(Non-linear) dimensionality reduction

Jiří Kléma<br>Department of Computer Science, Czech Technical University in Prague

http://cw.felk.cvut.cz/wiki/courses/a4m33sad/start

## Outline

- motivation, task definition,
- linear approaches such as PCA
- and its non-linearization: kernel PCA,
- distance preserving approaches
- multidimensional scaling,
- Isomap,
- locally linear embedding,
- tSNE,
- self-organizing maps
- a vector quantization approach,
- their relation to k-means clustering,


## Task definition

- Input: $\mathbf{X}=\left\{\mathbf{x}_{\mathbf{i}}\right\}_{i=1}^{m} \subset \mathcal{X}$ of dimension $D$ (typically $\mathbb{R}^{D}$ ),
- assumed: $\mathbf{X}$ at least approximately lies on a manifold with $d<D$,
- output:
- a transformed space $\mathcal{T}$ of dimension $L$,
- dimensionality reduction mapping $\mathbf{F}: \mathcal{X} \rightarrow \mathcal{T}$,
- reconstruction mapping $\mathbf{f}: \mathcal{T} \rightarrow \mathcal{M} \subset \mathcal{X}$,
- such that:
- $L<D, L$ is as small as possible, at best $L=d$ (the intrinsic dimension),
- the manifold approximately contains all the sample points

$$
\left\{\mathbf{x}_{\mathbf{i}}\right\}_{i=1}^{m} \underset{\sim}{\subset} \mathcal{M} \stackrel{\text { def }}{=} f(\mathcal{T})
$$

- or alternatively, the reconstruction error of the sample is small

$$
E_{d}(\mathbf{X}) \stackrel{\text { def }}{=} \sum_{i=1}^{m} d\left(\mathbf{x}_{\mathbf{i}}, \mathbf{f}\left(\mathbf{F}\left(\mathbf{x}_{\mathbf{i}}\right)\right)\right.
$$

## Example of a manifold and its mapping

- A spiral of radius $R$ and step $s$
- 1D non-linear manifold in $\mathbb{R}^{3}$,
- no noise, the given $f(t)$ guarantees zero reconstruction error.


Carreira-Perpinan: A Review of Dimension Reduction Techniques

## Motivation

- Visualize the data and understand them (typically project into 2 or 3-D),
- compress the data to minimize storage and retrieval cost,
- identify hidden causes/latent variables that govern the process under study,
- learn in the lower-dimensional space
- possibly obtain better results with fewer training samples in shorter time,
- a lot of work has already been done during the projection.




## The challenges of high-dimensional spaces

- The curse of dimensionality
- in the absence of simplifying assumptions, the sample size needed to estimate a function with $D$ variables to a given degree of accuracy grows exponentially with $D$,
- the geometry of high-dimensional spaces
- the empty space phenomenon,
- guess what is the ratio of the volumes of unit hypersphere and unit hypercube ...



## A brief review of PCA

- PCA could be seen as fitting a (hyper)ellipsoid to the data
- the new axes have the direction of the highest variance,
- they match the axes of the encapsulating/confidence ellipsoid.

- What happens in terms of covariances and redundancy?
- left image: $\sigma_{Y}^{2}>\sigma_{X Y}^{2}>\sigma_{X}^{2}$,
- right image: $\sigma_{P 1}^{2} \gg \sigma_{P 2}^{2}, \sigma_{P_{1} P_{2}}^{2}=0$,
- in general, PCA diagonalizes the covariance matrix,
- in other words, PCA removes linear relationship (redundancy) between variables.


## A brief review of PCA

- For $\mathbf{X}$ with zero centered variables

$$
\sum_{i=1}^{m} \mathbf{x}_{\mathbf{i}}=0
$$

- the covariance matrix can be computed as follows

$$
\mathbf{C}_{\mathbf{X}}=\frac{1}{m} \sum_{i=1}^{m} \mathbf{x}_{\mathbf{i}} \mathbf{x}_{\mathbf{i}}^{T}=\frac{1}{m} \mathbf{X}^{T} \mathbf{X}
$$

- by definition, PCA constructs a space transformation matrix $\mathbf{P}_{D \times D}$, such that

$$
\mathbf{X P}=\mathbf{T} \text { and } \mathbf{C}_{\mathbf{T}} \text { is a diagonal matrix }
$$

- as $\mathbf{P}$ is not known, $\mathbf{C}_{\mathbf{T}}$ cannot be calculated and diagonalized directly,
- instead, $\mathbf{C}_{\mathbf{T}}$ can be expressed in terms of $\mathbf{C}_{\mathbf{X}}$ and $\mathbf{P}$

$$
\begin{aligned}
\mathbf{C}_{\mathbf{T}} & =\frac{1}{m} \mathbf{T}^{T} \mathbf{T}=\frac{1}{m}(\mathbf{X P})^{T}(\mathbf{X P})= \\
& =\frac{1}{m} \mathbf{P}^{T}\left(\mathbf{X}^{T} \mathbf{X}\right) \mathbf{P}=\mathbf{P}^{T} \mathbf{C}_{\mathbf{X}} \mathbf{P}
\end{aligned}
$$

## A brief review of PCA

- any real symmetric matrix is diagonalized by a column matrix of its eigenvectors $\mathbf{E}$,
- $\mathrm{C}_{\mathrm{X}}$ is real and symmetric, it follows that

$$
\mathbf{C}_{\mathbf{X}}=\mathbf{E D E}^{T}
$$

- the only trick is to select $\mathbf{P}$ to be a matrix where each column $\mathrm{p}_{\mathrm{i}}$ is an eigenvector of $\mathbf{C}_{\mathrm{X}}$

$$
\mathbf{P}=\mathbf{E}
$$

- we also know that any orthogonal matrix has the same inverse and transpose,
- the above-selected $\mathbf{P}$ is necessarily orthogonal

$$
\mathbf{P}^{T} \mathbf{P}=\mathbf{I} \Rightarrow \mathbf{P}^{-1}=\mathbf{P}^{T}
$$

- then, it is easy to show that $\mathbf{P}$ diagonalizes $\mathbf{C}_{\mathbf{T}}$

$$
\begin{aligned}
\mathbf{C}_{\mathbf{T}} & =\mathbf{P}^{T} \mathbf{C}_{\mathbf{X}} \mathbf{P}=\mathbf{P}^{T}\left(\mathbf{E D E}^{T}\right) \mathbf{P}= \\
& =\left(\mathbf{P}^{T} \mathbf{P}\right) \mathbf{D}\left(\mathbf{P}^{T} \mathbf{P}\right)=\left(\mathbf{P}^{-1} \mathbf{P}\right) \mathbf{D}\left(\mathbf{P}^{-1} \mathbf{P}\right)=\mathbf{D}
\end{aligned}
$$

- PCA is solved by finding the eigenvectors of $\mathrm{C}_{\mathbf{X}}$.


## A brief review of PCA

- what happens when $D$ is large?
- consider images, the color of each pixel is a feature, megapixel resolution,
- large $\mathbf{C}_{\mathbf{X}}$, unfeasible computation of its eigenvectors,
- provided that $m$ is reasonable ( $m \ll D$ ), we can employ the following trick,
- instead of the original eigenvector decomposition

$$
\frac{1}{m} \mathbf{X}^{T} \mathbf{X} \mathbf{u}_{\mathbf{k}}=\lambda_{k} \mathbf{u}_{\mathbf{k}}
$$

- we will consider

$$
\frac{1}{m} \mathbf{X X}^{T} \mathbf{v}_{\mathbf{k}}=\gamma_{k} \mathbf{v}_{\mathbf{k}}
$$

- and multiply both sides by $\mathbf{X}^{T}$

$$
\frac{1}{m} \mathbf{X}^{T} \mathbf{X} \mathbf{X}^{T} \mathbf{v}_{\mathbf{k}}=\gamma_{k} \mathbf{X}^{T} \mathbf{v}_{\mathbf{k}}
$$

- it is obvious that the substitution $\mathbf{X}^{T} \mathbf{v}_{\mathbf{k}}=\mathbf{u}_{\mathbf{k}}$ and $\gamma_{k}=\lambda_{k}$ matches the original eigenvector decomposition formula,
- PCA can be solved by decomposition of the $m \times m$ scalar-product matrix.


## kernel PCA - the idea behind

- Introduce an intermediate feature space $\mathcal{U}$
$-\mathcal{X} \rightarrow \mathcal{U} \rightarrow \mathcal{T}, \mathcal{U}$ linearizes the original manifold.

http://www.research.rutgers.edu


## kernel PCA

- We choose a (non-linear) feature space transformation

$$
\phi: \mathcal{X} \rightarrow \mathcal{U}
$$

- the transformation is implicit, we only know the kernel function

$$
\forall \mathbf{x}_{\mathbf{i}}, \mathbf{x}_{\mathbf{j}} \in \mathcal{X}: \mathbf{K}\left(\mathbf{x}_{\mathbf{i}}, \mathbf{x}_{\mathbf{j}}\right):=\left\langle\phi\left(\mathbf{x}_{\mathbf{i}}\right), \phi\left(\mathbf{x}_{\mathbf{j}}\right)\right\rangle=\phi\left(\mathbf{x}_{\mathbf{i}}\right)^{T} \phi\left(\mathbf{x}_{\mathbf{j}}\right)
$$

- as with the PCA, we will assume the covariance matrix, now in the transformed space

$$
\mathbf{C}_{\mathbf{U}}=\frac{1}{m} \sum_{i=1}^{m} \phi\left(\mathbf{x}_{\mathbf{i}}\right) \phi\left(\mathbf{x}_{\mathbf{i}}\right)^{T}
$$

- note that the data are assumed to be centered

$$
\sum_{i=1}^{m} \phi\left(\mathbf{x}_{\mathbf{i}}\right)=0
$$

- similarly to PCA, we will find $\mathrm{C}_{\mathrm{U}}$ eigenvectors v to decorrelate variables in $\mathcal{T}$

$$
\mathbf{C}_{\mathbf{U}} \mathbf{v}=\lambda \mathbf{v} \rightarrow \frac{1}{m} \sum_{i=1}^{m} \phi\left(\mathbf{x}_{\mathbf{i}}\right) \phi\left(\mathbf{x}_{\mathbf{i}}\right)^{T} \mathbf{v}=\lambda \mathbf{v}
$$

## kernel PCA

- $\phi\left(\mathbf{x}_{\mathbf{i}}\right)$ are not available, we need to replace them by $\mathbf{K}$,
- for $\lambda \geq 0$, $\mathbf{v}$ 's are in the span of $\phi\left(\mathbf{x}_{\mathbf{i}}\right)$,
- they can be written as linear combination of the object images

$$
\mathbf{v}=\sum_{i=1}^{m} \alpha_{i} \phi\left(\mathbf{x}_{\mathbf{i}}\right)
$$

- we will substitute for $\mathbf{v}$ into the eigenvector formula (the last in the previous slide)

$$
\lambda \sum_{j=1}^{m} \alpha_{j} \phi\left(\mathbf{x}_{\mathbf{j}}\right)=\frac{1}{m} \sum_{i=1}^{m} \phi\left(\mathbf{x}_{\mathbf{i}}\right) \phi\left(\mathbf{x}_{\mathbf{i}}\right)^{T} \sum_{j=1}^{m} \alpha_{j} \phi\left(\mathbf{x}_{\mathbf{j}}\right)
$$

- and use the trick to introduce the dot product, we will multiply by $\phi\left(\mathbf{x}_{\mathbf{k}}\right)^{T}$

$$
\lambda \sum_{j=1}^{m} \alpha_{j} \phi\left(\mathbf{x}_{\mathbf{k}}\right)^{T} \phi\left(\mathbf{x}_{\mathbf{j}}\right)=\frac{1}{m} \sum_{j=1}^{m} \alpha_{j} \sum_{i=1}^{m}\left(\phi\left(\mathbf{x}_{\mathbf{k}}\right)^{T} \phi\left(\mathbf{x}_{\mathbf{i}}\right)\right)\left(\phi\left(\mathbf{x}_{\mathbf{i}}\right)^{T} \phi\left(\mathbf{x}_{\mathbf{j}}\right)\right)
$$

- the kernel function replaces all the occurrences of $\phi$, when iterating $\forall k=1 \ldots m$ we obtain

$$
\lambda \mathbf{K} \alpha=\frac{1}{m} \mathbf{K}^{2} \alpha
$$

- to diagonalize $\mathbf{C}_{\mathbf{T}}$, we solve the eigenvalue problem for the kernel matrix

$$
m \lambda \alpha=\mathbf{K} \alpha
$$

## kernel PCA

- the last issue is to extract the principal components, the final object images,
- i.e., the projections of $\phi\left(\mathbf{x}_{\mathbf{i}}\right)$ onto the eigenvectors in $\mathcal{U}$

$$
t_{i k}=\mathbf{v}_{\mathbf{k}}^{T} \phi\left(\mathbf{x}_{\mathbf{i}}\right)=\sum_{j=1}^{m} \alpha_{j} \phi\left(\mathbf{x}_{\mathbf{j}}\right)^{T} \phi\left(\mathbf{x}_{\mathbf{i}}\right)=\sum_{j=1}^{m} \alpha_{j} \mathbf{K}\left(\mathbf{x}_{\mathbf{i}}, \mathbf{x}_{\mathbf{j}}\right)
$$

- note that both the intermediate factors $\mathbf{v}_{\mathbf{k}}$ and $\phi\left(\mathbf{x}_{\mathbf{i}}\right)$ remain implicit!



## Remarks on kernel PCA

- kernel PCA works for non-linear manifolds
- effectively compresses them,
- its complexity does not grow with the dimensionality of $\mathcal{U}$
- one can work with a large number of components too, i.e., increase the dimension $(L>D)$,
- in PCA $L \leq D$
- it can pay-off in subsequent classification,
- kernel matrix $\mathbf{K}$ grows quadratically with the number of data points $m$
- for large data sets with small dimensionality $(m \gg D)$ more expensive than PCA,
- one may need to (properly) subsample the data,
- from the optimization point of view cannot get trapped in local minima,
- unlike PCA cannot reconstruct objects from their principal components
$-\mathbf{f}$ is not available.


## Multidimensional scaling (MDS)

- The main idea
- points close together in $\mathcal{X}$ should be mapped close together in $\mathcal{T}$,
- minimizes the stress function

$$
\operatorname{stress}(\mathbf{T}, f)=\sqrt{\frac{\sum_{i, j=1}^{m}\left(f\left(\delta_{i j}\right)-d_{i j}\right)^{2}}{\sum_{i, j=1}^{m} d_{i j}^{2}}}
$$

$-\delta_{i j}=d_{\mathcal{X}}\left(\mathbf{x}_{\mathbf{i}}, \mathbf{x}_{\mathbf{j}}\right), d_{i j}=d_{\mathcal{T}}\left(\mathbf{t}_{\mathbf{i}}, \mathbf{t}_{\mathbf{j}}\right)$ - typically Euclidean,
$-f$ is a proximity transformation function (e.g., identity, monotonic $\rightarrow$ metric, ordinal),

- whole class of methods that differ in
- the method for calculation of proximities $\delta$,
- the parametrization of stress function,
- the method that minimizes the stress function (e.g., gradient descent, Newton).


## Geodesic distance

- What does Euclidean distance say about a non-linear manifold?
- We shall better preserve geodesic distance between points
- a minimum of the length of a path joining both points that is contained in the manifold,
- these paths are called geodesics,
- as a result, we unfold the manifold.


Carreira-Perpinan: A Review of Dimension Reduction Techniques

## Isomap [Tenenbaum et al., 1998]

- Classical MDS with geodesic distance
(A) Euclidean vs geodesic distance to express intrinsic similarity in the input space,
(B) the neighborhood graph $G(K=5, m=1000)$ and its shortest path (red) as an approximation to the true geodesic path,
(C) 2D embedding, which best preserves the shortest path distances in $G$, the straight lines in the embedding (blue) represent simpler and cleaner approximations to the true geodesic paths than do the corresponding graph paths.


Tenenbaum et al.: A Global Geometric Framework for Nonlinear Dimensionality Reduction

## Isomap [Tenenbaum et al., 1998]

- The Isomap algorithm
- determine the nearest neighbors
* all points in a fixed radius or K-nearest neighbors,
- construct a neighborhood graph
* each point gets connected to its neighbors,
* edge length equals the Euclidean distance between the points,
- compute the shortest paths between all pairs of points
* Floyd's or Dijkstra's algorithm $\left(\mathcal{O}\left(m^{3}\right)\right.$ resp. $\left.\mathcal{O}\left(K m^{2} \log (m)\right)\right)$,
* could be time consuming and result in a large non-sparse matrix,
* use quantization to compress the graph,
- construct a lower dimensional embedding
* use classical MDS.


## Isomap [Tenenbaum et al., 1998]

- Isomap has provable convergence guarantees,
- given the infinite (sufficient) input data, the method perfectly recovers the original distances
- impractical concerning the next property,
- otherwise, the geodesic distance can overestimate the true distance,
- cubic complexity in the number of objects can be too much
- selection of $M$ reference vectors can reduce the time to $\mathcal{O}\left(M^{3}\right)$,
- we have to guarantee that the manifold is uniformly sampled,
- finding a proper value of $K$ is not easy
- too small/large $K$ : insufficient graph/improper connections originate.


Swiss roll
ideal projection


Isomap outcome

## Locally Linear Embedding (LLE) [Roweis, Saul, 2000]

- Remember: manifold is a topological space that is locally Euclidean
- we can locally fit (linear) models using Euclidean distance,
- when compiled they create a global model.


Roweis, Saul: Nonlinear Dimensionality Reduction by LLE.

## Locally Linear Embedding (LLE) [Roweis, Saul, 2000]

## - The LLE algorithm

- each data point and its neighbors lie close to a locally linear patch of the manifold,
- each point can be written as a linear combination of its neighbors,
- $m$ local models, the weights chosen to minimize the reconstruction error

$$
\hat{\mathbf{x}}_{\mathbf{i}}=\sum_{j \in \mathcal{N}\left(\mathbf{x}_{\mathbf{i}}\right)} w_{i j} \mathbf{x}_{\mathbf{j}} \text { such that } \sum_{i=1}^{m}\left\|\mathbf{x}_{\mathbf{i}}-\hat{\mathbf{x}}_{\mathbf{i}}\right\|^{2} \text { is minimized and } \sum_{j \in \mathcal{N}\left(\mathbf{x}_{\mathbf{i}}\right)} w_{i j}=1
$$

- the same weights should reconstruct the point in $L$ dimensions
* the weights characterize the intrinsic geometric properties of each neighborhood,
- global embedding fits the positions $\mathrm{t}_{\mathrm{i}}$ in the low-dimensional space
* we minimize the embedding cost function, the weights are fixed

$$
\hat{\mathbf{t}}_{\mathbf{i}}=\sum_{j \in \mathcal{N}\left(\mathbf{x}_{\mathbf{i}}\right)} w_{i j} \mathbf{t}_{\mathbf{j}} \text { such that } \sum_{i=1}^{m}\left\|\mathbf{t}_{\mathbf{i}}-\hat{\mathbf{t}}_{\mathbf{i}}\right\|^{2} \text { is minimized. }
$$

## Locally Linear Embedding (LLE) [Roweis, Saul, 2000]

- The ultimate case of piecewise linear modelling
- approximation of the manifold by a combination of linear models,
- in here, we have a linear model for each object,
- a special case of kernel PCA constructing a data-dependent kernel matrix
- for some problems it is difficult to find a kernel for kernel PCA,
- advantages
- efficient for large datasets, optimization does not involve local minima,
- single parameter to tune $(K)$,
- invariant to scaling, rotation and translation,
- disadvantages
- improper for representing future data,
- can be unstable in sparse areas of the input space,
- tends to collapse a lot of instances near the origin of $\mathcal{T}$.


## t-Distributed Stochastic Neighbor Embedding (t-SNE)

- distance preserving visualization technique [van der Maaten, 2008] (like e.g., MDS),
- puts emphasis on preserving small pairwise distances between objects,
- large distances allowed to be modelled as being larger,
- as a result, two essential characteristics
- the local data structures retained,
- ability to reveal global structure such as the presence of clusters at several scales.



## t-Distributed Stochastic Neighbor Embedding (t-SNE)

- The key ideas are in the design of the stress function driving gradient descent search
- convert Euclidean distances in both spaces into joint probabilities,
- $p_{i j}$ in the original space $\mathcal{X}$ and $q_{i j}$ in the reduced space $\mathcal{T}$,
- minimize their Kullback-Leibler divergence

$$
S=K L(P \| Q)=\sum_{i} \sum_{j \neq i} p_{i j} \log \frac{p_{i j}}{q_{i j}}
$$

- The first "trick" lies in asymmetry of KL divergence
- large $p_{i j}$ modeled by small $q_{i j} \rightarrow$ big penalty!
- small $p_{i j}$ modeled by large $q_{i j} \rightarrow$ small penalty!
- tends to preserve large $p_{i j}$ 's and thus small distances in the original space.


## t-Distributed Stochastic Neighbor Embedding (t-SNE)

- The other "tricks" consist in definition of $p_{i j}$ and $q_{i j}$
- the empirical probability that an object $j$ is a neighbor of an object $i$

$$
p_{j \mid i}=\frac{\exp \left(-\left\|\mathbf{x}_{\mathbf{i}}-\mathbf{x}_{\mathbf{j}}\right\|^{2} / 2 \sigma_{i}\right)}{\sum_{k \neq i}\left(-\left\|\mathbf{x}_{\mathbf{i}}-\mathbf{x}_{\mathbf{k}}\right\|^{2} / 2 \sigma_{i}\right)}
$$

- i.e., it is normally distributed wrt their distance (and decreases quickly with it),
$-\sigma_{i}$ is the kernel bandwidth,
- $\sigma_{i}$ is locally adjusted so that a fixed number of objects falls in mode of the Gaussian,
- the symmetric joint probability

$$
p_{i j}=\frac{p_{j \mid i}+p_{i \mid j}}{2 m}
$$

## t-Distributed Stochastic Neighbor Embedding (t-SNE)

- In the reduced space $\mathcal{T}$ we permit higher probabilities for larger distances,
- the normal distribution used in $p_{i j}$ turns into the heavy-tailed t-distribution in $q_{i j}$

$$
q_{i j}=\frac{\left(1+\left\|\mathbf{t}_{\mathbf{i}}-\mathbf{t}_{\mathbf{j}}\right\|^{2}\right)^{-1}}{\sum_{k \neq l}\left(1+\left\|\mathbf{t}_{\mathbf{k}}-\mathbf{t}_{\mathbf{l}}\right\|^{2}\right)^{-1}}
$$

- in KL divergence, $p_{i j}$ and $q_{i j}$ shall agree as much as possible, but they may map to different distances in both the spaces,
- as a result, a moderate distance in the high-dimensional space can be faithfully modeled by a much larger distance in the map,
- the reduced map gets insensitive to distant points (they can be placed to many places without big changes in $q_{i j}$ ).


Tails in normal and student distributions.

## t-Distributed Stochastic Neighbor Embedding (t-SNE)

- The overall picture
- the gradient descent gradually minimizes the stress function for the individual objects

$$
\frac{\partial S}{\partial \mathbf{t}_{\mathbf{i}}} \propto \sum_{j \neq i}\left(p_{i j}-q_{i j}\right)\left(1+\left\|\mathbf{t}_{\mathbf{i}}-\mathbf{t}_{\mathbf{j}}\right\|^{2}\right)^{-1}\left(\mathbf{t}_{\mathbf{i}}-\mathbf{t}_{\mathbf{j}}\right)
$$

- all the other objects get connected via springs that are either stretched or compressed,
- the resultant summed force tells us where to move the point in every gradient update.



## Self-organizing map (SOM)

- unsupervised neural network producing a low dimensional (typically 2D) discretized map
- the map is composed of neurons,
- the neurons are mutually linked by a neighborhood relationship (e.g., make a grid),
- the neurons compete for the right to respond to the individual input objects,
- the map preserves the topology of the input space
- dimension reduction = the map from the position of the corresponding neuron in the input space to its (discrete) position in the grid.


Borgelt: XSOM visualization

## SOM learning algorithm

1. Initialize weight vectors of all the neurons $\mathbf{w}_{i}^{0} \in \mathcal{X}\left(\mathbb{R}^{D}\right), \forall i=1, \ldots, N$

- randomly or rather sample evenly from the subspace spanned by the largest principal component eigenvectors,

2. resample the instance set $\mathbf{X}$

- A: sample systematically, B: permute the instance set, C: randomly (bootstrap),

3. get the next instance $\mathbf{x}_{i} \in \mathbb{R}^{D}$ from the resample,
4. find the best matching unit (BMU), i.e., the neuron nearest to $\mathbf{x}_{i}$,
5. change the weights of neurons in the neighborhood of BMU (including BMU)

$$
\mathbf{w}_{i}^{t+1}=\mathbf{w}_{i}^{t}+\alpha^{t} e^{-\frac{d\left(w_{i}^{t} . B M U\right)}{\left.2\left(\sigma^{t}\right)^{2}\right)}}\left(\mathbf{x}_{i}-\mathbf{w}_{i}^{t}\right)
$$

- both the neighborhood size $\sigma^{t}$ and the learning rate $\alpha^{t}$ decrease in time,
- $d$ is the distance in terms of the neighborhood relationship (e.g., the grid distance),

6. go to step 3 (or 2 when the sample is finished) until the given number of cycles is reached.

## SOM learning algorithm - illustration



## How to visualize and interpret SOM?

- U-matrix (unified distance matrix)
- visualizes the distances between the neurons,
- dark color between a pair of neurons corresponds to a large distance of their weight vectors,
- light areas are candidate clusters while dark areas are interpreted as cluster separators.


Hollmen: U-matrix


## Relation to k-means

- The same competitive learning principle
- the centroids/neurons compete for the right to respond to the individual input objects,
- $k$-means employs the winner takes all principle (only the nearest centroid wins),
- in SOM it is the BMU and its neighbors who win (for $\sigma \rightarrow 0$ equal to k -means),
- The same vector quantization approach
- minimize the distance between objects and their representatives,
- $k$-means concerns the nearest centroid only, SOM deals with the whole neighborhood,
- The main difference lies in
- the most frequent way of using them (clustering versus dimensionality reduction),
- $k$ corresponds to the number of clusters, the number of neurons typically much larger.
- SOM can be seen as constrained $k$-means.


## Summary - dimensionality reduction, manifold learning

- Difficult problem namely for the curse of dimensionality
- huge sample sizes needed to guarantee reasonable parameter estimates, non-empty neighborhoods, etc.,
- the intrinsic dimensionality estimation is not reliable,
- strong assumptions greatly simplify the task,
- the key role of PCA has not been undermined by any non-linear method yet
- they work for well-sampled smooth manifolds, but not necessarily for real data,
- besides the curse of dimensionality, the problems could be caused by insufficiency of objective functions or numerical problems during their optimization,
- there is a large pool of non-linear reduction methods,
- the key properties are effectivity and efficiency including convergence,
- other issues
- setting hyperparameters?
- implicit/explicit definition of $\mathbf{F}$ and $\mathbf{f}$,
- additivity - can I drop a coordinate from $L$ mapping to obtain $L$ - 1 mapping?


## Recommended reading, lecture resources

:: Reading

- Jonathon Shlens: A Tutorial on Principal Component Analysis.
- Google research, 2014,
- http://arxiv.org/pdf/1404.1100.pdf,
- Scholkopf, Smola, Müller: Kernel Principal Component Analysis.
- Artificial Neural Networks, 1997,
- http://link.springer.com/chapter/10.1007/BFb0020217,
- Carreira-Perpinan: A Review of Dimension Reduction Techniques.
- Tech. report, University of Sheffield, 1997,
- http://www.pca.narod.ru/DimensionReductionBrifReview.pdf,
- van der Maaten, Hinton: Visualizing High-Dimensional Data Using t-SNE.
- Journal of Machine Learning Research 9(Nov):2579-2605, 2008,
- https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf.

