Outlook

- Networks with feedback connections
- Associative memory
- Hopfield model

- Self-organizing maps
- Hardware for neural networks
- Dimensionality reduction
- Principal Component Analysis
- Independent Component Analysis

Self-Organizing Maps

- a network to map similar input patterns (close to each other in input space)
- onto contiguous locations in output space

- network based on unsupervised learning
- often used with 2D output space for data visualization

- a.k.a Self-organizing feature maps
- a.k.a Kohonen network

(Kohonen 1981) (Kohonen: Self-Organizing Maps, Springer 2001)

A third category of neural networks

- Perceptron and MLP feed-forward networks
- recursive feed-back networks

Self-organizing maps:

- *neighboring cells* in a neural network
- *compete* in their activities
- by mutual lateral interactions
- and *develop adaptively*
- into *specific detectors* of *signal patterns*
- unsupervised self-organizing learning

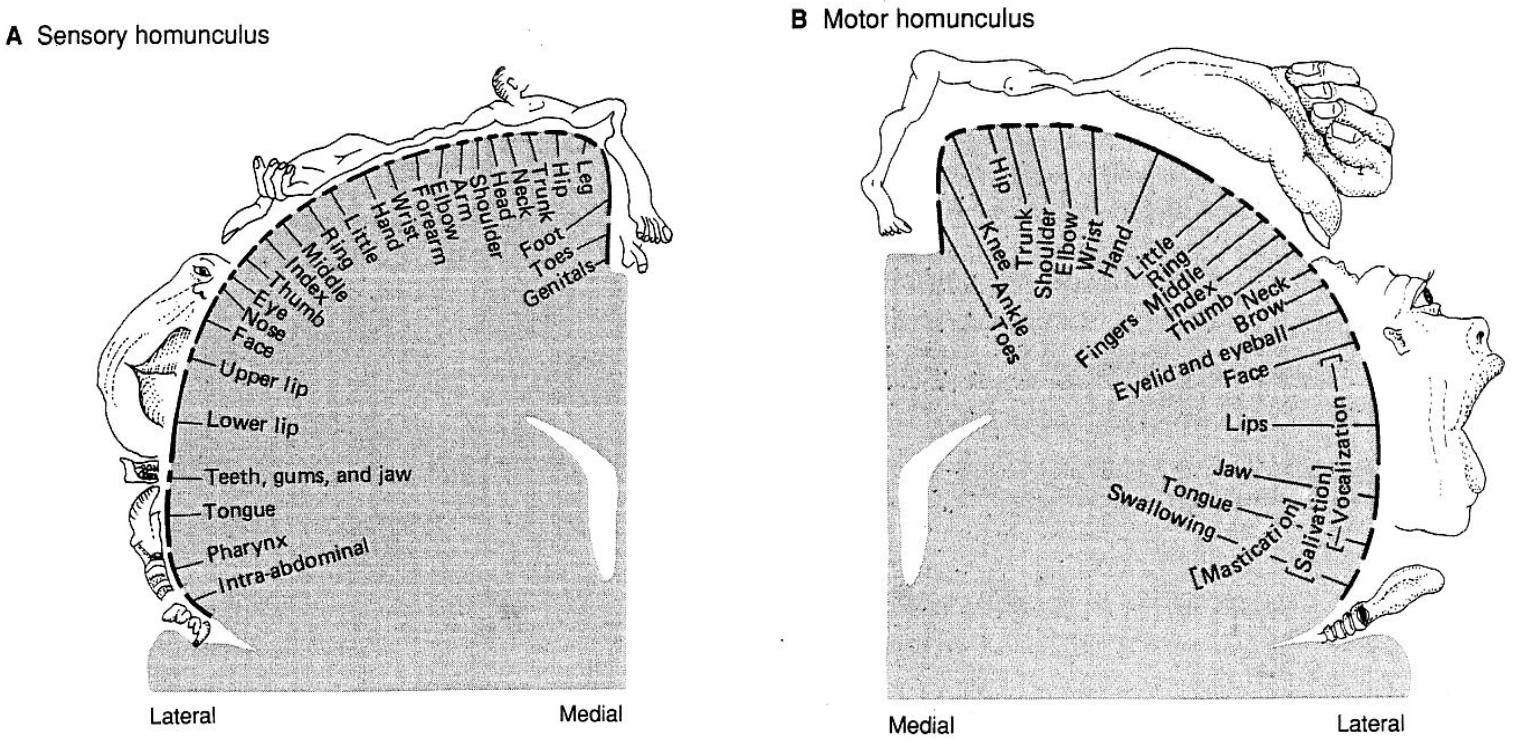
SOM: relation to neurobiology

- topographical organization of the brain
- *location* of the neural response often corresponds
- to a *specific modality* and *quality* of the sensory signal

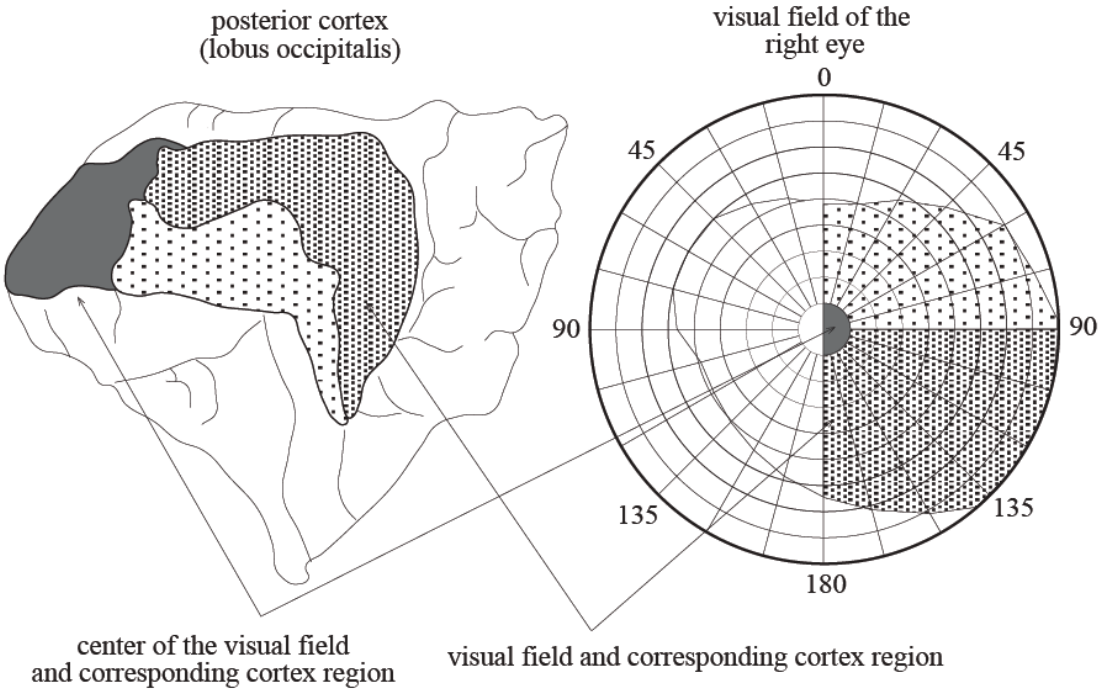
- (primary) visual cortex areas V_1 and V_2
- auditory cortex
- somatotopic areas
- motor-map areas

- internal representations of the brain organized spatially
- mapping high-dimensional inputs to 2D cortex surface

SOM: relation to neurobiology



SOM: relation to neurobiology



- mapping of visual-field (retina coordinates) to brain regions in the visual cortex

SOM: relation to neurobiology

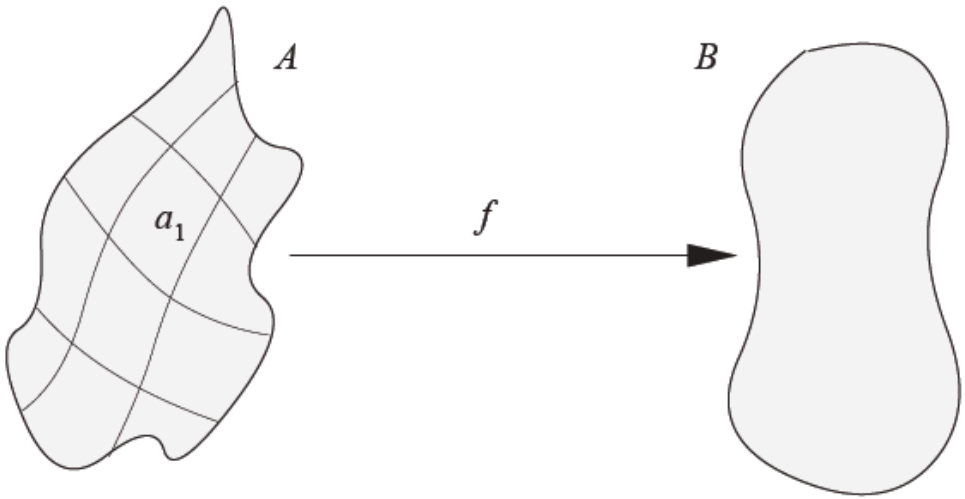
- visualization of *eye-dominance* pattens in the visual cortex
- dark (light) stripes indicate the left (right) eye

Motivation

In many cases we have experimental data which is coded using n real values, but whose effective dimension is much lower. Consider the case in which the data set consists of the points in the surface of a sphere in three-dimensional space. Although the input vectors have three components, a two-dimensional Kohonen network will do a better job of charting this input space than a three-dimensional one. Some researchers have proposed computing the effective dimension of the data before selecting the dimension of the Kohonen network, since this can later provide a smoother approximation to the data.

The dimension of the data set can be computed experimentally by measuring the variation in the number of data points closer to another data point than a given ε, when ε is gradually increased or decreased. If, for example, the data set lies on a plane in three-dimensional space and we select a data point ξ randomly, we can plot the number of data points $N(\varepsilon)$ not further away from ξ than ε. This number should follow the power law $N(\varepsilon) \approx \varepsilon^2$. If this is the case, we settle for a two-dimensional network. Normally, the way to make this computation is to draw a plot of $\log(N(\varepsilon)$ against $\log(\varepsilon)$. The slope of the regression curve as ε goes to zero is the fractal dimension of the data set.

Task



- network maps input space A into output space B
- every region of the input space should be covered
- for an input in a subregion (e.g. a_1) only one neuron should fire
- inputs from neighbor subregions in A should be neighbors in B

Classical vector-quantization

- *approximate* a given probability density function
- of vector input variables x
- using a finite number of *codebook* vectors

- given the codebook
- find the reference vector m_c closest to x
- e.g. minimize $E = \int ||x - m_c||^r p(x) dx$
- the expected r-th power of the reconstruction-error

- in general, no closed-form solution
- use iterative approximation algorithms

Components of the self-organizing map

- a distance measure in the n-dimensional input space
- for example, Euclidean distance

- a set of M neurons S_i
- with a real-valued weights-vector $w_i = (w_{i1}, \ldots, w_{in})$
- neurons calculate $||w - x||$
- the neuron closest to a given input x is activated

- a position in the map space
- a neighborhood function

Neighborhood function

- $\phi(i, k)$ represents the strength of the coupling between neurons i and k during learning
- for example, $\phi(i, k) = 1$ for units within radius r and 0 outside.

- learning usually starts with large radius, so that the network can adapt to global structure of the data
- gradually reduce the radius during learning to optimize the local structure

Topological neighborhood functions

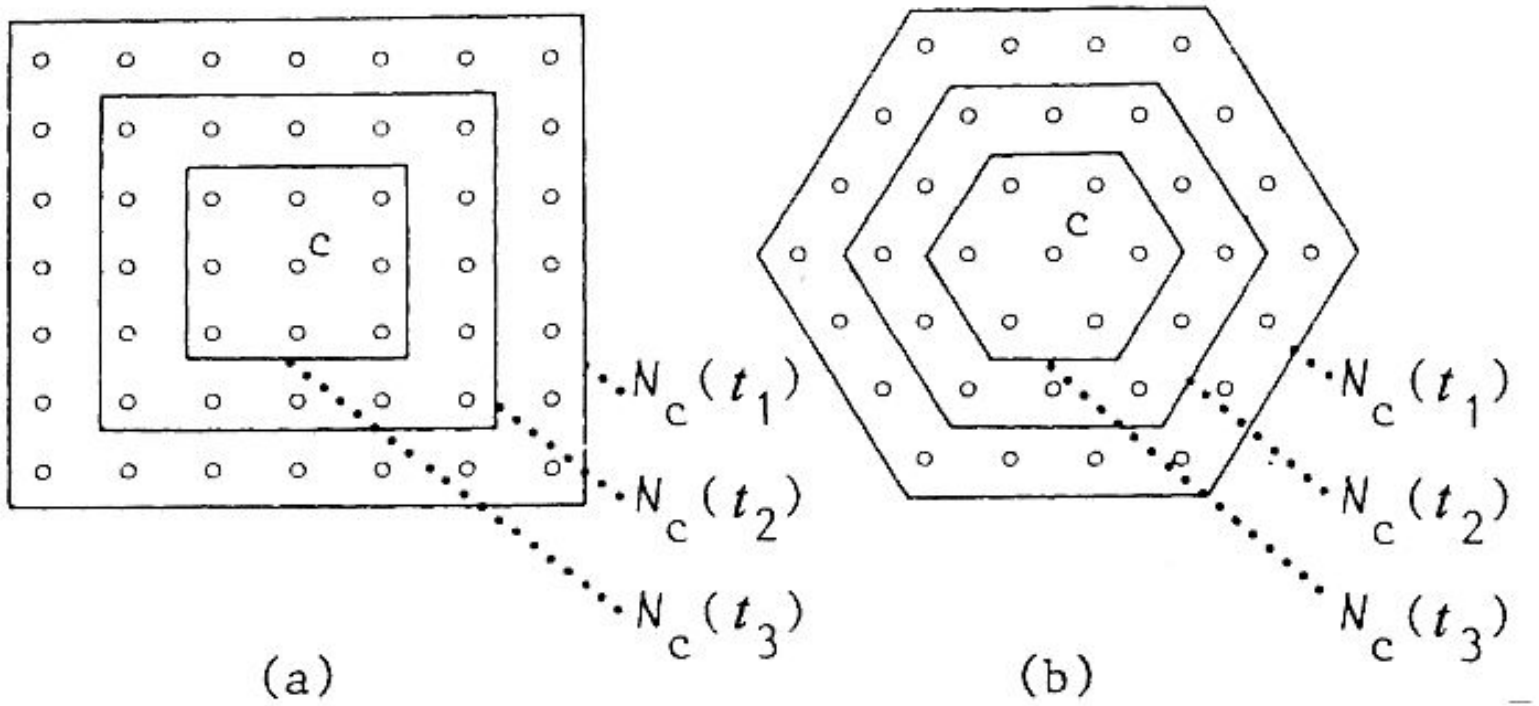


Fig. 3.4. a, b. Two examples of topological neighborhood ($t_1 < t_2 < t_3$)

Typical neighborhood functions

In the literature, two simple choices for $h_{ci}(t)$ occur frequently. The simpler of them refers to a *neighborhood set* of array points around node c (Fig. 3.4). Let their index set be denoted N_c (notice that we can define $N_c = N_c(t)$ as a function of time), whereby $h_{ci}(t) = \alpha(t)$ if $i \in N_c$ and $h_{ci}(t) = 0$ if $i \notin N_c$. The value of $\alpha(t)$ is then identified with a *learning-rate factor* $(0 < \alpha(t) < 1)$. Both $\alpha(t)$ and the radius of $N_c(t)$ are usually decreasing monotonically in time (during the ordering process).

Another widely applied, smoother neighborhood kernel can be written in terms of the Gaussian function,

$$h_{ci}(t) = \alpha(t) \cdot \exp\left(-\frac{||r_c - r_i||^2}{2\sigma^2(t)}\right) , \tag{3.4}$$

where $\alpha(t)$ is another scalar-valued "learning-rate factor," and the parameter $\sigma(t)$ defines the width of the kernel; the latter corresponds to the radius of $N_c(t)$ above. Both $\alpha(t)$ and $\sigma(t)$ are some monotonically decreasing functions of time.

The learning algorithm

- the set of neurons with their weight vectors w_i
- the neighborhood function ϕ in input space
- a learning constant η

- learning constant and neighborhood function are (usually) changed during the learning

The learning algorithm

start: The n-dimensional weight vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$ of the m computing units are selected at random. An initial radius r, a learning constant η, and a neighborhood function ϕ are selected.

step 1: Select an input vector ξ using the desired probability distribution over the input space.

step 2: The unit k with the maximum excitation is selected (that is, for which the distance between $\mathbf{w}_i$ and ξ is minimal, $i = 1, \ldots, m$).

step 3: The weight vectors are updated using the neighborhood function and the update rule

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta\phi(i,k)(\xi - \mathbf{w}_i), \quad \text{for} \quad i = 1, \ldots, m.$$

step 4: Stop if the maximum number of iterations has been reached; otherwise modify η and ϕ as scheduled and continue with step 1.

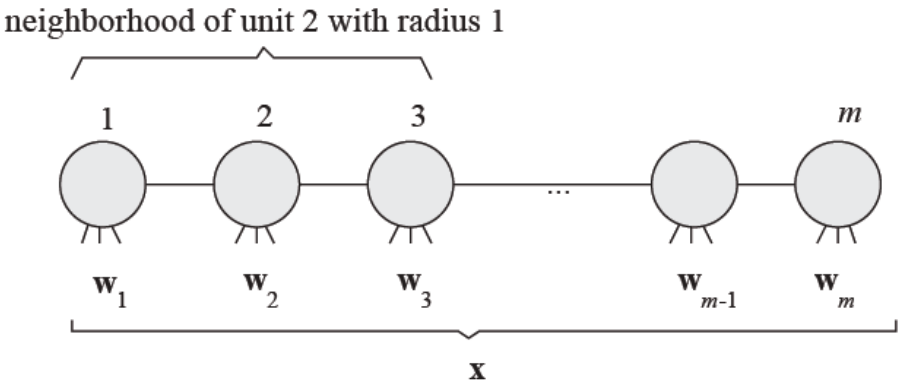
Learning attracts the weights to the inputs

The modifications of the weight vectors (step 3) attracts them in the direction of the input ξ. By repeating this simple process several times, we expect to arrive at a uniform distribution of weight vectors in input space (if the inputs have also been uniformly selected). The radius of the neighborhood is reduced according to a previous plan, which we call a *schedule*. The effect is that each time a unit is updated, neighboring units are also updated. If the weight vector of a unit is attracted to a region in input space, the neighbors are also attracted, although to a lesser degree. During the learning process both the size of the neighborhood and the value of ϕ fall gradually, so that the influence of each unit upon its neighbors is reduced. The learning constant controls the magnitude of the weight updates and is also reduced gradually. The net effect of the selected schedule is to produce larger corrections at the beginning of training than at the end.

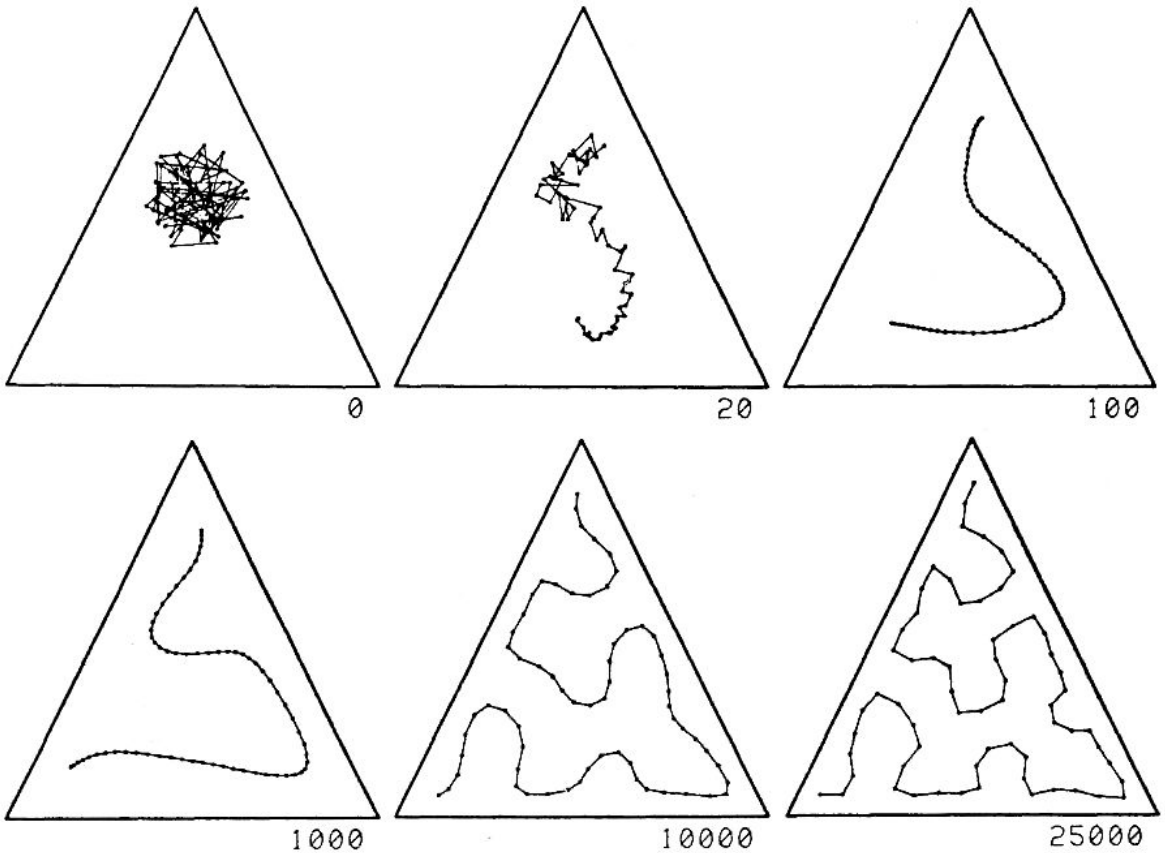
Figure 15.5 shows the results of an experiment with a one-dimensional Kohonen network. Each unit is represented by a dot. The input domain is a triangle. At the end of the learning process the weight vectors reach a distribution which transforms each unit into a "representative" of a small region of input space. The unit in the lower corner, for example, is the one

Example: 1D chain of neurons

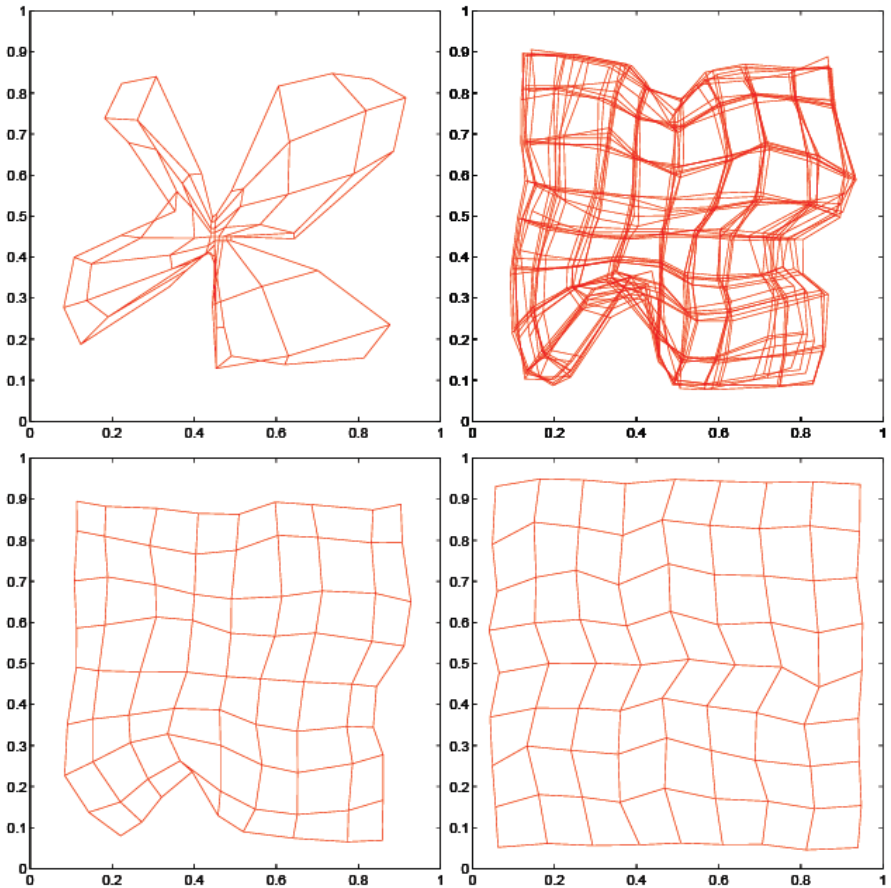
Consider the problem of charting an n-dimensional space using a one-dimensional chain of Kohonen units. The units are all arranged in sequence and are numbered from 1 to m (Figure 15.4). Each unit becomes the n-dimensional input $\mathbf{x}$ and computes the corresponding excitation. The n-dimensional weight vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$ are used for the computation. The objective of the charting process is that each unit learns to specialize on different regions of input space. When an input from such a region is fed into the network, the corresponding unit should compute the maximum excitation. Kohonen's learning algorithm is used to guarantee that this effect is achieved.



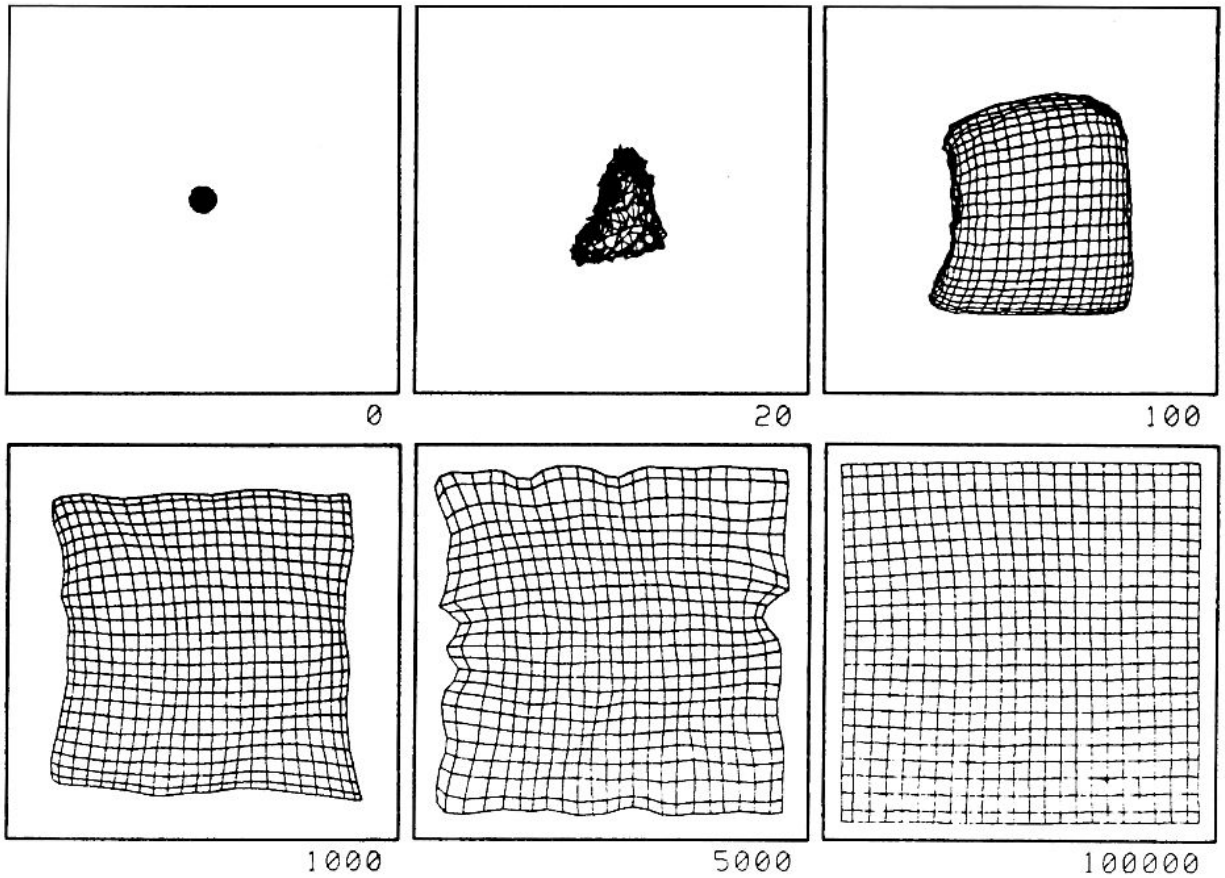
Learning a triangle with a 1D chain



Learning a square with a 2D network



Time evolution



Mapping square and triangular input-space to 2D

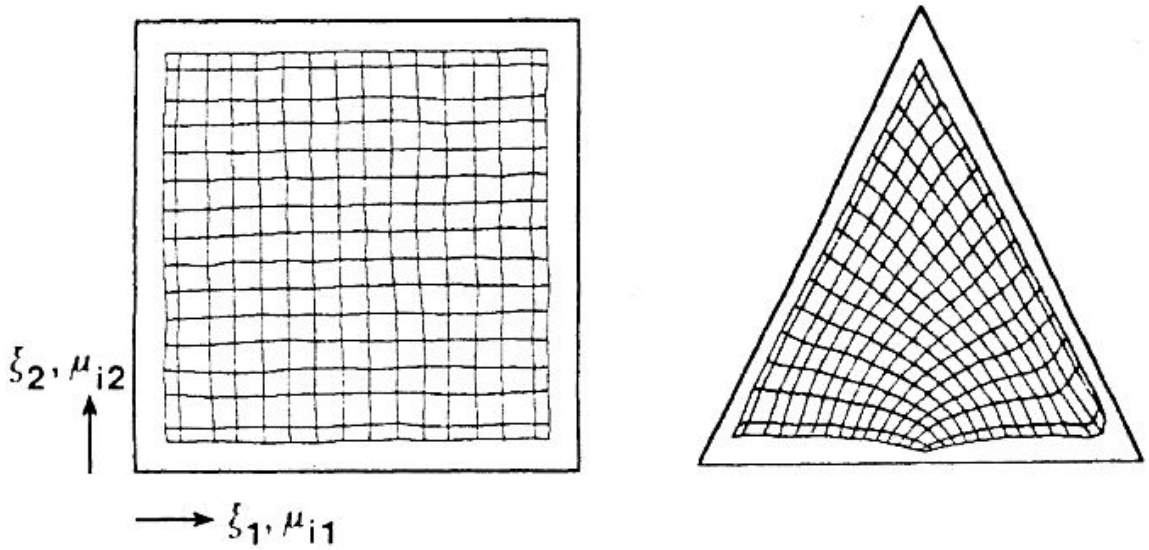
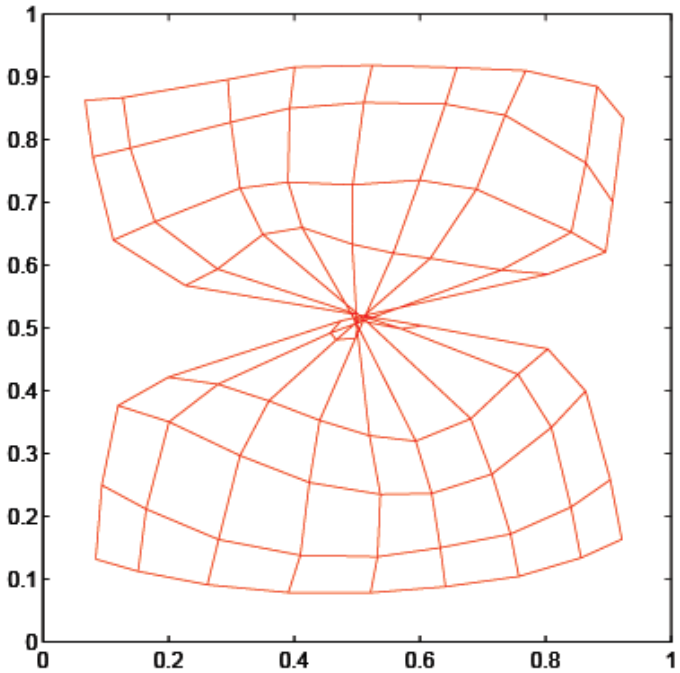
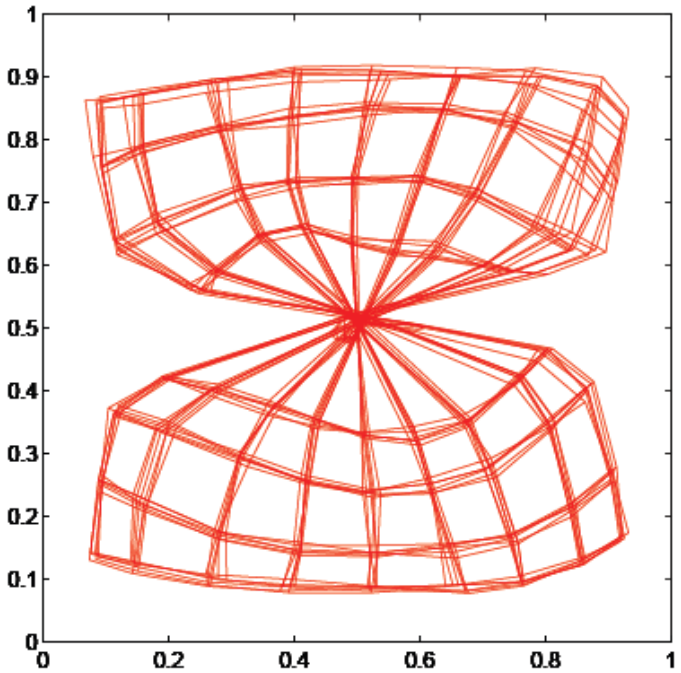


Fig. 3.5. Two-dimensional distributions of input vectors (framed areas), and the networks of reference vectors approximating them

No general convergence proof...





Mapping irregular input-spaces

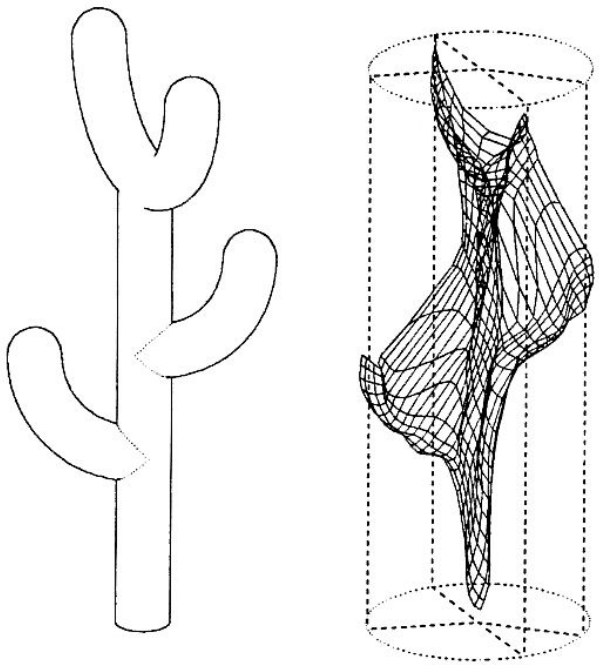


Fig. 3.9. SOM for a structured distribution of $p(x)$. For clarity, the three-dimensional $p(x)$, having uniform density value inside the "cactus" shape and zero outside it, is shown on the left, whereas the "net" of reference vectors is displayed on the right in a similar coordinate system

(Kohonen 2001)

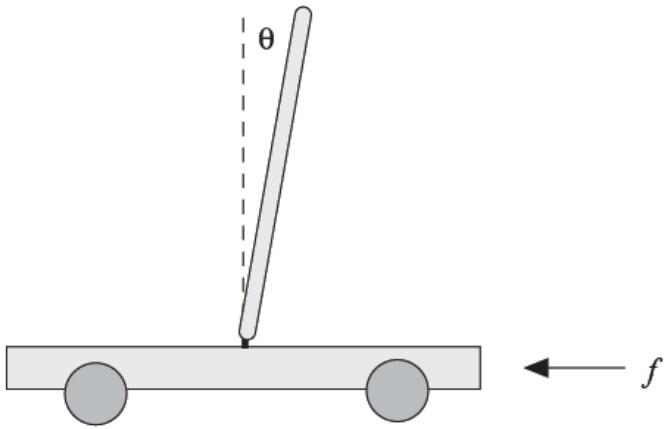
Function approximation

- learn a function $f(x, y)$ over a given domain?

- set $P = \{x, y, f(x, y) | x \in [0, 1]\}$ is a surface in 3D space
- use P as the input data for the SOM learning
- adapt a planar grid to the surface

- lookup: find the neuron (i, j) with minimal distance between the given input (x, y) and its weights (w_1, w_2).
- network output is the weight w_3 of the selected neuron

Function approximation: pole-balancing



- try to balance the pole by moving the cart
- θ the current angle between pole and the vertical
- force needed to keep balance is $f(\theta) = \alpha \sin(\theta) + \beta d\theta/dt$
- constants α and β (pole length and weight)

Function approximation: learned map for $f(x, y)$

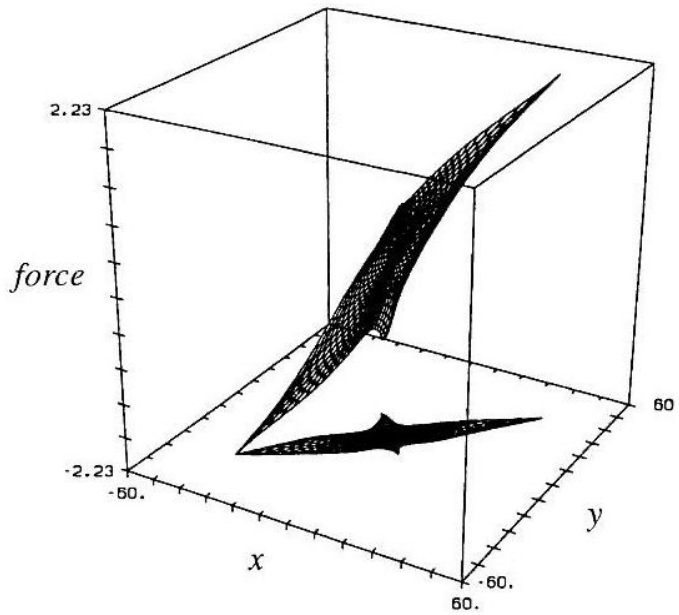


Fig. 15.17. Control surface of the balancing pole [Ritter et al. 1990]

- network to approximate $f(x, y) = 5\sin(x) + y$
- note: map can be updated (automatically) with feedback from the system itself.

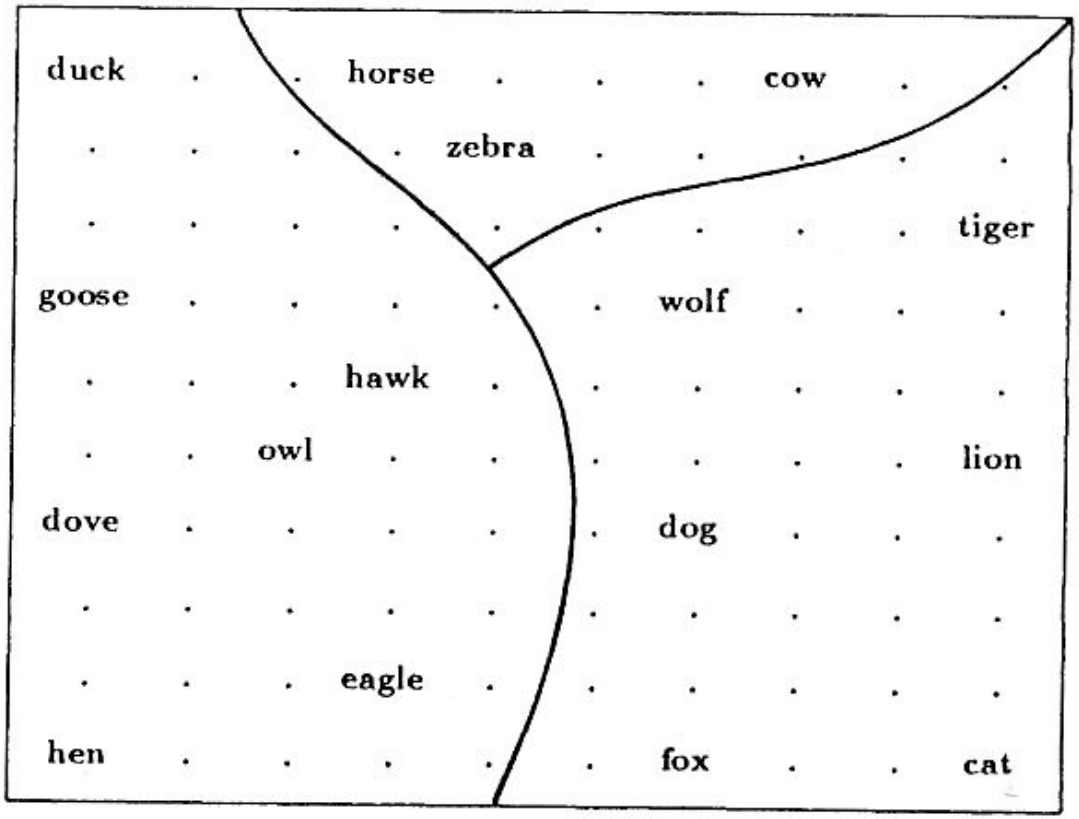
Some classification examples

- animals
- finnish speech samples
- world-bank poverty statistics

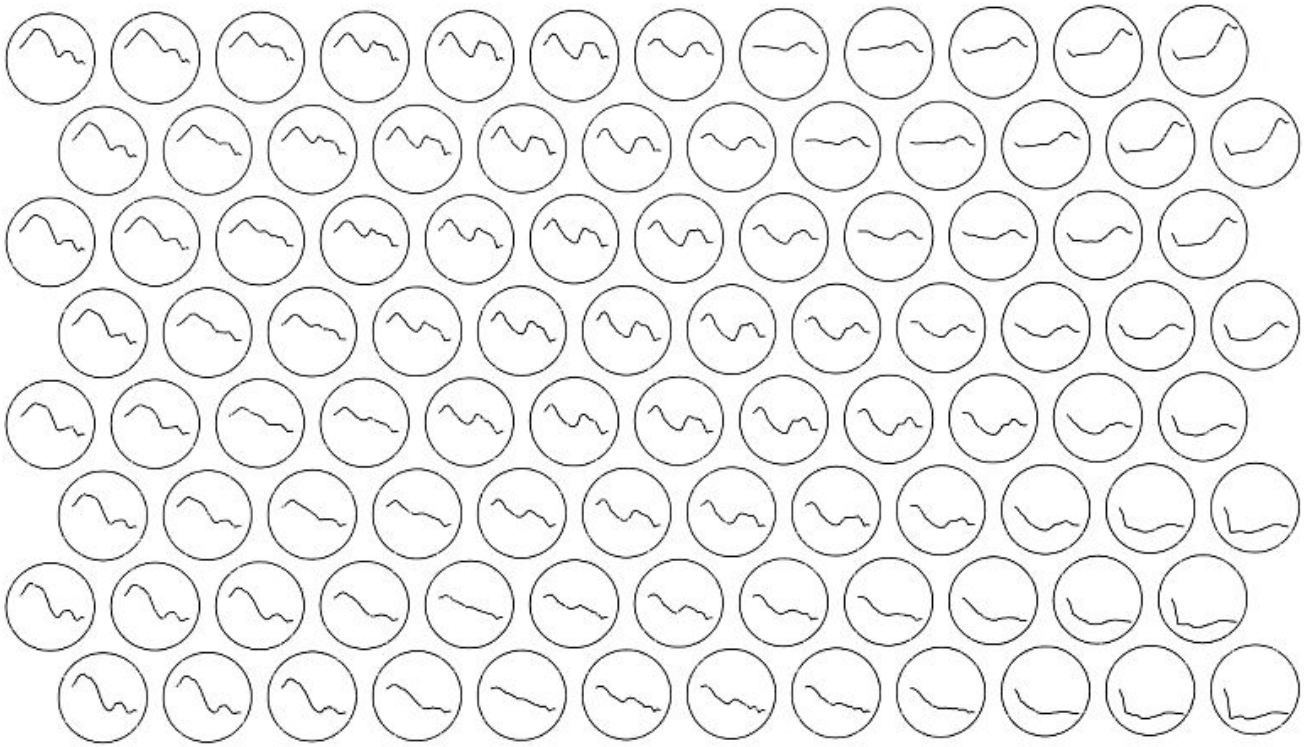
Animals: input data

		dove	hen	duck	goose	owl	hawk	eagle	fox	dog	wolf	cat	tiger	lion	horse	zebra	cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Animals: map after learning



Finnish speech samples



- note: similar waveforms are kept neighbors

World-bank poverty statistics

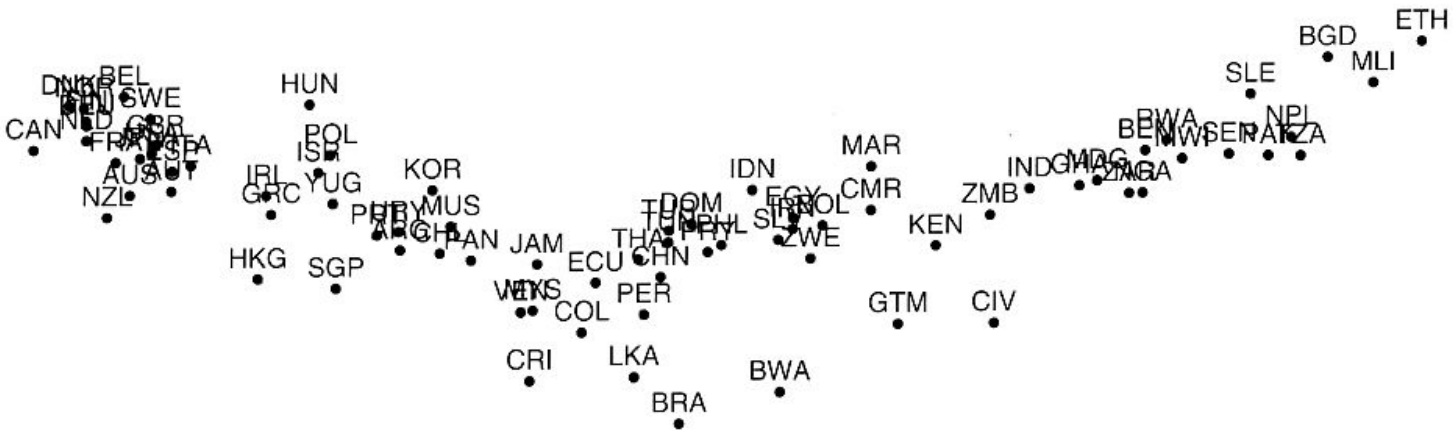


Fig. 1.5. The data set used in this illustration consisted of statistical indicators of 77 countries picked up from the World Development Report published by the World Bank [1.26]. Each component of the 39-dimensional data vectors describes a different aspect of the welfare and poverty of one country. Missing data values were neglected when computing the principal components, and zeroed when forming the projections. The three-letter codes for the countries, abbreviations of their names, may be self-explanatory. The data set was projected linearly as points onto the two-dimensional linear subspace obtained with PCA [1.27]

Poverty map: cartesian topology

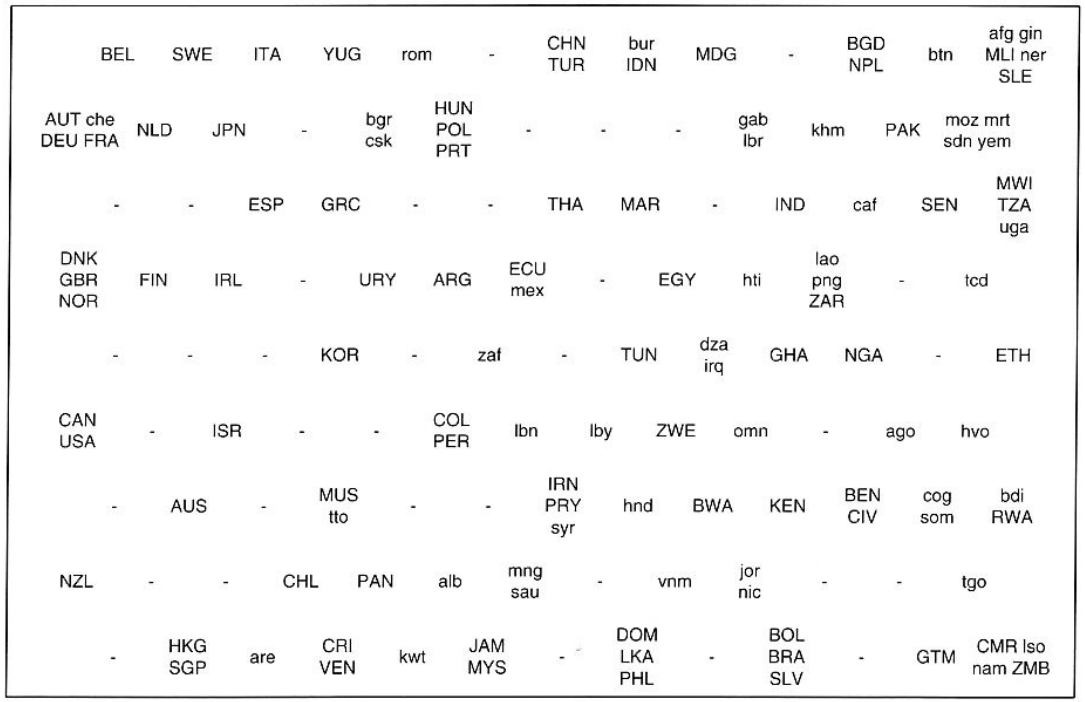
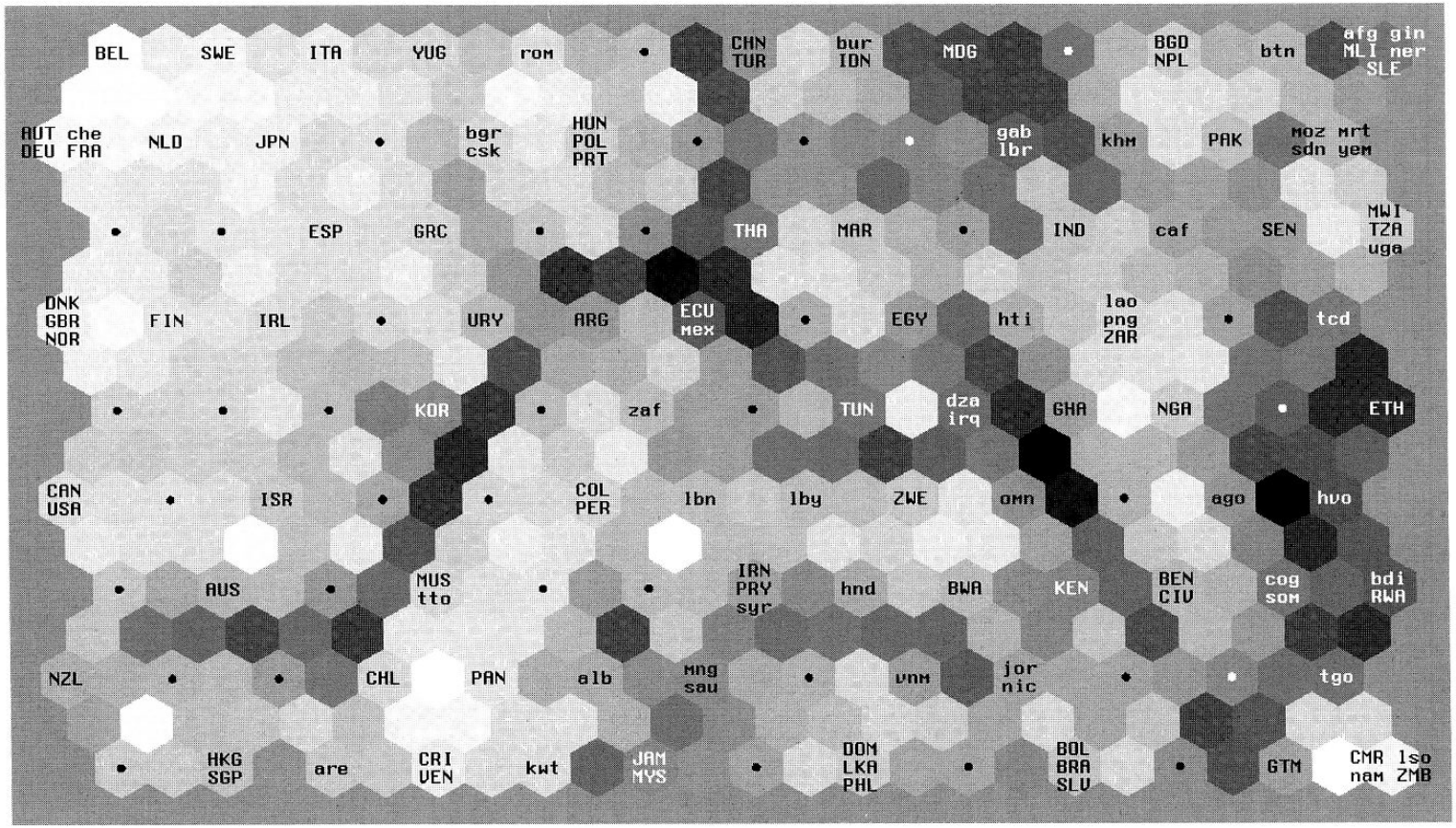
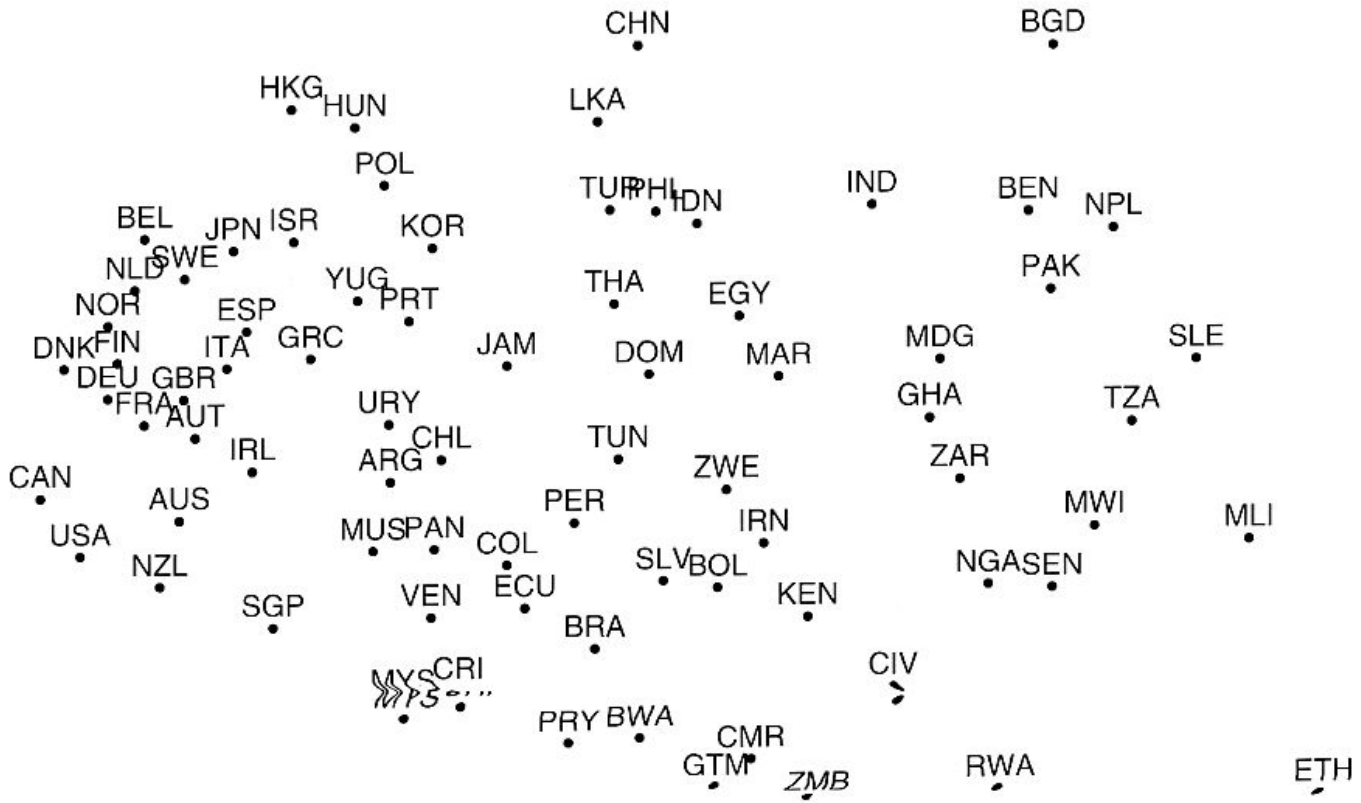


Fig. 3.30. "Poverty map" of 126 countries of the world. The symbols written in capital letters correspond to countries used in the formation of the map; the rest of the symbols (in lower case) signify countries for which more than 11 attribute values were missing, and which were mapped to this SOM after the learning phase

Poverty map: hexagonal topology



Poverty map: traditional Sammon's projection



SOM: summary

- a third type of neural network

- map high-dimensional input-space to output-space
- based on distance-measure in input-space
- and neighborhood functions

- unsupervised learning and self-organization
- learning rearranges the neurons
- useful for visualization and data-analysis

Digression: hardware for neural networks

- unmatched performance of the human/animal brain
- parallel processing of $10^6 \ldots 10^{11}$ neurons
- roughly $10^3 \ldots 10^5$ synapses per neuron
- neuron spiking roughly 1 msec.
- reaction-times (humans) about 0.1 seconds
- basic synapse operation is weighted-sum (multiply-accumulate)

- about $10^9 \ldots 10^{19}$ MAC operations/second
- at roughly 25 Watt
- compare: Core i7 3 GHz: about 10^{10} MAC/sec, 200 Watt

Hardware for neural networks?

- human/animal brain a highly optimized structure
- neurons on micrometer scale, axons up to 10 meters
- synapses on roughly nanometer scale
- fully 3D-interconnection (white matter)
- partly pre-configured, mostly self-organized
- very energy efficient
- very fault-tolerant, immune to noise

- any chance to mimic this with electronics / optics?
- currently: use standard PCs or standard parallel computers
- e.g., IBM blue-gene system for neural-network simulation

Neural network performance? associative memory

- assume Hopfield-network and un-correlated patterns
- storage-capacity $\alpha = 0.14N$ patterns of N bit (Hebb rule)
- optimal storage capacity $\alpha = 2N$ patterns
- useful range for good association is up to $\alpha \approx 0.5N$ patterns
- binary-couplings network up to $\alpha \leq 0.4N$ max
- fully parallel processing, fault-tolerant

- standard algorithm needs one distance-calculation per pattern
- distance-calculations in parallel, plus decision tree
- $O(N^2)$ operations

Hardware for neural networks

- active research topic
- initial interest triggered by 1982/1984 Hopfield papers
- and 1986 PDP/backpropagation papers

- concentrate on accelerating MAC operations
- parallel processing and interconnection topologies
- different base technologies:
  - digital electronics / VLSI
  - analog electronics / VLSI
  - optical processing
  - biochemistry / nanomaterials
- lots of interesting architectures
- so far, no mainstream applications

Hardware for neural networks: digital VLSI

- concentrating on acceleration of MAC operations
- relying on hardware multipliers and adders
- large VLSI chips incorporating multiple (pipelined) MAC units

- dozens of proposals during the 1980's
- systems also integrating learning rules in hardware
- obsolete since introduction of MAC/DSP hardware into mainstream processors

- special architectures (e.g. binary couplings) still attractive
- but high one-up engineering effort for design and test
- parallel algorithms often only marginally better than highly optimized software on standard CPUs

Hardware for neural networks: analog VLSI

- mimic neurons or synapses
- with highly optimized analog electronic circuits

- analog multipliers, adders, comparators
- imprecise, but very small and extremely fast
- mimic (3D) neuron interconnection structures on (2D) VLSI

- several demo circuits and architectures
- very good performance
- learning very difficult to implement
- usually, hard-coded interconnection patterns
- hard-coded to one specific algorithm/learning-rule

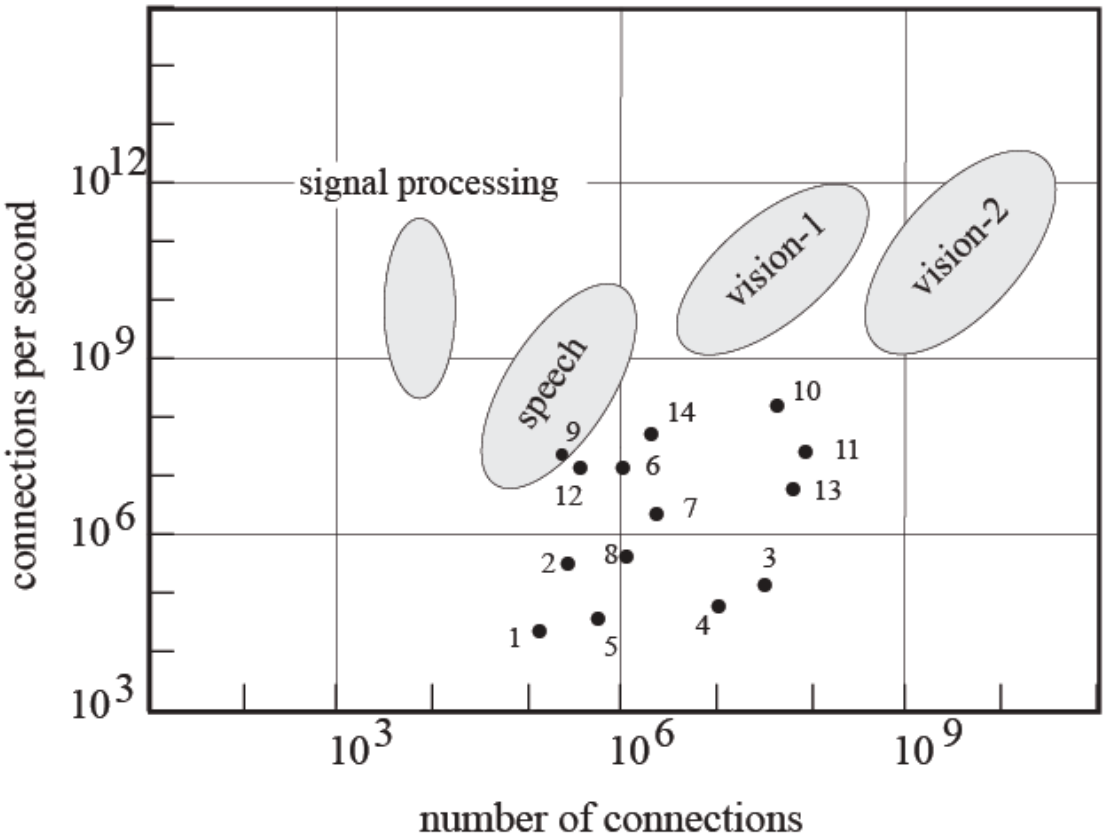
Hardware for neural networks: optical

- implement neurons with special (non-linear) optical components
- arrange neurons in 2D-grid across optical axis
- use filters/holograms to implement the couplings/weight-matrix

- in principle, millions of parallel neurons on one optical axis
- enormous theoretical performance
- successful demonstrations of simple algorithms

- currently, hard-coded coupling matrices (filters, holograms)
- no ideas on how to implement/realize learning/adaption
- feedback and complex algorithms impossible
- no fault-tolerance / self-organization

Hardware for neural networks



1 : PC/AT
2 : SUN 3
3 : VAX
4 : Symbolics
5 : ANZA
6 : Delta 1
7 : Transputer
8 : Mark III, V
9 : Odyssey
10: MX-1/16
11: CM-2 (64K)
12: Warp (10)
13: Butterfly (64)
14: CRAY XMP 1-2

Neural networks: conclusion

- biological motivation
- Huxley/Hodgkin neuron model
- basic neuroscience of the human brain

- McCulloch-Pitts neuron
- Perceptron and classification
- MLP with backpropagation learning

- Hopfield-model and associative memory
- Kohonen-network and self-organizing maps

- hardware acceleration