Nonlinear Manifold Learning

Shihua Zhang

Fall 2019

Contents

Many Data are High-dimensional

This is an image of $n = 32 \times 32 \times 3$ dimensionality.

Many Data are High-dimensional

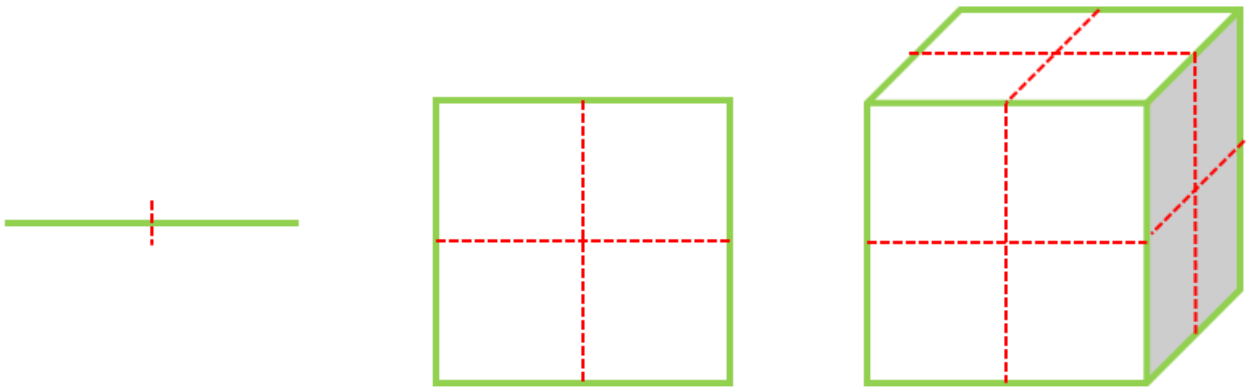
This is an image of $n = 32 \times 32 \times 3$ dimensionality.

We are cursed by high dimensionality!

- Insufficient Data
- Invalid Distance
- Redundant Information

What is the Curse of Dimensionality?

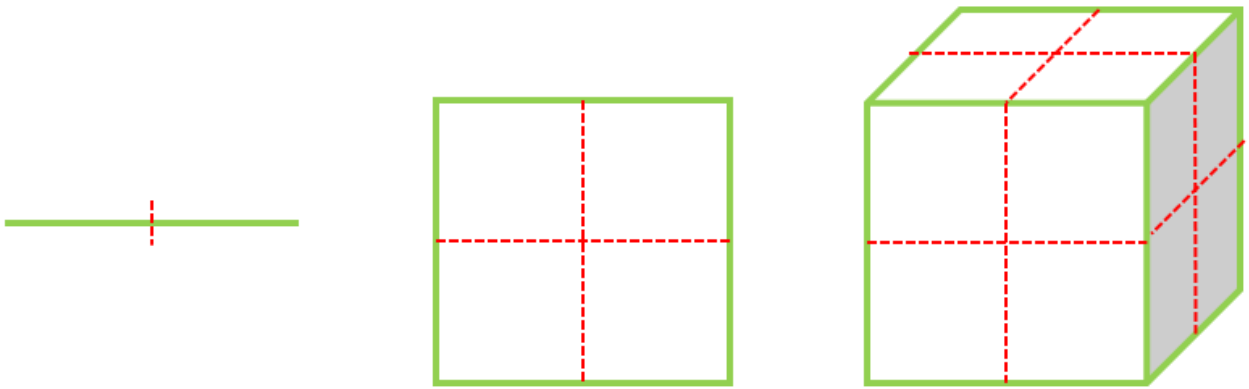
Problem: Given a cubic in $\mathbb{R}^n$, divide each dimension into 2 parts and put a data point in each sub-cubic. How many data points do we need?



2 when $n = 1$ 4 when $n = 2$ 8 when $n = 3$

What is the Curse of Dimensionality?

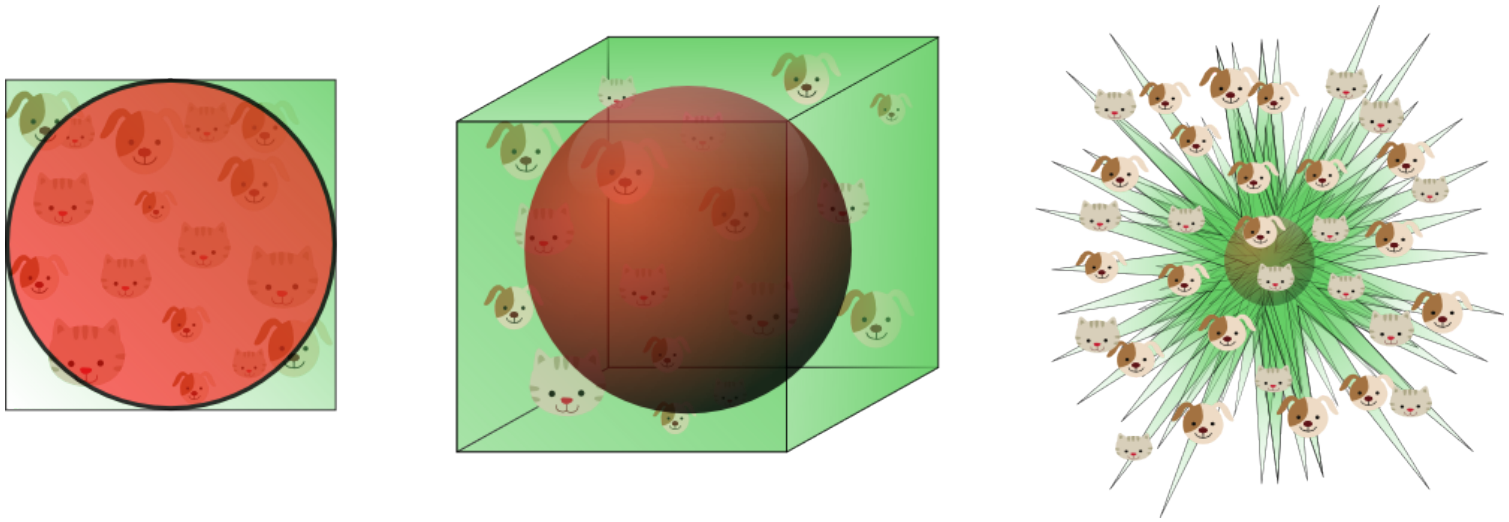
Problem: Given a cubic in $\mathbb{R}^n$, divide each dimension into 2 parts and put a data point in each sub-cubic. How many data points do we need?



2 when $n = 1$ 4 when $n = 2$ 8 when $n = 3$

What if $n = 32 \times 32 \times 3$?
$2^{3072} \simeq 10^{900}$ **data points are needed!!!**

What is the Curse of Dimensionality?



As the dimensionality increases, a larger percentage of the training data resides in the corners of the feature space [Spruyt, 2014].

How to Conquer This Problem?

Feature Selection

Reducing the dimensionality through variable selection.

How to Conquer This Problem?

Feature Selection

Reducing the dimensionality through variable selection.

Feature Extraction (Dimension Reduction)

Reducing the dimensionality by creating a reduced set of linear or nonlinear transformations of the input variables.

Dimension Reduction is Useful

- Data visualization: Insights into the low-dimensional structures in the data;

Dimension Reduction is Useful

- Data visualization: Insights into the low-dimensional structures in the data;
- Less storage requirements;

Dimension Reduction is Useful

- Data visualization: Insights into the low-dimensional structures in the data;
- Less storage requirements;
- Speeding up learning algorithms;

Dimension Reduction is Useful

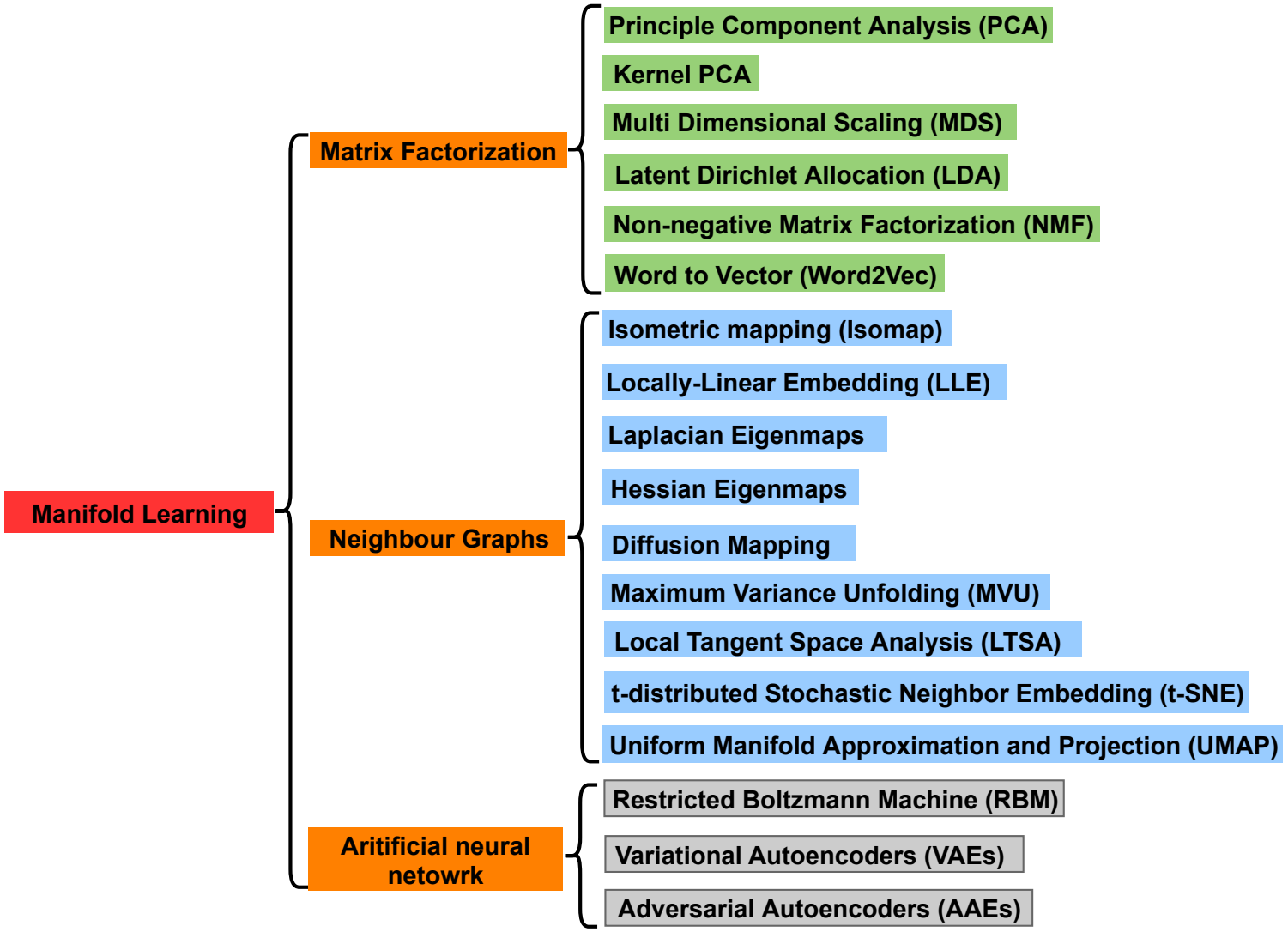
- Data visualization: Insights into the low-dimensional structures in the data;
- Less storage requirements;
- Speeding up learning algorithms;
- Data with fewer dimensions $\Rightarrow$ requires less training data$\Rightarrow$ Less chances of overfitting $\Rightarrow$ Better generalization

Dimension Reduction is Useful

- Data visualization: Insights into the low-dimensional structures in the data;
- Less storage requirements;
- Speeding up learning algorithms;
- Data with fewer dimensions $\Rightarrow$ requires less training data$\Rightarrow$ Less chances of overfitting $\Rightarrow$ Better generalization

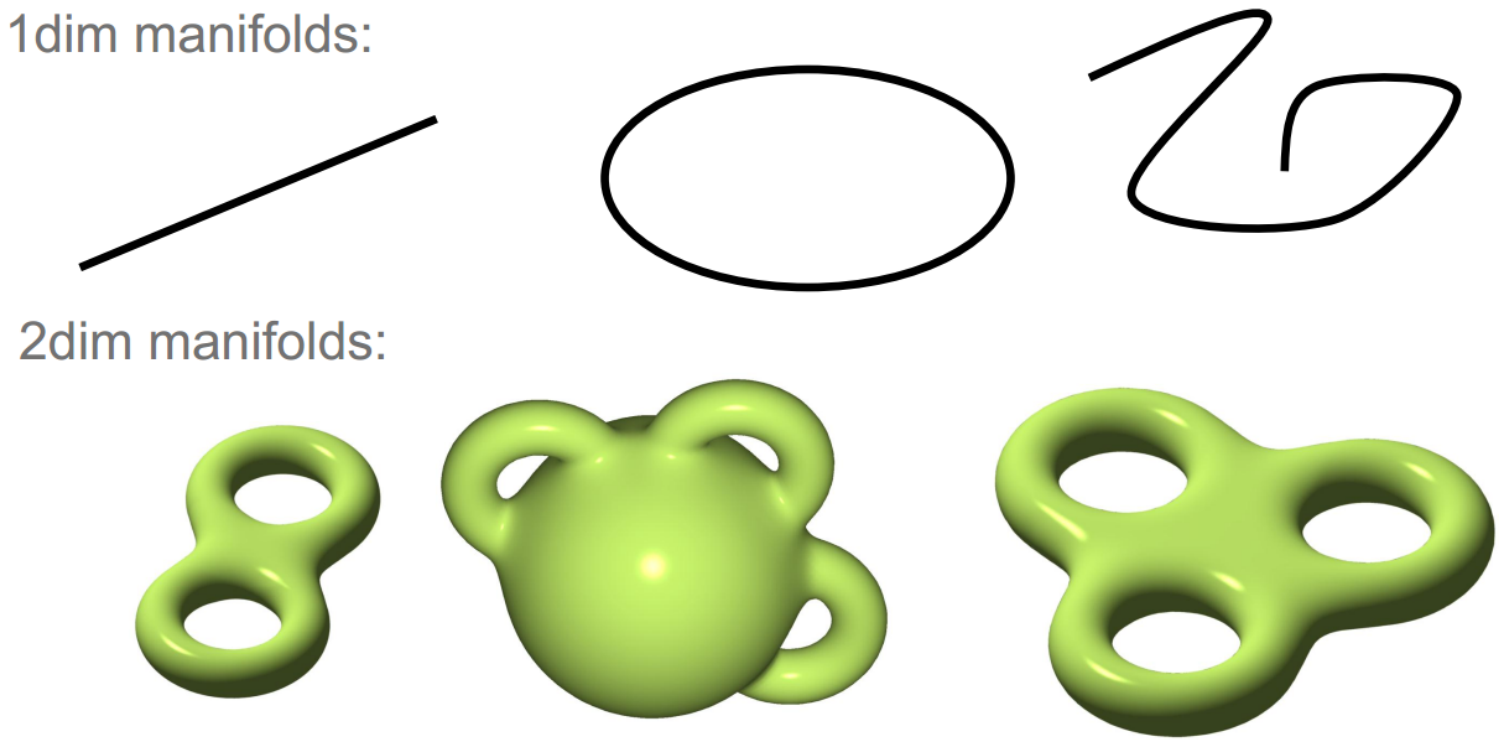
(Nonlinear) Dimension Reduction methods can be viewed as Manifold Learning.

Dimension Reduction



What is Manifold?

Manifold = any object which is nearly “flat” on small scales.



Basic Definitions

Topology

Let X be a set and let τ be a family of subsets of X. Then τ is called **a topology** on X if

- Both the empty set and X are elements of τ;
- Any union of elements of τ is an element of τ;
- Any intersection of finitely many elements of τ is an element of τ.

Basic Definitions

Topology

Let X be a set and let τ be a family of subsets of X. Then τ is called **a topology** on X if

- Both the empty set and X are elements of τ;
- Any union of elements of τ is an element of τ;
- Any intersection of finitely many elements of τ is an element of τ.

Topological Space

If τ is a topology on X, then the pair (X, τ) is called a topological space.

Basic Definitions

Manifold

A manifold is a topological space that resembles Euclidean space near each point $\Rightarrow$ each point of an n-dimensional manifold has a neighborhood that is homeomorphic to the Euclidean space of dimension n.

Homeomorphic

A function $f : X \rightarrow Y$ between two topological spaces is a homeomorphism if it has the following properties:

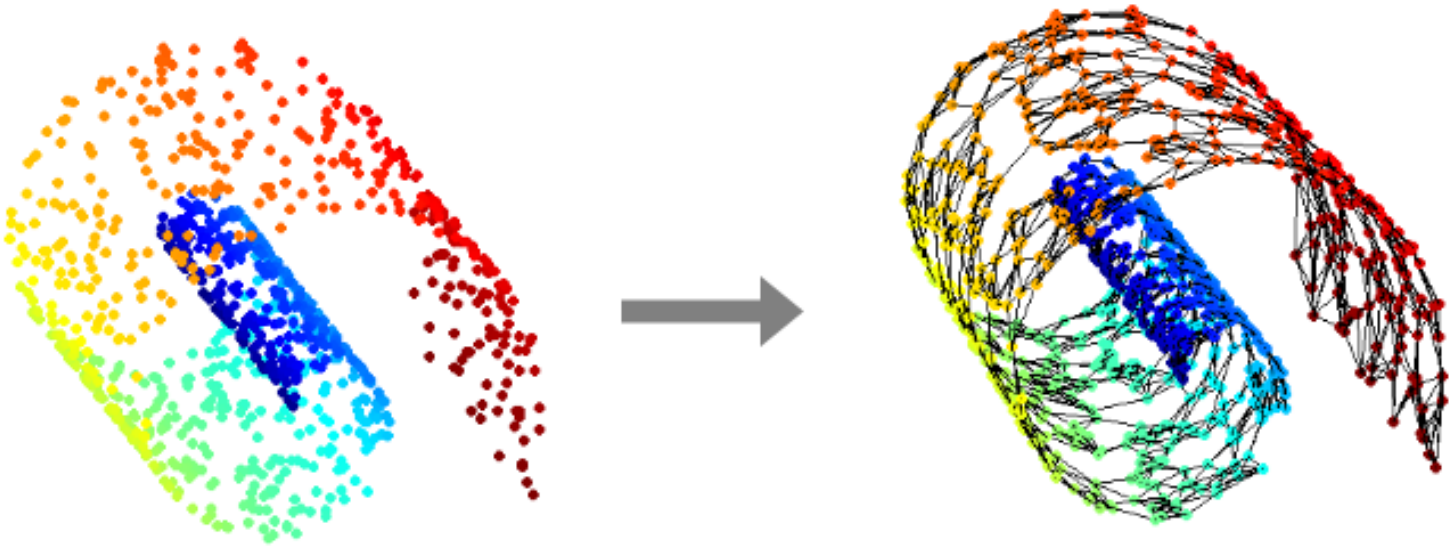
- f is a bijection (one-to-one correspondence);
- f is continuous;
- The inverse function f^{-1} is continuous. If such a function exists, X and Y are homeomorphic.

Geodesic Distance

Manifold can be continuous. But in real world, we only have concrete data points such as $X = \{x_1, x_2, \ldots, x_n\}$.

How to measure the distance in the 'manifold' with given data?

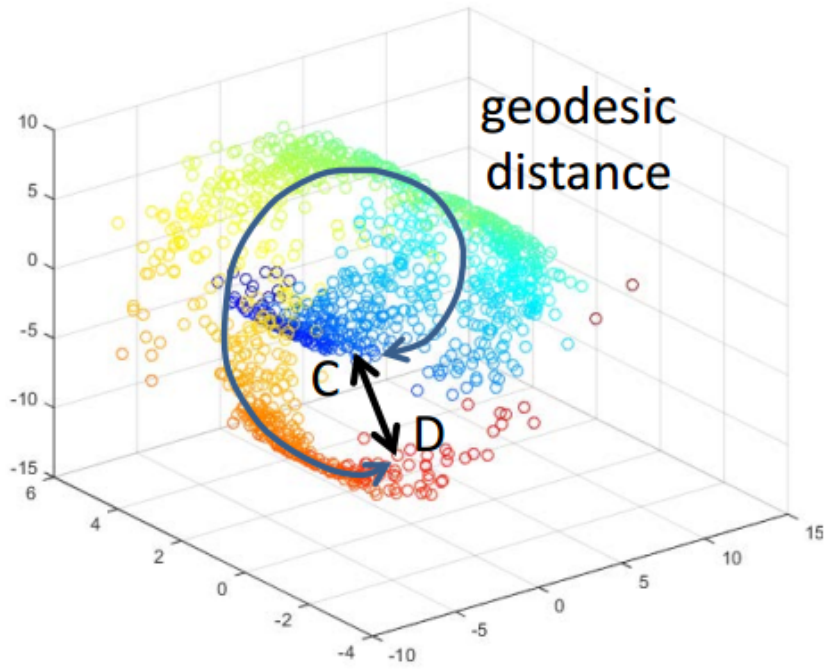
First, use data points to construct a neighbourhood graph.



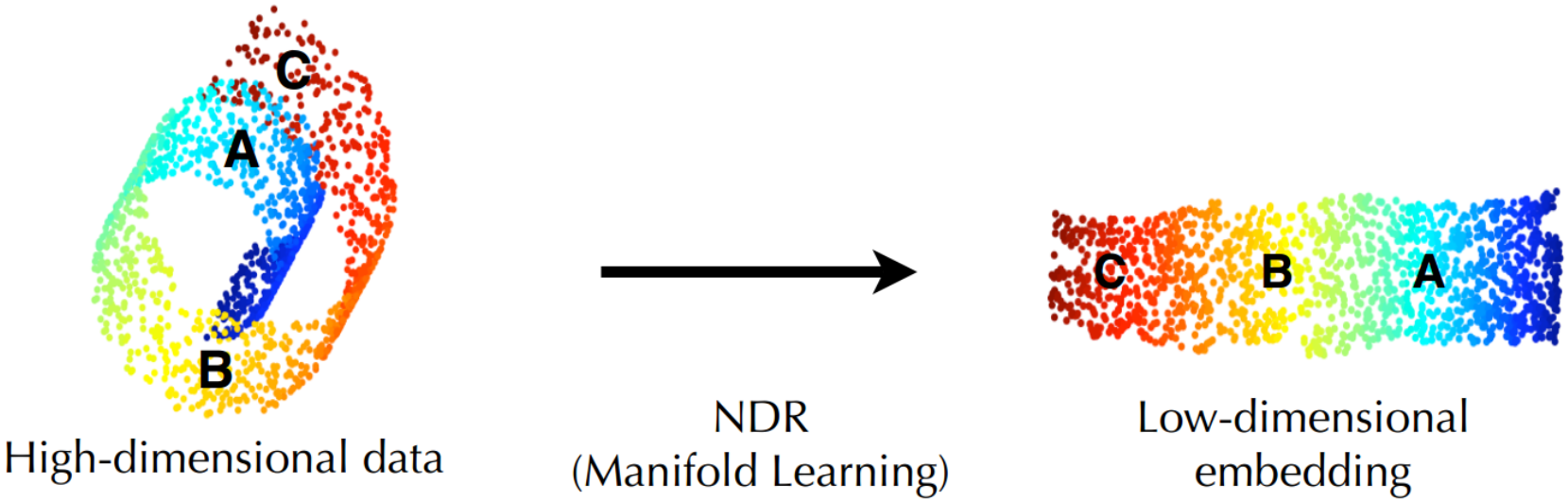
Geodesic Distance

Geodesic distance

The distance between two vertices in a graph is the number of edges in a shortest path (also called a graph geodesic) connecting them.



Manifold Learning



Manifold Learning (ML)

ML is to achieve (nonlinear) dimension reduction (NDR).

Problem Classification

Manifold learning or dimension reduction is not restricted to unsupervised learning.

1. Data with labels;
2. Data with different data types: boolean, integral, ordinal;
3. Time series data.

Problem Classification

Manifold learning or dimension reduction is not restricted to unsupervised learning.

1. Data with labels;
2. Data with different data types: boolean, integral, ordinal;
3. Time series data.

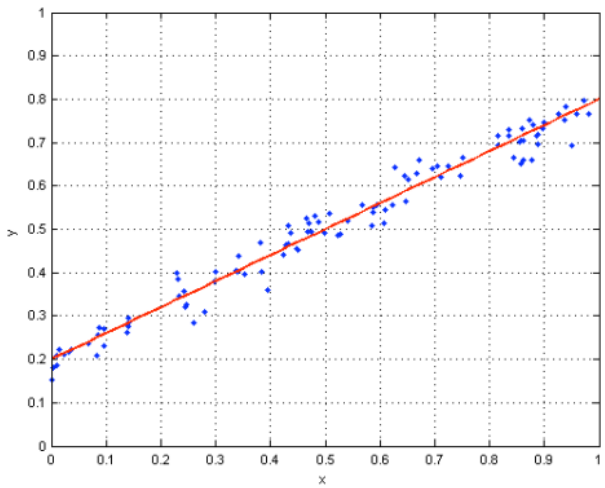
However,

1. Most unsupervised algorithms can be extended to supervised or semi-supervised ones;
2. Numerical algorithms can be generalized to variable abstract data types by modifying the metric or loss function;

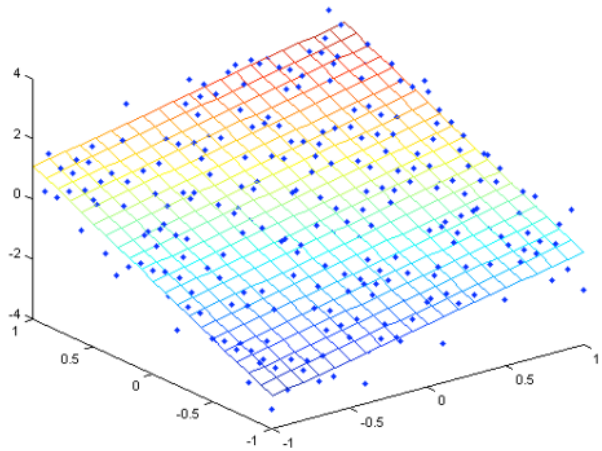
Let's focus on unsupervised algorithms and numerical data types.

Linear Manifold

Assume the high-dim data lies on/near a linear subspace.



$D_{high}=2$
$D_{low}=1$



$D_{high}=3$
$D_{low}=2$

Well-known and stable tools exist for determining the parameters of this subspace.

- Principal Components Analysis [Hotelling, 1933];
- Multidimensional Scaling [Kruskal and Wish, 1978].

Principle Component Analysis (PCA)

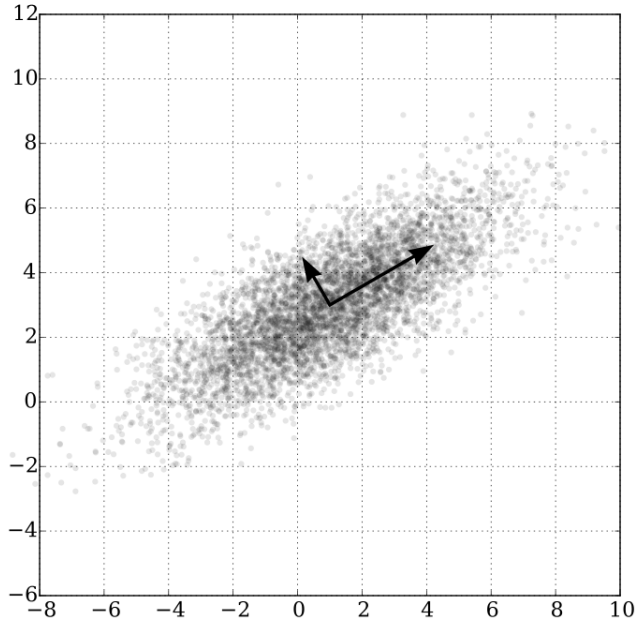
Inputs: n samples $x_1, x_2, \ldots, x_n \in \mathbb{R}^D$.
Outputs: n samples $y_1, y_2, \ldots, y_n \in \mathbb{R}^K$.

Goal

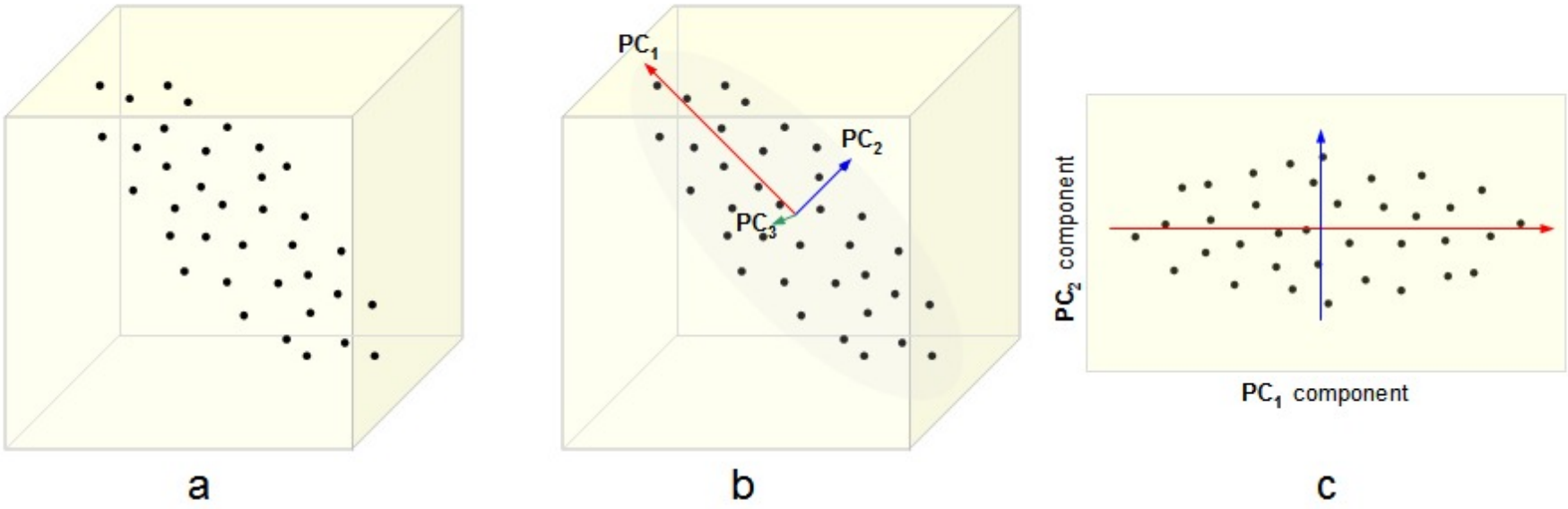
We want the new low dimensional representation $y_1, y_2, \ldots, y_n$ projected by W to express $x_1, x_2, \ldots, x_n$ as completely as possible.

$\Rightarrow$ Maximum separability (difference)

$\Rightarrow$ Maximum variance of $y_1, y_2, \ldots, y_n$



Principle Component Analysis (PCA)



Formulation of PCA

Assume $\sum_i x_i = 0$, and the new linear coordinate system is $\omega_1, \omega_2, \ldots, \omega_D$, $\|\omega_i\|_2 = 1$, $\omega_i^T \omega_j = 0$ $(i \neq j)$,then

$$y_i = W^T x_i,$$

The variance of x_i is

$$V_i = \sum_i y_i y_i^T = \sum_i W^T x_i x_i^T W,$$

To maximum the variance of all samples, we have

$$\max_W \operatorname{tr}(W^T X X^T W)$$

$$\text{s.t.} W^T W = I,$$

Solution of PCA

We can solve it by the method of **Lagrange Multipliers**:

$$L = \text{tr}(W^T X X^T W) - \lambda(W^T W - I)$$

$$\text{Let } \frac{\partial L}{\partial \omega_i} = X X^T \omega_i - \lambda_i \omega_i = 0, i = 1, 2, \ldots, D$$

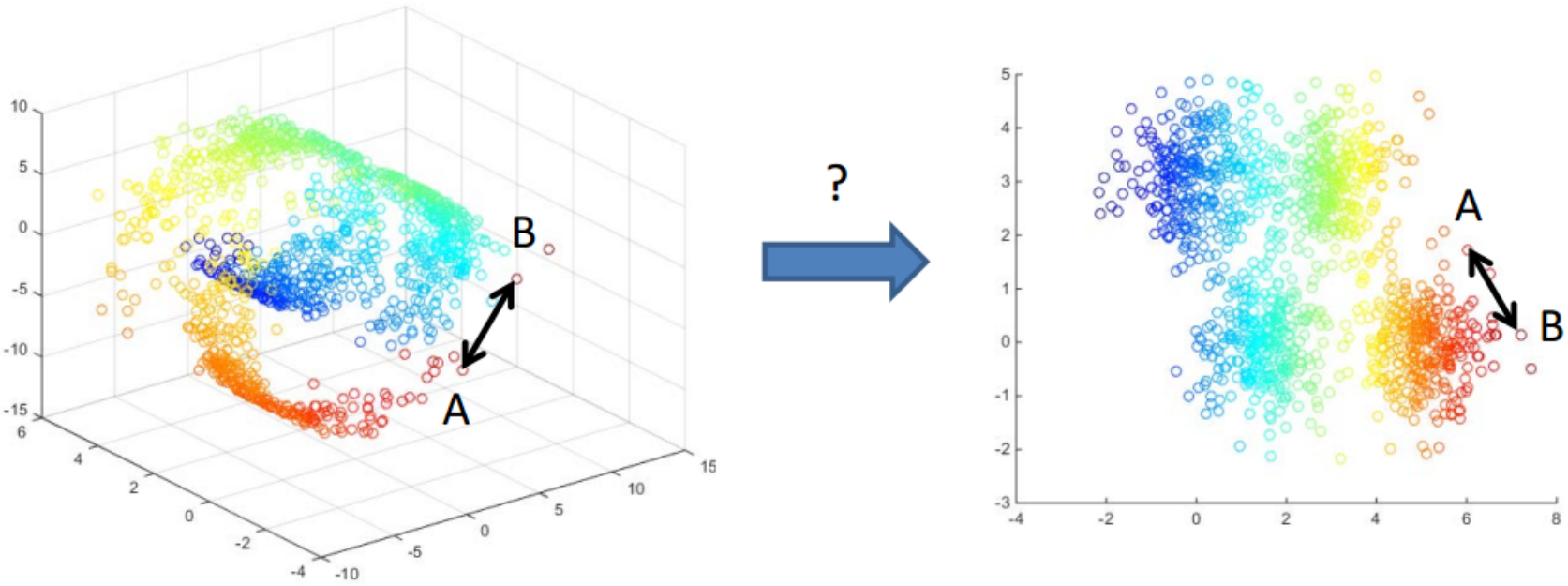
XX^T is the covariance matrix of samples, and after eigenvalue decomposition, we get

$$\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_D, \tag{1}$$

Take eigenvectors of the top K eigenvalue (principle component) as $W^* = (\omega_1, \omega_2, \ldots, \omega_K) \Rightarrow W^*$ projects $x_1, x_2, \ldots, x_n$ to their low dimensional representation $y_1, y_2, \ldots, y_n$.

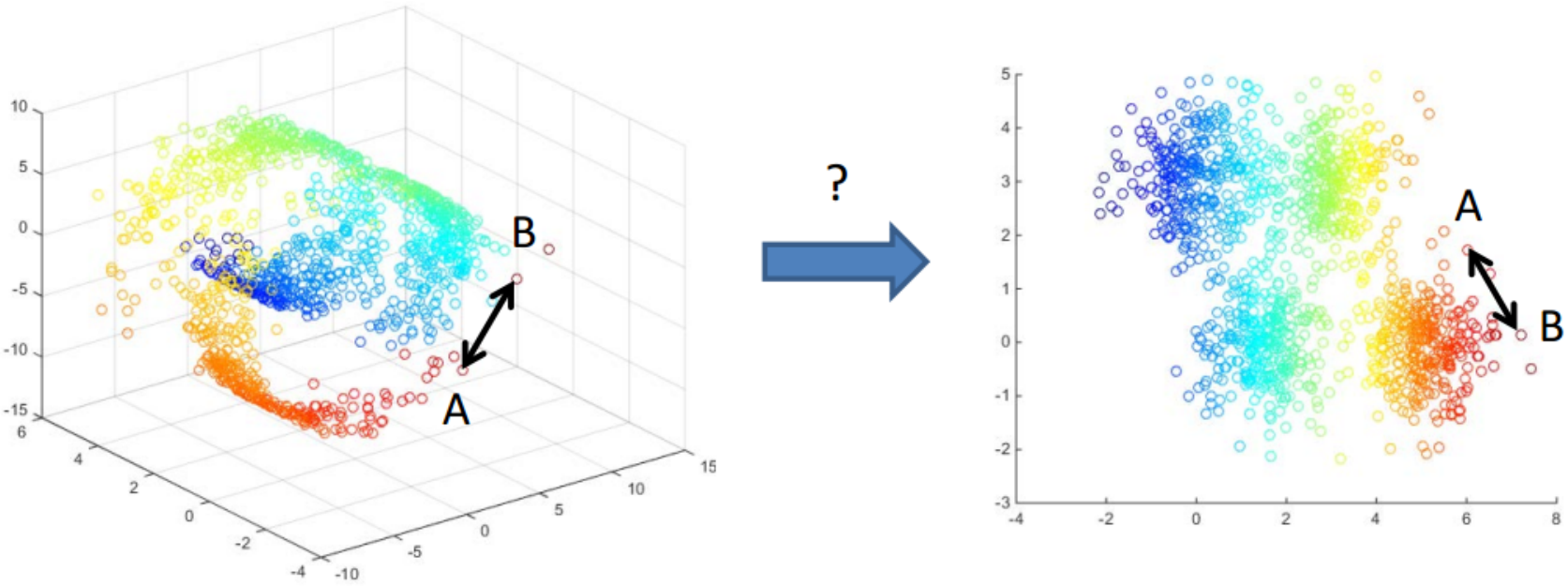
Multiple Dimensional Scaling (MDS)

Preserve the pairwise distances $\rightarrow$ Preserve the structure?



Multiple Dimensional Scaling (MDS)

Preserve the pairwise distances $\rightarrow$ Preserve the structure?



MDS: Given $n(n-1)/2$ pairwise distances $d_{ij} = \|x_i - x_j\|$, find a low dimensional embedding $x \rightarrow y$ such that $\|y_i - y_j\| \approx d_{ij}$.

Multiple Dimensional Scaling (MDS)

Given centered mean-zero data $X = (x_1, x_2, \ldots, x_n)$, $\sum_{i=1}^{n} x_i = 0$, $x_i \in \mathbb{R}^d$ $(i = 1, 2, \ldots, n)$, and assume $\sum_{i=1}^{n} y_i = 0$ $(i = 1, 2, \ldots, n$, then we can express the dot products $G_{ij} =< y_i, y_j >$ in terms of pairwise distances d_{ij}

$$G_{ij} = \frac{1}{2}\left[\frac{1}{n}\sum_{k}\left(d_{ik}^2 + d_{kj}^2\right) - d_{ij}^2 - \frac{1}{n^2}\sum_{kl} d_{kl}^2\right], \tag{2}$$

We then seek new vectors y_i, so as to minimize the error function

$$\text{err}(Y) = \sum_{ij}\left(G_{ij} - y_i^\top y_j\right)^2 = \|G - Y^\top Y\|_F^2. \tag{3}$$

Multiple Dimensional Scaling (MDS)

Matrix $G = (< x_i, x_j >)$ is known as a Gram matrix.
Using the eigen-decomposition of the Gram matrix

$$\begin{aligned} G &= \sum_{\alpha=1}^{n} \lambda_\alpha \vec{v}_\alpha \vec{v}_\alpha^{\mathrm{T}} \text{ with } \lambda_1 \geqslant \cdots \geqslant \lambda_n \geqslant 0 \\ &= V \Lambda V^T, \end{aligned}$$

Thus, the optimal approximation of G is given by an outer-product of the most significant eigenvectors. Then

$$y_{\alpha i} = \sqrt{\lambda_\alpha} v_{\alpha i} \quad \text{for } \alpha = 1, 2, \ldots, \mathrm{d}^*, \tag{4}$$

That is

$$Y = \Lambda_*^{1/2} V_*^T \in \mathbb{R}^{d^* \times n}, \tag{5}$$

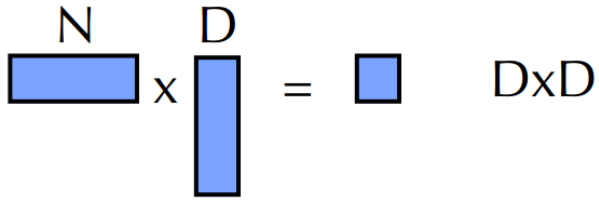
where $\Lambda_* = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{d^*})$.

PCA vs MDS

PCA and MDS are in some sense dual to each other

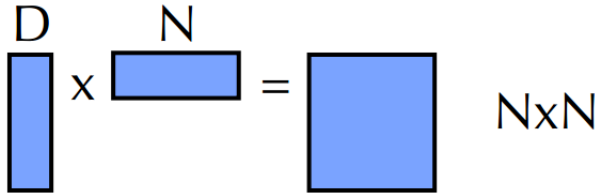
In PCA, we compute the $D \times D$ covariance matrix

$$C_{ij} = \frac{1}{n} \sum_k x_{ik} x_{jk}$$



In MDS, we compute the $N \times N$ Gram matrix

$$G_{ij} = \overrightarrow{x_i} \circ \overrightarrow{x_j}$$



For Euclidean distances d_{ij} in MDS, the two methods yield the same embedding results (up to an arbitrary rotation)

Isometric Feature Mapping (IsoMap)

What is the problem of MDS when the data are not on linear subspace?

Isometric Feature Mapping (IsoMap)

What is the problem of MDS when the data are not on linear subspace?

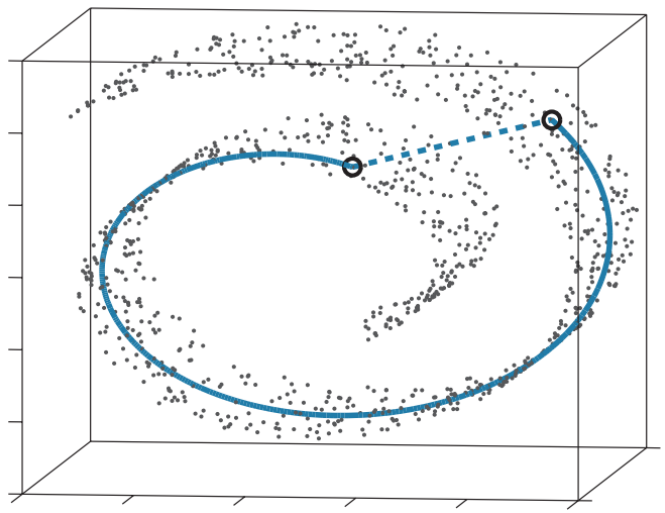
MDS seeks an embedding that preserves pairwise distances between data points.

Isometric Feature Mapping (IsoMap)

What is the problem of MDS when the data are not on linear subspace?

MDS seeks an embedding that preserves pairwise distances between data points.

Geodesic distances measured on the manifold may be longer than the corresponding Euclidean straight line distance d_{ij}



Isometric Feature Mapping (IsoMap)

Idea : Use geodesic rather than Euclidean distances in MDS

Isometric Feature Mapping (IsoMap)

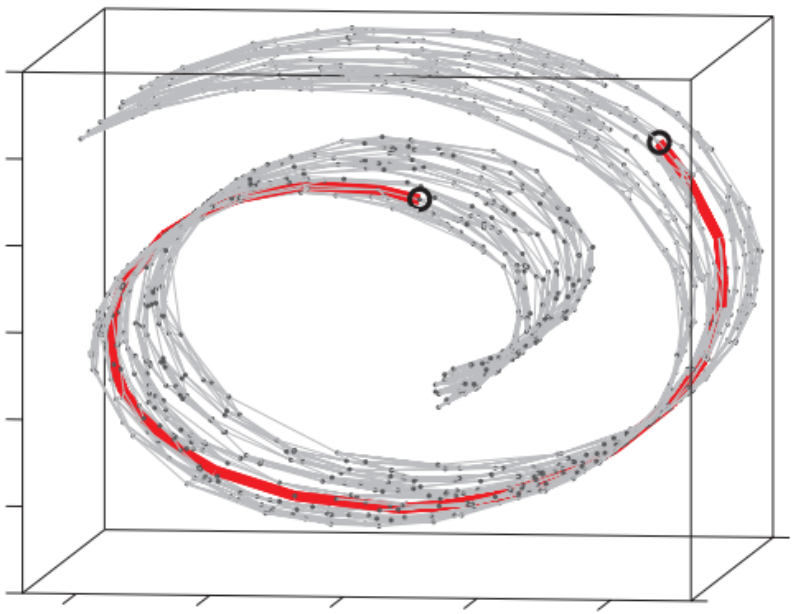
Idea : Use geodesic rather than Euclidean distances in MDS

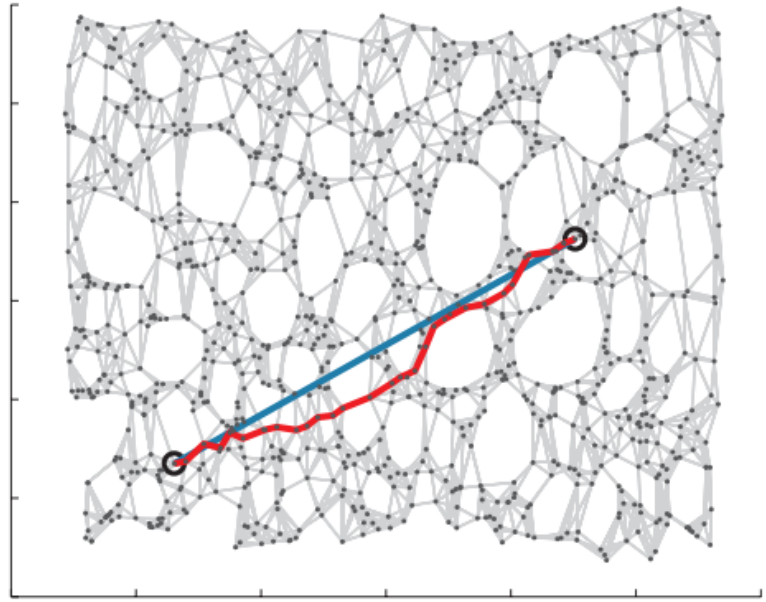
But - How can we compute geodesics without knowing the manifold?

Isometric Feature Mapping (IsoMap)

Idea : Use geodesic rather than Euclidean distances in MDS

But - How can we compute geodesics without knowing the manifold?

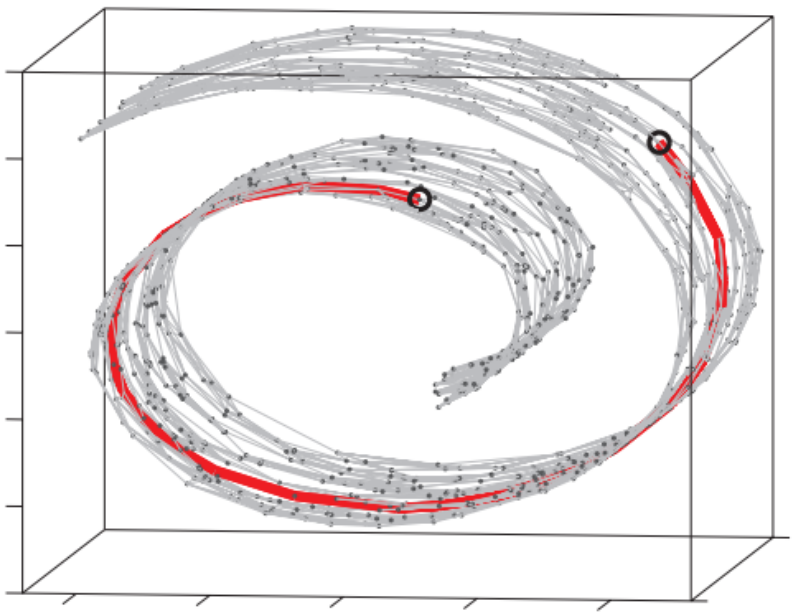


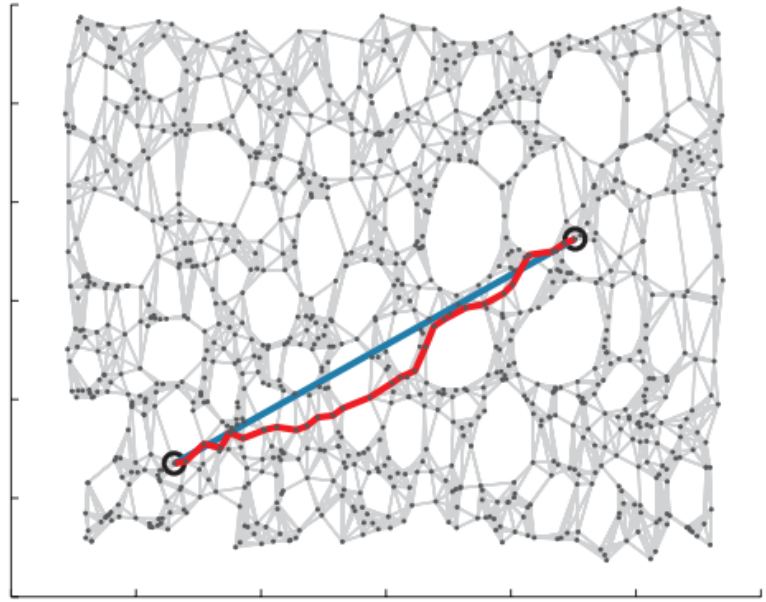


Isometric Feature Mapping (IsoMap)

Idea : Use geodesic rather than Euclidean distances in MDS

But - How can we compute geodesics without knowing the manifold?





Build an **adjacency graph** and approximate geodesic distances by **shortest-paths through the graph**!

Isometric Feature Mapping (IsoMap)

Step 1: Build the adjacency graph over high-dim points X

Neighborhood selection

- K-nearest neighbors;
- neighbors within a fixed radius (epsilon-ball).

Assume graph is fully connected

no isolated islands of points

Assume graph neighborhoods reflect manifold neighborhoods

- no short-cuts between distant points on manifold;
- sensitive to choice of neighborhood size.

Isometric Feature Mapping (IsoMap)

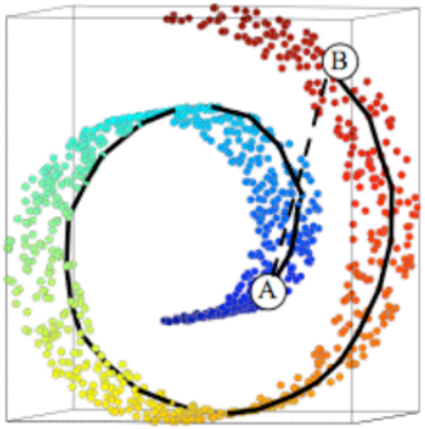
Step 2: Compute approximate geodesics

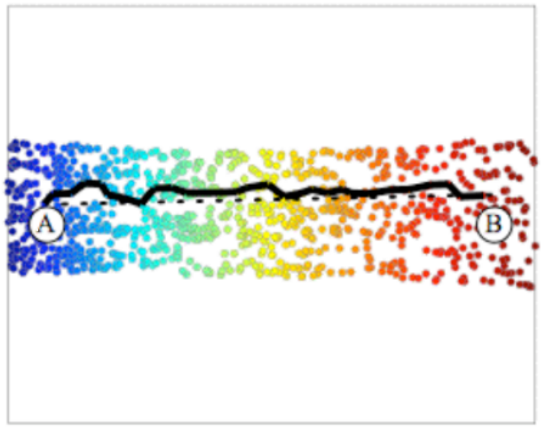
- Weight graph edges by inter-point distances;
- Apply Dijkstras all-pairs shortest-paths algorithm $O(N^2 lgN + N^2 k)$.

Isometric Feature Mapping (IsoMap)

Step 3: Apply MDS to geodesic distances

- Top **d** eigenvectors of Gram matrix give the embedded d-dimensional points;
- Dimensionality of manifold may be estimated by the number of significant eigenvalues (just as in PCA/MDS).





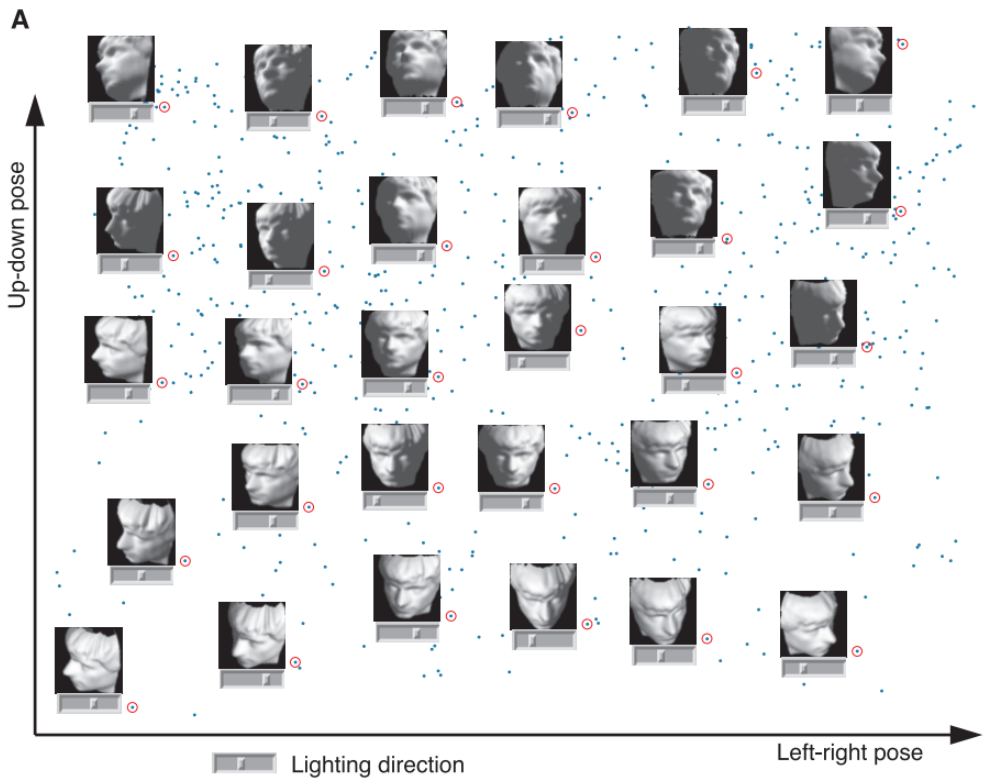
N = 1024 points
k = 12 nearest neighbours

IsoMap - Example

Faces - varying pose and illumination

3 true degrees of freedom (dof) in total

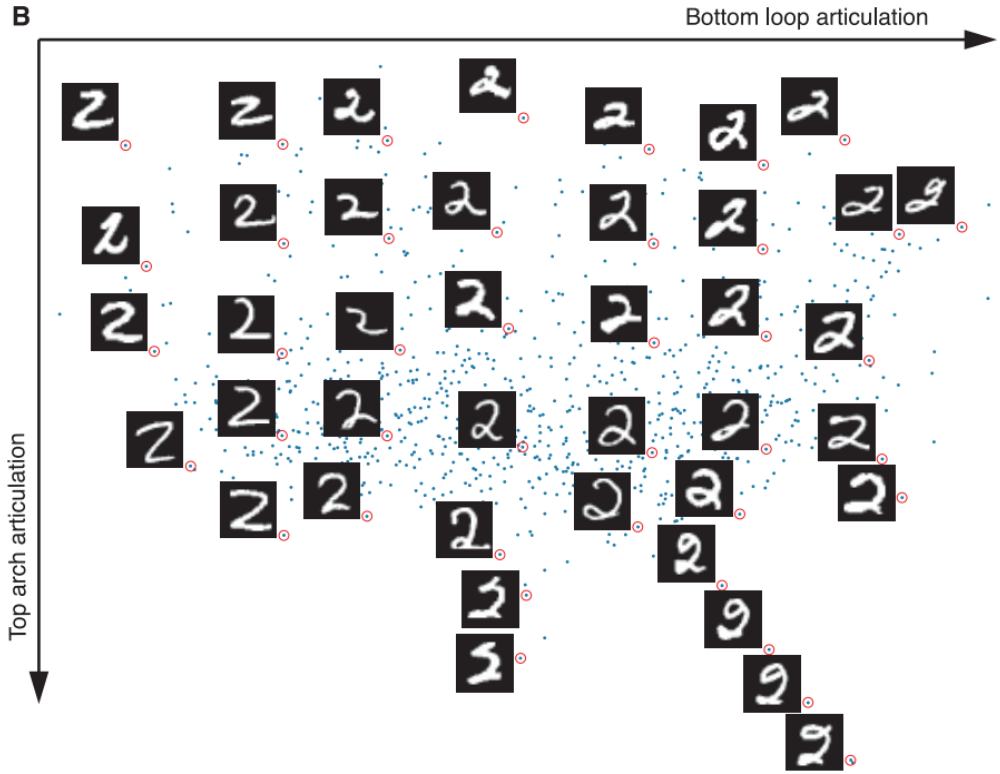
- 64×64 pixel images;
- $N = 698$;
- $k = 6$.



IsoMap - Example

MNIST

- 28×28 pixel images;
- N = 1000 ;
- k = 6.



IsoMap: Pros and Cons

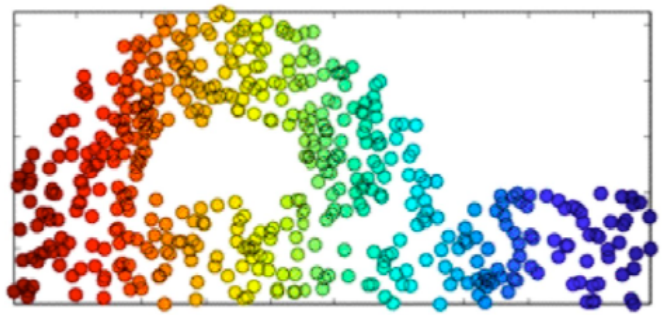
Pros

- **Strengths inherited from MDS**
  - Polynomial time algorithm
  - No local optima
  - Non-iterative
  - Automatic intrinsic dimensionality estimate
- **Isomap adds a single heuristic parameter**
  - graph neighbourhood size k
- **Guaranteed asymptotic convergence**
  - For data living on a convex submanifold of Euclidean space, and given large enough sample n, Isomap is guaranteed to recover the true manifold, up to a rotation and translation

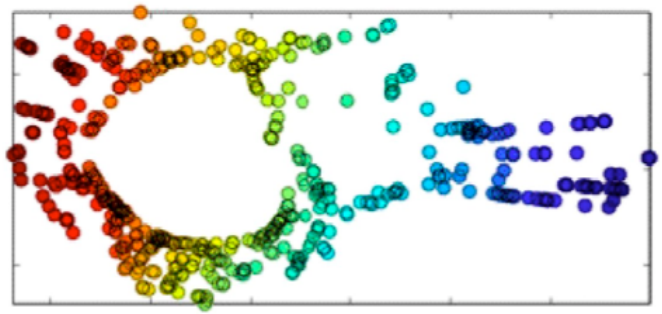
IsoMap: Pros and Cons

Cons

- **Sensitive to "short-cuts" due to k being too large**
- **Does not scale well to very large n**
  - $n \times n$ dense eigenvector problem is expensive
- **Convexity assumption**
  - Cannot handle manifolds with "holes"



Input



IsoMap embedding

Extensions of IsoMap

- Accelerate IsoMap
  - Robust Kernel IsoMap [Choi and Choi, 2007];
  - Parallel Transport Unfolding [Budninskiy et al., 2018].

- Improve Robustness
  - LandMark ISOMAP [Silva and Tenenbaum, 2003];
  - C Isomap [Silva and Tenenbaum, 2003];

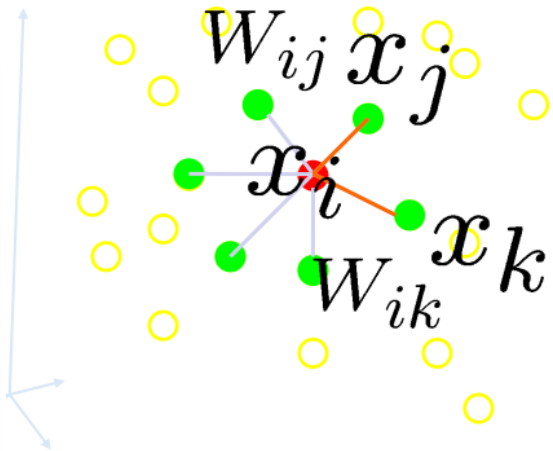
Locally-Linear Embedding (LLE)

Think locally, fit globally!

LLE aims to preserve local manifold geometry in its embedding [Roweis and Saul, 2000]

Main Idea

- Assume manifold is locally linear: we expect each d-dim data point to lie on or near a **locally linear** patch of the manifold;
- Characterize each point x_i as a **convex linear combination** of its k-nearest neighbors x_j;
- Seek an embedding that preserves these weights.



Locally-Linear Embedding (LLE)

Step 1: Compute k-nearest neighbors for each point x_i
Same as in Isomap

Step 2: Compute weights W_{ij} that best reconstruct x_i as a convex sum of its neighbors x_j

$$\arg\min_{W} \Phi(W) = \sum_i \left\| \vec{x}_i - \sum_{j \in \mathcal{N}_i} W_{ij} \vec{x}_j \right\|^2$$
$$\text{subject to } \sum_j W_{ij} = 1$$

- This is easily solved using a Lagrange multiplier;
- Note: local weights are invariant to translation, rotation and scale;
- Weights should be preserved under a well-behaved embedding.

Locally-Linear Embedding (LLE)

Step 3: Choose embedded coordinates y_i that minimize reconstruction error using previously computed weights W_{ij}

$$\arg\min_{\vec{y}} \Theta(\vec{y}) = \sum_i \left\| \vec{y}_i - \sum_{j \in \mathcal{N}_i} W_{ij} \vec{y}_j \right\|^2$$

$$\text{subject to } \sum_i y_i = 0 \quad \text{(zero mean)} \tag{6}$$

$$\frac{1}{n} \sum_i y_i y_i^\top = I_d \quad \text{(unit covariance)}$$

Locally-Linear Embedding (LLE)

Let $Y = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^{K \times n}$

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

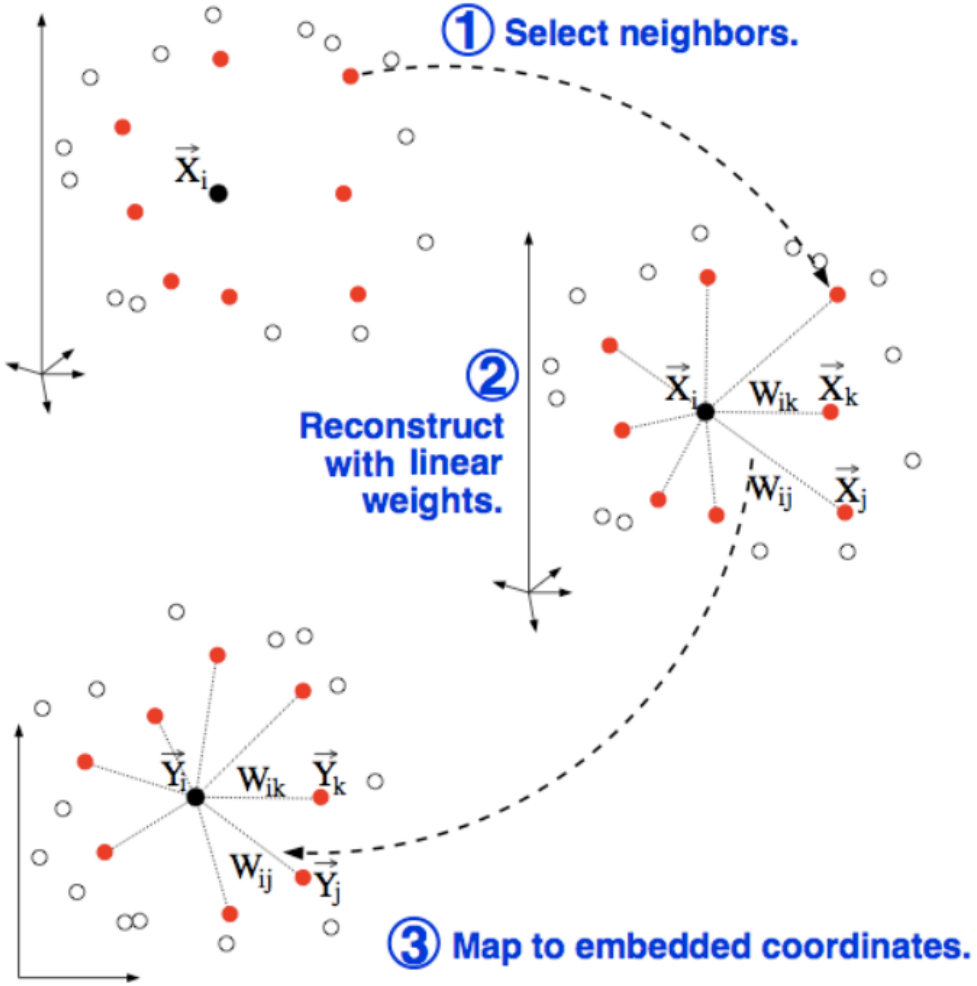
then Eq 6 can be written to

$$\begin{aligned} &\arg\min_{Y} \operatorname{tr}(YMY^T) \\ &\text{subject to } YY^T = I. \end{aligned}$$

The d eigenvectors of the **smallest** d eigenvalues give the embedding Y^T.

Locally-Linear Embedding (LLE)

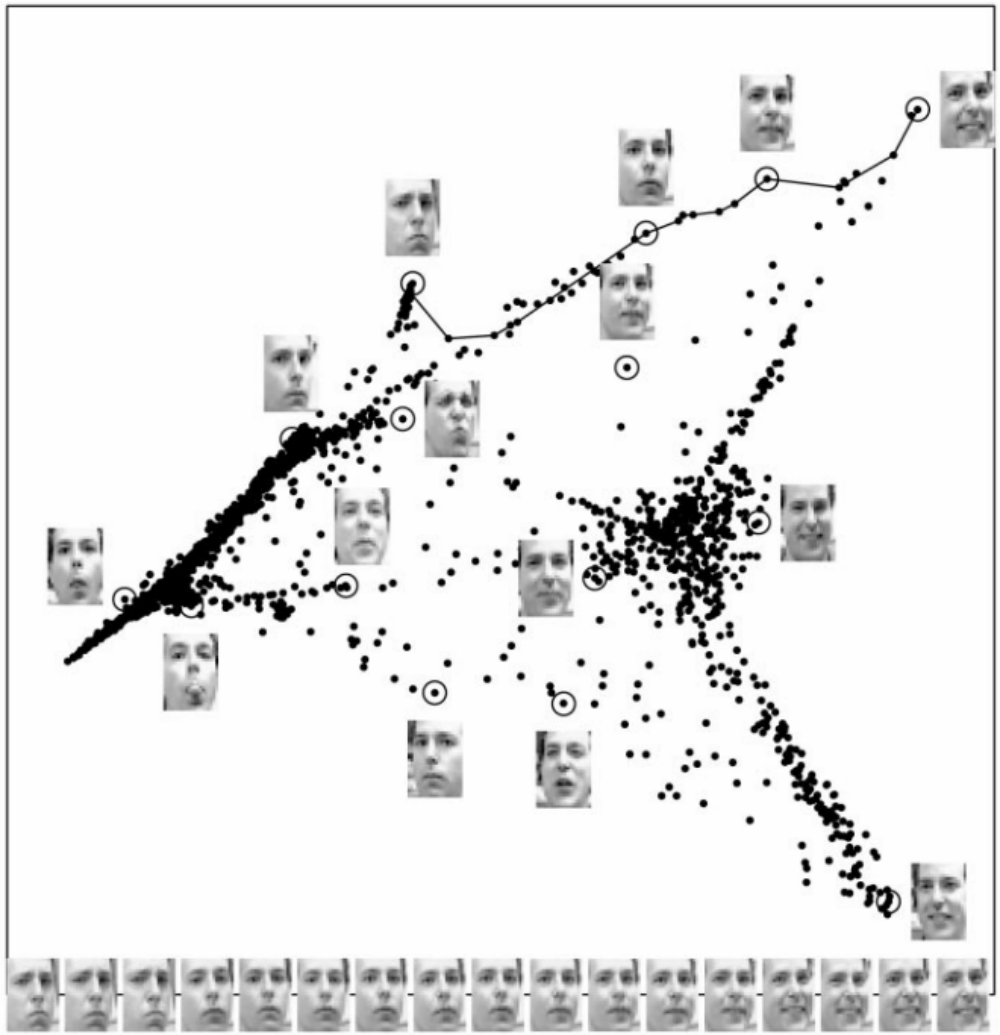
1. **Compute the neighbors of each data point, $\vec{X}_i$.**
2. **Compute the weights W_{ij} that best reconstruct each data point $\vec{X}_i$ from its neighbors, minimizing the cost in eq. (1) by constrained linear fits.**
3. **Compute the vectors $\vec{Y}_i$ best reconstructed by the weights W_{ij}, minimizing the quadratic form in eq. (2) by its bottom nonzero eigenvectors.**



LLE - Example

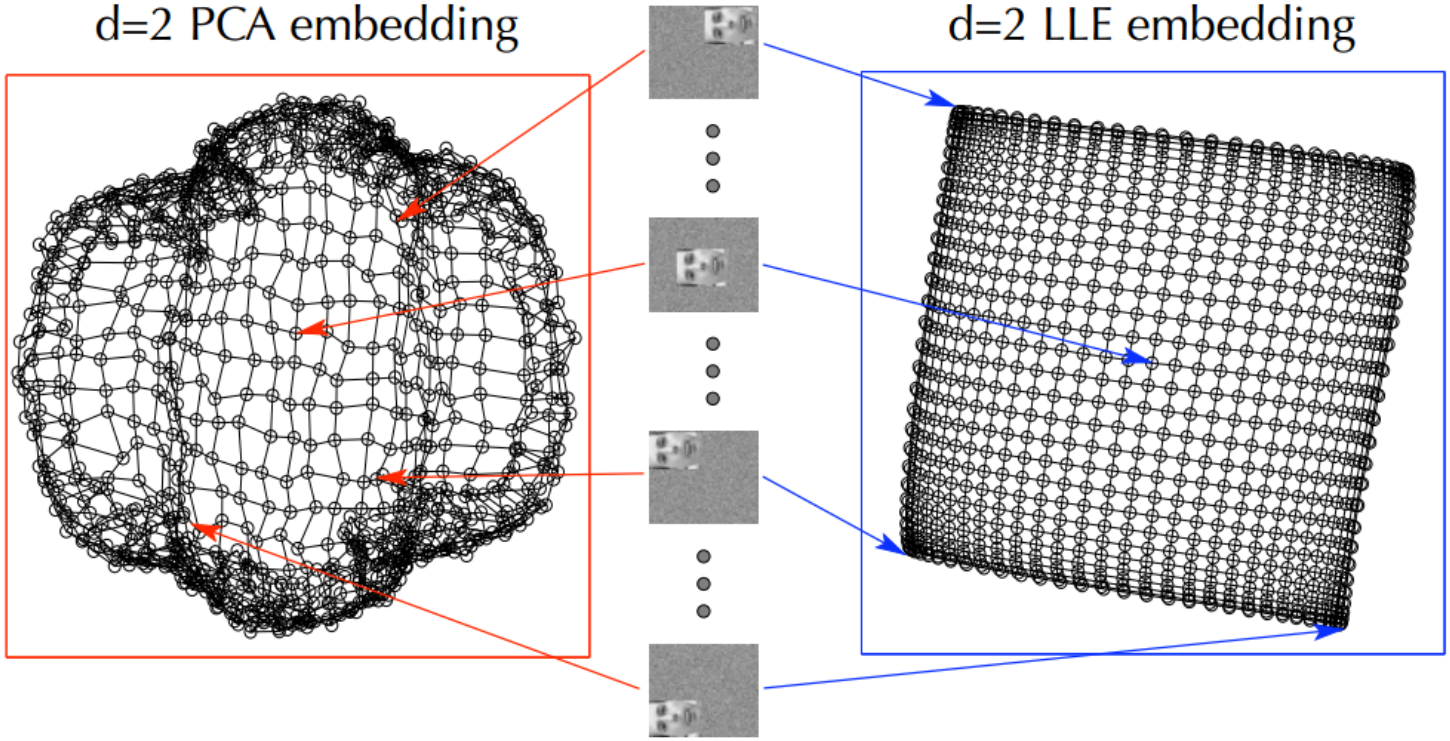
Face variations

- 20x28 pixel images;
- N=1965;
- k=12;
- d=2.

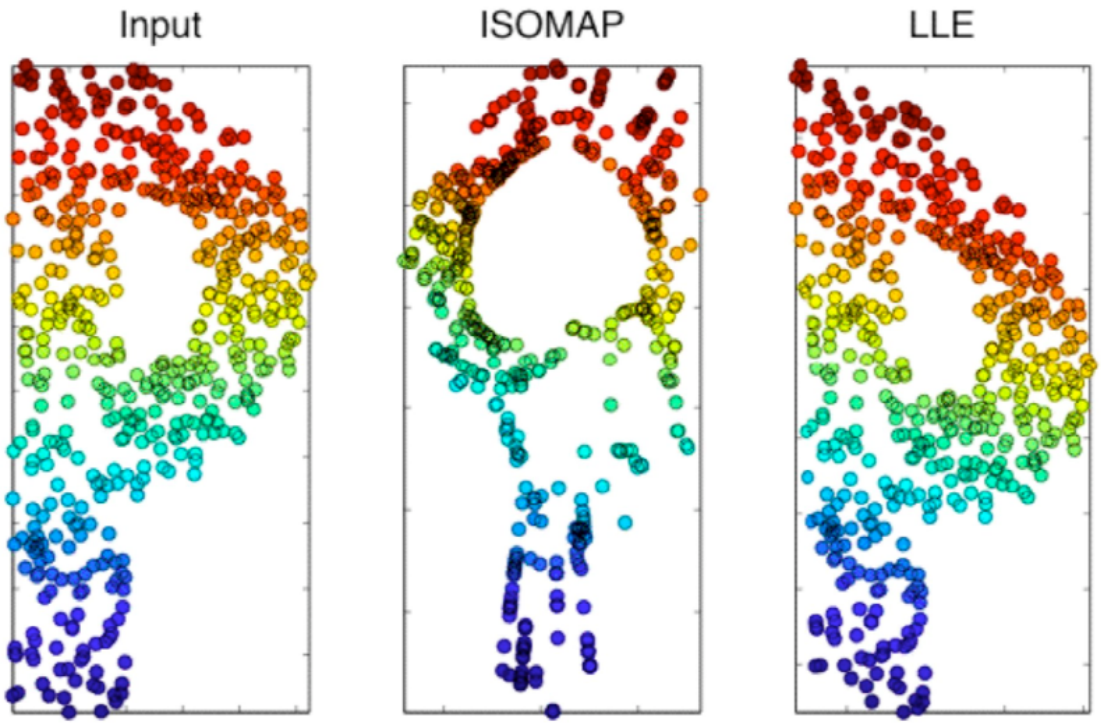


PCA vs LLE

- Input: 30x30 images of a translating face (N=961);
- PCA fails to recover a meaningful 2-d embedding;
- LLE discovers the 2 translational degrees of freedom in the input.



IsoMap vs LLE



- LLE handles non-convex manifolds (those with holes) a little better than IsoMap;
- Not perfect - we'd prefer this particular 2d-embedding to be a simple isometry.

LLE - Pros and Cons

Pros

- Graph-base, eigenvector method;
- Polynomial time algorithm;
- No local optima;
- Non-iterative;
- Single heuristic parameter (neighbourhood size k);
- Better handling of non-convex manifolds than IsoMap.

LLE - Pros and Cons

Cons

- Also sensitive to short-cuts (noise);
- Quadratic complexity on the training set size;
- Unlike IsoMap, no robust method to compute the intrinsic dimensionality;
- No robust method to define the neighborhood size k;
- No asymptotic guarantees.

IsoMap vs LLE

IsoMap

- Computes top d eigenvectors of a dense $n \times n$ matrix;
- Preserves distances;
- Asymptotic guarantee of finding true manifold.

LLE

- Computes bottom $d + 1$ eigenvectors of a sparse $n \times n$ matrix;
- Preserves local linear geometry;
- Copes with holes rather better.

Follow-up Methods

IsoMap has three steps:

1. Nearest-neighbor search;
2. Compute graph distances;
3. Spectral embedding via multidimensional scaling.

Follow-up Methods

IsoMap has three steps:

1. Nearest-neighbor search;
2. Compute graph distances;
3. Spectral embedding via multidimensional scaling.

LLE has three steps:

1. Nearest-neighbor search;
2. Constrained least-squares fits;
3. Spectral embedding.

Many follow-up algorithms only change one among the three steps.

Laplacian Eigenmaps

The first and third steps of the Laplacian eigenmap algorithm are very similar to those of LLE [Belkin and Niyogi, 2002].

Step 1 Nearest-neighbor search

Define a K-neighborhood N_i^K or an ϵ-neighborhood N_i^ϵ of the point x_i, In general, let N_i denote the neighborhood of x_i.

Step 2 Weighted adjacency matrix

Let $W = (w_{ij})$ be a symmetric $(n \times n)$ weighted adjacency matrix defined as follows:

$$w_{ij} = w(x_i, x_j) = \begin{cases} \exp\{-\frac{||x_i - x_j||^2}{2\sigma^2}\}, & \text{if } x_j \in N_i; \\ 0, & \text{otherwise} \end{cases}$$

Laplacian Eigenmaps

Step 3 Spectral embedding

Let $\mathbf{D} = (d_{ij})$ be a diagonal matrix with $d_{ii} = \sum_{j \in N_i} w_{ij} = (\mathbf{W1}_n)_i$ $(i = 1, 2, \ldots, n)$. The symmetric matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the graph Laplacian for the graph.
The $(t \times n)$-matrix $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ is determined by minimizing,

$$\sum_i \sum_j w_{ij} ||\mathbf{y}_i - \mathbf{y}_j||^2 = \text{tr}\{\mathbf{YLY}^T\},$$

In other words, we seek the solution

$$\hat{\mathbf{Y}} = \arg \min_{\mathbf{Y}:\mathbf{YDY^T}=\mathbf{I_t}} \text{tr}\{YLY^T\}.$$

where we restrict $\mathbf{Y}$ such that $\mathbf{YDY}^T = \mathbf{I}_t$ to prevent a collapse onto a subspace of fewer than $t1$ dimensions.

Diffusion Maps

The first and second steps of diffusion maps [Coifman et al., 2005] are the same as for Laplacian eigenmaps. But in the third step, a Markov chain is constructed over a graph of the data points.

Step 3 Spectral embedding

Let $X(t)$ denote a Markov random walk over $\mathcal{G}$ using the weights $\mathbf{W}$ and starting at $t = 0$. The transition probability is

$$p(x_j|x_i) = P\{\mathbf{X}(t) = x_j|\mathbf{X}(t) = x_i\} = \frac{w(x_i, x_j)}{\sum_{j=1}^{n} w(x_i, x_j)}.$$

Diffusion Maps

The transition matrix $\mathbf{P} = (p(x_j|x_i))$ has a set of eigenvalues $\lambda_0 = 1 \geqslant \lambda_1 \geqslant \cdots \geqslant \lambda_{n1} \geqslant 0$ and a set of left and right eigenvectors:

$$\phi_j^T \mathbf{P} = \lambda_j \phi_j^T, \mathbf{P}\psi_j = \lambda_j \psi_j,$$

where $\phi_k^T \psi_l = 1$ if $k = l$ and zero otherwise.
The largest eigenvalue, $\lambda_0 = 1$, has associated right eigenvector $\phi_0 = \mathbf{1}_n = (1, 1, \cdots, 1)^T$ and left eigenvector ϕ_0.
Thus, $\mathbf{P}$ is diagonalizable as the product

$$\mathbf{P} = \Psi \Lambda \Phi^T,$$

where $\Phi = (\phi_0, \cdots, \phi_{n-1})$, $\Psi = (\psi_0, \cdots, \psi_{n-1})$, and $\Lambda = \text{diag}\{\lambda_0, \cdots, \lambda_{n-1}\}$.

Diffusion Maps

The matrix $\mathbf{P}^m = (p_m(x_j|x_i))$ is $\mathbf{P}$ multiplied by itself m times. The transition probability of going from point x_i to point x_j in m time steps is

$$p_m(x_j|x_i) = P\{\mathbf{X}(t+m) = x_j|\mathbf{X}(t) = x_i\},$$

We define the **diffusion distance**

$$d_m^2(x_i, x_j) = ||p_m(\cdot|x_i) - p_m(\cdot|x_j)||^2 = \sum_z (p_m(z|x_i) - p_m(z|x_j))^2 w(z), \tag{7}$$

where weight function $w(z) = 1/\phi_0(z)$, and $\phi_0(z)$ is the probability of reaching point z after taking an infinite number of steps.

Diffusion Maps

The matrix $\mathbf{P}^m$ can be written as

$$\mathbf{P}^m = \Psi \Lambda^m \Phi^T,$$

with ijth element,

$$p_m(x_j|x_i) = \phi_0(x_j) + \sum_{k=1}^{n-1} \lambda_k^m \psi_k(x_i)\phi_k(x_j).$$

The diffusion distance becomes

$$d_m^2(x_i, x_j) = \sum_{k=1}^{n-1} \lambda_k^{2m} (\psi_k(x_i) - \psi_k(x_j))^2,$$

Diffusion Maps

We can approximate closely the diffusion distance using only the first t eigenvalues and corresponding eigenvectors,

$$d_m^2(x_i, x_j) = \sum_{k=1}^{n-1} \lambda_k^{2m} (\psi_k(x_i) - \psi_k(x_j))^2 = ||\Psi_m(x_i) - \Psi_m(x_j)||^2,$$

where the diffusion map is

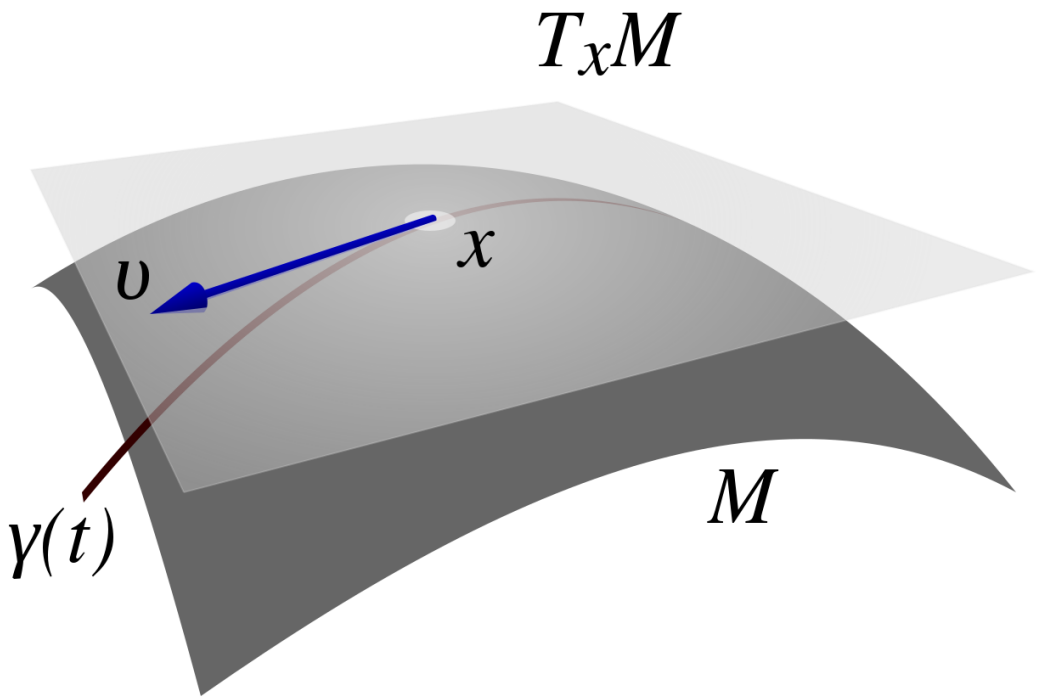
$$\Psi_m(x) = (\lambda_1^m \psi_1(x), \ldots, \lambda_t^m \psi_t(x))^T.$$

Thus, The coordinates of the n t-vectors are given by

$$\hat{\mathbf{Y}} = (\hat{y}_1, \ldots, \hat{y}_n) = (\Psi_m(x_1), \ldots, \Psi_m(x_n)).$$

Hessian LLE

The Hessian LLE (HLLE) [Donoho and Grimes, 2003] has the same first and third step as LLE, but the second step is different.
First, consider the differentiable manifold $M \subset \mathcal{R}^r$. Let $T_x(M)$ be a tangent space of the point $x \in M$, where $T_x(M)$ has the same number of dimensions as M itself.



Hessian LLE

The tangent Hessian matrix

1. We endow $T_x(M)$ with a (non-unique) system of orthonormal coordinates having the same inner product as $\mathcal{R}^r$;
2. Let N_x be a neighborhood of x such that each point $x' \in N_x$ has a unique closest point $\xi \in T_x(M)$; a point in N_x has local coordinates, $\xi = \xi(x) = (\xi_1(x), \ldots, \xi_t(x))^T$, and these coordinates are referred to as tangent coordinates;
3. Suppose $f : M \to \mathcal{R}$ is a C^2-function near x. If the point $x' \in N_x$ has local coordinates $\xi = \xi(x) \in \mathcal{R}^t$, then the rule $g(\xi) = f(x')$ defines a C^2-function $g : U \to \mathcal{R}$, where U is a neighborhood of $0 \in \mathcal{R}^r$.

Hessian LLE

The tangent Hessian matrix

The tangent Hessian matrix, which measures the curviness of f at the point $x \in M$, is defined as the ordinary $(t \times t)$ Hessian matrix of g,

$$\mathbf{H}_f^{\tan}(\mathbf{x}) = \left(\frac{\partial^2 g(\xi)}{\partial \xi_i \partial \xi_j} | \xi = \mathbf{0} \right).$$

Step 1 Nearest-Neighbor Search

Fix an integer K and let N_i^K denote the K nearest neighbors of the data point x_i using Euclidean distance.

Hessian LLE

Step 2 Estimate Tangent Hessian Matrices

1. Assuming local linearity of the manifold M in the region of the neighborhood N_i^K, form the $(r \times r)$ covariance matrix $\mathbf{M}_i$ of the K neighborhood-centered points $x_j \overline{x}_i$, $j \in N_i^K$, where $\overline{x}_i = n^1 \sum_{j \in N_i^K} x_j$, and compute a PCA of the matrix M_i;
2. Assuming $K \geqslant t$, the first t eigenvectors of M_i give the tangent coordinates of the K points in N_i^K and provide the best-fitting t-dimensional linear subspace corresponding to x_i;
3. construct an LS estimate, $\hat{\mathbf{H}}_i$, of the local Hessian matrix $\mathbf{H}_i$ as follows: build a matrix $\mathbf{Z}_i$ by putting all squares and cross-products of the columns of $\mathbf{M}_i$ up to the t—th order in its columns, including a column of 1s; so, $\mathbf{Z}_i$ has $1 + t + t(t+1)/2$ columns and K rows;

Hessian LLE

4. apply a GramSchmidt orthonormalization to Z_i. The estimated $(t(t+1)/2K)$ tangent Hessian matrix $\hat{\mathbf{H}}_i$ is given by the transpose of the last $t(t+1)/2$ orthonormal columns of Z_i.

Step 3 Spectral embedding

The estimated local Hessian matrices, $\hat{\mathbf{H}}_i$, $i = 1, 2, \ldots, n$, are used to construct a sparse, symmetric, $(r \times r)$-matrix $\hat{H} = (\hat{H}_{kl})$, where

$$\hat{H}_{kl} = \sum_i \sum_j ((\hat{\mathbf{H}}_i)_{jk} (\hat{\mathbf{H}}_i)_{jl}).$$

$\hat{H}$ is a discrete approximation to the functional H. Now apply Step 3 of the LLE algorithm on $\hat{H}$.

Stochastic Neighbor Embedding (SNE)

SNE [Hinton and Roweis, 2003] converts the high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities.
The similarity of datapoint x_j to x_i is the conditional probability $p_{j|i}$, that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i.

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

where σ_i (we will talk about how to determine it later) is the variance of the Gaussian that is centered on datapoint x_i.

SNE

For the low-dimensional counterparts u_i and u_j of the high-dimensional datapoints x_i and x_j, we denote similar conditional probability by $q_{j|i}$ and set the variance of the Gaussian to $\frac{1}{\sqrt{2}}$,

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

Since we are only interested in modeling pairwise similarities, we set $p_{i|i} = 0$ and $q_{i|i} = 0$.

SNE

SNE minimizes the sum of Kullback-Leibler divergences over all datapoints using a gradient descent method.

The cost function C

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}},$$

P_i represents the conditional probability distribution over all other datapoints given datapoint x_i, and Q_i represents the conditional probability distribution over all other map points given map point u_i.

SNE

How to determine the variance σ_i of the Gaussian that is centered over each high-dimensional datapoint, x_i?

SNE performs a binary search for the value of σ_i that produces a P_i with a fixed perplexity that is specified by the user.

The perplexity

$$Perp(P_i) = 2^{H(P_i)},$$

where $H(P_i)$ is the Shannon entropy of P_i measured in bits

$$H(P_i) = -\sum_j p_{j|i} log_2 p_{j|i}.$$

SNE

The minimization of the cost function is performed using a gradient descent method

$$\frac{\delta C}{\delta u_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(u_i - u_j),$$

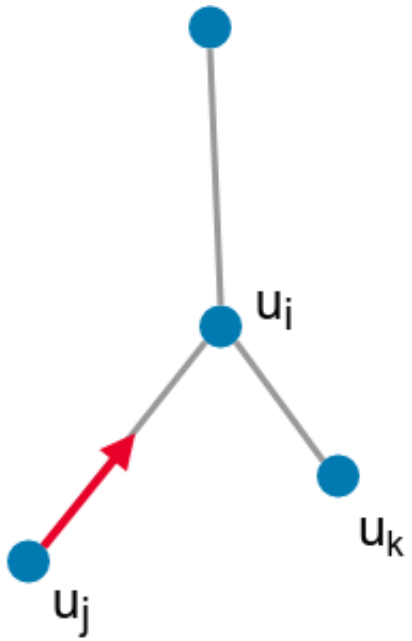
In order to speed up the optimization and to avoid poor local minima, a relatively large momentum term is added to the gradient.

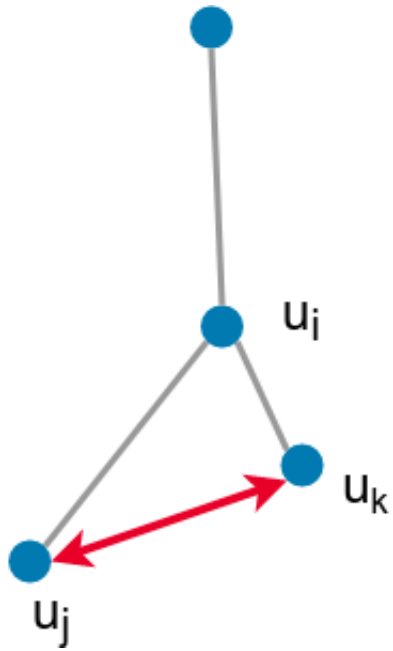
$$Y^{(t)} = Y^{(t-1)} + \eta \frac{\delta C}{\delta Y} + \alpha(t)(Y^{(t-1)} - Y^{(t-2)}),$$

where $Y(t)$ indicates the solution at iteration t, η indicates the learning rate, and $\alpha(t)$ represents the momentum at iteration t.

SNE

The gradient may be interpreted as the resultant force created by a set of springs between the map point y_i and all other map points y_j. All springs exert a force along the direction $(y_i - y_j)$.
The gradient may be interpreted as the resultant force created by a set of springs between the map point u_i and all other map points u_j. All springs exert a force along the direction $(u_i - u_j)$.





SNE

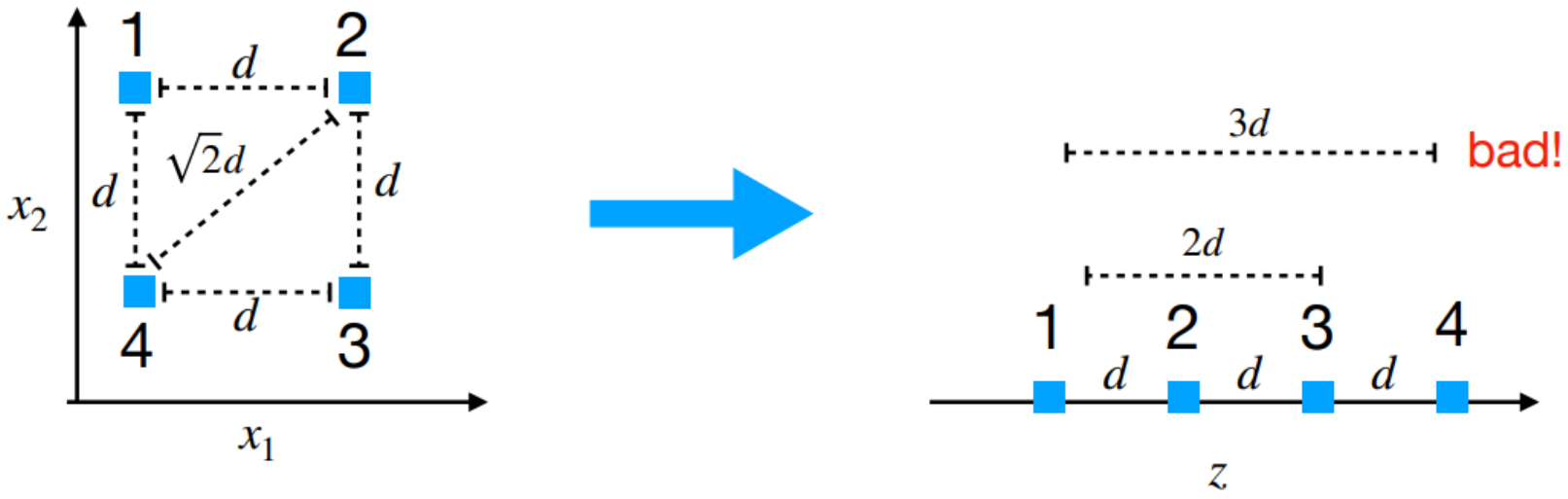
$$\frac{\delta C}{\delta u_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(u_i - u_j).$$

1. The spring between u_i and u_j repels or attracts the map points depending on whether the distance between the two in the map is too small or too large to represent the similarities between the two high-dimensional datapoints.
2. The force exerted by the spring between u_i and u_j is proportional to its length, and also proportional to its stiffness, which is the mismatch $(pj|i - q_{j|i} + p_{i|j} - q_{i|j})$ between the pairwise similarities of the data points and the map points.

What's the Problem?

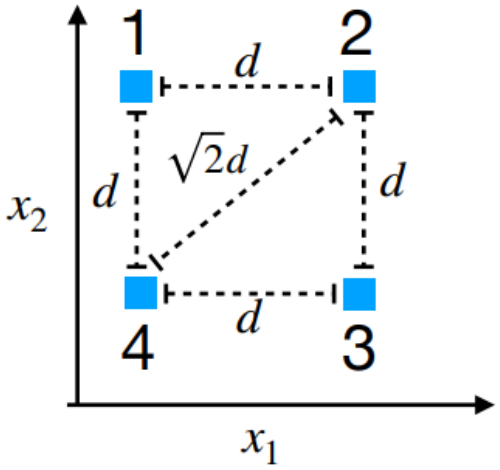
Although SNE constructs reasonably good visualizations, it is hampered **(1)** by a cost function that is difficult to optimize and **(2)** by a problem we refer to as the crowding problem.

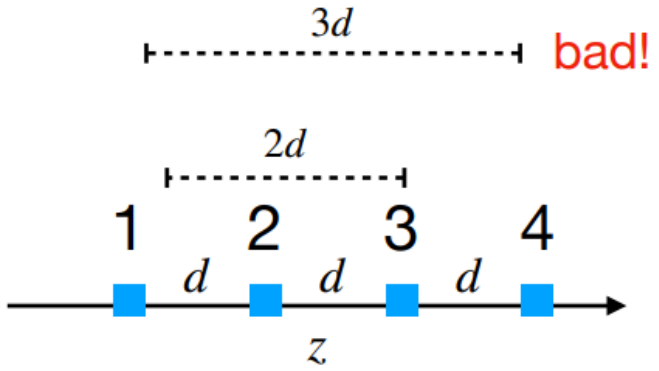
Crowding Problem

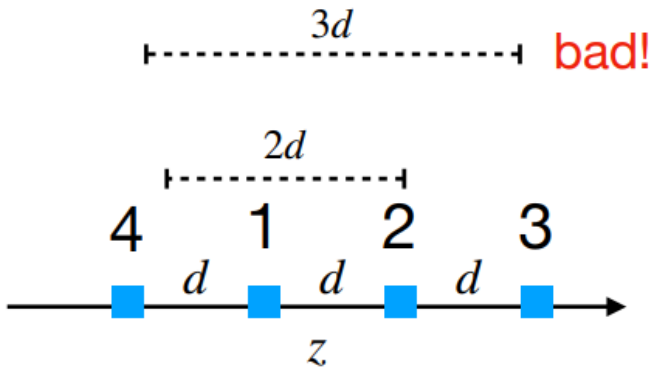


What's the Problem?

Crowding Problem

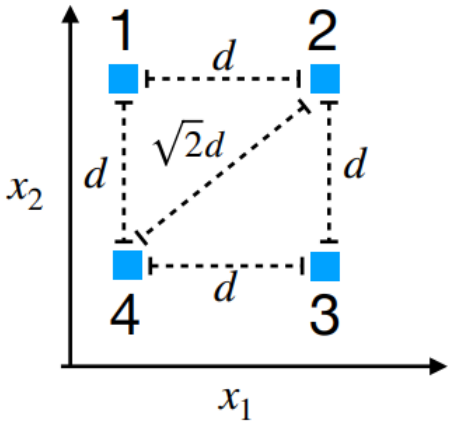


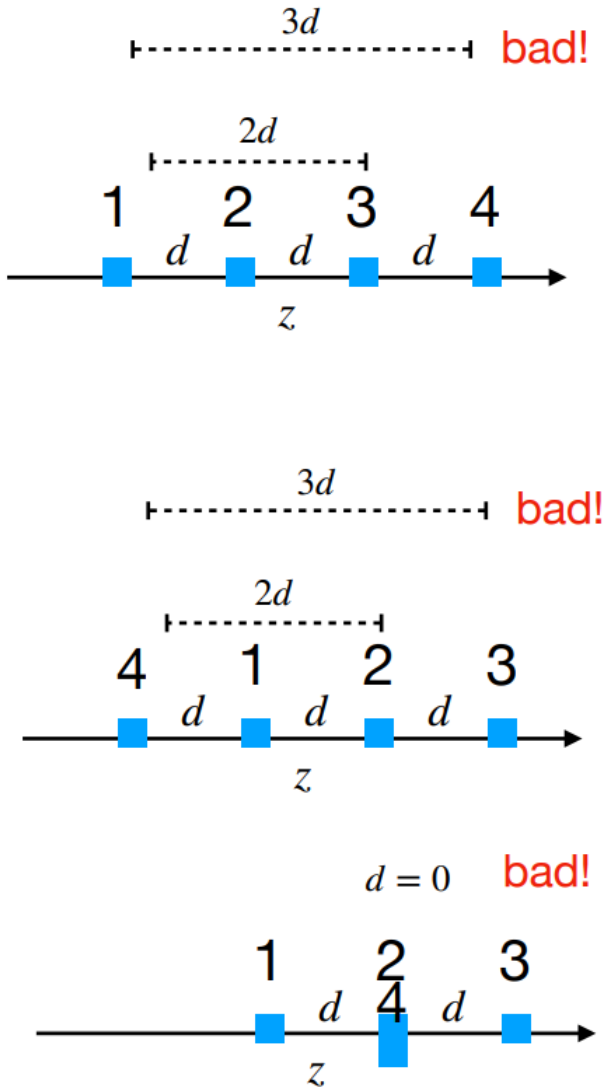




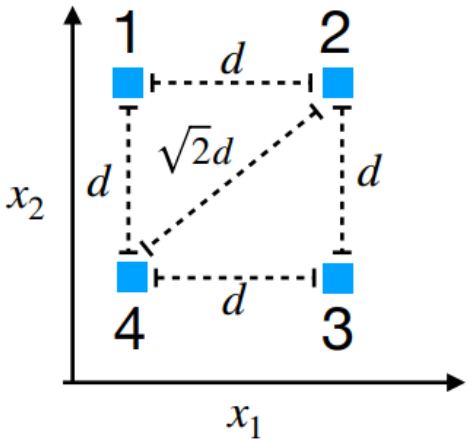
What's the Problem?

Case where distance representation in low dimension is impossible!

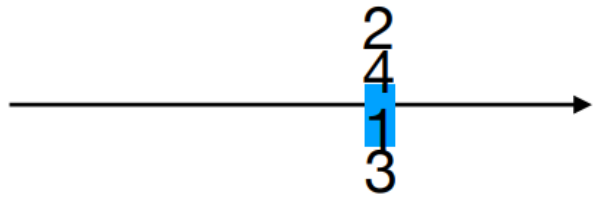




What's the Problem?



What would regular SNE do?



Crowding problem!

Squashes all points!

Symmetric SNE Makes SNE Faster

SNE minimizes the sum of the Kullback-Leibler divergences between the conditional probabilities $p_{j|i}$ and $q_{j|i}$;
Symmetric SNE [Cook et al., 2007] minimizes a single Kullback-Leibler divergence between a joint probability distribution.

Cost function in Symmetric SNE

$$C = KL(p||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}.$$

$p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ for $\forall i, j$.

The pairwise similarities in the low-dimensional map

$$q_{ij} = \frac{\exp(-||u_i - u_j||^2)}{\sum_{k \neq l} \exp(-||u_k - u_l||^2)}.$$

Symmetric SNE Makes SNE Faster

The pairwise similarities in the high-dimensional space is

$$p_{ij} = \frac{\exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} \exp(-||x_k - x_l||^2/2\sigma^2)},$$

But for an outlier in high-dimensional datapoint x_i,
all pairwise distances $||x_i - x_j||^2$ are large for x_i
$\Rightarrow$ the values of p_{ij} are extremely small for all j
$\Rightarrow$ the location of u_i has very little effect on the cost function
$\Rightarrow$ the position of u_i is not well determined by the positions of the other map

How to solve this problem?

t-SNE

Define the joint probabilities p_{ij} in the high-dimensional space to be the symmetrized conditional probabilities

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

This ensures that $\sum_j p_{ij} > \frac{1}{2n}$ for all datapoints x_i, as a result of which each datapoint x_i makes a significant contribution to the cost function.

t-SNE

The main advantage: the simpler form of its gradient, which is faster to compute. The gradient of symmetric SNE is fairly similar to that of asymmetric SNE, and is given by

$$\frac{\delta C}{\delta u_i} = 4 \sum_j (p_{ij} - q_{ij})(u_i - u_j).$$

BUT Symmetric SNE still can't solve the crowding problem!
However, Symmetric cost function is applied in t-SNE.

How t-SNE prevent crowding problem?

A Student t-distribution with one degree of freedom (which is the same as a Cauchy distribution) as the heavy-tailed distribution is employed in the low-dimensional map.

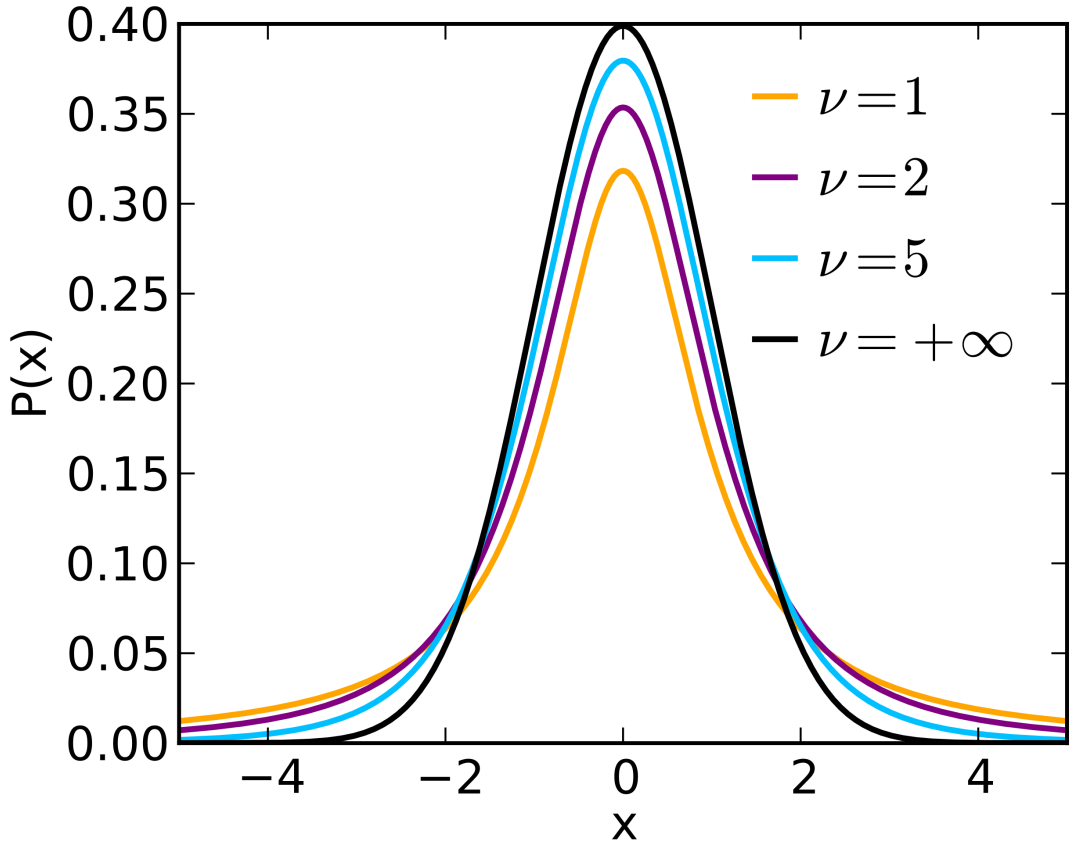
The density function of t-distribution is

$$f(x) = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})}(1 + \frac{x^2}{v})^{-\frac{v+1}{2}},$$

if $v = 1$, it is the standard Cauchy distribution

$$f(x) = \frac{1}{\pi(1 + x^2)}.$$

t-SNE



As v increases, the distribution gets close to a Gaussian distribution.

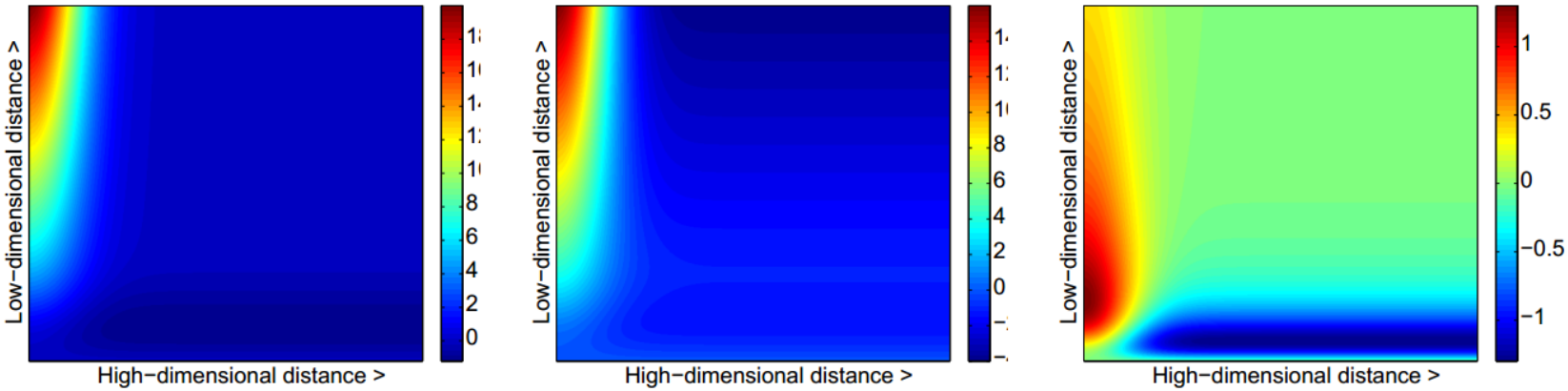
t-SNE

Using Cauchy distribution, the **joint** probabilities q_{ij} are defined as

$$q_{ij} = \frac{(1 + ||u_i - u_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||u_k - u_l||^2)^{-1}}.$$

Notice: only joint probabilities in **low** dimensional space are changed from SNE!

t-SNE



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

Negative gradient if points are too close in low-dim space to provide some repulsion against crowding.
The gradient of the Kullback-Leibler divergence between P and the Student-t based joint probability distribution Q

t-SNE gradient

$$\frac{\delta C}{\delta u_i} = 4 \sum_j (p_{ij} - q_{ij})(u_i - u_j)(1 + ||u_i - u_j||^2)^{-1}.$$

t-SNE Experiments

The MNIST data set

60000 grayscale images of handwritten digits. The digit images have $28 \times 28 = 784$ pixels.

The Olivetti faces data set

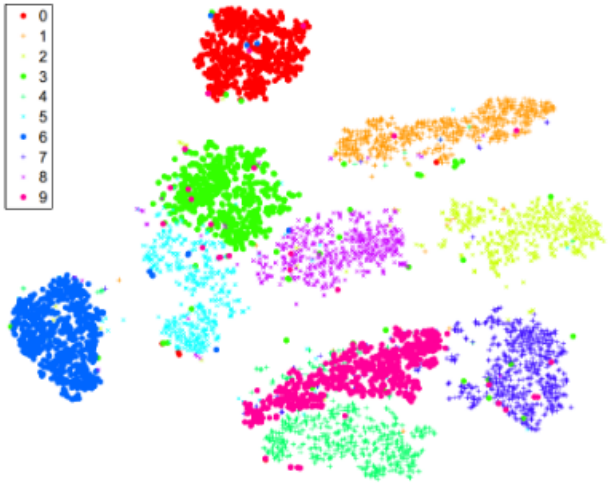
Images of 40 individuals with small variations in viewpoint, large variations in expression, and occasional addition of glasses. 400 images (10 per individual) of size $92 \times 112 = 10304$ pixels, and is labeled according to identity.

The COIL-20 data set

Images of 20 different objects viewed from 72 equally spaced orientations, yielding a total of 1,440 images. The images contain $32 \times 32 = 1024$ pixels.

t-SNE Experiments

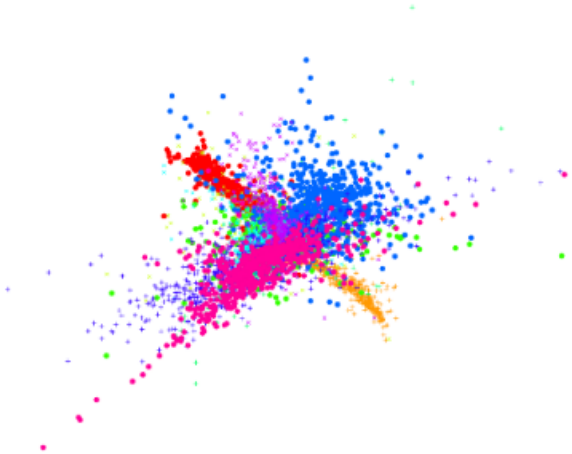
Visualizations of 6,000 handwritten digits from the MNIST data set



t-SNE

Sammon mapping

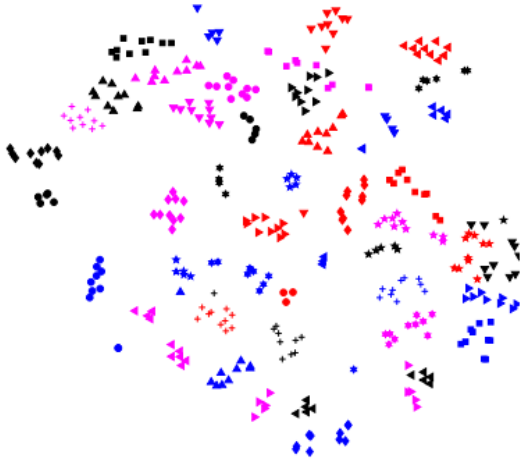
Isomap



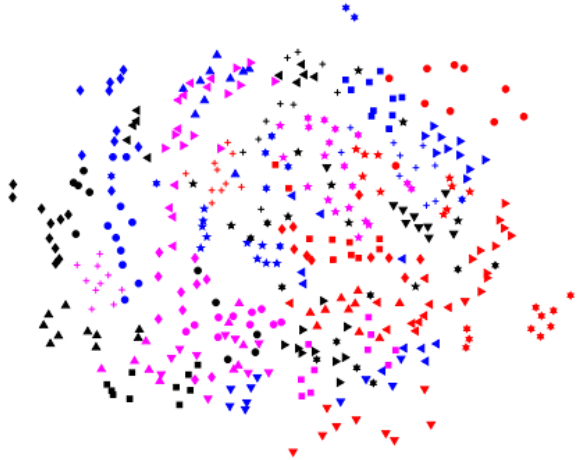
LLE

t-SNE Experiments

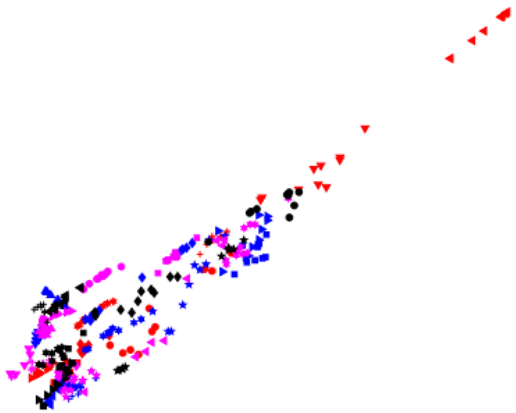
Visualizations of the Olivetti faces data set.



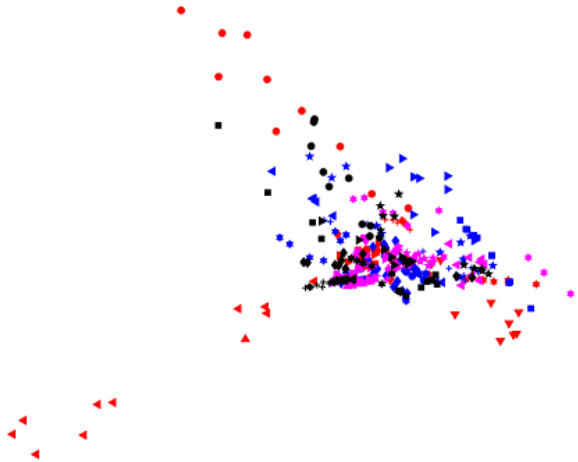
(a) Visualization by t-SNE.



(b) Visualization by Sammon mapping.



(c) Visualization by Isomap.

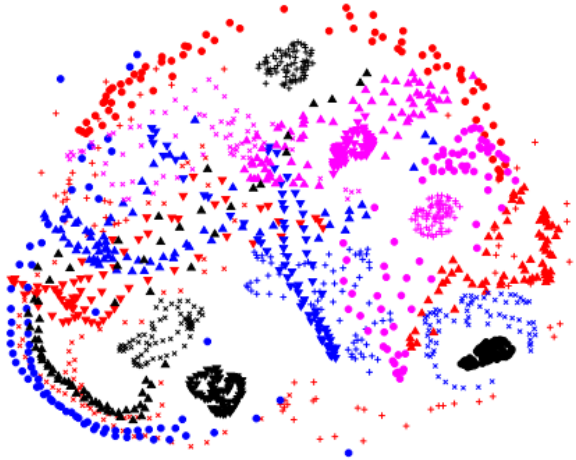


(d) Visualization by LLE.

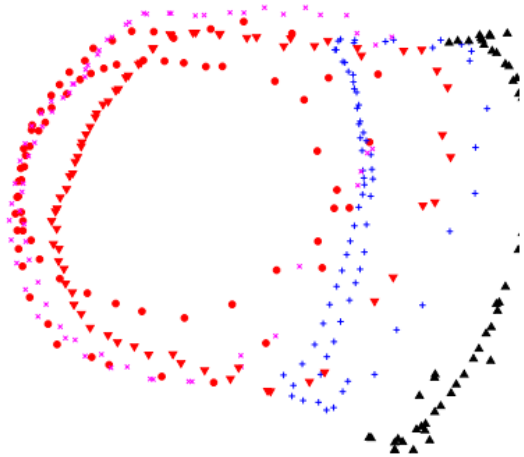
t-SNE Experiments

Visualizations of the COIL-20 data set.

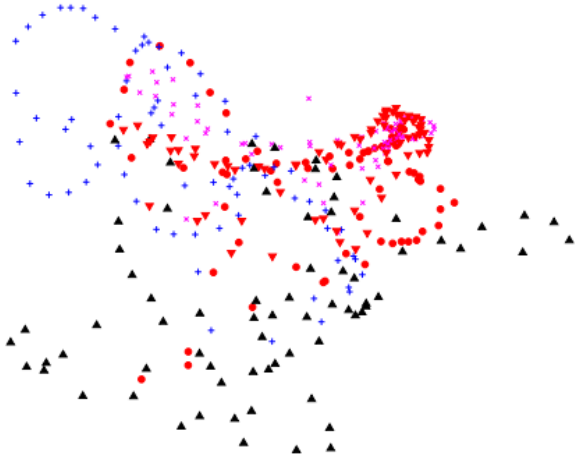
(a) Visualization by t-SNE.



(b) Visualization by Sammon mapping.



(c) Visualization by Isomap.



(d) Visualization by LLE.

t-SNE Pros

1. Solves the crowding problem efficiently;
2. Great for visualizing datasets in 1D or 2D;
3. The optimization of the t-SNE cost function is much easier than the optimization of the cost functions of SNE and UNI-SNE;

t-SNE Cons

1. It is not obvious how t-SNE will perform when the dimensionality of the data is reduced to $d > 3$ dimensions;
2. Mainly based on local properties of the data $\rightarrow$ sensitive to the curse of the intrinsic dimensionality of the data [Bengio et al., 2009];
3. Instead of MDS, Isomap, LLE, and diffusion maps, cost function of t-SNE is not convex, as a result of which several optimization parameters need to be chosen.

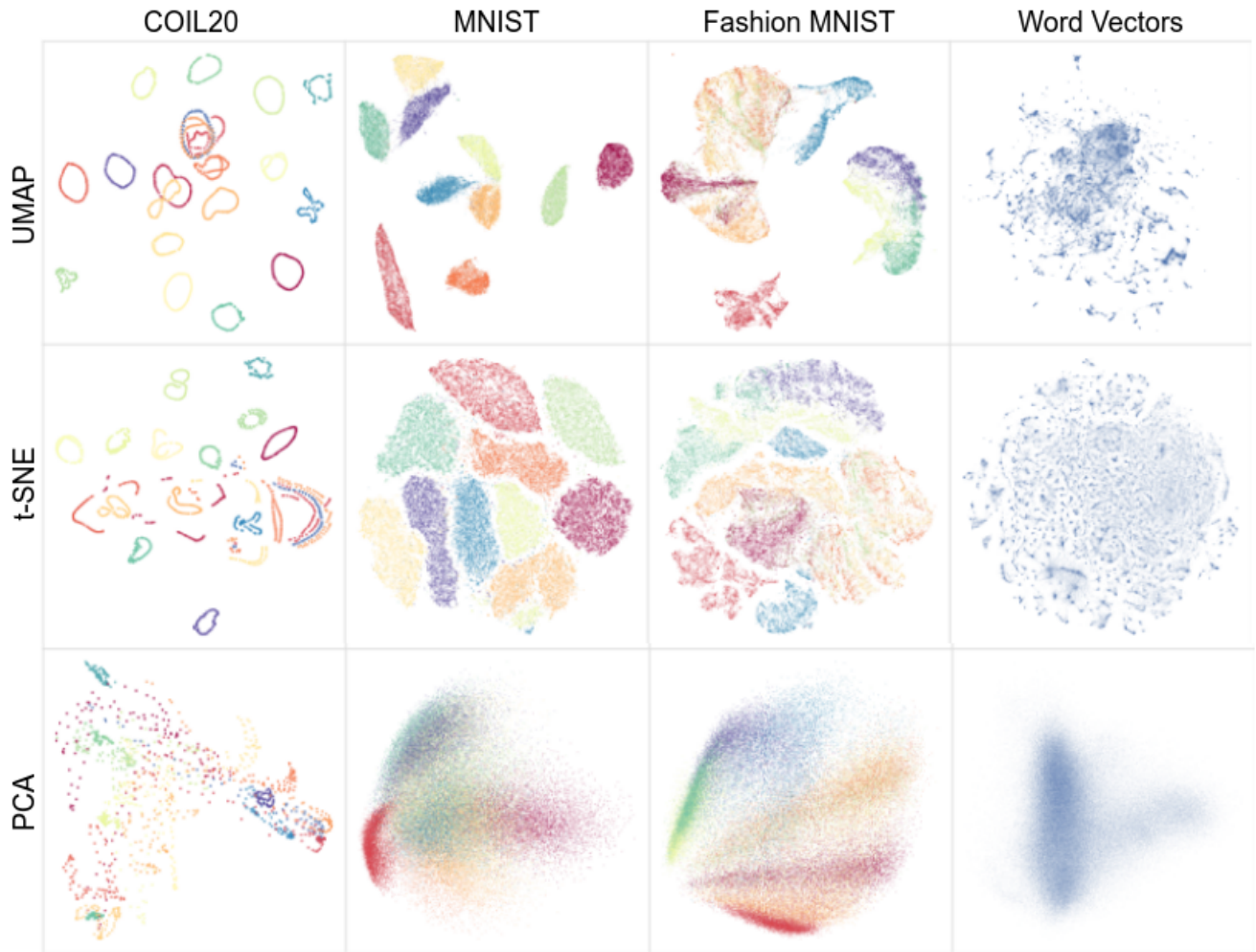
UMAP

Uniform Manifold Approximation and Projection (UAMP) [McInnes et al., 2018] is constructed from a theoretical framework based in Riemannian geometry and algebraic topology.
UMAP partly overcomes those cons of t-SNE:

1. UMAP algorithm is competitive with t-SNE for visualization quality;
2. UAMP preserves more of the global structure with superior run time performance (time complexity $O(n^{1.14})$ compared to $O(n^2)$ in t-SNE);
3. UMAP has no computational restrictions on embedding dimension.

UMAP

UMAP relects much of the large scale global structure that is well represented by PCA, while also preserving the local structure similar to t-SNE.



UMAP-Runtime

	UMAP	FIt-SNE	t-SNE	LargeVis	Eigenmaps	Isomap
Pen Digits (1797x64)	9s	48s	17s	20s	**2s**	**2s**
COIL20 (1440x16384)	**12s**	75s	22s	82s	47s	58s
COIL100 (7200x49152)	**85s**	2681s	810s	3197s	3268s	3210s
scRNA (21086x1000)	**28s**	131s	258s	377s	470s	923s
Shuttle (58000x9)	**94s**	108s	714s	615s	133s	–
MNIST (70000x784)	**87s**	292s	1450s	1298s	40709s	–
F-MNIST (70000x784)	**65s**	278s	934s	1173s	6356s	–
Flow (100000x17)	**102s**	164s	1135s	1127s	30654s	–
Google News (200000x300)	**361s**	652s	16906s	5392s	–	–

UAMP

How UAMP works? How it comes out?

We firstly put UMAP in the t-SNE and LargeVis methods family to tell the story.

1. We are concerned with defining similarities between two objects i and j in the high dimensional input space X and low dimensional embedded space Y, these are normalized and symmetrized in various ways;

2. Quantities with the subscript ij are symmetric, i.e. $v_{ij} = v_{ji}$. Extending the conditional probability notation used in t-SNE, $j|i$ indicates an asymmetric similarity, i.e. $v_{j|i} \neq v_{i|j}$.

UMAP I

t-SNE review

t-SNE defines input probabilities in three stages.

1. For each pair of points,i and j, in X, a pair-wise similarity, v_{ij}, is calculated:
$$v_{j|i} = exp(-||x_i - x_j||_2^2/2\sigma_1^2).$$
2. the similarities are converted into n conditional probability distributions by normalization:
$$p_{j|i} = \frac{v_{j|i}}{\sum_{k \neq i} v_{k|i}},$$
σ_i is chosen by searching for a value such that the perplexity of the probability distribution p_{ji} matches a user-specified value;

3. These probability distributions are symmetrized and then further normalized over the entire matrix of values to give:

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

Similarities between pairs of points in the output space Y are defined as

$$w_{ij} = (1 + ||y_i - y_j||_2^2)^{-1},$$

after normalized

$$q_{ij} = \frac{w_{ij}}{\sum_{k \neq l} w_{kl}}.$$

UMAP

From t-SNE to LargeVis [Tang et al., 2016], and to UMAP

$$C_{t-SNE} = \sum_{i \neq j} p_{ij} \log p_{ij} - p_{ij} \log q_{ij}$$

$$q_{ij} = \frac{w_{ij}}{\sum_{k \neq l} w_{kl}}$$

Without this normalization

only calculate $v_{j|i}$ for n nearest neighbors of i

$$C_{LV} = \sum_{i \neq j} p_{ij} \log w_{ij} + \gamma \sum_{i \neq j} \log (1 - w_{ij})$$

$$p_{j|i} = \frac{v_{j|i}}{\sum_{k \neq i} v_{k|i}}$$

Without this normalization

$$C_{UMAP} = \sum_{i \neq j} v_{ij} \log v_{ij} + (1 - v_{ij}) \log (1 - v_{ij})$$

constant contributions

$$-v_{ij} \log w_{ij} - (1 - v_{ij}) \log (1 - w_{ij})$$

optimizable contributions

Equal

$$C_{UMAP} = \sum_{i \neq j} v_{ij} \log \left(\frac{v_{ij}}{w_{ij}} \right) + (1 - v_{ij}) \log \left(\frac{1 - v_{ij}}{1 - w_{ij}} \right)$$

UMAP

The above changes only optimize the time efficiency, then how UMAP preserves more of the global structure?
UMAP does not use the same definitions for v_{ij} and w_{ij} as in t-SNE and LargeVis.
In the high dimensional space, the similarities $v_{j|i}$ are the local fuzzy simplicial set memberships, based on the smooth nearest neighbors distances:

$$v_{j|i} = \exp[(-d(x_i, x_j) - \rho_i)/\sigma_i],$$

$v_{j|i}$ is calculated only for n approximate nearest neighbors and $v_{j|i} = 0$ for all other j.
$d(x_i, x_j)$ is the distance between x_i and x_j.
ρ_i is the distance to the nearest neighbor of i.

UMAP

How to understand local fuzzy simplicial set, even this formula?

ρ_i is defined as

$$\rho_i = \min\{d(x_i, x_{i_j}) | 1 \leqslant j \leqslant k, d(x_i, x_{i_j}) > 0\},$$

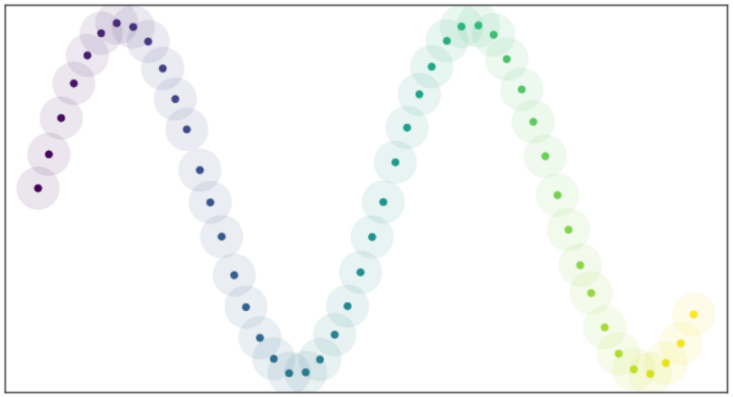
σ_i is the normalizing factor, which is chosen such that

$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right) = \log_2(k),$$

So $v_{j|i} = exp[(-d(x_i, x_j) - \rho_i)/\sigma_i]$ can be viewed as a normalized smooth distance.

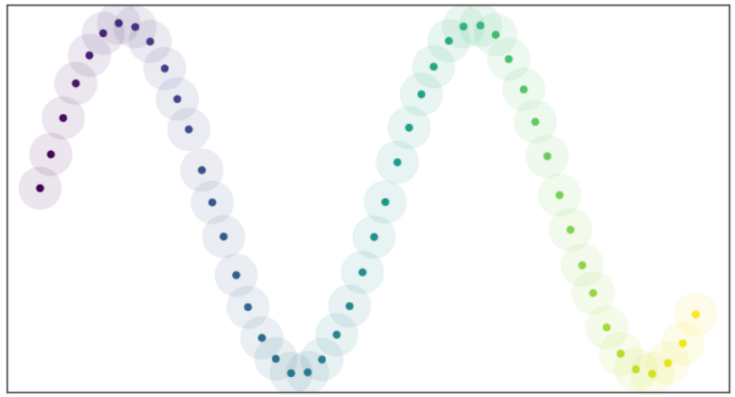
UMAP

If the data is uniformly distributed on the manifold then the cover will be “good”.

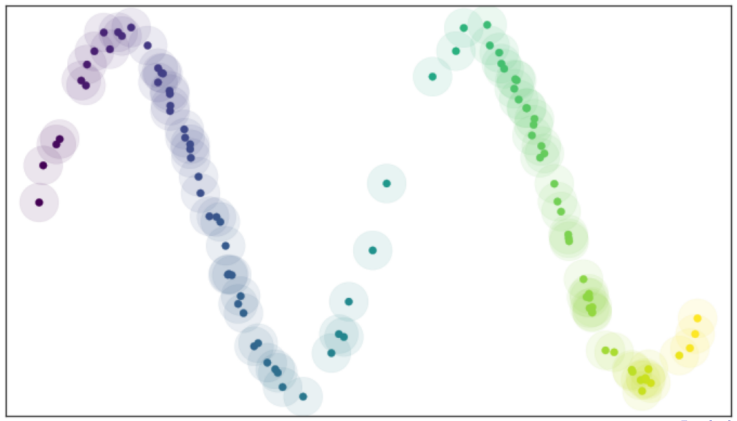


UMAP

If the data is uniformly distributed on the manifold then the cover will be “good”.

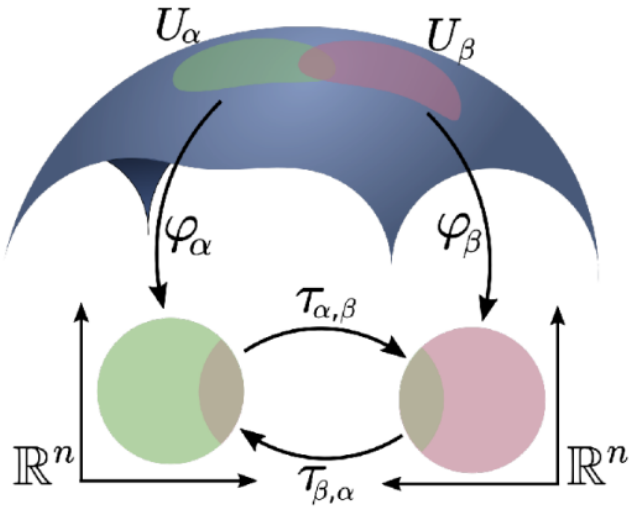


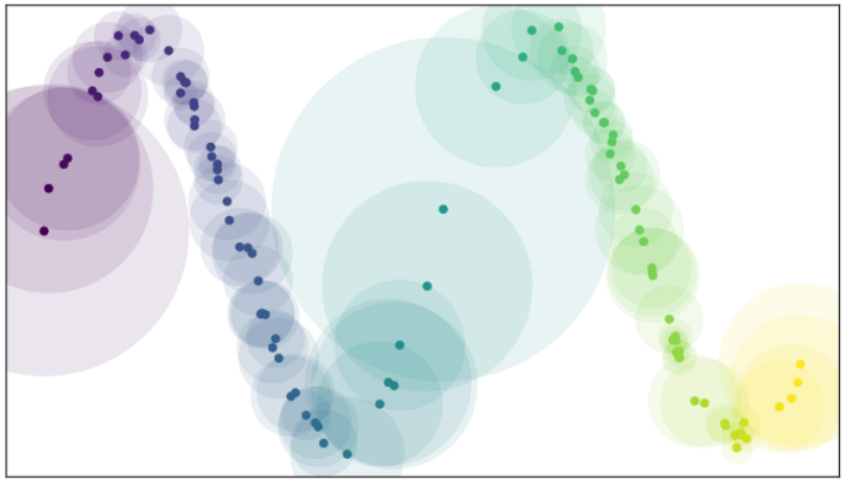
However, the real data is often not uniformly distributed on the manifold like this



UMAP

But we can define a Riemannian metric on the manifold to make the data uniformly distributed on the manifold!

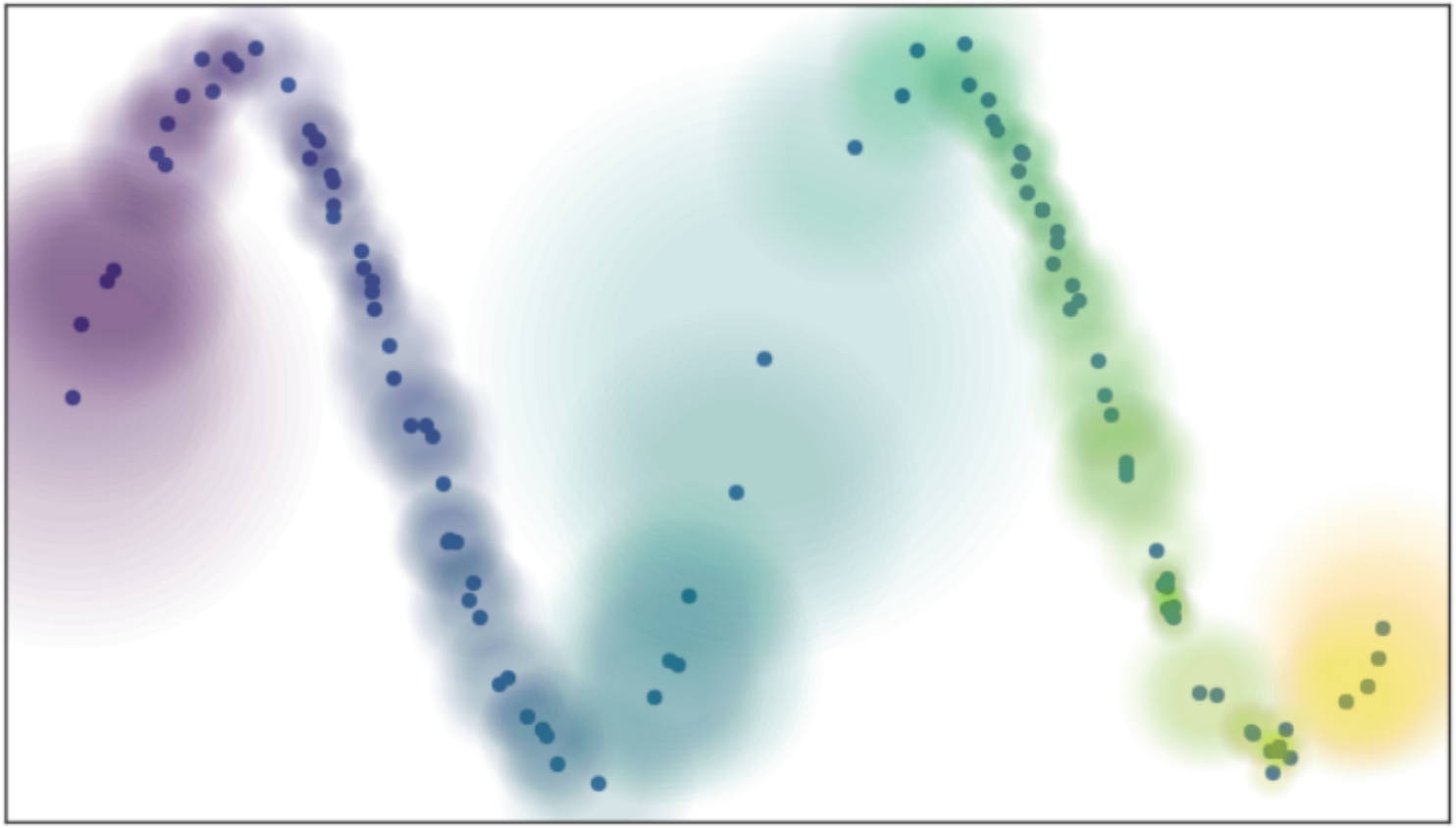




So formula such as $v_{j|i} = exp[(-d(x_i, x_j) - \rho_i)/\sigma_i]$ is actually a Riemannian metric.

UMAP

Why choose a fixed radius? Why not have a fuzzy cover?
So in UMAP, the local fuzzy simplicial set is constructed



UMAP

The local fuzzy simplicial set is constructed by the following Algorithm

Algorithm 2 Constructing a local fuzzy simplicial set

function LocalFuzzySimplicialSet(X, x, n)
knn, knn-dists $\leftarrow$ ApproxNearestNeighbors(X, x, n)
$\rho \leftarrow$ knn-dists[1] ▷ Distance to nearest neighbor
$\sigma \leftarrow$ SmoothKNNDist(knn-dists, n, ρ) ▷ Smooth approximator to knn-distance
fs-set$_0 \leftarrow X$
fs-set$_1 \leftarrow \{([x, y], 0) \mid y \in X\}$
for all $y \in$ knn **do**
$d_{x,y} \leftarrow \max\{0, \text{dist}(x, y) - \rho\}/\sigma$
fs-set$_1 \leftarrow$ fs-set$_1 \cup ([x, y], \exp(-d_{x,y}))$
return fs-set

UMAP

Algorithm 3 Compute the normalizing factor for distances σ

function SmoothKNNDist(knn-dists, n, ρ)

Binary search for σ such that $\sum_{i=1}^{n} \exp(-(\text{knn-dists}_i - \rho)/\sigma) = \log_2(n)$

return σ

Symmetrization is carried out by fuzzy set union using the probabilistic t-conorm and can be expressed as:

$$v_{ij} = (v_{j|i} + v_{i|j}) - v_{j|i} v_{i|j}),$$

UMAP

The low dimensional similarities are given by

$$w_{ij} = (1 + a||y_i - y_j||_2^{2b})^{-1}$$

where a and b are user-defined positive values. The final UMAP algorithm is

Algorithm 1 UMAP algorithm

function UMAP(X, n, d, min-dist, n-epochs)
  for all $x \in X$ **do**
    fs-set$[x]$ $\leftarrow$ LocalFuzzySimplicialSet(X, x, n)
  top-rep $\leftarrow \bigcup_{x \in X}$ fs-set$[x]$ ▷ We recommend the probabilistic t-conorm
  $Y \leftarrow$ SpectralEmbedding(top-rep, d)
  $Y \leftarrow$ OptimizeEmbedding(top-rep, Y, min-dist, n-epochs)
  return Y

UMAP

The Spectral embedding algorithm is

Algorithm 4 Spectral embedding for initialization

function SPECTRALEMBEDDING(top-rep, d)

$A \leftarrow$ 1-skeleton of top-rep expressed as a weighted adjacency matrix

$D \leftarrow$ degree matrix for the graph A

$L \leftarrow D^{1/2}(D - A)D^{1/2}$

evec $\leftarrow$ Eigenvectors of L (sorted)

$Y \leftarrow$ evec$[1..d+1]$ ▷ 0-base indexing assumed

return Y

UMAP-Cons

1. UMAP lacks the strong interpretability of Principal Component Analysis (PCA) and related techniques such a NonNegative Matrix Factorization (NMF);
2. One of the core assumptions of UMAP is that there exists manifold structure in the data, so care must be taken with small sample sizes of noisy data;
3. UMAP concerns itself primarily with accurately representing local structure, although it captures more global structure than t-SNE etc;
4. The use of approximate nearest neighbor algorithms, and the negative sampling used in optimization, can result in suboptimal embeddings in small datasets.

Summary

1. For each popular method such as t-SNE, there are many methods as extensions to fix the original algorithm's problems;
2. Can a method outperforms UAMP in the future? Or are we satisfied with UMAP? Of course not. However, we can learn from UMAP that "Nothing is more pratical than a good theory";
3. The connection between dimension reduction and clustering is fascinating, some methods do the two tasks at the same time, such as deep continual clustering [Shah and Koltun, 2018];
4. In some works, manifold theory is used to interpret deep learning, while in some other wroks, deep learning is used to do dimension reduction in the viewpoint of manifold.

References I

Belkin, M. and Niyogi, P. (2002).

Laplacian eigenmaps and spectral techniques for embedding and clustering.
In Advances in neural information processing systems, pages 585–591.

Bengio, Y. et al. (2009).

Learning deep architectures for ai.
Foundations and trends® in Machine Learning, 2(1):1–127.

Budninskiy, M., Yin, G., Feng, L., Tong, Y., and Desbrun, M. (2018).

Parallel transport unfolding: A connection-based manifold learning approach.
arXiv preprint arXiv:1806.09039.

Choi, H. and Choi, S. (2007).

Robust kernel isomap.
Pattern Recognition, 40(3):853–862.

Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S. W. (2005).

Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps.
Proceedings of the national academy of sciences, 102(21):7426–7431.

Cook, J., Sutskever, I., Mnih, A., and Hinton, G. (2007).

Visualizing similarity data with a mixture of maps.
In Artificial Intelligence and Statistics, pages 67–74.

Donoho, D. L. and Grimes, C. (2003).

Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data.
Proceedings of the National Academy of Sciences, 100(10):5591–5596.

References II

Hinton, G. E. and Roweis, S. T. (2003).
Stochastic neighbor embedding.
In Advances in neural information processing systems, pages 857–864.

Hotelling, H. (1933).
Analysis of a complex of statistical variables into principal components.
Journal of educational psychology, 24(6):417.

Kruskal, J. B. and Wish, M. (1978).
Multidimensional scaling, volume 11.
Sage.

McInnes, L., Healy, J., and Melville, J. (2018).
Umap: Uniform manifold approximation and projection for dimension reduction.
arXiv preprint arXiv:1802.03426.

Roweis, S. T. and Saul, L. K. (2000).
Nonlinear dimensionality reduction by locally linear embedding.
science, 290(5500):2323–2326.

Shah, S. A. and Koltun, V. (2018).
Deep continuous clustering.
arXiv preprint arXiv:1803.01449.

Silva, V. D. and Tenenbaum, J. B. (2003).
Global versus local methods in nonlinear dimensionality reduction.
In Advances in neural information processing systems, pages 721–728.

References III

Spruyt, V. (2014).

The curse of dimensionality in classification.
https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/.
Accessed April 16, 2014.

Tang, J., Liu, J., Zhang, M., and Mei, Q. (2016).

Visualizing large-scale and high-dimensional data.
In Proceedings of the 25th international conference on world wide web, pages 287–297. International World Wide Web Conferences Steering Committee.

Tenenbaum, J. B., De Silva, V., and Langford, J. C. (2000).

A global geometric framework for nonlinear dimensionality reduction.
science, 290(5500):2319–2323.