Non-linear dimensionality reduction and kernels: eigenmaps, isomaps, locally linear embeddings

Review of Dimensionality Reduction

- Dimensionality reduction can be done through

  1. feature selection: only keeps the most relevant variables from the original dataset.

  2. dimensionality reduction: finds the smaller set of new variables, containing basically the same information as the original variables.

- Dimensionality reduction can also be categorized into:
  - linear dimensionality reduction (e.g. PCA, SVD)
  - non-linear dimensionality reduction (e.g. autoencoders, kernel PCA and others).

Non-linear dimensionality reduction

- **This lecture:** find a nonlinear low dimensional representation of data that reflects the data topology
- **Methods covered:**
  - Isomaps
  - Locally Embedding Space (LLE)
  - Eigenmaps

Motivation

While the data is represented in high dimensions, it usually only has a few degrees freedom.

Examples:

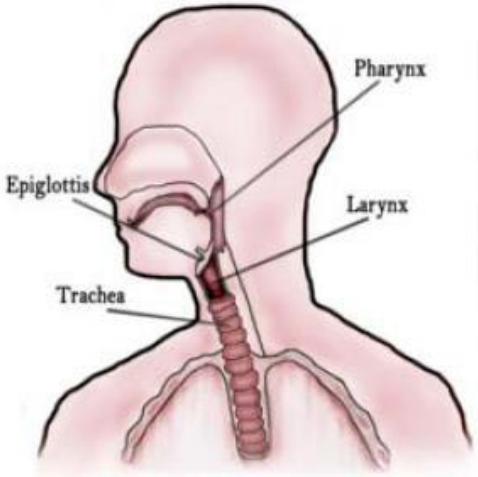
- Scanned images of handwritten characters

  Representation: 28x28 pixels (= 784 dim)

  Few attributes such as: shape, tilt and cursiveness govern the actual written character.

- Natural processes with physical constraints - speech

  Few anatomical characteristics, such as size of the vocal chords, pressure applied, etc. govern the speech signal.



Topology

- Topology: "*the study of qualitative properties of certain objects that are invariant under a certain kind of transformation, especially those properties that are invariant under a certain kind of invertible transformation*"
- "A topologist is one who doesn't know the difference between a doughnut and a coffee cup" –John L. Kelley (In General Topology 1995, 88 footnote)

Manifolds

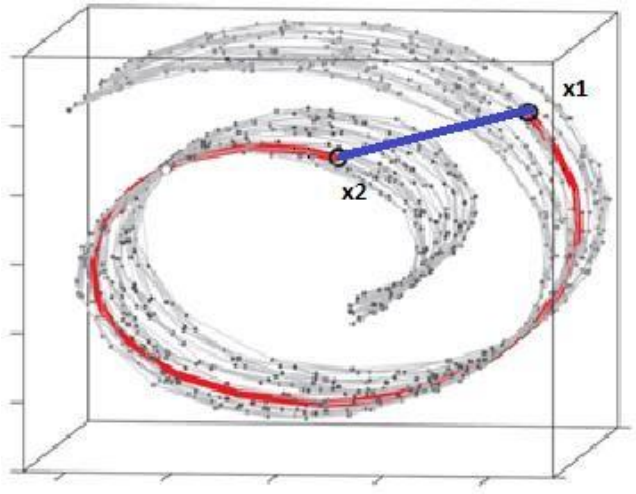
- Definition: A manifold is a topological space that locally resembles Euclidean space **near each point**.

Manifolds

- Three examples of manifolds
- All three are two-dimensional data embedded in 3D
  - Linear, “S”-shape, “Swiss roll”
- For all three examples, we would like to recover:
  - Their two-dimensional representation
  - “Consistent” coordinates of the data in the 2D

Manifolds and local distances

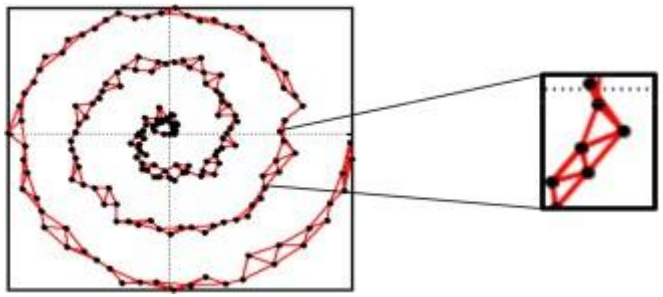
- In general, the distances induced by data (manifolds) may not be Euclidean. That is the global distances don't respect the geometry.
- Local distances can be still approximated with Euclidean distances

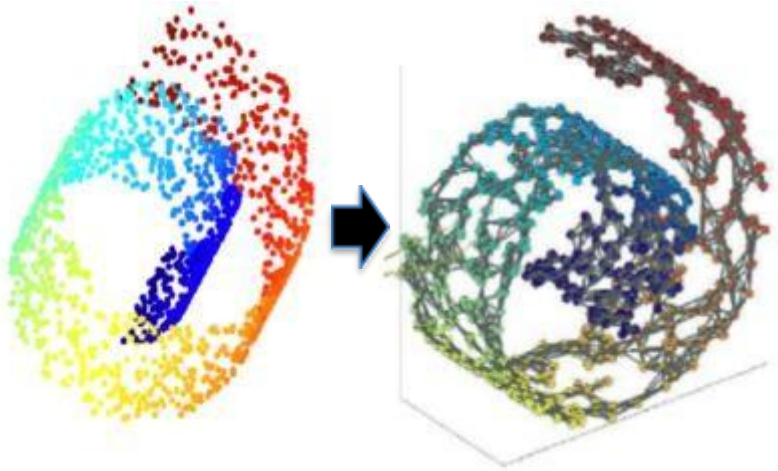


- **Idea for the dimensionality reduction:**
  - Define global distances/similarity in terms of local distances/similarity
  - Use these to define a low-dimensional embedding (low-dimensional representation) of the data
- **The idea can be implemented with the help of the Neighborhood graph**

Neighborhood Graph

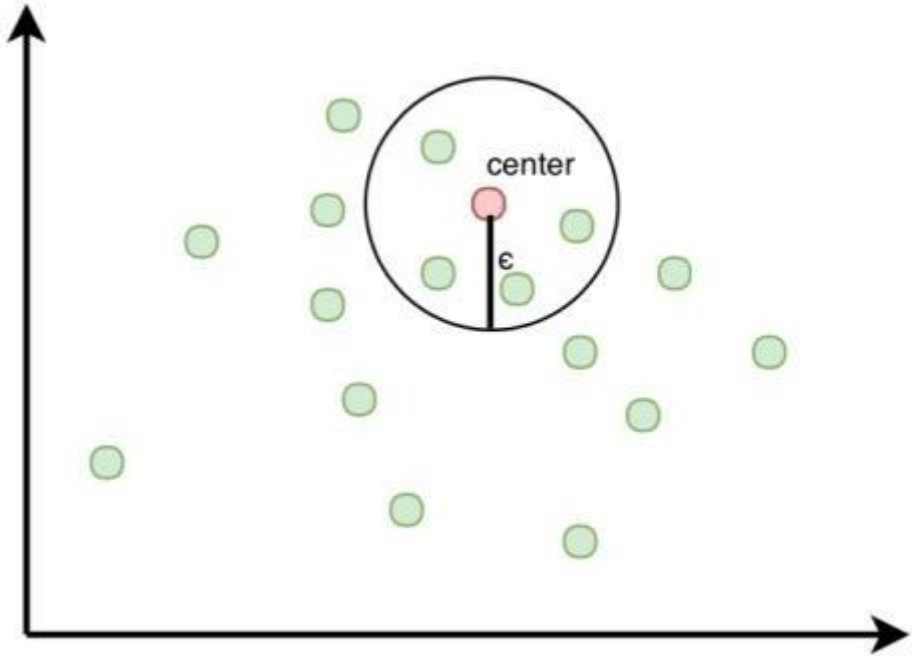
- **A neighborhood graph**
  - Vertices = data points
  - Edges and their weights reflect local similarity or local distances
  - Only points close to each other (neighbors) are connected





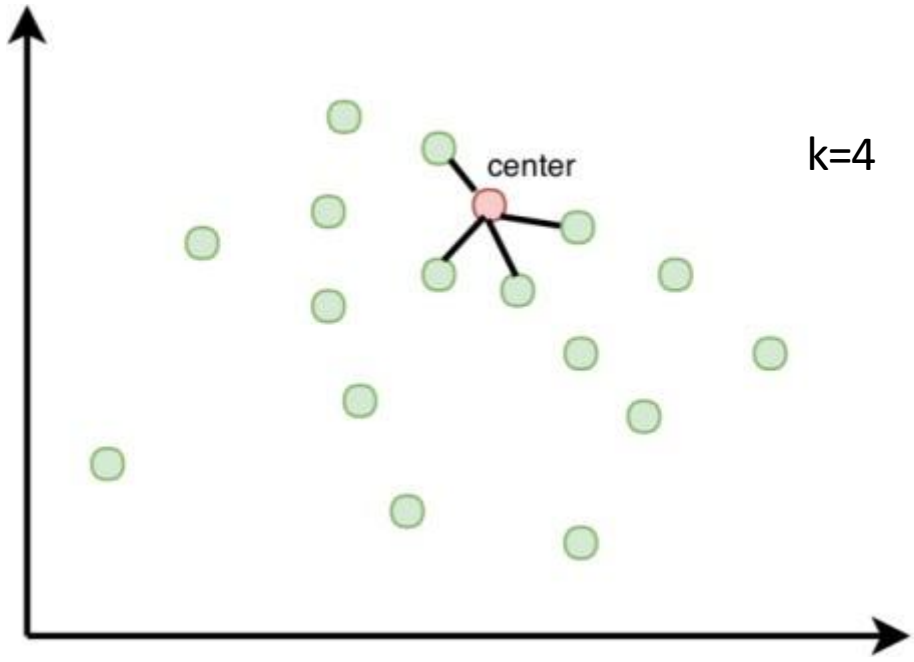
Neighborhood Graph

Approach 1: select and connect all points within the ε neighborhood



Neighborhood Graph construction

Approach 2: pick and connect k nearest neighbor points



Neighborhood Graph

- **Distance-based neighborhood graph:**
  - Edge weight: Euclidean distance between two data points – $d(x_i, x_j)$
- **Similarity-based neighborhood graph:**
  - Edge weight:
    - Simple: $W_{ij} = 1$ if connected, 0 otherwise
    - Kernel: $W_{i,j} = exp\left(-\frac{\|x_i - x_j\|^2}{t}\right)$ if connected, 0 otherwise

Non-linear dimensionality reduction

- **Three methods to define lower dimensional embedding of data instances:**
  - Isomaps
  - Locally Embedding Space (LLE)
  - Eigenmaps
- All of these rely on the neighborhood graph connecting only data instances close to each other

- First let us introduce Multi-dimensional scaling (MDS) method …

Multidimensional Scaling (MDS)

- MDS is a classical approach that can map the original high dimensional space to a lower dimensional space. It attempts to preserve the pairwise distance among the data points.
- It is used when we want to visualize high dimensional data say in 2 or 3D

Multidimensional Scaling (MDS)

Idea: MDS maps points to a low dimensional space (say of dimension k) such that the Euclidean distances between the points in this new space approximate the original distance matrix.

$$\Delta := \begin{pmatrix} \delta_{1,1} & \delta_{1,2} & \dots & \delta_{1,I} \\ \delta_{2,1} & \delta_{2,2} & \dots & \delta_{2,I} \\ \vdots & \vdots & & \vdots \\ \delta_{I,1} & \delta_{I,2} & \dots & \delta_{I,I} \end{pmatrix}$$

Map input points x_i to z_i such that

- Classical MDS: the norm $\|\cdot\|$ is the Euclidean distance $\|z_i - z_j\| \approx \delta_{i,j}$
- Objective function:

$$\min_{z_i,\dots,z_I} \sum_{i<j} \left(\|z_i - z_j\| - \delta_{i,j} \right)^2$$

Isomap

Algorithm

- **Step 1:** Construct the neighborhood graph G with edge weights corresponding to local distances
- **Step 2:** compute the shortest distance between all pairs of points:
  - The shortest distance can be calculated via Floyd or Dijkstra algorithm.

$$\Delta := \begin{pmatrix} \delta_{1,1} & \delta_{1,2} & \dots & \delta_{1,I} \\ \delta_{2,1} & \delta_{2,2} & \dots & \delta_{2,I} \\ \vdots & \vdots & & \vdots \\ \delta_{I,1} & \delta_{I,2} & \dots & \delta_{I,I} \end{pmatrix}$$

- **Step 3:** Construct the k-dimensional embedding
  - Use classical MDS to find a k-dimensional embedding

$$\min_{z_i,\dots,z_I} \sum_{i<j} \left(\left\| z_i - z_j \right\| - \delta_{i,j} \right)^2$$

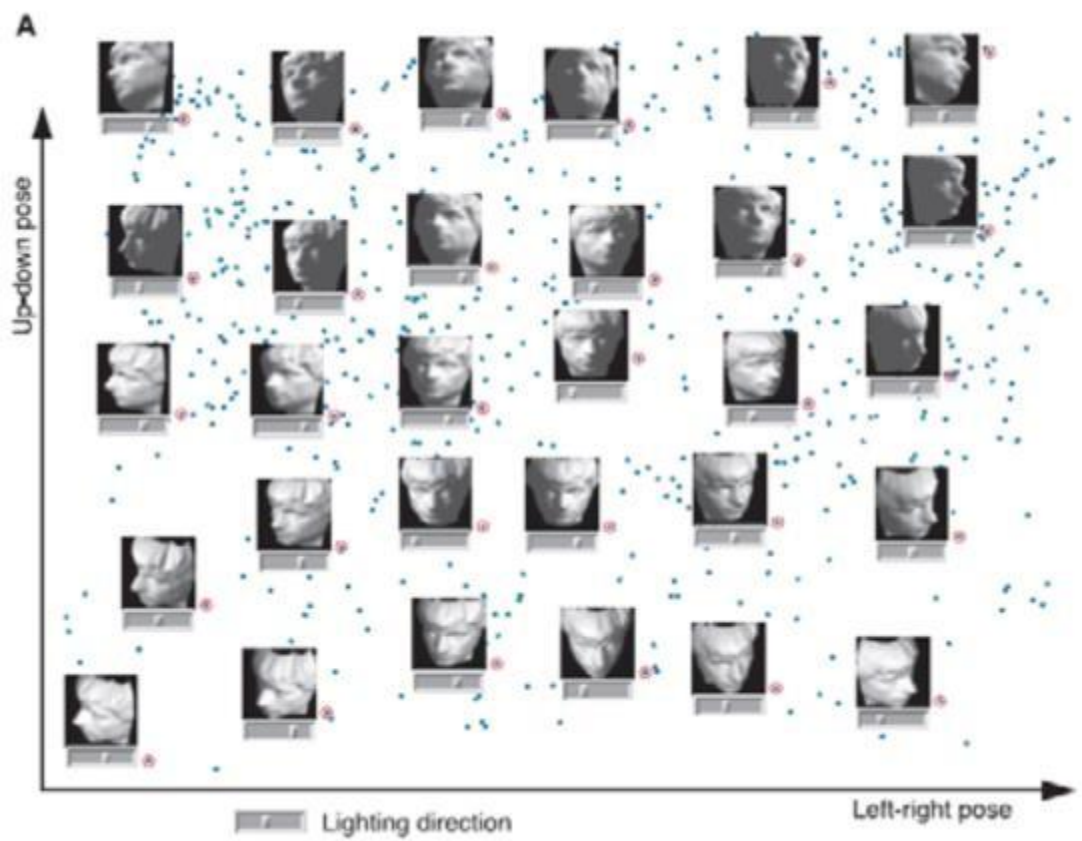
All $z_1, z_2, \dots z_I$ are k-dimensional

Isomap

- **Advantages:**
  - Non-linear dimensionality reduction
  - Non-iterative polynomial time algorithm
  - Guarantee of globally optimality:
    - For intrinsically Euclidean manifolds, a guarantee of asymptotic convergence to the true structure
    - The ability to discover manifolds of arbitrary dimensionality
- **Disadvantage:**
  - Sensitive to noise
  - Few free parameters

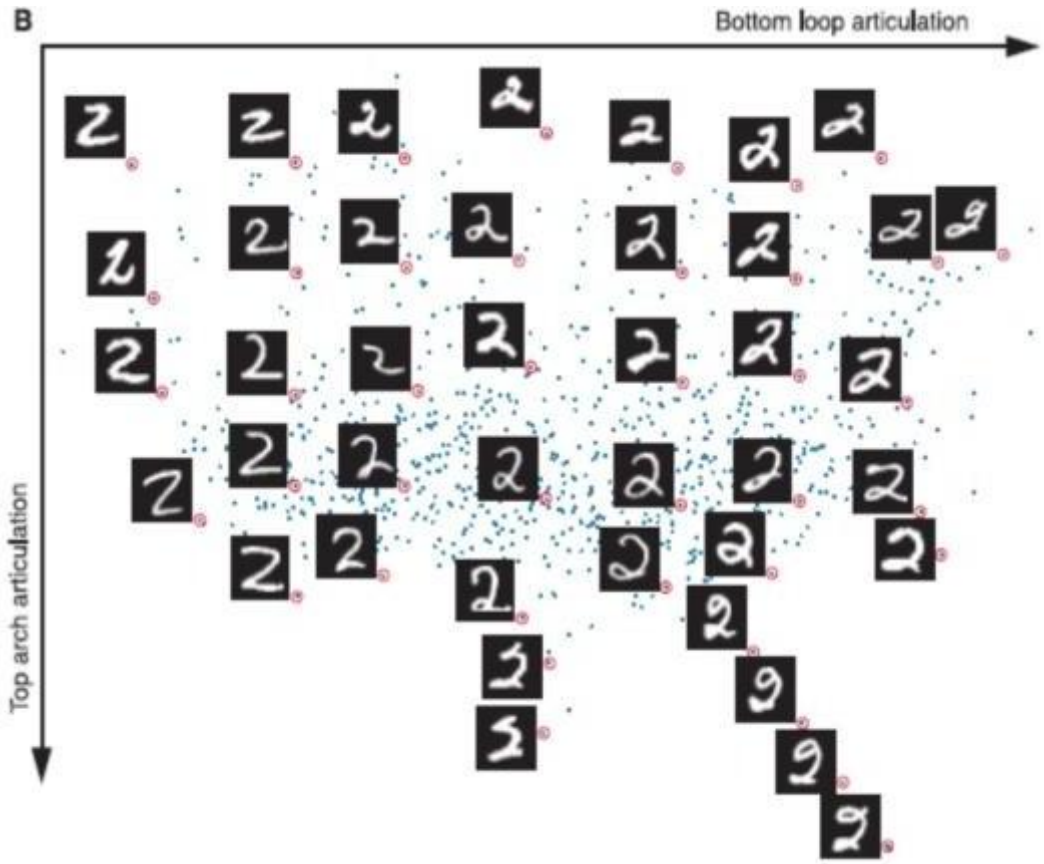
Isomaps: Example

- Dimensionality Reductior for visual perception
  - 64 x 64 image
  - 698 raw images
  - Isomap (k = 6)



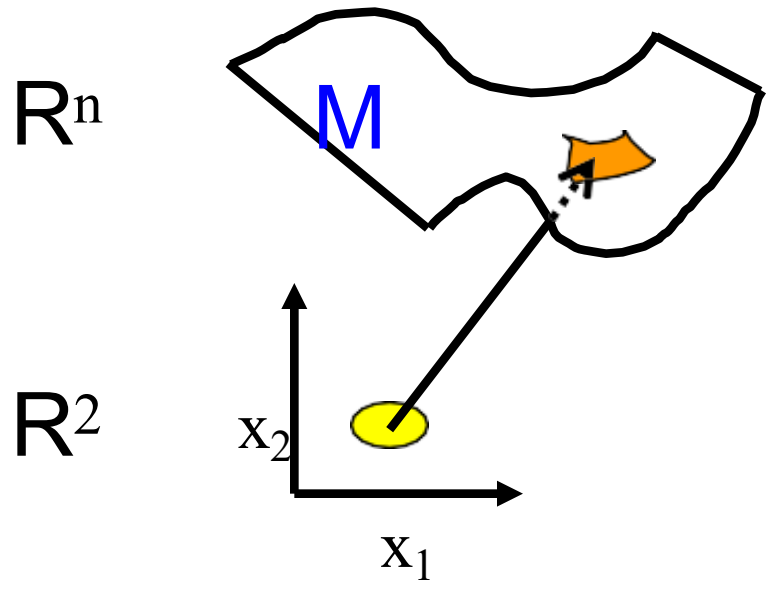
Isomap: Example

- Handwritten ‘2’
  - 1000 handwritten 2s
  - Isomap ($\epsilon = 4.2$)



Locally Linear Embedding (LLE)

- Manifold Characteristics/Key Assumption
  - We expect each data point and its neighbors to lie on or close to a locally linear patch
  - But, how to combine all local patches together?

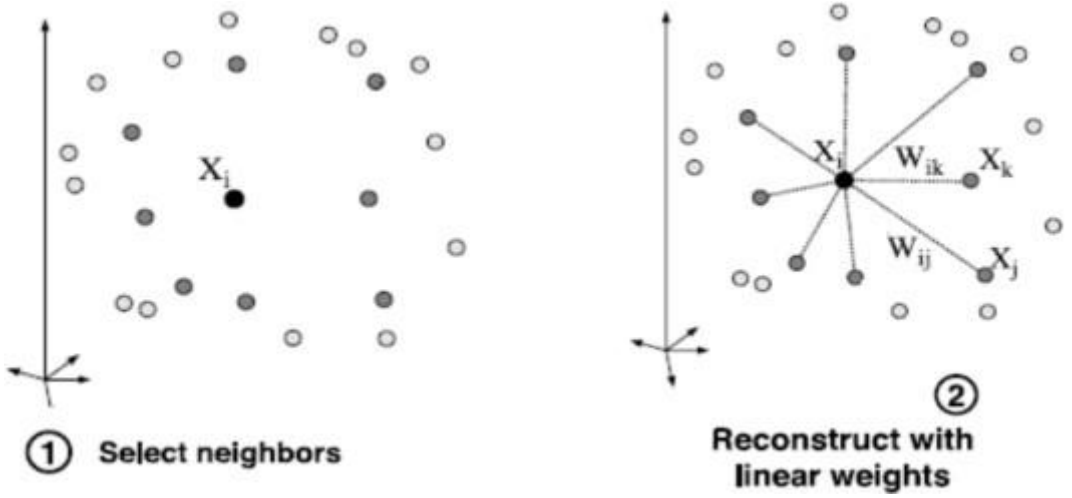


LLE: Intuition

- Assume that manifold is approximately “linear” when viewed locally (in a small neighborhood)
- A good projection should preserve this local geometric property as much as possible

LLE algorithm

- **Step 1:** Select neighbors for each data instance x_i
- **Step 2:** Each data instance is written as a convex combination of its neighbors. Weights of the convex combination 'reconstruct' each point from its neighbors.



LLE Algorithm

- **Step 2:** The weights chosen aim to minimize the reconstruction error.

$$\min_{W} \left\| X_i - \sum_{i=1}^{K} W_{ij} X_j \right\|^2$$

Note: Assign weights under two constraints:

- $W_{ij} = 0$ if X_j does not belong to set of neighbors of X_i
- The rows of the weight matrix sum to one i.e.

$$\sum_j W_{ij} = 1$$

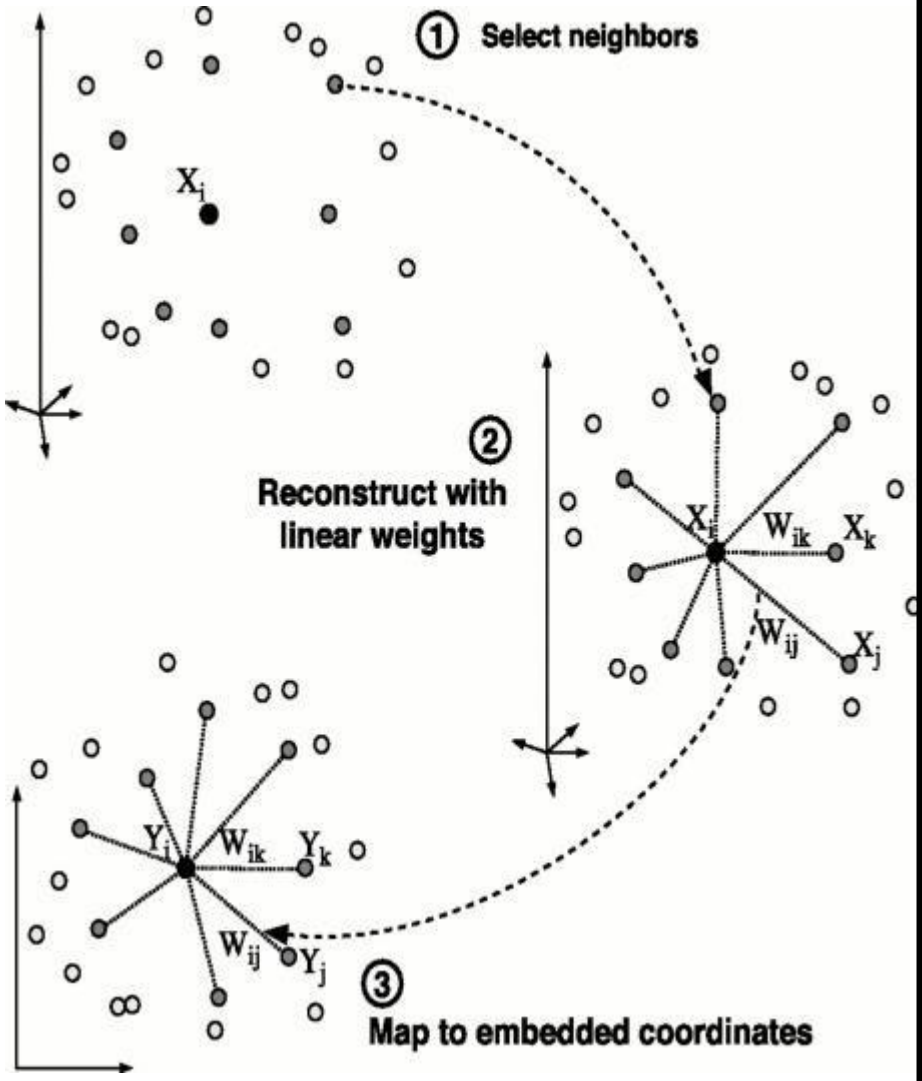
LLE Algorithm

- **Step 3:** Map R^n to a low-dimensional embedding R^k

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}^{R^n} \rightarrow \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}^{R^k}$$

The cost function can be minimized by solving a sparse NxN eigenvalue problem:

$$\min_y \sum_{i=1} \left\| Y_i - \sum_j W_{ij} Y_j \right\|^2$$



LLE: Eigenvalue Problem

The following is a more direct and simpler derivation for Y:

Define $M = (I - W)^T (I - W)$

$$\Phi(Y) = \sum_i \left\| Y_i - \sum_j W_{ij} Y_j \right\|^2 = \sum_i \left\| Y_i - [Y_1, Y_2, \ldots, Y_n] W_i^T \right\|^2$$

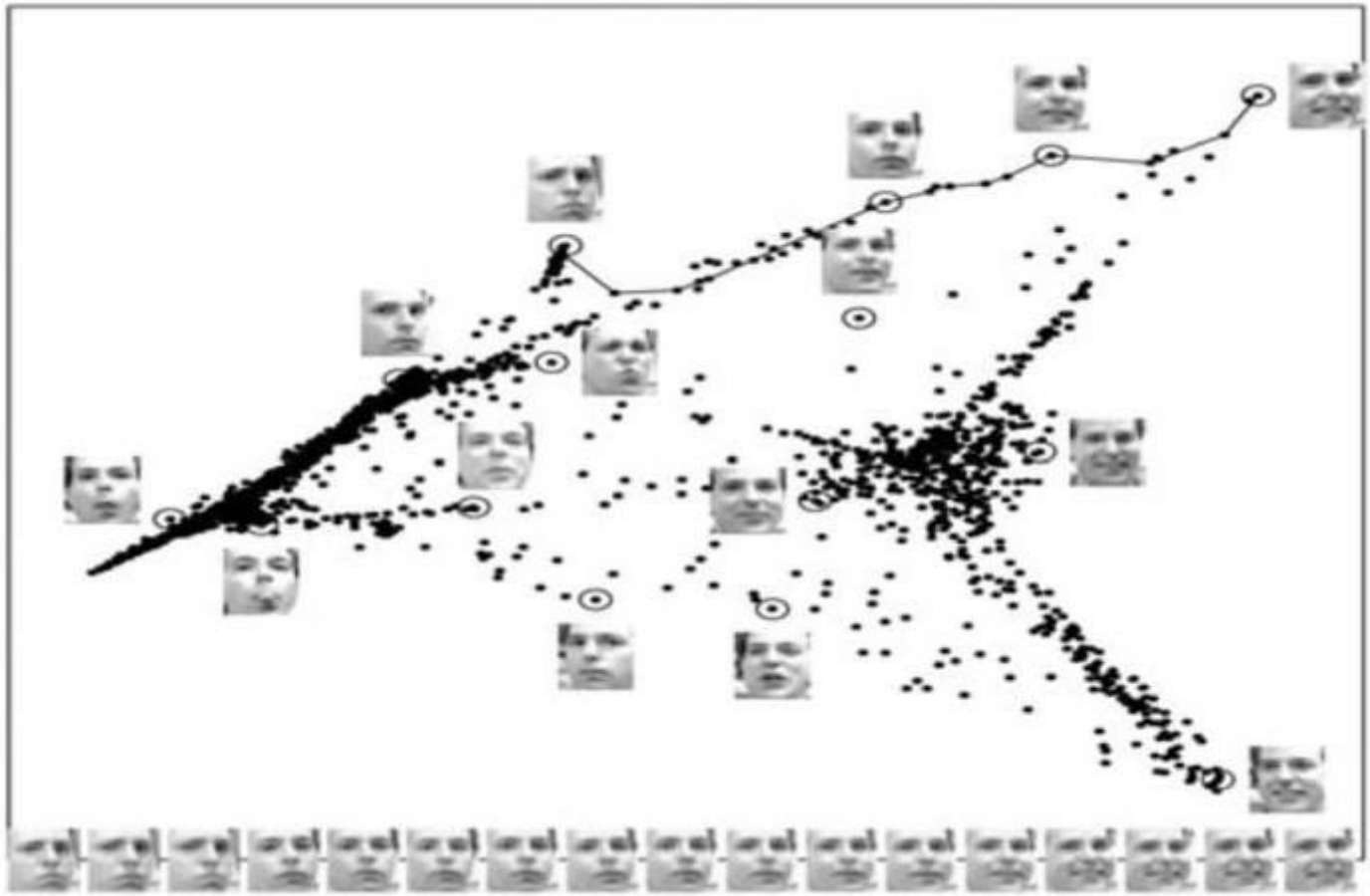
$$= \left\| [Y_1, Y_2, \ldots, Y_n] - [Y_1, Y_2, \ldots, Y_n][W_1^T, W_2^T, \ldots, W_n^T] \right\|_F^2$$

$$= \left\| Y - YW^T \right\|_F^2 = \left\| Y(I - W^T) \right\|_F^2 = trace(Y(I - W)^T (I - W) Y^T)$$

$$= trace(YMY^T)$$

where $Y = [Y_1, Y_2, \ldots, Y_n]$

LLE Example



Images of faces mapped into the embedding space described by the first two coordinates of LLE. Representative faces are shown next to circled points. The bottom images correspond to points along the top-right path (linked by solid line) illustrating one particular mode of variability in pose and expression.

LLE: effect of k

- Require dense data points on the manifold for good estimation

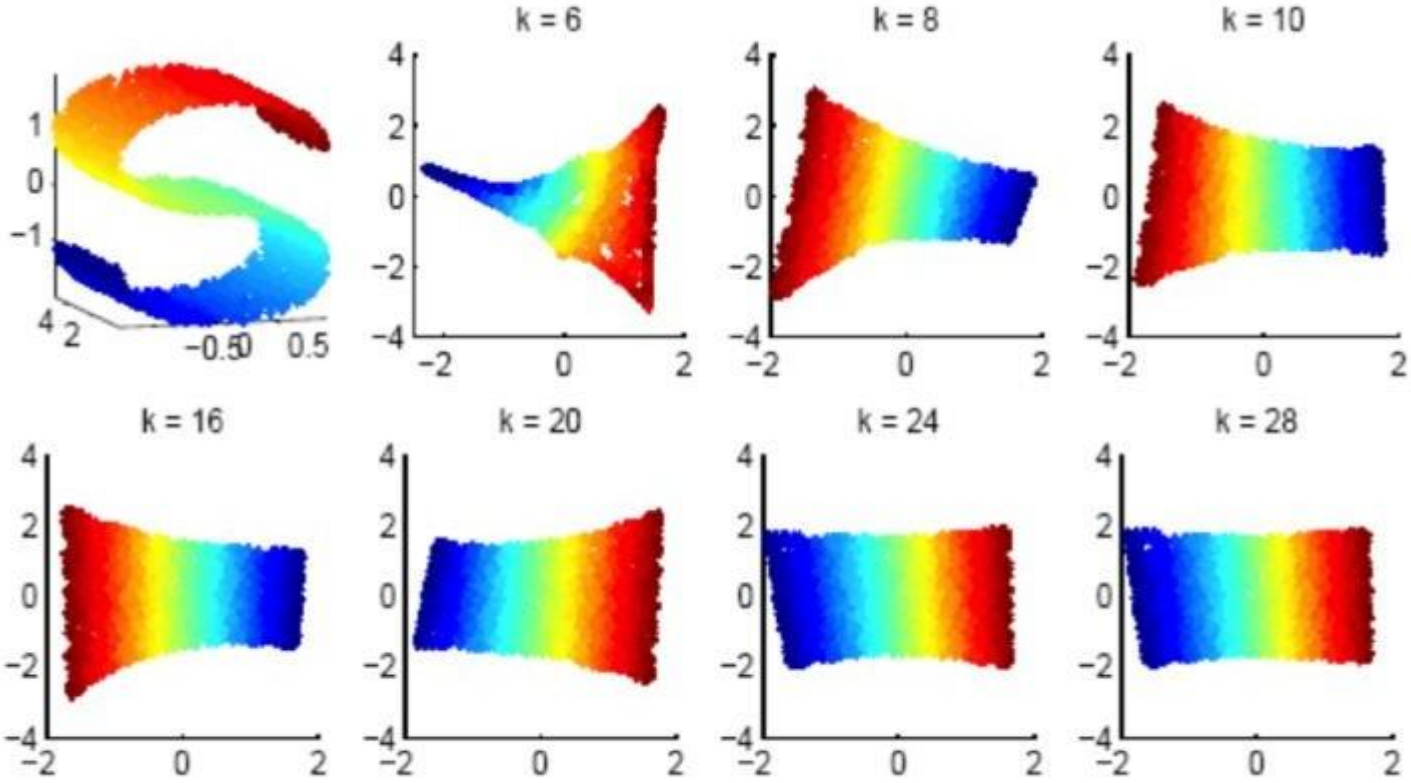


FIG. 5. *S-curve (top left) and computed 2D coordinates by LLE with various neighborhood size k.*

Limitations of LLE

- require dense data points on the manifold for good estimation
- A good neighborhood seems essential to their success
  - How to choose k?
  - Too few neighbors
    - Results in rank deficient tangent space and lead to over-fitting
  - Too many neighbors
    - Tangent space will not match local geometry well

Laplacian Eigenmaps

- Problem: Given a set (x_1, x_2, …, x_n) of n points in R^d, find a set of points (y_1, y_2,…,y_n) in R^k ($k << d$) such that y_i represents x_i.

- Steps
  - Build the adjacency graph
  - Choose the weights for edges in the graph
  - Eigen-decomposition of the graph Laplacian
  - Form the low-dimensional embedding

Laplacian Eigenmaps Algorithm

- **Step 1:** Construct the neighborhood graph with weights modeling local similarities:
  - Simple: 1 if connected; 0 otherwise
  - Kernels: Gaussian or Heat kernels

$$W_{ij} = e^{-\frac{\left\| x_i - x_j \right\|^2}{t}}$$

- **Step 2:** Construct Graph-Laplacian matrix
  - Construct diagonal weight matrix D from the weight matrix

$$D_{ii} = \sum_j W_{ji}$$

  - Construct Laplacian matrix L = D-W
  - Laplacian is a symmetric, positive semi-definite matrix

Laplacian Eigenmaps Algorithm

- **Step 3:** Finding of the lower dimensional embedding
  - Find $Y = (y_1, y_2, \ldots, y_n)^T$ that preserves local similarities
  - Note: each y is low-dimensional

  Objective: minimize

$$\sum_{ij} (y_i - y_j)^2 W_{ij}$$

  - The above objective function can be rewritten as:

$$\frac{1}{2} \sum_{i,j} (y_i - y_j)^2 W_{ij} = Y^T L Y$$

  - **Solution:** Compute eigenvalues and eigenvectors of the generalized eigenvector problem

Generalized Eigenvector Problem

- Suppose we have the n points such that (x_1, x_2, …, x_n) in R^d
- Construct the weight matrix between each point

$$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n1} & \cdots & w_{nn} \end{pmatrix}$$

- Based on equation $|W - \lambda I| = 0$, we can compute the Eigenvalues $\lambda = (\lambda_0, \lambda_1, \ldots, \lambda_n)$

Euclidean Embedding Space

Once we have all the Eigenvalues, then we solve equation $Lv = \lambda Dv$ to compute the Eigenvector v

$$Lv = \lambda Dv$$

$$Lv - \lambda Dv = 0$$

$$(L - \lambda D)v = 0$$

Since $(L - \lambda D)$ has the dimension $n * n$

Then Eigenvector v has the dimension $n * 1$

Euclidean Embedding Space

- Let $v_0, v_1, \ldots, v_{n-1}$ be the eigenvector solutions to the equation $Lv = \lambda Dv$, ordered according to their eigenvalues,

$$
\begin{aligned}
Lv_0 &= \lambda_0 Dv_0 \\
Lv_1 &= \lambda_1 Dv_1 \\
&\ldots \\
Lv_{n-1} &= \lambda_{n-1} Dv_{n-1} \\
Lv_n &= \lambda_n Dv_n \\
0 = \lambda_0 &\leq \lambda_1 \leq \ldots \leq \lambda_{n-1}
\end{aligned}
$$

Euclidean Embedding Space

- Then we leave out the eigenvector v_0 corresponding to eigenvalue λ_0 and use the next k eigenvectors $(v_1, v_2, \ldots, v_k)$ for embedding in k-dimensional Euclidean space.
- The k-dimensional embedding space is denoted as $Y = (y_1, y_2, \ldots, y_k)^T$

Euclidean Embedding Space

Once we find all Eigenvectors, then, we can project the data points to lower dimension R^k embedding space

$$X=(x_1,x_2,...,x_n)^{R^d} \rightarrow Y=\begin{matrix} y_1 & y_2 & \text{L} & y_k \end{matrix}\begin{pmatrix} v_1(1) & v_2(1) & ... & v_k(1) \\ v_1(2) & v_2(2) & ... & v_k(2) \\ \vdots & \vdots & \ddots & \vdots \\ v_1(n) & v_2(n) & ... & v_k(n) \end{pmatrix}^T$$

We reduce the dimension from n * d to n * k

$y_1 = (v_1(1), v_1(2), \ldots, v_1(n))$represents the coordinates of all n data points at 1st dimension in the embedding space.

Optimal embedding space

- Objective function:

$$\min \frac{1}{2}\sum_{i,j}(y_i - y_j)^2 W_{ij} \qquad \underset{\substack{y^T Dy=1 \\ y^T D1=0}}{\operatorname{argmin}} y^T Ly$$

- The constraint $y^T Dy = 1$ removes arbitrary scaling factor
- The constraint $y^T D1 = 0$ removes a translation invariance in y

Laplacian Eigenmaps: Embedding Space

After the optimization, each data point can be maped into the optimal k-dimensional embedding space $Y = (y_1, y_2, \ldots, y_k)^T$

$$x_i^{R^{dn}} \rightarrow (v_1(i), v_2(i), \ldots, v_k(i))$$

$v_j(i)$ is the coordinate of point x_i at jth dimension at k-dimensional space, where $1 \leq j \leq k$

Laplacian Eigenmaps Example

- Suppose we want to project $X = \{x_1, x_2, ..., x_n\}^{R^m}$ into 2 dimensional space
- Let $\{\lambda_0, \lambda_1, \lambda_2 ..., \lambda_n\}$ to be the calculated eigenvalues with increasing order $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_n$
- Let $v = \{v_1, v_2, ..., v_n\}$ be the eigenvector solutions of equation $Lv = \lambda Dv$
- Take $\{v_1, v_2\}$ for embedding in 2-dimensional space

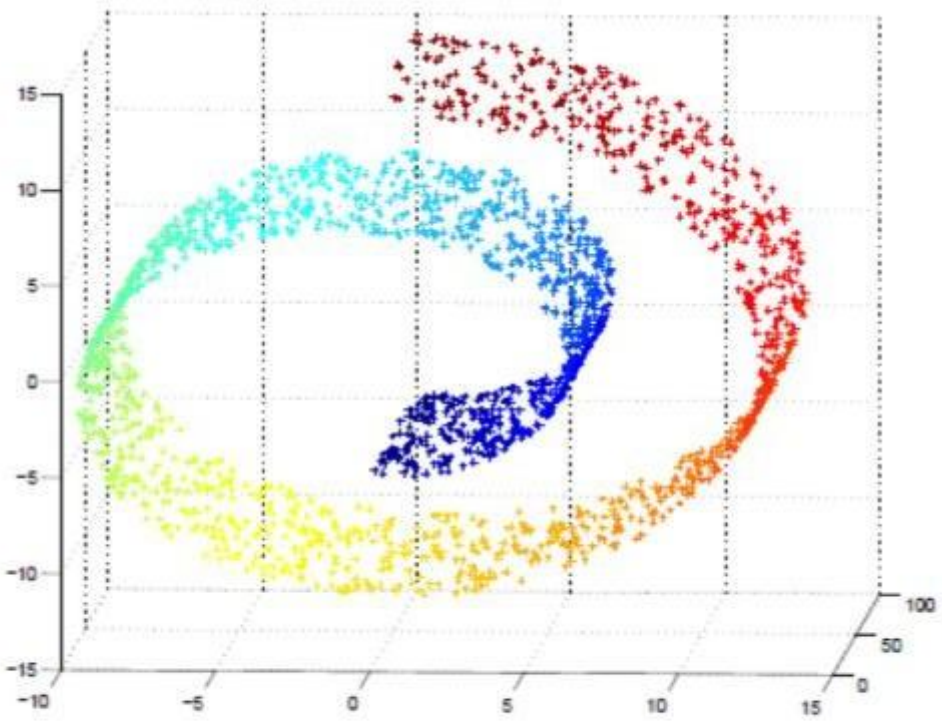
$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}^{R^m} \rightarrow \begin{pmatrix} v_1(1) & v_2(1) \\ v_1(2) & v_2(2) \\ \vdots & \vdots \\ v_1(n) & v_2(n) \end{pmatrix}^{R^2}$$

Locally Linear Embedding and Laplacian Eigenmap

- LLE is connected with Laplacian Eigenmap
- LLE minimizes $Y^T(I-W)^T(I-W)y$ to reduce eigenvectors of $(I-W)^T(I-W)$
- Actually, finding eigenvectors of $(I-W)^T(I-W)$ can be re-interpreted as finding eigenvectors of iterated Laplacian L^2.

Laplacian Eigenmap Example

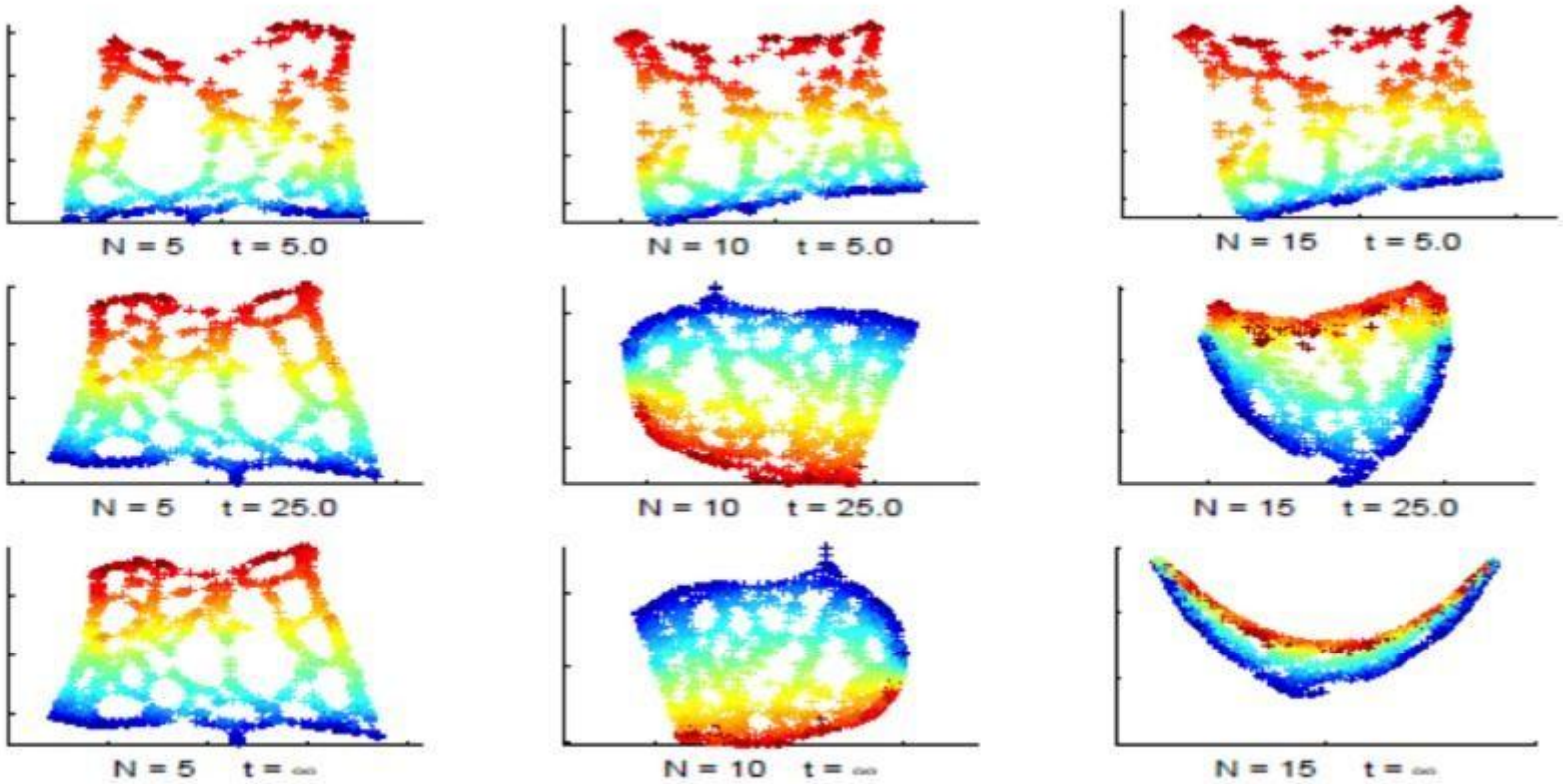
- Swiss roll



2000 random data points on the manifold

Laplacian Eigenmap Example

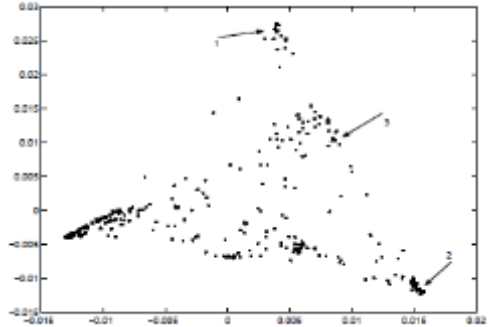
- 2D embedding of the swiss roll

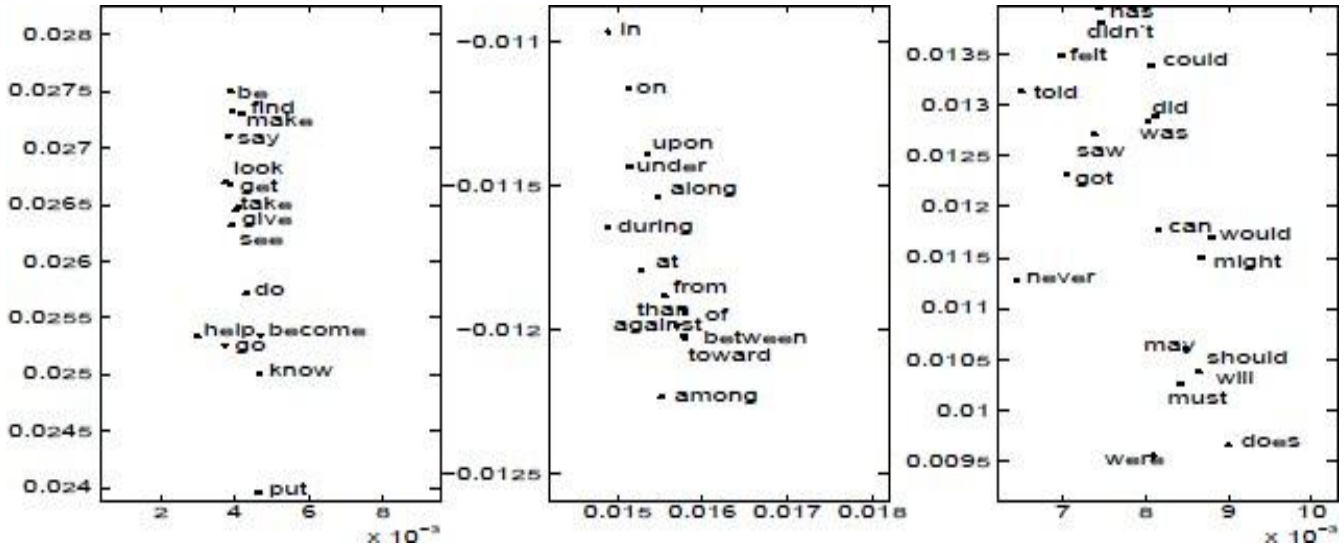


Free parameters, N and t. N = Number of neighbors, t = Heat kernel parameter

Laplacian Eigenmap Example

- 300 most frequent words from Brown corpus
- Each word is represented by a 600 dimensional vector
- Laplacian Eigenmap with N = 14, t = inf





Framgents labeled by arrows, from left to right. The first is exclusively infinites of verbs, the second contains prepositions and the third mostly modal and auxiliariy verbs

Summary

- Isomap, LLE and Laplacian Eigenmap: non-linear dimensionality reduction technique
- Useful for learning manifolds, understanding low dimensional data embedded in high dimensional space.
- Linear dimensionality reduction technique (PCA, SVD) fails for this type of data.
- All three can preserve local geometry (inter-point relationships)

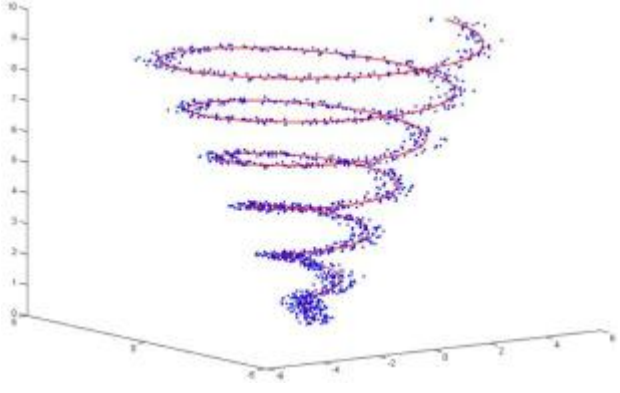
References and acknowledgments

- Roweis, S. T. & Saul, L. K. (2000), 'Locally linear embedding', Science 290, 2323–2326.
- Tenenbaum, J. B., de Silva, V. & Langford, J. C. (2000), 'A global geometric framework for nonlinear dimensionality reduction', Science 290, 2319–2323.
- http://www.cs.nyu.edu/~roweis/lle/
- http://isomap.stanford.edu/
- http://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction
- http://web.mit.edu/6.454/www/www_fall_2003/ihler/slides.pdf
- http://cseweb.ucsd.edu/~saul/teaching/cse291s07/laplacian.pdf
- www.public.asu.edu/~jye02/CLASSES/Spring.../Lec19-Isomap.ppt
- www.public.asu.edu/~jye02/CLASSES/Spring.../Lec20-LLE.ppt
- www.public.asu.edu/~jye02/CLASSES/.../Lec21-LaplacianEig.ppt

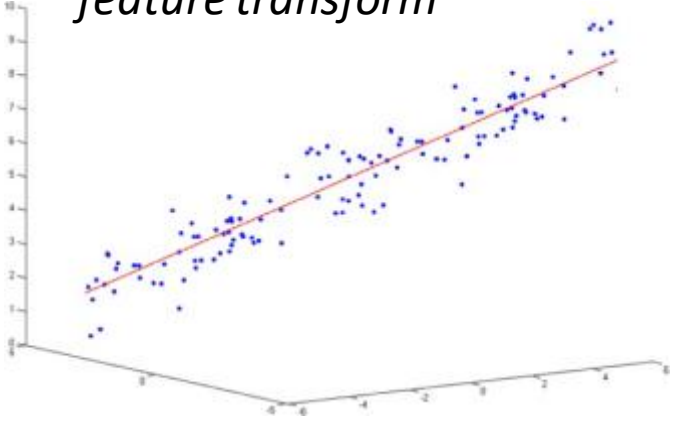
Kernel-PCA (k-PCA)

Can we kernelize PCA to implicitly do a linear dimension reduction (ie PCA) in a non-linear feature space?

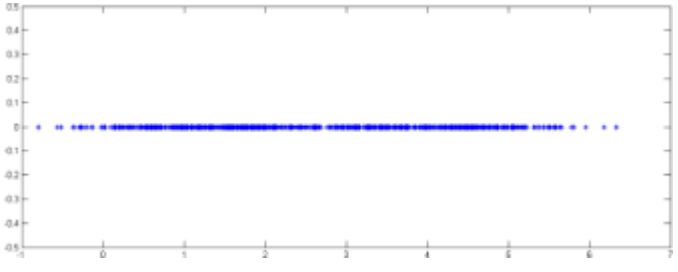
High dimensional manifold data



Explicit non-linear feature transform



PCA



Kernel-PCA
(done implicitly)

Kernel-PCA (k-PCA)

How to kernelize PCA?

Observation:
PCA directions are the principal eigenvectors of the (centered) matrix XX^T
But… we want the inner product X^TX

Let $X = USV^T$

- Since $XX^T = US^2U^T$, we are interested in finding how any datapoint x is projected onto the vectors U, ie the projection U^tx

If data is in a feature space $\phi(X)$

- Problem: cannot compute U (no efficient way to compute outer product)
- If dot product can be computed efficiently, then easy to compute V and S
- Since $U = \phi(X)\ V\ S^{-1}$
- For any datapoint $\phi(x)$, its projection $U^T\ \phi(x) = S^{-1}\ V\ \phi(X)\ .\ \phi(x)$

can be computed efficiently!

Kernel-PCA (k-PCA)

We can view many of the manifold embedding methods as a special case of kernel PCA (with specific choice of the kernel)!

- PCA: $K = X^T X$ (linear kernel)
- Classical-MDS: $K = -\frac{1}{2} H D^{eucl} H$, where H is the centering matrix
  (Euclidean) distance gets converted into inner product matrix K
- Isomap: $K = -\frac{1}{2} H D^{geodesic} H$
  Basically MDS with geodesic distances
- Locally Linear Embedding: $A := (I - W)(I - W)^T$; $K = L^{-1}$ or $K = (\lambda_{max} I - A)$
  W are the learned locally linear weights from LLE
- Laplacian Eigenmap: $K = L^{-1}$ or $K = (\lambda_{max} I - L)$
  L is the graph Laplacian used in LE

Kernel-PCA (k-PCA)

Observation

- Many manifold embedding methods assume a specific kernel form

Idea: why don't we directly ***learn the kernel*** which yields a good embedding?

Maximum Variance Unfolding (MVU) (aka Semi-Definite Embedding (SDE))

Maximum Variance Unfolding (MVU)

In order to find an optimal kernel for non-linear embedding, we need to define what we want the embedding to do.

Goal: Find a low-dim. map that preserves the "local geometry"

local geometry = distances between neighboring points!

So, how to learn a kernel that preserve the local distances?

For points x_i and x_j in any neighborhood

$$\begin{aligned} ||x_i - x_j||^2 &= ||\phi(x_i) - \phi(x_j)||^2 \\ &= K_{ii} + K_{jj} - 2K_{ij} \end{aligned}$$

- K needs to be PSD
- Σ_{ij} K_{ij} = 0

Can formulate the optimization as follows…

Maximum Variance Unfolding (MVU)

[Weinberger and Saul '04]

Maximize$_K$ tr(K)

Maximize the spread of points

Constraints:

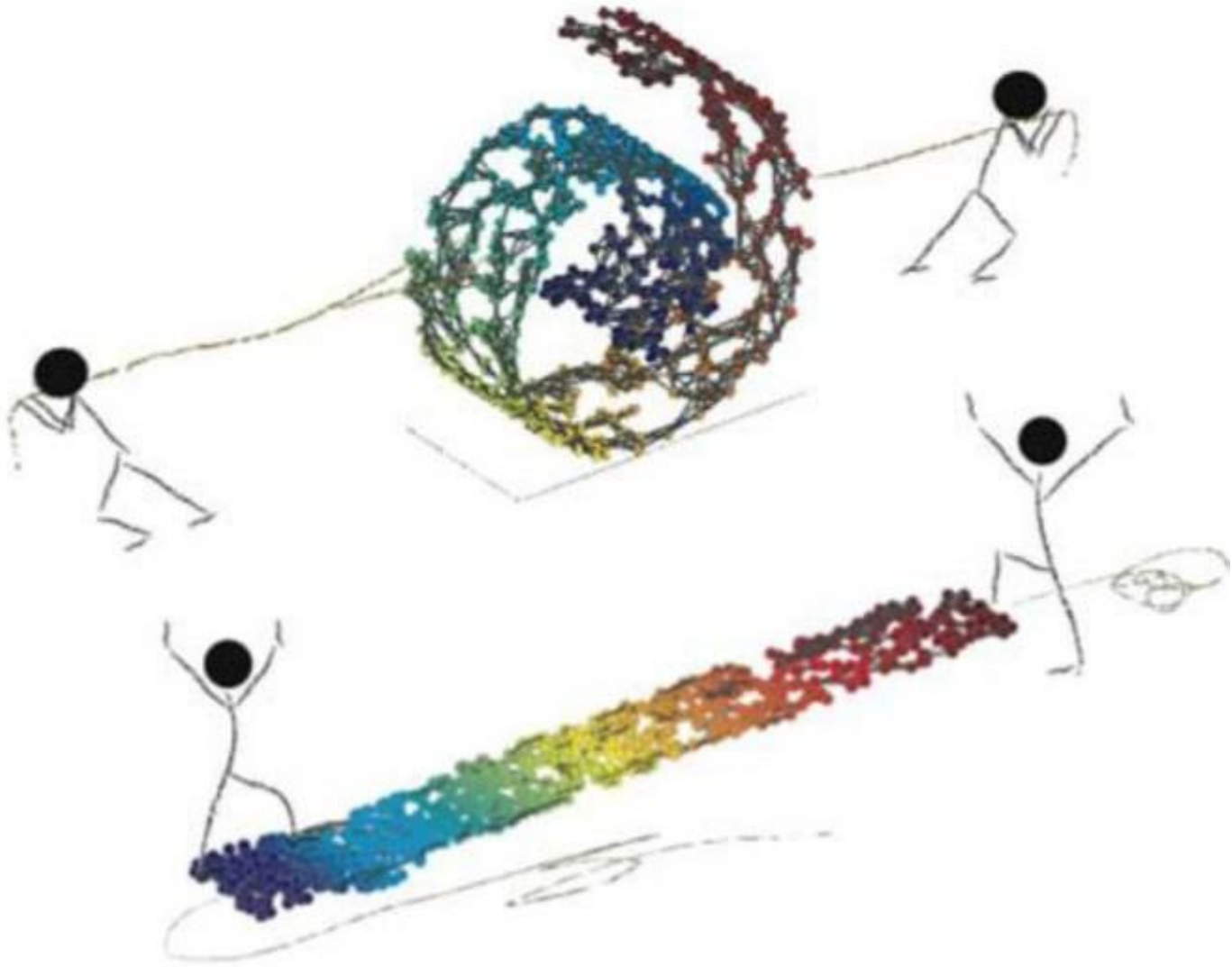
$$K_{ii} + K_{jj} - 2K_{ij} - ||x_i - x_j||^2 \text{ if } i \text{ and } j \text{ are neighbors.}$$

$$\sum_{ij} K_{ij} = 0$$

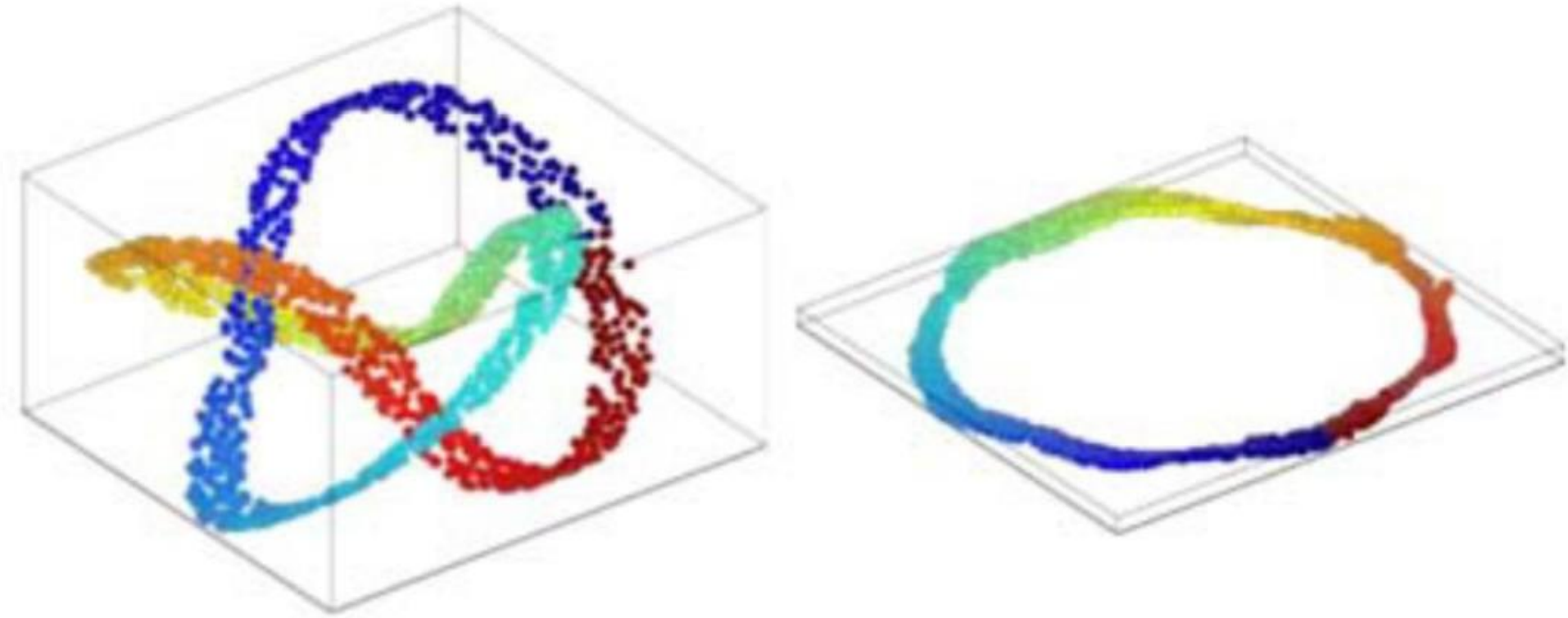
$$K \succeq 0$$

Can be solved by any SDP solver

Maximum Variance Unfolding (MVU)



Maximum Variance Unfolding (MVU)



Stochastic Neighbor Embedding (SNE)

Goal: Find a low-dim. map that preserves the "local geometry"

local geometry = similarity between points in local neighborhoods

Similar goal to LE, BUT a drastically different solution!

Idea:

Model the neighborhood structure/information as a probability distribution, then find a low-dimensional mapping that matches the same distribution!

Notation:

- $x_1, \ldots, x_n$ given high dim. data (given)
- $y_1, \ldots, y_n$ mapped low dim. Representation (to be learned)
- $p_{j|i}$ = probability of x_j being the neighbor of x_i (computed from data)
- $q_{j|i}$ = probability of y_j being the neighbor of y_i (to be matched to $p_{j|i}$)

Stochastic Neighbor Embedding

[Hinton and Roweis '03]

Stochastic Neighbor Embedding approach:

Probability model for high-dim input data

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

Meta parameter controlling the neighborhood size

Probability model for low-dim mapped data

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

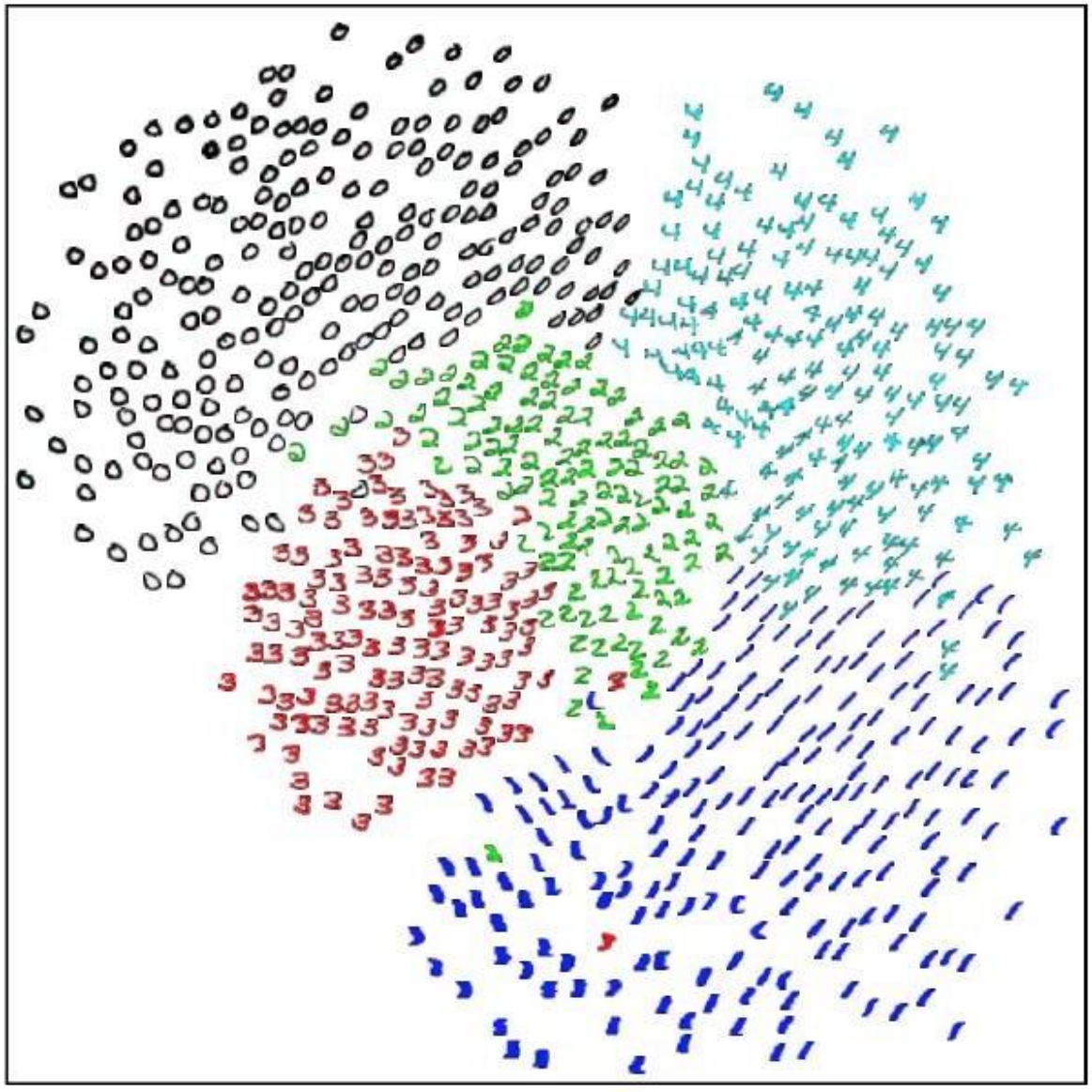
y's are the variables that need to be learned

Key optimization: Maximize the similarity between the distributions

$$\text{minimize}_y \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

Highly non-convex, just do gradient descent and settle with the local optimal solution

Stochastic Neighbor Embedding



The individual class clusters are well all together producing an effective visualization

But the clusters are NOT well separated

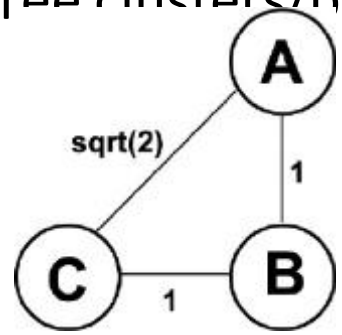
The issue: “crowding problem”

t-distributed Stochastic Neighbor Embedding

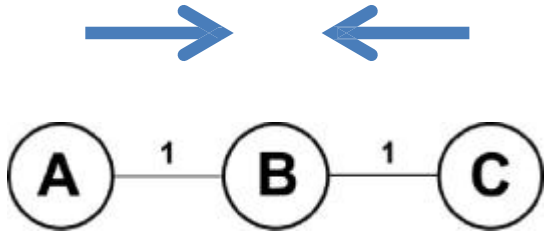
The crowding problem:

High dimensional data is being cramped into a low dimensional space, to match the probabilities, the clusters can "crowd" together

Consider three clusters/points A, B, C



Organization in high dimensions



Organization in low dimensions

Because of the gaussian-type neighborhood structure in low dimensions, large distance between A and C will be **penalized** a lot causing them to be mapped close (ie crowd) to each other

t-distributed Stochastic Neighbor Embedding

[Van der Maaten and Hinton '08]

Solution to the crowding problem

Idea: instead of using a thin-tailed Gaussian in the lower dimensions, we can use a heavier-tailed distribution, e.g. student's t-distribution!

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

Symmetrize the high dimensional neighborhood distribution

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

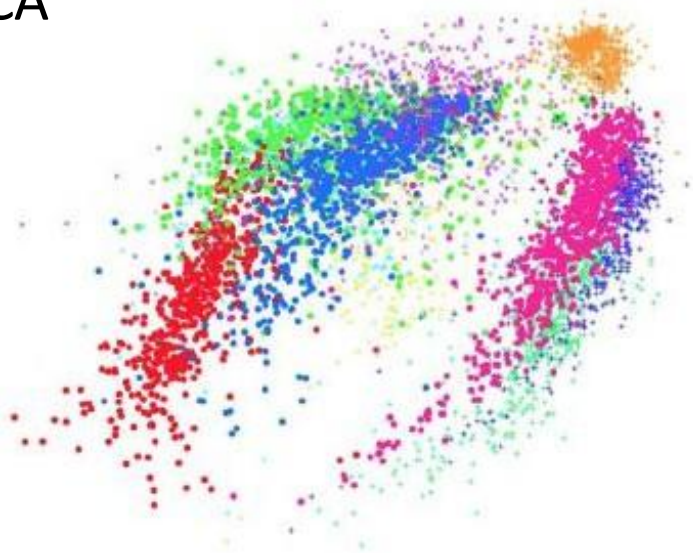
Use the heavier tailed student's t-distribution

Final optimization:

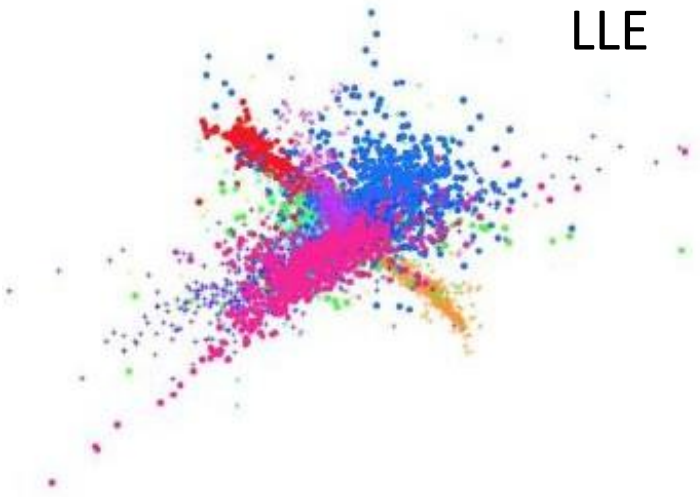
$$\text{minimize}_y \quad \sum_i KL(P_i||Q_i) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

t-SNE

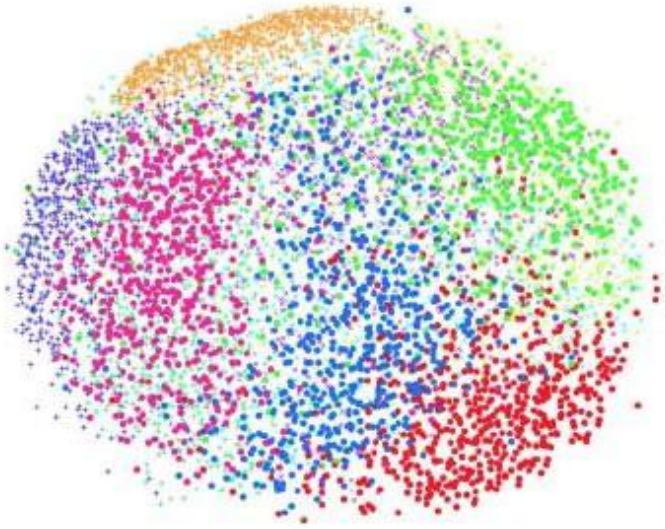
PCA



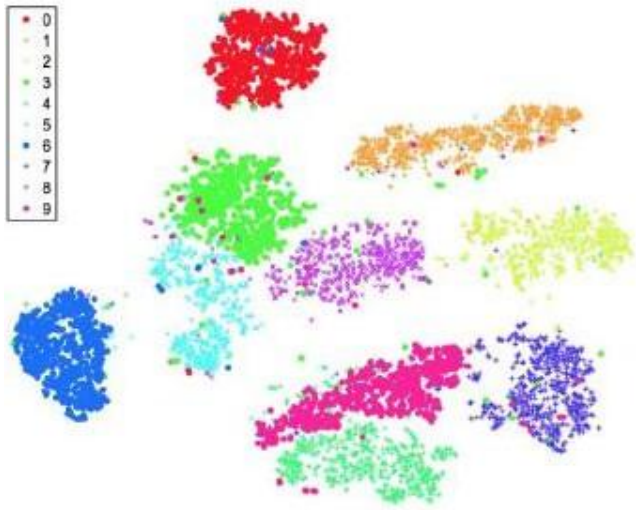
LLE



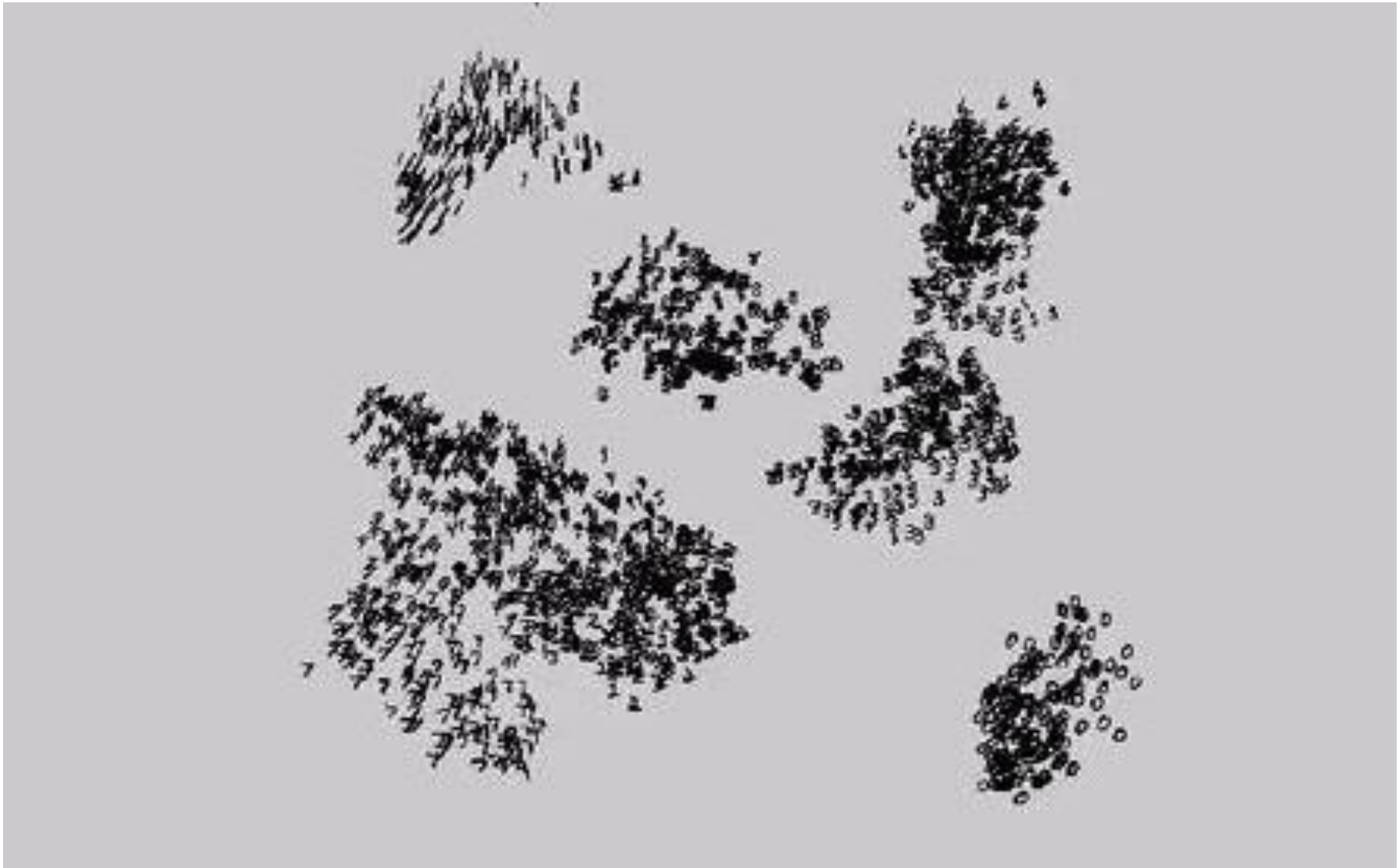
Sammon mapping



t-SNE

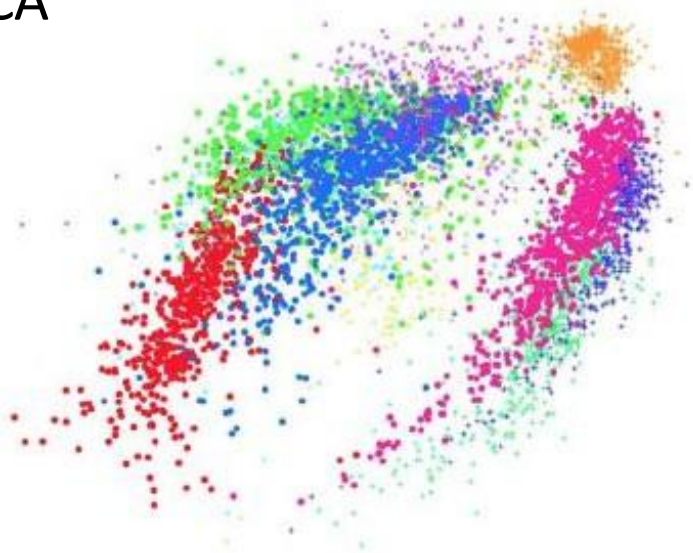


t-SNE

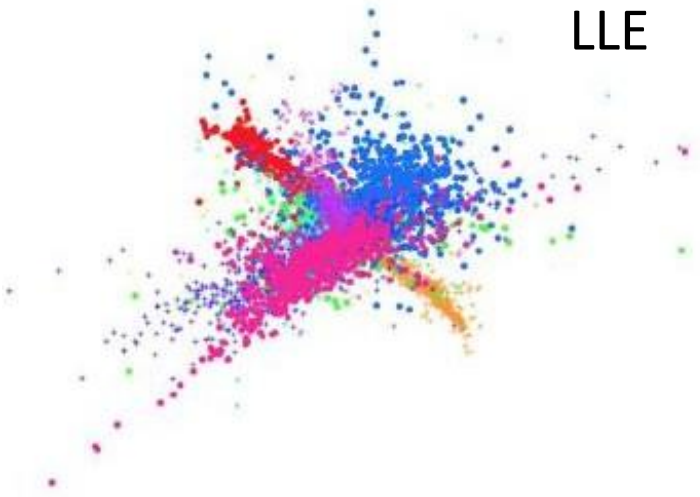


t-SNE

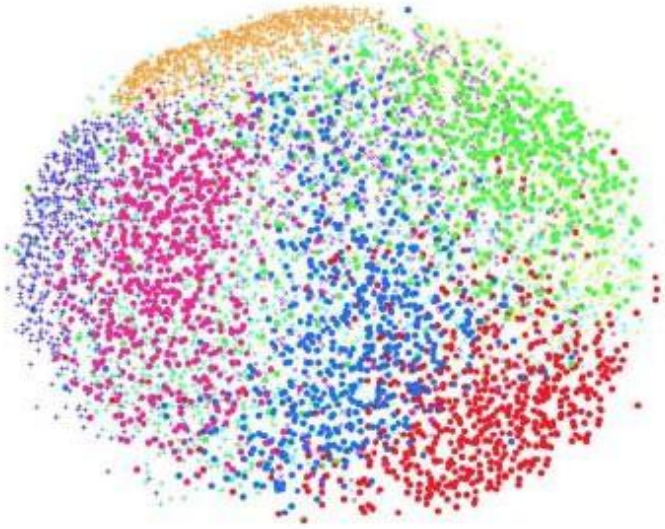
PCA



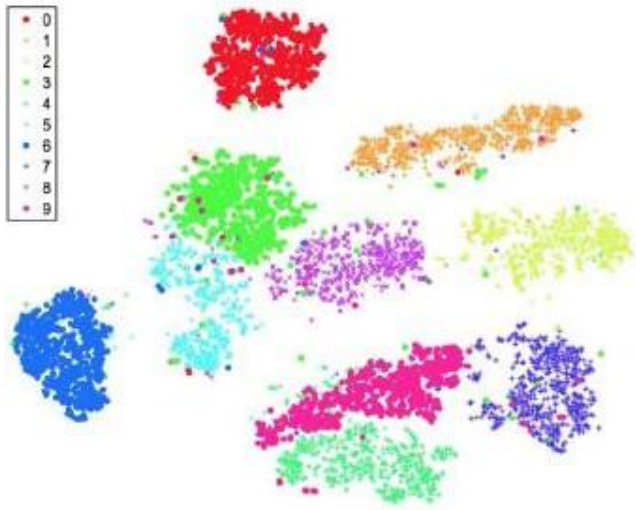
LLE



Sammon mapping



t-SNE



t-SNE thoughts

- Remarkably effective in visualizing cluster structure in data

- Arguably the best ultra low-dimensional technique that "just works"!

  A few contenders are popping up (eg. UMAP)

  [McInnes, Healy and Melville '18]

- Tends to cluster even when there may not be any clusters!

  Can result in false cluster discovery

  [Im, Verma and Branson '18]

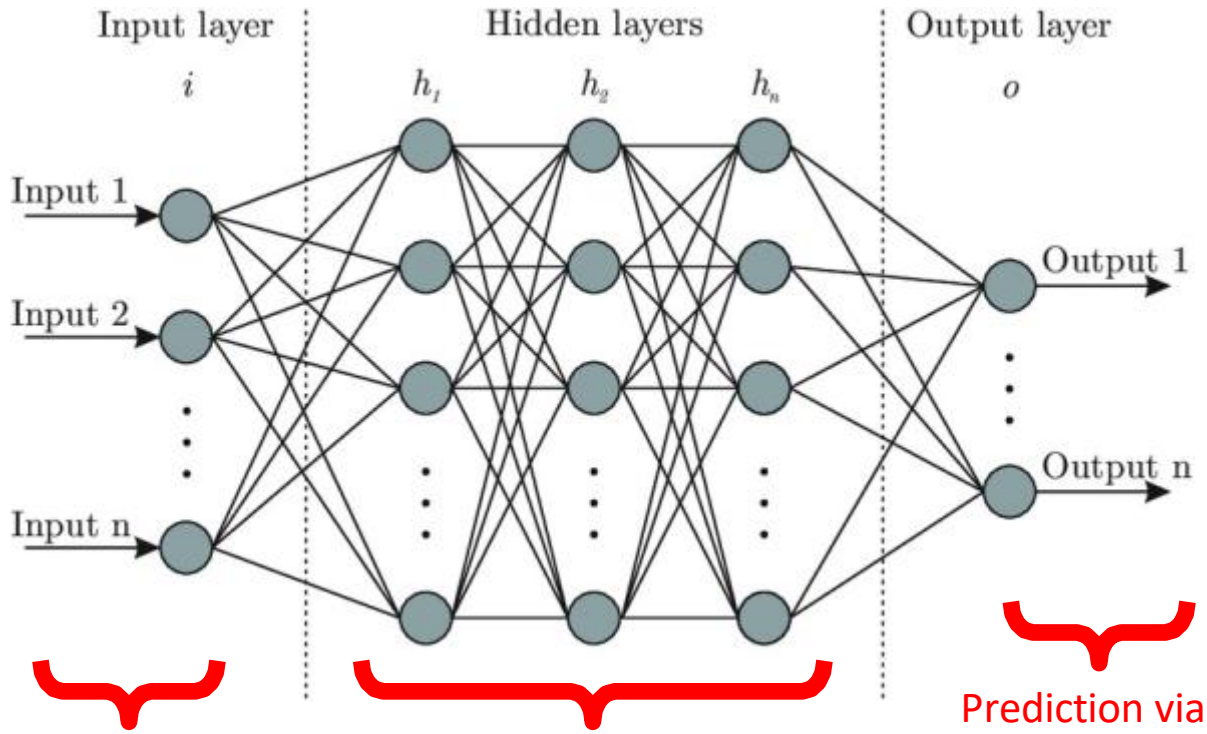
- Recent results have quantified the sufficient conditions needed for t-SNE to be provably successful in revealing the cluster structure in data.

  [Lindermann and Steinerberger'17, Arora et al. '18]

Auto-Encoders

Can we use Neural Networks to do (non-linear) dimension reduction?

Neural Networks



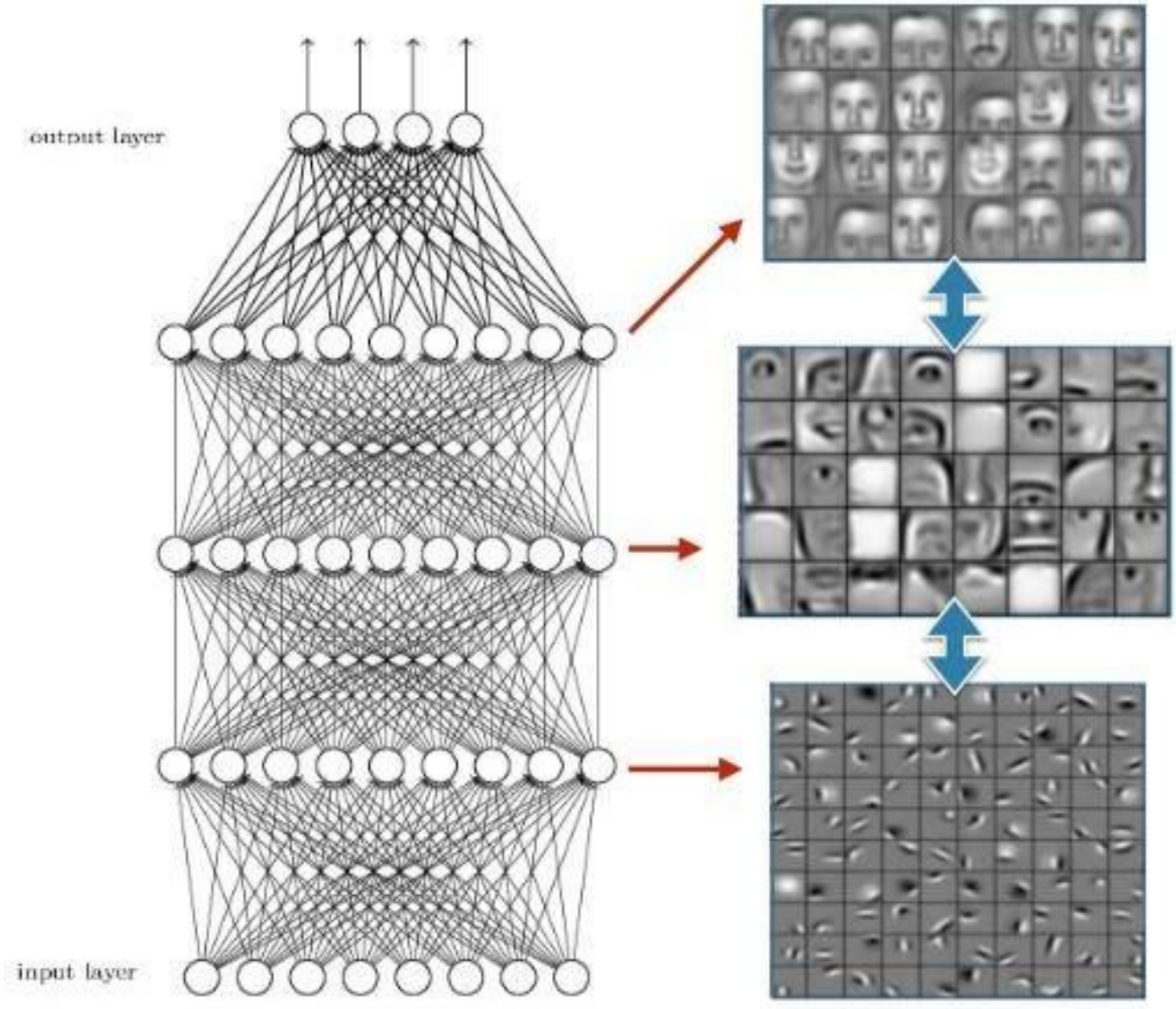
Weight learning is usually done by back-propagating the error signal (generated by the output label)

Original representation

Each hidden layer changes the data representation

Prediction via a simple softmax on the data representation according to the last hidden layer

Auto-Encoders



Hidden Layers

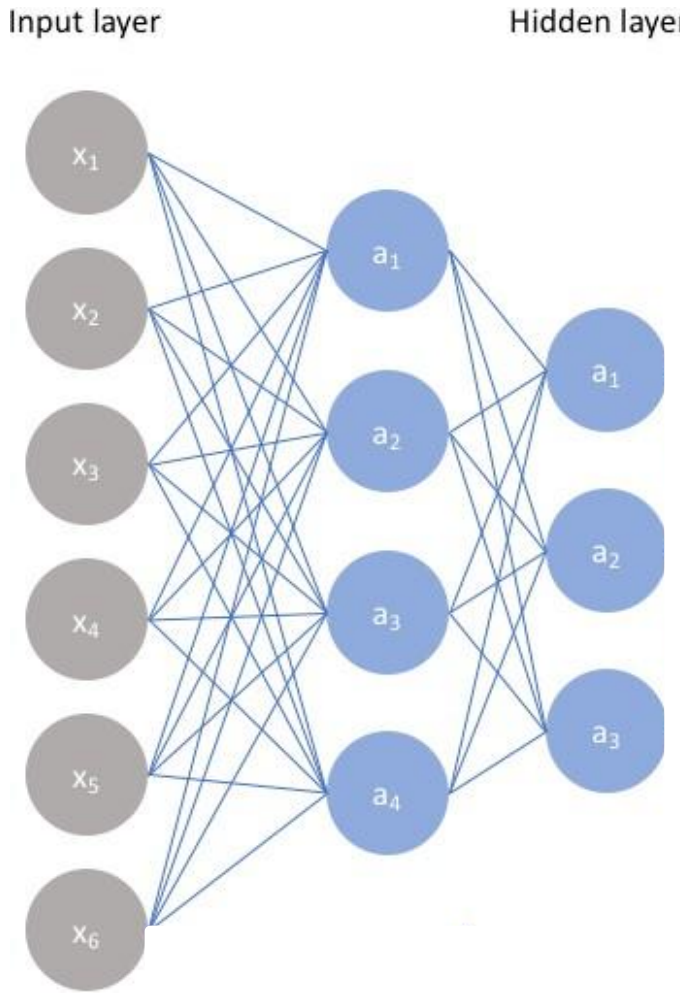
Final/deep hidden layers
(the representation
used for prediction)

Intermediate
hidden layers

Initial hidden layers

Auto-Encoders

Idea: To re-represent input data in lower dimensions, we can have the hidden layers of a NN with a narrow width



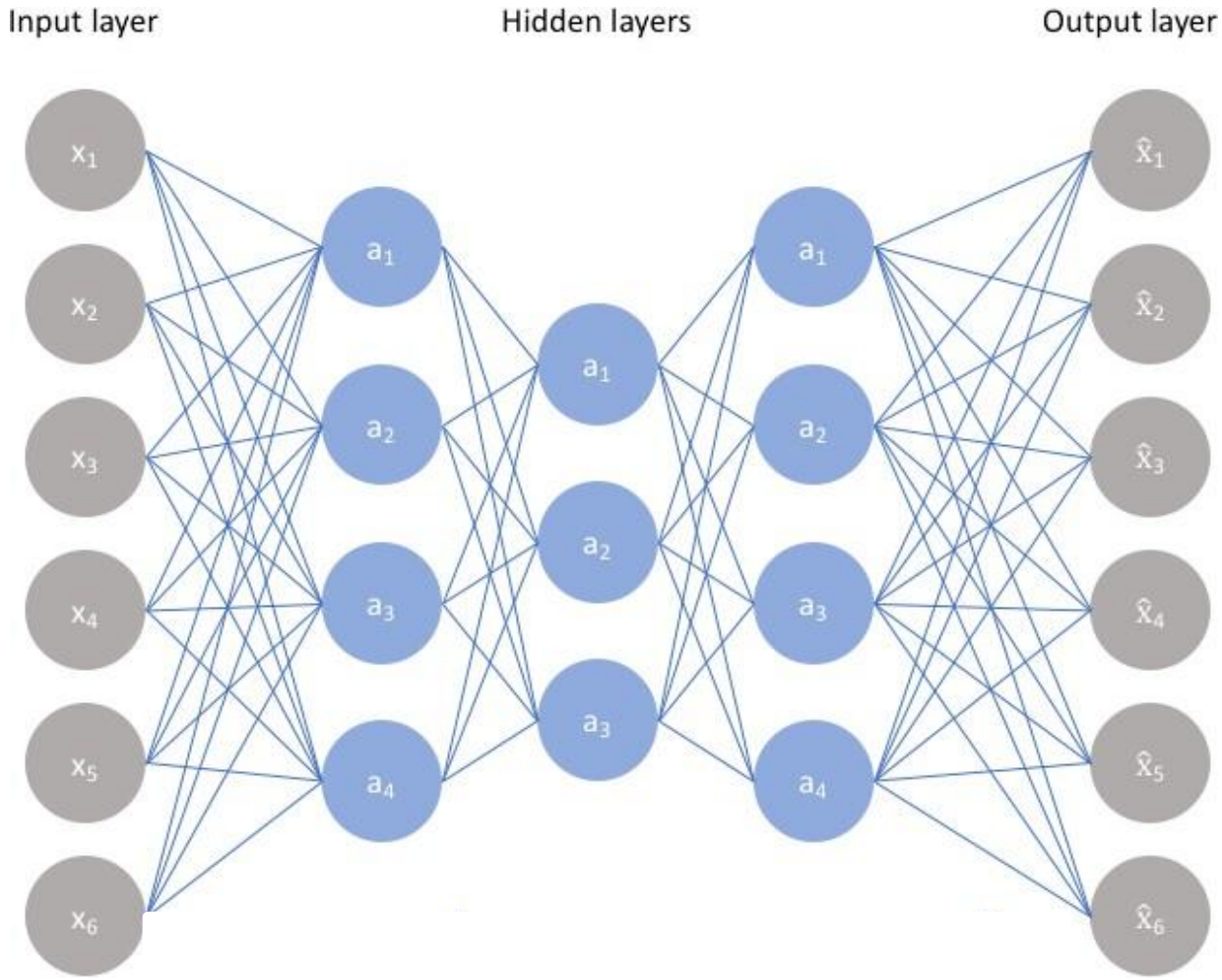
There are no labels, so…

- what should be the output?
- how can we learn the weights?

Idea:
Since the internal representation is supposed to represent the input, we should be able to retrieve/produce the input from the internal representation!

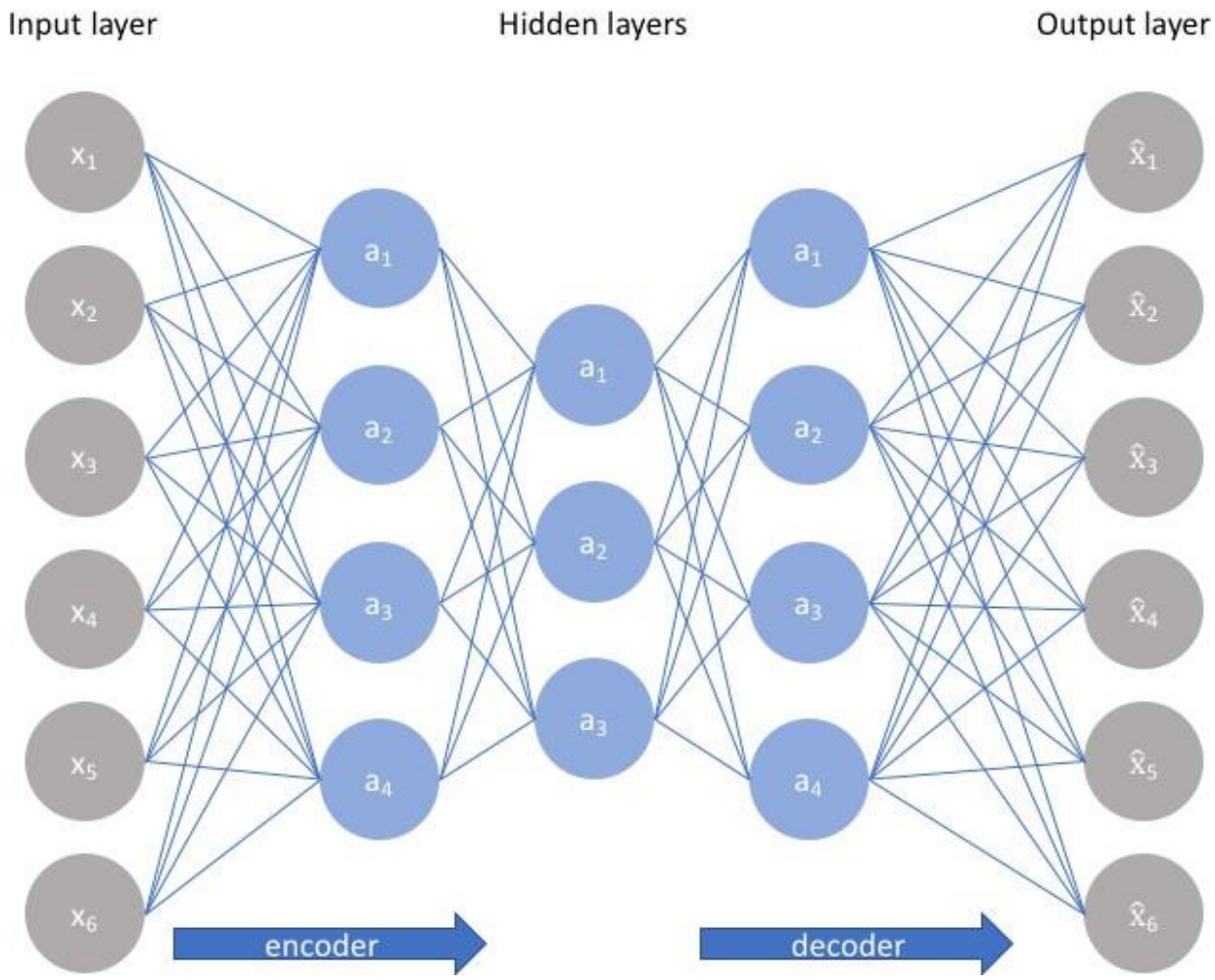
Auto-Encoders

Idea: To re-represent input data in lower dimensions, we can have the hidden layers of a NN with a narrow width



Auto-Encoders

The parameters are learned by backpropagating the error in input reconstruction



Auto-Encoders

Types of Auto-Encoders

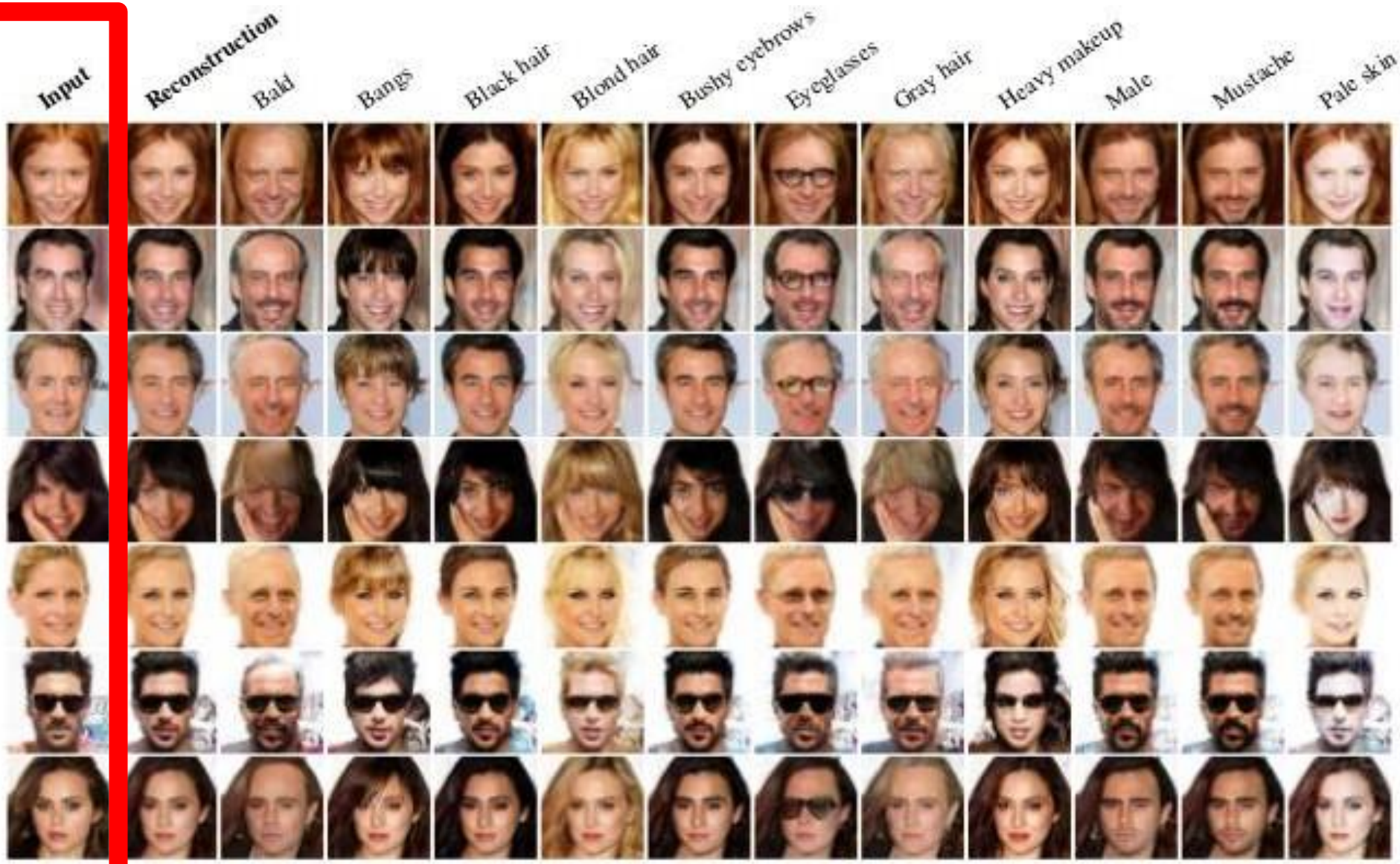
- The specific type we saw is called undercomplete auto-encoder
  (hidden layer is of lower dimension than the input layer)

- Overcomplete auto-encoder
  (hidden layer is of higher dimension; it must be regularized)
  can be used for sparse representations

- Denoising auto-encoders (DAE)
  input is deliberately made noisy and the output is required to be a clean representation

- Variational Auto-Encoders (VAE)
  A generative model which can help generate new example datapoints

VAE Potential



Questions

Before computing the covariance matrix S in PCA, you must:

A. Center the data by subtracting the mean from each point

B. Standardize each feature to unit variance

C. ℓ_2-normalize every vector to unit length

D. Remove outliers with k-means

Questions

Kernel PCA performs its eigendecomposition on which matrix?

A. The raw data matrix X(size $N \times D$)

B. The kernel matrix K(size $N \times N$)

C. The graph Laplacian of the data graph

D. The pairwise distance matrix D

Questions

When is classical MDS (multidimensional scaling) equivalent to PCA?

A. Always, regardless of distance

B. When the dissimilarities are Euclidean distances

C. Only when the number of points equals the dimensionality

D. When the kernel is linear

Questions

Which method explicitly preserves local neighborhoods by reconstructing each point from its neighbors in both the original and low-dimensional spaces?

A. Locally Linear Embedding (LLE)
B. Isomap
C. t-SNE
D. PCA

Questions

Compared with SNE, t-SNE differs primarily because it…

A. Uses cosine similarity and minimizes JS divergence

B. Uses a Gaussian in the low-dim space and asymmetric KL

C. Uses a Student-t distribution in the low-dim space and minimizes a symmetric KL divergence

D. Uses Earth-Mover distance in a randomized objective

Questions

K-means++ initialization chooses centers how?

A. Uniformly at random from $\mathbb{R}^D$

B. All from a small random subset of points

C. First center uniformly at random; each next center is picked from the data with probability proportional to $D\left(x\right)^2$)squared distance to nearest chosen center)

D. By running one iteration of standard K-means and taking the largest cluster's points

Questions

What is the decision boundary between any two clusters in basic K-means (with Euclidean distance)?

A. Circular arc

B. Quadratic curve

C. Linear (a hyperplane)

D. Nonparametric and arbitrary

Questions

Learning with Prototypes (LwP) using Euclidean distance is equivalent to which classifier?

A. k-NN with $k = 1$

B. Logistic regression with L_2regularization

C. A linear classifier with $w = 2(\mu_+ - \mu_-)$and $b = \| \mu_- \|^2 - \| \mu_+ \|^2$

D. Quadratic discriminant analysis

Questions

Which matrix appears (inverted) inside the Mahalanobis distance?

A. The identity matrix

B. The kernel matrix

C. The covariance matrix S

D. The graph Laplacian

Questions

Compared to LwP, which statement about k-NN at test time is most accurate?

A. k-NN typically stores all training data and can be slower because it computes distances to all points

B. k-NN stores only prototypes and is faster

C. k-NN trains a parametric model with few parameters

D. k-NN requires no distance computations at test time

Questions

For labeled data with overlapping classes, which projection is more suitable and why?

A. PCA, because it maximizes total variance

B. Fisher Linear Discriminant (LDA), because it pushes class means far apart while keeping within-class variance small

C. Kernel PCA, because it centers the kernel matrix

D. MDS, because it preserves all pairwise distances