Unsupervised learning

Johan Suykens

K.U. Leuven, ESAT-STADIUS
Kasteelpark Arenberg 10
B-3001 Leuven (Heverlee), Belgium
Email: johan.suykens@esat.kuleuven.be
http://www.esat.kuleuven.be/stadius

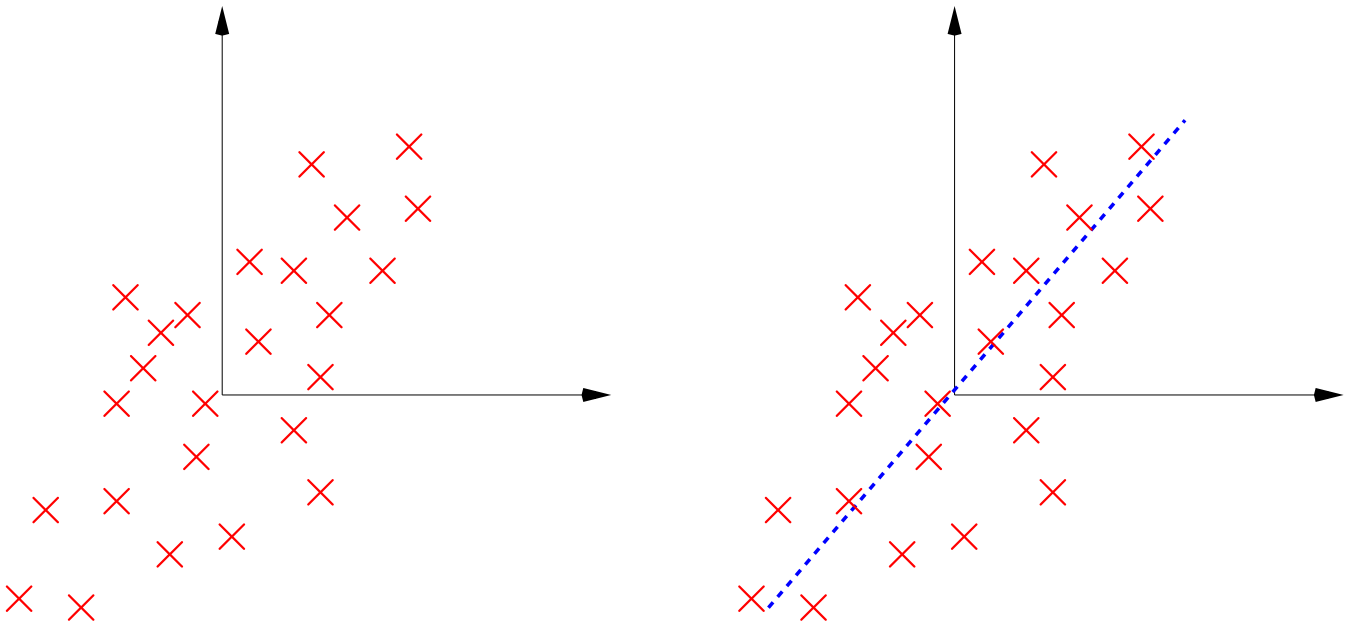
Lecture 7

Overview

- Principal component analysis
- Oja's rule
- Nonlinear PCA
- K-means clustering
- Vector quantization
- Self-organizing maps

Principal component analysis (1)

- Given a data cloud, potentially in a high dimensional input space



- assume an ellipsoidal data cloud
  search for direction(s) in the data of **maximal variance**

Principal component analysis (2)

- Given data $\{x_i\}_{i=1}^N$ with $x_i \in \mathbb{R}^n$ (assumed zero mean)
- Find **projected variables** $w^T x_i$ with maximal variance

$$\begin{aligned} \max_w E\{(w^T x)^2\} &= w^T E\{xx^T\} w \\ &= w^T C\, w \end{aligned}$$

  with **covariance matrix** $C = E\{xx^T\}$ and $E\{\cdot\}$ the expected value.

- For N given data points one has $C \simeq \frac{1}{N} \sum_{i=1}^{N} x_i x_i^T$.
-

Principal component analysis (2)

- Given data $\{x_i\}_{i=1}^N$ with $x_i \in \mathbb{R}^n$ (assumed zero mean)
- Find **projected variables** $w^T x_i$ with maximal variance

$$\begin{aligned} \max_w E\{(w^T x)^2\} &= w^T E\{xx^T\} w \\ &= w^T C\, w \end{aligned}$$

  with **covariance matrix** $C = E\{xx^T\}$ and $E\{\cdot\}$ the expected value.

- For N given data points one has $C \simeq \frac{1}{N} \sum_{i=1}^{N} x_i x_i^T$.
- Problem: the optimal solution for w in the above problem is unbounded. Therefore an additional constraint should be taken: a common choice is to impose $w^T w = 1$.

Principal component analysis (3)

- The problem formulation becomes then:

$$\max_{w} w^T C\, w \quad \text{subject to} \quad w^T w = 1$$

-

-

Principal component analysis (3)

- The problem formulation becomes then:

$$\max_{w} w^T C\, w \quad \text{subject to} \quad w^T w = 1$$

- Constrained optimization problem solved by taking the Lagrangian $\mathcal{L}$:

$$\mathcal{L}(w;\lambda) = \frac{1}{2} w^T C w - \lambda(w^T w - 1)$$

  with Lagrange multiplier λ.

-

Principal component analysis (3)

- The problem formulation becomes then:

$$\max_{w} w^T C\, w \quad \text{subject to} \quad w^T w = 1$$

- Constrained optimization problem solved by taking the Lagrangian $\mathcal{L}$:

$$\mathcal{L}(w;\lambda) = \frac{1}{2} w^T C w - \lambda(w^T w - 1)$$

  with Lagrange multiplier λ.

- Solution is given by the **eigenvalue problem**

$$Cw = \lambda w$$

  with $C = C^T \geq 0$, obtained from setting $\partial\mathcal{L}/\partial w = 0$, $\partial\mathcal{L}/\partial\lambda = 0$.

Principal component analysis (4)

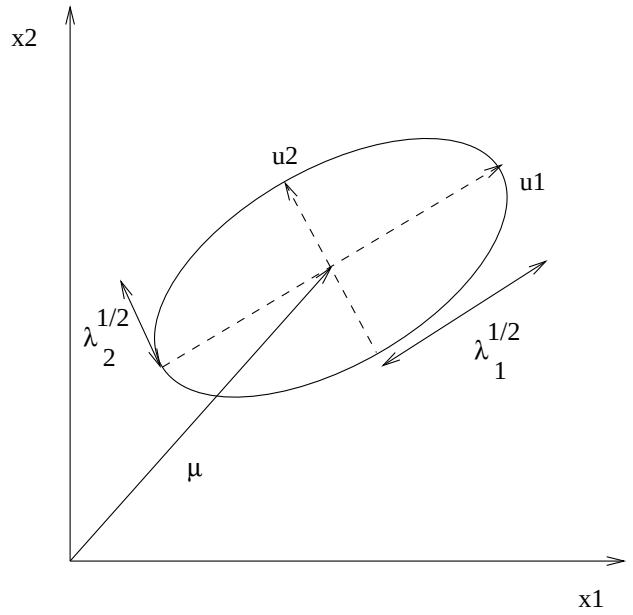


Illustration of an eigenvalue decomposition $Cu = \lambda u$

- the eigenvalues λ_i are real and positive

Principal component analysis (4)

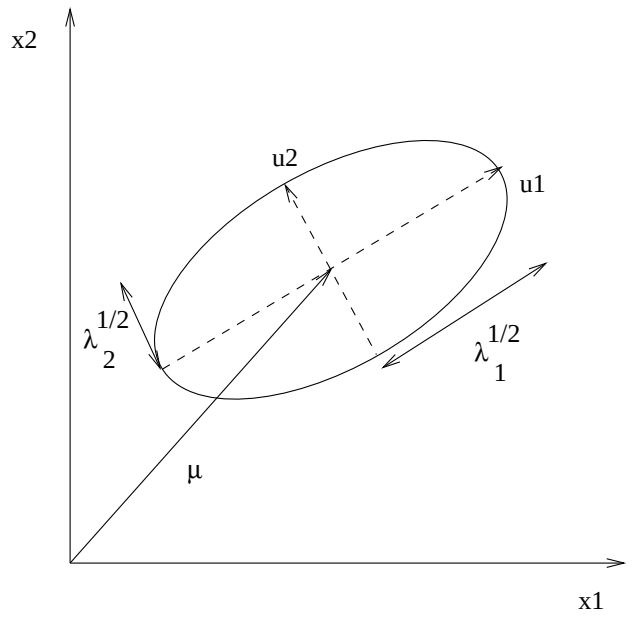


Illustration of an eigenvalue decomposition $Cu = \lambda u$

- the eigenvalues λ_i are real and positive
- the eigenvectors u_1 and u_2 are orthogonal with respect to each other

Principal component analysis (4)

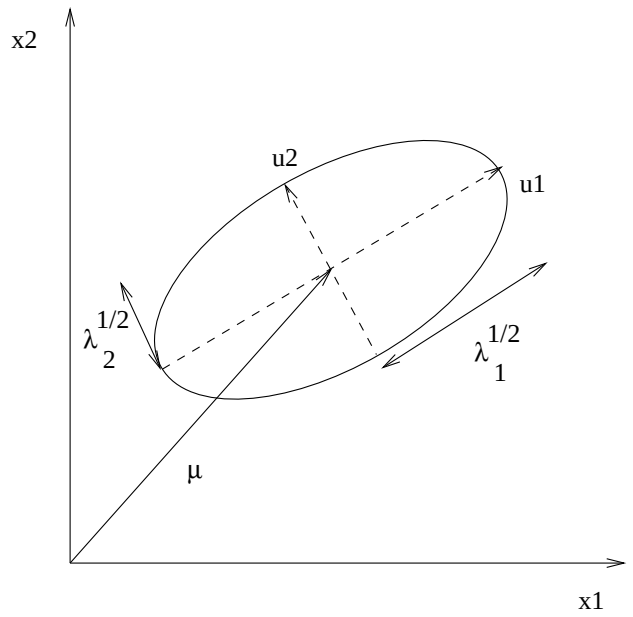


Illustration of an eigenvalue decomposition $Cu = \lambda u$

- the eigenvalues λ_i are real and positive
- the eigenvectors u_1 and u_2 are orthogonal with respect to each other
- maximal variance solution is the direction u_1 corresponding to $\lambda_{\max} = \lambda_1$.

Principal component analysis (4)

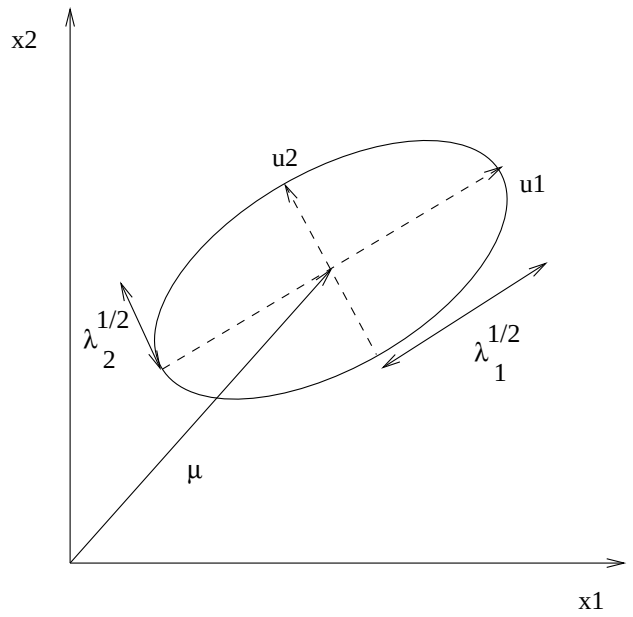


Illustration of an eigenvalue decomposition $Cu = \lambda u$

- the eigenvalues λ_i are real and positive
- the eigenvectors u_1 and u_2 are orthogonal with respect to each other
- maximal variance solution is the direction u_1 corresponding to $\lambda_{\max} = \lambda_1$.
- note that $\mu = 0$ (the data should be made zero-mean beforehand)

Principal component analysis: dimensionality reduction

- **Aim:**
  Decreasing the dimensionality of the given input space by mapping vectors $x \in \mathbb{R}^n$ to $z \in \mathbb{R}^m$ with $m < n$.

- A point x is mapped to z in the lower dimensional space by

$$z_j = u_j^T x$$

  where u_j are the eigenvectors corresponding to the m **largest eigenvalues** and $z = [z_1 z_2 ... z_m]^T$.

-

Principal component analysis: dimensionality reduction

- **Aim:**
  Decreasing the dimensionality of the given input space by mapping vectors $x \in \mathbb{R}^n$ to $z \in \mathbb{R}^m$ with $m < n$.

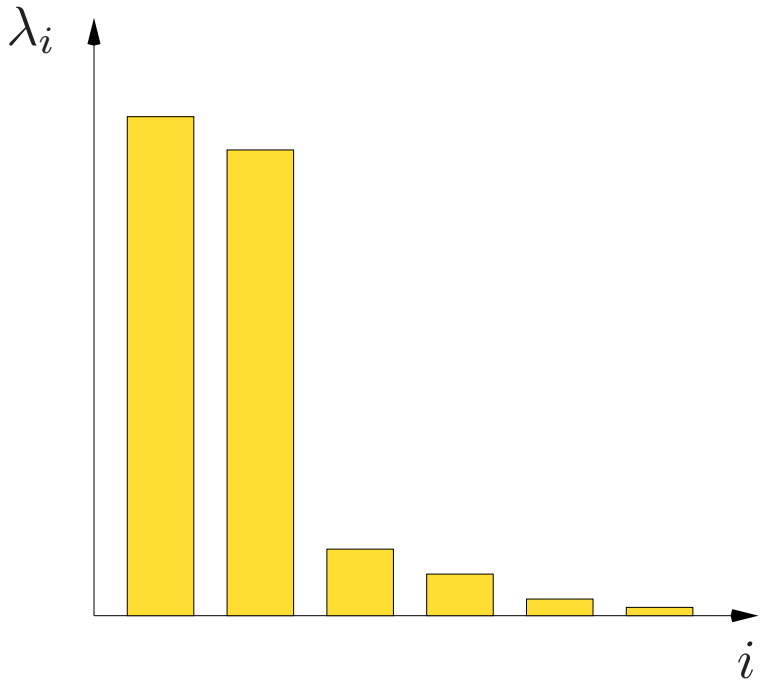
- A point x is mapped to z in the lower dimensional space by

$$z_j = u_j^T x$$

  where u_j are the eigenvectors corresponding to the m **largest eigenvalues** and $z = [z_1 z_2 ... z_m]^T$.

- The error resulting from the dimensionality reduction is characterized by the neglected eigenvalues, i.e. $\sum_{i=m+1}^{n} \lambda_i$.

Principal component analysis: dimensionality reduction (1)



In this example reducing the original 6-dimensional space to a 2-dimensional space is a good choice because the largest two eigenvalues λ_1 and λ_2 are much larger than the other ones.

Principal component analysis: reconstruction problem (2)

- Consider $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$ with $m \ll n$ (dimensionality reduction).
- **Encoder** mapping:

$$z = G(x)$$

  Decoder mapping:

$$\tilde{x} = F(z)$$

-
-

Principal component analysis: reconstruction problem (2)

- Consider $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$ with $m \ll n$ (dimensionality reduction).
- **Encoder** mapping:

$$z = G(x)$$

  Decoder mapping:

$$\tilde{x} = F(z)$$

- Objective: squared distortion error (**reconstruction error**)

$$\min E = \frac{1}{N} \sum_{i=1}^{N} \|x_i - \tilde{x}_i\|_2^2 = \frac{1}{N} \sum_{i=1}^{N} \|x_i - F(G(x_i))\|_2^2$$

-

Principal component analysis: reconstruction problem (2)

- Consider $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$ with $m \ll n$ (dimensionality reduction).
- **Encoder** mapping:

$$z = G(x)$$

  Decoder mapping:

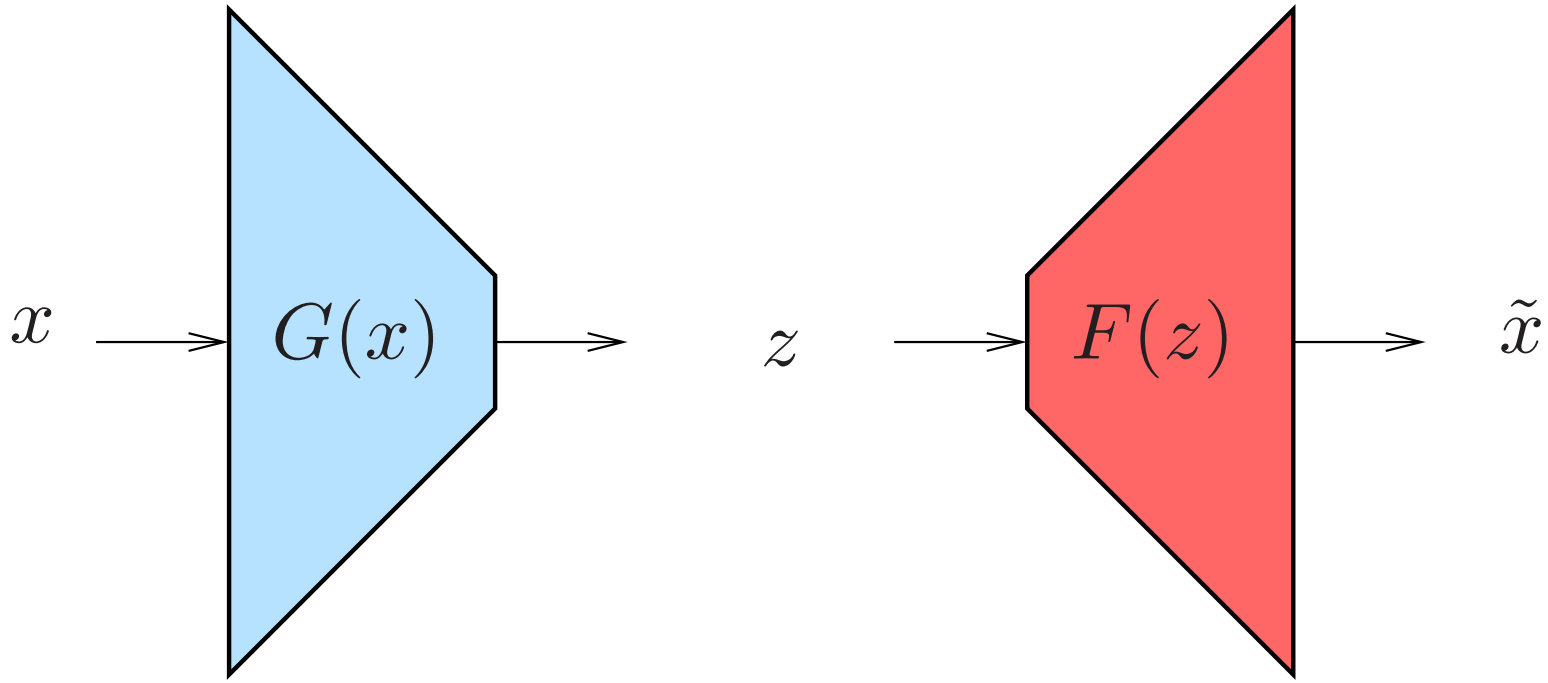
$$\tilde{x} = F(z)$$

- Objective: squared distortion error (**reconstruction error**)

$$\min E = \frac{1}{N} \sum_{i=1}^{N} \|x_i - \tilde{x}_i\|_2^2 = \frac{1}{N} \sum_{i=1}^{N} \|x_i - F(G(x_i))\|_2^2$$

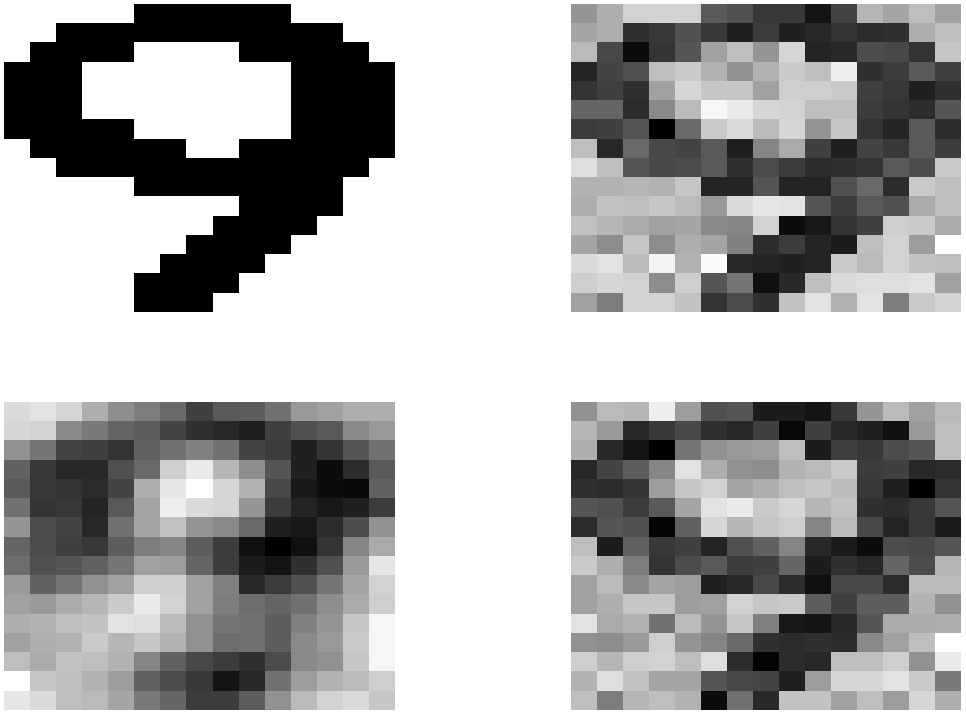
- Taking the mappings F, G linear corresponds to **linear PCA** analysis

Principal component analysis: reconstruction problem (2)



Information bottleneck

Principal component analysis: denoising example

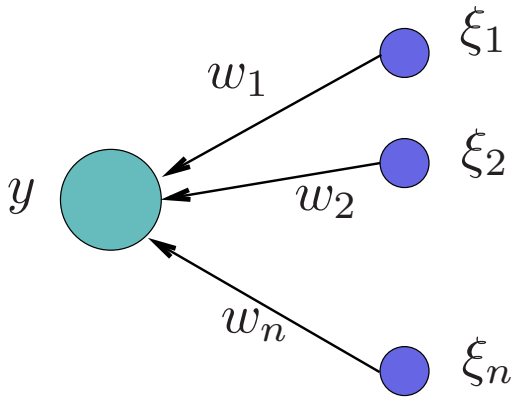


Images: 16×15 pixels ($n = 240$)
Training on $N = 200$ clean digits (not containing digit 9)
Test data: (bottom-left) denoised digit 9 after reconstruction using 10 principal components; (bottom-right) 180 principal components

Oja's rule (1)

- Consider a **linear neuron** with output y and inputs ξ_j:

$$y = \sum_j w_j \xi_j = w^T \xi$$



- Consider time dependency with time t

$$y(t) = w(t)^T \xi(t)$$

- **Oja's learning rule** [Oja, 1982]

$$w(t+1) = w(t) + \eta y(t)(\xi(t) - y(t)w(t))$$

Oja's rule (2)

- Averaging over time t gives

$$E\{w(t+1) - w(t)\} \quad = \quad E\{\eta y(t)(\xi(t) - y(t)w(t))\}$$

-

Oja's rule (2)

- Averaging over time t gives

$$\begin{aligned} E\{w(t+1) - w(t)\} &= E\{\eta y(t)(\xi(t) - y(t)w(t))\} \\ &= E\{\eta w(t)^T \xi(t)(\xi(t) - \xi(t)^T w(t) w(t))\} \end{aligned}$$

-

Oja's rule (2)

- Averaging over time t gives

$$\begin{aligned} E\{w(t+1) - w(t)\} &= E\{\eta y(t)(\xi(t) - y(t)w(t))\} \\ &= E\{\eta w(t)^T \xi(t)(\xi(t) - \xi(t)^T w(t) w(t))\} \\ &= \eta E\{\xi(t)\xi(t)^T w(t) - w(t)^T \xi(t)\xi(t)^T w(t) w(t))\} \end{aligned}$$

-

Oja's rule (2)

- Averaging over time t gives

$$\begin{aligned} E\{w(t+1) - w(t)\} &= E\{\eta y(t)(\xi(t) - y(t)w(t))\} \\ &= E\{\eta w(t)^T \xi(t)(\xi(t) - \xi(t)^T w(t) w(t))\} \\ &= \eta E\{\xi(t)\xi(t)^T w(t) - w(t)^T \xi(t)\xi(t)^T w(t) w(t))\} \\ &= \eta[Cw(t) - w(t)^T Cw(t)\, w(t)] \end{aligned}$$

-

Oja's rule (2)

- Averaging over time t gives

$$\begin{aligned} E\{w(t+1) - w(t)\} &= E\{\eta y(t)(\xi(t) - y(t)w(t))\} \\ &= E\{\eta w(t)^T \xi(t)(\xi(t) - \xi(t)^T w(t) w(t))\} \\ &= \eta E\{\xi(t)\xi(t)^T w(t) - w(t)^T \xi(t)\xi(t)^T w(t) w(t))\} \\ &= \eta[Cw(t) - w(t)^T Cw(t)\, w(t)] \end{aligned}$$

  with covariance matrix $C = E\{\xi(t)\xi(t)^T\}$, and assuming $w(t)$ and $\xi(t)$ to be statistically independent.

- One expects $w(t+1) = w(t)$ at equilibrium and a learning process converging to w^* with

$$Cw^* = \lambda_{\max} w^*$$

  with $\lambda_{\max} = w^{*T} C w^*$ when $w^{*T} w^* = 1$.

Oja's rule: Hebbian learning interpretation

- Oja's learning rule: $w(t+1) = w(t) + \Delta w(t)$

$$\Delta w(t) = \eta y(t)(\xi(t) - y(t)w(t))$$

- One can view this as **Hebbian learning**

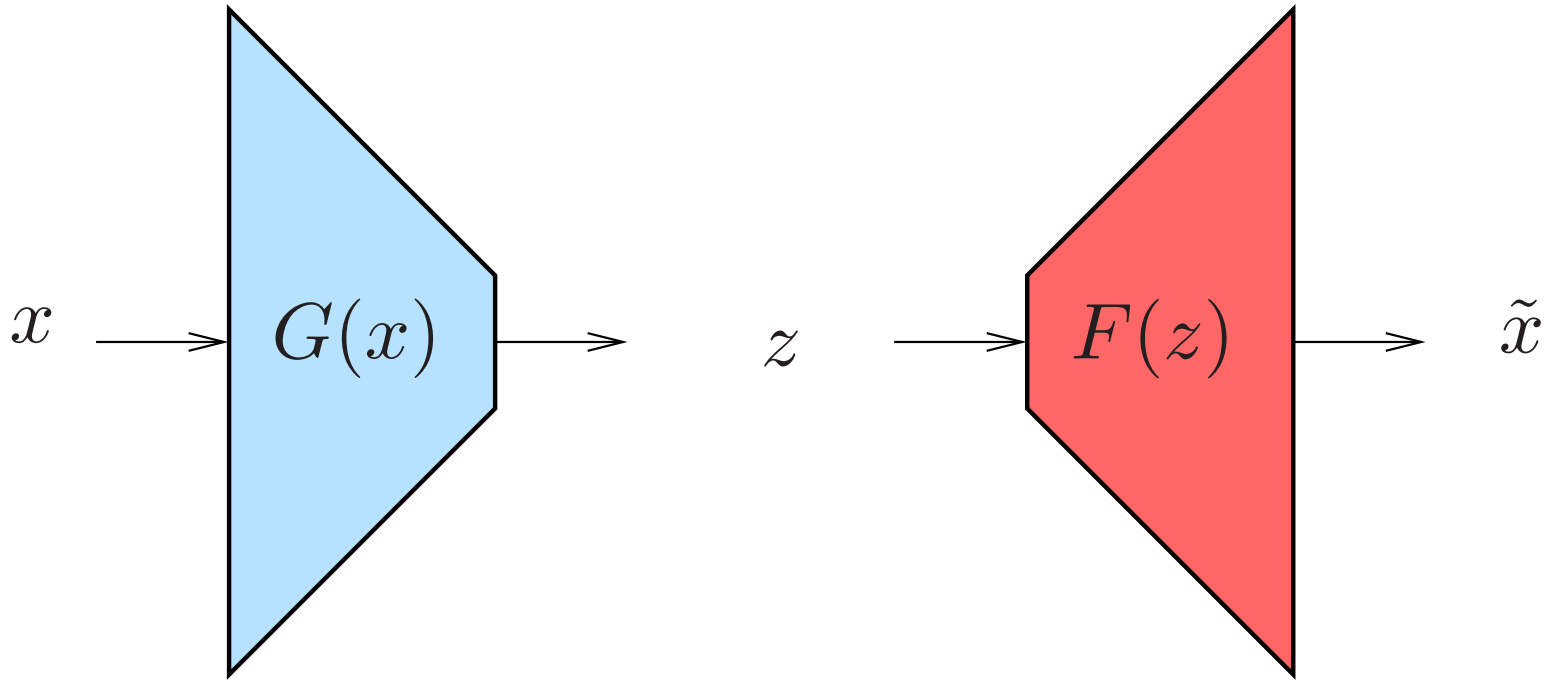
$$\Delta w(t) = \eta y(t)\tilde{\xi}(t)$$

for an effective input $\tilde{\xi}$

$$\tilde{\xi}(t) = \xi(t) - y(t)w(t)$$

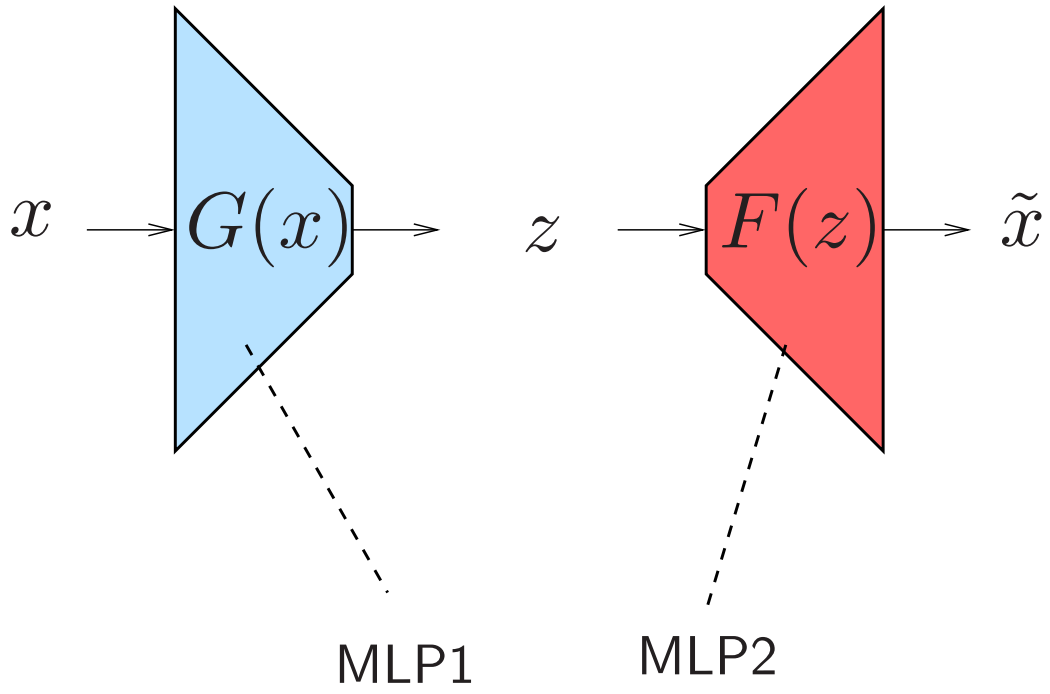
where the second term $-y(t)w(t)$ is a forgetting or **leakage term**.

Reconstruction problem again: nonlinear mappings (1)



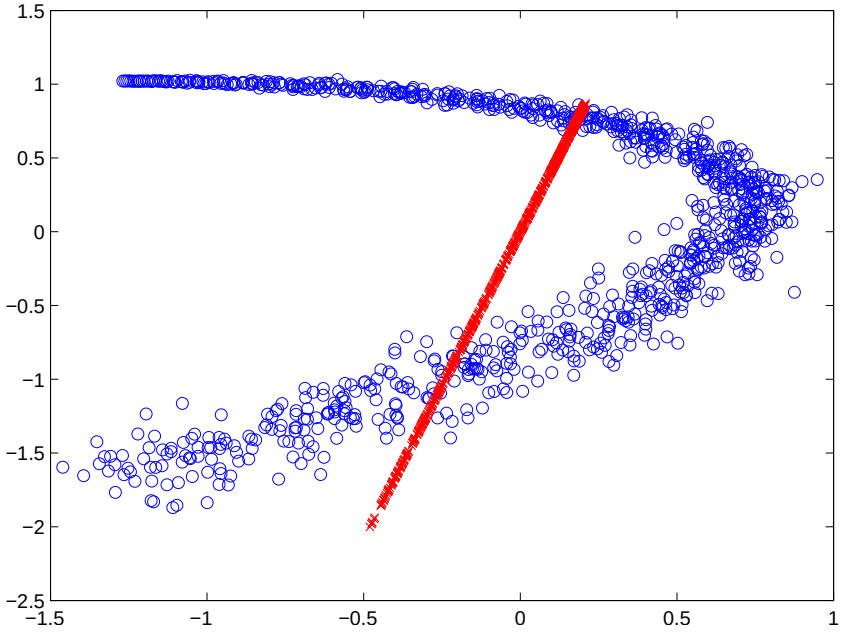
Instead of F, G linear, consider now F, G to be **nonlinear** and parameterized by multilayer perceptrons.

Reconstruction problem again: nonlinear mappings (2)

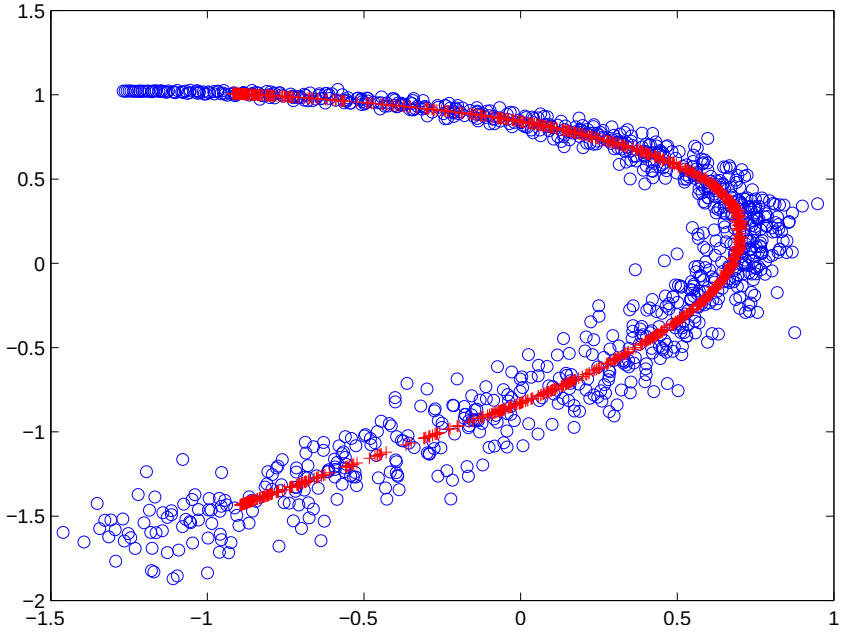


Nonlinear principal component analysis using multilayer perceptrons: learn the interconnection matrices of MLP1 and MLP2 by minimizing the reconstruction error.

Linear versus nonlinear PCA (1)

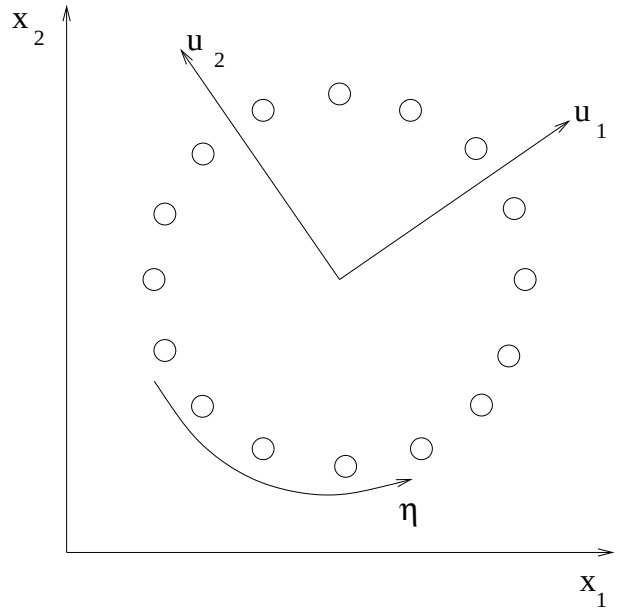


linear PCA



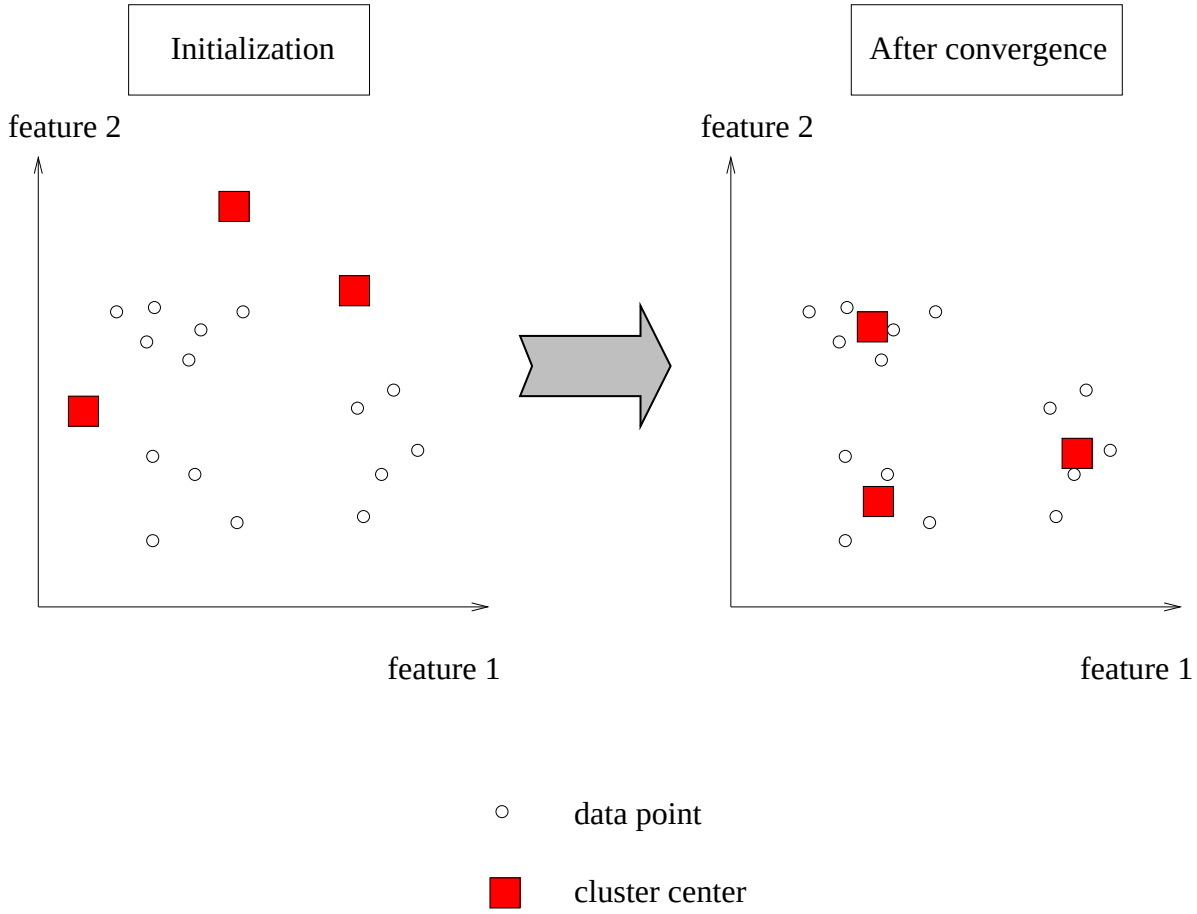
nonlinear PCA

Linear versus nonlinear PCA (2)



The data set in two dimensions has intrinsic dimensionality 1
The data can be explained in terms of the single parameter η
Linear PCA is unable to detect the lower dimensionality

Clustering: K-means algorithm (1)



Cluster = set of points that are close in distance to the cluster center
(according to the distance measure as chosen by the user)

Clustering: K-means algorithm (2)

1. Choose K **initial** cluster centers $z_1(1), ..., z_K(1)$.
2.
3.
4.

Clustering: K-means algorithm (2)

1. Choose K **initial** cluster centers $z_1(1), ..., z_K(1)$.
2. At iteration k, distribute the data $\{x\}$ among the K **cluster domains**:

$$x \in S_j(k) \text{ if } \|x - z_j(k)\| < \|x - z_i(k)\| \text{ , } \forall i \neq j$$

   where $S_j(k)$ is the set of data with cluster center $z_j(k)$.

3.

4.

Clustering: K-means algorithm (2)

1. Choose K **initial** cluster centers $z_1(1), ..., z_K(1)$.
2. At iteration k, distribute the data $\{x\}$ among the K **cluster domains**:

$$x \in S_j(k) \text{ if } \|x - z_j(k)\| < \|x - z_i(k)\| \ , \forall i \neq j$$

   where $S_j(k)$ is the set of data with cluster center $z_j(k)$.

3. **New cluster centers:**

$$z_j(k+1) = \frac{1}{N} \sum_{x \in S_j(k)} x, \quad j = 1, 2, ..., K$$

   It minimizes performance indices $J_j = \sum_{x \in S_j(k)} \|x - z_j(k+1)\|^2, \forall j$

4.

Clustering: K-means algorithm (2)

1. Choose K **initial** cluster centers $z_1(1), ..., z_K(1)$.
2. At iteration k, distribute the data $\{x\}$ among the K **cluster domains**:

$$x \in S_j(k) \text{ if } \|x - z_j(k)\| < \|x - z_i(k)\| \ , \forall i \neq j$$

   where $S_j(k)$ is the set of data with cluster center $z_j(k)$.

3. **New cluster centers:**

$$z_j(k+1) = \frac{1}{N} \sum_{x \in S_j(k)} x, \quad j = 1, 2, ..., K$$

   It minimizes performance indices $J_j = \sum_{x \in S_j(k)} \|x - z_j(k+1)\|^2, \forall j$

4. If $z_j(k+1) = z_j(k)$ for $j = 1, 2, ..., K$ the algorithm has converged. Otherwise go to step 2.

Vector quantization

VQ: competitive learning (or stochastic approximation) version
Data $x(k)$ for $k = 1, 2, ...$
Initial centers $c_j(0)$ for $j = 1, 2, ..., s$ (s centers, prototypes)

1. Determine nearest center $c_j(k)$ to data point $x(k)$.
   For squared error loss function:

$$j = \arg\min_l \|x(k) - c_l(k)\|$$

   Finding the nearest center is called a **competition** among centers.

2.

Vector quantization

VQ: competitive learning (or stochastic approximation) version
Data $x(k)$ for $k = 1, 2, ...$
Initial centers $c_j(0)$ for $j = 1, 2, ..., s$ (s centers, prototypes)

1. Determine nearest center $c_j(k)$ to data point $x(k)$.
   For squared error loss function:

$$j = \arg\min_l \|x(k) - c_l(k)\|$$

   Finding the nearest center is called a **competition** among centers.

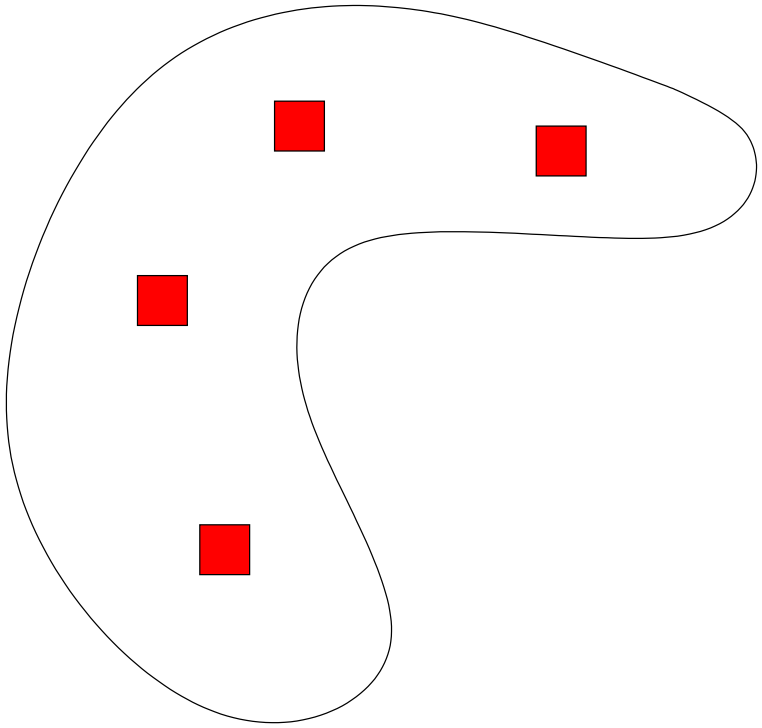
2. **Update center:**

$$\begin{aligned} c_j(k+1) &:= c_j(k) + \gamma(k)[x(k) - c_j(k)] \\ k &:= k+1 \end{aligned}$$

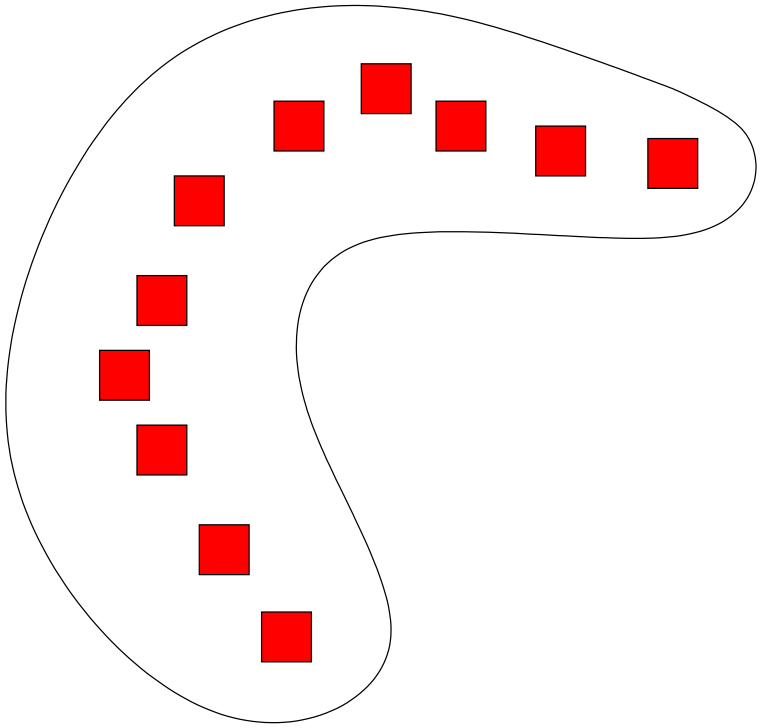
   where $\gamma(k)$ meets the conditions for stochastic approximation.

Vector quantization: prototype vectors

Characterizing the **data distribution** with **prototype vectors**



few prototypes



many prototypes

Vector quantization: distortion measure

- The vector quantizer Q is a mapping $Q : \mathbb{R}^n \rightarrow \mathcal{C}$ where $\mathcal{C} = \{c_1, c_2, ..., c_s\}$ (winning units).

- The space $\mathbb{R}^n$ is **partitioned** into the regions $\mathcal{R}_1, ..., \mathcal{R}_s$ where

$$\mathcal{R}_j = \{x \in \mathbb{R}^n : Q(x) = c_j\}$$

- The algorithm minimizes a **distortion measure** of the form

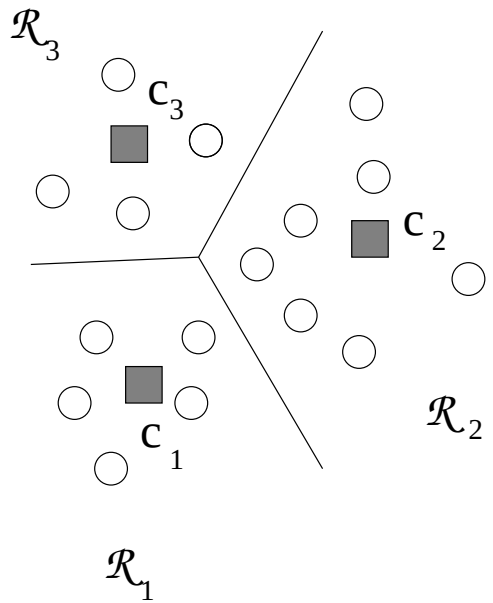
$$\int \|x - \sum_j c_j \mathcal{I}_{[x \in \mathcal{R}_j]}\|^2 p(x) dx$$

  where $\mathcal{R}_j$ corresponds to the j-th partition region, and $\mathcal{I}_{[z]} = 1$ if z is true and $\mathcal{I}_{[z]} = 0$ otherwise.

Vector quantization: Voronoi partition

- The partition regions of a vector quantizer are non-overlapping and cover the entire input space $\mathbb{R}^n$.

- Optimal vector quantizer: **Voronoi partition**.



Self-organizing maps (1)

Objectives of Self-Organizing Maps (SOM):

1. try to represent the underlying density of the input data by means of **prototype vectors** (similar to vector quantization)

2. project the higher dimensional input data to a map of neurons (also called nodes or units) such that the data can be visualized. Typically one has a projection to a **2-dimensional grid of neurons**.

In this way the SOM compresses information while preserving the most important topological and metric relationships of the primary data items on the display.

Self-organizing maps (2)

- **Dimensionality reduction** by projecting to a 2-dimensional map:

Input training data	$x_i \in \mathbb{R}^n$	for $i = 1, ..., N$
Prototype vectors	$c_j \in \mathbb{R}^n$	for $j = 1, ..., s$
Map coordinates	$z_j \in \mathbb{R}^2$	for $j = 1, ..., s$

  where

N	number of training data
s	number of neurons

- The **"neurons"** have in fact two positions:

in "input" space:	prototypes $c_j \in \mathbb{R}^n$
in "output" space:	map coordinates $z_j \in \mathbb{R}^2$

  for $j = 1, ..., s$.

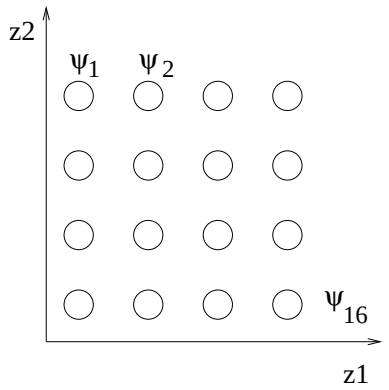
Self-organizing maps (3)

- One typically takes a 2-dimensional grid of neurons:

$$\psi = \{\psi_1, \psi_2, ..., \psi_s\}$$

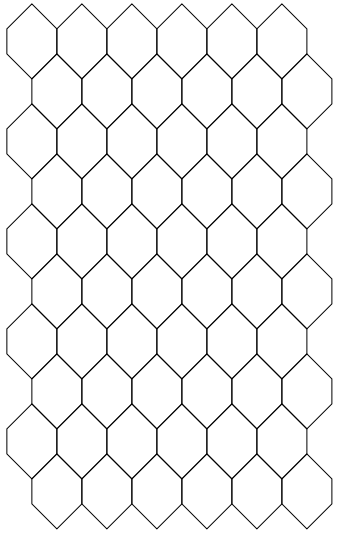
  where $\psi(j)$ or ψ_j denotes the jth element of ψ.

- An illustration is given for $s = 16$ neurons:

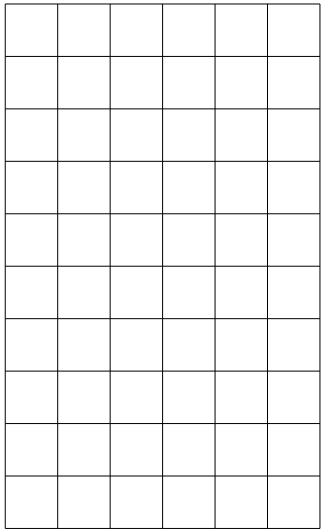


Self-organizing maps (4)

- Some possible grid choices:



Hexagonal grid



Rectangular grid

- Often the hexagonal grid is chosen

- Typical choice for number of neurons: $s = 5\sqrt{N}$

SOM: on-line learning version

1. At time k, determine the **winning unit** (also called best matching unit)

$$z(k) = \psi(\arg\min_j \|x(k) - c_j(k-1)\|)$$

2.

3.

SOM: on-line learning version

1. At time k, determine the **winning unit** (also called best matching unit)

$$z(k) = \psi(\arg\min_j \|x(k) - c_j(k-1)\|)$$

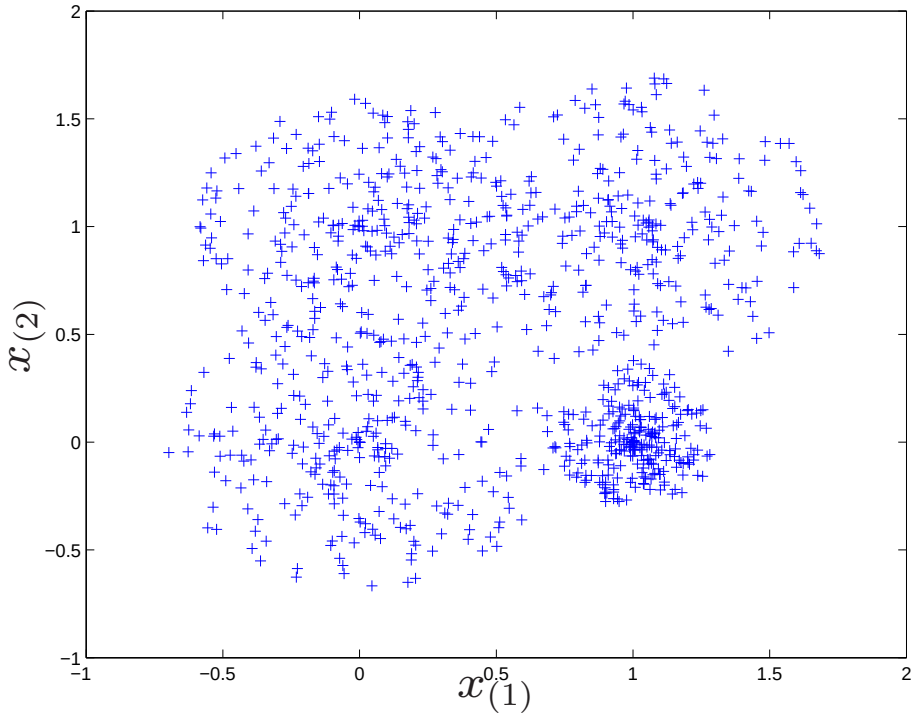
2. **Update** all units

$$\begin{aligned} c_j(k) &= c_j(k-1) + \beta(k) K_{\sigma(k)}(\psi(j), z(k)) \left[x(k) - c_j(k-1)\right] \\ k &:= k+1 \end{aligned}$$

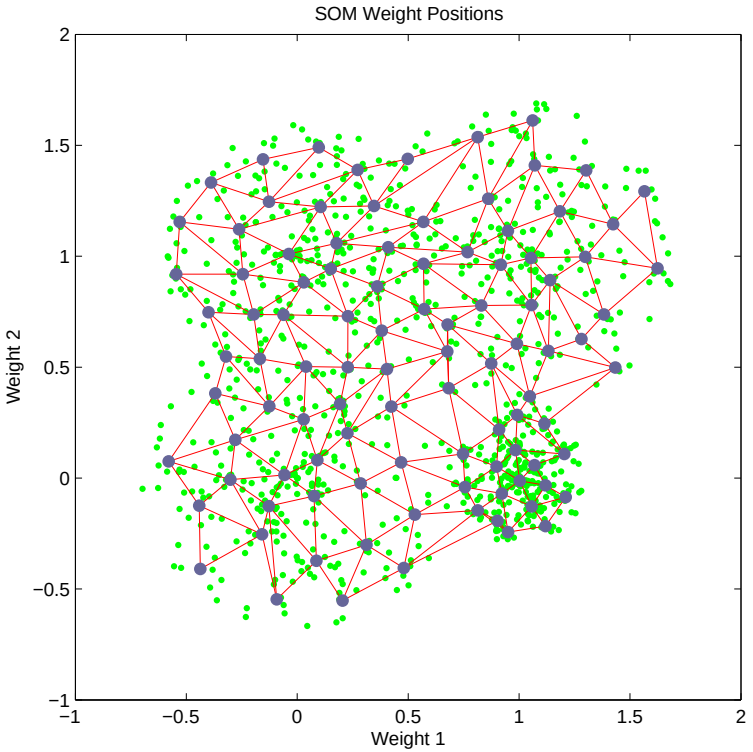
for $j = 1, ..., s$.
$K_{\sigma(k)}$ denotes a kernel function with width $\sigma(k)$, e.g. $K_\sigma(z, z_i) = \exp(-\|z - z_i\|^2/2\sigma^2)$ where the **neighborhood size** is controlled by σ.

3. Decrease the learning rate $\beta(k)$ and the width $\sigma(k)$

SOM example in Matlab (1)

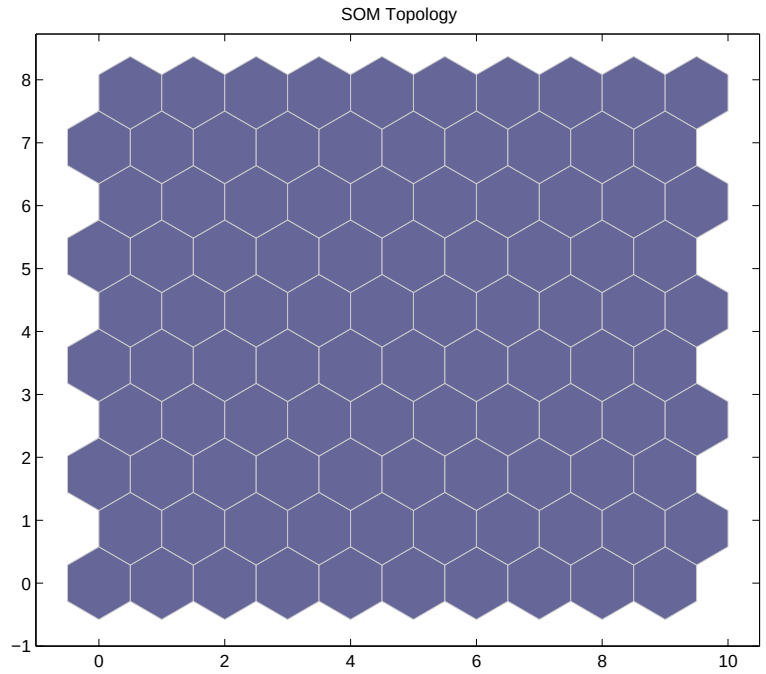


given data in input space

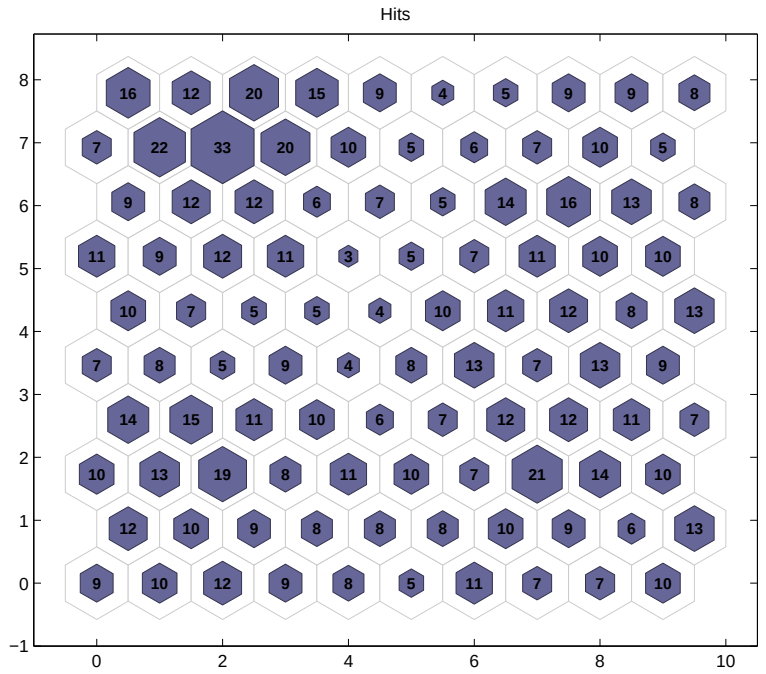


location of prototype vectors

SOM example in Matlab (2)



SOM topology (10×10 neurons)



winning units (number of hits)

Additional optional material

- E. Oja, "A simplified neuron model as a principal component analyzer", *J. Math. Biology*, 15: 267-273, 1982
- E. Oja, "Principal components, minor components and linear neural networks", *Neural Networks*, Vol 5, pp. 927-935, 1992
- T. Kohonen, "The self-organizing map", *Proceedings of the IEEE*, Vol.78, No.9, pp. 1464-1480, 1990
- T. Kohonen, S. Kaski, K. Lagus, J. Salojarvi, J. Honkela, V. Paatero, A. Saarela, "Self organization of a massive document collection", *IEEE Transactions on Neural Networks*, Vol. 11, No. 3, pp. 574-585, 2000
- M. Scholz, M. Fraunholz, J. Selbig, "Nonlinear principal component analysis: neural network models and applications", In Principal Manifolds for Data Visualization and Dimension Reduction, (Eds. A.N. Gorban et al.), Vol. 58 of LNCSE, pp 44-67, Springer Berlin Heidelberg, 2007.