq \theta \leq \frac{1}{2} \tag{3}$$

where the phases θ are normalized so that the stripe period is 2, rather than 2π. On this scale the stripe width is 1. Arranging our coordinates so that the stripe detector is at the origin, its innervation pattern is

$$T(\vec{x}) = \int_{-\Delta\psi}^{\Delta\psi} d\psi \int_{-1/2}^{1/2} d\theta \ \mathrm{Stripe}(\psi, \theta, \vec{x})$$

where

$$\mathrm{Stripe}(\psi, \theta, \vec{x}) = \mathrm{Stripe}(0, 0, \vec{x}')$$

and

$$x' = \theta + x\cos\psi + y\sin\psi$$

$$y' = \theta - x\sin\psi + y\cos\psi$$

$$\mathrm{Stripe}(0, 0, (x', y')) = \sum_{j=-\infty}^{j=\infty} (-1)^j \mathrm{Bit}(x', j - \frac{1}{2}, j + \frac{1}{2})$$

$$\mathrm{Bit}(x, x_0, x_1) = \begin{cases} 1, & \text{if } x_0 \leq x \leq x_1; \\ 0, & \text{otherwise.} \end{cases}$$

So $\mathrm{Stripe}(0, 0, (x', y'))$ is a square wave in x', and the general Stripe is a two-dimensional rotation and translation of this pattern.

A discretization of the integral expression for $T(\vec{x})$ is shown in Figure 1, with $\Delta\psi = \frac{\pi}{6}$ and each integral approximated by a Riemann sum with 30 summands. Figure 2 is a close-up, showing the region of large T. The contour intervals are .1 and .2, respectively, and in both figures the very long contours are all at zero. The hills alternate in sign. The isolated points show the x, y grid used to sample the function, and the contour-plotting procedure of a standard statistical package [Ref: S] was used to interpolate smooth contours.

The principal features to notice about $T(\vec{x})$ are the alternating series of positive and negative vertically elongated bumps along the x-axis, and their spreading out starting at $x = \pm 6$.

We may check that this T is a reasonable innervation pattern for a stripe detector by asking for its response to stripe patterns of various phases and angles, both those that should turn on the given stripe detector and those that shouldn't. In other words the possible stripe

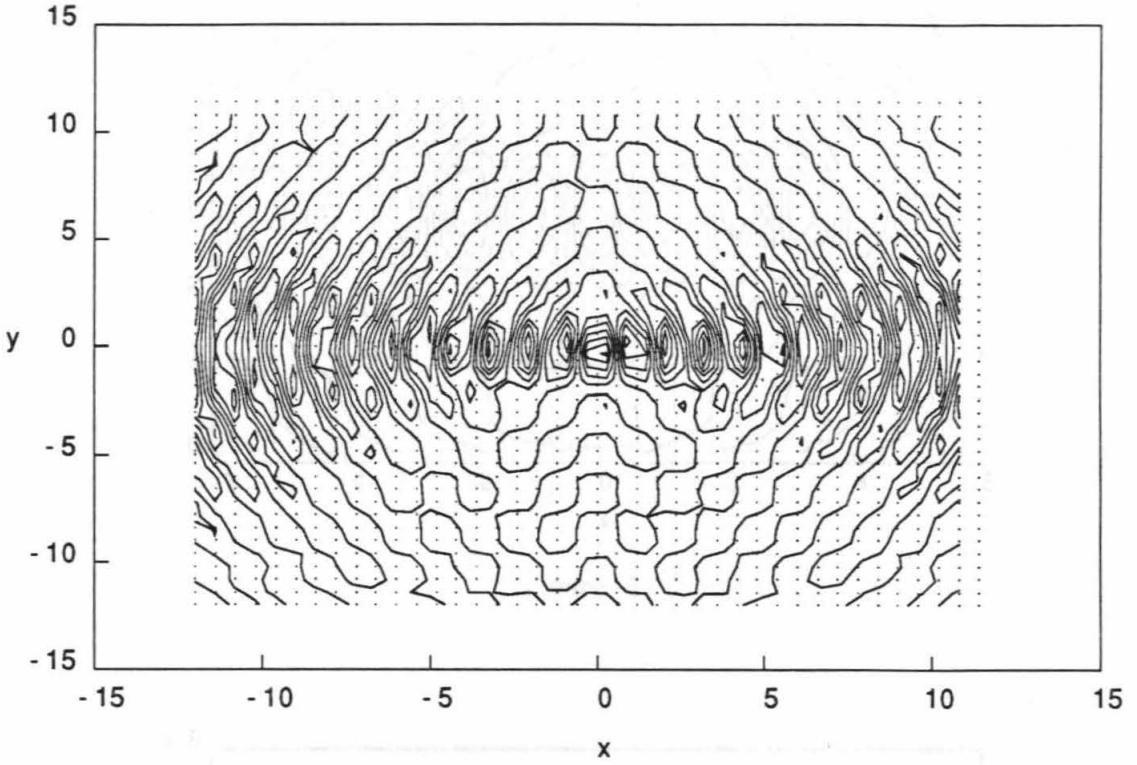

Figure 1

patterns are characterized by two parameters, ψ and θ, and a given stripe detector S_i should have large net input $R_i = \sum_j T_{ij} P_j$ only for configurations P_j characterized by the parameter range in Equation (3).

In Figure 3, the response function R is computed as a discretized integral with $T(\vec{x})$ truncated to zero outside the region $-6 \leq x, y \leq 6$, so

$$R(\psi, \theta) = \int dx \int dy T_{\text{trunc}}(\vec{x}) \text{ Stripe}(\psi, \theta, \vec{x}). \tag{4}$$

Once again, and in all future contour plots, the isolated points represent the sampling grid. The integrals were each approximated by 30 summands.

The response function R is nearly perfect; thresholding it slightly above zero leaves a positive response for almost exactly the stripe parameter range desired.

The special pattern of $T(\vec{x})$ encourages us to save wiring costs by removing all connections with relatively low weights; most connections will be so removed due to T's alternating concentrations of weight along the horizontal axis. Figure 4 shows T with all values within .4 of zero removed, and Figure 5 shows the corresponding response function. The network still works very well.

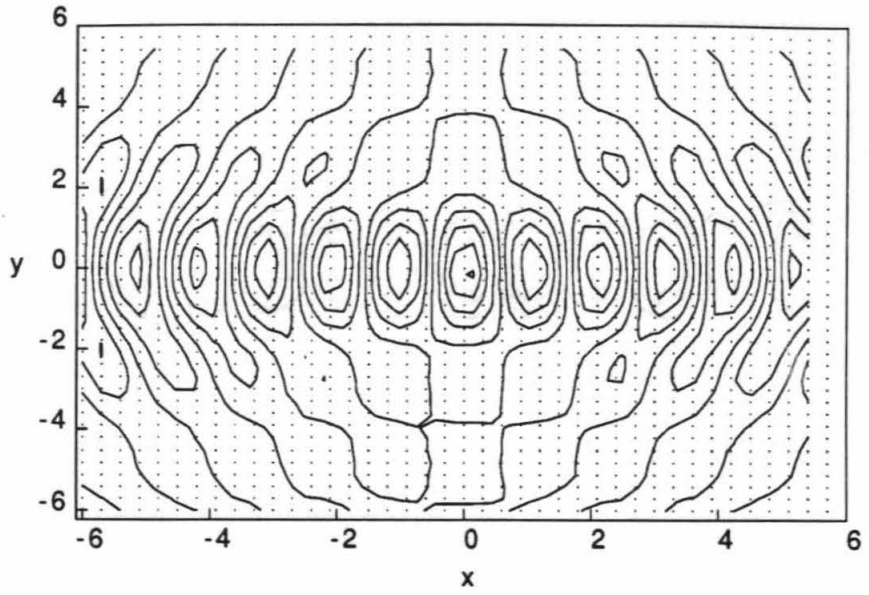

Figure 2

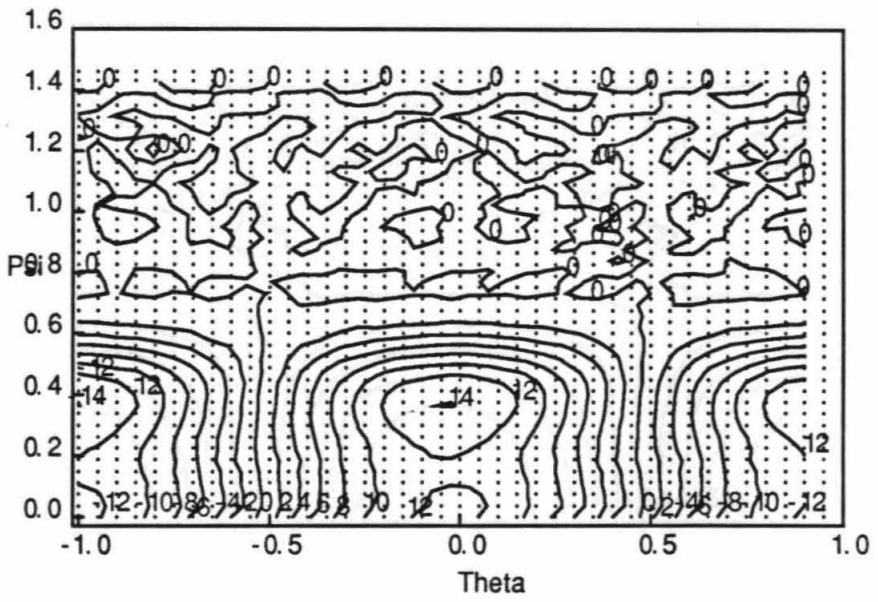

Figure 3

Finally, the success of the network shown in Figure 4 suggests a further wirelength-saving trick: the collection of partial input sums, one from each hill in Figure 4, which are then summed to produce the total input to a stripe-detecting neuron. The advantage of this is that if, for example, the hills are made of uniform shape (though not necessarily uniform

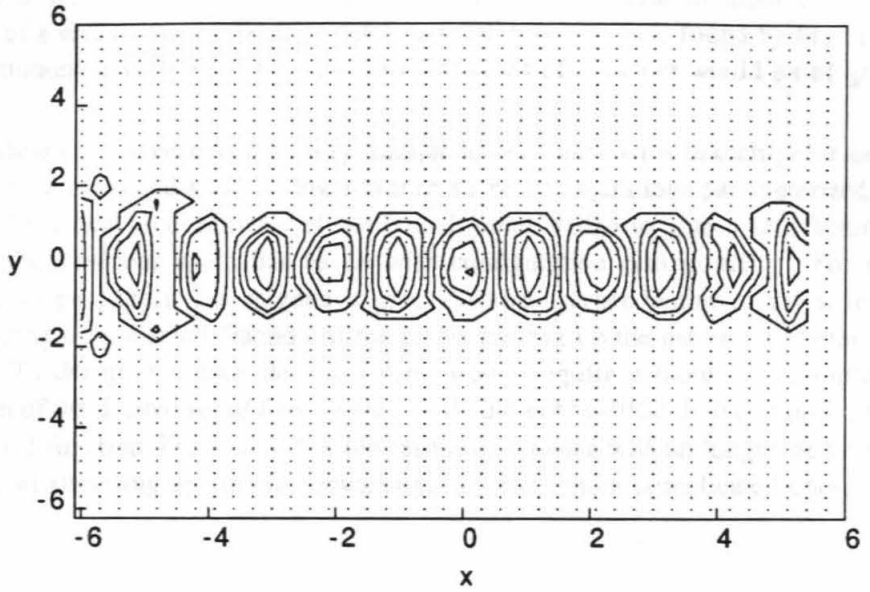

Figure 4

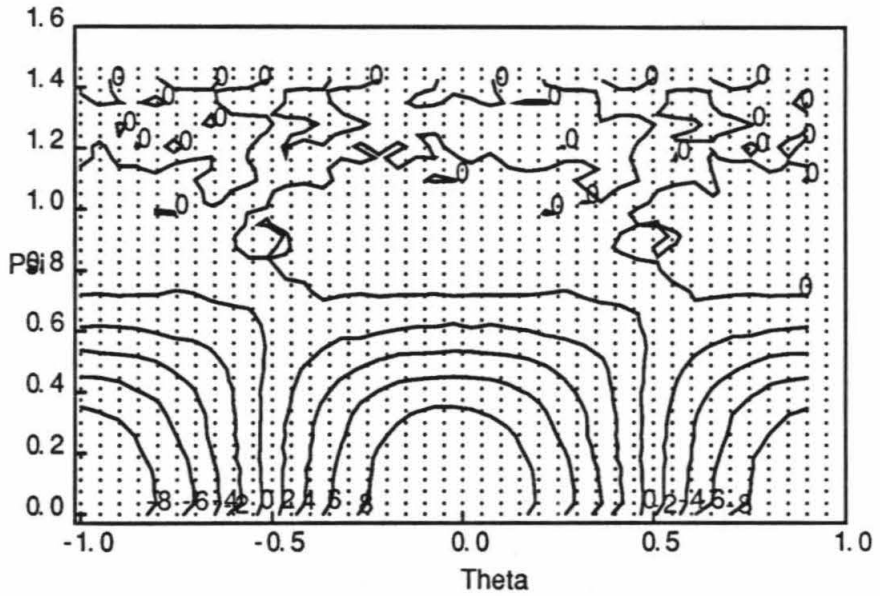

Figure 5

height), different S neurons separated by integral x can share most of the partial sums; the only difference between neurons would be which partial sums to add with what strengths. (If the hills in Figure 4 had been Gaussians or similarly uniform and circular, this trick would just amount to using lower resolution, i.e., sampling the image at no more than one pixel per

unit square.) Each partial sum could be gathered by a linear (non-thresholding) interneuron. (Interneurons are intermediate neurons which do not participate in input or output.) The possibility of a wirelength explanation for a new cell type was also found by Misha Mahowald [private communication]. Such functional explanations of structure would be of great interest in biology.

It should be recorded that very similar experiments with branch-point or ridge-end detectors, using eq(1) and a set of single-branch stripe configurations parameterized by a stripe angle and the x and y coordinates of the branch point, failed to give a satisfactory response function. The problem again has to do with configuration orthogonality. For example, a defect in a stripe pattern can migrate a great distance perpendicular to the stripe direction while changing only a small fraction of the pixels making up the pattern (a matter of distance vs. area). To design a branch detector, then, would require a more direct approach to the stabilization of the desired set of fixed points. It should not be difficult to compute a satisfactory neighborhood function $T(\vec{x} - \vec{x}_0)$, but the resulting network will no longer be an example of the method of summing up simple networks to construct more complicated ones.

II.3 A. Hallucinating Network

As discussed in Chapter I, one way to approach the image-processing portion of the fingerprint identification problem is to create a network which turns random input into a stable fingerprint-like pattern. The demand for scale-invariant processing (so that there is no preferred ridge width) and the need to minimize total wirelength jointly encourage hierarchical designs. How can such designs be integrated into neural network theory?

A hierarchical design may be thought of as a severe restriction on the synaptic matrix T_{ij}, forcing all but a few connections to be zero. The few nonzero connections are between an element of a hierarchy and its immediate neighbors: parents, children, and siblings, or bosses, subordinates, and coworkers, to use two social analogies. In practice, hierarchical designs often also allow next-nearest neighbor connections (e.g., aunts) out to some distance. The existence of a small number of allowed neighbors encourages one to try to specify the set of allowed connections geometrically, and to think of a hierarchy as a local network in some peculiar geometry. In sum, I propose to elevate the continuous and geometrical approach used in the stripe detector example to a general principle of network design, and in particular to design hierarchies this way.

II.3.1 A Continuous Hierarchy

The constraint that our network be robust appears at first to conflict with the scale invariance and wirelength minimization constraints: scale invariance and wirelength minimization draw us towards hierarchical designs but the neurons and synapses near the top of the hierarchy are not expendable; the network is especially vulnerable to defects affecting these neurons and synapses. A plausible solution to this difficulty, which also introduces geometric continuity and controllable fanout in a simple form, will be illustrated first for the case of a network with one-dimensional visual input and statistical properties similar to those of a binary tree with neurons at its nodes and synaptic connections at its links.

The standard technique for controlling wiring costs is to enforce some kind of locality on the connections in a network, i.e., to use short wires much more than long ones. In a d-dimensional medium (for integral d), the total cost of the wires of length between l and $l+dl$ is roughly [the number of wires in the interval $(l, l+dl)$] x [the cost of a wire of length l]. Here the cost of a wire is taken to be its volume, proportional to l^d to produce logarithmic, or nearly constant, time delays as a function of l. The reasons for this scaling are set forth in [Mead and Rem 82; Mead 83] and are related to the use of exponential amplifier horns to drive long wires. For sufficiently short metal wires on present-day chips the wire resistance does not much limit the signal delays so that the wire volume required to produce constant delays is just proportional to the wire length, l.

To determine the number of wires $N(l)dl$ in the interval $(d, d+dl)$, we guess that each order of magnitude $(l, 10l)$ should have about the same wire volume spent on it. This assumption is reasonable since it multiplies the total wiring cost by the logarithm of the system size, by comparison with a purely locally interconnected network. The logarithm is the minimum factor required to accommodate the wiring costs of a binary tree.

Introducing $x = \log l$, the equal-cost-per-decade condition means that for constant Δx (as a function of x) one has

$$\int_x^{x+\Delta x} N(l) l^d dl = \text{constant}(x)$$

$$0 = \frac{d}{dx}\int_x^{x+\Delta x} N(l)l^d dl = \frac{d}{dx}\int_x^{x+\Delta x} N(l)l^{d+1}dx$$

$$0 = N(l)l^{d+1}\Big|_x^{x+dx}$$

which implies that $N(l) \propto l^{-(d+1)}$. In VLSI d=3 for wire physics and the result is $N \sim l^{-4}$. For short wires this expression is modified to $N \sim l^{-2}$. The effect of this modification is to favor the longest wires whose resistance is still negligible.

Thus, if we have many short wires we can afford some long wires, and vice versa; the costs of short and long wires are balanced when the number of wires of length l is $l^{-(d+1)}$. Long wires are important since few computations can be continuously mapped onto the d-dimensional medium (d=2 or 3) available to us while preserving spatial locality in that medium. Hierarchical designs such as the binary tree are examples of local organizations with a few long wires; one can map a binary tree into a d-dimensional medium so that the number of wires of length l is proportional to $l^{-d}\frac{dl}{l}$ [Mead 83]. Literal hierarchies such as the binary tree have the drawback, however, that, although they use long and short wires in a balanced way, they do not use them well except for $d \approx 1$. The tree is equally mappable into any dimension $d > 1$; it does not make use of the increasing connectivity of higher dimensions. In a tree, every connection is a bottleneck. Quantitatively, if we take a patch of a d-dimensional local network whose shape has been arranged to minimize the number of external connections W for a given number of internal neurons N (in this case the patch will be a sphere), we get

$$W = N^{1-1/d}. \tag{5}$$

$$(W = \text{surface} = L^{d-1};\ N = \text{volume} = L^d).$$

For a similar patch of binary tree the answer depends on the binary expansion of N but if we average over a factor of 2 or more of possible N values, we get

$$W = \log N, \text{ or}$$

$$d_{\text{tree}} \to 1 (= \text{“}1 + \epsilon\text{”}).$$

This definition of a network's "bottleneck dimension" as $\lim_{N\to\infty} 1/(1 - \log W/\log N)$ does not refer to a network's spatial embedding; it is an intrinsic dimension. This definition has the consequence that for two graphs of equal N, the one with larger dimension cannot be embedded in the other because there are not enough wires available.

If d is the integer dimensionality of an image then the bottleneck dimension of an image-processing network may be reduced to that of a tree over d dimensions, $d + \epsilon$, by the following geometric construction, illustrated for $d = 1$. The binary tree in Fig. 6 has the following scaling behavior: that when $x_{\max}$ is doubled and $l_{\max}$ is increased by one, the resulting network is nearly isomorphic to the original one, differing in having higher "resolution" (an extra layer of neurons at the bottom). In the case $l_{\max} \to \infty$, the isomorphism would be complete. The few special long wires near $l = l_{\max}$ appear to preclude any local geometric interpretation of Fig. 6, but setting $z = 2^l$ (Fig. 7) reveals a geometric interpretation: neurons may be connected only when they lie within a z-dependent neighborhood size. The size changes with z to maintain constant fanout. The fact that each neuron has <u>two</u> possible

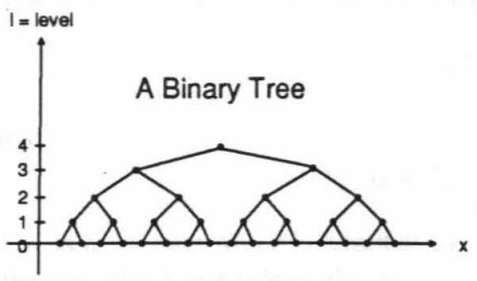

Figure 6

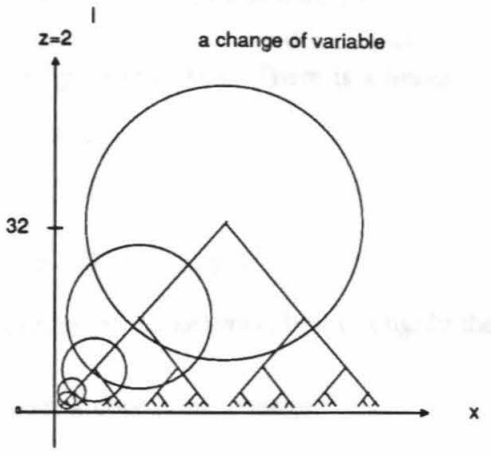

Figure 7

neighborhood radii, differing by a factor of two, is awkward and will be further discussed in the next section where non-spherical neighborhoods will be introduced.

Of course, these neighborhoods could also be drawn on Figure 6, where they would appear as highly squashed ellipses. In Figure 7 they are circular and are determined by just one number: the radius. This radius is roughly the mean spacing of neurons as a function of z, so that the neighborhood sizes can be specified by a density of neurons $\rho(z)$, with radius $= R(z) \approx \rho^{-1/2}(z)$. For a binary tree, $\rho(z) = z^{-2}$. The scaling law is also convenient to state: under a change of coordinates $x \to \lambda x, z \to \lambda z$ one gets "approximately" the same network, with extra resolution near $z = 0$. This means that we have the approximately scale-invariant quantities

$$\theta = \arctan \frac{(x_0 - x_1)}{(z_0 - z_1)}$$

and

$$\gamma = r/R(z) = \frac{\sqrt{(\Delta x)^2 + (\Delta z)^2}}{R(z)}$$

to describe the neighborhood of a neuron. Thus we have some useful parameters out of which to construct approximately scale-invariant networks. In Fig. 7, once it has been decided that two neurons have $\gamma < 1$ and can thus be connected, their connection strength may be determined by complicated rules having to do with binary numbers. We make this second decision based on simpler rules involving γ and θ. Thus T_{ij} will be a function of γ and θ, and a function of x_0, x_1, z_0 and z_1 only through these quantities. Also, rather than relying on a highly patterned grid of neuron locations, we will allow any set of locations which produces the density function ρ with sufficient accuracy.

This description of a binary tree is interesting because it also covers several other networks. By taking $\rho(z) =$ constant we get a two-dimensional local network, which does not suffer the robustness problem associated with top-level neurons but does suffer long communication delays due to its lack of long wires. It is also very much more expensive than the tree. By considering $\rho(z) \propto z^{-\alpha}, 0 < \alpha < 2$, one may hope to get both long wires and behavior continuity. The proposal, then, is for local rules in a space with fractional dimension $1 + \epsilon$ $(1 \leq 1 + \epsilon \leq 2)$, where ϵ is now a real number rather than an infinitesimal and depends on α

in a calculable way. Clearly, wirelength minimization requires that ϵ be minimized.

The image-processing version of these ideas involves a $2+\epsilon$-dimensional network with ordinary dimensions x and y and a third hierarchy coordinate z. There is a density function

$$\rho(z) = K^{3-\alpha} z^{-\alpha} \tag{6}$$

where

$$0 \leq \alpha \leq 3, \ \frac{1}{K} \leq z \leq 1, \ 0 \leq x, y \leq 1$$

and K is a constant which determines the resolution of the network, being roughly the number of neurons along one side of the image.

The scaling behavior is as follows: under a coordinate change

$$x' = \lambda x, y' = \lambda y, z' = \lambda z$$

we use

$$\rho(z)dxdydz = \rho'(z')dx'dy'dz'$$

to deduce

$$\begin{aligned} \rho'(z') &= \lambda^{-3}\rho(z) \\ &= (K/\lambda)^{3-\alpha}(\frac{z}{\lambda})^{-\alpha} \\ &= (K/\lambda)^{3-\alpha}(z')^{-\alpha} \end{aligned}$$

so that the network can be self-similar if its behavior is independent of its granularity, K. There are similar local statistical systems in condensed matter physics which have this property at phase transitions (i.e., after the careful adjustment of one parameter) and we would like to find a corresponding behavior in this case to obtain scale-invariant pattern recognition. For the connection between neural networks and statistical systems see [Hopfield 82].

In Appendix II the (intrinsic) bottleneck dimension and an embedding dimension are calculated as a function of α; they are

$$d_{\text{bottleneck}} = \begin{cases} 3 & \text{if } \alpha < 1 \\ \frac{3}{\alpha} & \text{if } 1 < \alpha < \frac{3}{2} \\ 2 & \text{if } \frac{3}{2} < \alpha \end{cases} \tag{7a}$$

and

$$d_{\text{embedding}} = \max(2, 3-\alpha, 3-2\alpha/3). \tag{7b}$$

These are compared in Fig. 8, and agree for $\alpha > \frac{3}{2}$. In the simulations reported in the rest of this chapter, α was set to 2.1.

II.3.2 Necessary Refinements of the Hierarchy

Experience shows that the foregoing construction of a fractional-dimension network must be refined in several ways if the networks are to be computationally reasonable. First, it is not sufficient to distribute all the neurons randomly and independently with density $\rho(z)$. Severe percolation and bottleneck problems arise from the variance associated with such a distribution, unless the number of neurons so distributed, and hence the wirelength cost, is

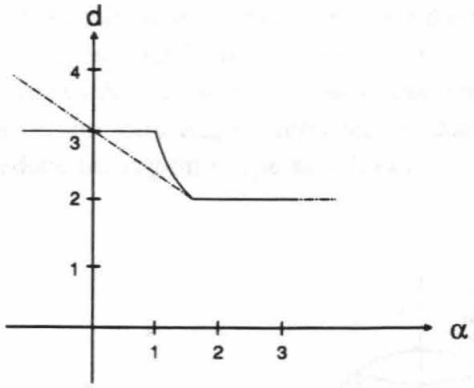

Figure 8

unreasonably high. A more uniform distribution scheme for neurons, still consistent with $\rho(z)$, is to divide (x, y, z) space into cubes whose size depends on z and place a neuron or a clump of neurons at the center of each cube, as in Fig. 9. This divides the (x, y, z) space into layers of generally incommensurate width. The relative x and y positions of cubes in different layers are determined by pseudorandom numbers. Many other regular placement schemes for neurons are possible, but we will use this one. For simplicity, periodic boundary conditions are used. The cube sizes were altered slightly so that an integral number of identical cubes would fit in a layer.

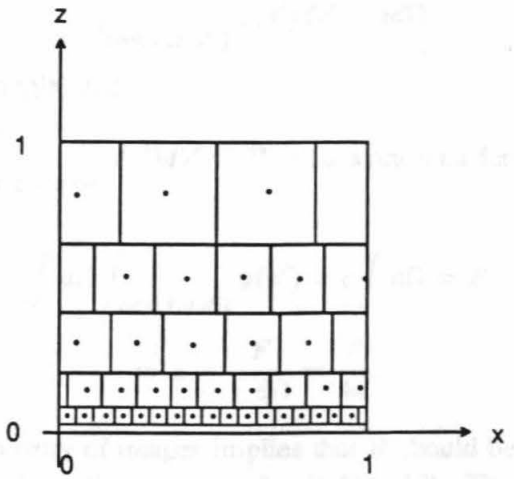

Figure 9

Another necessary refinement has to do with maintaining reasonable fanout and fanin. For $\alpha = 0$, where we have a 3-dimensional network, neurons can be connected if they lie within a sphere of radius $R \propto \rho^{-1/3}(z)$. This is not always a good estimate of the interneuron distance, due to the presence of a singularity in $\rho(z)$ at $z = 0$. The problem is that the interneuron distance may change significantly over $\Delta z \approx$ the interneuron distance, so that within a sphere at z_0 there are many more neurons with $z < z_0$ than with $z > z_0$, and the fanin depends drastically on θ, the polar coordinate. This undermines buildability since the minimal fanin required to prevent bottlenecks will be associated with an unacceptably large

maximal fanin from another direction. It is necessary to alter the shape of the neighborhood from a sphere to some kind of egg so that fanin will be independent of θ. More precisely, we want a region described by a radius $R(z,\theta)$ which varies so that the number of neurons within a narrow cone depends only on the solid angle subtended by that cone. Fig. 10a illustrates the situation, and we may deduce the region shape as follows:

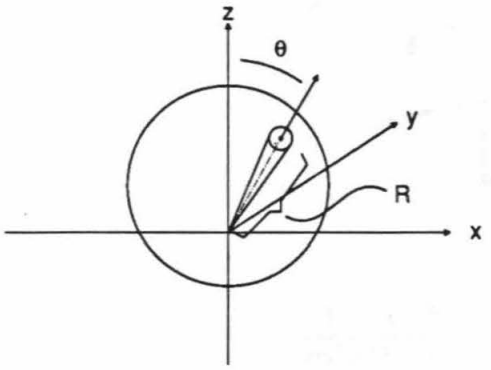

Figure 10a

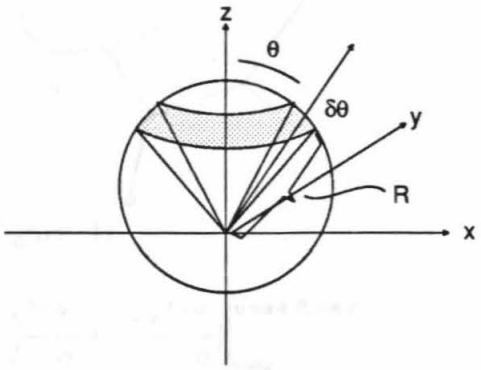

Figure 10b

$$\int_{\text{cone}(\theta,\phi,d\Omega)} \rho(z')dV = cd\Omega \tag{8}$$

(where Ω represents solid angle) and

$$\int_{\text{egg volume}} \rho(z')dV = F = \text{constant total fanin}$$

so that

$$\int d\Omega \int_{\text{cone}(\theta,\phi,d\Omega} \rho(z') = c\int d\Omega = F \tag{9}$$

$$\Rightarrow c = \frac{F}{\int d\Omega} = \frac{F}{4\pi}.$$

Now rotational symmetry of images implies that R should be independent of ϕ so we can integrate (6) over ϕ to obtain the solid annulus in Fig. 10b. Then

$$\int_{\text{annulus}(\theta,d\theta)} \rho(z')dV = \frac{F}{4\pi}\int_{\text{annulus}} d\Omega = \frac{1}{2}F\sin\theta d\theta$$

Introducing r through $z' = z + r\cos\theta$ and recalling that $\rho(z')= K^{3-\alpha}(z')^{-\alpha}$, we can compute

$$\frac{1}{2}F\sin\theta d\theta = K^{3-\alpha}2\pi\int_0^R r^2 dr\int_\theta^{\theta+d\theta} \sin\theta'(z + r\cos\theta')^{-\alpha}d\theta'$$

$$\frac{FK^{\alpha-3}}{4\pi} = \int_0^{R(\theta)} r^2(z + r\cos\theta)^{-\alpha}dr$$

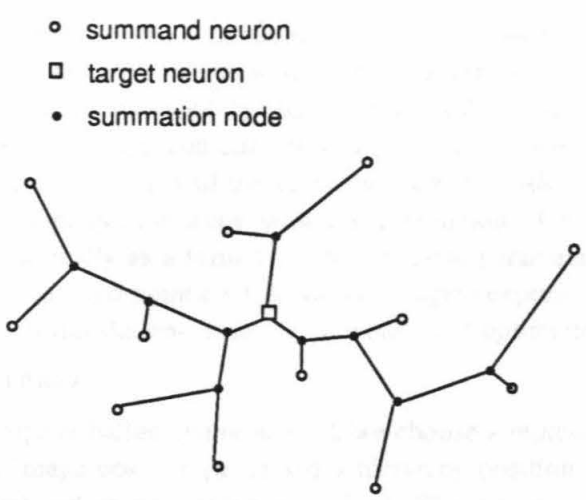

Figure 11

$$= \frac{1}{\cos^3\theta}\left[\frac{v^{3-\alpha}}{1-\alpha} - 2z\frac{v^{2-\alpha}}{2-\alpha} + z^2\frac{v^{1-\alpha}}{1-\alpha}\right]\Bigg|_{v=z}^{v=z+R\cos\theta}$$

yielding

$$\frac{FK^{\alpha-3}\cos^3\theta}{4\pi} + z^{3-\alpha}\left(\frac{1}{3-\alpha} - \frac{2}{2-\alpha} + \frac{1}{1-\alpha}\right) =$$

$$\frac{(z+R\cos\theta)^{3-\alpha}}{3-\alpha} - 2z\frac{(z+R\cos\theta)^{2-\alpha}}{2-\alpha} + z^2\frac{(z+R\cos\theta)^{1-\alpha}}{1-\alpha} \tag{10}$$

as an implicit equation for $R(z,\theta,K,F)$, to be solved by Newton's method. In simulations, this equation was solved for several values of θ per z value and linear interpolation used for other θ values.

In particular, for $\alpha \to 0$,

$$\frac{FK^{-3}}{4\pi} = \int_0^{R(z,\theta)} r^2 dr = \frac{1}{3}R^3(z,\theta)$$

$$\text{so}\frac{F}{\frac{4}{3}\pi R^3} = K^3 = \rho$$

and R is independent of z and θ in this, the three-dimensional case.

As has been mentioned in connection with Figures 4 and 5, there is another absolutely necessary refinement of our networks: the use of branching wires to save wirelength. The summation of a neuron's many inputs can be done in several stages, since addition is associative: sums of sums are sums. Then each summand requires a short wire instead of a long one, as in the tree design in Figure 11. There are a few long wires as overhead: several long wires per neuron + one short wire per synapse, vs. one long wire per synapse in the more straightforward implementation where each T_{ij} corresponds to a wire. This wirelength-conserving measure requires a new network element, beyond synapses and neurons: the summation node. A summation node acts like a neuron without the threshold and the nonlinear transfer function; i.e., it is a linear neuron. In biological and electronic technology, branching wires may serve well enough if one is summing currents; for voltages one may get an average rather than a sum. This may also be acceptable if the input tree is balanced. There are similar prospects for the use of output trees.

Given these refinements of the continuous hierarchy idea, we have a geometric approach to network design that provides the connection sparseness information, necessary on wirelength cost grounds, but which is not provided by the other methods of network design which we have discussed: network summation and control of fixed points. Nevertheless, these methods are still useful for providing the values of the sparse nonzero synaptic weights. In this chapter we will combine the continuous hierarchy with the summation of more easily understood networks expressed algebraically as a formula with adjustable parameters for T_{ij}. In Chapter III we will stabilize selected fixed point configurations by again expressing the synaptic matrix as a function of scale- and translation- invariant variables, and optimizing the function.

II.3.3 The Synapse Formula

To create a fingerprint-hallucinating network we choose a representation in which each neuron has not only an image position (x, y) and a hierarchy position z but also a "color" c = 0 or 1 (black or white) and an orientation $\phi \in [0, \pi)$. The orientation is supposed to be a distributed representation of the local ridge or valley orientation. Since ridges do not have a signed direction, the orientation is only defined up to multiples of π. Since the argument in Appendix I was not known at the time this network was designed, its neurons will take the values ± 1. The network's local representation for a north-running valley, then, is all neurons having value -1 (off) save the color=0, orientation= $\pi/2$ neurons that have value +1. This amounts to a unary representation of angle and, indeed, of each of the five indices, though in the case of the z index it is more like a unary representation of $\log(z)$.

To specify the network it suffices to give the $\vec{w} = (x, y, z, \phi, c)$ coordinates of the neurons, and the connections as a function of $\vec{w}_i$ and $\vec{w}_j$. As in Figure 6, (x, y, z) space was divided into cubes whose size depended only on z so as to produce the desired density function $\rho(z)$. At the center of each cube was placed one neuron for each combination of two colors and m angles of the form $i\pi/m$, $i = 0, \ldots, m-1$ and $m = 4$ or 6. Thus, there were $2m$ neurons in each cube.

It remains to specify the synapse and threshold values for a neuron at $\vec{w}_i$, and a representative input-supplying neuron at $\vec{w}_j$, as a function of $\theta, \gamma, \phi_0, \phi_1, c_0, c_1$, and $\psi = \arctan(y/x)$ (also a scale-invariant quantity). The synapse value T_{ij} was

$$T_{ij} = f(\theta)g(\gamma)\Big\{-I\delta(\gamma)+a[\frac{1}{2} - \sin^2(\phi_i - \phi_j)] + \sigma_{c_j}^{c_i}[1 - 2c\gamma \sin^2\theta \sin^2(\psi - \phi_i)]\Big\} \qquad (11)$$

where

$$\sigma_{c_j}^{c_i} = \begin{cases} +1 & \text{if } c_i = c_j \\ -1 & \text{otherwise} \end{cases}$$

and the threshold was $H(z)$= $H \times$ [total input to a neuron if all its neighbors were on]. Because the color and angle terms each average to 0, the total synaptic input to a neuron comes from $I\delta(\gamma)$ and is just the sum of the synaptic weights from other neurons within its cube, or "clump." This sum in turn is dominated by the $I\delta(\gamma)$ term in the present networks (since I is relatively large) and it may be best to simply let $h(z)$= $-(2m-1)HIf(\frac{\pi}{2})g(0)$. The parameters c, a, I and H are adjustable constants and f and g are adjustable functions. Although not a sum of outer products, this connection matrix is a sum of intuitively understandable terms. The meanings of the various terms are:

$g(\gamma)$: g =0 for $\gamma > 1$ cuts off synapses outside the neighborhood. Otherwise, we took $g = 1$.

$f(\theta)$: A function ≥ 0 which if peaked at $0, \frac{\pi}{2}$ or π would favor top-down organization, independent two-dimensional networks, or bottom-up organization, respectively.

$-I\delta(\gamma)$: $\delta(\gamma)= 1$ for $\gamma = 0$, 0 otherwise. δ doesn't have to be so discontinuous. This term is a mutual inhibition term which allows us to control the number of +1 neurons in a cube; we want one on out of every $2m$. In the networks which work, $I\delta(\gamma)$ is numerically the largest of the terms in T; the others are corrections to it. The effect of this term is to convert the $2m$ neurons in a cube into a $2m$-way "flip-flop", or mutually inhibiting neuron clump. The entire network may be thought of as interacting $2m$-flops, instead of interacting neurons. If $\delta(\gamma)$ were more spread out spatially, one could still control the average number of neurons turned on in a region of the network but not the number turned on in one $2m$-flop.

H(average input): H determines how many winners there can be in the competition caused by the mutual inhibition term $-I\delta(\gamma)$. $H \approx 1.05$ forces one winner; $H \approx 1.2$ forces zero for the parameter values used in this work. In detail, the threshold h for a neuron is computed by multiplying H by what would be the total input to a neuron if all its neighbors were on. If the network consisted of uncoupled clumps, which is a good first approximation due to the large values of I used, then H scales with I and $H = 0$ forces half of the ± 1 neurons in each clump to be on. Clearly, for large I we can consider just the synapse I terms in computing neural thresholds, so that thresholds become a property of a clump rather than of the entire network. Then H and I are the clump internal parameters, just as h is the neural internal parameter.

$a[\frac{1}{2} - \sin^2(\phi_i - \phi_j)]$: encourages conformity of internal angles. Both angular terms involve squares of trigonometric functions and can be rewritten as trigonometric functions of double angles. Hence, angles are only used modulo π, as promised.

$\sigma_{c_j}^{c_i} = \{+1$ if $c_i = c_j$; -1 otherwise$\}$ favors color conformity among neighbors.

$-2\sigma_{c_j}^{c_i}\gamma \sin^2(\theta)\sin^2(\psi - \phi_i)$: favors color contrast among neighbors of similar z, separated by a vector perpendicular to what $\vec{w}_i$ thinks is the local ridge axis, i.e., favors color contrast when ψ and ϕ_i differ by $\frac{\pi}{2}$. This encourages side-by-side ridges of alternating color. This term is the sole ridge-making term, and the only term which can allow represented and actual angles to interact. It also influenced the choice of representation: in this term two colors and one angle all interact, so that if separate color and angle neurons were used ($2+m$ instead of $2m$ neurons per site), this would be an illegal three-neuron interaction.

With this network it proved possible to hallucinate stripe patterns whose stripe width was roughly the size of the smallest spacing between neurons, but no larger. To get larger stripe widths two changes to Equation (11) were necessary, the first change innocuous and the second serious. These changes involved the parameters s_{off} and c', respectively, in the new version of the color conformity term:

$$\sigma_{c_j}^{c_i}\left[1 - 2(c + c'z)\gamma \max(0, \sin^2(\theta) - s_{\text{off}})\sin^2(\psi - \phi_i)\right]. \tag{12}$$

The s_{off} term confines the ridge-making interaction to pairs of neurons of roughly the same scale. Unfortunately, the $c'z$ term spoils the algebraic scale invariance of the synapse formula, but it was necessary to compensate for the observed network preference for small stripe widths. One probable consequence of this difficulty is that c and c' will have to be readjusted when the total number of neurons, or the resolution parameter K, changes.

II.3.4 Results

The network described by the synapse equations (11) and (12) was repeatedly run starting from random initial configurations. For efficiency's sake a nonrandom update order was used: the hierarchy levels were updated in order, and within one hierarchy level the clumps were updated in a scanned order, and within a clump the color and angle neurons were updated in a fixed order. Occasional comparisons with a random update order failed to reveal any systematic behavior differences between the two update methods; apparently, the random starting configuration was enough to eliminate any waves or other update-related phenomena. The visual appearance of the fixed-point configurations obtained for each run was used to manually adjust the free network parameters for the next run, with the purpose of producing scale-invariant stripelike patterns. The results are presented here.

II.3.4.1 Bottom-up Network Behavior

Various final configurations resulting from random initial conditions are shown in Figures 12-13, together with the values of the parameters $\alpha, a, c, I, H, m, f_0, f_1, f_2, f_3, f_4, f_5, K$, and F which produced them. In Figure 12 we see all four levels of a $K = 13$ network displayed, with color ± 1 neurons displayed separately for clarity. The bottom level is clearly a stripe pattern. Higher levels suffer from having more than one neuron on per 12-flop.

Note the small values of the top-down strengths f_0 and f_1 in these figures. Here $f(\theta)$ has been linearly interpolated between the points $(0°, f_0)$, $(80°, f_1)$, $(85°, f_2)$, $(105°, f_3)$, $(110°, f_4)$, and $(180°, f_5)$. For efficiency, θ itself was interpolated between stored arctangent values. Each parameter (save m) can be varied to some extent without affecting the results. The figures display only the bottom level of each configuration; for the present value of $f(\theta)$ the network is largely driven from the bottom up so that higher layers (those of lower resolution) look similar. Bottom-up behavior is encouraged by large $f(\theta)$ near $\theta = \pi$; independent layer behavior would be encouraged by θ near $\frac{\pi}{2}$; top-down behavior would be encouraged by a large weight near $\theta = 0$.

To adjust the parameters to produce figures 12-13 it is necessary first to get just one neuron out of each clump of $2m$ to be on at a time by setting the intraclump competition parameter, I, to an arbitrary positive value and adjusting the relative threshold H. This is best done in a network with uncoupled layers so that $f(\theta) = 0$ except near $\theta = \frac{\pi}{2}$. Then, angular conformity should be turned on with a. This produces domains of aligned internal angle. I must be large compared to a to maintain the integrity of a clump and prevent several neurons from turning on in one clump, but if it is too large then the network will "freeze" and fall into angularly disordered stable states. Finally, turning on c will stretch the domains in the direction of their internal angle and compress them in the perpendicular direction.

II.3.4.2 Hierarchically Balanced Behavior

Figures 12-13 are produced by networks with very weak top-down connections, and consequently the displayed configurations have narrow stripes, two or three lattice spacings wide. It proved sufficiently difficult to obtain wide stripes that extra terms were added to the synapse formula (Equation 12) to do so, as explained in Section II.3.3. With the extra terms, one of which eliminates the algebraic scale invariance of the synapse formula (Equation 11), one may also obtain wide stripes as shown in figure 14. In figure 15 we see a more subtle behavior, which appears to have a variety of stripe widths within one stable configuration

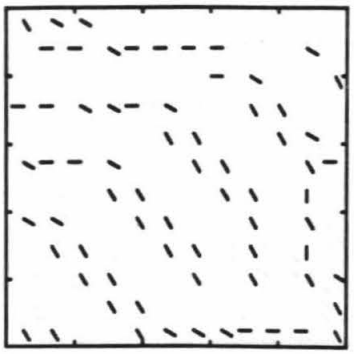

Figure 12a level 0 color 1

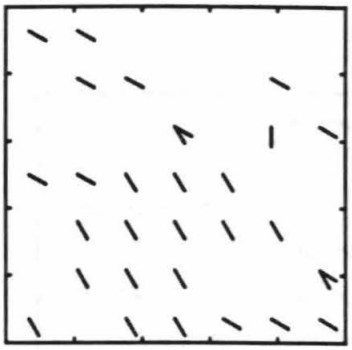

Figure 12b level 1 color 1

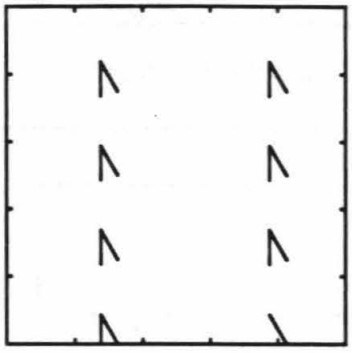

Figure 12c level 2 color 1

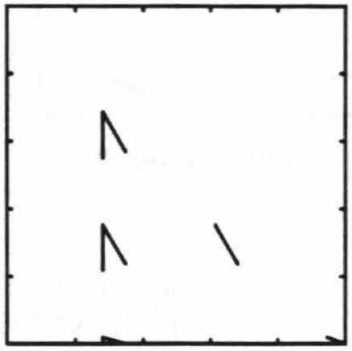

Figure 12d level 3 color 1

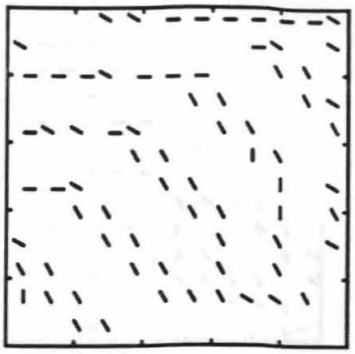

Figure 12e level 0 color -1

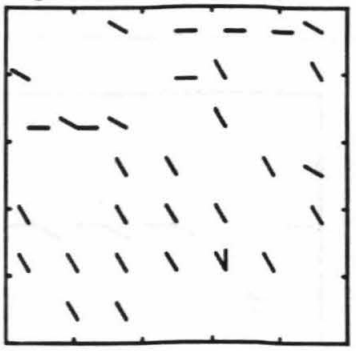

Figure 12f level 1 color -1

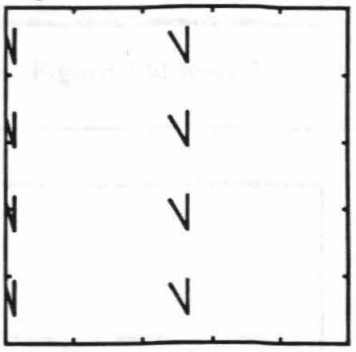

Figure 12g level 2 color -1

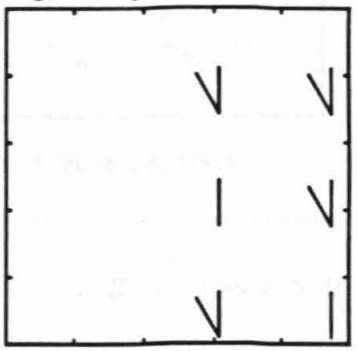

Figure 12h level 3 color -1

Figure 12: $a = 4\ c = 3\ i = 30\ K = 13\ m = 6\ H = 1.05$

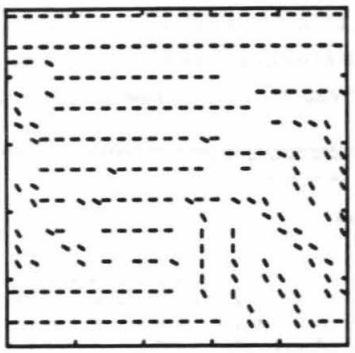

Figure 13a level 0

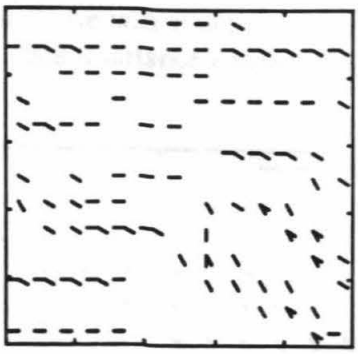

Figure 13b level 1

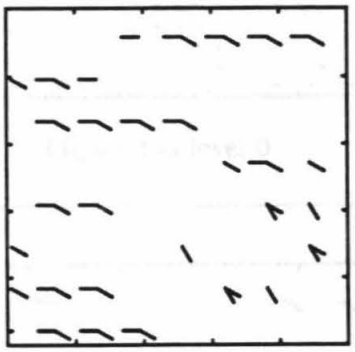

Figure 13c level 2

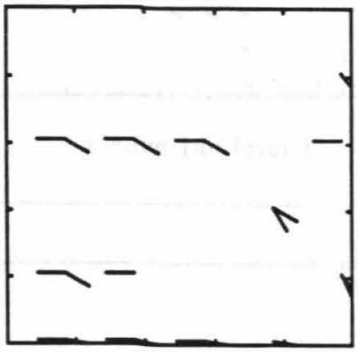

Figure 13d level 3

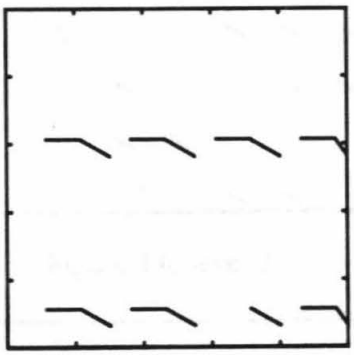

Figure 13e level 4

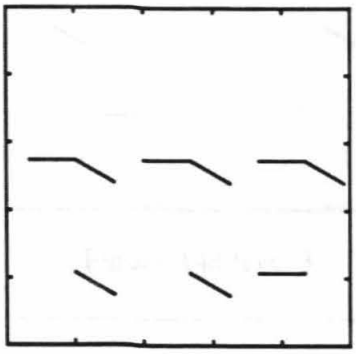

Figure 13f level 5

Figure 13: color=0 $a = 4$ $c = 3.5$ $i = 42$ nbrs $= 12$ $K = 23$ $m = 6$ $H = 1.05$ $\vec{f} = (5, 5, 10, 10, 10, 80)$

although none of the stripes is very long. Further, it appears that the scale invariance was obtained by creating small stripes inside of larger stripes of the opposite color. This is unfortunate, since one of the principal types of fingerprint noise which we would like to

eliminate is opposite-color holes in a stripe. In Chapter III we will examine networks at the scale of one stripe width and demonstrate simultaneous scale-invariance and hole suppression. The networks will, however, involve several new ideas.

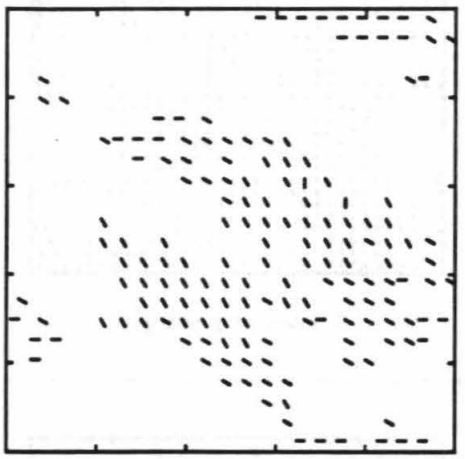

Figure 14a level 0

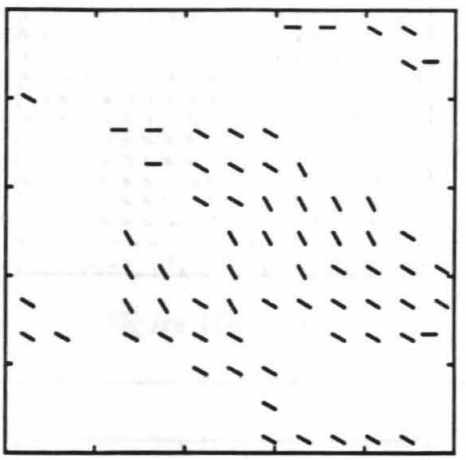

Figure 14b level 1

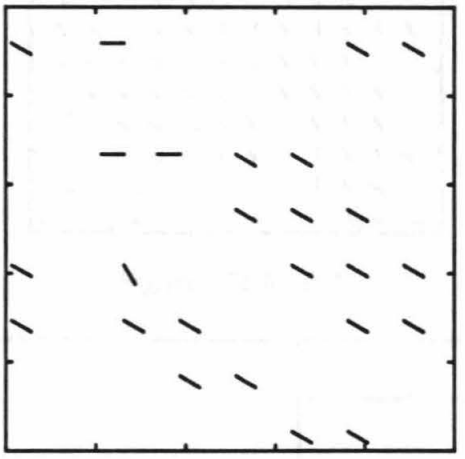

Figure 14c level 2

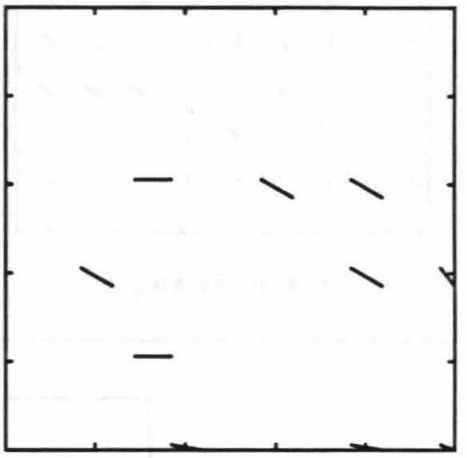

Figure 14d level 3

Figure 14: $a = 4$ $c = 4$ $c' = 6$ $i = 75$ $K = 23$ $m = 6$ $s_{off} = .5$ $H = 1.05$ $\vec{f} = (30, 30, 10, 10, 20, 20)$

It happens that a slight modification of the network of figure 18 produces another interesting behavior (figure 16): simultaneous presence of color=0, color=1 and uncolored (all neurons off) regions. This could be useful in tasks such as optical character recognition which require widely separated curves in an unoriented background.

The figures show that a variety of fingerprint-like images can be created, and therefore suggest that the most difficult constraint on our networks, that of doing a useful network

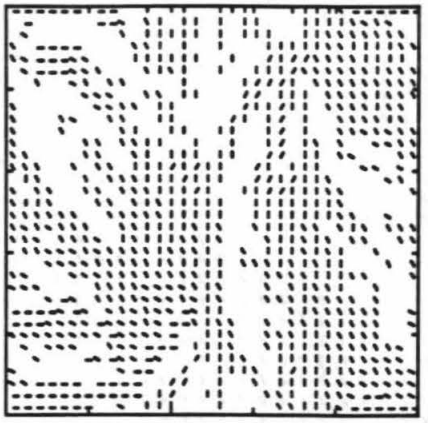

Figure 15a level 0

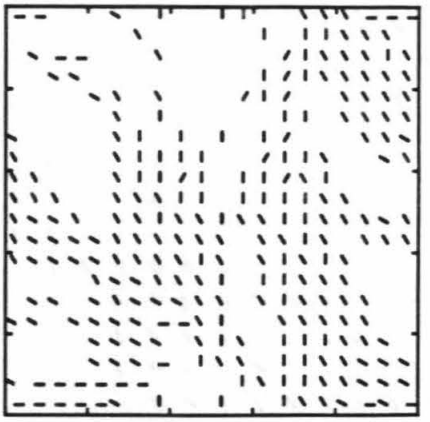

Figure 15b level 1

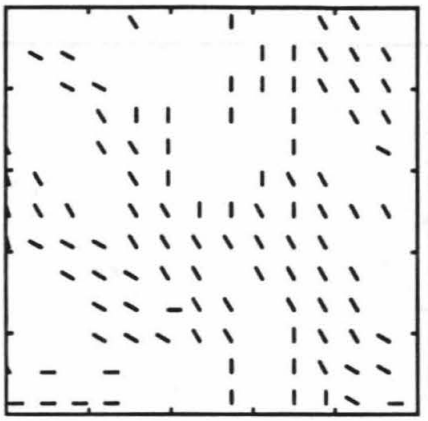

Figure 15c level 2

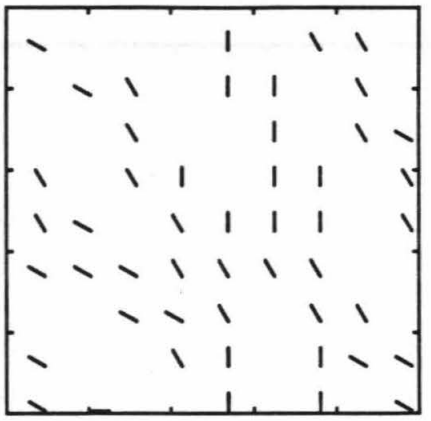

Figure 15d level 3

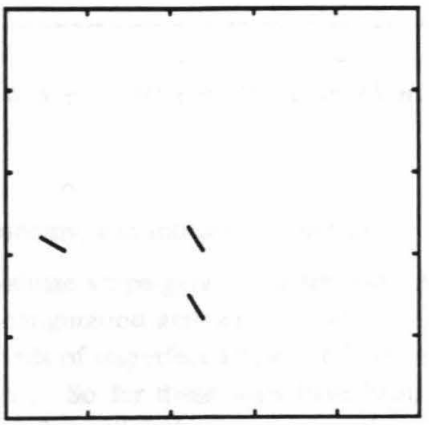

Figure 15e level 4

computation, can be met within a framework that has plausible solutions to the problems of

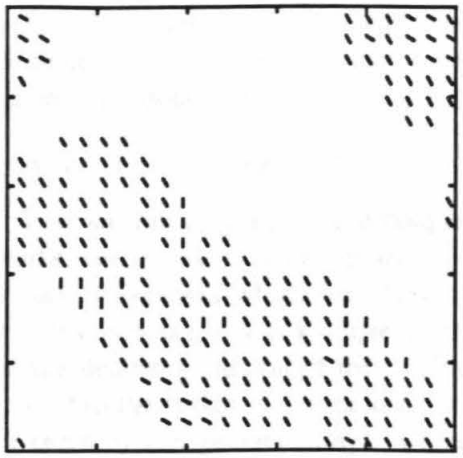

Figure 16a color 0 level 0

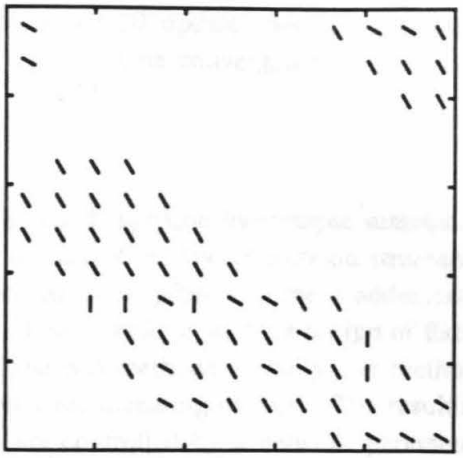

Figure 16b color 0 level 1

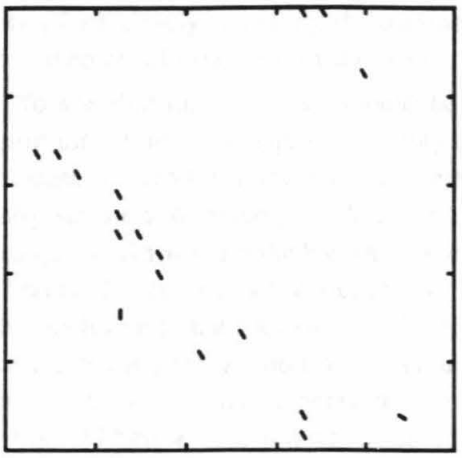

Figure 16c color 1 level 0

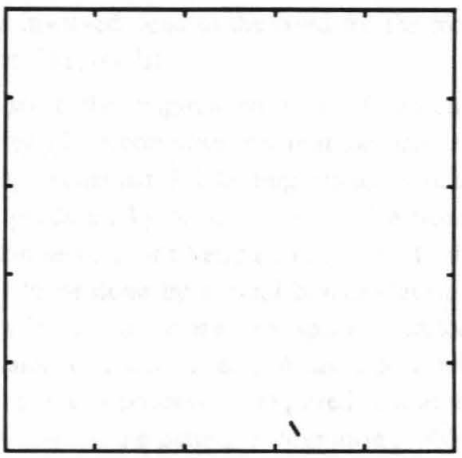

Figure 16d color 1 level 1

Figure 16: $a = 4$ $c = 3$ $c' = 10$ $i = 75$ $K = 23$ $m = 6$ $s_{off} = .5$ $H = 1.05$ $\vec{f} = (30, 30, 10, 10, 20, 20)$

scale invariance, wirelength economy, and robustness built in.

It is also possible to evaluate stripe-generating networks numerically, for example, by starting with a perfect stripe configuration and seeing how far it evolves before stopping, or seeing to what extent various kinds of imperfect stripe configurations correct their defects and approach a perfect stripe pattern. So far these tests have been about as reliable as visual observation of the fixed-point configuration; they gain in precision what they lose by failing to anticipate new defect types, such as compact regions where all neurons are off, or decoupled stripe patterns on different levels. In Chapter III we will use such tests more extensively on a somewhat different hierarchical network.

The configurations shown here typically took about 10 update sweeps through the network to converge, and were simulated on a VAX 11-750. One convergence took about 1 hr for $K = 13$, 2 hours for $K = 23$, and 6 hours for $K = 35$.

II.4. Conclusions

Two network programming techniques were discussed: synapse-by-synapse summation of networks, and the use of a continuous geometry to determine the "neighborhood structure", i.e., the set of synapses which are allowed to be nonzero. Hopfield's content addressable memory network was given as a simple example of the first technique, and the design of fixed-width stripe detectors introduced the combination of the two methods. Finally, the methods were applied to the design of a scale-invariant fingerprint-hallucinating network. The resulting network certainly can generate stripes whose widths are controlled by a network parameter, and is also capable of a limited form of scale invariance for a fixed setting of the network parameters.

These results were obtained at some cost: the algebraic scale invariance of the synapse formula is spoiled by the width-controlling parameter. This, and the increasing computational difficulty of effectively searching the parameter space involved, lead to the need for the more extensive theoretical treatment of the network given in Chapter III.

To see that our networks should be able to solve the original problem of detecting fingerprint branch minutia requires not only that Figures 12-13 convince one that the networks can eliminate fingerprint noise, but that there be some mechanism for finding branches in the artificially simple and stereotyped fingerprint pattern produced by removing the noise from a real fingerprint. Once again the hierarchical organization serves. At a height $z \approx$ the local ridge width, branch detection is a local operation which could be done by special branch-detecting neurons which fit into the framework of local rules in a fractional dimensional space; therefore, the wirelength, robustness, and scale invariance constraints can still be met. What is done with the branch detection signals depends on the further fingerprint processing required, but at this point we would have an explicit representation of the answer to the pattern-recognition problem posed at the outset.

The idea of local dynamics in a space of fractional dimension appears to be a fairly general principle for efficient network organization, and is conducive to the use of analogies and techniques from well-studied (local) systems of integral dimension.

Appendix I

There is a set of stripe patterns which define the configurations of S and P neurons which we would like to be fixed points. For a stripe configuration of angle ψ and phase θ, we may define

$$\text{Stripe}(\psi,\theta,\vec{x})=\begin{cases}+1, & \text{if position } \vec{x} \text{ is black in the given stripe field;}\\ -1, & \text{otherwise.}\end{cases}$$

We are to control the two possible values of P neurons, a_{min} and a_{max}, and the two possible values of S neurons, b_{min} and b_{max}, in order to make the various stripe configurations as orthogonal as possible.

If we call the entire configuration $C(\psi,\theta)=(P(\psi,\theta),S(\psi,\theta))$, then "maximizing the orthogonality" of these configurations certainly involves the inner products $C(\psi,\theta)\cdot C(0,0)$ in some combination. The maximum inner product is $C(0,0)\cdot C(0,0)$. Adopting the notation $\langle X\rangle_{\psi,\theta}$ to represent averaging X over all allowed ψ and θ, we will be able with our four parameters to set

$$\frac{\langle C(\psi,\theta)\cdot C(0,0)\rangle_{\psi,\theta}}{C(0,0)\cdot C(0,0)}=0$$

and to minimize

$$\frac{\langle [C(\psi,\theta)\cdot C(0,0)]^2\rangle_{\psi,\theta}}{(C(0,0)\cdot C(0,0))^2}.$$

The stripe configuration has

$$P_{\vec{x}}(\psi,\theta)=\frac{a_{max}+a_{min}}{2}+\frac{a_{max}-a_{min}}{2}\ \text{Stripe}(\psi,\theta,\vec{x})$$

$$\equiv a^{+}+a^{-}\ \text{Stripe}(\psi,\theta,\vec{x})$$

and similarly

$$S_{\vec{x},\phi,c}(\psi,\theta)=b^{+}+b^{-}[\delta^{\psi}_{\phi}-1+\delta^{\psi}_{\phi}\sigma^{c}_{1}\ \text{Stripe}(\psi,\theta,\vec{x})]$$

where

$$\delta^{a}_{b}=\begin{cases}1, & \text{if } a=b;\\ 0, & \text{otherwise,}\end{cases}$$

$$\sigma^{a}_{b}=\begin{cases}1, & \text{if } a=b;\\ -1, & \text{otherwise.}\end{cases}$$

Then

$$P(\psi,\theta)\cdot P(0,0)=(a^{+})^2+a^{+}a^{-}\langle\ \text{Stripe}(0,0,\vec{x})\rangle_{\vec{x}}+a^{+}a^{-}\langle\ \text{Stripe}(\psi,\theta,\vec{x})\rangle_{\vec{x}}$$

$$+(a^{-})^2\langle\text{Stripe}(\psi,\theta,\vec{x})\ \text{Stripe}(0,0,\vec{x})\rangle_{\vec{x}}$$

$$=(a^{+})^2+(a^{-})^2\delta^{\psi}_{0}I(\theta)$$

where

$$I(\theta)=\begin{cases}1-2\theta, & \text{if } 0\le\theta\le 1;\\ 1+2\theta, & \text{if } -1\le\theta\le 0;\\ \text{determined by periodicity,} & \text{otherwise.}\end{cases}$$

Likewise,

$$S(\psi,\theta)\cdot S(0,0)=\sum_{c,\phi}\left[(b^+)^2+b^+b^-(\delta_\phi^\psi-1+\delta_\phi^0-1)+(b^-)^2(\delta_\phi^\psi-1)(\delta_\phi^0-1)+(b^-)^2\delta_\phi^\psi\delta_\phi^0\delta_0^\psi I(\theta)\right]$$

$$=2A\left[(b^+)^2+2b^+b^-(\frac{1}{A}-1)+(b^-)^2(1-\frac{2}{A}+\delta_0^\psi)\right]+2(b^-)^2I(\theta)\delta_0^\psi$$

where A is the number of distinct angles.

We then compute that

$$\frac{1}{2}C(\psi,\theta)\cdot C(0,0)=t_1+\delta_0^\psi[t_2+t_3I(\theta)]$$

where

$$t_1=\frac{1}{2}(a^+)^2+(A-1)(b^+-b^-)^2+(b^+)^2-(b^-)^2$$

$$t_2=(b^-)^2$$

$$t_3=\frac{1}{2}(a^-)^2+(b^-)^2.$$

The interesting quantities are

$$\langle\frac{1}{2}C(\psi,\theta)\cdot C(0,0)\rangle_{\psi,\theta}=\frac{1}{A}\sum_\psi\frac{1}{2}\int_{-1}^{1}d\theta\frac{1}{2}C(\psi,\theta)\cdot C(0,0)=t_1+\frac{1}{A}t_2$$

$$\frac{1}{2}C(0,0)\cdot C(0,0)=t_1+t_2+t_3$$

$$\langle\frac{1}{2}[C(\psi,\theta)\cdot C(0,0)]^2\rangle_{\psi,\theta}=t_1^2+\frac{2}{A}t_1t_2+\frac{1}{A}[t_2^2+\frac{t_3^2}{2}]$$

since $\langle I(\theta)\rangle_{\psi,\theta}=0$.

Introducing the scaled variables $\tilde{a}=a^+/b^-$, $\tilde{b}=b^+/b^-$, $\hat{a}=a^-/b^-$, the first requirement that $\langle C\cdot C\rangle=0$ reduces to

$$\frac{1}{2}\tilde{a}^2+(A-1)(\tilde{b}-1)^2+\tilde{b}^2=1-\frac{1}{A}$$

which, as $A\to\infty$, can only be satisfied if $\tilde{b}\to 1$. This in turn implies that $\tilde{a}\to 0$. Only $\hat{a}$ remains undetermined, and it is determined by the second requirement that we minimize

$$R=\frac{\langle[\frac{1}{2}C\cdot C]^2\rangle}{[\frac{1}{2}C(0,0)\cdot C(0,0)]^2}\approx\frac{1}{2A}\frac{(\hat{a}^2+2)^2+8}{(\hat{a}^2+4)^2}$$

which implies $\hat{a}^2=2$. Any finite $\tilde{a}\neq 0,\pm\infty$ can be absorbed into the relative values of weights between neurons of different types, and we are interested in doing this to set $\hat{a}=1$ and thereby obtain the standard types of neurons, $a_{max}=1, a_{min}=-1, b_{max}=1, b_{min}=0$.

Appendix II

There are many plausible definitions of dimension for a network; one intrinsic dimension is the bottleneck dimension, which we will compute. This dimension is defined by

Equation (5) in the text. Instead of minimizing the number of wires poking through an arbitrary surface, we will use a less general surface: a prism with height $\Delta z = h$ and sides $\Delta x = \Delta y = w$. The number of neurons inside such a prism is

$$N = w^2 \int_z^{z+h} \rho(z)dz \text{ where } \rho(z) = K^{3-\alpha} z^{-\alpha} \tag{A1}$$

and the number of wires cleaved by its sides is

$$W = \int_S \rho(z)\rho^{-1/3}(z)dS \tag{A2}$$

$$W = w^2\rho^{2/3}(z)\Theta(z - K^{-1}) + w^2\rho^{2/3}(z+h)\Theta(1-z-h) + 4w\int_z^{z+h} \rho^{2/3}(z)dz\Theta(1-w) \ (w \leq 1) \tag{A3}$$

where Θ is a step function ($\Theta(x) = 1$ if $x > 0$, otherwise 0) and accounts for the possible absence of various sides when w, z or h adopt extreme values. W is to be minimized while N is held fixed; then their asymptotic relationship (as $N \to \infty$) is to be extracted. Since (for $\alpha \geq 0$) the greatest concentration of wires is at the bottom surface of the prism, we do well in minimizing W to eliminate this term at the outset by taking $z = K^{-1}$. Then if we set

$$a = wK \text{ and } b = hK$$

we find

$$N = a^2 \frac{[(1+b)^{1-\alpha} - 1]}{1-\alpha}$$

$$W = a^2(1+b)^{-2\alpha/3}\Theta(K-b) + 4a\frac{[(1+b)^{1-2\alpha/3} - 1]}{1-2\alpha/3}\Theta(K-a).$$

Since we are taking the limit $N \to \infty$, we may generally consider $b \gg 1$ to simplify the algebra. However, there is a subtlelty which prevents us from actually allowing $a = K$ as a possible prism: we must first take the limit $K \to \infty$, and then $N \to \infty$ rather than vice versa. $a = K$ implies

$$N = K^2 \frac{[(1+b)^{1-\alpha} - 1]}{1-\alpha} \geq O(K^2)$$

so that we cannot look at any finite N to extract the desired dimension. For $\alpha > 1$, however,

$$N \approx a^2/(\alpha - 1) \text{ (since } b \gg 1)$$

and there is no such exclusion of $b = K$.

For $b \gg 1$ there are three cases, according to whether $\alpha < 1$, $1 < \alpha < \frac{3}{2}$, or $\frac{3}{2} < \alpha$.

Case $\alpha < 1$: $(1+b)^{1-\alpha} \approx b^{1-\alpha} \gg 1$ and $(1+b)^{1-2\alpha/3} \approx b^{1-2\alpha/3} \gg 1$ so

$$N = a^2 b^{1-\alpha}/(1-\alpha)$$

$$a = b^{\frac{-1+\alpha}{2}}\sqrt{(1-\alpha)N}$$

$$W = (1-\alpha)Nb^{\frac{\alpha}{3}-1} + 4\sqrt{(1-\alpha)N}\,b^{\frac{1}{2}-\frac{\alpha}{6}}$$

$$W = (1-\alpha)Nc^2 + 4\sqrt{(1-\alpha)N}c^{-1} \text{ where } c = b^{\frac{\alpha}{6}-\frac{1}{2}}$$

and one can compute

$$\frac{\partial W}{\partial c} = 0 \Rightarrow W \propto N^{2/3}$$

and $1 - 1/d = 2/3$ (from Equation 5) which implies $d = 3$; that is, the bottleneck dimension is 3.

Case $\alpha > 3/2$: $(1+b)^{1-\alpha} \approx b^{1-\alpha} \ll 1$ and $(1+b)^{1-2\alpha/3} \approx b^{1-2\alpha/3} \ll 1$

$$N = \frac{a^2}{\alpha - 1} \text{ (does not determine b)}$$

$$W = a^2 b^{-2\alpha/3}\Theta(K-b) + \frac{4a}{2\alpha/3 - 1}$$

$$W = (\alpha-1)Nb^{-2\alpha/3}\Theta(K-b) + \frac{4\sqrt{(\alpha-1)N}}{2\alpha/3-1}$$

b is free so we may set $b = K$ to get rid of the $O(N)$ term.

$$W = O(N^{1/2})$$

which implies

$$d = 2. \tag{A4}$$

Case $1 < \alpha < 3/2$: We can ignore $(1+b)^{1-\alpha}$ but not $(1+b)^{1-2\alpha/3} \approx b^{1-2\alpha/3}$.

$$N = \frac{a^2}{\alpha-1} \Rightarrow a = \sqrt{(\alpha-1)N}$$

$$W = a^2 b^{-2\alpha/3}\Theta(K-b) + \frac{4ab^{1-2\alpha/3}}{1-2\alpha/3}$$

Here setting $b = K$ would result in a disastrously large second term in W, so

$$W = (\alpha-1)Nb^{-2\alpha/3} + 4\frac{\sqrt{(\alpha-1)N}}{1-2\alpha/3}b^{1-2\alpha/3}$$

and minimizing with respect to b results in $W = O(N^{1-\alpha/3})$ and

$$d = \frac{3}{\alpha} \tag{A5}$$

which smoothly interpolates between $d = 3$ and $d = 2$. Thus, we have Equation (7a).

There is a second, embedded dimension which is easier to compute and depends on the wire lengths as well as their number. (The bottleneck dimension does not depend on wire length, though our approximate calculation of it used such information due to the replacement of a general subnet with a prism.)

Divide x, y, z into cubes of side $\frac{1}{K}$, and ask how many cubes are occupied by 0-dimensional neurons or 1-dimensional wires. Where $\rho > K^3$, all cubes are occupied by

at least one neuron; elsewhere neurons are sparse and a fraction ρK^{-3} of the cubes have neurons. Also, in the sparse neuron region the number of cubes containing wires is (total wire length)/(cell side). The total number of occupied cells is approximately

$$C = K^3 \int_{\frac{1}{K}}^{\frac{2}{K}} dz + \sum_{x} \sum_{y} \sum_{z=\frac{1}{K}}^{1} \rho(z) K^{-3} +$$

$$\frac{1}{K^{-1}} \int_{\frac{2}{K}}^{1} dz \rho(z) \langle \text{fanout} \rangle \langle \text{wirelength between nearby neurons} \rangle$$

$$C = K^3 \int_{\frac{1}{K}}^{\frac{2}{K}} dz + \int_0^1 dx \int_0^1 dy \int_{\frac{2}{K}}^1 \rho(z) dz + \frac{F}{\frac{1}{K}} \int_{\frac{2}{K}}^1 dz \rho(z) \rho^{-1/3}(z) \qquad (A6)$$

$$C = K^2 + \frac{K^{3-\alpha} - K^2}{1-\alpha} + F \frac{K^{3-2\alpha/3} - K^2 2^{1-2\alpha/3}}{1-2\alpha/3}$$

so

$$d = \lim_{K \to \infty} \log_K C$$

(by analogy with integral dimension local networks) implies

$$d = \max(2, 3-\alpha, 3-2\alpha/3).$$

Thus, we have Equation (7b).

In practice we are interested in controlling total wire + neuron volume when the network has been embedded in a 2- or 3- dimensional medium, so this dimension is probably more relevant for measuring costs than the intrinsic one. The intrinsic bottleneck dimension, however, is the proper one for avoiding bottlenecks, i.e., for ensuring network functionality. Both of these dimensions may be too high since a neural network can be first mapped into a network some of whose nodes are neurons and some of whose nodes are linear summation nodes as in Fig. 11. This mapping results in wire sharing and possibly in a lower dimension. The best way to insert such summation nodes may depend on the embedding medium.

III A One-Dimensional Scale-Invariant Network

The stripe-generating network of Chapter II was intriguing and had interesting scaling behavior but did not solve the problem of generating realistic fingerprint patterns at all scales. An acceptable network was hard to find, possibly indicating that the assumed synapse function was of the wrong form. There is in principal no reason to assume any particular form for the synapse function, aside from a locality constraint. Within a neighborhood of nonzero connections one could adjust each synaptic weight independently, trying all the while to optimize some measure of network performance and cost. Starting from this point of view, we will develop a new and more effective form for the synapse function and test it.

The first problem with this approach is that there are many synaptic weights T_{ij}, many more independently controllable quantities than would seem necessary to get the simple behaviors desired. One would like to impose rotation, translation and scale invariance on T_{ij}, for example, by adopting a special form such as

$$T_{ij} = T'(x_i - x_j, y_i - y_j, z_i/z_j, \sigma_{c_j}^{c_i}, \phi_i - \phi_j),$$

where σ_b^a is defined following Equation (II.11). Then locality implies that each of T''s arguments is zero outside of a region whose size is independent of the system size. Of course, since the grid of $\vec{x}$ positions is irregular we will need T' to be defined on a much finer grid; we will in fact define it as a continuous function. To insure that there are enough independent T' variables to control the network's fixed points in detail, we will start out with a neural $\vec{x}$ grid which has a great many points per minimal stripe width. An additional effect of the small $\vec{x}$ grain size is to eliminate the percolation problems that result from a large variance in the number of grid points per stripe. Thus, in the continuous limit the network is very homogeneous, and we hope to get close to that limit with a minimal number of neurons by optimizing the synaptic weights.

So the ranges of T''s arguments are limited by the locality of T, but each argument of T' and of $s(\vec{x})$ has a very small grain size. To make up for the extra computation that simulating such a fine-grained network requires, we will examine networks which are only a few stripe-widths wide, using periodic boundary conditions. In short, we will look at T microscopically. Eventually we will guess a special form for T' with many fewer independent parameters but the same small grid spacing, making the network optimization tractable.

The next logical question is, what behavior to optimize? We will measure the stability of various desired fixed points by computing the distance between the desired fixed point and the actual fixed point which the network lands on when it starts from the desired fixed-point configuration. This distance is to be minimized for stripe configurations. Also, we will examine some special configurations which are to be destabilized, and measure the distance between the actual fixed point reached from such a configuration and a desired stripe configuration which "should have" been reached instead.

Finally, various severe limitations will be imposed on this network to get at the important scaling questions with a minimum of computation. We will revert to $1+\epsilon$-dimensional networks, rather than $2+\epsilon$; our "images" will be 1-dimensional. This corresponds to taking a vertical slice through the $2+\epsilon$-dimensional network in a plane transverse to the prevailing stripe direction. We will keep only color, not angular internal indices. As previously mentioned, we will examine network configurations containing only a few stripes, using periodic boundary conditions. In return for accepting these limitations, we will get a much clearer view of what is going on in a local patch of our network.

III.1. Stripe Stability Equation

Since the vertical slice network no longer has an internal angle index, each "clump" of competing neurons is just an antagonistic pair responding to opposite colors: a black-detecting and a white-detecting neuron. In anticipation of the day when an angle index should return, and in consideration of the arguments in Chapter II's Appendix I, each neuron takes the values 0 or 1. The desired fixed-point configurations involve the $(s, \bar{s})$ pairs alternating between (1,0) and (0,1) below some $z = w$ =stripe width, and $(s, \bar{s})$ above that z, as in Figure 1.

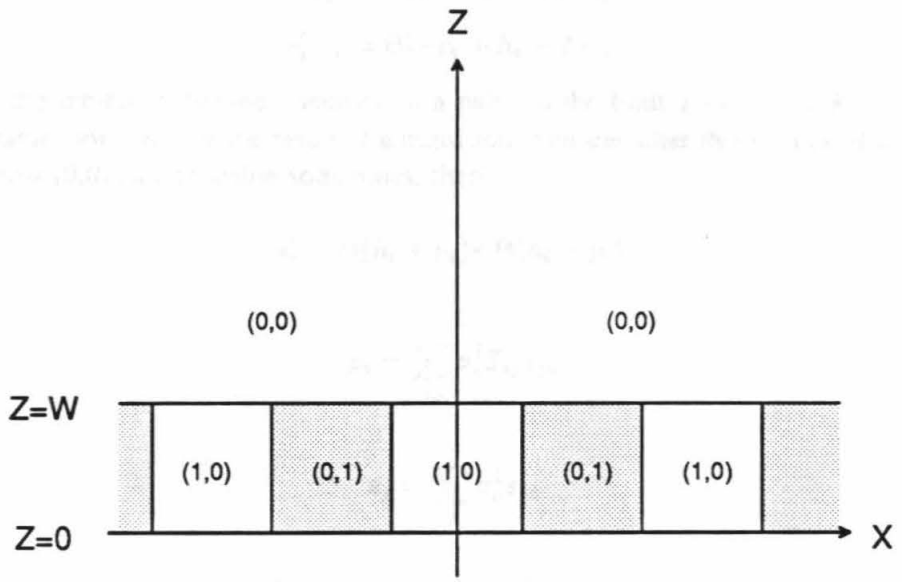

Figure 1

This amounts to the introduction of 3-state neurons with values ± 1 and 0. If the network is designed so that

$$T_{ic,jc'} = \sigma^c_{c'} T_{i,j}$$

(in this equation the indices $\vec{x}$ and c are packaged up into i) and

$$h_{ic} = h_{ic'} = h_i,$$

then the neural update rule is

$$s'_{ic} = \Theta\Big[\sum_{jc'} T_{ic,jc'} s_{jc'} + h_{ic}\Big]$$

where

$$\Theta(x)=\begin{cases}+1 & \text{if } x \geq 0;\\ -1 & \text{otherwise.}\end{cases}$$

This rule is the version of Equation (I.1) appropriate for 0/1 neurons and is equivalent to

$$s_i = s_{i,c=1} - s_{i,c=-1} = \pm 1 \text{ or } 0$$

and

$$s'_{ic} = \Theta\Big[\sum_j T_{ij}\sigma^c_{c_j} s_{jc} + h_i\Big].$$

Introducing the "input to a clump"

$$p_i = \sum_{jc} \sigma^1_c T_{ij} s_{jc}$$

the update rule is

$$s'_{i,1} = \Theta[p_i + h_i - I s_{i,-1}]$$

$$s'_{i,-1} = \Theta[-p_i + h_i - I s_{i,1}]$$

where I = the inhibition between neurons in a pair. In the limit $I \to \infty$, $(s, \bar{s}) = (1, 1)$ can never be stable, nor can it be the result of a transition from the other three states; if in addition $h_i < 0$ so that (0,0) can be stable sometimes, then

$$s'_i = \Theta(h_i + p_i) - \Theta(h_i - p_i).$$

But

$$p_i = \sum_{jc} \sigma^1_c T_{ij} s_{jc}$$

and

$$s_j = \sum_c \sigma^1_c s_{jc}$$

imply

$$s'_i = \Theta\Big[h_i + \sum_j T_{ij} s_j\Big] - \Theta\Big[h_i - \sum_j T_{ij} s_j\Big]. \tag{1}$$

Thus the neural update rule for 3-state neurons can be written entirely in terms of the states of the other 3-state neurons in the network, without having to refer to the states of the pairs of mutually inhibiting 0/1 neurons which make up the composite neurons.

We will have occasion to determine a common $h = h_i$ by minimizing the number, D, of unstable neurons in a given configuration s_i, e.g., the stripe configuration in Figure 1.

$$D(h) = \sum_{i|s_i=0} \Theta(|p_i| + h) + \sum_{i|s_i=1} \Theta(-p_i - h) + \sum_{i|s_i=-1} \Theta(p_i - h)$$

$$= \sum_{i|s_i=0} \Theta(|p_i| + h) + \sum_i |s_i| + \sum_{i|s_i \neq 0} \Theta(h + p_i s_i)$$

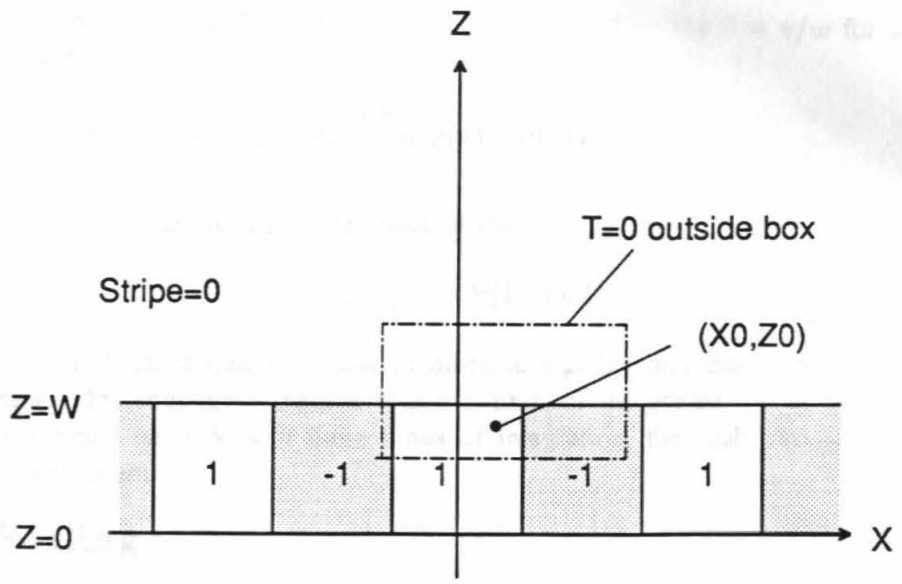

Figure 2

(since $\Theta(-x)= 1 - \Theta(x)$)

$$D(h)= \text{const} + \sum_i (1 - 2|s_i|)\Theta(h - \pi_i)$$

where

$$\pi_i = \begin{cases} -|p_i| & \text{if } s_i = 0, \\ -p_i s_i & \text{otherwise.} \end{cases}$$

Permute the indices so that the π_i occur in increasing order. Then if $D_0 = 0$ and

$$D_k = \sum_{i=1}^{k} (1 - 2|s_i|),$$

the smallest D_k indicates

$$\pi_k < h < \pi_k - 1. \tag{2}$$

This determines h.

With $\pm 1, 0$ neurons as above, and a scale- and translation-invariant synaptic function

$$T(x - x_0, z/z_0),$$

pure stripe configuration are given by

$$\text{Stripe}(x, z, w)= \begin{cases} \sum_{j=-\infty}^{j=\infty} (-1)^j \text{Bit}(x/w, j - \frac{1}{2}, j + \frac{1}{2}), & \text{if } z \le w; \\ 0, & \text{otherwise,} \end{cases}$$

(as in Figure 2) and the input p_i to a clump (a 3-state neuron) in a pure stripe configuration is

$$P(x_0, z_0, w)= \int\int dx dz \rho(z)\ \text{Stripe}(x, z, w) T(\frac{x - x_0}{z_0}, \frac{z}{z_0})$$

as illustrated in Figure 2 for a T which is 0 outside a box in x, z and for a coordinate system centered on the stripe of width w.

Since Stripe(x, z, w)= Stripe$(x/w, z/w, 1)$, and introducing $\hat{v} = v/w$ for all spatial variables v, we have

$$P(x_0, z_0, w) = \int_{-\infty}^{\infty} dx \int_0^{\infty} dz \rho(z) \text{ Stripe}(\hat{x}, \hat{z}, 1) T(\frac{\hat{x} - \hat{x}_0}{\hat{z}_0}, \frac{\hat{z}}{\hat{z}_0}) \tag{3}$$

so that for $\rho(z) = z^{-2}$ the result is scale-invariant and

$$P(x_0, z_0, w) = P(\hat{x}_0, \hat{z}_0, 1).$$

Thus, if $\rho(z) = z^{-2}$, all stripes are stable or unstable together and this network behavior is scale-invariant. This conclusion requires that the limits of integration are as given, and not finite. In the actual network with finite limits of integration, the scale invariance must be checked by experiment.

III.2. Scaling

In Chapter II, $\rho(z)$ was $z^{-\alpha}$ with $\alpha = 2.1$, rather than the tree-like $\alpha = 3$, which is the solution adopted here (but for a one-dimensional image, so $\alpha = 2$). A lower α had the advantage of placing more neurons near the top of the hierarchy, where the network is most vulnerable to defects such as missing neurons. Why are we removing this desirable feature?

First, notice the effect of $\alpha \neq 2$ in Equation (3):

$$P(x_0, z_0, w) = w^{2-\alpha} P(\hat{x}_0, \hat{z}_0, 1).$$

This $P(x_0, z_0, w)$ is then to be compared with $h(x_0, z_0)$ in Equation [update rule] to determine the new configuration and how much it differs from Stripe$(\hat{x}, \hat{z}, 1)$. Then translation invariance requires $h(x_0, z_0) = h(z_0)$, and behavior scale invariance requires that h scale with P, i.e., $h(z_0) = w^{2-\alpha} h(\hat{z}_0)$, which implies $h(z_0) = h(1) z_0^{2-\alpha}$. A varying threshold, however, is equivalent to a constant threshold together with an inversely varying T; the substitution

$$h(z_0) \rightarrow h(z_0) z_0^{\alpha-2} = h(1)$$

$$T(\vec{x}_0, \vec{x}) \rightarrow z_0^{\alpha-2} T(\vec{x}_0, \vec{x})$$

changes no behavior. Under this substitution the form of T_{ij} is changed, but not so severely as one might believe:

$$z_0^{\alpha-2} T(\frac{\hat{x} - \hat{x}_0}{\hat{z}_0}, \frac{\hat{z}}{\hat{z}_0}) = z^{\alpha-2} T'(\frac{\hat{x} - \hat{x}_0}{\hat{z}_0}, \frac{\hat{z}}{\hat{z}_0})$$

so

$$\rho(z) T(\vec{x}_0, \vec{x}) = z^{-2} T'(\vec{x}_0, \vec{x})$$

and the general scale-invariant network is behaviorally equivalent to one with $h(z)$= constant, $\alpha = 2$, and $T(\vec{x}_0, \vec{x})$ of the form $T((\hat{x} - \hat{x}_0)/\hat{z}_0, \hat{z}/\hat{z}_0)$.

The use of the special form h =constant, $\alpha = 2$ is further encouraged by the fact that contour plots of $P(\hat{x}_0, \hat{z}_0, 1)$ are especially meaningful: contours of height h separate regions of $s' = \pm 1$ form $s' = 0$. One can choose the contour (and hence the h) which makes the separated regions look most like Figure 1. We have already described an automatic procedure for choosing h, but it is nice to be able to do it visually as well.

It is possible to get the higher redundancy that would result from choosing an $\alpha < 2$ by letting $T(\vec{x}_0, \vec{x})$ shrink with increasing $\vec{x}$, since

$$\rho(z)T(\vec{x}_0, \vec{x}) = z^{-2}T(\frac{\hat{x} - \hat{x}_0}{\hat{z}_0}, \frac{\hat{z}}{\hat{z}_0}) = z^{-\alpha}\Big[z^{\alpha-2}T(\frac{\hat{x} - \hat{x}_0}{\hat{z}_0}, \frac{\hat{z}}{\hat{z}_0})\Big] = \bar{\rho}(z)\bar{T}(\vec{x}_0, \vec{x}).$$

The penalty one pays is a larger dynamic range for possible T values: very small synaptic weights must be accurately built into the circuit implementing the network. To what extent this is allowed is a technological question. Such a tradeoff is equivalent to the replacement of every neuron by some multiplicity $m(z)$ of neurons, each receiving the same input as the original and putting out $1/m(z)$ of the original neuron's output strength. The model neurons are connected to each other with nearly uniform strength. This redundancy strategy is the most simple-minded, and much better ones are possible. They involve extra cell types to carry "parity bits," by analogy with well-known error-correcting codes. The simple redundant network is very similar to the tree-of-meshes network sometimes discussed in connection with fault-tolerance in Wafer Scale Integration [Leighton and Leiserson 85].

One major difference between this type of $\alpha < 2$ network and that used in Chapter II is in fanout: the number of neighbors to one neuron grows with z in this network but not in the older network of Chapter II. In this subtle way, the older network is not scale-invariant. This is most evident in the $\alpha = 0$ limit, where the hierarchy axis becomes a full extra dimension and there is translation invariance in the z direction but no scale invariance.

III.3. Guess for T

Now we need $T((\hat{x} - \hat{x}_0)/\hat{z}_0, \hat{z}/\hat{z}_0)$. It is tempting to consider all the desired fixed-point stripe configurations s_i^m and solve the linear fixed-point Equations

$$s_i^m = \sum_j T_{ij} s_j^m. \tag{4}$$

The solution can be translation-invariant because the set of allowed s_i^m is translation-invariant, and it can be made scale-invariant by adopting a scale-invariant measure on the stripe widths or, in the case of finitely many configurations, by sampling the set of allowed configurations uniformly in $\log w$ instead of w.

There are a number of objections to this approach, however. The solution to the linear algebra problem will not in general be a local T_{ij}. It will be more constrained than it needs to be, since all we need is that

$$s_i^m = g\Big[\sum_j T_{ij} s_j^m\Big]$$

where g is the nonlinear transfer function and has range $\{0, \pm 1\}$. In addition, we are using the stabilization of stripe patterns as a stand-in for a more complicated set of behavior requirements. We actually want both more complicated fixed points (such as stripes with slowly varying width) and some control over basins of attraction (e.g., ambiguous stripe width configurations evolve towards the larger stripe width). For these purposes we require only an approximate solution to (4), preferably local and simply parameterized for use in satisfying further behavior constraints. In short, we need a cartoon solution to Equation (4).

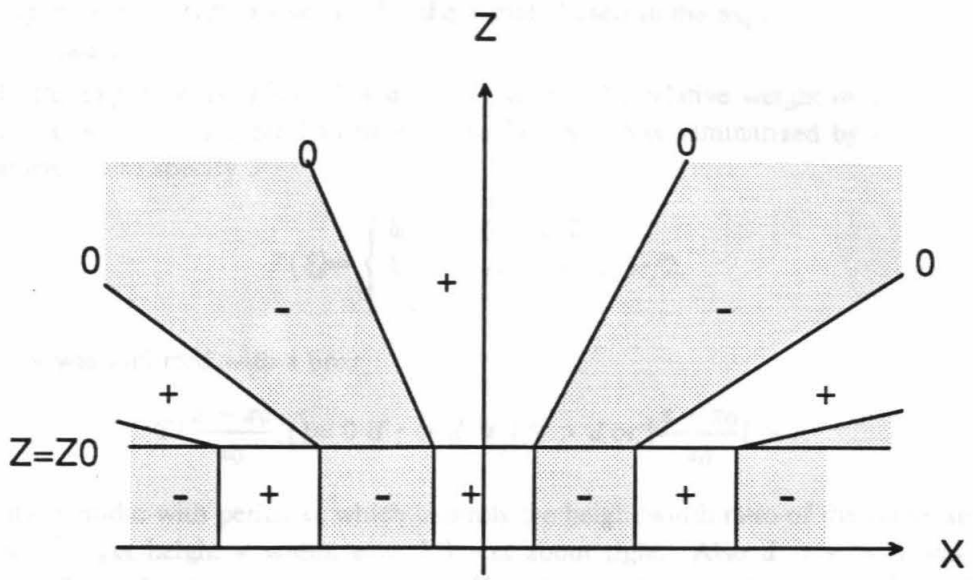

Figure 3

Such a solution is suggested by the sum of outer products of the stable configurations,

$$T(\vec{x}_0, \vec{x}) = \sum_m s^m(\vec{x}_0) s^m(\vec{x}).$$

Since the stripe configurations are indexed by their size w and their offsets in the x direction, this expression for T equals

$$\int_{-1/2}^{1/2} d\phi \int_0^\infty \frac{dw}{w} \text{ Stripe}(x_0 - \phi w, z_0, \text{width} = w = \text{height}) \text{ Stripe}(x - \phi w, z, w).$$

Let $z_+ = \max(z, z_0)$ and $z_- = \min(z, z_0)$, and let x_+ and x_- be the corresponding x values. Then because of Stripe's periodicity,

$$T(\vec{x}_0, \vec{x}) = \int_{-1/2}^{1/2} d\phi \int_{z_+}^\infty \frac{dw}{w} \text{ Stripe}(-\phi w, z_-, w) \text{ Stripe}(x_+ - x_- - \phi w, z_+, w)$$

$$= \int_{-1/2}^{1/2} d\phi \int_1^\infty \frac{d\omega}{\omega} \text{ Stripe}(\frac{x_+ - x_-}{z_+} - \phi\omega) \text{ Stripe}(-\phi\omega)$$

$$= F(\frac{|x_+ - x_-|}{z_+})$$

after the divergent integral has been cut off. Here F is a complicated function, but it has only one argument. So

$$T(\vec{x}_0, \vec{x}) = F(\frac{|x - x_0|}{\max(z, z_0)}).$$

When we allow for an unknown varying $h(z)$ to be absorbed into T (by the arguments which lead to Equation (4)), making $h(z)$ constant by introducing the compensating function f, we get

$$T(\frac{\hat{x} - \hat{x}_0}{\hat{z}_0}, \frac{\hat{z}}{\hat{z}_0}) = F(\frac{|x_+ - x_-|}{z_+}) f(z/z_0) \tag{5}$$

whose chief feature is its kinked $T = 0$ contours, as in Figure 3.

Figure 3 is the cartoon we sought; the actual T used in the experiments agrees with it and with Equation (5).

In the experiments, $f(\varsigma)= 1 + a(\varsigma - 1)$ so that the relative weight of top-down and bottom-up connections is controlled by a. Also Figure 3 was summarized by introducing a few parameters $\vec{b}$ to specify F:

$$F(\xi)= \begin{cases} b_0 & \text{if } |\xi| < c/2, \\ b_1 & \text{if } c/2 < |\xi| < 3c/2, \\ \ldots & . \end{cases}$$

and locality was enforced with a box:

$$T(\frac{x - x_0}{z_0}, \varsigma)= 0 \text{ if } \varsigma > d \text{ or } 1/\varsigma > d \text{ or } |\frac{x - x_0}{z_0}| > e.$$

F is nearly periodic with period c, which controls the height/width ratio of the stable stripes. Empirically, to get height = width, $c = 1.3$ was about right. Also $d = e = 2$ was used throughout. Two other important parameters affect the distribution of neurons, $\rho(z)= Cz^{-2}$: C determines the base of the exponential used to distribute neurons, and was controlled through the ratio $\Delta z/z$, a constant. Here Δz is the spacing between adjacent neuron levels at height z. This "relative resolution" (abbreviated as "relres" in the figure captions) was 7 for most of the experiments reported below, although 5 also worked. 3 did not work and may require more adjustable parameters than we have used here. For a binary tree (base 2 exponential) $\Delta z/z \approx 1$. The minimal z is also important and was generally 0.15 . z_{max} is much less important and was 2.5 unless otherwise specified. These values are appropriate for examining stripes of width .5 - 1.

The guess in Equation (5) for how to mix color and scale indices in T_{ij} probably applies also to the mixture of angles and relative heights, and for much the same reasons. Both forms differ substantially in their z-dependence from the color and angle interactions assumed in Chapter II. As we shall see, in the case of color at least the results are much better for the new network.

III.4. Results

For the previously quoted values of $d = 2, e = 2$, only b_0 and b_1 affect regions of the $T(\xi,\varsigma)$ function inside the $T \neq 0$ box. The simplest experiment tried was to let $a = 0$, $b_0 = 2$, $b_1 = 1$ and start the network in a pure stripe configuration of unit width such as that shown in Figure 4a. Here the x-axis runs from -3 to 3 and the z axis runs from 0 to 3, as will be the case in other configuration displays unless noted otherwise. The constant threshold h is set by the algorithm given in Section III.1, whose purpose is to maximize the stability of the given configuration. The Figure also shows the distribution of neurons in (x, z) space. The three neural values -1,0,1 are shown as -, dot and + respectively. The fixed point reached after 4 update sweeps is shown in Figure 4b; it is a distance .0641 from the initial configuration, meaning that that fraction of neurons has different values in the two configurations.

The untruncated total input to a neuron $P(x, z)$ is more informative and continuous. It is shown in Figure 5 for the configuration of 4b. The contours $P = \pm h$ are candidates for separating the possible $s = -1, 0, 1$ values from each other in the truncation phase of the update rule, Equation (1). The $P = 0$ contour would clearly be unacceptable as a dividing

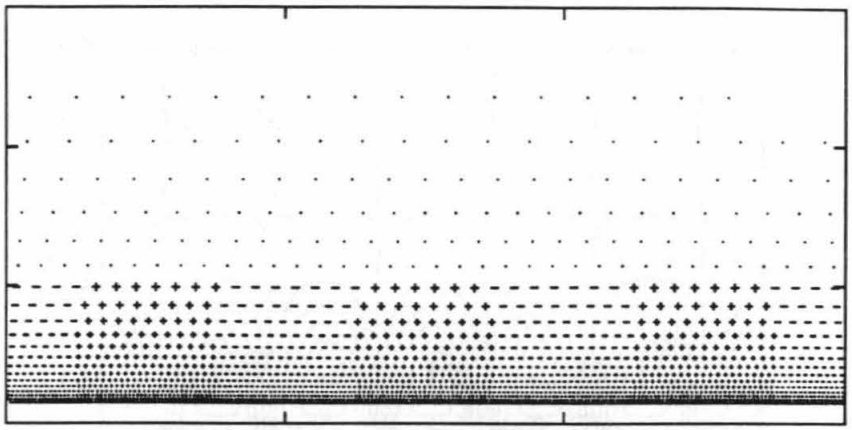

Figure 4a Input

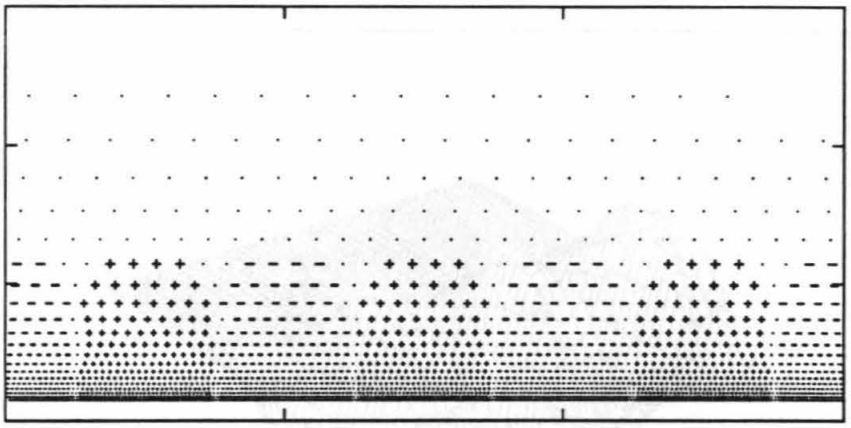

Figure 4b Output

Figure 4: Input and output

line for $s = \pm 1$ neurons, showing that we need $h < 0$. The next higher contour shown, $P = 100$, looks good as such a dividing line since it allows $z > 1$ to fall in between $\pm h$, thereby producing $s = 0$ in that region. The actual h used was 42. The most awkward feature of Figure 5 is the rapid decrease of P near $z = z_{min}$ since this prevents the use of larger h values, which would decrease the network's sensitivity to accidental bumps in P for $z > 1$ in more complicated configurations.

The $T(\xi, \varsigma)$ function used in these experiments is shown in Figure 6, and is a more detailed version of Figure 3. $T = 0$ outside the area plotted. Note the artificially bumpy contours produced by the contour-plotting package [S] and the distribution of sample points, which is that used in the network computation.

Figure 7, the first evidence of scale-invariant behavior for the present network, shows a stable configuration with width=.5 . The constant h was changed from -89 (as will be explained momentarily) to -103 to optimize the stability of this configuration, but both width

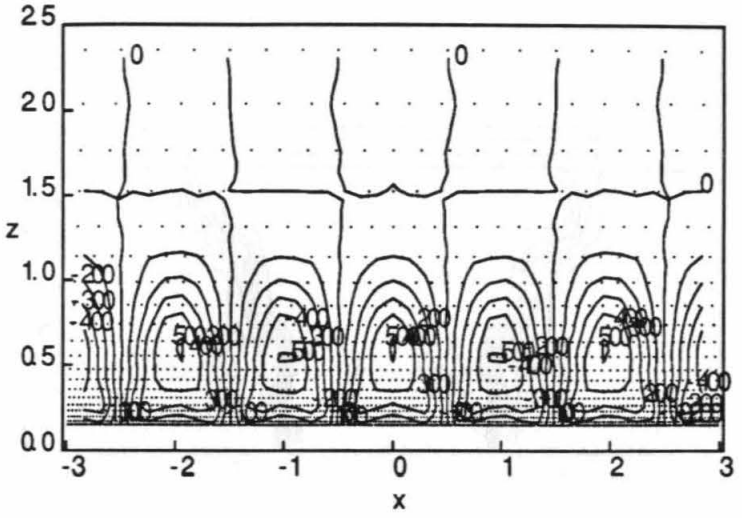

Figure 5.a Contour plot

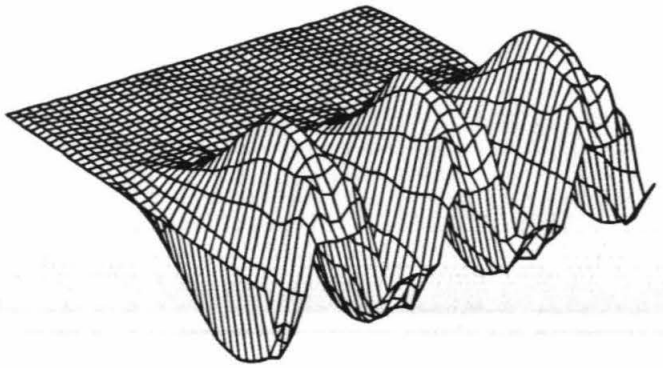

Figure 5.b Perspective plot

Figure 5 Function $P(x_0, z_0)$, the total input to a neuron

= 1 and width = .5 stripes are stable at the intermediate $h = -96$. The intermediate value is used in evolving future configurations to give both small and large stripes a reasonably equal chance at stability, but it may be that the difference between the two cases is due to the relatively large $z_{min} = .15$ and would decrease if z_{min} were decreased.

$P(x, z)$, being an integral or at least well approximated by one, is a continuous function and it must pass through zero between stripes of opposite sign. Thus in all the stable configurations we see a gap between the regions of $s = 1$ and $s = -1$, even when the initial configuration had no such gap. It is significant that, although the first update sweep introduces

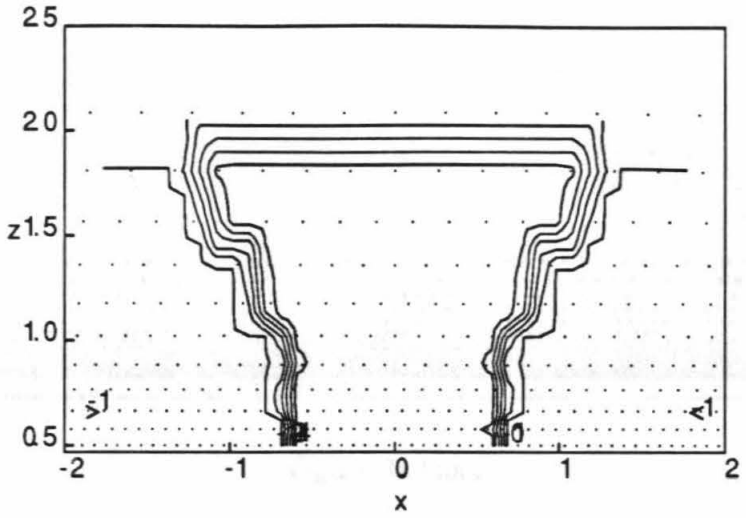

Figure 6 $T(\xi, \varsigma)$

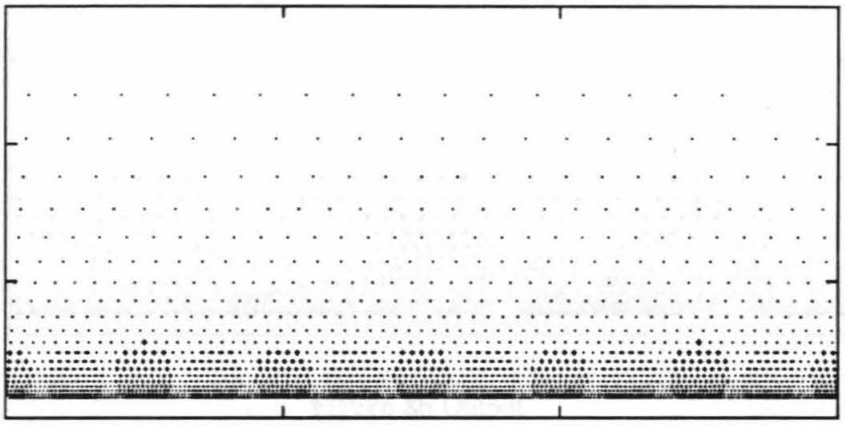

Figure 7: Also stable

a gap, subsequent sweeps do not widen it much. If it kept on widening the network could converge to the stable state having all neurons turned off, or to some other distant and inappropriate configuration. Introducing such a gap into the initial configuration as is done in Figure 8 reduces the total distance between initial and final configurations, making that distance a more sensitive measurement of the quality of network behavior (small distances are good). The ratio of gap width to stripe width was coarsely optimized for every setting of the network parameters, being usually about .1 . For the optimum gap of the $(b_0, a) = (2, 0)$ network, the optimal h is -89 rather than -42, and the distance travelled for a pure stripe configuration is reduced from .0641 to .0238. All later stripy initial configurations use the same value of gap width/stripe width.

It is also of importance that the stable stripes have a constant ratio of height to width,

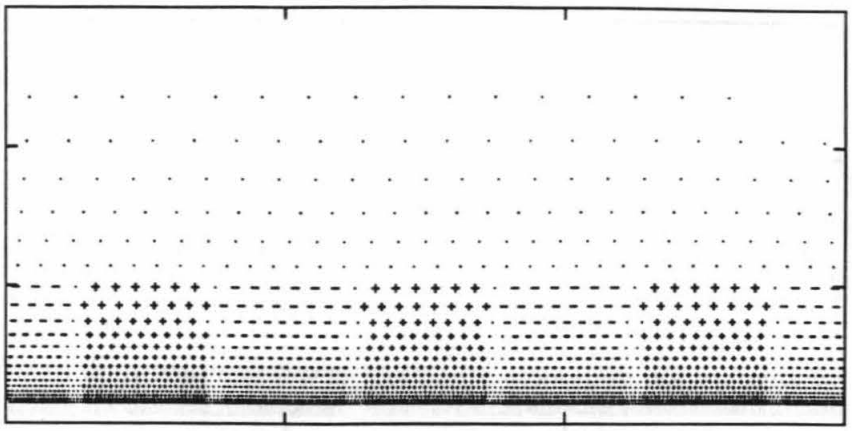

Figure 8a Input

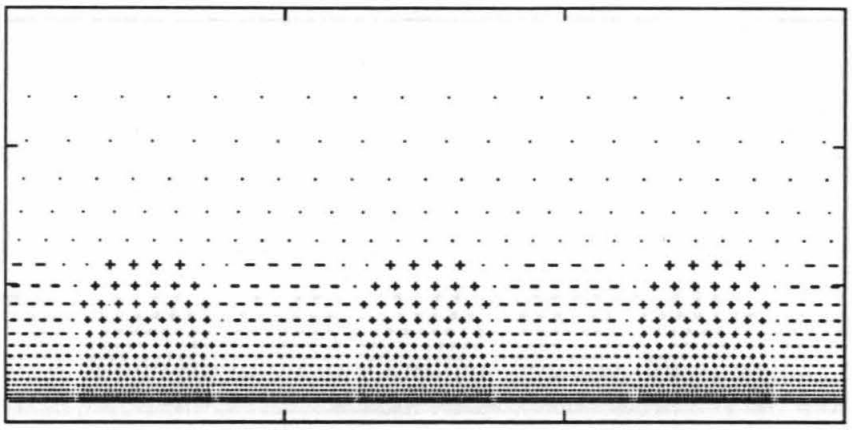

Figure 8b Output

Figure 8: Input and output

so that detecting a stripe of width w can be reduced to a local operation at $z \approx w$. It is possible for this condition to be violated, for example if both stripe boundaries and stripe interiors were locally stable so that the boundaries could be separated by great lengths. Then the initial condition of Figure 9a would be stable; in fact the short stripes grow up to become properly shaped stripes. Similarly in Figure 10 overly tall stripes are cut down to size. Figure 9 suggests an image input mechanism: A thin layer (e.g., $\Delta z = z_{min}$) of neurons at the bottom of the hierarchy are set to ± 1 depending on the color of the image at the corresponding x, and the stripes are allowed to grow up to their natural height.

The first "bug" in the $(b_0, a) = (2, 0)$ network shows up in Figure 11, where the input configuration has ambiguous stripe width and is unfortunately nearly stable. The parameter a was first introduced in order to more heavily weight the top-down connections, allowing the

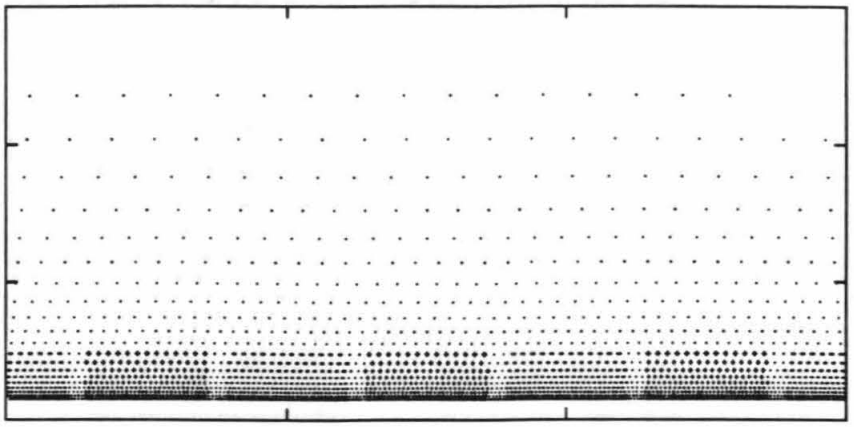

Figure 9a Input

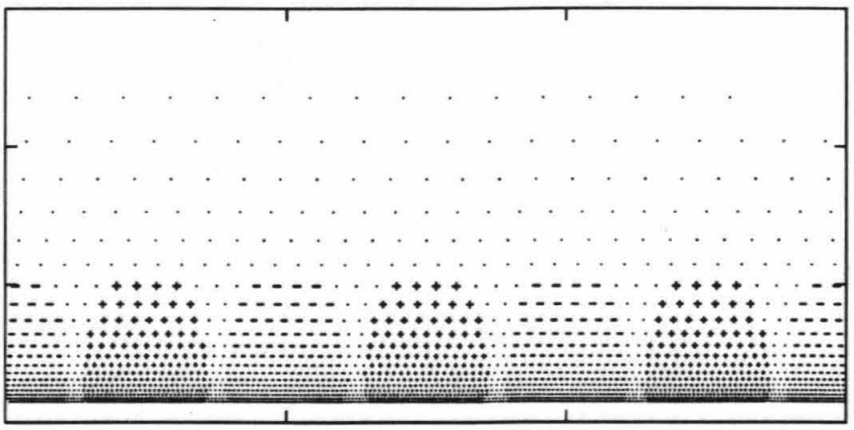

Figure 9b Output

Figure 9: Input and output

larger of the two stripe widths to win. This ambiguity bug occurs as the fixed point reached from a random starting configuration, as seen in Figure 12. It also occurs in the more important case of a "hole" in a stripe making three smaller stripes, in analogy to the sweat pores in a fingerprint. This situation and its unfortunate result are shown in Figure 13. The starting point in Figure 13a is a pair of equal and opposite width discontinuities very close together. A single isolated width discontinuity is also troublesome but less important, not having an analog in typical fingerprint images.

The $T(\xi, \varsigma)$ function obtained by optimizing the new parameter a, so that for networks with relative resolution $\Delta z/z = 7$ the T parameters are $(b_0, b_1, a) = (2, -1, 1)$, is displayed in Figure 14. In direct correspondence with Figure 5, Figure 15 shows contour and perspective plots of the total input $P(x, z)$ to a neuron at (x, z) in a stable stripe configuration. There are

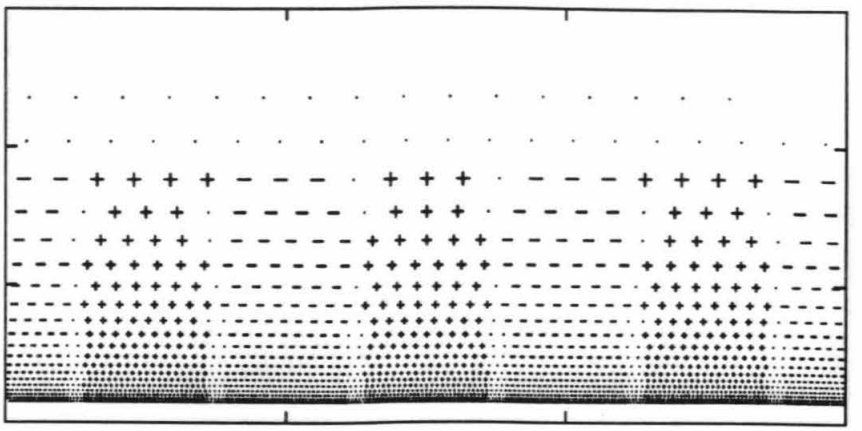

Figure 10a Input

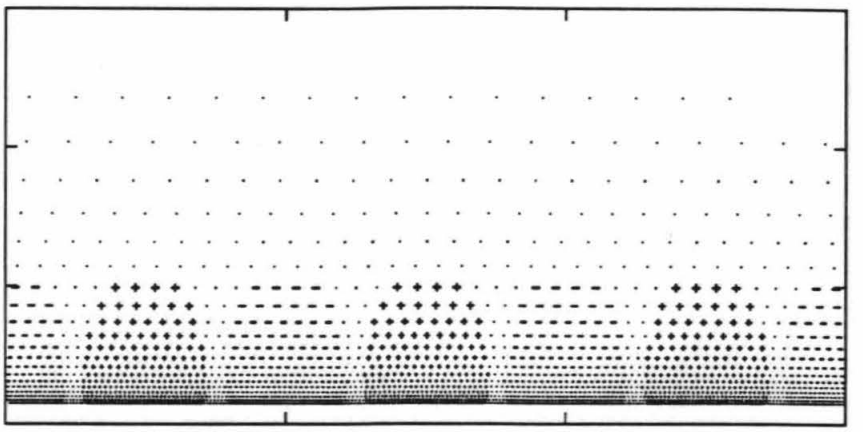

Figure 10b Output

Figure 10: Input and output

only minor differences between the P functions for the two networks.

The fixed point reached from a configuration of ambiguous stripe width is shown in Figure 16. The improvement over Figure 11 is very large. Similar results are obtained when the ratio of widths is three instead of two. Likewise, the discontinuity problem is cleared up, as shown in the fixed-point configuration of Figure 17 (c.f. Figure 13). If there are five instead of three small stripes substituted for the big one, the network still suppresses them and converges to the desired fixed point.

One may quantify a network's performance for pure stripes and for the ambiguity and discontinuity test cases by measuring the distance between the actual and desired fixed points. For a pure stripe input, the desired output is the same as the input. For the ambiguous and discontinuous stripe width input cases it is desired that the largest reasonable stripe width

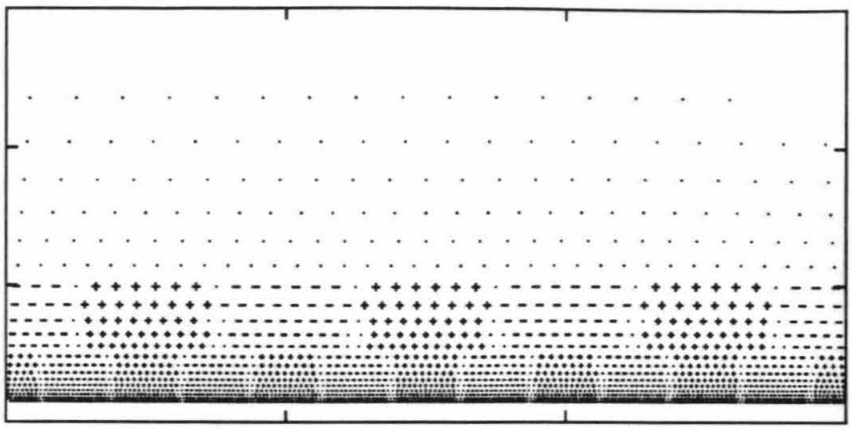

Figure 11a Input

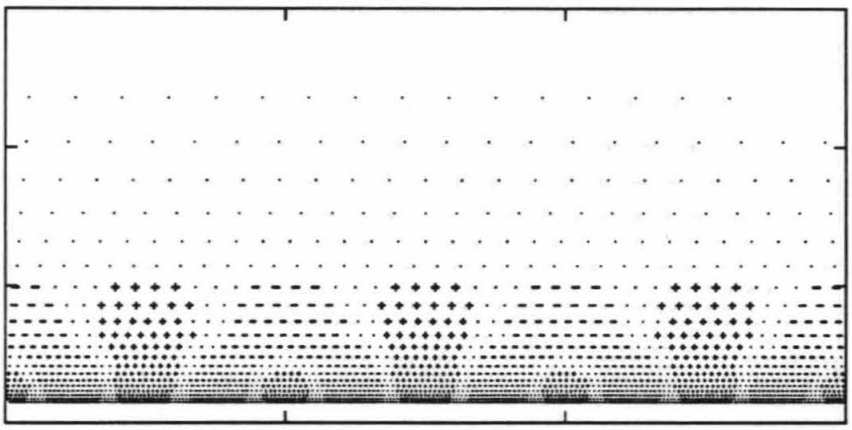

Figure 11b Output

Figure 11: Input and output - ambiguous stripe width

win, to suppress holes in ridges or valleys. (In the two-dimensional network, which size wins should depend on more complicated two-dimensional geometry.) The goal configuration is a pure stripe configuration with the larger stripe width.

In the contour plots of Figure 18, the axes are b_0 (horizontal) and a (vertical). The evolution was stopped after eight sweeps to save computing time. A few networks and configurations were evolved to their fixed points to roughly check the correlation between the measured distance after eight sweeps and the ultimate distance between goal and output configurations. The three test configurations have been tried out on the $\Delta z/z = 7$ networks used to produce all previous figures (18a,b,c), and also on $\Delta z/z = 5$ networks which use fewer neurons and less total wire length (18d,e,f). We would, of course, like to get the same behavior while spending less.

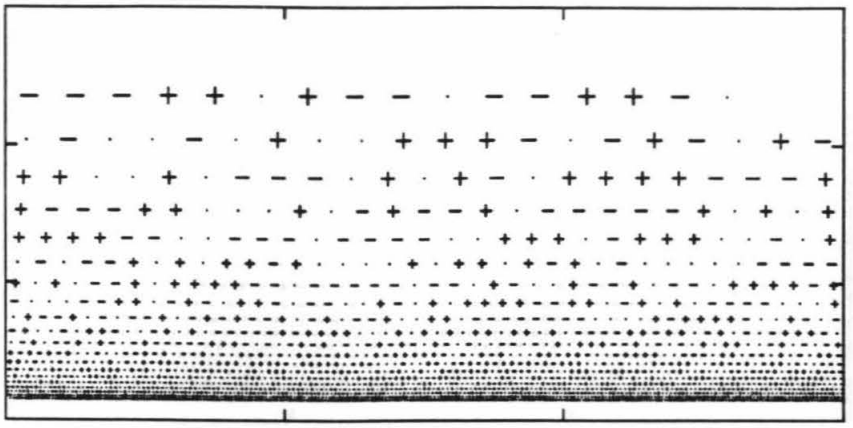

Figure 12a Input

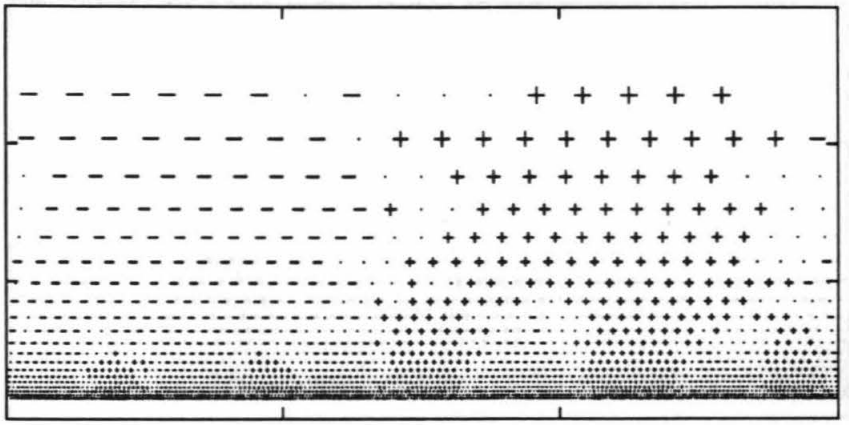

Figure 12b Output fixed point - 20 sweeps

Figure 11: Input and output

Figure 18a shows that $b_0 < 1.5$ does not support pure stripes. Figure 18b shows that $a \approx 1.5$ is favored, regardless of b_0, for removing ambiguity. By looking at the fixed-point configurations one may check that the .3 distance contour is acceptable, but no higher ones are. Figure 18c provides a more complicated constraint favoring $1.5 < b_0 < 3.5$ and $a \leq 1$. The acceptable distances are actually .1 or less, more stringent than for Figure 18b. $(b_0, a) = (2, 1)$ works well, as seen in Figures 16 and 17.

The less expensive networks used in Figures 18d,e, and f have quite similar constraints. The ambiguity constraint is more difficult to satisfy, and a boundary at $a \approx 2.2$ has moved down to $a \approx 1.8$. $(b_0, a) = (2, 1.2)$ works well, satisfying all constraints in the long-time limit. Both for relative resolution equal to 7 and to 5, all tested configurations converged to a fixed point before about 40 update sweeps. In all of these plots, there is a boundary near $a = 2$

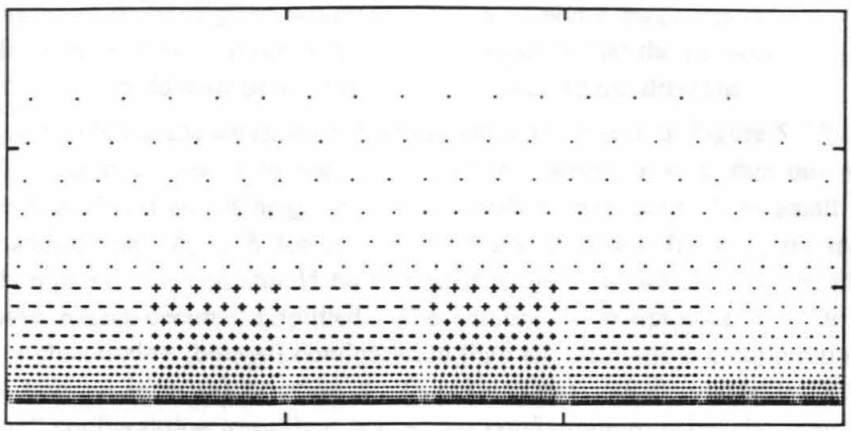

Figure 13a Input

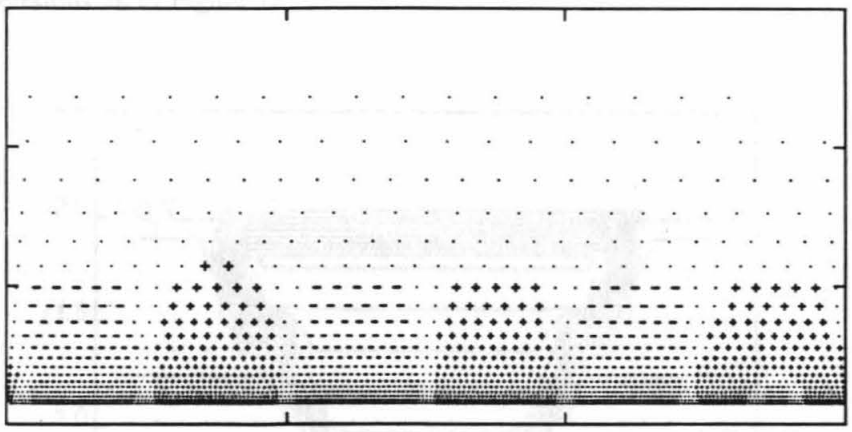

Figure 13b Output after 18 sweeps

Figure 12: Input and output - discontinuous stripe width

which may be related to the fact that for this value of a and for our choice of $z_{min} = .5$, some bottom-up connections are just becoming negative since $f(z_{min}) = 0$.

One further adventure with these networks should be described. There is a configuration similar to the "discontinuous" stripe width configuration dealt with earlier in that both are slices through patterns commonly occurring in fingerprints. The "discontinuous" configuration was a slice through a sweat pore; the new one is a slice through an image imperfection, such as a piece of dirt caught in a valley, which reverses the color of an entire stripe width. Thus, alternating stripes of width w are interrupted by three consecutive stripes of width w having the same color. Since this configuration arises from a "bridge" between neighboring ridges in two dimensions, it is referred to as a bridge configuration. For the networks with $\Delta z/z = 7$ and $(b_0, a) = (2,0)$ or $(2,1)$, the bridge configuration is unstable (as it should be) but evolves

to a wildly inappropriate fixed point which has stripes of width z_{max}, the largest stripes that could be stable in the network. What is desired, of course, is that the network simply flip the color of the offending middle stripe so that the colors once again alternate.

The clue that led to the solution of this problem may be seen in Figure 5. The variation in P within the region $z > w$ is so much less than that within $z < w$ that our method of choosing the cutoff P contour of height h is excessively conservative. Very small h's result from just demanding that $|P| < h$ for $z > w$; then any deviation from a pure input stripe will produce bumps in P which should be truncated to zero but which will actually exceed the small chosen h and become amplified. The solution is to optimize h to do what we want on a more challenging (bumpy) configuration than the pure stripe configuration. Figure 19a shows the bumpy $P(x_0, z_0)$ function for $(b_0, a)= (1, 1)$. So h was optimized to drive the new "bridge" configuration towards a pure stripe configuration, using the same threshold optimization algorithm as before. The resulting h was generally about $3/2$ the old value. Also (b_0, a) were reoptimized in the same way as before and found to satisfy both old and new constraints at $(1, 1.5)$. Thus, one set of parameter values for h, b_0 and a simultaneously satisfied all the imposed network constraints. All the constraints are plotted in Figure 20, with the same conventions as in Figure 18.

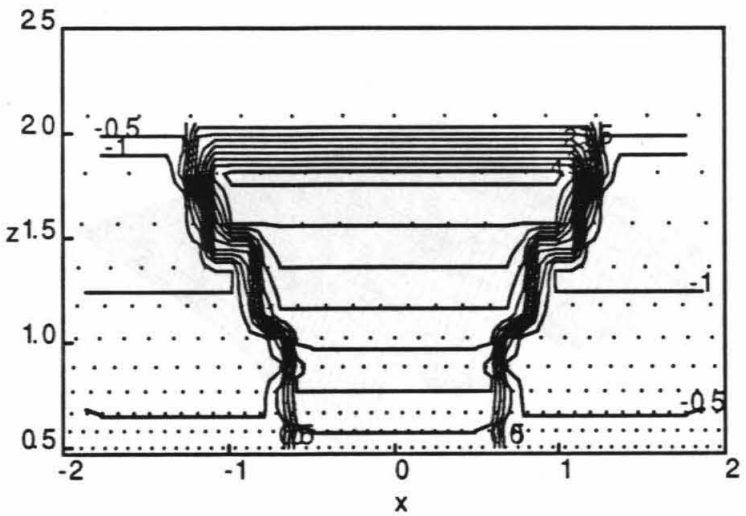

Figure 14

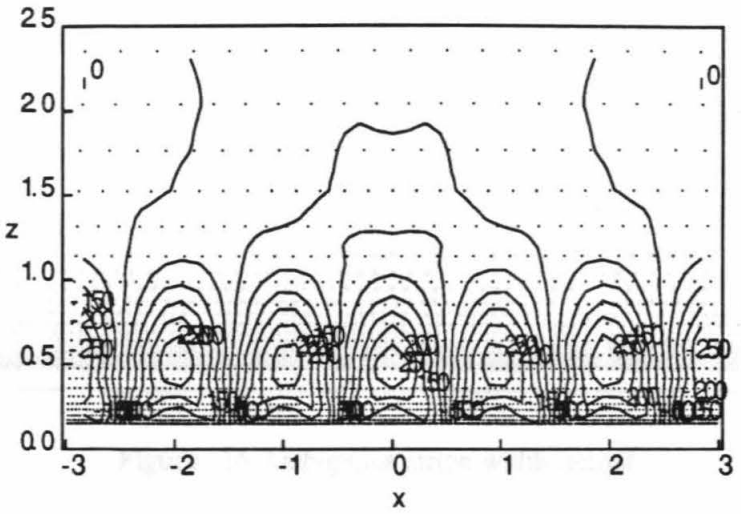

Figure 15a Contour plot

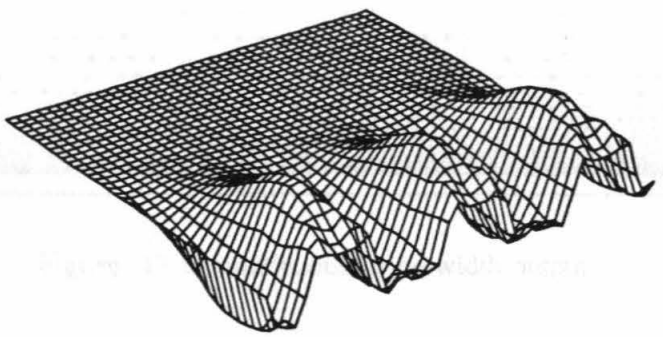

Figure 15b Perspective plot

Figure 15 Function $P(x_0, z_0)$, the total input to a neuron, $a = 1$ network

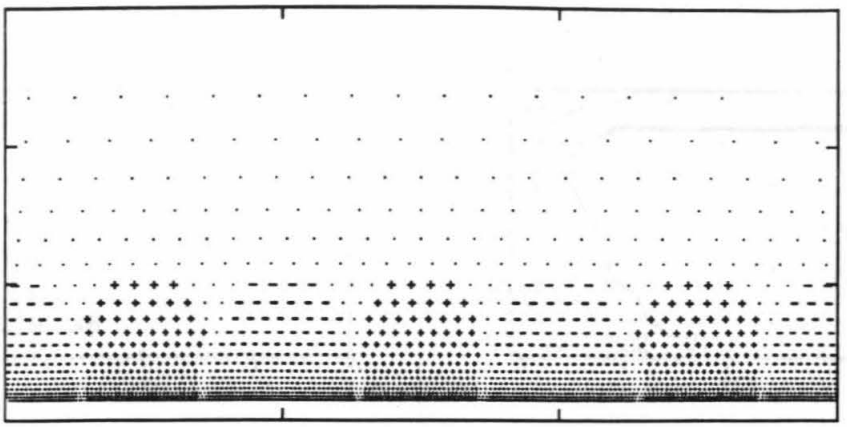

Figure 16 Ambiguous stripe width output

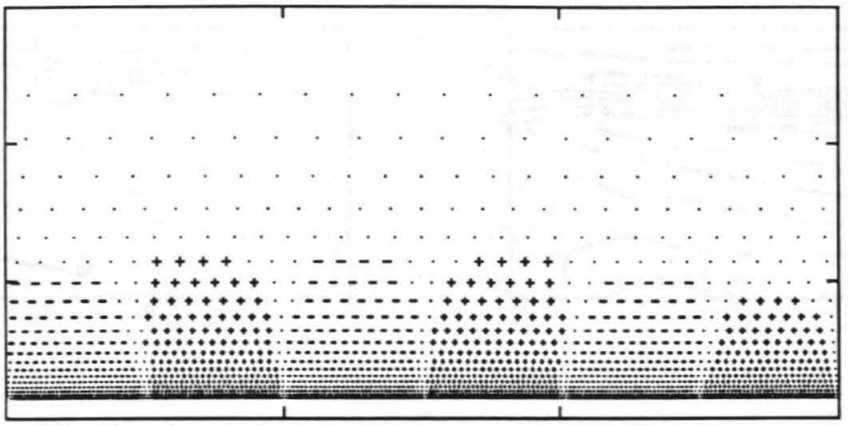

Figure 17 Discontinuous stripe width output

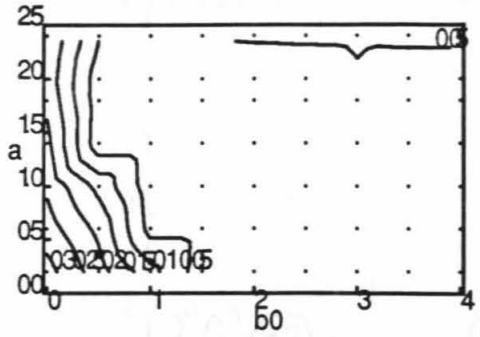

Figure 18a relres 7 pure stripe

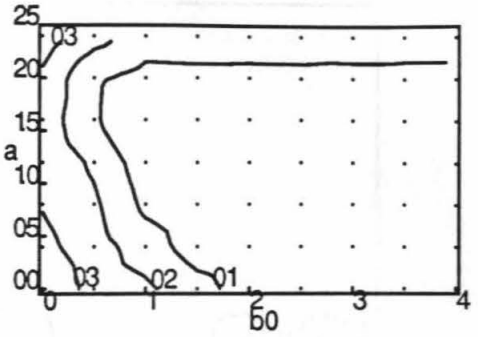

Figure 18d relres 5 pure stripe

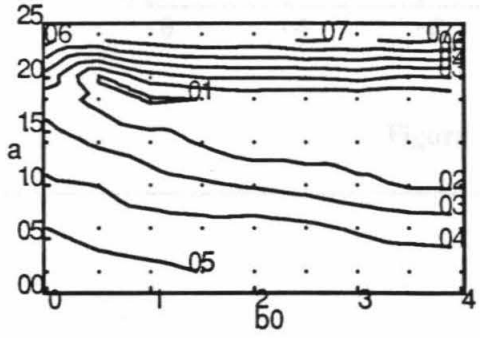

Figure 18b relres 7 ambiguous

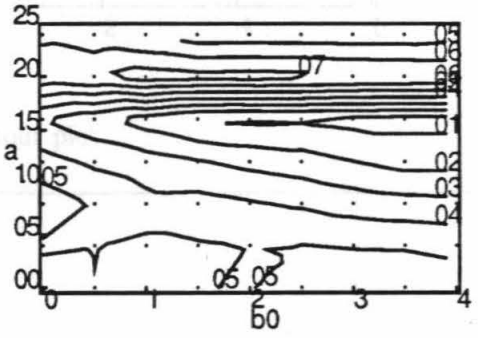

Figure 18e relres 5 ambiguous

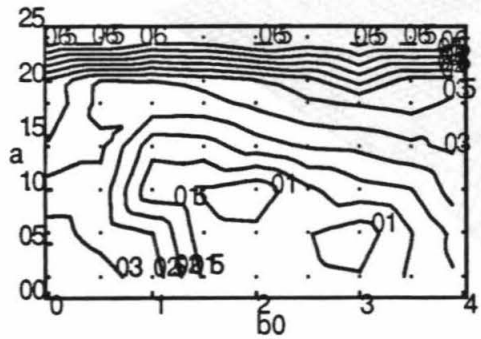

Figure 18c relres 7 discontinuous

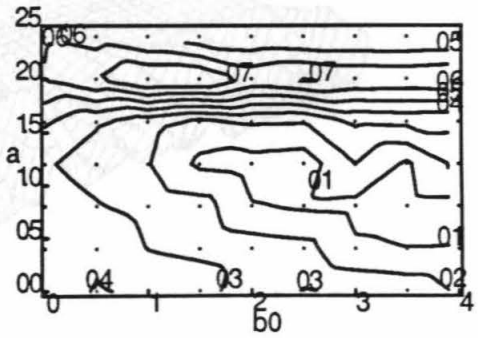

Figure 18f relres 5 discontinuous

Figure 18: Distance scores for three configurations and two neuron distributions

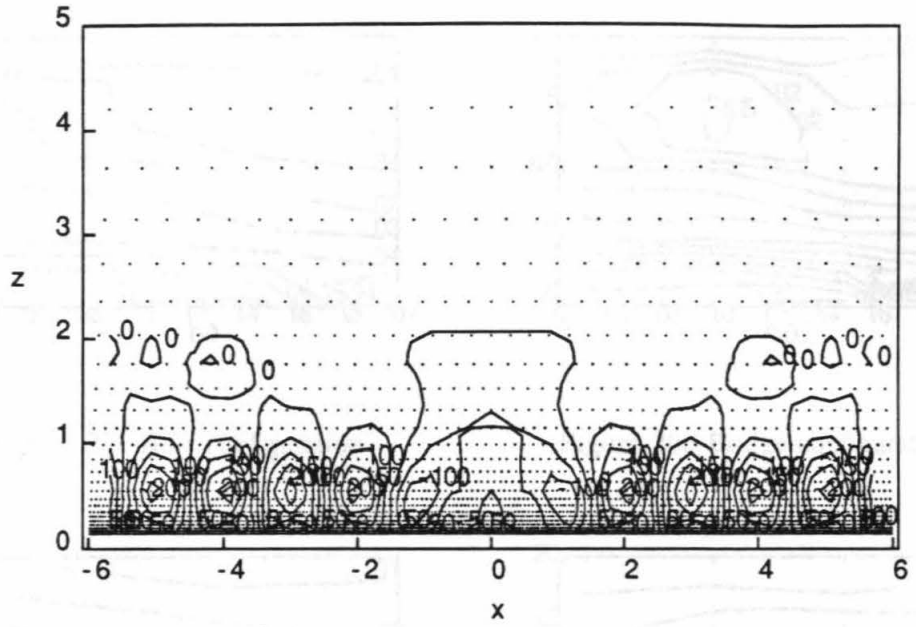

Figure 19a Contour plot

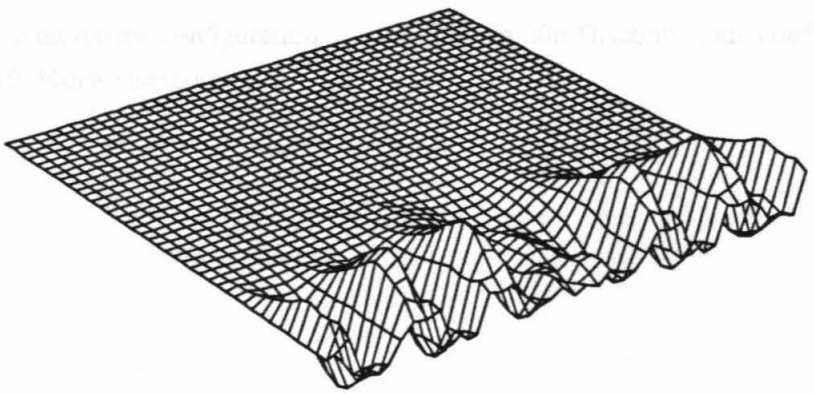

Figure 19b Perspective plot

Figure 19 Function $P(x_0, z_0)$ for bridge configuration

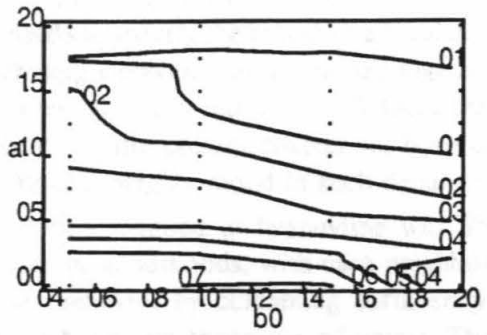

Figure 20a Hole configuration

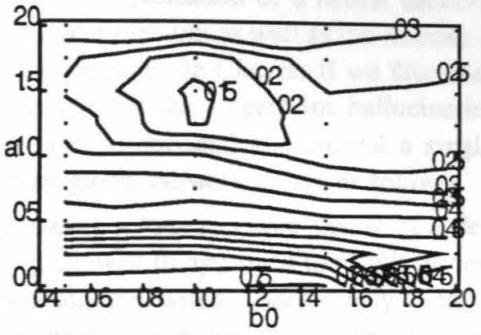

Figure 20b Bridge configuration

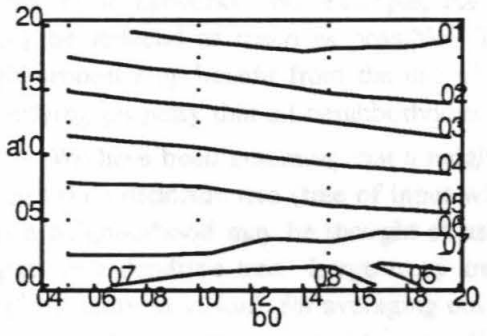

Figure 20c Ambiguous configuration

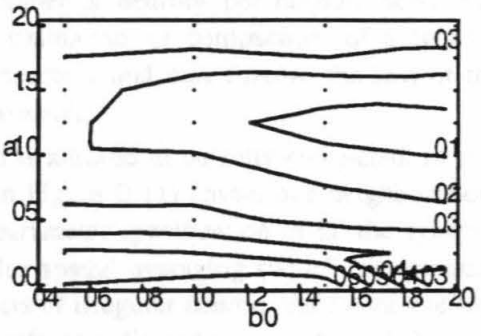

Figure 20d Discontinuous configuration

Figure 20 More constraints

III.5. Summary and Conclusions

To use a continuous medium for understanding the organization of a neural network, one needs to specify the nature of a local neighborhood in the medium as well as the manner in which neighborhoods are connected together to form the medium. In Chapter II we discussed the latter question, introducing a hierarchical organization for the fingerprint hallucination network. In the present chapter we have discussed how to understand and control a single, prototypical neighborhood in such detail that the desired global network behavior follows.

The required understanding was obtained by taking a "microscopic" view of a few adjacent neighborhoods, with each neighborhood being allowed to approach the limit of continuous behavior by containing sufficiently many neurons and wires. Such finely detailed neighborhoods are expensive, of course. They may nevertheless earn their keep, either because the elementary single-neighborhood behavior that the network requires is fairly complicated, or because they can provide an especially simple and understandable solution to the simultaneous constraints of network homogeneity, scale-invariance, and translation-invariance – a solution which can then be approximated with less costly and less easily understood single-neighborhood networks. For example, the large number of neurons per neighborhood may simply be reduced as much as possible. This approximation, or compaction, of a detailed neighborhood may benefit from the use of analog neurons, and may involve the loss of the simplifying property that all neighborhoods are equivalent.

We have been assuming that a neighborhood is allowed to be fully connected, so that one neuron's dendritic tree (tree of input wires, as in Figure II.11) covers one neighborhood. Thus a neighborhood may be thought of as a few-parameter specification of all the synaptic weights in a dendritic tree. Large trees are good for spatial averaging (which can produce noise immunity in vision), for averaging out the effects of irregular neuron placement due (for example) to scale-invariance, and for encoding relatively complicated neighborhood behaviors with many parameters governing the synaptic weights.

Figure 21b illustrates the approach to network continuity taken here, in contrast to the minimal neighborhood size approach illustrated in Figure 21a and used in Chapter II's network. In Figure 21b the dendritic trees grow as the density of neurons increases, but the neighborhood radius stays constant. A second limit $z \to 0$ decreases the radius and thus the size of the smallest perceivable image feature. The $z \to 0$ limit is for network performance, and the dense neighborhood limit is for simplicity and theoretical tractability. Of course, Figures 21a and 21b by no means exhaust the possibilities for approaching continuity. But Figure 21b depicts a "microscope" which can probably be used to understand many networks with regular but somewhat complicated geometric structure. In particular, we have used it to determine the synaptic weights (up to a few free parameters, found experimentally) of the one-dimensional scale-invariant stripe network.

For the one-dimensional stripe network, several methods were used to find the synaptic weights within a finite radius neighborhood with infinite neuron density. First, the sum of outer products formula ($T_{ij} = \sum s_i s_j$) was used to guess a parameterized form of the synapse function. This involved summing all the stripe configurations which turn on a given neuron s_i at $x = 0, z = z_0$; the relevant stripe patterns must all have width $> z_0$ and the proper phase in x to avoid turning off neuron i ($s_i = 0$). The summation is depicted in Figure 22, where one can see that the width $> z_0$ restriction on the allowed stripe patterns produces a kink, at $z = z_0$, in the synapse function's zero contours.

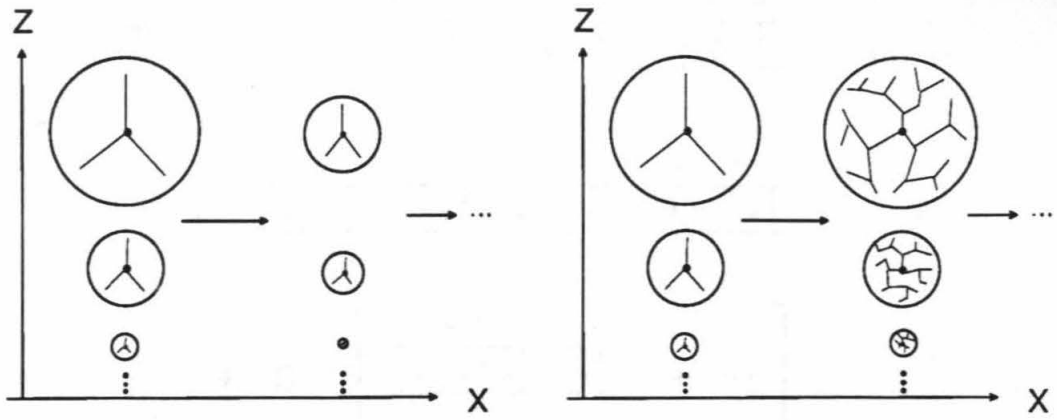

Figure 21a Infinitesimal Neighborhoods **Figure** 21b Finite Neighborhoods

Figure 21 Two Approaches to Continuity

From the sum in Figure 22 one can guess that the synapse function is consistent with the cartoon in Figure 22c, but it is hard to get any more information from the sum and in fact not much more is needed. Neighborhood locality was imposed and the synapse function was parameterized by a very few parameters (one of which controls the relative strength of top-down and bottom-up interactions in the hierarchy). These parameters were chosen to be just numerous and diverse enough to allow the simultaneous satisfaction of a handful of network behavior constraints: that a pure stripe configuration be stable, that a stripe configuration with a "bridge" in it revert to a pure stripe configuration, and so forth. Testing the satisfaction of each of these constraints involved running the network until it found a fixed point, and measuring the Hamming distance between the desired and the achieved fixed points. Thus the network's actual behavior determined the free parameters in the cartoon form for the synaptic weights. The successful network is clearly seen to be a continuous medium in Figure 5, which shows the untruncated input to the neurons in several adjacent neighborhoods of a pure stripe fixed-point configuration.

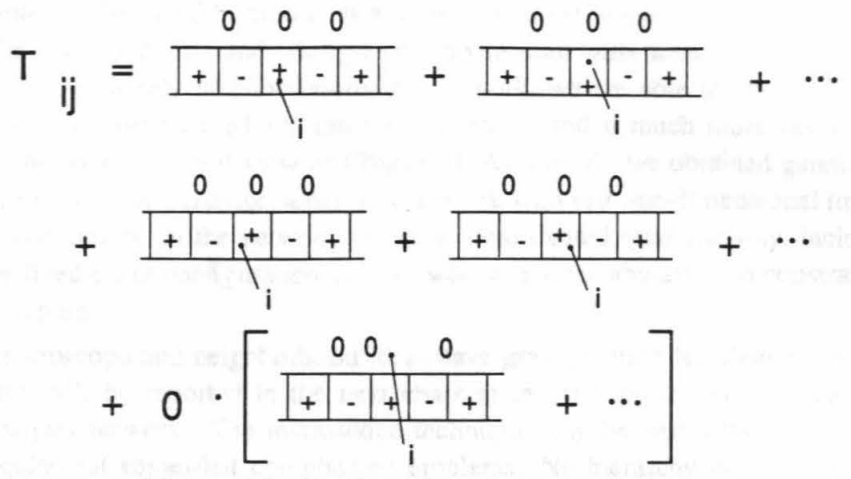

Figure 22.a Configurations to sum in j-plane

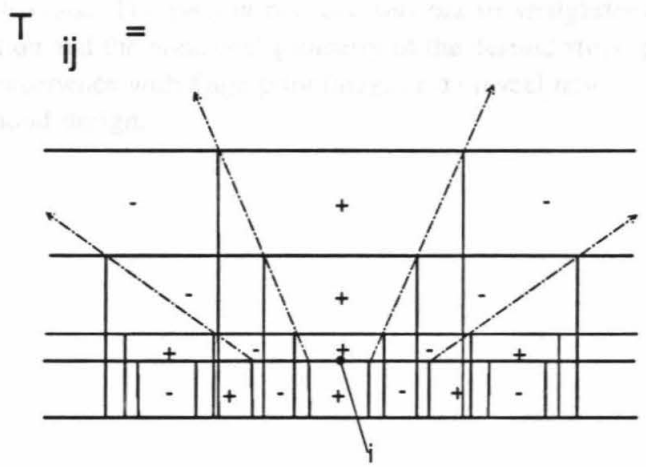

Figure 22.b Superimposed

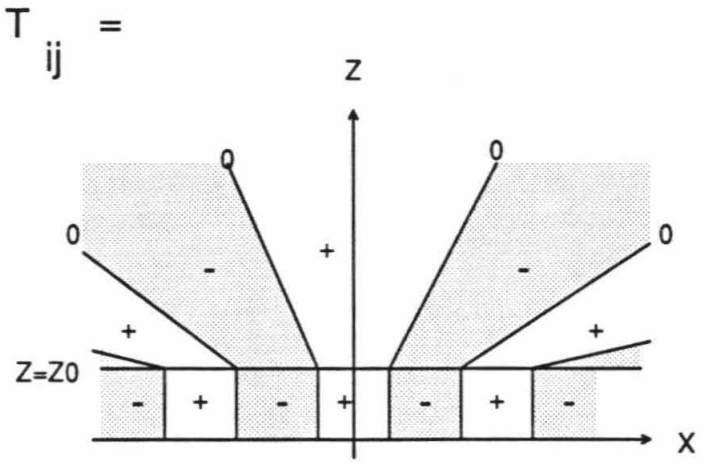

Figure 22.c Sum, depending on weights.

Figure 22 Guess Neighborhood Connections Using $T_{ij} = \sum s_i s_j$

By defining the "neighborhood" as a scale of network organization between the single neuron and the entire network, and taking a continuum limit with infinitely many neurons per neighborhood and infinitely neighborhoods per network, we are able to get a "microscopic" view of a continuous neural medium and thereby understand it much more deeply than was possible with the coarse-grained view of Chapter II. As a result, we obtained genuinely scale-invariant stripe-generating behavior, albeit in a network with just one-dimensional image input. A number of constraints on the network behavior were treated quantitatively, including constraints on the fixed-point configurations, or allowed output vocabulary, and constraints on the input/output mapping.

The microscope and neighborhood ideas have great promise for clearing up some difficulties (which will be reported in the next chapter) in the two-dimensional scale-invariant fingerprint analysis network. The microscope technique may be worth trying for other geometrically regular but somewhat complicated problems. No hierarchy need be involved, but the technique may be too much work to be worth applying to the design of especially simple neighborhood behaviors. The present network was not so straightforward because of its hierarchical organization and the nontrivial geometry of the desired stripe patterns in the hierarchy. Also, further experience with fingerprint images may reveal new constraints which will complicate neighborhood design.

IV Testing Networks with Fingerprint Images

It remains to test our networks with real fingerprint image data. Because of its computational expense, the "microscopic" network of Chapter III was not used directly on two-dimensional images; further development is necessary to apply the lessons learned there to the coarse- grained network of Chapter II. Consequently, we tried only Chapter II's network on the fingerprint data.

The networks discussed in Chapter II were defined by the synapse formula (Chapter II's Equations 11 and 12) and its associated adjustable parameters. They were evaluated by starting them in a random state (each neuron independently assigned a random value of ± 1) and examining the resulting fixed point for stable patterns of stripes, or by starting them in a desirable stripe-like configuration and seeing how far away from the initial configuration the networks wandered. The second method of testing sounds much closer to starting from a fingerprint image than does the first, but there are some differences. An image comes with an assignment of colors (black and white) to pixel locations, but not with any choice of angles for pixel locations as was assumed in examining fixed-point configurations previously. So a plausible input configuration may be obtained by turning on all neurons in a clump which have the correct color for their spatial position, regardless of the neurons' angle index. This results in 6 out of every 12 neurons's being turned on ($s = +1$) in the input state, rather than one out of every 12, in the usual case where the network has two possible colors and six possible angles. The second difference concerns the hierarchy: it is desired that the stable states of the network have stripe height proportional to stripe width, but it is unfair to start the network off this way since the stripe width is one of the image properties which the network is supposed to compute. So it was arbitrarily decided to let the image affect the starting configuration only for the network's bottommost level; all other levels were started with all neurons in their off state ($s = -1$).

It appears, then, that to process a fingerprint image the network will have to be used in a slightly different way than that previously reported. To explore this difference the network was first tested on the fabricated images which are somewhat simpler than the real ones ultimately used.

4.1. New Input Method

The artificial images used to test and develop the new input method are shown in Figure 1; they may be compared with the real fingerprint image and its closeups, shown in Figure 2. All of the latter images were collected by Megdal [Megdal 83]. The first logical experiment to try is to combine the optimized parameter settings from Chapter II (specifically those of the hierarchically balanced network of Figure III.18) with the initial configuration obtained from the smaller image in Figure 1a and run the network. It converges to the null configuration (all neurons off) but if I is decreased slightly to compensate for the smaller system size, then it converges to the configuration of Figure 3. The result is a stripe-like

configuration which, sadly, has nothing to do with the input image in Figure 1a. In Figure 3, we have $a = 4$ $c = 3.7$ $c' = 8$ $i = 50$ $K = 20$ $H = 1.05$ $\vec{f} = (30, 30, 10, 10, 20, 20)$ (recall that $\vec{f}$ gives the relative weights of top-down,sideways, and bottom-up connections) $\alpha = 2.1$ and $s_{off} = .5$.

Figure 1a K=10

Figure 1b K=20

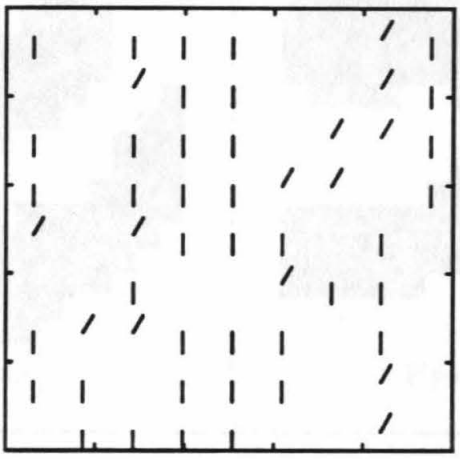

Figure 3

Thus, it is possible for a network to move from a stripy but overactive input state to a stripy fixed configuration and "forget" about the input along the way. This suggests that the input image be used as a magnetic field term (altering the neural thresholds) as well as a starting configuration, since such a term affects the evolution of the network at all times and cannot be forgotten. This type of input is equivalent to a set of light-sensitive input neurons which affect the rest of the network but are not affected by it, so that for a fixed image they have fixed values. This term must be introduced with some adjustable weight similar to the old threshold H, and in fact the new threshold $H_{image}s_{image}$ (with $s_{image} = \pm 1$ or 0) is scaled in the same way as H is: both are multiplied by the total input an average neuron of the given level would have if all its neighbors were on.

Figure 2a Complete fingerprint

Figure 2b "elbow"

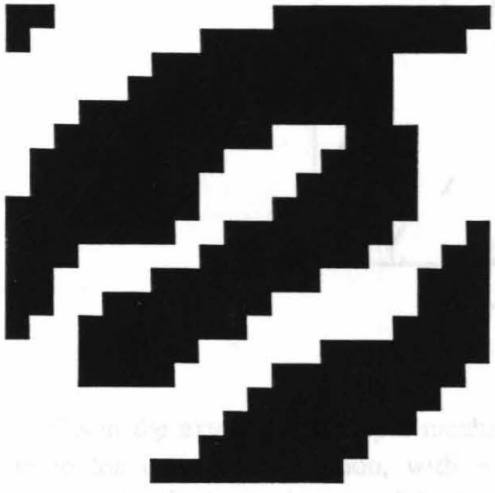

Figure 2c "thin"

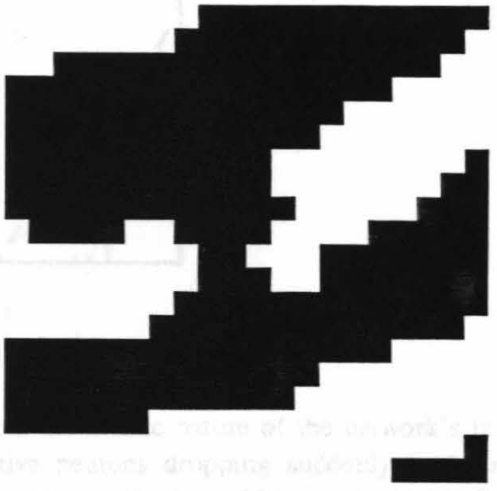

Figure 2d "bridge"

Figure 2. Fingerprint image and 20 x 20 input windows

One virtue of applying the input magnetic field to the bottom layer of the network, rather than to any higher layers, is that it can compensate for the bottom-up connections whose absence most strongly affects this layer. In fact, the input image may be thought of as the input from hypothetical very high-resolution missing layers below the lowest layer in the network.

To find acceptable values for H_{image} and the other parameters, it was first determined that $H_{image} \approx -.15$ is close to the smallest $|H_{image}|$ which allows no disagreement between input and output color, and the other parameters were tuned to produce reasonably lined up angles under the constraint that colors must agree with the input. The small artificial image of Figure 1a was used for these tests. The resulting image and parameter values are shown in Figure 4; note that a z-dependent I term I', analogous to c' in Equation (II.12), has been introduced to get reasonable behavior at several levels in the hierarchy simultaneously. Also,

the value of a (the angle conformity term) has been drastically decreased, and the relative strength $f_b \equiv f_5 = f_6$ of bottom-up connections has been increased.

For Figure 4, $H_{image} = -.15$ $a = .5$ $c = 4$ $c' = 6$ $i = 20$ $i' = 15$ $\alpha = 2.1$ $K = 10$ $m = 6$ $s_{off} = .5$ $H = 1.05$ $\vec{f} = (30, 30, 10, 10, 100, 100)$.

Thus, the new input method allows part of the desired computation to be done: the change of representation from a binary image to a hierarchical array of color and angle values. Further developments will improve this part of the computation and approach the next part: the modification of the input image to eliminate all features save stereotypical stripes, branches and ridge endings. Such an image would allow minutia detection to be a simple and local operation.

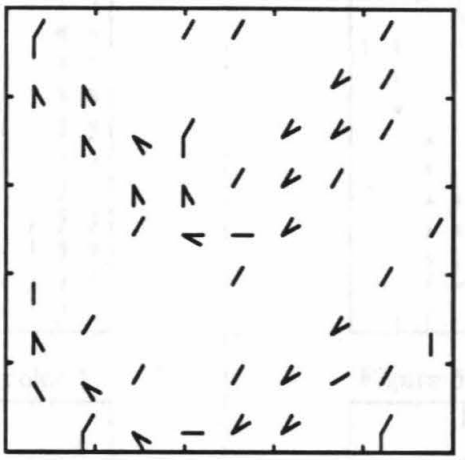

Figure 4

Given the external field input mechanism and the drastic nature of the network's response to the initial configuration, with $n/2n$ active neurons dropping suddenly to $1/2n$ neurons on in each clump, it is possible that the best input method would be to start with the zero initial configuration (all neurons off) and rely on the image magnetic field term alone for input. Then the transient behavior would be much less violent. This possibility has not been explored for the present network.

4.2. Minor Improvements

In preparation for testing on real images it is desirable to use a larger image size, so that both wide and narrow stripes can be tried out. Also, the network of Figure 1a had periodic boundary conditions and it must be checked that zero boundary conditions (all neurons off outside the image) also work. Using the image of Figure 1b as input, with zero boundary conditions and a smaller H_{image} to encourage reasonable image modifications, the configuration of Figure 5 results. Note that f_b has been decreased somewhat to allow intermediate levels to receive top-down input as well as bottom-up; a value of $f_b = 50$, however, fails to turn on any neurons except those on the bottom level (level 0).

One of the phenomena encountered in these manual parameter searches is the networks' sensitivity to H. Any major change of another parameter, or of network size, requires that

Figure 5a level 0 color 1

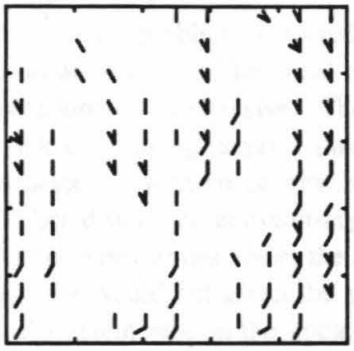

Figure 5b level 1 color 1

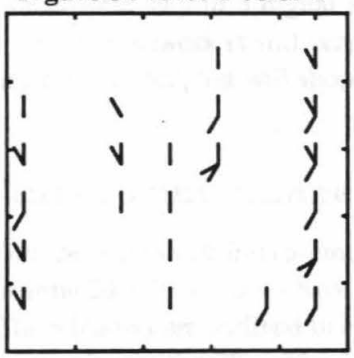

Figure 5c level 2 color 1

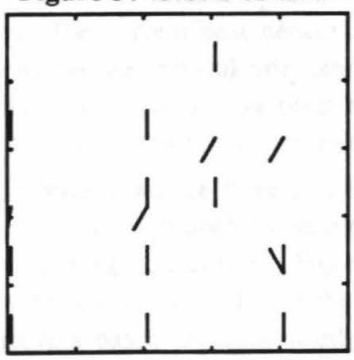

Figure 5d level 3 color 1

Figure 5e level 0 color -1

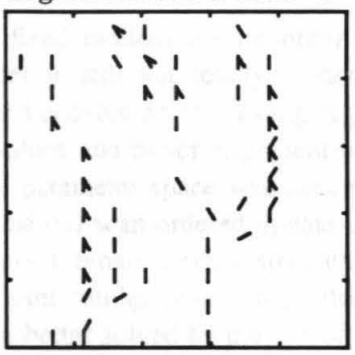

Figure 5f level 1 color -1

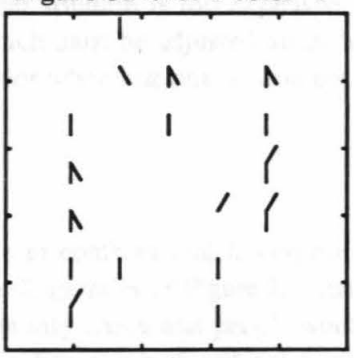

Figure 5g level 2 color -1

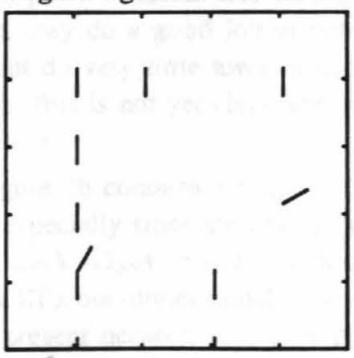

Figure 5h level 3 color -1

Figure 5: $a = .5$ $c = 4$ $c' = 6$ $i = 20$ $i' = 15$ $K = 20$ $H = 1.05$ $H_{image} = -.07$ $\vec{f} = (30, 30, 10, 10, 65, 65)$. **15 sweeps to converge**

H be changed by an amount that is usually greater than the allowed range of H for the old network. An excessive value of H results in too many neurons per clump being on, and therefore less definite angle information. Too small a value for H results ultimately in all neurons in a clump turning off, but first a more subtle bug appears. A single neuron per clump turns on, but it always has the same angular orientation. This is a result of the update order used: when H favors just one neuron on per clump, and initially half the neurons are on, during the first sweep neurons are indiscriminately turned off (always in the same angular order) until just one (always pointing the same way) remains on. The resulting configuration may be quite stable due to the angular conformity term. In short, the special configuration which has all angles pointing in the favored direction and color agreeing with the input image has an unfair advantage due to the updating order of the neurons within a sweep.

This update problem is solved by introducing a fixed random update order for all sweeps in what follows. The fixed random update order is still not totally random, but it is computationally inexpensive. The new update order uncovers an interesting region of parameter space, allowing larger a (angular conformity) values and better alignment of local orientation detectors with stripe direction. This region of parameter space was occupied by networks afflicted with the constant-angle bug so long as the old scan-ordered update scheme was used. The improvement under the new scheme is clearly a dynamic effect since changing the update order would not affect the stability of a fixed-point configuration. In an electronic or biological implementation the update problem would be better solved by the use of analog neuron values and simultaneous update. Simultaneous update of analog neurons corresponds to a very small time step in a digital simulation. The latter solution is too expensive to use here. The resulting network is still very sensitive to H, which must be adjusted often during a manual parameter search, but will show much better behavior when just one neuron per clump is on.

4.3. Fingerprint Image Results

With these network improvements it is at last time to confront real fingerprint image data. Interesting 20 x 20 windows have been taken from the fingerprint of Figure 2, obtained by Megdal. The windows are outlined in Figure I.1, so that one may check that people would have enough information in the 20 x 20 window to discriminate between real and noise minutiae. The image has been filtered by convolution with a 3 x 3 pixel Gaussian and then globally thresholded. The current best networks are disappointing; they do a good job of converting the image to the hierarchical orientation representation, but do very little towards using this representation to suppress fingerprint noise. The reason for this is not yet clear, and it could be that further minor parameter changes could fix the problem.

In Figure 2 we see three interesting windows. Figure 2b contains a ridge end which could be turned into a branch by an overzealous network, especially since the nearby "elbow" suggests a missing connection. Figure 2c has thin and thick ridges in one neighborhood, and Figure 2d contains a bridge of the sort which Chapter III's one-dimensional network was able to suppress based on ridge-width information. The present network fails to suppress it using two-dimensional shape information. The final network parameters were obtained by minimizing h_{image}; any further decrease results in the introduction of more defects in the bridge configuration, rather than their removal. f_b could then be decreased a bit to encourage information to flow both up and down the hierarchy. The results for the three real fingerprint

windows, as well as for the artificial training image, are shown in Figures 6,7,8,9. (Beware of the color reversal between Figures 8a and 8b, and between Figures 9a and 9b.) Some slight simplification of the images is apparent, but not nearly enough to be of any help to a branch or ridge end detector. The main effect of the network is to deduce the direction of stripe flow everywhere and at all the relevant scales of image resolution. For the wider stripes, this is definitely a nonlocal operation which makes use of the higher levels of the hierarchy: the network must send color information up to the scale where orientation may be detected locally and then ship angle information back down.

Figure 6a Input

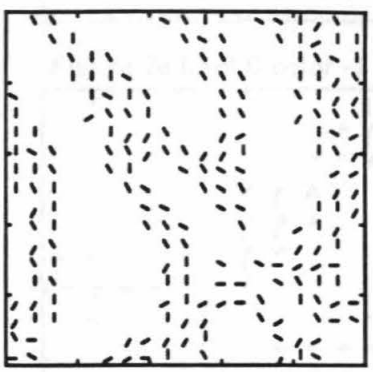

Figure 6b level 0 color 0

Figure 6: Network as in Figure 7. 11 sweeps to converge

The network of Figure 7 converges quite rapidly, usually in 10-20 sweeps. This is consistent with experience with the CAM. For such a small window size (20 x 20), however, we cannot tell what the asymptotic convergence time is as a function of system size: it could be constant, or proportional to the system diameter, though not proportional to the network area. An optimal algorithm would apparently require just time $\sim$ log(diameter) due to the use of a few long wires to obtain and use global ridge width information.

One possible explanation for the network's failure, aside from the simple possibility that it will succeed if the parameter search is extended, is that changing to the orientation representation and using that representation to modify the image are two different computations which are best done with two different parameter settings. Then one could imagine doubling the network to perform the two tasks or performing them in sequence with one network by allowing an external command to switch each synapse's value between the two relevant possibilities. This modification would make the computation somewhat sequential.

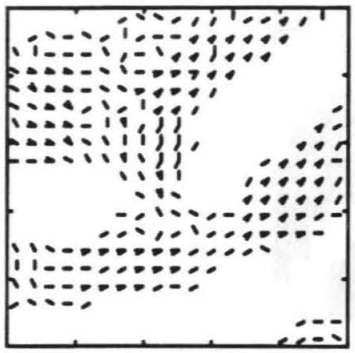

Figure 7a level 0 color 1

Figure 7b level 1 color 1

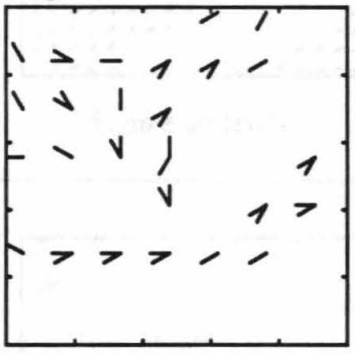

Figure 7c level 2 color 1

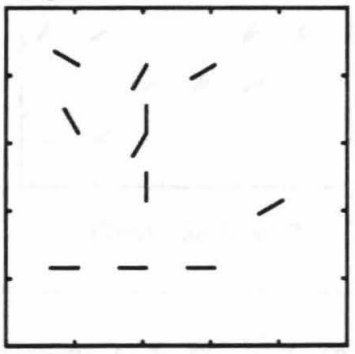

Figure 7d level 3 color 1

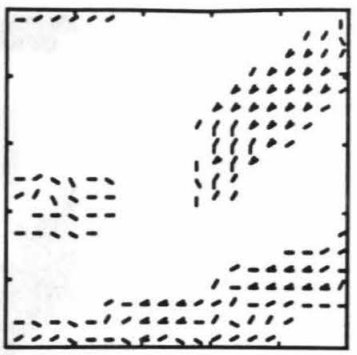

Figure 7e level 0 color -1

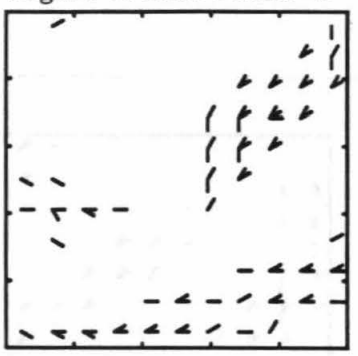

Figure 7f level 1 color -1

Figure 7g level 2 color -1

Figure 7h level 3 color -1

Figure 7: $a = 1$ $c = 4$ $i = 20$ $i' = 15$ $K = 20$ $m = 6$ $H = 1.07$ $H_{image} = -.05$ $\vec{f} = (30, 30, 10, 10, 50, 50)$. **13 sweeps**

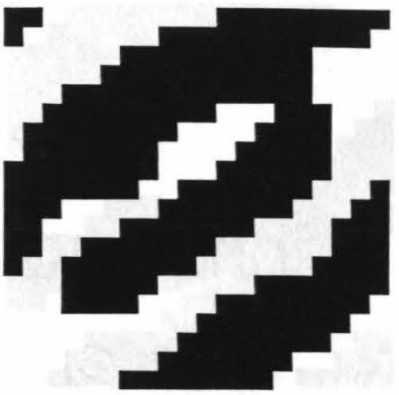

Figure 8a Input

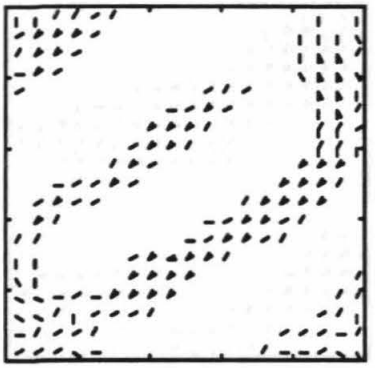

Figure 8b level 0

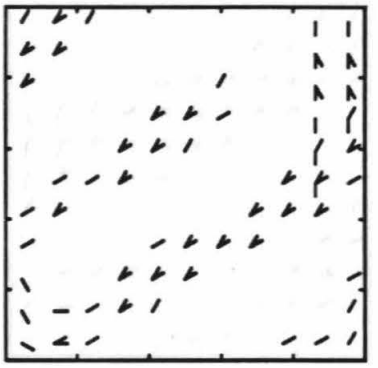

Figure 8c level 1

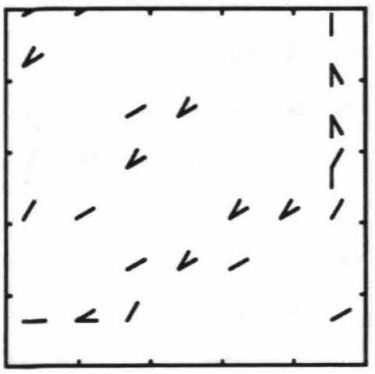

Figure 8d level 2

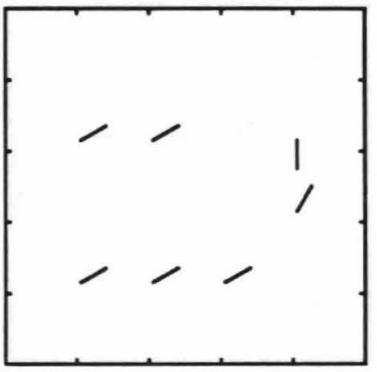

Figure 8e level 3

Figure 8: As in Figure 7. 19 sweeps

Figure 9a Input

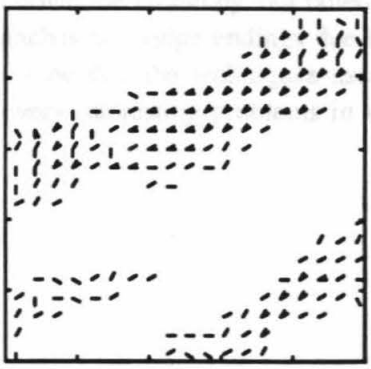

Figure 9b level 0

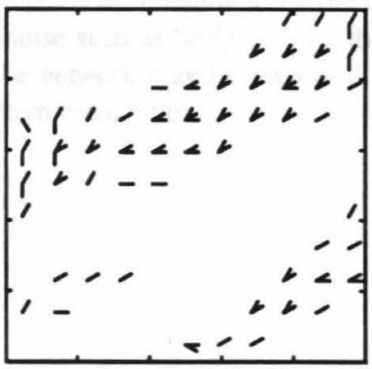

Figure 9c level 1

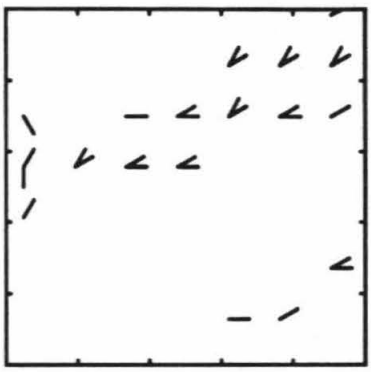

Figure 9d level 2

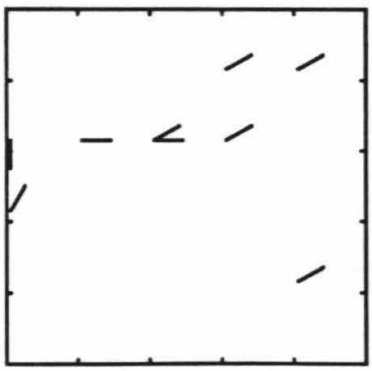

Figure 9e level 3

Figure 9: As in Figure 7. 10 sweeps

4.4 Conclusions

The collision with genuine fingerprint image data forced the $2+\epsilon$-dimensional neural network advertised in Chapter II to be modified in a number of important ways. First, the input method had to be altered to include the use of a magnetic field term as well as a starting configuration. Such a term is easy to implement electronically or biologically and was previously overlooked due to the use of "hallucination," or fingerprint output from random input, to find out what kinds of terms were needed in the synapse function. Second, the update method and boundary conditions had to be changed to more closely resemble what would be encountered in a biological or electronic version of the network. Third, the parameters appearing in the synapse formula had to be adjusted to the new circumstances with a manual search based on visual inspection of the frozen (fixed-point) configurations. Despite the extensive network modifications, the resulting network did only half of its job. It computed the hierarchical orientation representation for an input image, a nonlocal operation exercising communications both up and down the hierarchy, but failed to use the computed information to rid the image of spurious branches and ridge endings due to fingerprint noise such as bridges. Still, the partial results give hope that the techniques used to design the network may be improved to yield solutions to very interesting problems in vision and pattern recognition.

Bibliography

[Brown 84] Brown, C.M.,
Computer Vision and Natural Constraints,
Science, Vol. 244 No. 4655, 22 June 1984, p. 1299.

[CG&A 85] IEEE Computer Graphics and Applications,
April 1985, p. 14.

[Conrad 85] Conrad, M.,
On Design Principles for a Molecular Computer,
Communications of the ACM, Vol 28 Number 5, May 1985.

[Hillis 82a] Hillis, W.D.,
New Computer Architectures and their Relationships to Physics or Why Computer Science is No Good,
International Journal of Theoretical Physics, Vol.21, Nos 3/4, 1982 p. 255.

[Hillis 82b] Hillis, W.D.,
The Connection Machine: A Computer Architecture Based on Cellular Automata,
Physica 10D, 1984 p. 213.

[Hinton 83] Hinton, G., and Sejnowski, T.,
Analysing Cooperative Computation,
Proc. 5th Annual Conference of the Cognitive Science Society,
Rochester, New York, May 1983.

[Hopfield 82] Hopfield, J.J.,
Neural Networks and Physical Systems with Emergent Collective Computational Abilities,
Proceedings of the National Academy of Science USA 79, April 1982, pp. 2554-2558.

[Hopfield 84] Hopfield, J.J.,
Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons,
Proceedings of the National Academy of Science USA 81, May 1984, pp. 3088-3092.

[Hopfield 85] Hopfield, J.J. and Tank, D.,
'Neural' Computation of Decisions in Optimization Problems,
Biological Cybernetics, 52, p. 141-152, 1985.

[Johnston 85] Johnston, David,
Computer Could Point Finger at Murderers,
Los Angeles Times, Friday, June 28, 1985 Part V p.1.

[Kirkpatrick 85] Kirkpatrick, S., and Swendsen, R.,
Statistical Mechanics and Disordered Systems,
Communications of the ACM, April 1985 p. 363.

[Leighton 85] Leighton, T., and Leiserson, C.E.,
Wafer Scale Integration of Systolic Arrays,
IEEE Transactions on Computers, Vol. C-34, No. 5, May 1985 p.448.

[Mead 82] Mead, C.A. and Rem, M.,
Minimum Propagation Delays in VLSI,
IEEE Journal of Solid-State Circuits, Vol SC-17 No. 4, August 1982 pp.773-775.

[Mead 83] Mead, C.A.,
VLSI and the Foundations of Computation,
Information Processing 83, R.E.A. Mason, ed., Elsevier Science Publishers B.V., 1983.

[Megdal 83] Megdal, Barry,
VLSI Computational Structures Applied to Fingerprint Image Analysis,
PhD Thesis, Computer Science Dept., California Institute of Technology, 1983.

[Merzenich 84] Merzenich, M. M., Nelson, R.J., Stryker, M.P., Cyander, M.S., Schoppmann, A., Zook, J.M.,
Somatosensory Cortical Map Changes Following Digit Amputation in Adult Monkeys,
J. Comp. Neurol, 224, p. 591, 1984.

[NEC] Nippon Electric Company,
Automated Fingerprint Identification System,
(Advertising Booklet).

[Penrose 73a] Penrose, L.S., and Ohara, P.T.
The Development of the Epidermal Ridges,
Journal of Medical Genetics 10, 1973 p. 201.

[Penrose 73b] Penrose, L.S.,
Fingerprints and Palmistry,
The Lancet, 2 June 1973, p. 1239.

[Penrose 79] Penrose, R.,
The Topology of Ridge Systems,
Ann. Hum. Genet., 42, 435, p. 28, 1979.

[Platt 85] Platt, J.,
Sequential Threshold Circuits,

Technical Report Number 5197:TR:85,
Masters Thesis, Computer Science Dept., California Institute of Technology, 1985.

[Rosenfeld 84] Rosenfeld, A., ed.
Multiresolution Image Processing and Analysis
Springer-Verlag 1984.

[Sivilotti 85] Sivilotti, M., Emerling, M., and Mead, C.A.,
A Novel Associative Memory Implemented Using Collective Computation,
Proceedings, 1985 Chapel Hill Conference on VLSI, ed. Henry Fuchs, p. 329,
Computer Science Press 1985.

[Terzopoulos 84] Terzopoulos, D.,
Multilevel Reconstruction of Visual Surfaces: Variational Principles and Finite-Element Representations,
Multiresolution Image Processing and Analysis, A. Rosenfeld, ed.
Springer-Verlag 1984.

[Wilson 74] Wilson, K., Kogut, J.,
The Renormalization Group and the ϵ Expansion,
Physics Reports Vol 12C Number 2, August 1974..