9.4 Summary of the SOM Algorithm

- The essential ingredients of Kohonen's SOM algorithm are:
  - A continuous input space of activation patterns (data).
  - A topology of the network defined by a lattice of neurons. This in turn defines a discrete output space.
  - A time-varying neighborhood function $h_{j,i(\mathbf{x})}(n)$. This is defined around a winning neuron $i(\mathbf{x})$.
  - A learning-rate parameter $\eta(n)$. This starts from an initial value η_0 and then decreases, but never goes to zero.
- Instructions on how to choose the learning rate and the neighborhood during learning are given in Section 9.3.
- In particular, $\eta(n)$ should be kept at a small value (0.01 or less) for typically thousands of iterations.
- This provides a good statistical accuracy.

- The neighborhood function should contain only the nearest neighbors of the winning neuron during the convergence phase.
- After initialization, three basic steps are repeated in the SOM algorithm until it converges.
- These are: sampling, similarity matching, and updating.

- **Summary of the SOM algorithm:**

  1. *Initialization.* Choose randomly the initial weight vectors $\mathbf{w}_j(0)$, $j = 1, 2, \ldots, l$, of the l neurons in the lattice.

     - Alternatively, the weight vectors may be chosen randomly from the available input (data) vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

  2. *Sampling.* Take a sample vector $\mathbf{x}(n)$ from the input space for the iteration n.

  3. *Similarity matching.* Let $i(\mathbf{x})$ denote the index of best matching (winning) neuron for the sample vector $\mathbf{x}$.

     - At iteration n, $i(\mathbf{x})$ is found from the minimum Euclidean distance criterion

$$i(\mathbf{x}) = \text{ arg min } \| \mathbf{x}(n) - \mathbf{w}_j \|, \;\; j = 1, 2, \ldots, l$$

4. *Updating.* Update the weight vectors of all neurons using the rule

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)[\mathbf{x}(n) - \mathbf{w}_j(n)],$$

   - Both the learning parameter $\eta(n)$ and the neighborhood function $h_{j,i(\mathbf{x})}(n)$ are varied during learning.
   - This is done for achieving best results.

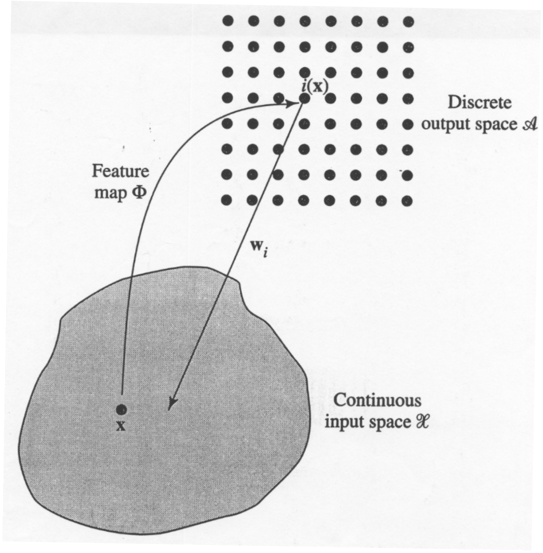
5. *Continuation.* Continue with step 2 until the feature map has converged.

- Notice the similarities (and some differences) with the k-means clustering algorithm (Section 5.13, part 2).

9.5 Properties of the Feature Map

- Assume now that the SOM algorithm has converged.
- The *feature map* computed by SOM describes important statistical properties of the input space (data).
- Let $\mathcal{H}$ denote a *spatially continuous input (data) space*.
- Let $\mathcal{A}$ denote a *spatially discrete output space*.
- The neurons of the output space are arranged in lattice form.
- Let Φ denote a nonlinear transformation called a feature map.
- Φ maps the input space $\mathcal{H}$ onto the output space $\mathcal{A}$: $\Phi : \mathcal{H} \rightarrow \mathcal{A}$.
- This is an abstract representation for defining the location of the winning neuron $i(\mathbf{x}) \in \mathcal{A}$.

- The weight vector $\mathbf{w}_i$ can be viewed as a pointer into the input space.
- These pointers constitute a kind of inverse mapping for the feature map;


$i(\mathbf{x})$
Discrete output space $\mathscr{A}$
Feature map Φ
$\mathbf{w}_i$
Continuous input space $\mathscr{X}$
$\mathbf{x}$

- The feature map Φ has some important properties.

Property 1: Approximation of the Input Space

- *The feature map Φ provides a good approximation to the input (data) space $\mathcal{H}$.*

- The feature map is represented by the weight vectors $\mathbf{w}_i$ in the input space $\mathcal{H}$.

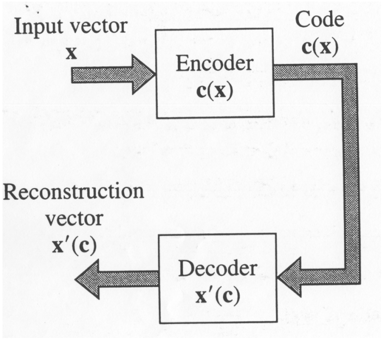
- The basic aim of the SOM algorithm: store a large set of input vectors $\mathbf{x} \in \mathcal{H}$.

- This is done by finding a smaller set of prototypes $\mathbf{w}_j \in \mathcal{H}$, providing a good approximation to the original input space $\mathcal{H}$.

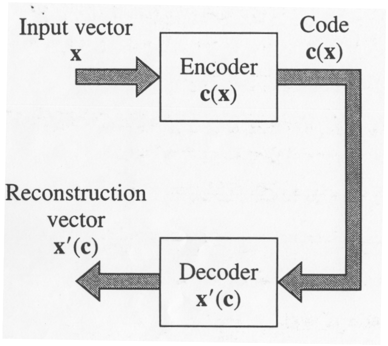
- Theoretical basis: *vector quantization theory*.

- In vector quantization, one tries to reduce the dimensionality of the data (compress the data) in an optimal way.

- In the following, we discuss the underlying theory.



- Simple encoder-decoder model.
- There $\mathbf{c}(\mathbf{x})$ is the *encoder* of the input vector $\mathbf{x}$.
- $\mathbf{x}'(\mathbf{c})$ is the *decoder* of $\mathbf{c}(\mathbf{x})$.



- Thus $\mathbf{c}(\mathbf{x})$ is the data compressing mapping that forms the code.
- The inverse mapping $\mathbf{x}'(\mathbf{c})$ approximates the data vector $\mathbf{x}$ from its coded version $\mathbf{c}(\mathbf{x})$.
- Both these mappings are generally nonlinear.
- Because of data compression, $\mathbf{x}'(\mathbf{c})$ is an approximation of the input vector $\mathbf{x}$.

- The randomly chosen input vectors $\mathbf{x}$ have a common probability density $f_{\mathbf{x}}(\mathbf{x})$.

- The optimum encoding-decoding scheme is determined by minimizing the *expected distortion*

$$D = \frac{1}{2}\mathsf{E}[d(\mathbf{x}, \mathbf{x}')] = \frac{1}{2}\int_{-\infty}^{\infty} d(\mathbf{x}, \mathbf{x}') f_{\mathbf{x}}(\mathbf{x}) d\mathbf{x}$$

- Here $d(\mathbf{x}, \mathbf{x}')$ is a *distortion measure*.

- A popular distortion measure is the squared Euclidean distance

$$d(\mathbf{x}, \mathbf{x}') = \parallel \mathbf{x} - \mathbf{x}' \parallel^2$$

- This leads to the standard mean-square distortion error

$$D = \frac{1}{2}\mathsf{E}[\parallel \mathbf{x} - \mathbf{x}' \parallel^2] = \frac{1}{2}\int_{-\infty}^{\infty} \parallel \mathbf{x} - \mathbf{x}' \parallel^2 f_{\mathbf{x}}(\mathbf{x}) d\mathbf{x}$$

- The *generalized Lloyd algorithm* contains the necessary conditions for minimizing the mean-squared distortion.

- These conditions are:

    1. Given the input vector $\mathbf{x}$, choose the code $\mathbf{c} = \mathbf{c}(\mathbf{x})$ to minimize the squared error distortion $\| \mathbf{x} - \mathbf{x}'(\mathbf{c}) \|^2$.
    2. Given the code $\mathbf{c}$, compute the reconstruction vector $\mathbf{x}' = \mathbf{x}'(\mathbf{c})$ as the centroid of those input vectors $\mathbf{x}$ that satisfy condition 1.

- Condition 1 is recognized as a *nearest-neighbor* encoding rule.

- The generalized Lloyd algorithm operates in a batch training mode.

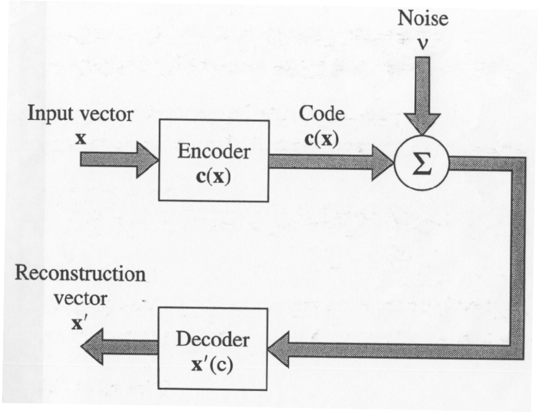
- The generalized Lloyd algorithm has also been called K-means algorithm.

- The algorithm alternately optimizes the encoder $\mathbf{c}(\mathbf{x})$ based on condition 1, and then the decoder $\mathbf{x}'(\mathbf{c})$ to satisfy condition 2.

- The algorithm terminates when a minimum of the expected distortion D is reached.

- For avoiding suboptimal local minimum solutions, the algorithm can be run several times with different initial code vectors.

- The algorithm is closely related to the SOM algorithm (Luttrell, 1989).
- This can be shown by considering a more general coding scheme shown below



- There a signal-independent *noise* process $\mathbf{v}$ has been added to the code $\mathbf{c}(\mathbf{x})$.
- $\mathbf{v}$ represents the distortion effect of a noisy communication channel over which the coded vectors are (possibly) transmitted.

- Let $\pi(\mathbf{v})$ be the probability density function (pdf) of the additive noise $\mathbf{v}$.

- For the noisy coding system, the modified expected distortion D_1 should be used.

$$D_1 = \frac{1}{2} \int_{-\infty}^{\infty} d\mathbf{x} f_{\mathbf{x}}(\mathbf{x}) \int_{-\infty}^{\infty} d\mathbf{v} \pi(\mathbf{v}) \parallel \mathbf{x} - \mathbf{x}'(\mathbf{c}(\mathbf{x}) + \mathbf{v}) \parallel^2$$

- Minimizing D_1 (instead of D) leads to the following modified conditions for the generalized Lloyd algorithm:

  1. Given the input vector $\mathbf{x}$, choose the code $\mathbf{c} = \mathbf{c}(\mathbf{x})$ to minimize the distortion measure

  $$D_2 = \int_{-\infty}^{\infty} \parallel \mathbf{x} - \mathbf{x}'[\mathbf{c}(\mathbf{x}) + \mathbf{v}] \parallel^2 \pi(\mathbf{v}) d\mathbf{v}$$

  2. Given the code $\mathbf{c}$, compute the reconstruction vector $\mathbf{x}' = \mathbf{x}'(\mathbf{c})$ to satisfy the condition

  $$\mathbf{x}'(\mathbf{c}) = \frac{M_1}{M_0} = \frac{\mathsf{E}[\mathbf{x}\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))]}{\mathsf{E}[\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))]} \qquad (1)$$

where the centroid and the normalization factor are respectively

$$M_1 = \int_{-\infty}^{\infty} \mathbf{x}\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))f_{\mathbf{x}}(\mathbf{x})d\mathbf{x},$$

$$M_0 = \int_{-\infty}^{\infty} \pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))f_{\mathbf{x}}(\mathbf{x})d\mathbf{x}.$$

- The derivations are sketched in Haykin's book on pp. 457-458, and are omitted in our basic course.

- If the probability density $\pi(\mathbf{v})$ of noise equals to a Dirac delta function $\delta(\mathbf{v})$, these conditions reduce to those obtained for the basic noiseless model.

- Assume that $\pi(\mathbf{v})$ is a smooth function of $\mathbf{v}$.

- Then condition 1 above can be approximated by that of the noiseless situation.

- This in turn reduces to a nearest neighbor encoding rule as before.

- Condition 2 for the noisy model can be realized using the stochastic descent learning algorithm

$$\Delta \mathbf{x}'(\mathbf{c}) = \eta \pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))[\mathbf{x} - \mathbf{x}'(\mathbf{c})] \tag{2}$$

- The update is applied to all $\mathbf{c}$ for which

$$\pi(\mathbf{c} - \mathbf{c}(\mathbf{x})) > 0$$

- For justifications, see again Haykin's book, p. 458.

- The update equation (2) above is equal to the SOM algorithm

- The correspondencies between SOM algorithm and the encoding-decoding model

Encoding-decoding model	SOM algorithm
Encoder $\mathbf{c}(\mathbf{x})$	Best-matching neuron $\mathbf{i}(\mathbf{x})$
Reconstruction vector $\mathbf{x}'(\mathbf{c})$	Synaptic weight vector $\mathbf{w_j}$
Probability density function $\pi(\mathbf{c} - \mathbf{c}(\mathbf{x}))$	Neighborhood function $h_{j,i(\mathbf{x})}$

- Hence, the generalized Lloyd algorithm for vector quantization is the batch training version of the SOM algorithm with zero neighborhood size $\pi(0) = 1$.

- Important points from the above discussion:

  - The SOM algorithm is a vector quantization algorithm, which provides a good approximation to the input space $\mathcal{H}$.
  - SOM can be derived from vector quantization considerations, too.
  - The neighborhood function $h_{j,i(\mathbf{x})}$ in SOM has the form of a probability density function.
  - The Gaussian neighborhood function used in SOM corresponds to assuming the noise $\mathbf{v}$ in the previous model to be zero-mean and Gaussian.

- One can easily derive a *batch version of SOM* by rewriting Eq. (1) in discrete form.

- See Problem 9.5 for the definition of batch SOM.

- Batch SOM does not depend on the order of presentation of the input vectors.
- There is no need for a learning-rate schedule.
- Batch SOM still requires the use of a neighborhood function.

Property 2: Topological Ordering

- *The feature map Φ computed by the SOM algorithm is topologically ordered.*
- This means that the spatial location of a neuron in the lattice corresponds to a particular domain or feature of input patterns.
- The topological ordering property directly follows from the SOM update rule:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n)h_{j,i(\mathbf{x})}(n)[\mathbf{x}(n) - \mathbf{w}_j(n)]$$

- This rule forces the weight vector $\mathbf{w}_i$ of the winning neuron $i(\mathbf{x})$ to move toward the input vector $\mathbf{x}$.
- It also moves the weight vectors in the neighborhood of the winning neuron to the same direction.
- Thus the feature map can be understood as an elastic or virtual net that is fitted to the input data.

- The weight vectors approximate the input space (data), characterizing its important properties.
- The feature map Φ is usually displayed in the input space.
- The weight vectors are shown as dots, and the neighboring weight vectors are connected with lines.

Property 3: Density Matching

- *The feature map Φ reflects the statistics of the input distribution.*
- Regions in which the data are dense occupy a larger domain in the output space than sparsely populated regions.
- Thus regions where the probability of the data is high are mapped with a better resolution.
- Let $f_{\mathbf{x}}(\mathbf{x})$ denote the probability density of the input vectors $\mathbf{x}$.
- $f_{\mathbf{x}}(\mathbf{x})$ must satisfy the normalization condition

  $$\int_{-\infty}^{\infty} f_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} = 1$$

  for being a true probability density.
- Let $m(\mathbf{x})$ denote the map *magnification factor*.

- $m(\mathbf{x})$ is the number of neurons in a small volume $d\mathbf{x}$ of the input space $\mathcal{H}$.

- The magnification factor must satisfy the condition

$$\int_{-\infty}^{\infty} m(\mathbf{x}) d\mathbf{x} = l$$

  where l is the total number of neurons in the network.

- For the SOM algorithm to match the input density exactly, we require that

$$m(\mathbf{x}) \propto f_{\mathbf{x}}(\mathbf{x})$$

- In two-dimensional feature maps the magnification factor $m(\mathbf{x})$ is not a simple function of the probability density $f_{\mathbf{x}}(\mathbf{x})$ of the input data.

- For one-dimensional maps one can derive a relationship between $m(\mathbf{x})$ and $f_{\mathbf{x}}(\mathbf{x})$.

- Two different results are reported in the literature, depending on the encoding method:

1. *Minimum distortion encoding* for the noisy coding model:

$$m(\mathbf{x}) \propto f_{\mathbf{x}}^{1/3}(\mathbf{x})$$

   - The same result holds for standard vector quantization.

2. *Nearest-neighbor encoding:*

$$m(\mathbf{x}) \propto f_{\mathbf{x}}^{2/3}(\mathbf{x})$$

   - This holds for the standard SOM algorithm.

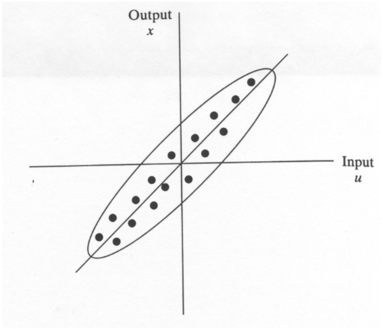
- Generally, SOM tends to underrepresent regions of high input density and overrepresent regions of low input density.

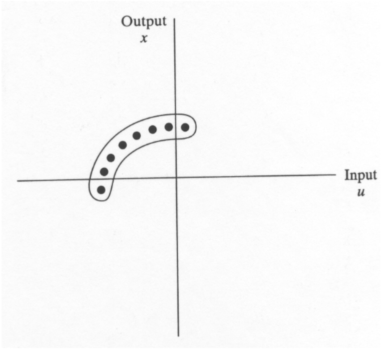
- There exist modifications of SOM which can faithfully represent the probability density $f_{\mathbf{x}}(\mathbf{x})$ of the input data.

- See Note 10 in Haykin's book on pp. 478-479.

Property 4: Feature Selection

- *SOM is able to select a set of best features for approximating nonlinearly distributed input data.*
- A culmination of Properties 1-3: Approximation of the input space, Topological ordering, Density matching

(a) 

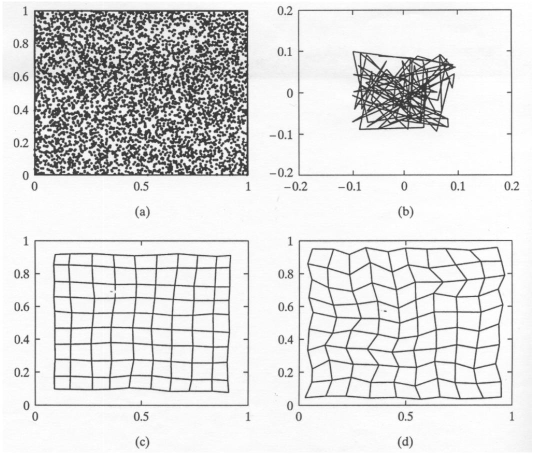
(b) 

- In Fig. (a), a linear mapping (line) is sufficient for describing well the data in one dimension.

- Such a mapping is obtained using standard principal components analysis (PCA).
- For the nonlinearly distributed data in Fig. (b), no linear mapping performs acceptably.
- However, SOM yields good results.
- In fact, SOM provides a nonlinear generalization of linear principal components analysis.

9.6 Computer Simulations

Two-Dimensional Lattice Driven by a Two-Dimensional Distribution



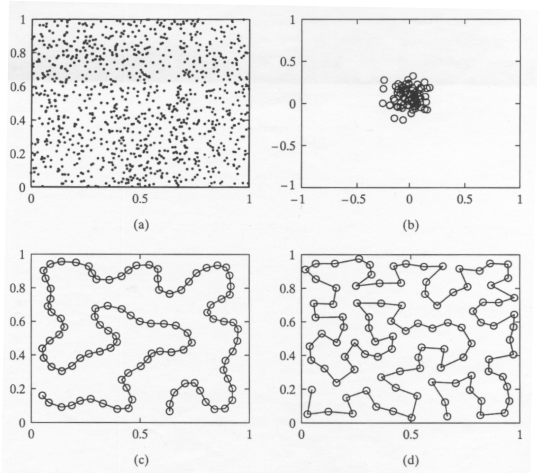
- 100 neurons arranged in a 10×10 lattice.
- Two-dimensional input vectors $\mathbf{x}$.
- Both components of $\mathbf{x}$ are uniformly distributed in the interval $(-1, +1)$ (see Fig. a).
- Randomly chosen initial weights shown in Fig. b.

- Figure c shows the values of the weights and the arising SOM network after the ordering phase.
- Fig. d depicts the final SOM map after the convergence phase.

- In the ordering phase, the map unfolds to form a mesh.
- During the convergence phase, SOM spreads out to fill the input space.

One-Dimensional Lattice Driven by a Two-Dimensional Distribution

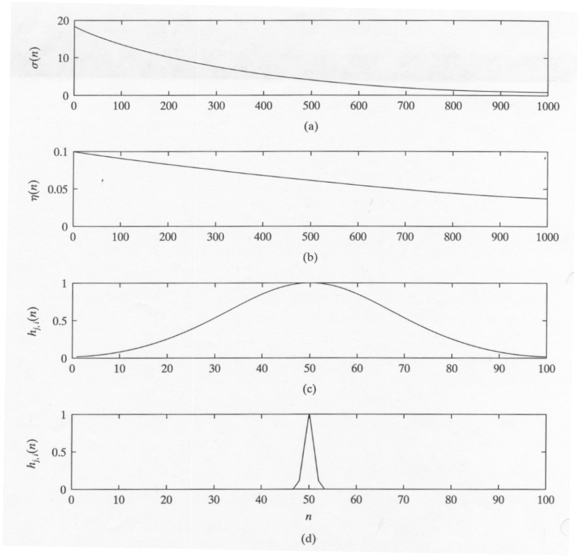
- Similar uniformly distributed data as previously.
- 100 neurons, but now in a one-dimensional lattice.



[illegible] (b) Initial condition of

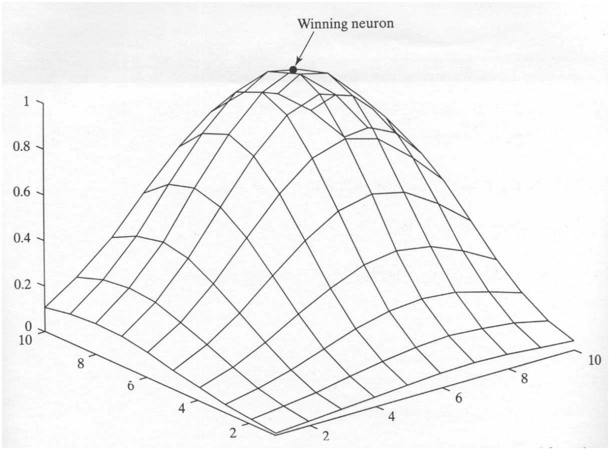
Parameter Specifications for the Simulations

- The neighborhood function and the learning-rate parameter



- Fig. a depicts the spread $\sigma(n)$ of the neighborhood.
- It starts with an initial value $\sigma_0 = 18$ and eventually shrinks to about 1.
- The learning parameter $\eta(n)$ starts from an initial value $\eta_0 = 0.1$ and then decreases to 0.037 (Fig. b).

- Figs. c and d show the shape of the Gaussian neighborhood function in the beginning and at the end of the ordering phase, respectively.
- During the convergence phase, both the learning parameter and the neighborhood continue to decrease close to zero.



- The initial value of the two-dimensional neighborhood used in the first experiment.