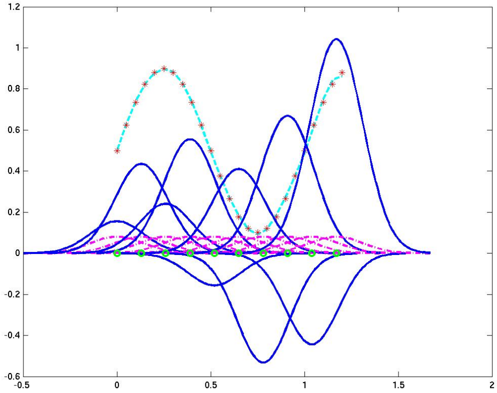
Function approximation using RBF network



$$F(\mathbf{x}_j) = \sum_{i=1}^{m_1} w_i \varphi(\| \mathbf{x}_j - \mathbf{t}_i \|)$$

$$j = 1 \ldots N,\ m_1 = 10,\ N = 25$$

- 10 basis functions and 25 data points.
- Basis function centers are plotted with circles and data points with asterisks.

5.13 Learning Strategies

- In RBF networks, learning proceeds differently for different layers.
- The linear output layer's weights are learned rapidly through a *linear* optimization strategy.
- The hidden layer's activation functions evolve slowly using some *non-linear* optimization strategy.
- The layers of a RBF network perform different tasks.
- It is reasonable to use different optimization techniques for the hidden and output layers.
- Learning strategies for the RBF networks differ in the method used for specifying the centers of the RBF network.
- In Haykin's book, four approaches for selecting the centers are represented. We discuss the first two of them.

1. Fixed Centers Selected at Random

- The simplest approach is to assume *fixed* radial-basis functions.
- The locations of the centers may be chosen *randomly* from the training data set.
- This is a sensible approach provided that the training data are representative for the problem.
- The radial-basis functions are typically chosen to be *isotropic* Gaussian functions:

$$G(\| \mathbf{x} - \mathbf{t}_i \|) = \exp\left(-\frac{m_1}{d_{max}^2} \| \mathbf{x} - \mathbf{t}_i \|^2\right)$$

  $i = 1, 2, \ldots, m_1$ where m_1 is the number of centers (basis functions).

- d_{max} is the maximum distance between the chosen centers.

- In effect, the standard deviation (width) of all the Gaussians is fixed at

$$\sigma = \frac{d_{max}}{(2m_1)^{1/2}}$$

- This choice ensures that the individual radial-basis functions are not too peaked or too flat.

- In this approach, only the linear weights of the output layer must be learned.

- A straightforward procedure for estimating the weights is to use the pseudoinverse method:

$$\mathbf{w} = \mathbf{G}^{+}\mathbf{d} = (\mathbf{G}^{T}\mathbf{G})^{-1}\mathbf{G}^{T}\mathbf{d}$$

- Here $\mathbf{G}^{+}$ is the *pseudoinverse* of the matrix $\mathbf{G}$.

- The element g_{ji} of the matrix $\mathbf{G}$ is defined by

$$g_{ji} = \exp\left(-\frac{m_1}{d^2} \parallel \mathbf{x}_j - \mathbf{t}_i \parallel^2\right)$$

  $i = 1, 2, \ldots, m_1,\ j = 1, 2, \ldots, N$, where $\mathbf{x}_j$ is the jth training vector.

- Pseudoinverses can be computed efficiently in a numerically robust way using the *singular-value decomposition*.

- Let $\mathbf{G}$ be a general real $N \times M$ matrix.

- The singular-value decomposition of $\mathbf{G}$ is defined by the expansion

$$\mathbf{U}^T\mathbf{G}\mathbf{V} = \mathbf{\Sigma}$$

$$\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_K), \quad K = \min(M, N)$$

  is a diagonal matrix containing the *singular values* of $\mathbf{G}$.

- The column vectors $\mathbf{u}_i$ of the orthogonal matrix

$$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N]$$

  are called the *left singular vectors* of $\mathbf{G}$.

- The column vectors $\mathbf{v}_i$ of the orthogonal matrix

$$\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M]$$

  are the *right singular vectors* of $\mathbf{G}$.

- Using singular-value decomposition theory, the $M \times N$ pseudoinverse of matrix $\mathbf{G}$ is defined by

$$\mathbf{G}^+ = \mathbf{V}\Sigma^+\mathbf{U}^T$$

- Here Σ^+ is itself an $N \times N$ diagonal matrix defined by

$$\Sigma^+ = \text{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_K}, 0, \ldots, 0\right)$$

- Efficient algorithms for computing pseudoinverses and pseudoinverse theory can be found in textbooks of linear algebra and numerical analysis.

- Random selection of centers is relatively insensitive to the use of regularization.

- Random selection of centers from a large training set is probably a kind of regularization method itself.

2. Self-Organized Selection of Centers

- The main problem with choosing fixed centers randomly: requires possibly a large training set for satisfactory performance.

- This limitation can be overcome by using a hybrid learning process consisting of two different stages:

  1. *Self-organized learning.* Appropriate locations of centers of the radial-basis functions are estimated in this stage.
  2. *Supervised learning.* The linear weights of the output layer are determined in this stage.

- It is preferable to learn these stages adaptively (iteratively).

- The self-organized learning stage is realized using some suitable *clustering* algorithm.

- This partitions the training data into homogenous groups.

- A basic clustering algorithm: *k-means clustering*.

- It places the centers of radial-basis functions in only those regions of the input space where significant amount of the data are present.

- Assume that m_1 is the number of radial-basis functions.

- Determination of a suitable value of m_1 may require experimentation.

- Let us denote the centers of the radial-basis functions at step n by $\mathbf{t}_1(n), \ldots, \mathbf{t}_{m_1}(n)$.

K-means clustering algorithm:

1. *Initialization.* Choose different random values $\mathbf{t}_k(0)$ for the initial centers.
2. *Sampling.* Take a sample vector $\mathbf{x}(n)$ from the input space for the iteration n.
3. *Similarity matching.* Let $k(\mathbf{x})$ denote the index of best matching (winning) center for the sample vector $\mathbf{x}$.

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

   $$k(\mathbf{x}) = \text{ arg min } \| \mathbf{x}(n) - \mathbf{t}_k(n) \|, \quad k = 1, 2, \ldots, m_1$$

4. *Updating.* Update the centers of the radial basis functions using the rule

   $$\begin{aligned} \mathbf{t}_k(n+1) &= \mathbf{t}_k(n) + \eta[\mathbf{x}(n) - \mathbf{t}_k(n)], \quad k = k(\mathbf{x}) \\ \mathbf{t}_k(n+1) &= \mathbf{t}_k(n), \text{ otherwise} \end{aligned}$$

5. *Continuation.* Increment n by 1 and continue the procedure from step 2 until convergence.

- The k-means clustering algorithm is a special case of the self-organizing map (SOM) to be discussed in Chapter 9.
- SOM or other more sophisticated versions of k-means clustering can also be used to determine the centers of the radial-basis functions.
- Assume now that the centers have been learned using some method.
- The weights of the output layer can be estimated for example using the simple adaptive LMS algorithm discussed in Chapter 3.
- The input vector to the LMS algorithm is the output vector of the hidden RBF layer.

5.14 Computer Experiment: Pattern Classification

- The classification problem is described in Section 4.8.
- Now the same problem is solved using RBF networks instead of MLP networks.
- Two overlapping Gaussian distributions corresponding to the classes $\mathcal{C}_1$ and $\mathcal{C}_2$.
- Regularized RBF networks based on strict interpolation are used for designing the classifier.
- The decision rule used earlier with MLP is used also here:
  Classify $\mathbf{x}$ *to the class* $\mathcal{C}_k$ *if*

$$F_k(\mathbf{x}) > F_j(\mathbf{x}) \text{ for all } j \neq k$$

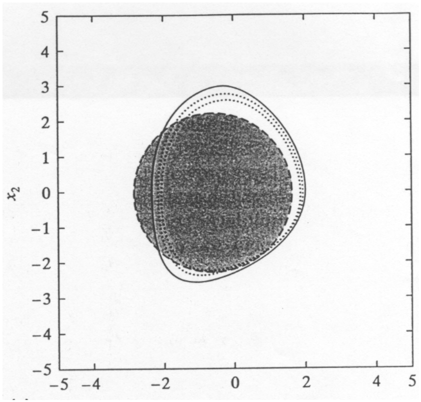
- Regularized RBF networks are able to estimate the optimal Bayesian classifier (posterior probabilities).
- Provided that binary-valued desired vectors are used; see Section 4.7.

- The weight vector $\mathbf{w}$ is computed for different values of the regularization parameter λ from the formula

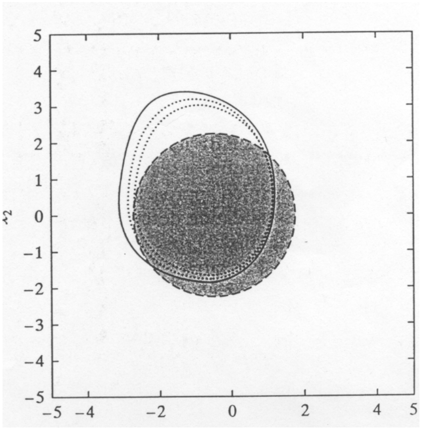
$$\mathbf{w} = (\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{d}$$

- The number of centers (basis functions) was either 20 or 100.
- 50 independent trials for each value of λ.
- Mean of probability of correct classification for varying regularization parameter

Centers	Regularization parameter λ					
m_1	0	0.1	1	10	100	1000
20	**57.49**	72.42	**74.42**	73.80	72.46	72.14
100	**50.58**	77.03	77.72	**77.87**	76.47	75.33



a



b

- Examples of best and worst performing networks are shown in Figures (a) and (b) for the case of 100 centers and $\lambda = 10$.

- **Conclusions on simulations:**

  1. Regularization improves dramatically the classification performance.
  2. The value of the regularization parameter does not affect much the performance if $\lambda \geq 0.1$.
  3. Increasing the number of centers (radial-basis functions) from 20 to 100 improves the performance by about 4.5%.

9. Self-Organizing Maps

9.1 Introduction

- Self-organizing maps are based on *competitive learning* discussed briefly in Section 2.5.
- Recall *winner-takes-all* principle: only one output neuron (winner of competition) is updated at a time.
- In a *self-organizing map (SOM)*, the neurons are placed at the nodes of a usually two-dimensional lattice.
- During the competitive learning process, the neurons become sensitive to different input features.
- Neurons spatially close to each other describe features relatively closer to each other.
- In effect, SOM forms a *nonlinear mapping* from the input space to the two-dimensional lattice.

- The map tries to describe the intrinsic properties of the data as well as possible.

- Self-organizing map is an *unsupervised learning method* in its basic form.

- The development of self-organizing maps was motivated by topological properties of the human brain.

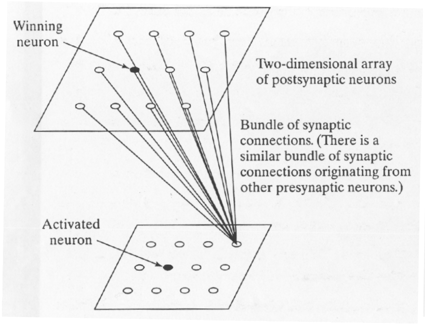
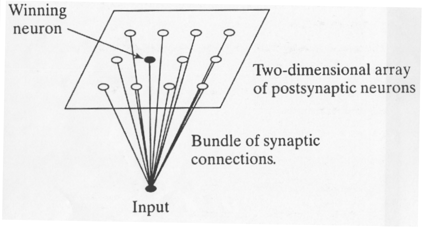
- They were developed at Helsinki University of Technology by Academian Teuvo Kohonen (1982).

- In the Laboratory of Computer and Information Science at HUT, both applications and theoretical properties of SOMs are still studied fairly extensively.

9.2 Two Basic Feature-Mapping Models

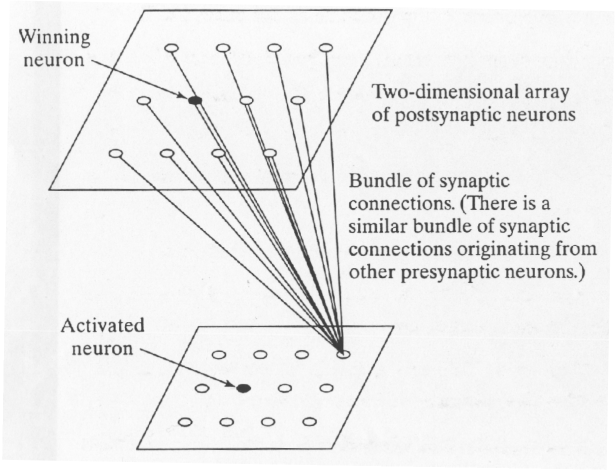
- Human brains are dominated by the cerebral cortex.
- It is probably the most complex known structure in the universe.
- The cerebral cortex forms a topographic mapping with the following properties:
  - At each stage of representation, each incoming piece of information is kept in its proper context.
  - Neurons dealing with closely related pieces of information are close together, having thus short synaptic connections.
- Our interest is to build artificial topographic maps.
- They learn through self-organization in a neurobiologically inspired manner.

- *Principle of topographic map formation* (Kohonen, 1990):
- *The spatial location of an output neuron in a topographic map corresponds to a particular domain or feature of the input data.*
- Based on this principle, two different *feature-mapping models* have been proposed.


Winning neuron
Two-dimensional array of postsynaptic neurons
Bundle of synaptic connections. (There is a similar bundle of synaptic connections originating from other presynaptic neurons.)
Activated neuron

Winning neuron
Two-dimensional array of postsynaptic neurons
Bundle of synaptic connections.
Input

- In both models, the output neurons are arranged in a two-dimensional lattice.
- Such a topology ensures that each neuron has a set of neighbors.

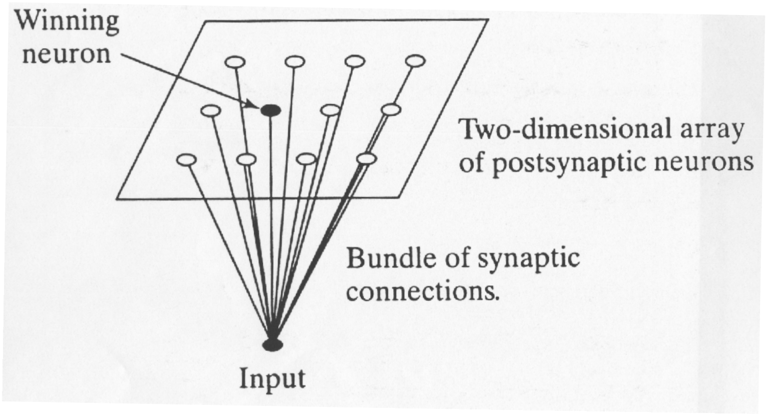
- The models differ in the specification of input patterns.


Winning neuron
Two-dimensional array of postsynaptic neurons
Bundle of synaptic connections. (There is a similar bundle of synaptic connections originating from other presynaptic neurons.)
Activated neuron

- The Willshaw-von der Malsburg model tries to explain some observed neurobiological details.

- There the input dimension is the same as the output dimension.

- However, this model is computationally not so successful.


Winning neuron
Two-dimensional array of postsynaptic neurons
Bundle of synaptic connections.
Input

- Kohonen's model does not explain neurobiological details.
- Anyway, it captures the essential features of computational maps in the brain.
- Kohonen's model is also computationally feasible.
- It is more general than the first model because of its ability to compress the input data.

- On the other hand, Kohonen's model belongs to the class of *vector-coding* algorithms.

- SOM optimally places a fixed number of vectors (code words) into a higher-dimensional space.

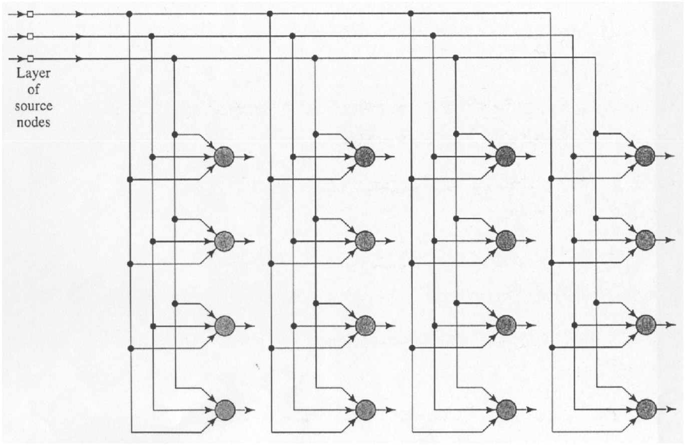
- Instead of self-organization, SOM can be derived using a vector quantization approach.

- This approach is motivated by communication-theoretic (data compression) considerations.

- The remainder of this chapter deals with Kohonen's self-organizing map.

9.3 Self-Organizing Map

- Principal goals of the self-organizing map (SOM):
- Transform data (input) vectors having an arbitrary dimension into a two-dimensional map usually.
- Perform the transform adaptively in a topologically ordered fashion.



- Two-dimensional lattice of neurons used commonly as the discrete map.
- Each neuron is connected to all the inputs.
- This network is a feedforward structure with a single two-dimensional computational layer.
- Sometimes it is sufficient or appropriate to use a one-dimensional SOM.
- All the neurons in the network should be exposed to a sufficient number of different input patterns.
- This ensures that the self-organizing process has time to mature properly.
- The learning algorithm for SOM starts by *initializing* the synaptic weights.
- Can be done by choosing small random values as the initial weights.

- After initialization, three essential processes are used for learning the self-organizing map.

    1. *Competition.* For each input vector, all the neurons compute their value of a discriminant function.

       - The neuron with largest value wins the competition.

    2. *Cooperation.* The spatial location of the winning neuron determines the neighborhood where weight vectors are updated.

    3. *Synaptic Adaptation.* The response of the winning neuron to a similar input pattern is increased by updating its weight vector suitably.

- In the following, we discuss these stages in more detail.

Competitive Process

- Assume that the input vectors $\mathbf{x} = [x_1, x_2, \ldots, x_m]^T$ are selected at random.
- Each neuron in the network has a synaptic weight vector

$$\mathbf{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T, \;\; j = 1, 2, \ldots, l$$

- The weight vectors have the same dimension m as the input vectors.
- The total number of neurons and weight vectors is l.
- Task: find the best match of the input vector $\mathbf{x}$ with the weight vectors $\mathbf{w}_j$.
- This can be done by computing the inner products $\mathbf{w}_j^T\mathbf{x}$, $j = 1, 2, \ldots, l$, and selecting the largest.
- Here the weight vectors $\mathbf{w}_j$ are assumed to have equal norms (lenghts).

- The best matching neuron with the index $i(\mathbf{x})$ defines the center of topological neighborhood of excited neurons.

- Maximization of the inner product $\mathbf{w}_j^T\mathbf{x}$ is equivalent to minimizing the Euclidean distance between the vectors $\mathbf{x}$ and $\mathbf{w}_j$.

- Thus the index of best matching neuron

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

- The neuron i above is called the best matching or winning neuron for the input vector $\mathbf{x}$.

- Depending on the application, the response of SOM can be either:
  - The index of the winning neuron (its position in the lattice);
  - or the weight vector closest to the input vector.

Cooperative Process

- Key question: how to define a neurobiologically correct topological neighborhood for the winning neuron?
- It is reasonable to make the neighborhood around the winning neuron i to decay smoothly with lateral distance.
- Let $h_{j,i}$ denote the *topological neighborhood* centered on the winning neuron i.
- The index j denotes a typical neuron in this neighborhood.
- Let $d_{j,i}$ denote the *lateral distance* between winning neuron i and excited neuron j.
- The neighborhood $h_{j,i}$ is assumed to be a unimodal function of the lateral distance $d_{j,i}$.

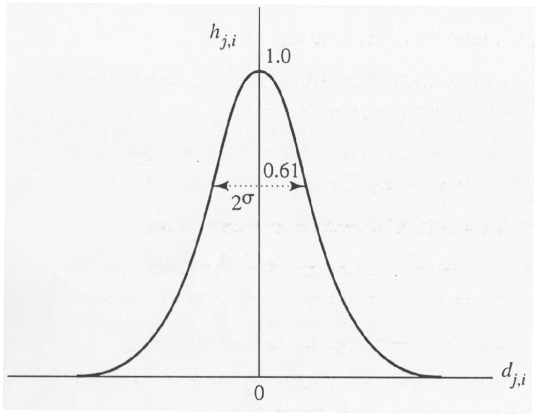
- It satisfies two distinct requirements:
  - The topological neighborhood $h_{j,i}$ is symmetric about its maximum point.

    This is the winning neuron with zero distance $d_{j,i} = 0$.
  - The amplitude of $h_{j,i}$ decreases monotonically with increasing distance $d_{j,i}$.

    $h_{j,i} \to 0$ when $d_{j,i} \to \infty$; this is a necessary condition for convergence.
- A typical choice of $h_{j,i}$ is the Gaussian function

$$h_{j,i(\mathbf{x})} = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right)$$

- Translation (and rotation) invariant.
- The "standard deviation" σ defines the effective width of the topological neighborhood



2^{σ} should be 2σ

- The spherical Gaussian neighborhood is biologically more appropriate than a rectangular neighborhood.
- It also makes the SOM algorithm converge faster.
- The neighborhood $h_{j,i}$ must depend on the distance of neurons in the output space rather than in the original input space.
- For one-dimensional map, $d_{j,i}$ is the integer $\mid j - i \mid$.

- Another unique feature of the SOM algorithm: the size of the topological neighborhood shrinks with time.

- This is realized by decreasing the width σ of the neighborhood $h_{j,i}$ with time.

- A popular choice is exponential decay with discrete time n:

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right), \;\; n = 0, 1, 2, \ldots$$

- Here σ_0 is the initial value of the width σ and τ_1 is a time constant.

- This yields the shrinking topological neighborhood function

$$h_{j,i(\mathbf{x})}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(n)}\right)$$

Adaptive Process

- Self-organization is achieved by adapting the weight vectors of the neurons suitably as a response to shown input vectors.
- Hebbian learning (Section 2.4) is useful for associative learning.

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n)$$

- However, the basic Hebbian rule alone is unsatisfactory for unsupervised learning or self-organization.
- Reason: all the synaptic weights are driven into saturation.
- This problem can be overcome by adding a *forgetting term* $g(y_j)\mathbf{w}_j$.
- Here $g(y_j)$ is some positive scalar function of the response y_j of the neuron j.

- The only requirement imposed on the function $g(y_j)$:
  The constant term in the Taylor series expansion of $g(y_j)$ is zero.
- This yields the condition

$$g(y_j) = 0 \text{ for } y_j = 0$$

- Then the change in the weight vector of neuron j in the lattice takes the form

$$\Delta \mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j)\mathbf{w}_j$$

- The first term $\eta y_j \mathbf{x}$ is the Hebbian term.
- η denotes the learning-rate parameter as usual.
- The second term $-g(y_j)\mathbf{w}_j$ is the forgetting term.
- The requirement for $g(y_j)$ can be satisfied by choosing

$$g(y_j) = \eta y_j$$

- Furthermore, the update rule can be simplified by setting

$$y_j = h_{j,i(\mathbf{x})}$$

- These choices yield the update rule

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

- In discrete-time formalism, the obtained update rule is

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

- This rule is applied to all neurons inside the topological neighborhood of winning neuron i.

- The adaptation rule moves the weight vector $\mathbf{w}_i$ of the winning neuron toward the input vector $\mathbf{x}$.

- During adaptation, the weight vectors tend to follow the distribution of the input vectors due to the neighborhood updating.

- Therefore, the SOM algorithm leads to a topological ordering of the feature map in the input space.

- This means that neurons that are adjacent in the lattice tend to have similar weight vectors.

- Also the learning-rate parameter $\eta(n)$ should be made time-varying.

- It should start at an initial value η_0, and then gradually decrease with increasing time n.

- A typical choice: exponential decay

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right), \;\; n = 0, 1, 2, \ldots$$

  where τ_2 is another time constant of the SOM algorithm.

Two Phases of the Adaptive Process: Ordering and Convergence

- The SOM algorithm starts from a complete disorder.
- If its parameters are chosen appropriately, it gradually leads to a nice organized representation of input vectors.
- The adaptation takes place in two phases.

  1. **Self-organizing or ordering phase.**

     - The topological ordering of weight vectors takes place here.

     - May take 1000 iterations or even more.

     - The learning parameter $\eta(n)$ should decrease slowly from about 0.1 to stay above 0.01 during this phase.

     - The neighborhood function $h_{j,i}(n)$ should initially contain almost all neurons, and then shrink slowly with time.

     - More detailed instructions are given in Haykin's book, pp. 452-453.

2. **Convergence phase.**

   - This phase is needed to fine tune the feature map.

   - The number of iterations here should be at least 500 times the number of neurons in the lattice.

   - The learning parameter $\eta(n)$ should be kept as a small constant (0.01 typically) for achieving a good statistical accuracy.

   - The neighborhood function should contain only the nearest neighbors of the winning neuron.