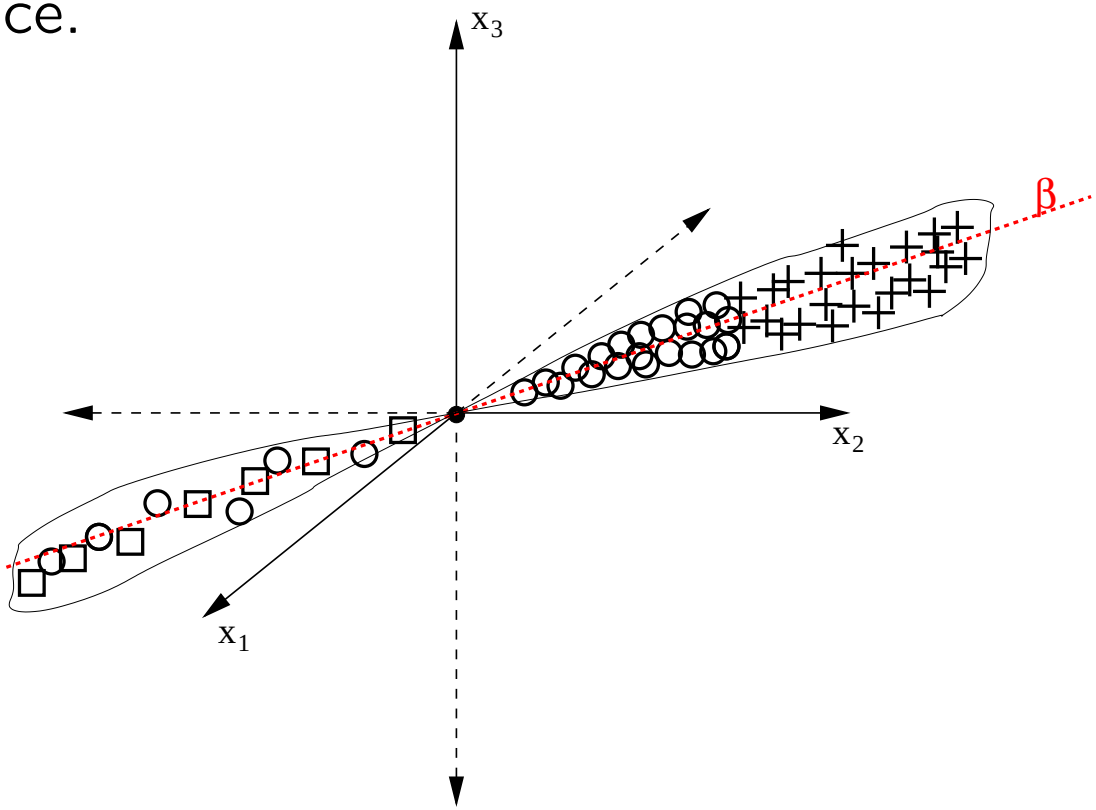
Intelligent Data Analysis

Topographic Maps of Vectorial Data

Peter Tiňo
School of Computer Science
University of Birmingham

Discovering low-dimensional spatial layout in higher dimensional spaces - 1-D/3-D example

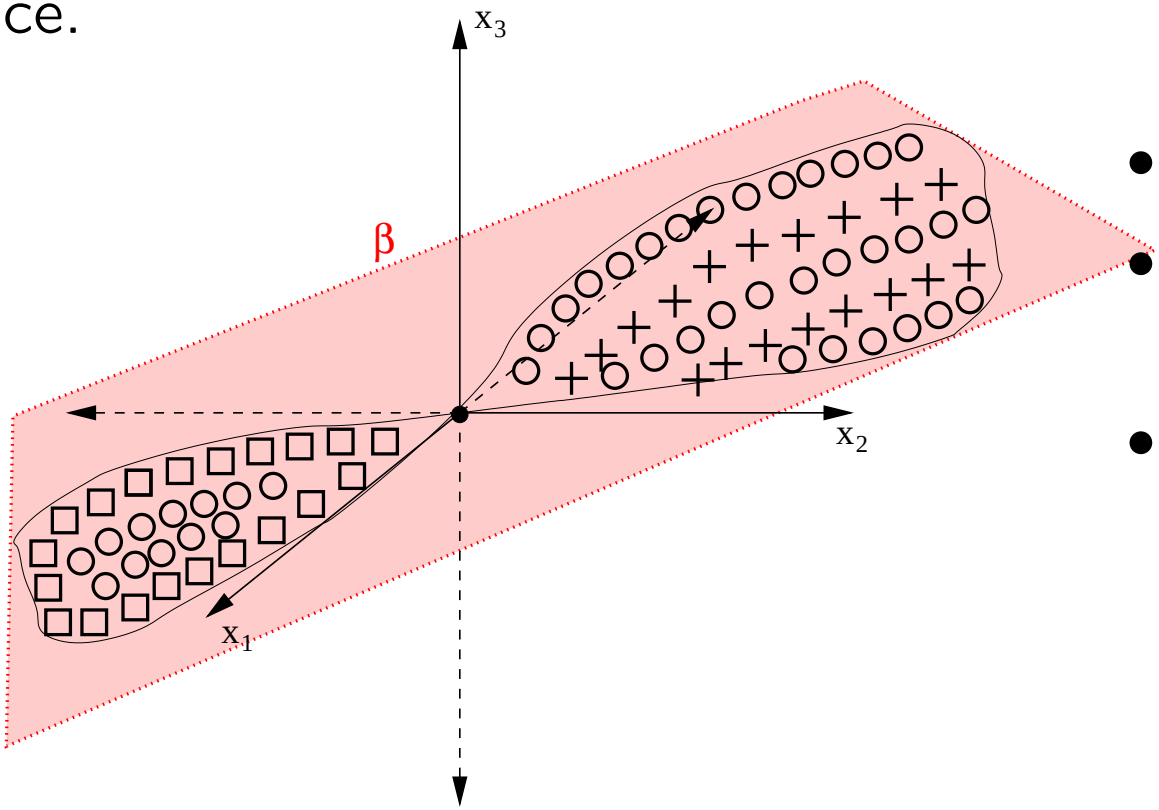
The structure of points $\mathbf{x} = (x_1, x_2, x_3)^T$ in $\mathbb{R}^3$ is inherently 1-dimensional and the points are linearly embedded in a 3-dimensional space.



- 3 types of points **x**
- Use PCA to get the direction of β.
- Draw the 1-D projections

Discovering low-dimensional spatial layout in higher dimensional spaces - 2-D/3-D example

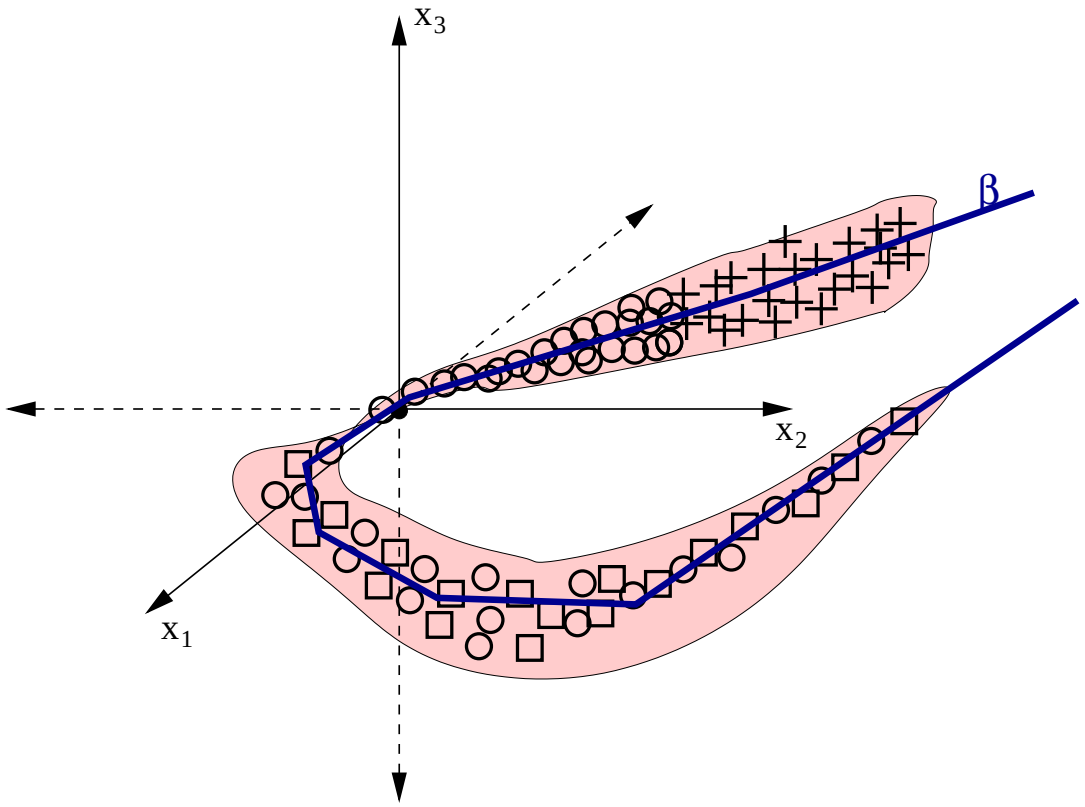
The structure of points $\mathbf{x} = (x_1, x_2, x_3)^T$ in $\mathbb{R}^3$ is inherently 2-dimensional and the points are linearly embedded in a 3-dimensional space.



- 3 types of points **x**
- Use PCA to get the principal directions of β.
- Draw the 2-D projections

Discovering low-dimensional spatial layout in higher dimensional spaces - 1-D/3-D example

The structure of points $\mathbf{x} = (x_1, x_2, x_3)^T$ in $\mathbb{R}^3$ is inherently 1-dimensional but this time the points are nonlinearly embedded in a 3-dimensional space.



- 3 types of points **x**
- In this case, we cannot use PCA to get the projection curve β.
- Draw the 1-D projections onto β

Discovering nonlinear low-dimensional spatial layout in higher dimensional spaces

Imagine that we have n-dimensional points $\mathbf{x} = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$ and the points are inherently 2-dimensional, i.e. lie on a smooth nonlinear 2-dimensional manifold embedded in $\mathbb{R}^n$.

PCA cannot be used to discover this structure, but what else can be used?

Draw an example of points lying on a smooth 2-dimensional manifold in $\mathbb{R}^3$ and draw the 2-dimensional projections.

Vector Quantization (VQ)

Imagine that we have many n-dimensional points $\mathbf{x}^i = (x_1^i, x_2^i, ..., x_n^i)^T \in \mathbb{R}^n,\ i = 1, 2, ..., N$.

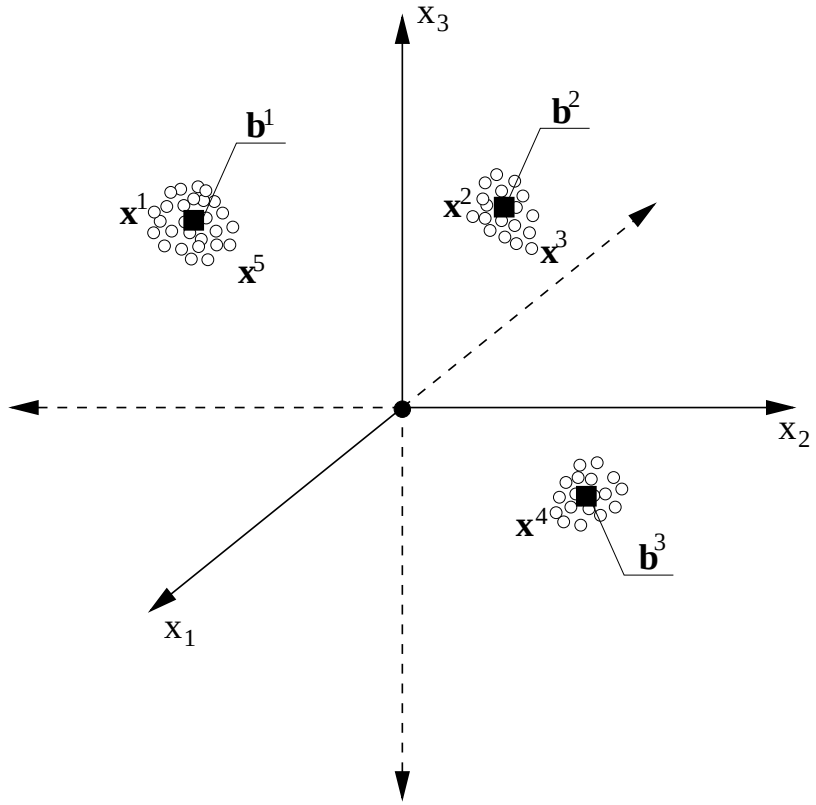
We would like to transmit all the points $\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^N$ through a communication channel.

Considering that we may be willing to accept a small loss in precision can we come up with an efficient communication protocol?

Idea:
Represent points by a small number of 'representative vectors' $\mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^M \in \mathbb{R}^n$ and transmit only those.

VQ

Take advantage of the cluster structure in the data $\mathbf{x}^i$, $i = 1, 2, ..., N$. To minimize the representation error, place the representatives $\mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^M$, known as codebook vectors, in the center of each cluster.



- To transmit $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4, ...,$ first transmit full information about the codebook $\{\mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3\}$
- Then instead of each point $\mathbf{x}^i$, transmit just the index of its closest representative codebook vector.

VQ - algorithm

To transmit data points $\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^N \in \mathbb{R}^n$, follow the following procedure:

1. Place a smaller number of codebook vectors $\mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^M \in \mathbb{R}^n$, $M < N$, into the cloud of points $\{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^N\}$.
2. Transmit all coordinates of the codebook vectors $\mathbf{b}^j = (b_1^j, b_2^j, ..., b_n^j)^T$, $j = 1, 2, ..., M$.
3. Go through the data points $\mathbf{x}^i$, $i = 1, 2, ..., N$, and instead of transmitting each point $\mathbf{x}^i$ transmit the identity (index) $win(i)$ of the codebook vector $\mathbf{b}^{win(i)}$ that is closest to $\mathbf{x}^i$:

$$win(i) = \underset{j=1,2,...,M}{\operatorname{argmin}} \|\mathbf{b}^j - \mathbf{x}^i\|$$

VQ - Placing the codebook vectors

There are many procedures for step 1 of the previous algorithm. Here is an iterative one:

1. Randomly place codebook vectors $\mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^M$ in $\mathbb{R}^n$.
   For example, make them identical with some data points chosen at random from $\{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^N\}$.
2. Cycle through the set of data points and for each point $\mathbf{x}^i$ do:
   (a) Find the closest codebook vector $\mathbf{b}^{win(i)}$.
   (b) Move $\mathbf{b}^{win(i)}$ a bit closer to $\mathbf{x}^i$:

$$\mathbf{b}_{new}^{win(i)} = \mathbf{b}_{old}^{win(i)} + \eta \cdot (\mathbf{x}^i - \mathbf{b}_{old}^{win(i)}),$$

where $\eta > 0$ is a small 'learning rate'.

VQ - Learning rate

The learning rate is actually not a constant, but is gradually decreased after each time step:

$$\eta(t) = \eta(0) \cdot e^{-\frac{t}{\tau}}.$$

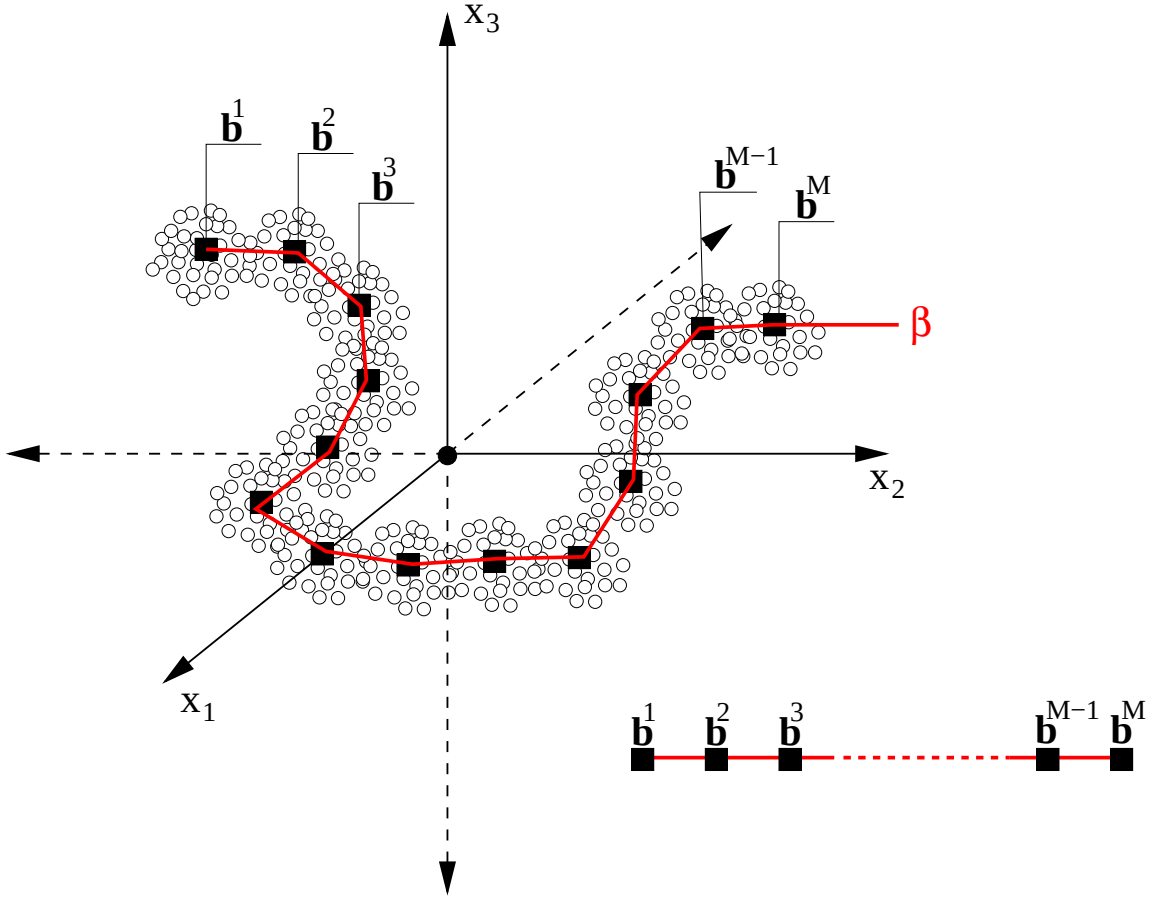
- $\tau > 0$
  determines the 'time scale', i.e. how fast will the learning rate decrease.

- $\eta(0)$
  is the initial learning rate.

This forces the training process for the codebook to converge.

Constrained VQ

You can discover a hidden 1-dimensional structure of high-dimensional points by running a VQ on them, but constrain the codebook vectors $\mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^M$ to lie on a one-dimensional 'bicycle chain'.



This way the codebook vectors will organize themselves along the 'curved noisy tube' (nonlinear 1-dimensional manifold)

Constrained VQ - Placing the codebook vectors

1. Randomly place codebook vectors $\mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^M$ in $\mathbb{R}^n$.

2. Cycle through the set of data points and for each point $\mathbf{x}^i$ do:

   (a) Find the closest codebook vector $\mathbf{b}^{win(i)}$.

   (b) Move $\mathbf{b}^{win(i)}$ a bit closer to $\mathbf{x}^i$:

$$\mathbf{b}_{new}^{win(i)} = \mathbf{b}_{old}^{win(i)} + \eta \cdot (\mathbf{x}^i - \mathbf{b}_{old}^{win(i)}).$$

   (c) Push towards $\mathbf{x}^i$ also the codebook vectors $\mathbf{b}^j$ that are neighbors of $\mathbf{b}^{win(i)}$ on the bicycle chain (1-dimensional grid of codebook vectors).
   For each codebook vector $\mathbf{b}^j$, $j = 1, 2, ..., M$,

$$\mathbf{b}_{new}^{j} = \mathbf{b}_{old}^{j} + h[win(i), j] \cdot \eta \cdot (\mathbf{x}^i - \mathbf{b}_{old}^{j}).$$

Quantifying the 'degree of neighborhood' - closeness on the bicycle chain

Function $h[win(i), j]$ quantifies how close on the bicycle chain are the codebook vectors $\mathbf{b}^{win(i)}$ and $\mathbf{b}^{j}$.

Note that the neighborhood relations between codebooks (which codebooks should be placed next to each other) are decided before seeing the actual data! Everything is decided based on indexes $win(i)$ and j of the codebook vectors, and not the codebook vectors $\mathbf{b}^{win(i)}$ and $\mathbf{b}^{j}$ themselves!

We would like:

- $h[win(i), win(i)] = 1$
- The further away codebook vector no. j is from $win(i)$, the smaller $h[win(i), j]$ should be.

Quantifying the 'degree of neighborhood'

Here is one suggestion:

$$h[win(i), j] = e^{-\frac{|win(i)-j|}{\sigma}}$$

- $|win(i) - j|$
  is the distance between the indexes $win(i)$ and j on the bicycle chain
- σ
  is the 'width' of the neighborhood

Neighborhood width σ

Initially, the neighborhood width σ should be large to allow for broad cooperation of codebook vectors, so that all of them get into the cloud of points

After this, the neighborhood width should get more and more specific (smaller) to allow for fine tuning of topographic structure of codebook vectors dictated by the order of indexes in the bicycle chain.

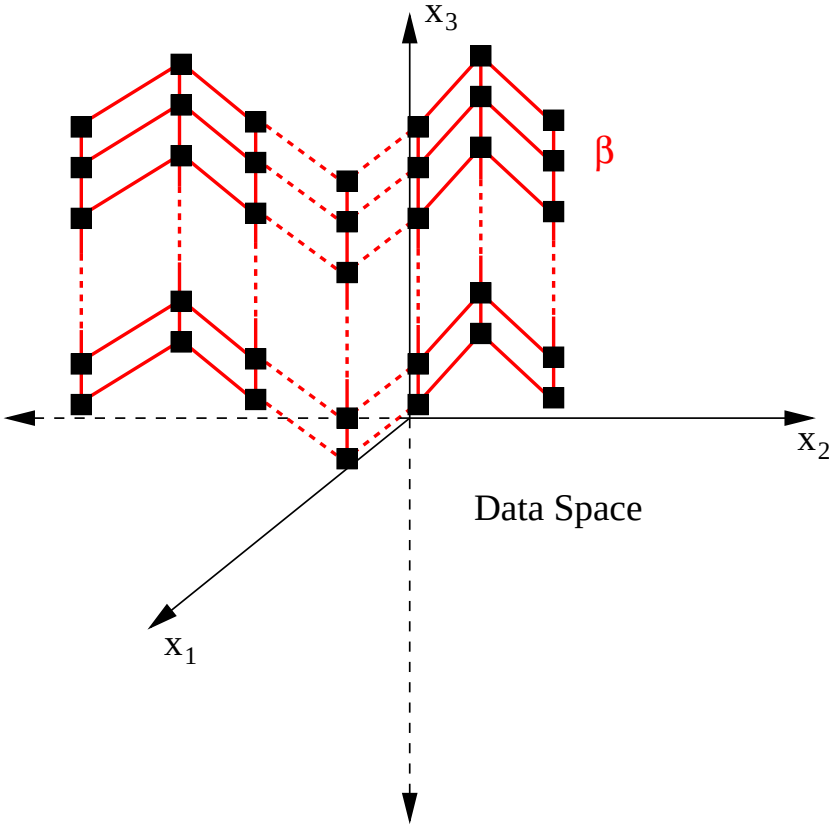
Suggestion: analogy to learning rate adaptation

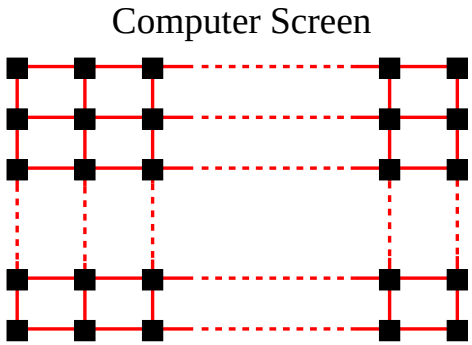
$$\sigma(t) = \sigma(0) \cdot e^{-\frac{t}{\nu}}.$$

- $\nu > 0$ – 'time scale'
- $\sigma(0)$ – initial neighborhood width.

Two-dimensional grid of codebook vectors

Generalize the notion of 'bicycle chain' of codebook vectors: Take advantage of two-dimensional structure of the computer screen. Cover it with a 2-dimensional grid of nodes.





2-dimensional topographic map

1. Given N data points $\mathbf{x}^i = (x_1^i, x_2^i, ..., x_n^i)^T \in \mathbb{R}^n$ and a 2-dim grid topography of codebook vectors.

2. Randomly place codebook vectors $\mathbf{b}^1, \mathbf{b}^2, ..., \mathbf{b}^M$ in $\mathbb{R}^n$.

3. Cycle through the data points. For each point $\mathbf{x}^i$ do:

   (a) Find the closest codebook vector $\mathbf{b}^{win(i)}$.

   (b) For each codebook vector $\mathbf{b}^j$, $j = 1, 2, ..., M$, do:

$$\mathbf{b}^j_{new} = \mathbf{b}^j_{old} + h[win(i), j] \cdot \eta \cdot (\mathbf{x}^i - \mathbf{b}^j_{old}).$$

$$h[win(i), j] = e^{-\frac{\|win(i) - j\|}{\sigma}}$$

   (c) Decrease the neighborhood width $\sigma(t)$ and learning rate $\eta(t)$.

Data visualization using top topographic maps

Train the topographic map

Represent points $\mathbf{x}^i \in \mathbb{R}^n$ by two-dimensional projections $\tilde{\mathbf{x}}^i$ on the computer screen.

$\tilde{\mathbf{x}}^i$ is the (2-dimensional) index $win(i)$ of the codebook vector $\mathbf{b}^{win(i)}$ that is closest to $\mathbf{x}^i$:

$$\tilde{\mathbf{x}}^i = win(i).$$