Data Analysis, Lecture 5 : Data visualization - non-linear methods

Jérémie Sublime

Data Analysis - Lecture 5

Data visualization: non-linear methods

Dr. Jérémie Sublime, Eng., MEng., PhD, HDR

LISITE Laboratory - DaSSIP Team - ISEP
LIPN - CNRS UMR 7030

jeremie.sublime@isep.fr

Plan

Outline

Limits of linear methods (1/2)

Techniques such as SVD, PCA and their variants perform a global transformation of the data using basic operations:

- Rotation
- Translation
- Rescaling

The problematic of non-linearity

- Using the 3 operations, linear methods assume that most of the information in the data is contained in a linear subspace.
- This raises the question of how to deal with data that are actually embedded in a non-linear subspace (a low-dimensional manifold).

Limits of linear methods (2/2)

Linear methods such as PCA cannot discover the structure of a data set with a non-linear shape.

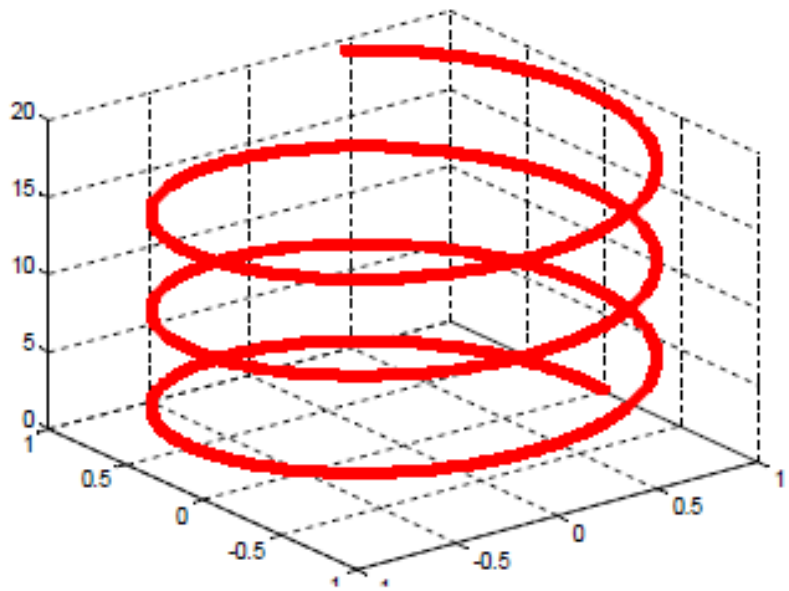


Figure: Example of a spiral data set that could be modeled in 2D or even 1D, but that PCA and non-linear methods cannot handle.

Euclidian distance VS Geodesic distance

- Euclidian based distances are one of the main problem when dealing with data embedded in a non-linear subspace.
- Custom distances embedded in the low-dimensional manifold, such as the **geodesic distance**, can help dealing with such data.

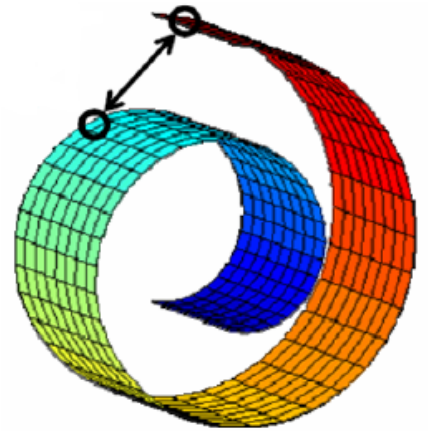


Figure: Euclidian distance

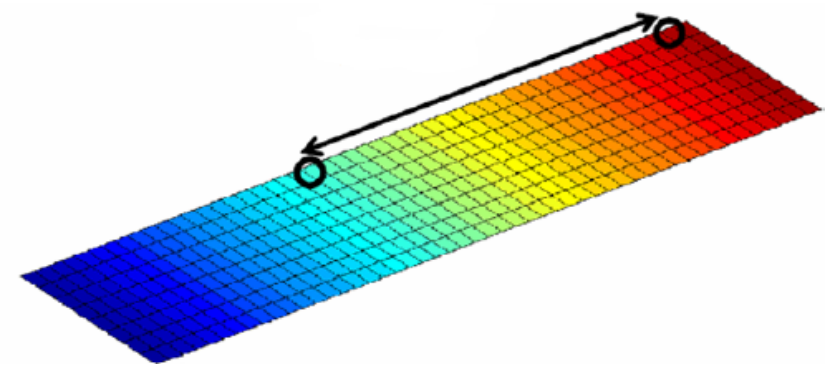


Figure: Geodesic distance

Outline

ISOMAP: Principle

The **Isometric feature mapping** (Isomap) [Tenebaum et al. 2000] is a reduction method that can handle non-linear subspaces. It is based on the following principles:

- For neighboring samples (close data points), the Euclidian distance provides a good approximation of the geodesic distance.
- For distant points, the geodesic distance can be approximated using an Euclidian distance-based **neighborhood graph** of the data and searching the shortest path between the two points.

ISOMAP: Algorithm

1. Build a distance neighborhood graph G using the Euclidian distance in the input space: This can be done using KNN, or by connecting the points within a radius ϵ.
2. Approximate the geodesic distance between all data points: Compute the shortest path between all points of the graph using **Dijkstra** algorithm or **Floyd** algorithm.
3. Apply the **Multi-dimensional scaling** (MDS) algorithm to the geodesic distance matrix from Step 2.

Remark

The Isomap algorithm can be seen as a non-linear variant of the MDS algorithm.

ISOMAP: Algorithmic complexity

The main issue of the Isomap algorithm is that it can be quite slow. For a data set of size N, an input space of dimension D, an output space of dimension d, and considering a neighborhood of size k in the first step, we have:

- Step 1: K-nearest neighbors $O(N^2 D)$
- Step 2: Djikstra $O(N^2 log(N) + N^2 k)$, Floyd $O(N^3)$
- Step 3: MDS algorithm $O(N^2 d)$

Multi-dimensional scaling: Algorithm

- Choose an objective function J and a step parameter $0 < \eta < 1$.
- Compute the distance matrix δ if it is not provided.
- Initialize the projected points randomly and compute the corresponding projected distance matrix.
- Optimize the position of the projected points until convergence using gradient descent

ISOMAP: The "Swiss roll" Example (1/3)

- The "swiss roll" data set contains 20000 points.
- We will show thereafter the development of the Isomap algorithm on this data set.

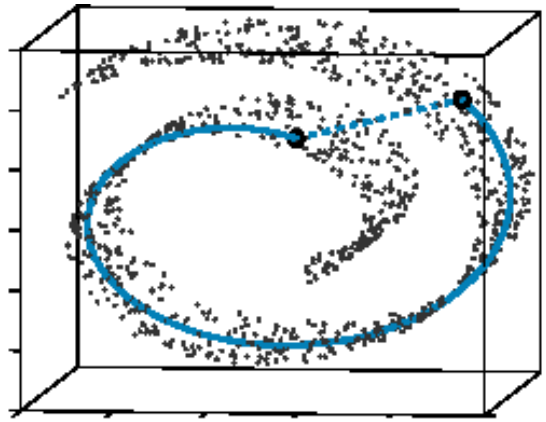


Figure: A 1000 points extract of the Swiss roll data set, with an example of Euclidian and Geodesic distances.

ISOMAP: The "Swiss roll" Example (2/3)

In the figure below, we show 3 illustrations of how the Isomap algorithm works on the Swiss roll data set with the following parameters:

- K-nearest neighbors (K=7)
- Approximation of the Geodesic distance: Djikstra algorithm

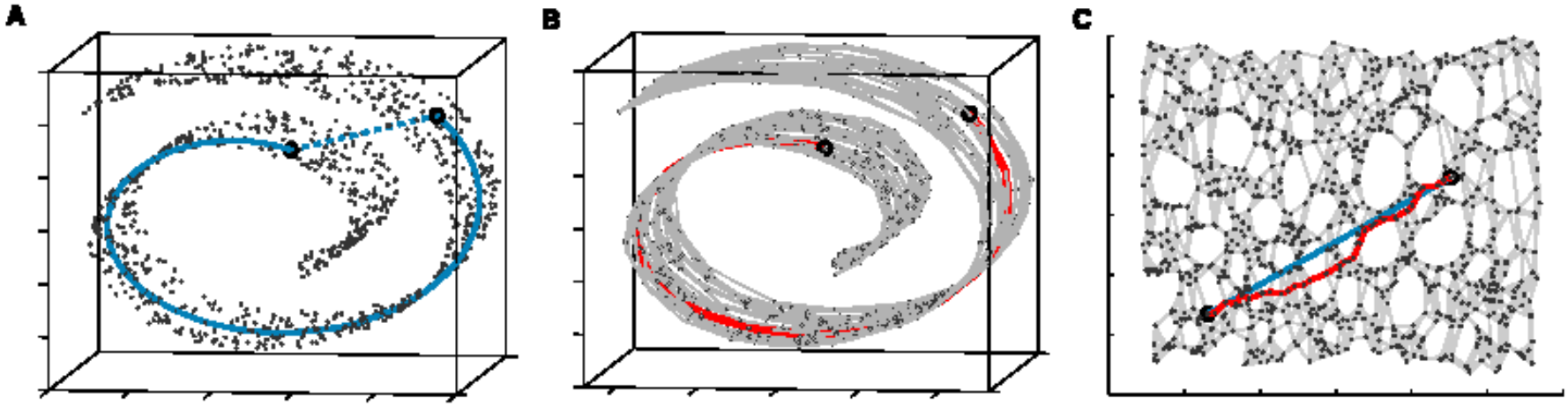


Figure: A 1000 points extract of the Swiss roll data set: The Euclidian distance is in dotted blue, the Geodesic distance in solid blue, and the approximated Geodesic distance is in red. [Tenebaum et al. 2000]

ISOMAP: The "Swiss roll" Example (3/3)

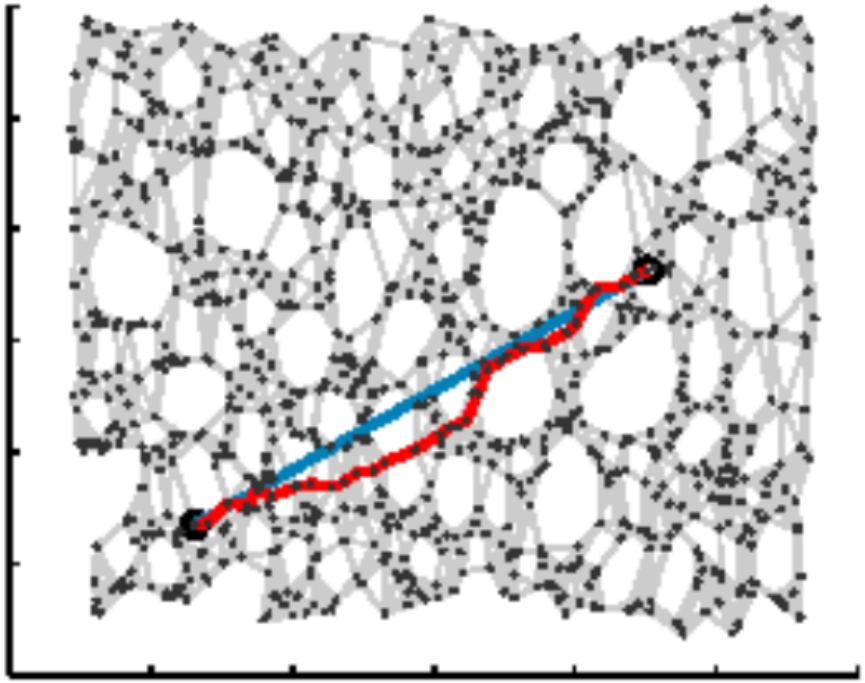


Figure: Final manifold in 2D that preserves pairwise geodesic distances. [Tenebaum et al. 2000]

ISOMAP: Images Example (1/2)

Figure: For each image we have $64 \times 64 = 4096$ pixels

ISOMAP: Images Example (2/2)

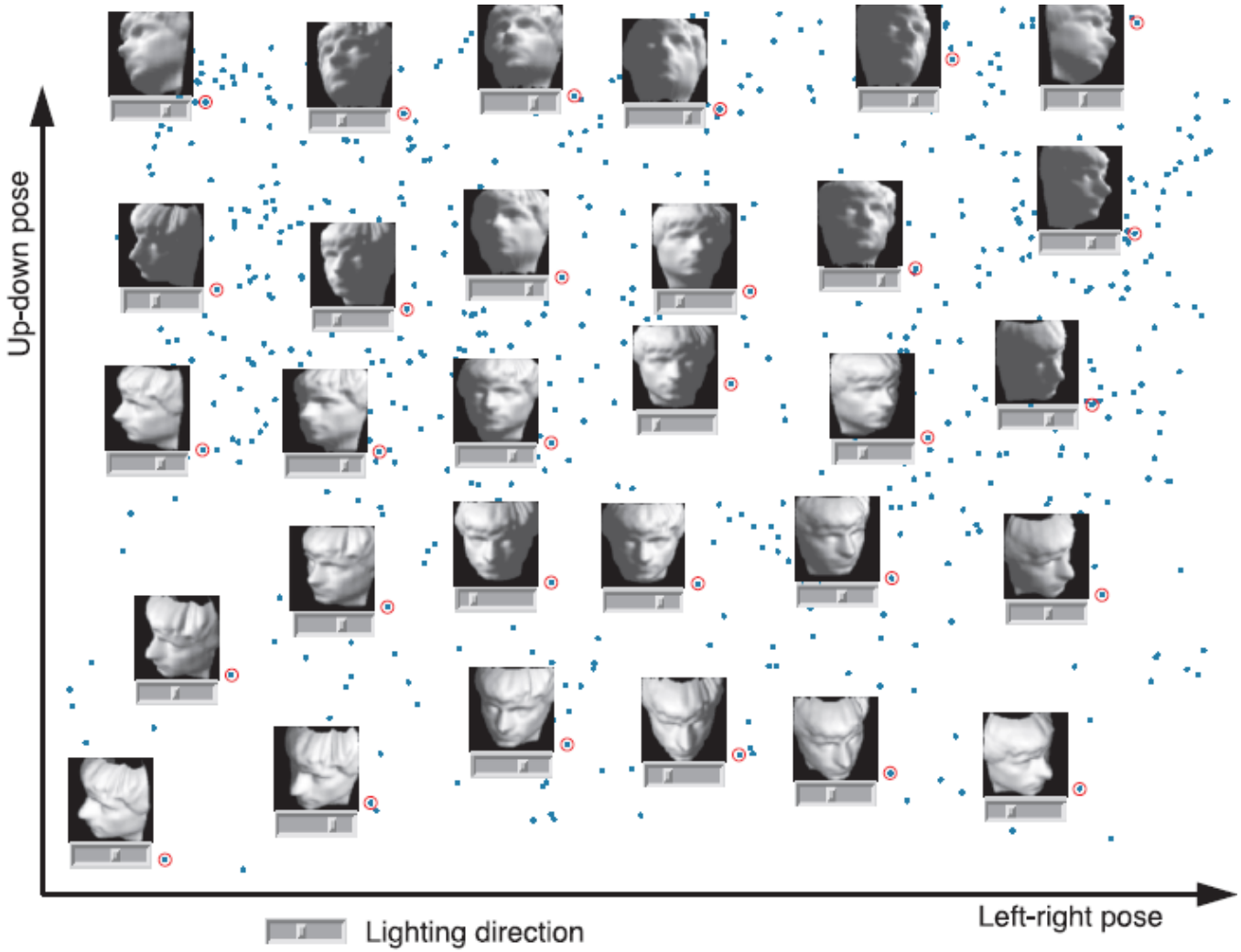
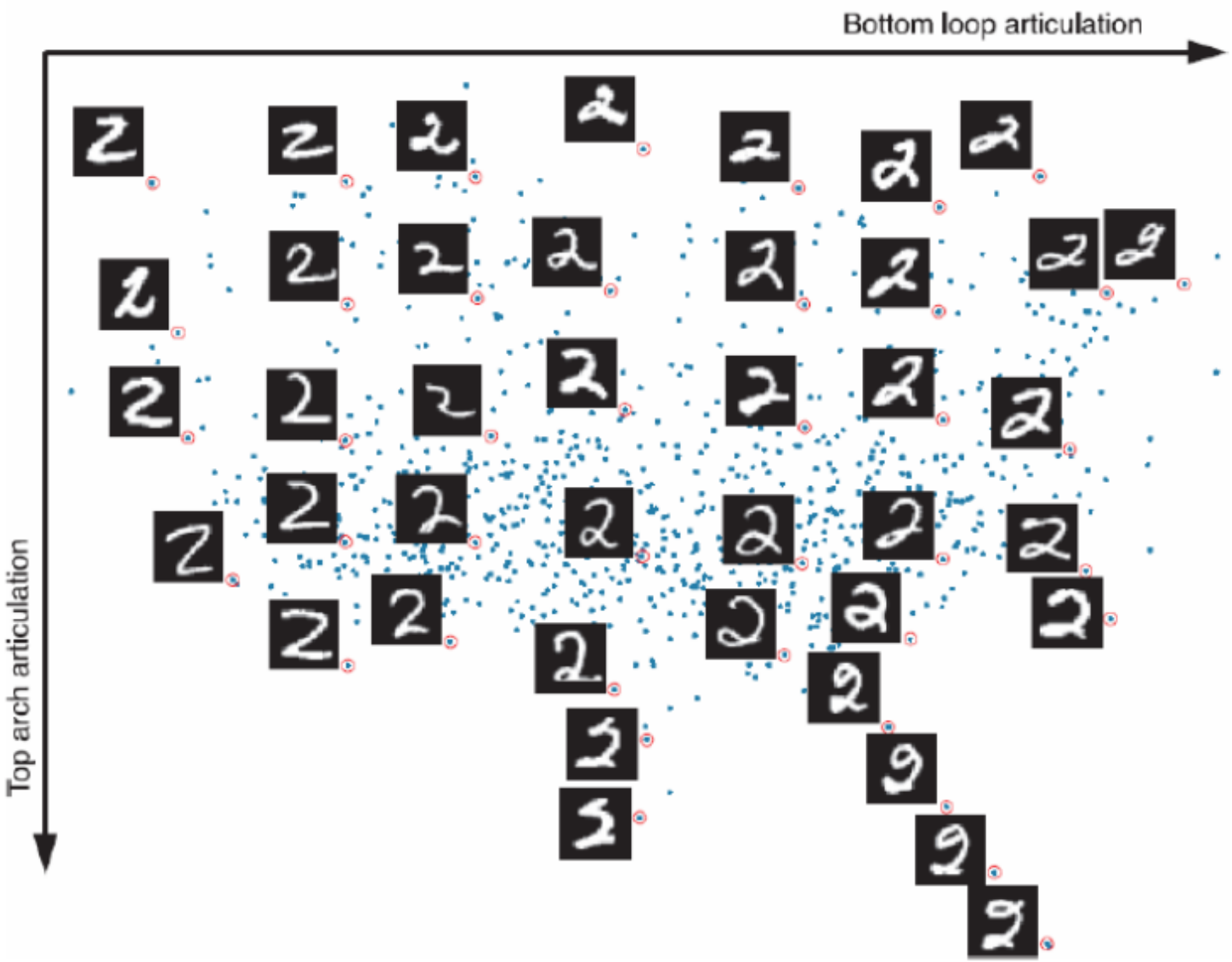


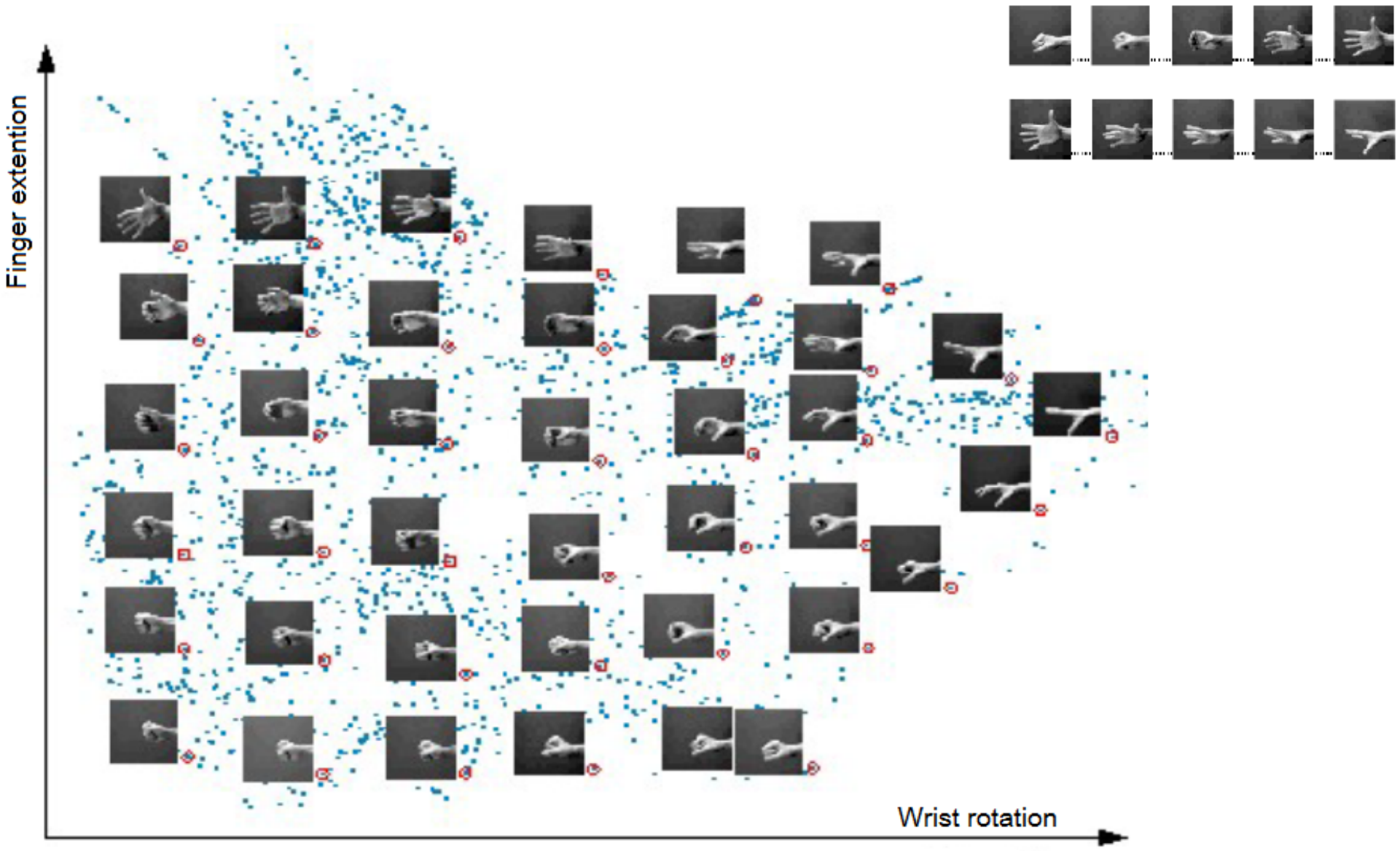
Figure: From D=4096 to d=2

ISOMAP: Hand writing Example



Original images

ISOMAP: Another Image Example



ISOMAP: Conclusion

Advantages of Isomap

- Handles non-linearity
- Is non-iterative
- Preserves the global properties of the data

Limitations of Isomap

- Sensitive to noise (it may create unwanted shortcut in the neighborhood graph)
- The optimal parameters k for KNN or ϵ can be difficult to determine:
  - Small k or ϵ may lead to linear shortcuts and a poor approximation of the geodesic distance, or in worst cases a partitioned graph.
  - Large k or ϵ slow the search for the shortest path, and may at some point loose the low-dimensional manifold.
- Relatively slow with large data sets $\sim O(N^2 log(N))$

Outline

Locally Linear Embedding: Principle

The locally Linear Embedding algorithm (LLE) [Roweis and Saul 2000] addresses the same problem as Isomap in a different way:

- It preserves the local properties of the data by representing each point by a linear combination of its nearest neighbors.
- It builds a projection to a linear low dimensional space preserving the neighborhood.

Locally Linear Embedding: Principle

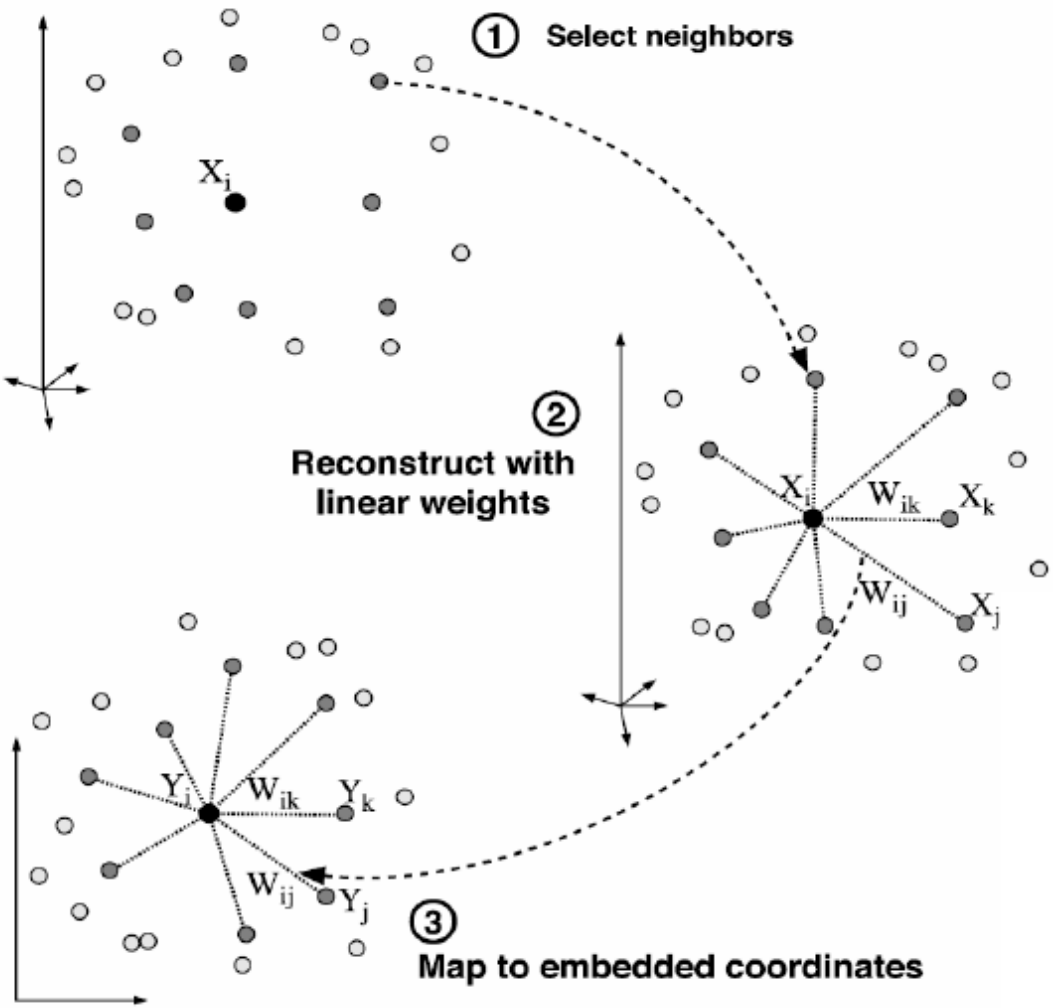
The locally Linear Embedding algorithm (LLE) [Roweis and Saul 2000] addresses the same problem as Isomap in a different way:

- It preserves the local properties of the data by representing each point by a linear combination of its nearest neighbors.
- It builds a projection to a linear low dimensional space preserving the neighborhood.

LLE algorithm

1. Compute the k nearest neighbors
2. Compute the weights needed to reconstruct each point using a linear combination of its neighbors
3. Project the results in a lower dimensional space using the weights

Locally Linear Embedding: Steps



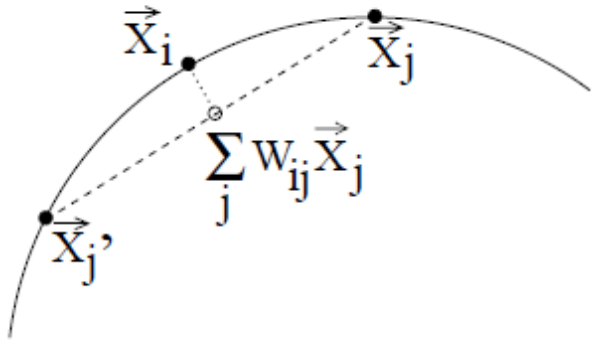


Figure: LLE algorithm [Roweis and Saul 2000]

Locally Linear Embedding: Computing the weights (1/2)

- The local geometry is modeled by linear weights that reconstruct each data point as a linear combination of its nearest neighbors.
- Let us note $W = (w_{ij})_{N \times N}$ the weight matrix, where w_{ij} is the weight given to x_j in the reconstruction of x_i.

Reconstruction error cost function

$$\epsilon(W) = \sum_{i=1}^{N} |x_i - \sum_{j \neq i} w_{ij} x_j|^2$$

The weights w_{ij} are minimized subject to two constraints:

1. Each data point is reconstructed only from its neighbors (i.e. $w_{ij} = 0$ is x_i and x_j are not neighbors)
2. Rows of the weight matrix W sum to 1: $\forall i \quad \sum_j w_{ij} = 1$

Locally Linear Embedding: Computing the weights (2/2)

Let us consider a sample x_i with its k nearest neighbors η_{ij}. With the constraint $\sum_j w_{ij} = 1$, the weights can be found in 3 steps:

Locally Linear Embedding: Computing the weights (2/2)

Let us consider a sample x_i with its k nearest neighbors η_{ij}. With the constraint $\sum_j w_{ij} = 1$, the weights can be found in 3 steps:

1. Compute the neighborhood correlation matrices C_{jk} and their inverse: $C_{jk} = (x_i - \eta_{ij}) \cdot (x_i - \eta_{ik})$

Locally Linear Embedding: Computing the weights (2/2)

Let us consider a sample x_i with its k nearest neighbors η_{ij}. With the constraint $\sum_j w_{ij} = 1$, the weights can be found in 3 steps:

1. Compute the neighborhood correlation matrices C_{jk} and their inverse: $C_{jk} = (x_i - \eta_{ij}) \cdot (x_i - \eta_{ik})$
2. Compute the lagrangian multiplier λ that enforces the constraint $\sum_j w_{ij} = 1$:

$$\lambda = \frac{1 - \sum_{j,k} C_{jk}^{-1}(x_i^T \eta_{ik})}{\sum_{j,k} C_{jk}^{-1}}$$

Locally Linear Embedding: Computing the weights (2/2)

Let us consider a sample x_i with its k nearest neighbors η_{ij}. With the constraint $\sum_j w_{ij} = 1$, the weights can be found in 3 steps:

1. Compute the neighborhood correlation matrices C_{jk} and their inverse: $C_{jk} = (x_i - \eta_{ij}) \cdot (x_i - \eta_{ik})$
2. Compute the lagrangian multiplier λ that enforces the constraint $\sum_j w_{ij} = 1$:

$$\lambda = \frac{1 - \sum_{j,k} C_{jk}^{-1}(x_i^T \eta_{ik})}{\sum_{j,k} C_{jk}^{-1}}$$

3. Compute the reconstruction weights:

$$w_{ij} = \sum_k C_{jk}^{-1}(x_i^T \eta_{ik} + \lambda)$$

Locally Linear Embedding: Mapping the data (1/2)

- Once the weights W have been computed, we seek to find the $Y = \{y_1, \cdots, y_n\}, y_i \in \mathbb{R}^d$ the new points in a low dimensional space.
- This can be done by searching Y that minimizes an embedding cost function similar to the reconstruction error cost function from the input space.

Embedding error cost function

$$\phi(Y) = \sum_{i=1}^{N} |y_i - \sum_{j \neq i} w_{ij} y_j|^2$$

Locally Linear Embedding: Mapping the data (2/2)

The embedding vectors Y_i are found by minimizing the cost function $\phi(Y)$.

- To make the optimization problem well-posed we introduce two constraints: $\sum_j y_j = 0$ and $\frac{1}{N}\sum_i Y_i Y_i^T = I$

Locally Linear Embedding: Mapping the data (2/2)

The embedding vectors Y_i are found by minimizing the cost function $\phi(Y)$.

- To make the optimization problem well-posed we introduce two constraints: $\sum_j y_j = 0$ and $\frac{1}{N}\sum_i Y_i Y_i^T = I$
- Then, the cost function becomes:

$$\phi(Y) = \sum_{i,j} M_{ij}(Y_i^T Y_j)$$

  - Where $M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k w_{ki} w_{kj}$
  - δ_{ij} is equal to 1 if $i = j$ and 0 otherwise

Locally Linear Embedding: Mapping the data (2/2)

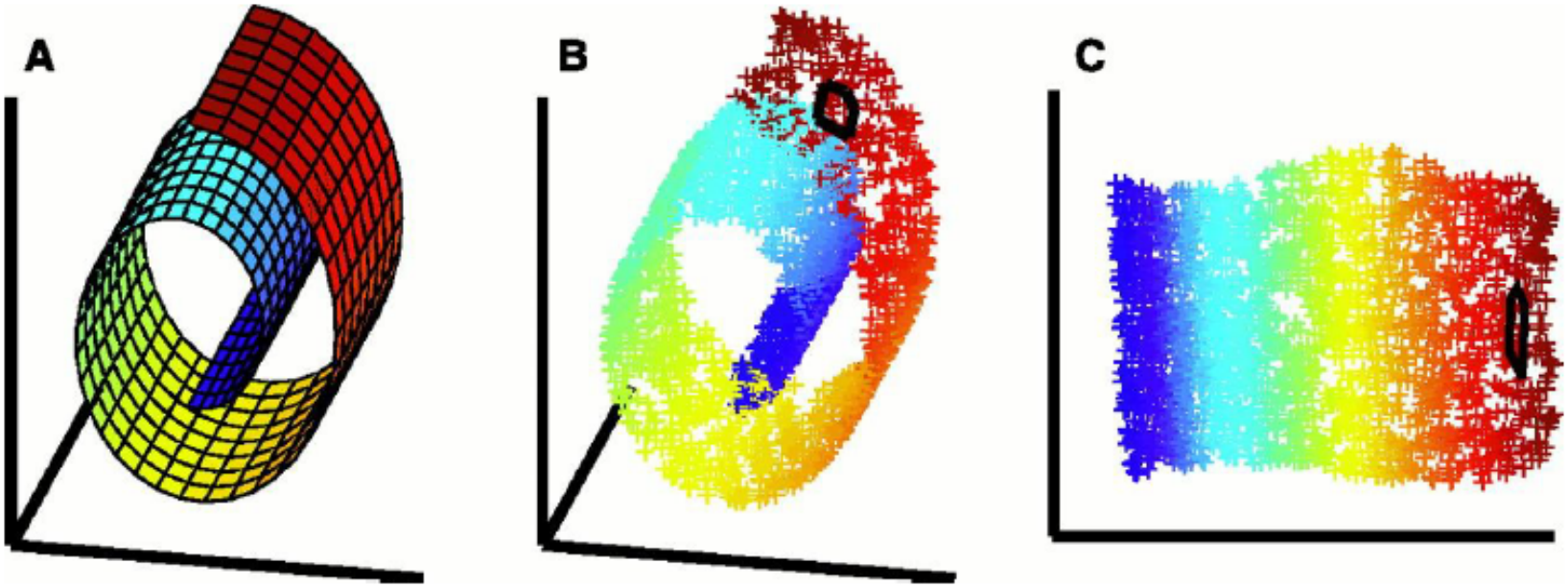
The embedding vectors Y_i are found by minimizing the cost function $\phi(Y)$.

- To make the optimization problem well-posed we introduce two constraints: $\sum_j y_j = 0$ and $\frac{1}{N}\sum_i Y_i Y_i^T = I$
- Then, the cost function becomes:

$$\phi(Y) = \sum_{i,j} M_{ij}(Y_i^T Y_j)$$

  - Where $M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k w_{ki} w_{kj}$
  - δ_{ij} is equal to 1 if $i = j$ and 0 otherwise
- The optimal embedding for a d-dimensional space is found by computing the bottom $d + 1$ eigenvectors of the matrix M.

Locally Linear Embedding: The "Swiss roll" Example



[Roweis and Saul, 2000]

Locally Linear Embedding: Face data set Example

- The initial data represent images of faces.
- In the 2D space, these images are grouped according to the position, lighting and expression.

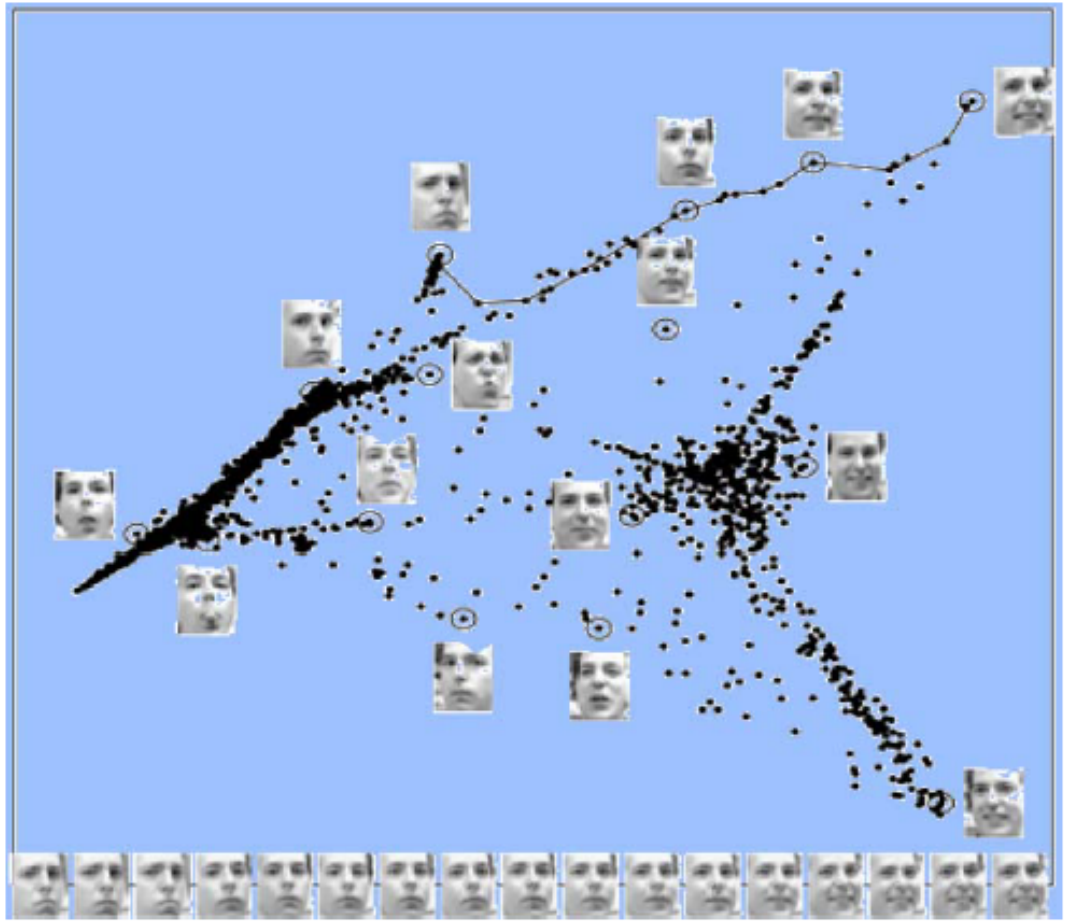
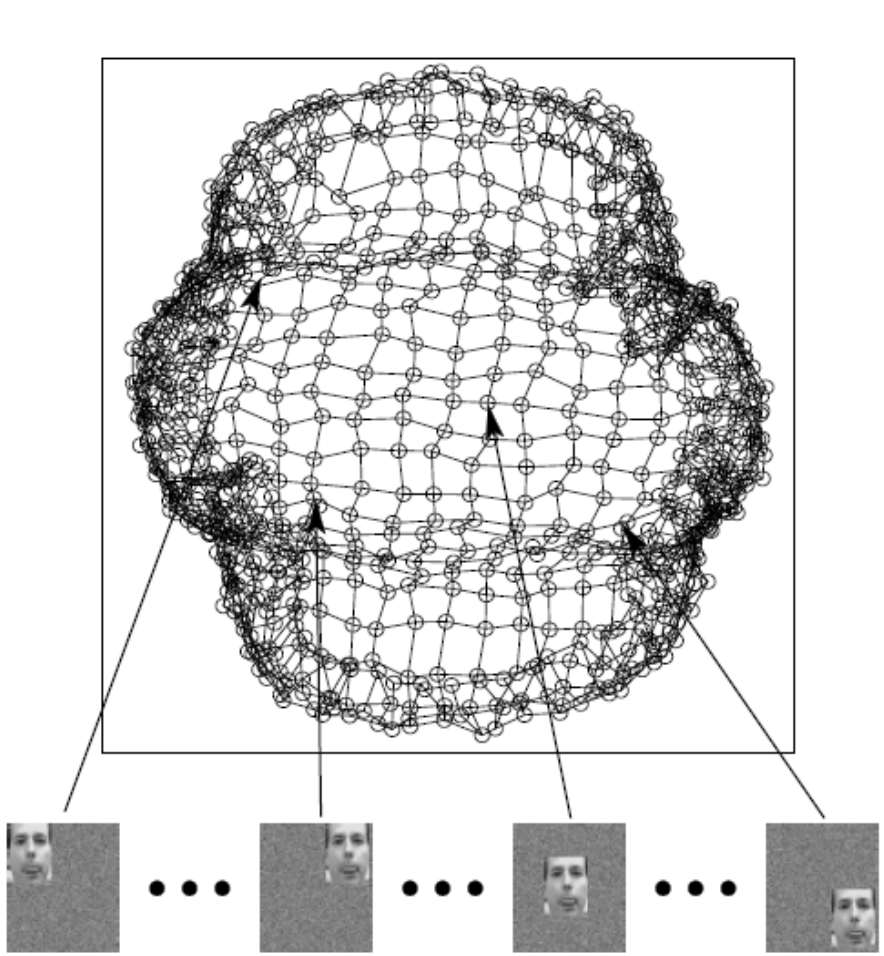


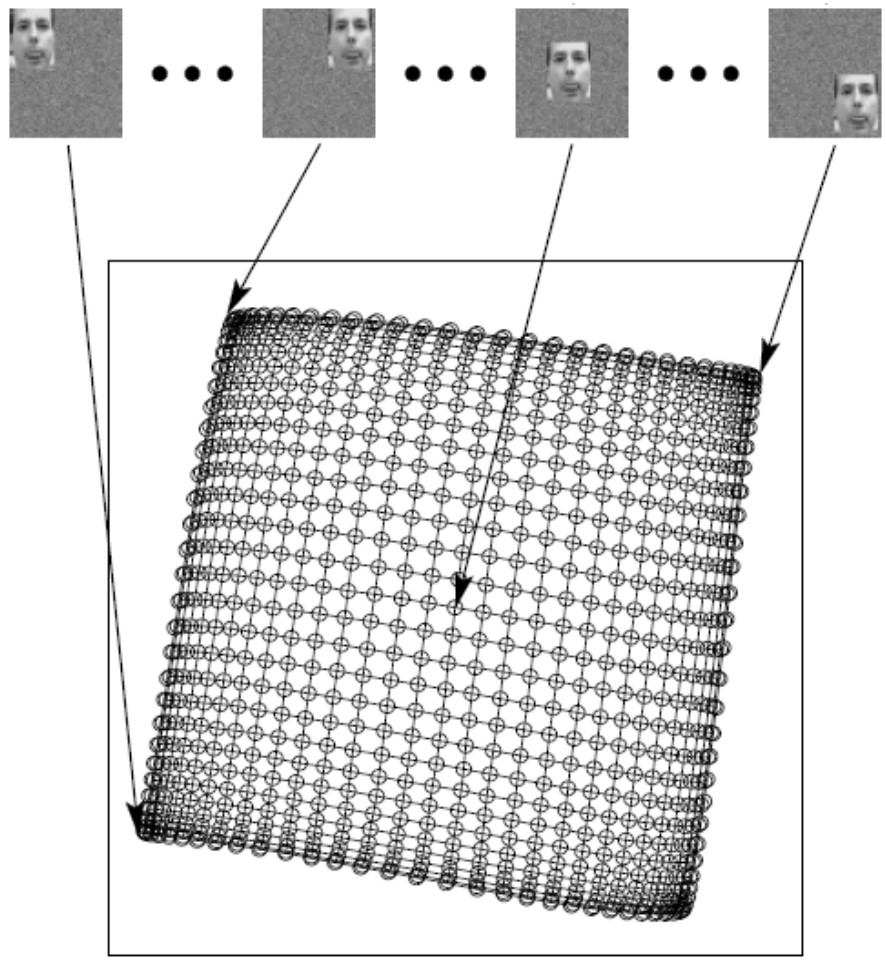
Figure: Images placed at the bottom of the figure correspond to successive points encountered on the line at the top right, sweeping a continuum of facial expression. [Roweis and Saul 2000]

Locally Linear Embedding: PCA VS LLE (1/2)

Unlike PCA, LLE preserves the local topology

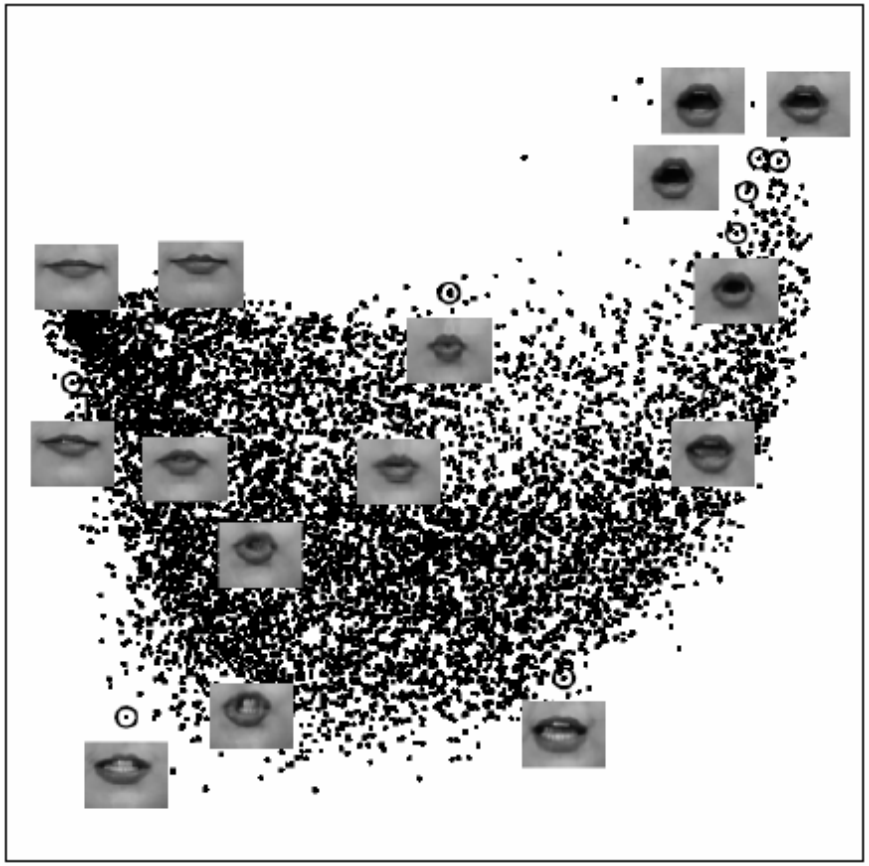


(a) PCA

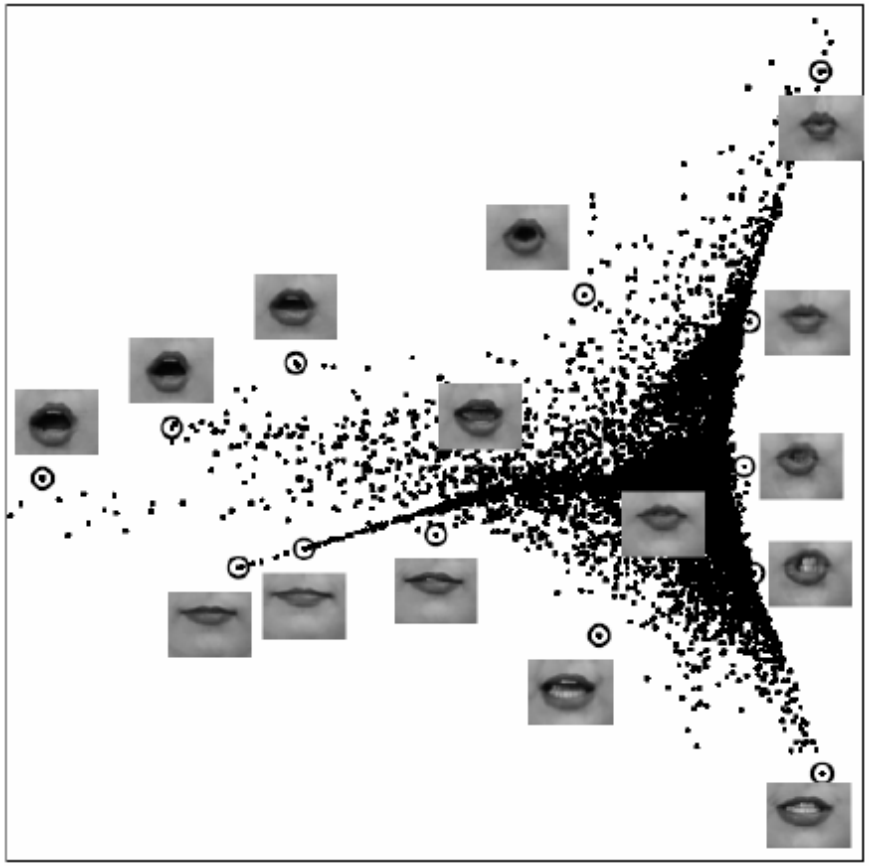


(b) LLE

Locally Linear Embedding: PCA VS LLE (2/2)



(c) PCA



(d) LLE

Locally Linear Embedding: Conclusions

Advantages of LLE

- Handles non-linearity
- Is non-iterative
- Preserves local topologies

Limitations of LLE

- Sensitive to noise
- The optimal parameters k for KNN can be difficult to determine:
- Relatively slow with large data sets due to the large number of covariance matrices inversions.

Outline

What is t-SNE ?

- t-distributed stochastic neighbor embedding (t-SNE) is a machine learning algorithm for data visualization, proposed in 2008 by van der Maaten and Hinton.
- It is based on the similar idea of neighborhood embedding proposed for LLE by Roweis and Saul.
- It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions.

Stochastic Neighbor Embedding (SNE)

SNE: Basic idea

- Encoding high dimensional neighborhood information as a distribution
- Intuition: Random walk between data points with a high probability to jump to a close point.
- Finding low dimensional points such that their neighborhood distribution is similar.

How to measure the distance between distribution ?

- The most common measure to do so is the KL divergence

SNE: local embedding algorithm (1/4)

- Consider the neighborhood around any input data point $x_i \in \mathbb{R}^D$
- Imagine that we have a Gaussian distribution centered around x_i
- Then the probability that x_i chooses some other datapoint x_j as its neighbor is in proportion with the density under this Gaussian.
- A point closer to x_i will be more likely than one further away

SNE: local embedding algorithm (2/4)

$P_{j|i}$ the probability that x_i chooses x_j as neighbor is computed as follows:

$$P_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

- $P_{i|i} = 0$
- σ_i the chosen size of the neighborhood:
  - a low σ_i will lead to a smaller local neighborhood with stronger probabilities
  - a high σ_i leads to uniform weights
  - σ_i can be set up differently for each point.
- The final distribution over pairs is symmetrized: $P_{ij} = \frac{1}{2N}(P_{j|i} + P_{i|j})$

SNE: local embedding algorithm (3/4)

- Given a dataset $X = \{x_1, \cdots, x_N\} \in \mathbb{R}^D$, we define the distribution P_{ij}
- Goal: Finding a good embedding $Y = \{y_1, \cdots, y_N\} \in \mathbb{R}^d, \quad d << D$, and ideally d should be 2 or 3.

Measuring and minimizing the embedding quality

- For all point in Y we can define Q in a similar way than P (notice the absence of σ_i and the assymetry):

$$Q_{ij} = \frac{\exp(-||y_i - y_j||^2)}{\sum_{k \neq i} \exp(-||y_i - y_k||^2)}$$

- We optimize Q to be close to P by minimizing the KL-divergence on the parameters $Y = \{y_1, \cdots, y_N\}$!

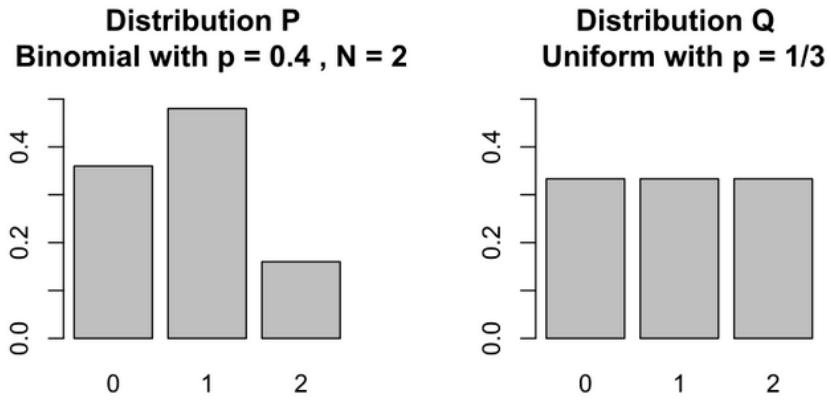
The KL divergence (1/2)

Measuring the distance between two discrete distributions P and Q:

$$KL(Q||P) = \sum_i Q(i) \log\left(\frac{Q(i)}{P(i)}\right)$$

- It is a metric function as it is not symmetric.

Example:



	0	1	2
Distribution P	0.36	0.48	0.16
Distribution Q	0.333	0.333	0.333

$$DL(Q||P) = 0.333 \ln(\frac{0.333}{0.36}) + 0.333 \ln(\frac{0.333}{0.48}) + 0.333 \ln(\frac{0.333}{0.16})$$
$$= -0.02596 - 0.12176 + 0.24408 = 0.09637$$

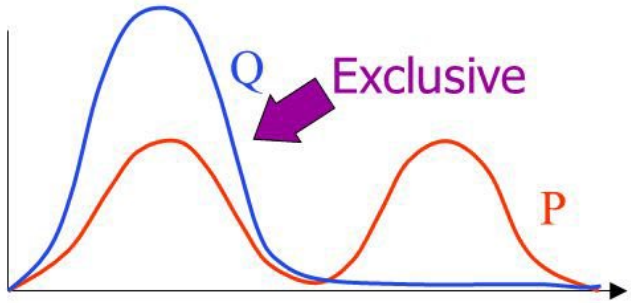
The KL divergence (2/2)

KL properties

- $KL(Q||P) \geq 0$ and is a convex function
- $KL(Q||P) = 0$ if and only if $Q = P$
- if $P(i) = 0$ but $Q(i) > 0$, then $KL(Q||P) = \infty$

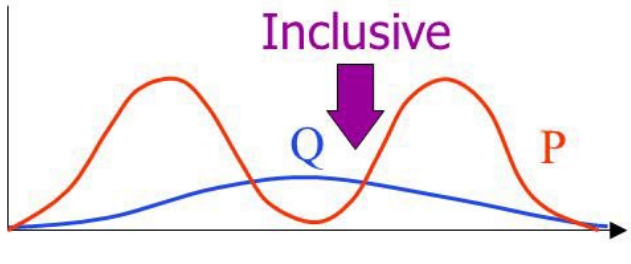
Minimising $KL(Q||P)$

$$= \sum_H Q(H) \ln \frac{Q(H)}{P(H|V)}$$



Minimising $KL(P||Q)$

$$= \sum_H P(H|V) \ln \frac{P(H|V)}{Q(H)}$$



credit: https://timvieira.github.io/blog/post/2014/10/06/kl-divergence-as-an-objective-function/

SNE: local embedding algorithm (4/4)

- We have P and we are looking for $Y = \{y_1, \cdots, y_N\} \in \mathbb{R}^d$ such that the distribution Q we infer will minimize $L(Q) = KL(P||Q)$ (notice the Q on the right, which is uncommon).
- Note that $KL(P||Q) = \sum_{ij} P_{ij} \log\left(\frac{P_{ij}}{Q_{ij}}\right) = -\sum_{ij} P_{ij}(Q_{ij}) + const$
- We can show that $\frac{\partial L}{\partial y_i} = \sum_j (P_{ij} - Q_{ij})(y_i - y_j)$

Remarks

- This is not a convex problem: No convergence guarantees ! Multiple restarts may be needed.
- Another issue may arise: crowding problems.

Crowding problems

- We have seen at the beginning of this class that in high dimension we have more room: it means that points can have a lot of different neighbors in a given radius.
- However, in lower dimension (1D, 2D or 3D), there is a lot less room in the same radius.
- This is the "crowding problem": we might not have enough room to accommodate all neighbors.
- This is one of the biggest weakness for LLE and SNE.

t-SNE solution to crowding problems

- Change the Gaussian in Q by a heavy tailed distribution : if Q changes slower, we have more "wiggle room" to place points.

t-SNE: what is "t" by the way ?

- Student-t Probability density: $p(x) \approx \left(1 + \frac{x^2}{v}\right)^{-(v+1)/2}$
  - For $v = 1$, we get: $p(x) \approx \frac{1}{1+x^2}$
- The probability of this distribution goes much more slower to zero than a Gaussian.
  - This is convenient for our crowding problem.

We can show that this is equivalent to averaging a Gaussian with some prior over σ^2. Then, we can redefine Q_{ij} so that:

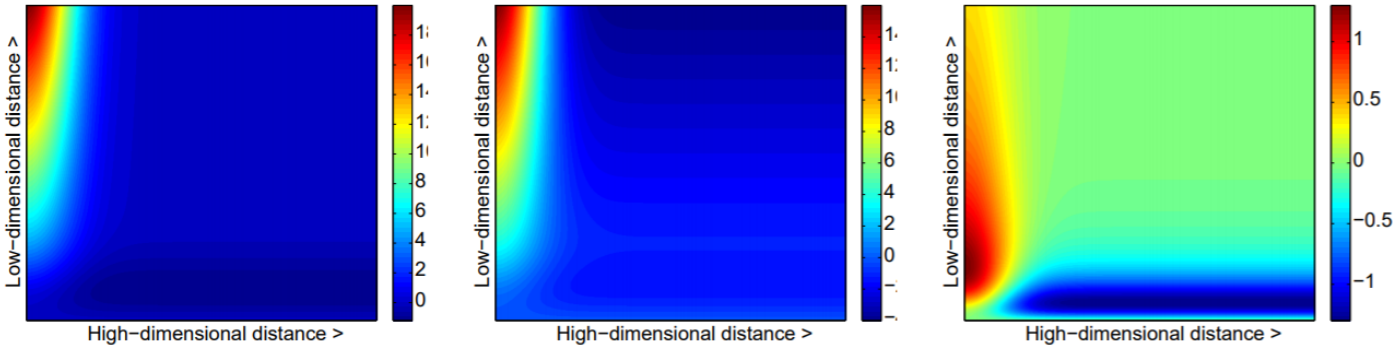
$$Q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + ||y_i - y_k||^2)^{-1}}$$

t-SNE gradients

- We can show that the gradients of t-SNE objective function are:

$$\frac{\partial L}{\partial y_i} = \sum_j \frac{(P_{ij} - Q_{ij})(y_i - y_j)}{1 + ||y_i - y_j||^2}$$

- Compared with the SNE gradients: $\frac{\partial L}{\partial y_i} = \sum_j (P_{ij} - Q_{ij})(y_i - y_j)$



(a) Gradient of SNE. (b) Gradient of UNI-SNE. (c) Gradient of t-SNE.

credit: "Visualizing Data using t-SNE"

- Both repulse close dissimilar points and attract far similar points, but the t-SNE has a smaller attraction term that reduces crowding issues.

t-SNE algorithm

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for t=1 **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
  end
end

source : "Visualizing Data using t-SNE"

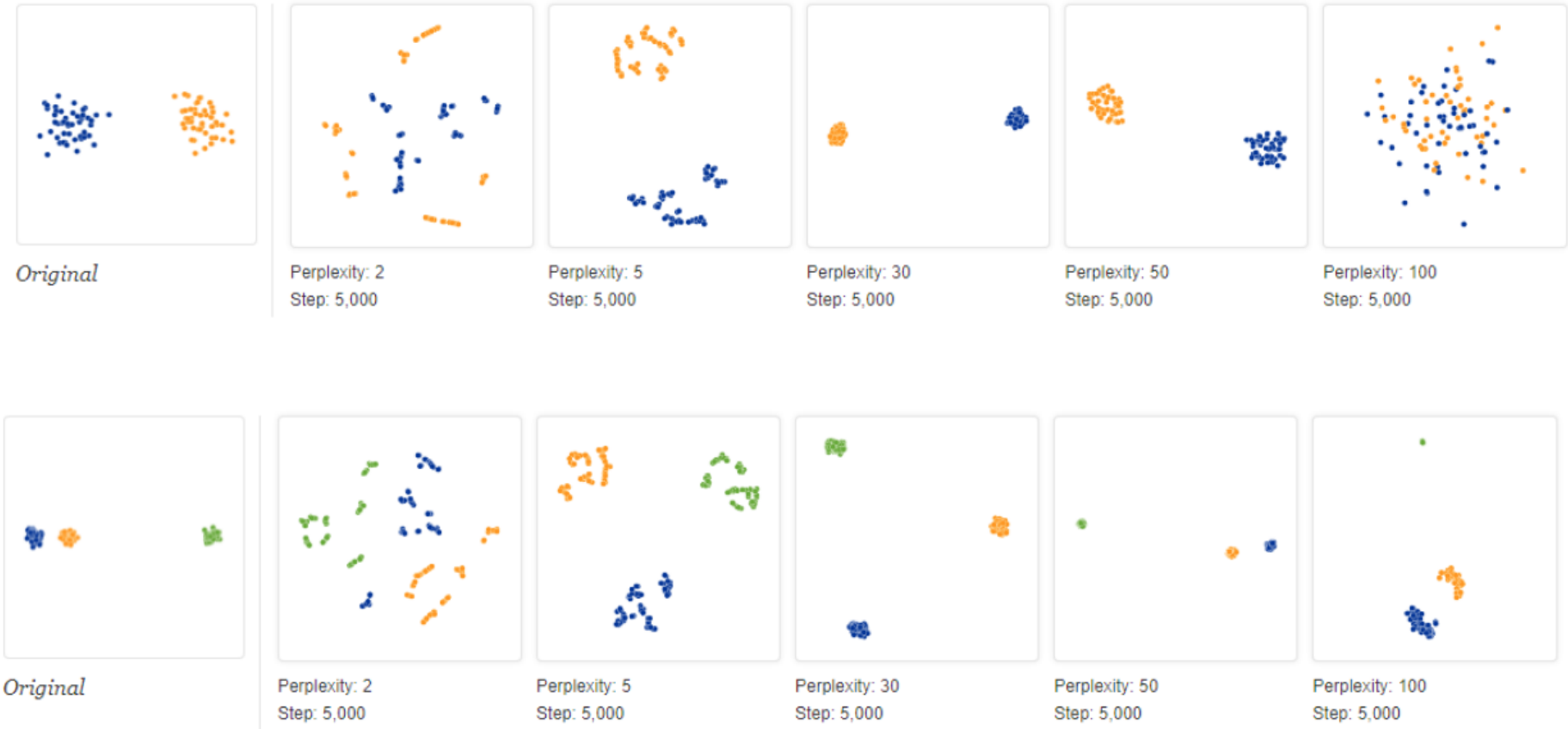
t-SNE: Perplexity (1/2)

We have seen that σ_i is a key neighborhood parameter. For each distribution $P_{j|i}$ (depending on σ_i), we define the perplexity:

$$Perp(P_{j|i}) = 2^{H(P_{j|i})} \quad \text{where} \quad H(P) = -\sum_i P_i \log(P_i)$$

- If P is uniform over k elements - perplexity is k.
  - Smooth version of k in kNN
  - Low perplexity means small σ^2
  - High perplexity means large σ^2
- We can define the desired perplexity and set σ_i to get that (bisection method): values between 5 and 50 usually work well.
- It is an important parameter as it allows to capture different scales in the data.

t-SNE: Perplexity (2/2)



source : https://distill.pub/2016/misread-tsne/

Local embedding: PCA vs t-SNE

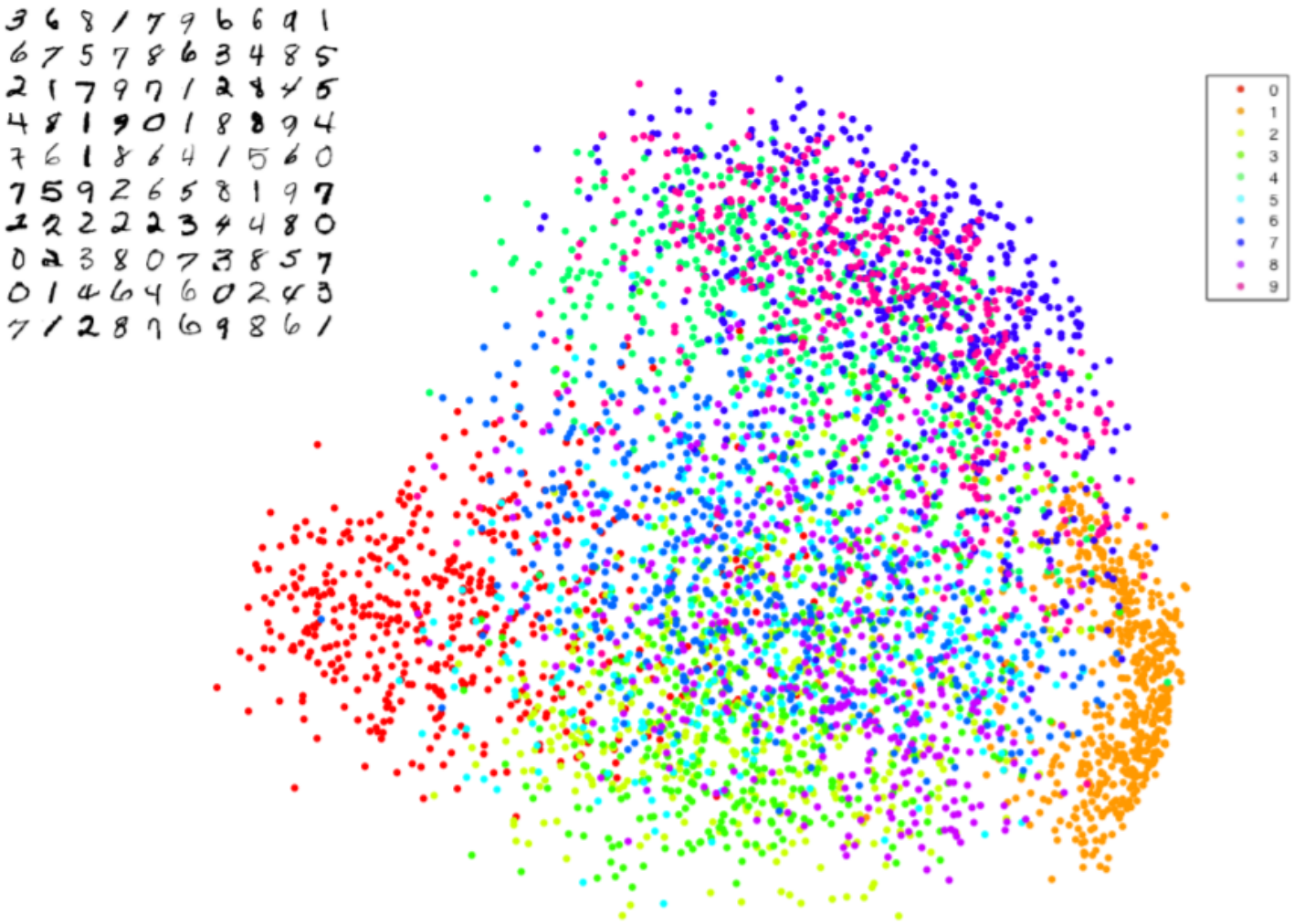
PCA

- PCA tries to find a global structure in a low dimensional subspace
- PCA can suffer from local inconsistencies: far away points become nearest neighbors and vice-versa.
- PCA is a projection method: new points can be projected

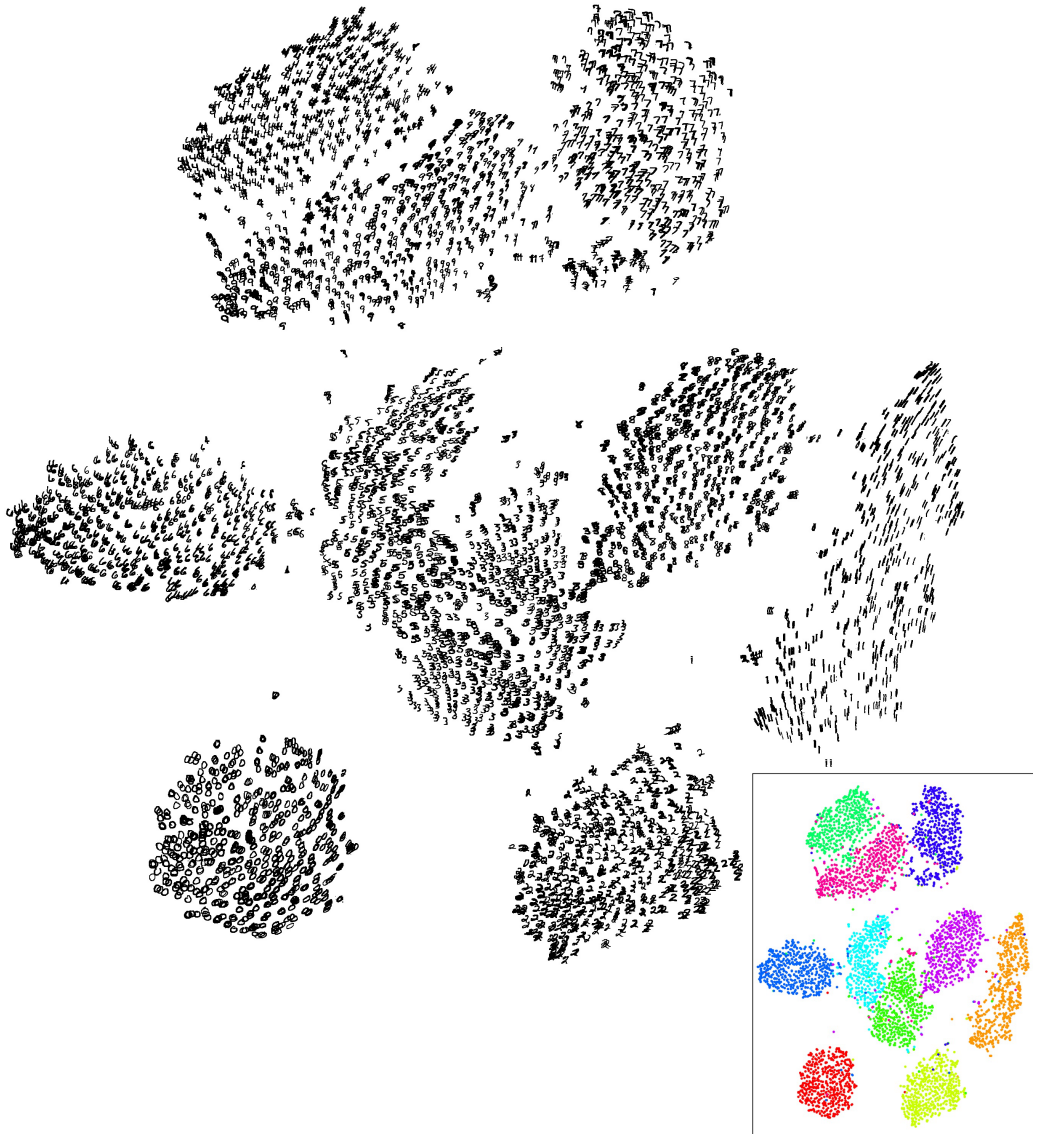
t-SNE (and LLE)

- t-SNE -like LLE- tries to preserve local structures: Low dimensional neighborhood should be the same as original neighborhood.
- The local embedding is approximated and is never perfect.
- t-SNE is not a projection method: There is no easy way to embed new points.

t-SNE vs PCA: PCA applied to MNIST



t-SNE vs PCA: t-SNE applied to MNIST

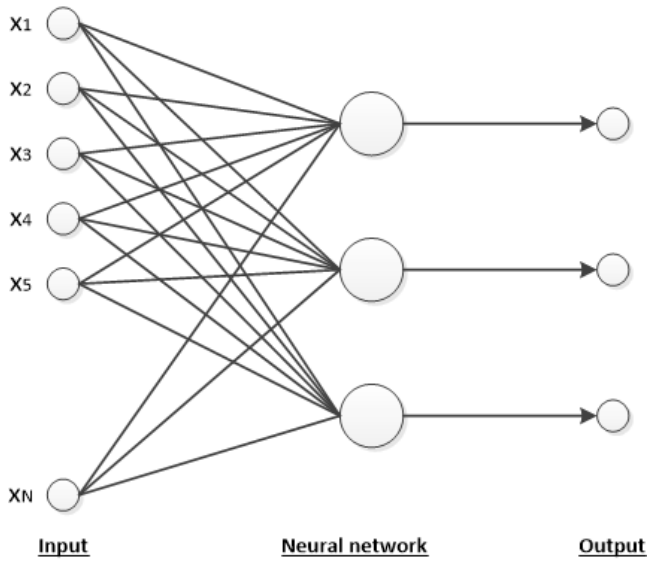


Outline

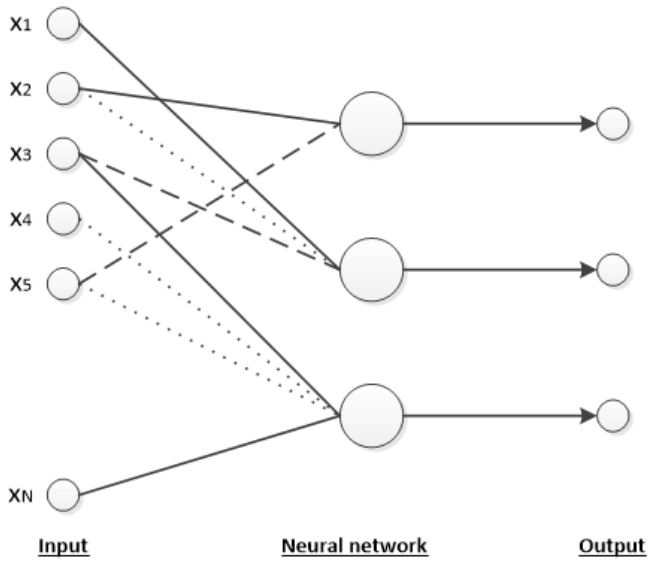
Neural networks

Neural networks are a variety of algorithms that mimic brain cells:

- Each neuron, or each group of neuron specializes in recognizing certain patterns.



(e) Untrained neural network



(f) Trained neural network

Figure: Example of a single layer neural network

Self-organizing maps: Introduction

Self-organization in brain cells

- Brain cells are self organizing themselves in groups according to incoming information.
- This incoming information is not only received by a single neural cell, but also influences other cells in its neighborhood.

Self-organizing maps (SOM) [T. Kohonen 1984] are a family of unsupervised neural networks based on this principle:

- It learns in an unsupervised way using prototypes that play the role of neurons.
- The prototypes follow a given topology so that neighborhood dependencies exist between them.
- Groups of neighbor prototypes specialize in recognizing similar groups of data.

Self-organizing maps: Architecture (1/2)

Self-organizing maps can have all sorts of 1D, 2D or 3D architecture that is convenient for visualization:

- 1D architecture: line of prototypes
- 2-Dimensional architectures (the most common): square or rectangular maps with any type of neighborhood (triangular neighborhood, square neighborhood, hexagonal neighborhood, or octagonal neighborhood)
- 3-Dimensional architectures (uncommon): Cubes, spheres.

Self-organizing maps: Architecture (2/2)

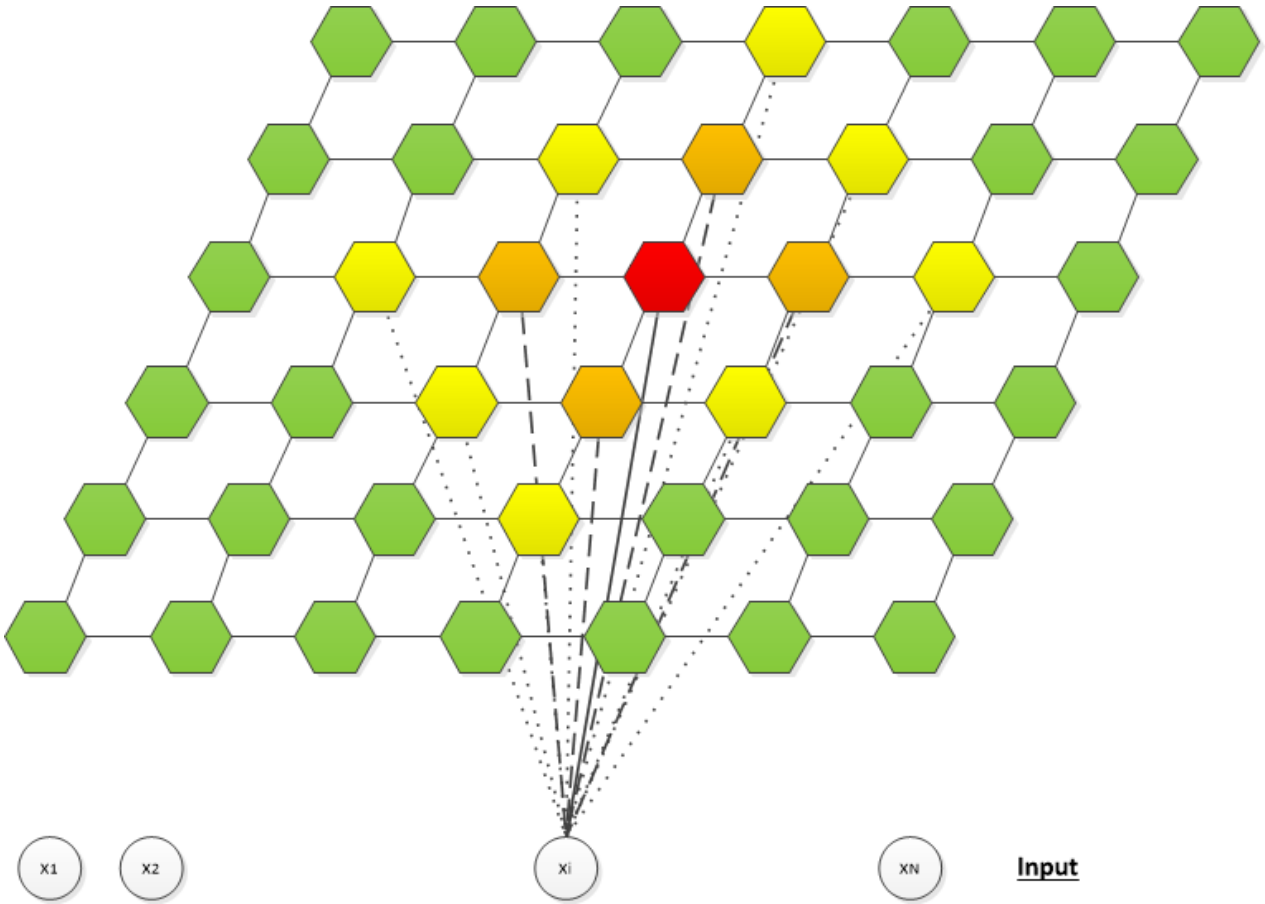


Figure: Example of a rectangle grid SOM architecture with 4 neighbors per neuron: Each example activates a winning neuron and also affects its neighbors.

Self-organizing maps: Cost function

For a data set $X = \{x_1, \cdots, x_N\}$ and a set of prototypes $W = \{w_1, \cdots, w_K\}$, the original SOM algorithm tries to minimize the following cost function:

$$C(W) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{K}_{k,\chi(x_i)} ||x_i - w_k||^2$$

- $\chi(x_i)$ is the id of the best matching prototype w_j so that: $\chi(x_i) = \text{argmin}_j ||x_i - w_j||^2$.
- $0 \leq \mathcal{K}_{k,j} \leq 1$ is the neighborhood topological function (e.g. $\mathcal{K}_{k,j} = 0$ if w_j and w_k are far on the topological grid).

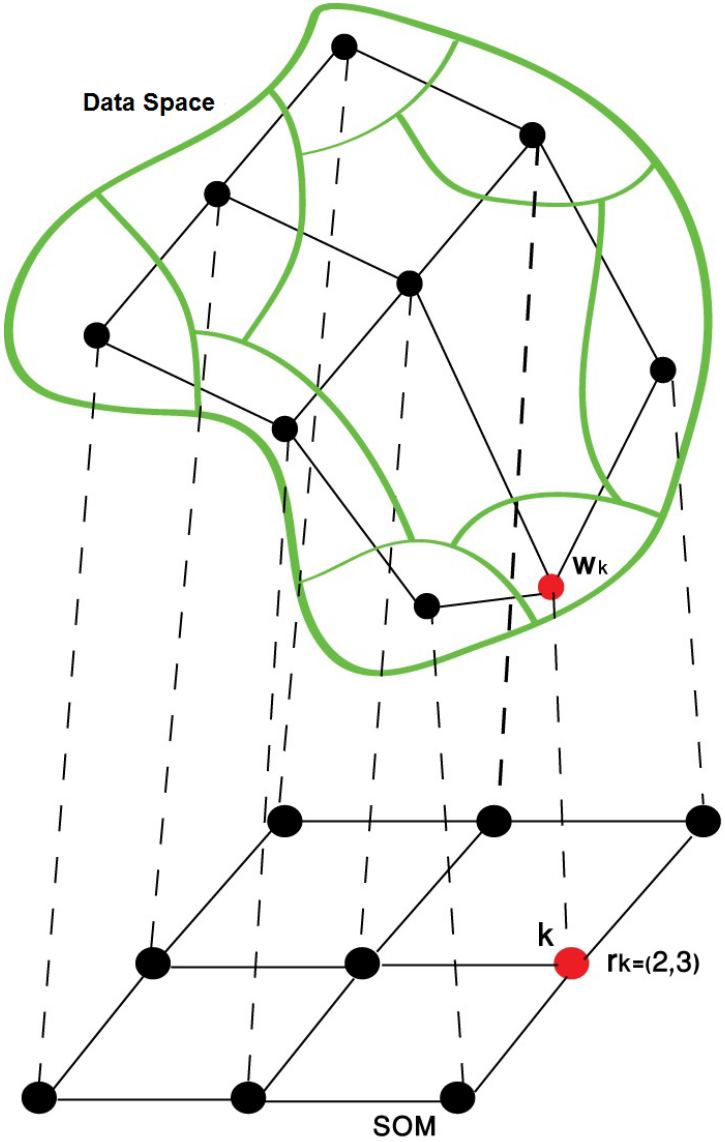
Self-organizing maps: Algorithm (1/2)

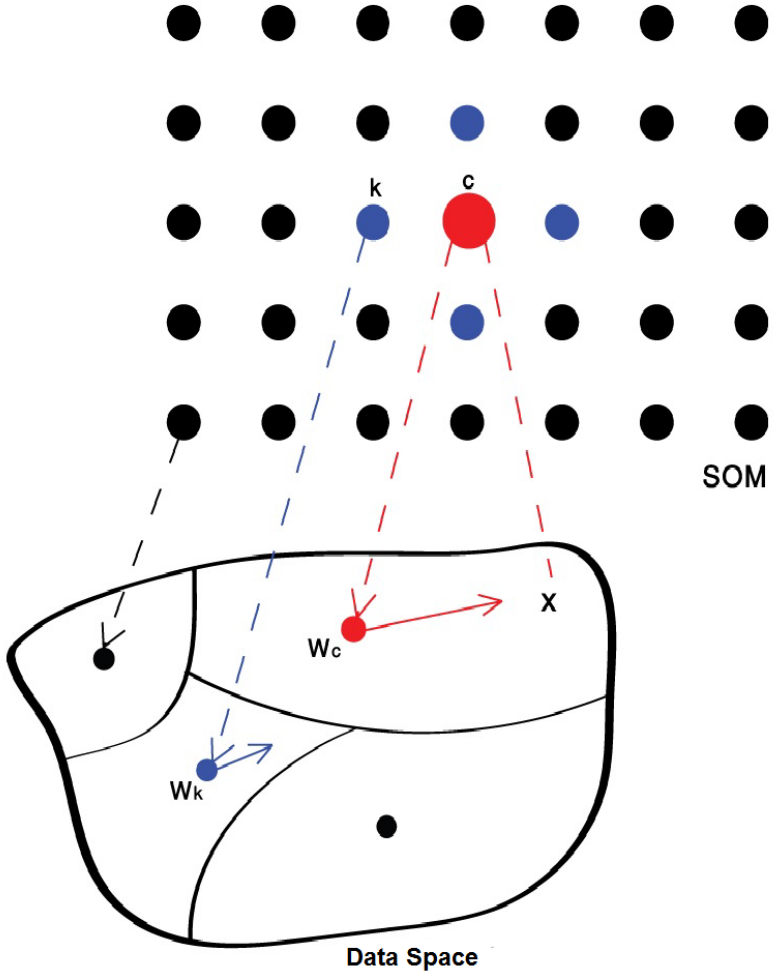
1. Initialization step
   - Define the topology of the map
   - Randomly initialize the prototypes for each neuron
   - Define a learning rate function $\epsilon(t)$
2. Competition step
   - Choose a data x_i randomly
   - Determine the winning neuron $\chi(x_i) = \text{argmin}_{1 \leq j \leq K} ||x_i - w_j||^2$
3. Adaptation step: Update the prototypes

$$w_k(t+1) = w_k(t) + \epsilon(t)\mathcal{K}_{k,\chi(x_i)}\left(x_i - w_k(t)\right)$$

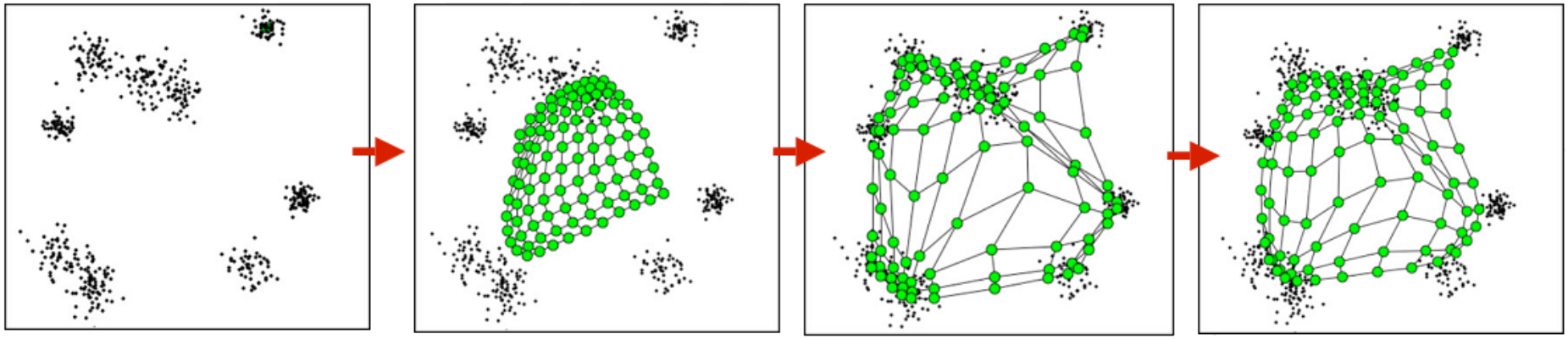
4. Repeat 2 and 3 until the prototypes updates are insignificant.

Self-organizing maps: Algorithm (2/2)





Self-organizing maps: Topological learning



Remark

- Isomap and LLE try to find a low dimensional manifolds in the data using neighborhood dependencies of the data.
- SOM has the opposite approach by trying to wrap a low-dimensional manifold around the data using prototypes neighborhood dependencies.

Self-organizing maps for visualization purposes

Once they are built, SOM can be used in several ways for visualization purposes:

- Cluster visualization on the 2D map using the best matching neuron
- Topological visualization of the data set using colors to project distance matrices between the prototypes on the map.
- Component plane analysis by projecting the components on the map.
- Data projection and transformation

Self-organizing maps: Visualization Example 1

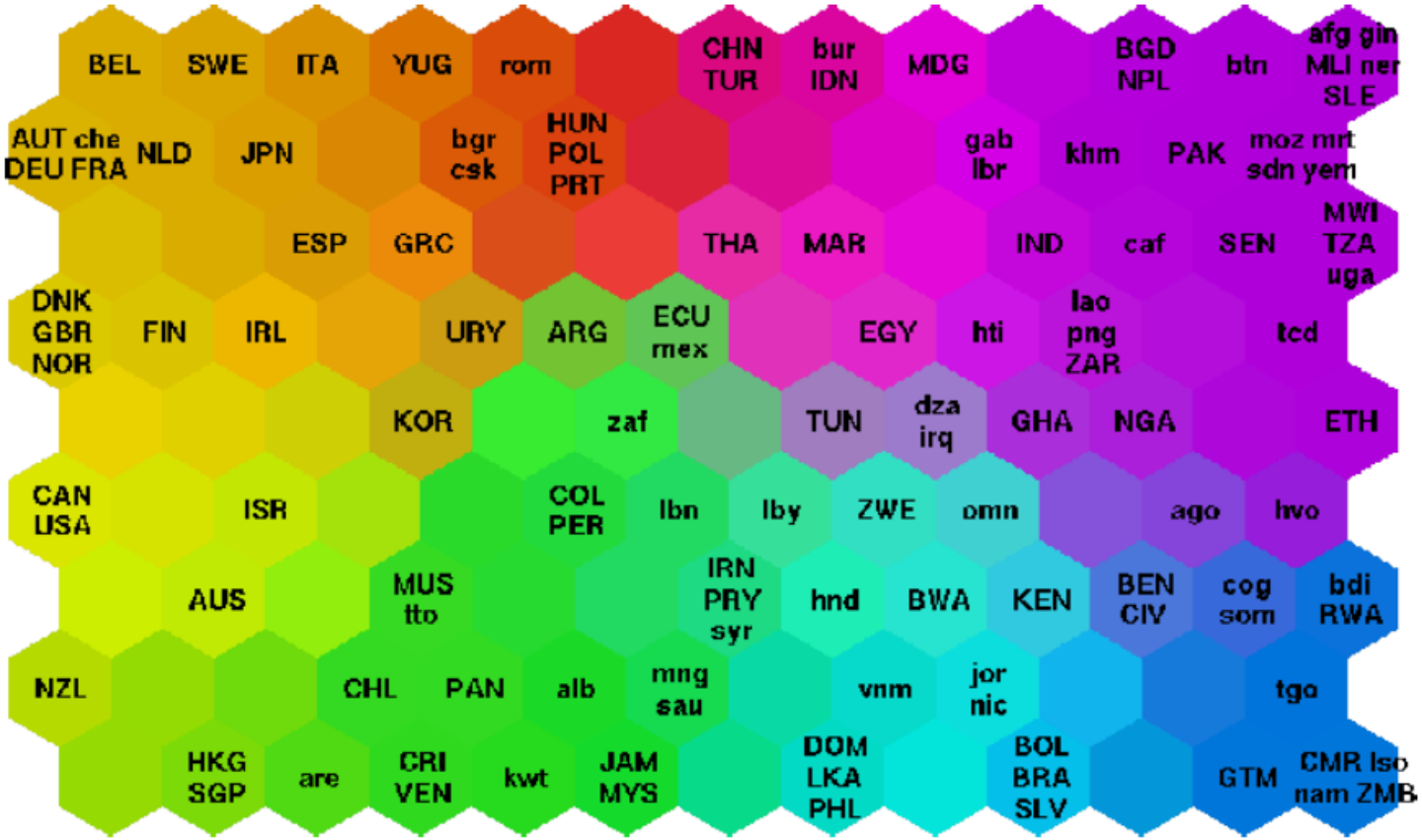


Figure: Visualizing world countries clusters projected on a topological mal based on their characteristics from the Worldbank data.

Self-organizing maps: Visualization Example 2 (1/2)

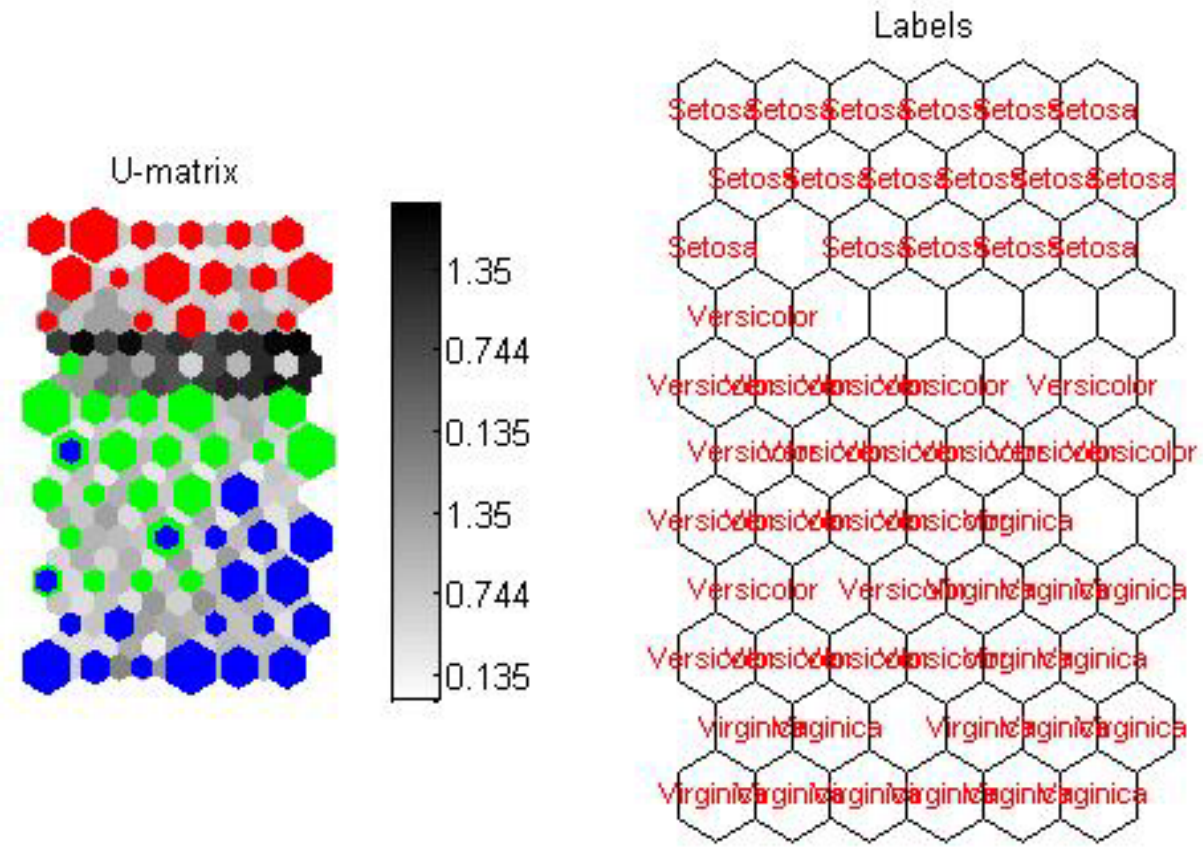


Figure: U-Matrix (distance) and class repartition for the Iris data set using SOM

Self-organizing maps: Visualization Example 2 (2/2)

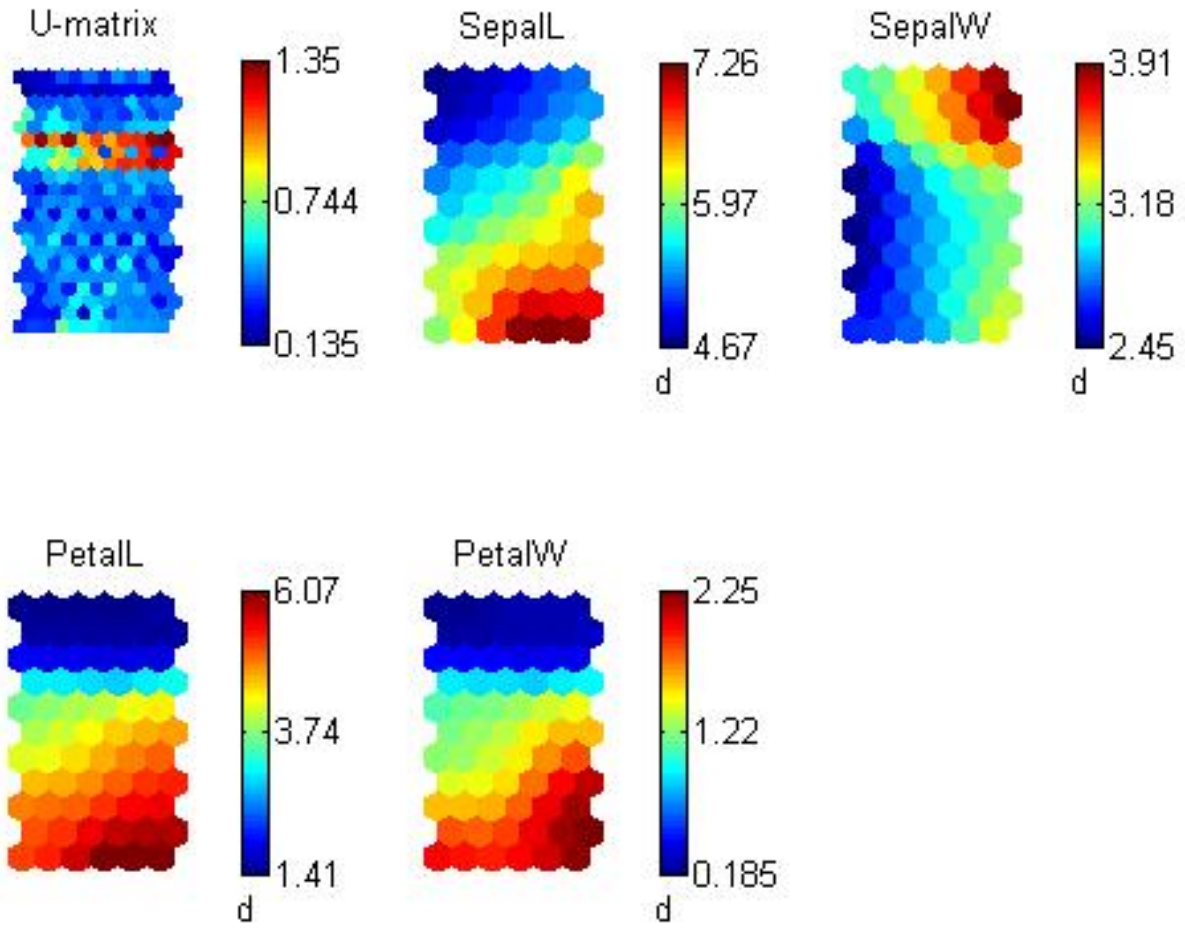
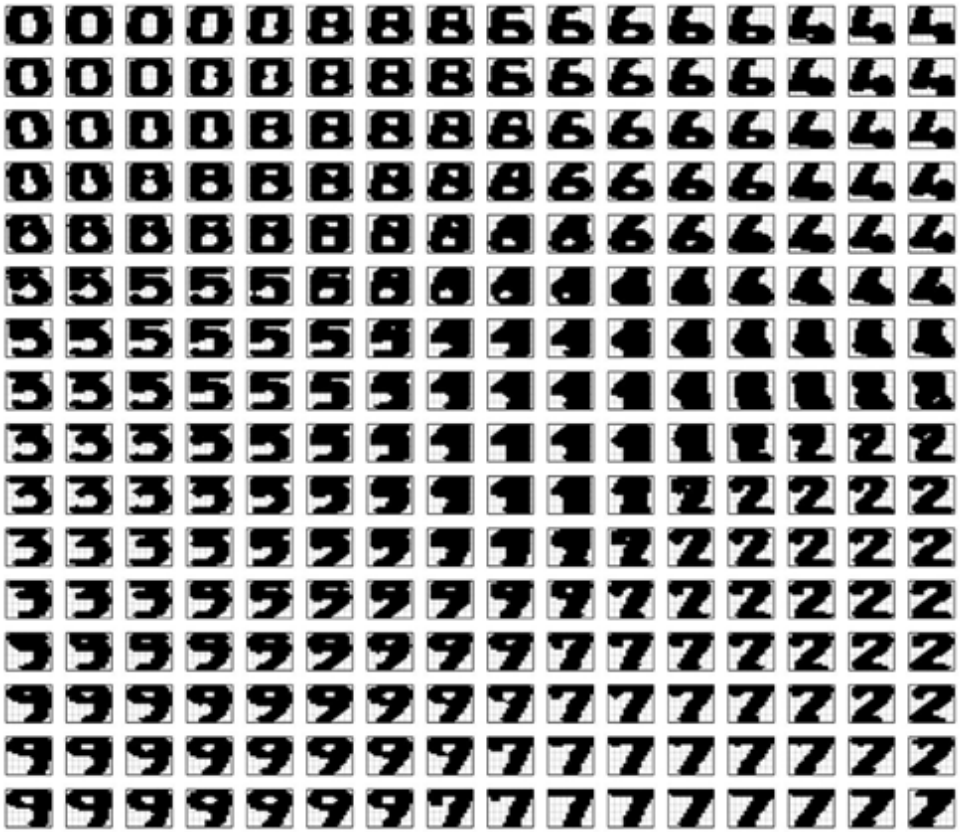
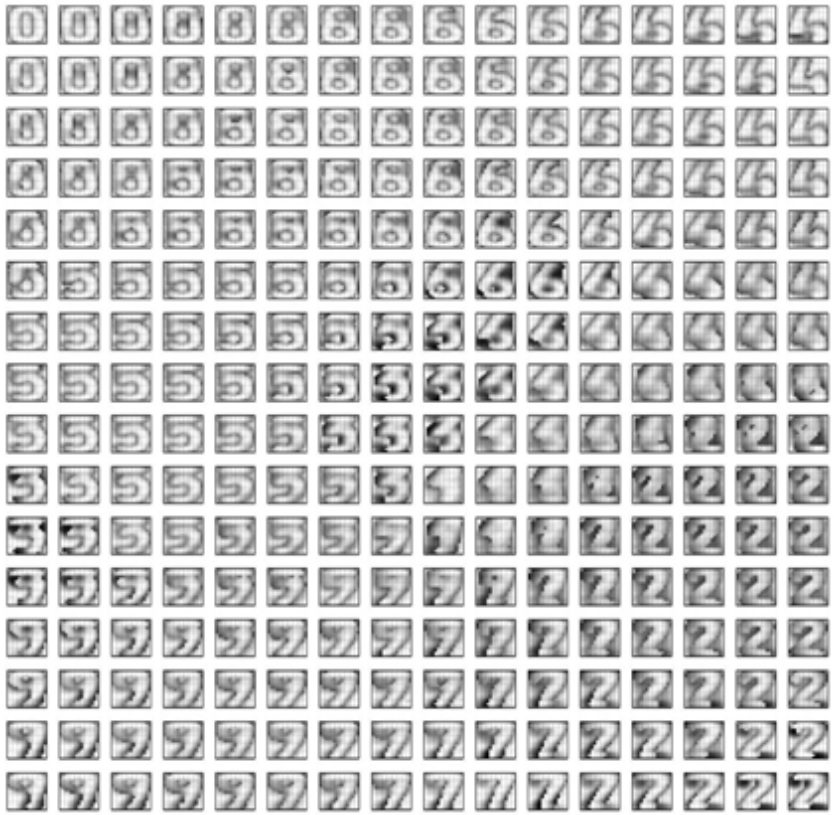


Figure: Feature projection for the Iris data set using SOM

Self-organizing maps: Visualization Example 3



(a) Original SOM



(b) Contrast only

Figure: Number recognition using SOM [Rogovschi et al. 2008]

Self-organizing maps: SOM vs PCA

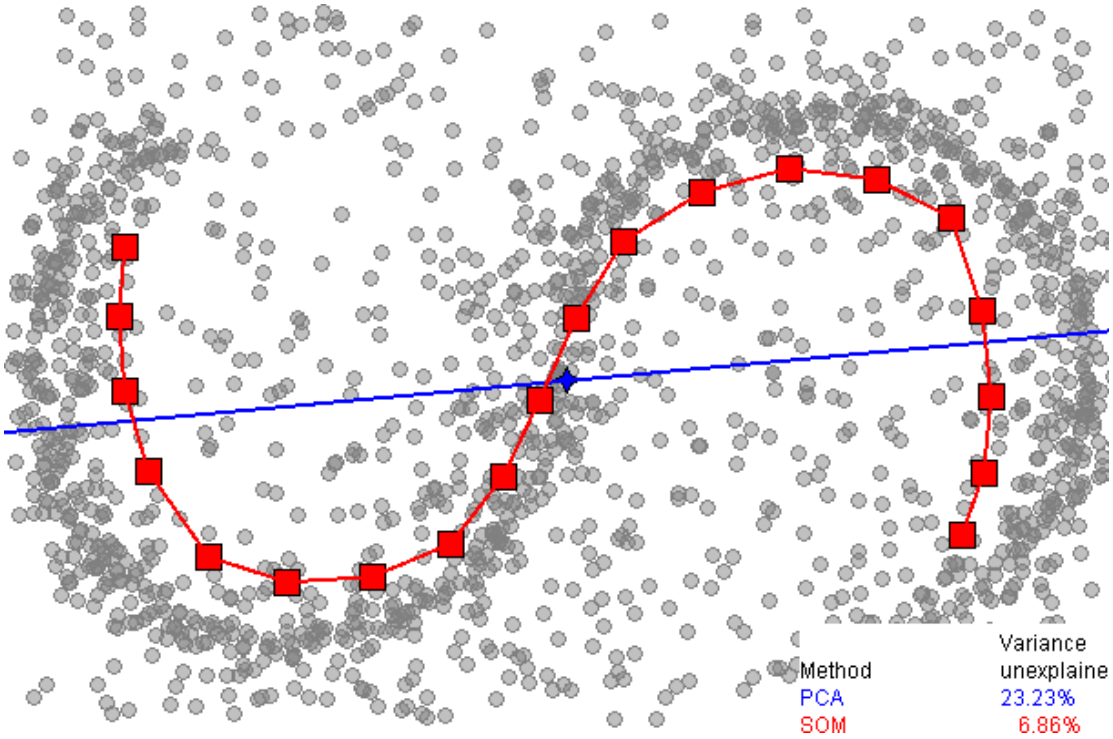


Figure: Projecting the data along a linear SOM network can work better than a PCA on non-linear data

Self-organizing maps: Conclusions

Advantages of SOM

- Is useful for all kinds of visualizations and does a pre-clustering
- Can handle non-linear data
- Is fast $O(KN)$
- The initial topology of the map does not matter much: If the map is big enough, it will work.

Limitations of SOM

- The learning rate is a critical parameter.
- There are plenty of variations of the algorithm each of them with different strengths and weaknesses.

Outline

Bibliography

Books:

- Christopher M. Bishop, Pattern Recognition and Machine Learning (2006)
- Tom M. Mitchell, Machine Learning (1997)

Articles:

- J. B. Tenenbaum, V. De Silva, and J. C. Langford: *A global geometric framework for nonlinear dimensionality reduction*, Science, 290:2319-2323, 2000.
- Sam Roweis and Lawrence Saul: *Nonlinear dimensionality reduction by locally linear embedding*, Science, 290:2323-2326, 2000
- van der Maaten, L.J.P.; Hinton, G.E.: *Visualizing Data Using t-SNE*. Journal of Machine Learning Research (9) 2579-2605, 2008.