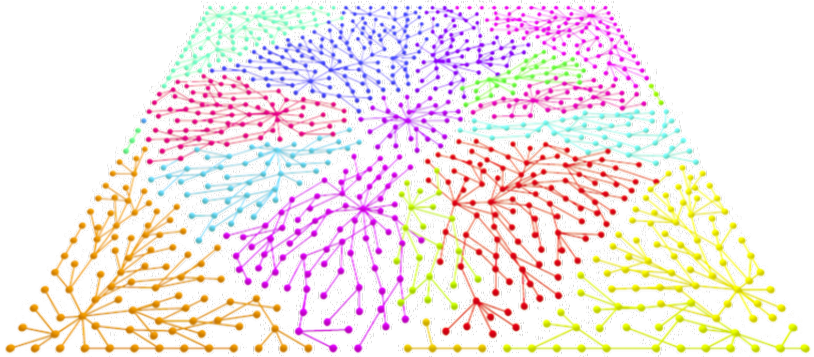
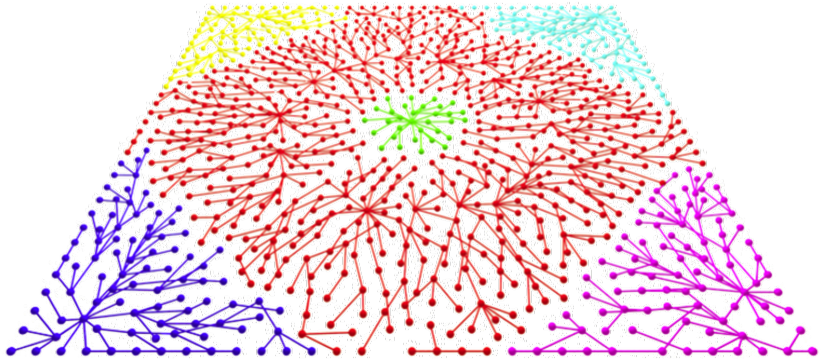
CS233, CME251: Geometric and Topological Data Analysis

Leonidas Guibas
Computer Science Department
Stanford University

Lecture 6
14 April 2021





Last Time: Graph Methods and Spectral Approaches

Spectral Graph Theory

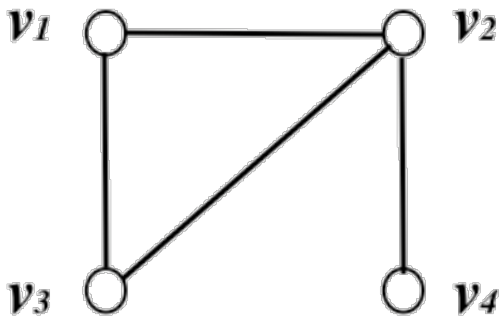
- The *spectral graph theory* studies the properties of graphs via the eigenvalues and eigenvectors of their associated graph matrices: the *adjacency matrix* and the *graph Laplacian* and its variants.
- Both matrices have been extremely well studied from an algebraic point of view.
- The Laplacian allows a natural link between discrete representations, such as graphs, and continuous representations, such as vector spaces and manifolds.
- The most important application of the Laplacian is *spectral clustering* that corresponds to a computationally tractable solution to the *graph partitionning problem*.
- Another application is *spectral matching* that solves for *graph matching*.

Adjacency Matrices

- For a graph with n vertices, the entries of the $n \times n$ adjacency matrix are defined by:

$$\mathbf{A} := \begin{cases} A_{ij} = 1 & \text{if there is an edge } e_{ij} \\ A_{ij} = 0 & \text{if there is no edge} \\ A_{ii} = 0 \end{cases}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$



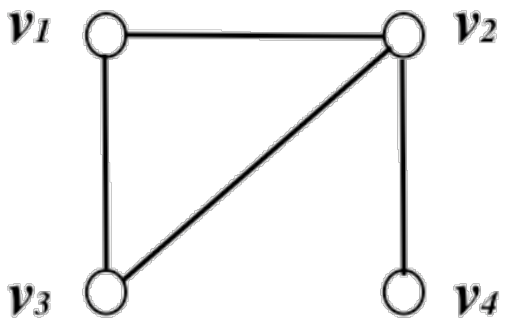
Graph (Unnormalized) Laplacian

- $\mathbf{L} = \nabla^\top \nabla$
- $(\mathbf{L}\boldsymbol{f})(v_i) = \sum_{v_j \sim v_i} (f(v_i) - f(v_j))$
- Connection between the Laplacian and the adjacency matrices:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

- The degree matrix: $\mathbf{D} := D_{ii} = d(v_i)$.

$$\mathbf{L} = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$



Undirected Weighted Graphs

- We consider *undirected weighted graphs*: Each edge e_{ij} is weighted by $w_{ij} > 0$.
- The Laplacian as an operator:

$$(\mathbf{L}\boldsymbol{f})(v_i) = \sum_{v_j \sim v_i} w_{ij}(f(v_i) - f(v_j))$$

- As a quadratic form:

$$\boldsymbol{f}^\top \mathbf{L} \boldsymbol{f} = \sum_{e_{ij}} w_{ij}(f(v_i) - f(v_j))^2$$

- $\mathbf{L}$ is symmetric and positive semi-definite.
- $\mathbf{L}$ has n non-negative, real-valued eigenvalues: $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.

← 0 always an eigenvalue

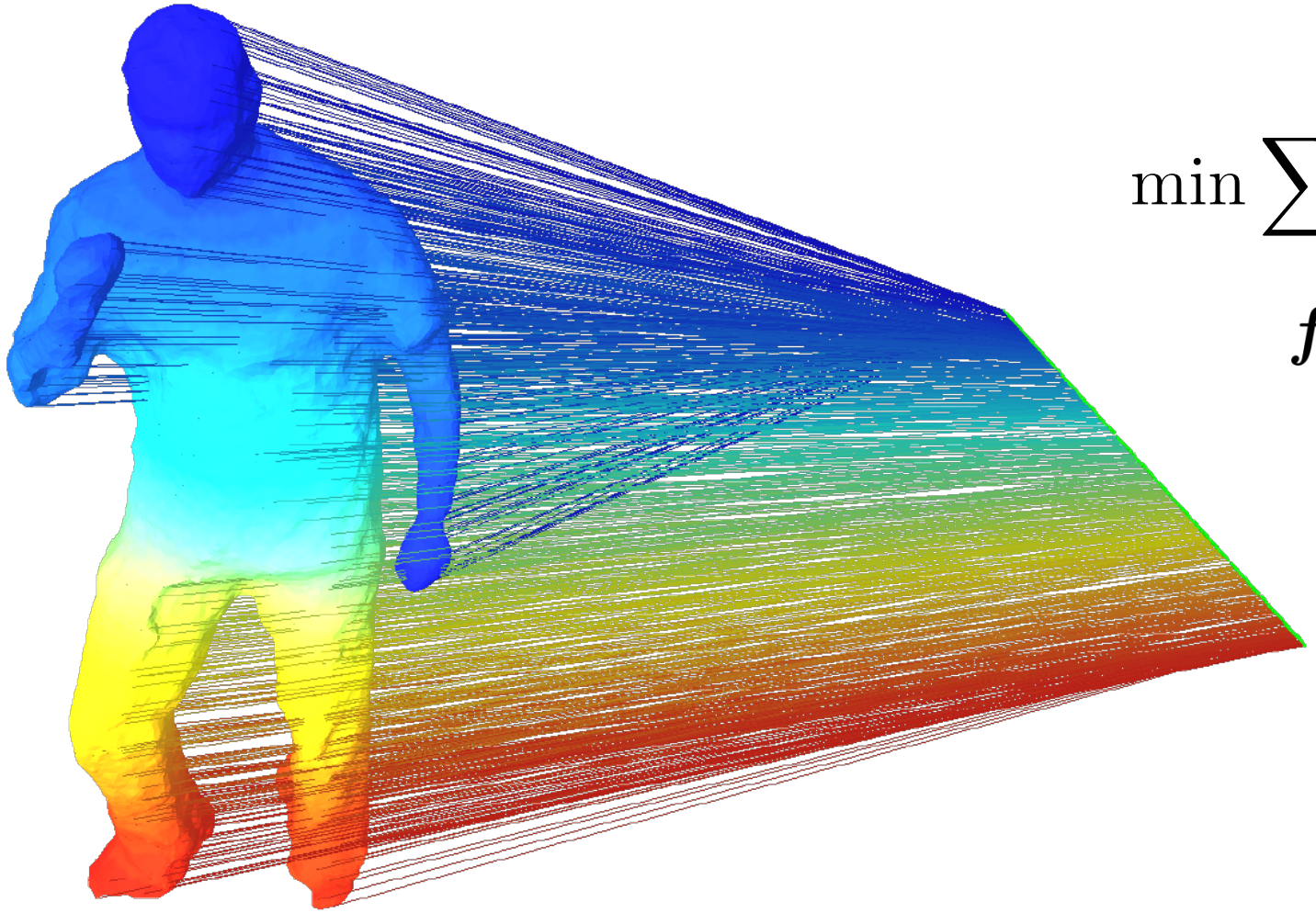
1-D Laplacian Embedding

- Map a weighted graph onto a line such that connected nodes stay as close as possible, i.e., minimize $\sum_{i,j=1}^{n} w_{ij}(f(v_i) - f(v_j))^2$, or:

$$\arg\min_{\boldsymbol{f}} \boldsymbol{f}^\top \mathbf{L}\boldsymbol{f} \text{ with: } \boldsymbol{f}^\top \boldsymbol{f} = 1 \text{ and } \boldsymbol{f}^\top \mathbf{1} = 0$$
$$\boldsymbol{f} \perp \mathbf{1}$$

- The solution is the eigenvector associated with the smallest nonzero eigenvalue of the eigenvalue problem: $\mathbf{L}\boldsymbol{f} = \lambda \boldsymbol{f}$, namely the Fiedler vector $\boldsymbol{u}_2$.
- For more details on this minimization see Golub & Van Loan *Matrix Computations*, chapter 8 (The symmetric eigenvalue problem).

1-D Embedding Example



$$\min \sum w_{ij}(f(v_i) - f(v_j))^2$$

$$\boldsymbol{f}^\top \boldsymbol{f} = 1 \quad \boldsymbol{f} \perp \mathbf{1}$$

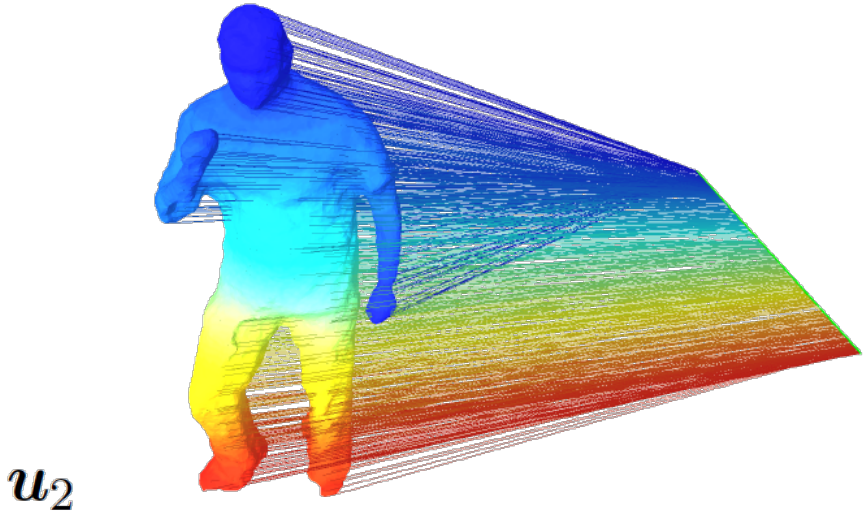
Higher-D Embeddings

- Embed the graph in a k-dimensional Euclidean space. The embedding is given by the $n \times k$ matrix $\mathbf{F} = [\boldsymbol{f}_1 \boldsymbol{f}_2 \dots \boldsymbol{f}_k]$ where the i-th row of this matrix – $\boldsymbol{f}^{(i)}$ – corresponds to the Euclidean coordinates of the i-th graph node v_i.
- We need to minimize (Belkin & Niyogi '03):

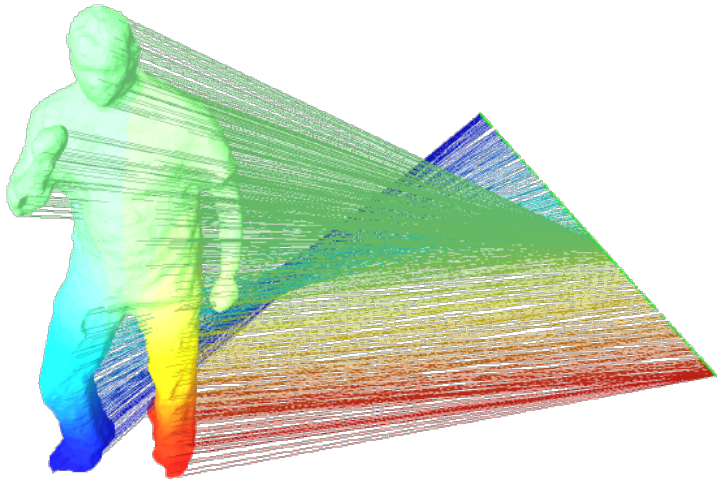
$$\arg \min_{\boldsymbol{f}_1 \cdots \boldsymbol{f}_k} \sum_{i,j=1}^{n} w_{ij} \|\boldsymbol{f}^{(i)} - \boldsymbol{f}^{(j)}\|^2 \text{ with: } \mathbf{F}^\top \mathbf{F} = \mathbf{I}.$$

- The solution is provided by the matrix of eigenvectors corresponding to the k lowest nonzero eigenvalues of the eigenvalue problem $\mathbf{L}\boldsymbol{f} = \lambda \boldsymbol{f}$.

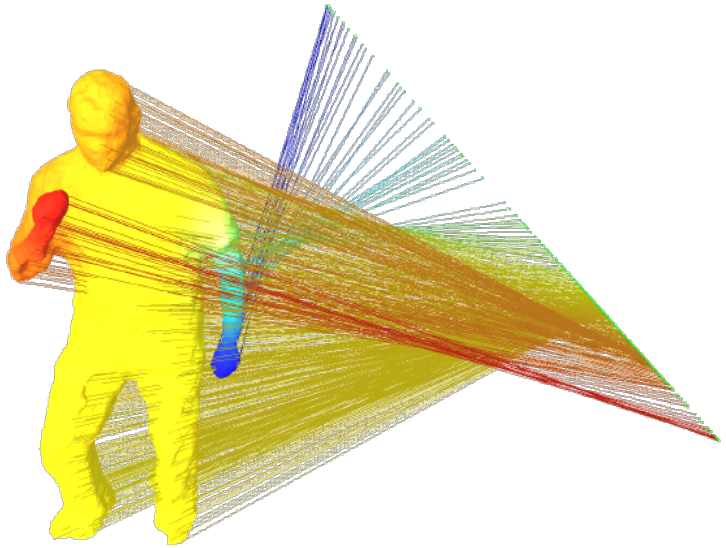
More Eigenvectors, More 1-D Embeddings



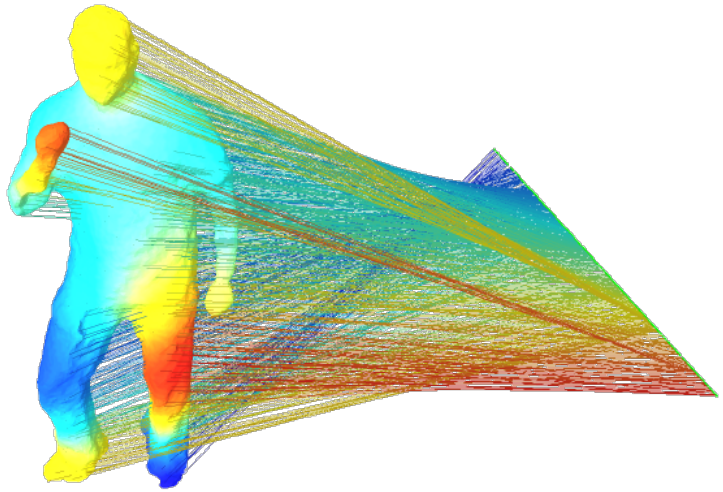
$\boldsymbol{u}_2$



$\boldsymbol{u}_3$



$\boldsymbol{u}_4$



$\boldsymbol{u}_8$

Laplacian Variants

- The normalized graph Laplacian (symmetric and semi-definite positive):

$$\mathbf{L}_n = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

- The transition matrix (allows an analogy with Markov chains):

$$\mathbf{L}_t = \mathbf{D}^{-1}\mathbf{A}$$

- The random-walk graph Laplacian:

$$\mathbf{L}_r = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{L}_t$$

- These matrices are similar:

$$\mathbf{L}_r = \mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}\mathbf{D}^{\frac{1}{2}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}_n\mathbf{D}^{\frac{1}{2}}$$

Random Walk Spectral Embedding

- The $n \times k$ matrix contains the first k eigenvectors of $\mathbf{L}_r$:

$$\mathbf{W} = \begin{bmatrix} \boldsymbol{w}_2 & \dots & \boldsymbol{w}_{k+1} \end{bmatrix}$$

- It is straightforward to obtain the following expressions, where $\boldsymbol{d}$ and $\mathbf{D}$ are the degree-vector and the degree-matrix:

$$\boldsymbol{w}_i^\top \boldsymbol{d} = 0, \ \forall i, 2 \leq i \leq n$$

$$\mathbf{W}^\top \mathbf{D} \mathbf{W} = \mathbf{I}_k$$

- The isometric embedding using the random-walk Laplacian:

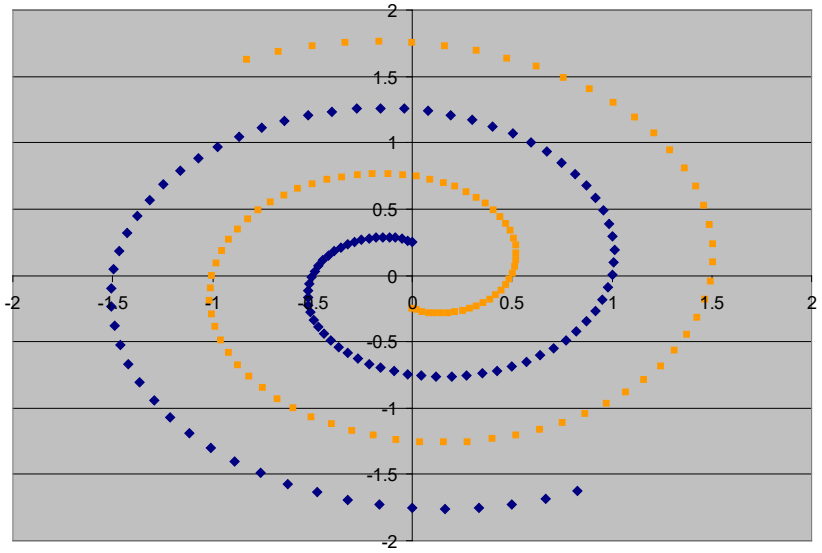
$$\mathbf{Y} = \mathbf{W}^\top = \begin{bmatrix} \boldsymbol{y}_1 & \dots & \boldsymbol{y}_n \end{bmatrix}$$

Spectral Clustering Using the Random-Walk Laplacian

- For details see (von Luxburg '07)
- Input: Laplacian $\mathbf{L}_r$ and the number k of clusters to compute.
- Output: Cluster $C_1, \ldots, C_k$.

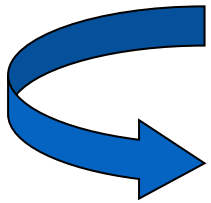
1. Compute $\mathbf{W}$ formed with the first k eigenvectors of the random-walk Laplacian.
2. Determine the spectral embedding $\mathbf{Y} = \mathbf{W}^\top$
3. Cluster the columns $\boldsymbol{y}_j, j = 1, \ldots, n$ into k clusters using the K-means algorithm.

Spirals Again



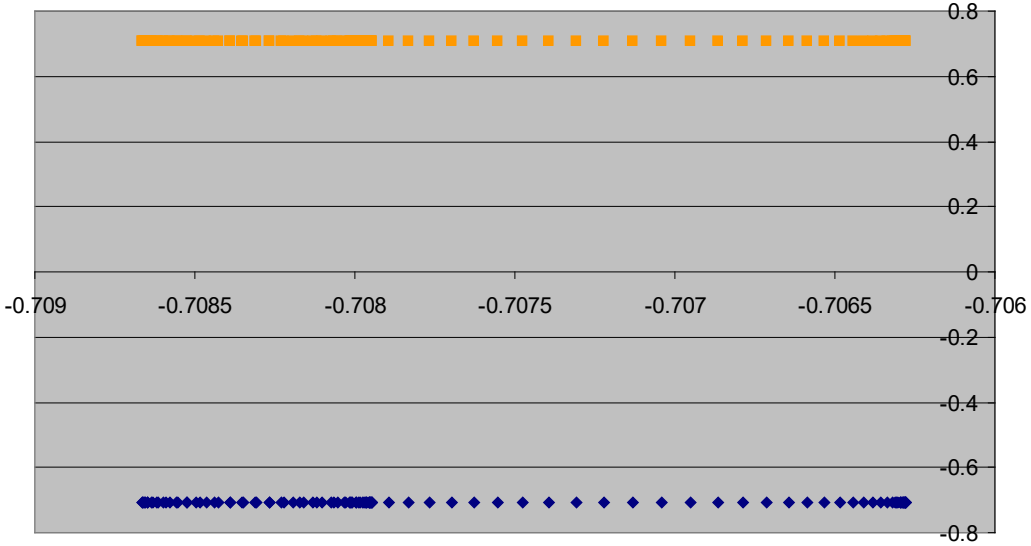
Dataset exhibits complex cluster shapes

$\Rightarrow$ Direct *k*-means performs very poorly in this space due to bias toward dense spherical clusters.



In the embedded space given by two leading eigenvectors, clusters are trivial to separate.

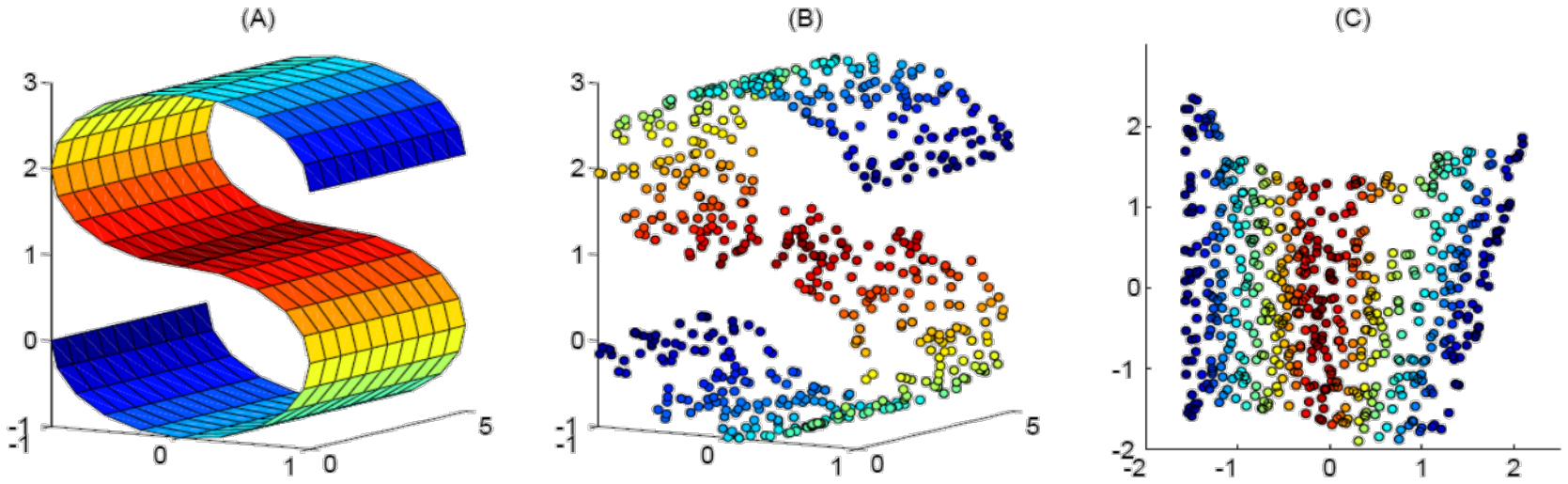
Build nearest neighbor graph



Non-Linear Dimensionality Reduction

Non-Linear Dimensionality Reduction

- Why do we need NLDR?
- Many data sets contain essential non-linear structures that “invisible” to PCA and MDS
- Must resort to some non-linear dimensionality reduction approaches



Data sampled from a non-linear manifold

The Choice of Distance

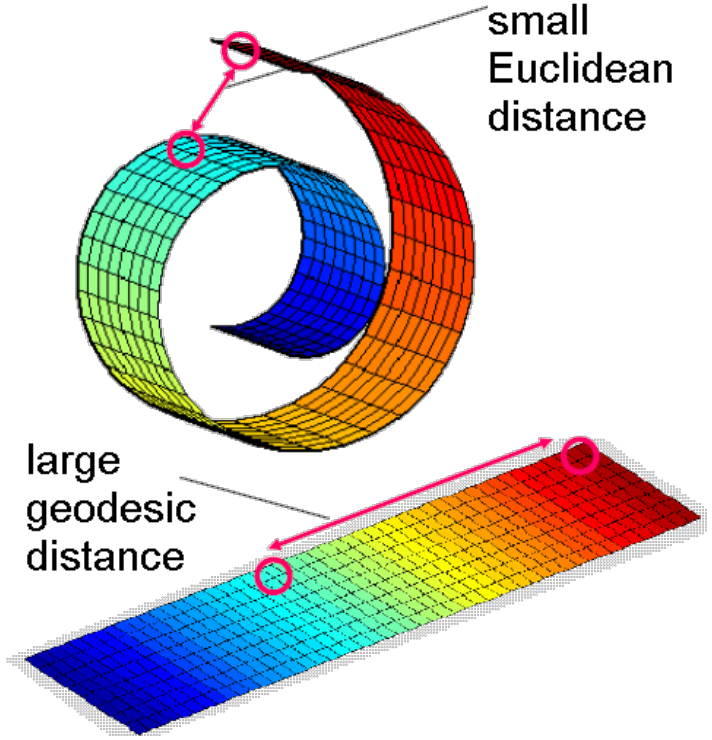
- We can try to capture the manifold structure through the right notion of distance directly on the manifold (geodesic distance)
- That's why PCA and MDS fail – they only see the Euclidean structures

question @16 stop following 12 views Actions

Question 1e Nearest Neighbor Search

For nearest neighbor search, is there a specified distance metric we should use? My instinct says L2 norm for rotational invariance and since PCA deals with variances, but I'm not 100% sure what the canonical choice is for PCA embeddings.

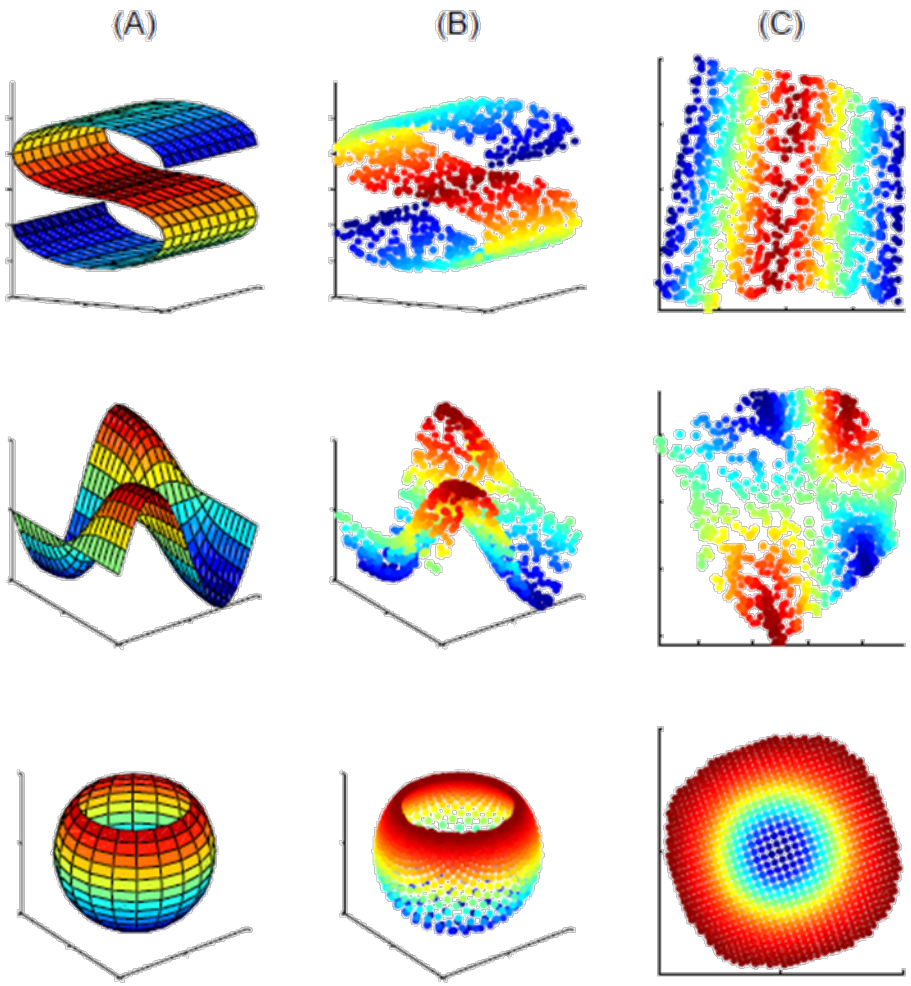
hw1



The Challenge of NLDR

- An unsupervised learning algorithm – it must discover the global internal coordinates of the manifold, without external signals that suggest how the data should be embedded in low dimensions

- Build a bridge between high- & low-dimensional spaces

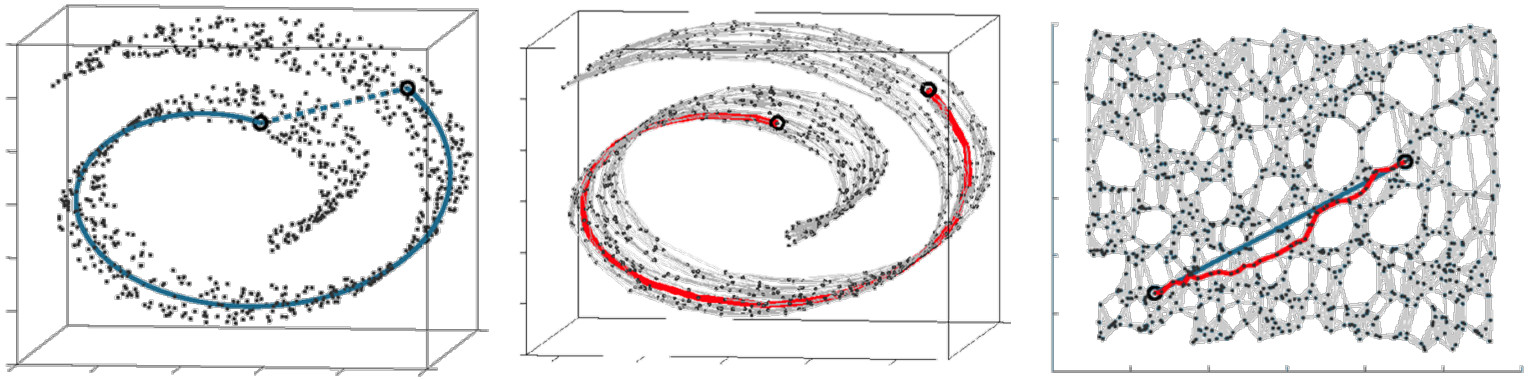


Isomap

ISOMAP (J. B. Tenenbaum, V. de Silva and J. C. Langford)

- Example of non-linear structure (Swiss roll)
  - Only the geodesic distances reflect the true low-dimensional geometry of the manifold

- ISOMAP (Isometric Feature Mapping)
  - Uses the geodesic manifold distances between all pairs
  - Preserves the intrinsic geometry of the data -- Preserves the geodesic distances
  - How to estimate geodesic distances between point pairs?



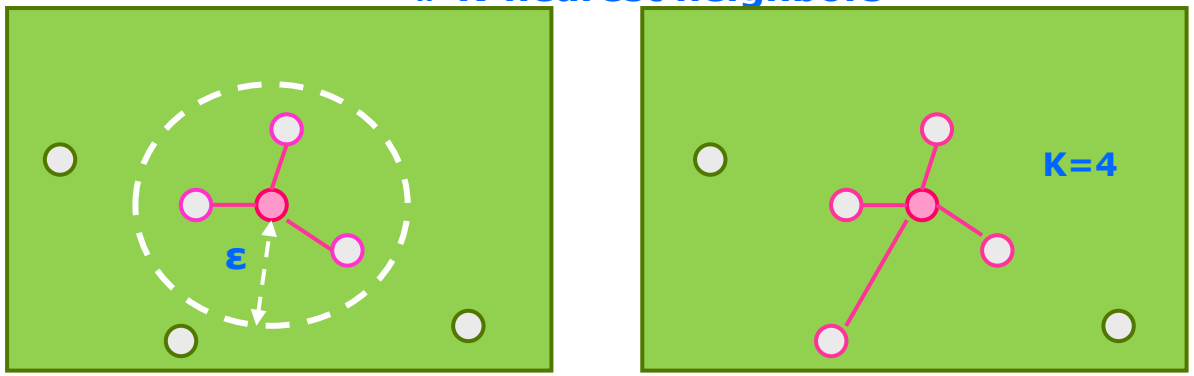
ISOMAP (Algorithm Description)

- **Step 1**
  - Form a near-neighbor graph G on the original data points, weighing the edges based on their original distances $d_X(i,j)$.
- **Step 2**
  - Estimate the geodesic distances $d_G(i,j)$ between all pairs of points on the sampled manifold by computing their shortest path distances in the graph G.
- **Step 3**
  - Construct an embedding of the data in *d*-dimensional Euclidean space *Y* that best preserves the distances (MDS).

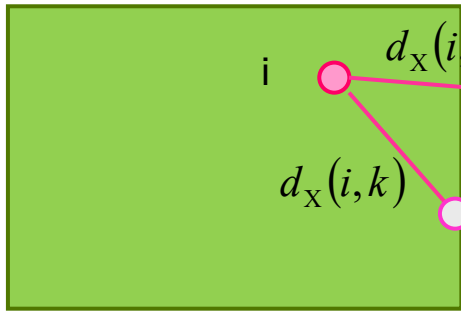
Near Neighbor Graph

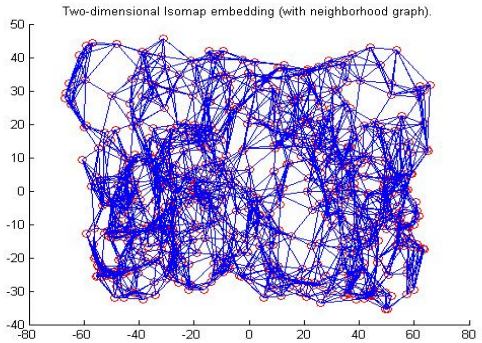
- **Step 1**
  - Determining neighboring points within a fixed radius based on the input space distance $d_X(i,j)$, or use a fixed # of neighbors

 # ε-radius **# K-nearest neighbors**



  - These neighborhood relations are represented as a weighted graph G over the data points.



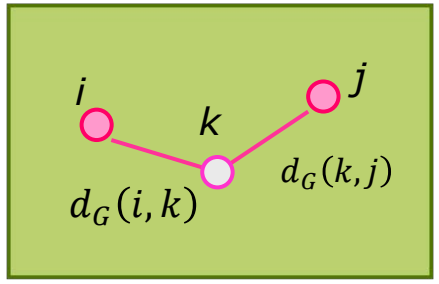


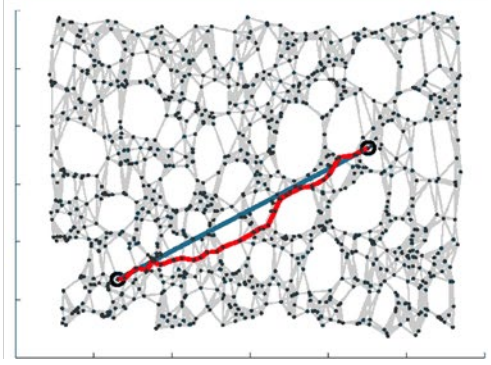
Shortest Path Computation

- **Step 2**
  - Estimating the geodesic distances $d_G(i,j)$ between all pairs of points on the manifold by computing their shortest path distances in the graph G.
  - Can be done using classic graph algorithms for the All Pairs Shortest Path (APSP) problem: Floyd/Warshall's algorithm or Dijkstra's algorithm

$$d_G(i,j) = d_X(i,j) \text{ neighboring i,j}$$
$$d_G(i,j) = \infty \quad \text{othewise}$$

$$for\ \mathrm{k} = 1,2,\ldots,\mathrm{N}$$
$$d_G(i,j)$$
$$= \min\{d_G(i,j),\ d_G(i,k) + d_G(k,j)\}$$

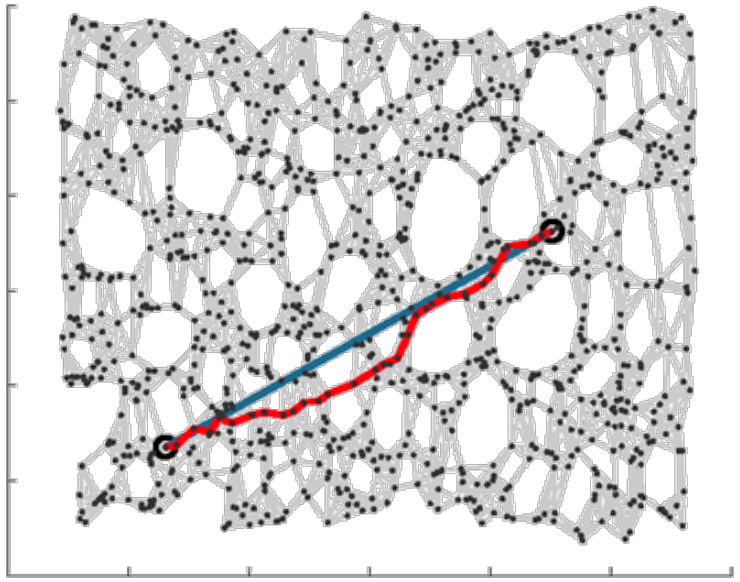




Euclidean Embedding

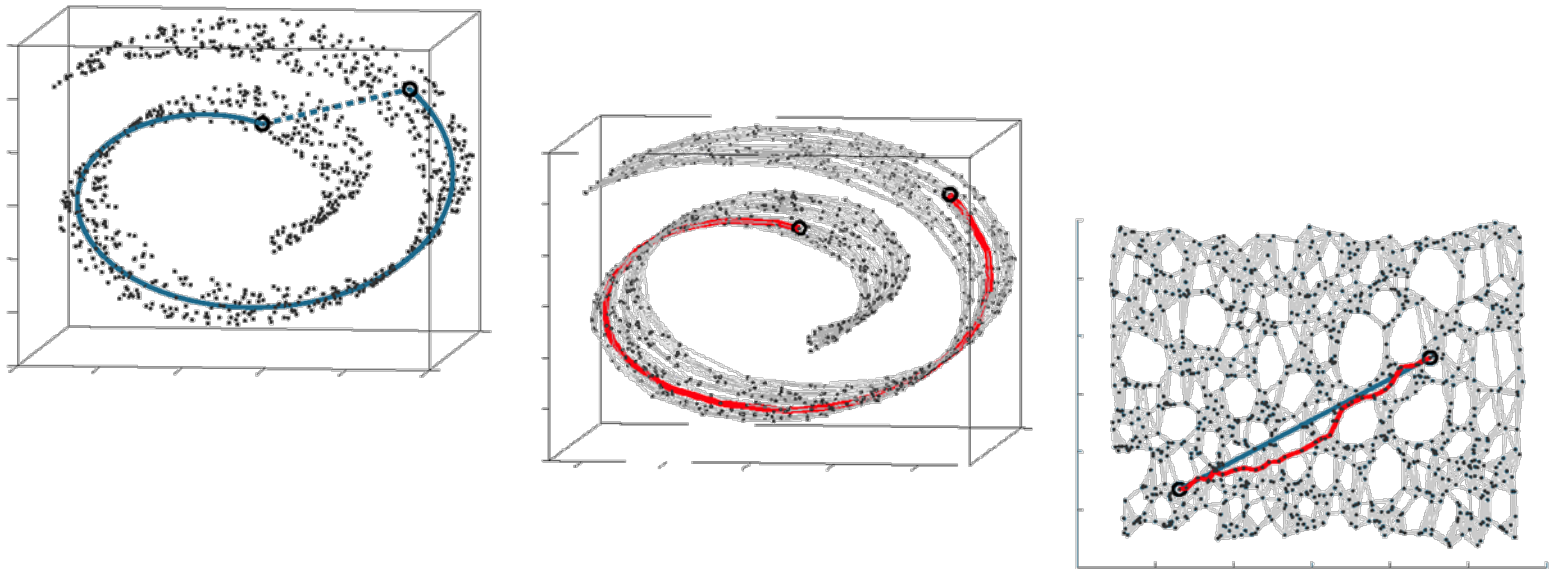
- **Step 3**
  - Constructing an embedding of the data in *d*-dimensional Euclidean space *Y* that best preserves the inter-point distances

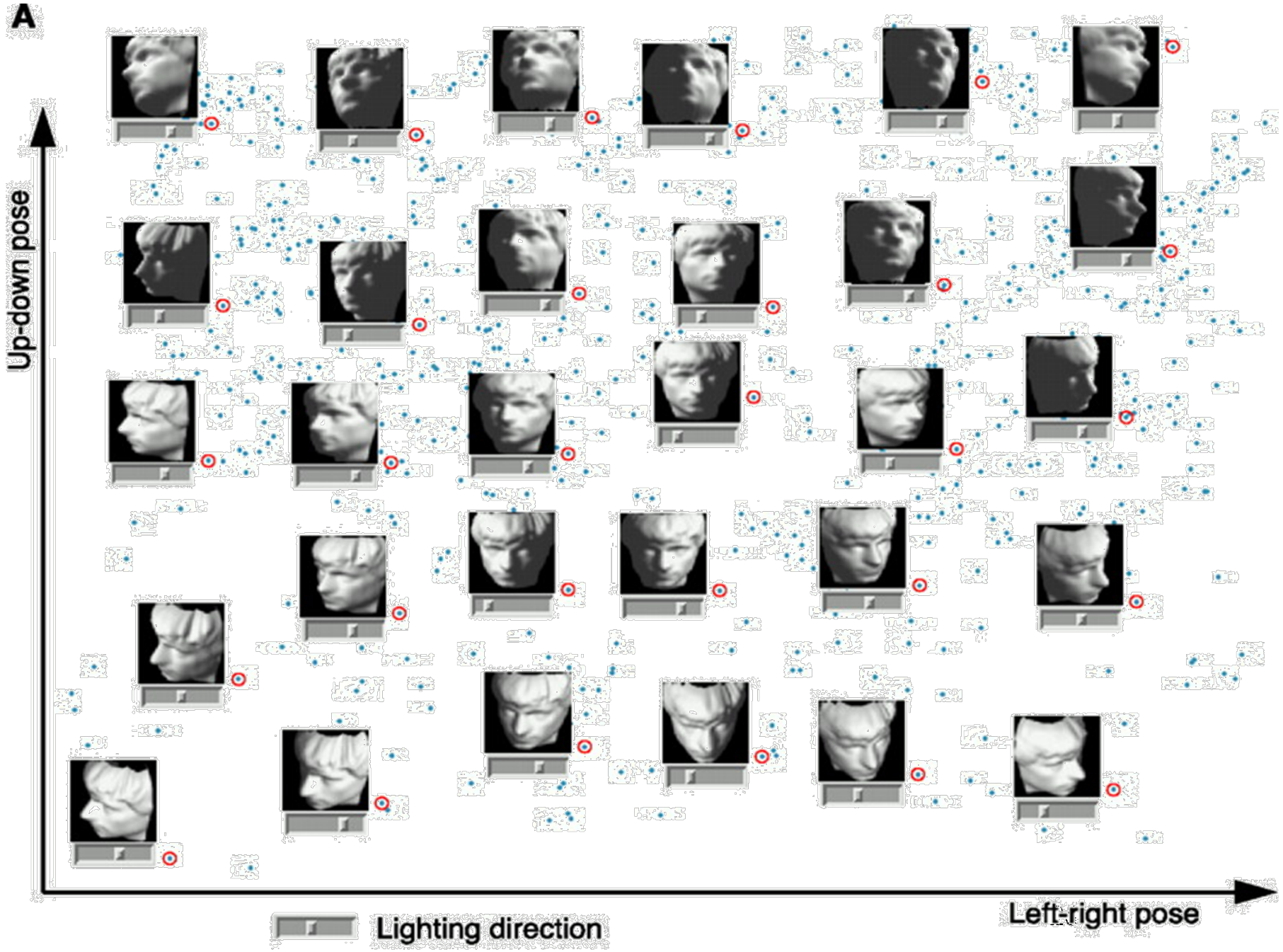
- This is of course nothing but an MDS problem



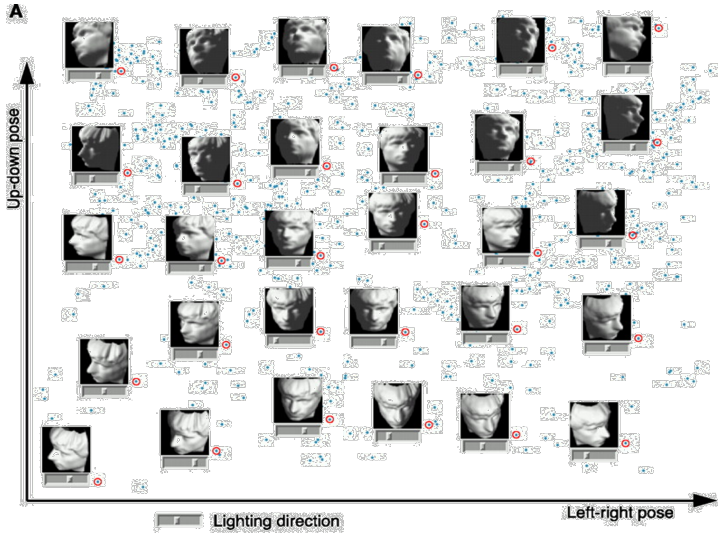
Recovery Guarantees

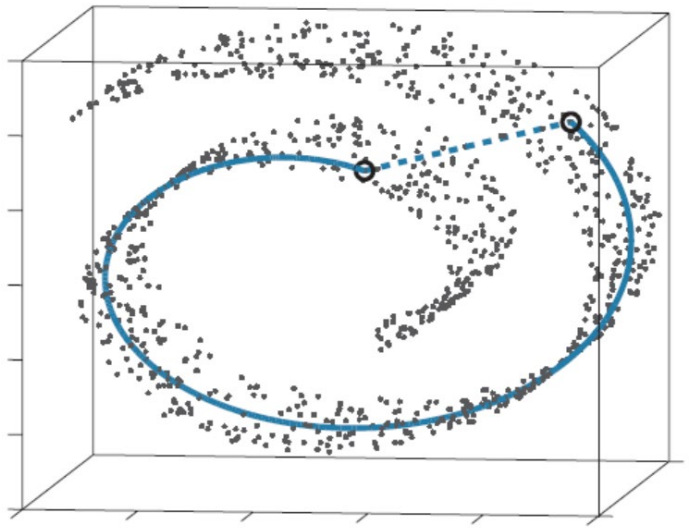
- Isomap is guaranteed asymptotically to recover the true dimensionality and geometric structure of non-linear manifolds.
- As the sample data point density increases, the graph distances provide increasingly better approximations to the intrinsic geodesic distances.

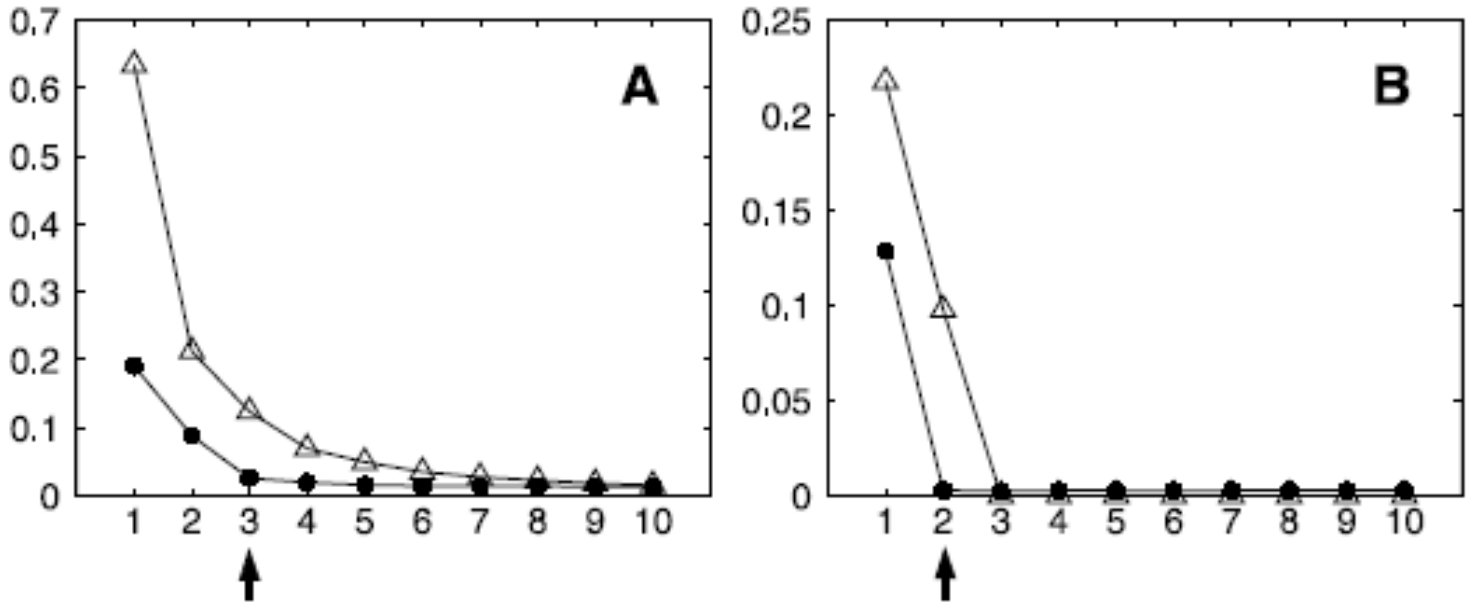



A
Up-down pose
Left-right pose
Lighting direction

ISOMAP Examples







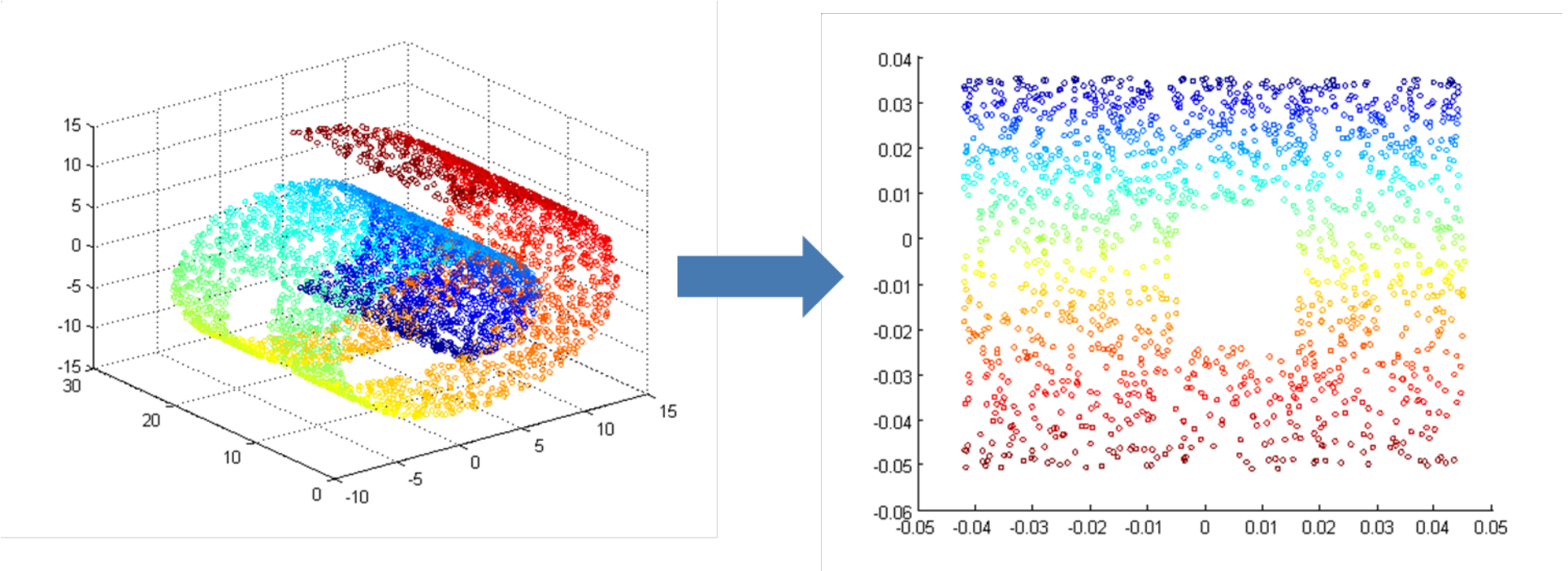
MDS : open triangles

Isomap : filled circles

Laplacian Eigenmaps

Laplacian Eigenmaps (M. Belkin, P. Niyogi)

- Start same as Isomap, but use a spectral embedding in lieu of MDS



Hole distorts long geodesic distances, but affects less diffusion distances

Locally Linear Embeddings

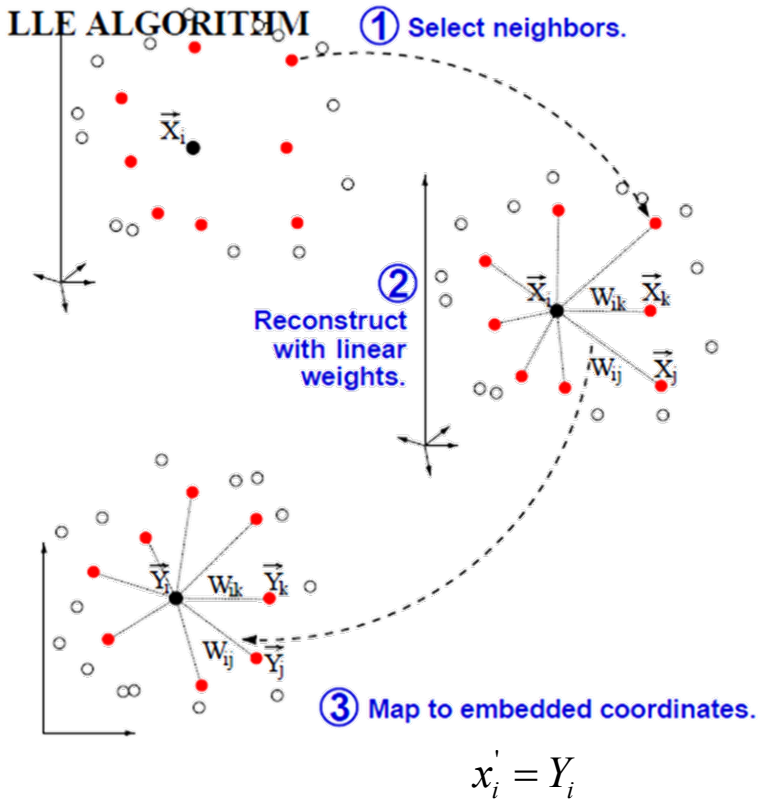
Locally Linear Embeddings (LLE) (S. T. Roweis and L. K. Saul)

- Define neighborhood relations between points (build NN graph)
  - *k* nearest neighbors
  - ε-balls
- Find weights that reconstruct each data point from its neighbors:

$$\min_{\sum_j w_{ij}=1} \left\| \mathbf{x}_i - \sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right\|^2$$

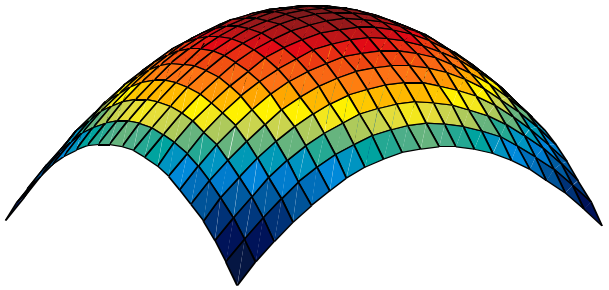
- Find low-dimensional coordinates so that the same weights hold: $\mathbf{x}_1', \ldots, \mathbf{x}_n' \in R^d$

$$\min_{\mathbf{x}_1', \ldots, \mathbf{x}_n'} \sum_i \left\| \mathbf{x}_i' - \sum_{j \in N(i)} w_{ij} \mathbf{x}_j' \right\|^2$$



From Local to Global

- The weights w_{ij} capture the local shape
  - Invariant to translation, rotation and scale of the neighborhood
  - If the neighborhood lies on a manifold, the *local* mapping from the global coordinates (R^D) to the surface coordinates (R^d) is almost linear
  - Thus, the weights w_{ij} should hold also for manifold (R^d) coordinate system!



$$\min_{\sum_j w_{ij}=1} \left\| \mathbf{x_i} - \sum_{j \in N(i)} w_{ij} \mathbf{x_j} \right\|^2$$

$$\min_{\mathbf{x_1}', \ldots, \mathbf{x_n}'} \sum_i \left\| \mathbf{x_i}' - \sum_{j \in N(i)} w_{ij} \mathbf{x_j}' \right\|^2$$

Solving the Minimizations

- Linear least squares (using Lagrange multipliers)

$$\min_{\sum_j w_{ij}=1} \left\| \mathbf{x_i} - \sum_{j \in N(i)} w_{ij} \mathbf{x_j} \right\|^2$$

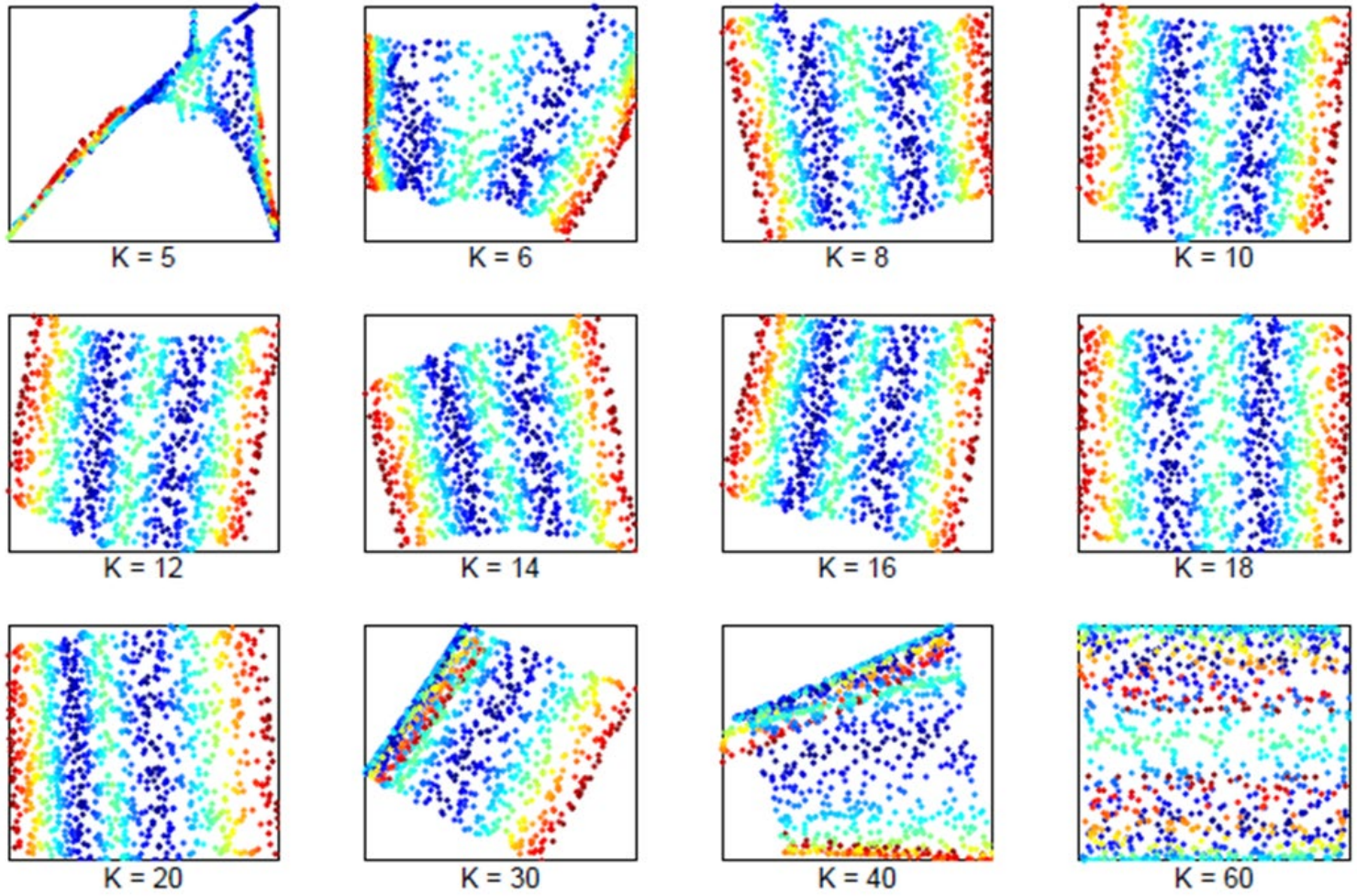
- To find $\mathbf{x_1}', \ldots, \mathbf{x_n}' \in R^d$ that minimize,

$$\min_{\mathbf{x_1}', \ldots, \mathbf{x_n}'} \sum_i \left\| \mathbf{x_i}' - \sum_{j \in N(i)} w_{ij} \mathbf{x_j}' \right\|^2$$

a sparse eigenvalue problem is solved. Additional constraints are added for conditioning:

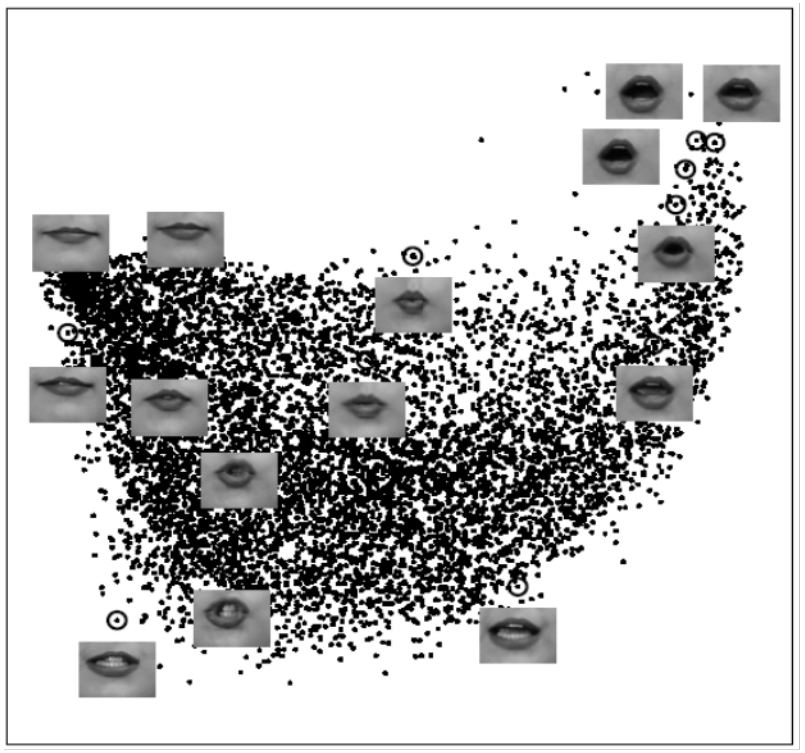
$$\sum_i \mathbf{x_i}' = 0, \qquad \frac{1}{n} \sum_i \mathbf{x_i}' \mathbf{x_i}'^T = I$$

Effect of Neighborhood Size on LLE

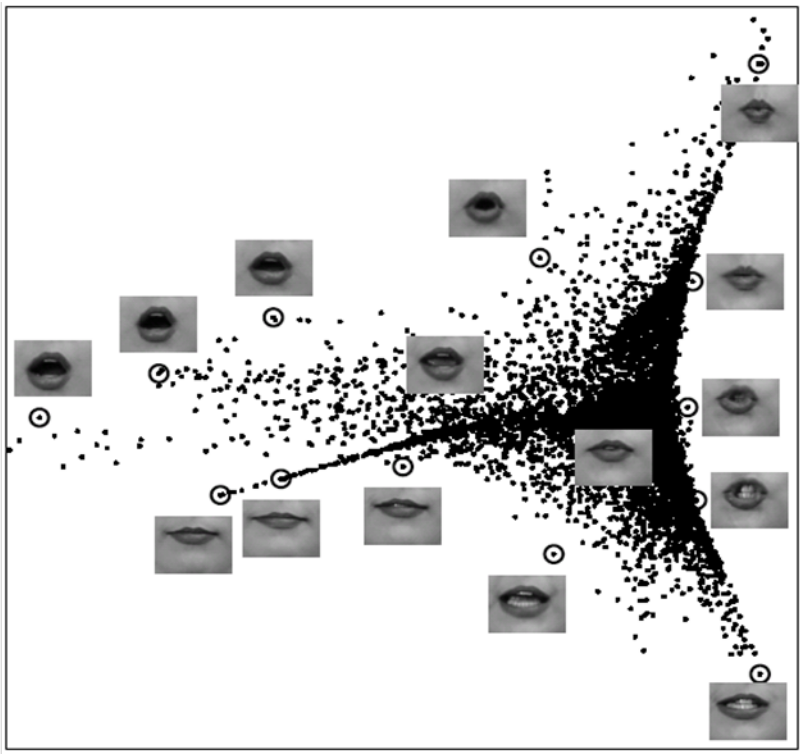


LLE Experiments

- Lips

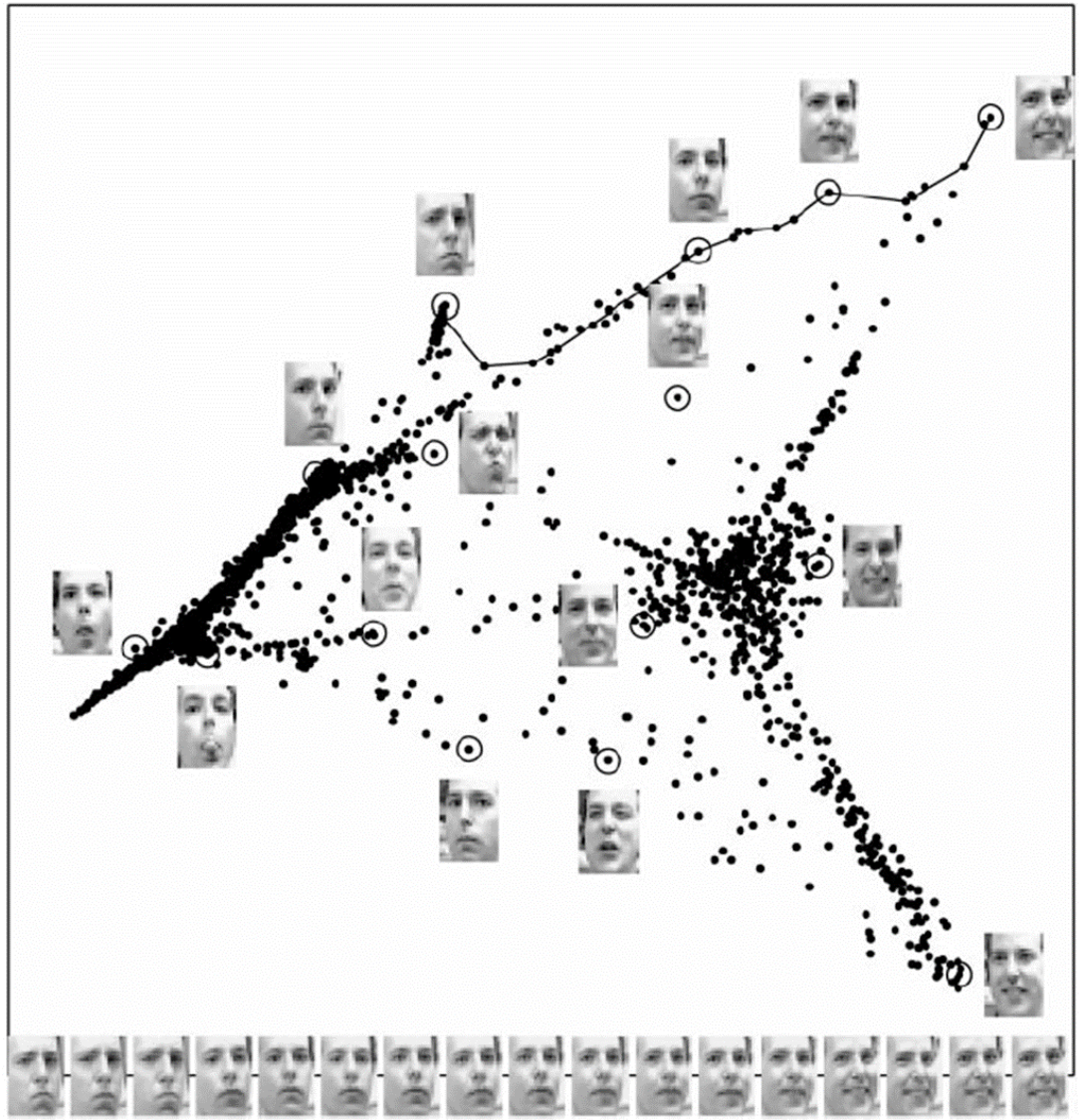


PCA



LLE

LLE Experiments: Faces

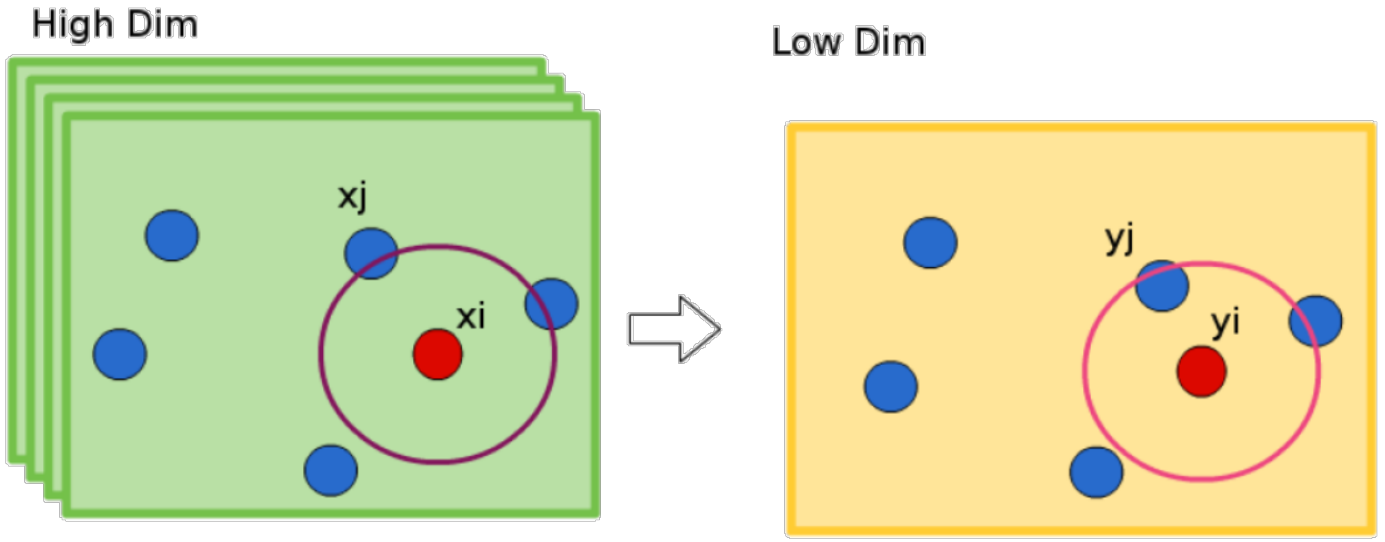


t-SNE

Laurens van der Maaten and Geoffrey Hinton, JMLR 2008

t-Distributed Stochastic Neighbor Embedding

Measure pairwise similarities between high-dimensional and low-dimensonal objects



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

Stochastic Neighbor Embedding

Converting the high-dimensional Euclidean distances into conditional probabilities that represent similarities

- Similarity of datapoints in High Dimension

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

- Similarity of datapoints in Low Dimension

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

- Cost function

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

Minimize the cost function using gradient descent

Symmetric SNE

- Minimize a single KL divergence between a joint probability distribution

$$C = KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} log \frac{p_{ij}}{q_{ij}}$$

- The obvious way to redefine the pairwise similarities is

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

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

However, in practice we symmetrize (or average) the conditionals

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

Set the bandwidth σ_i such that the conditional has a fixed perplexity (effective number of neighbors) $Perp(P_i) = 2^{H(P_i)}$, typical value is about 5 to 50

t-Distribution

Use heavier tail distribution than Gaussian in low-dim space, we choose

$$q_{ij} \propto (1 + ||y_i - y_j||^2)^{-1}$$

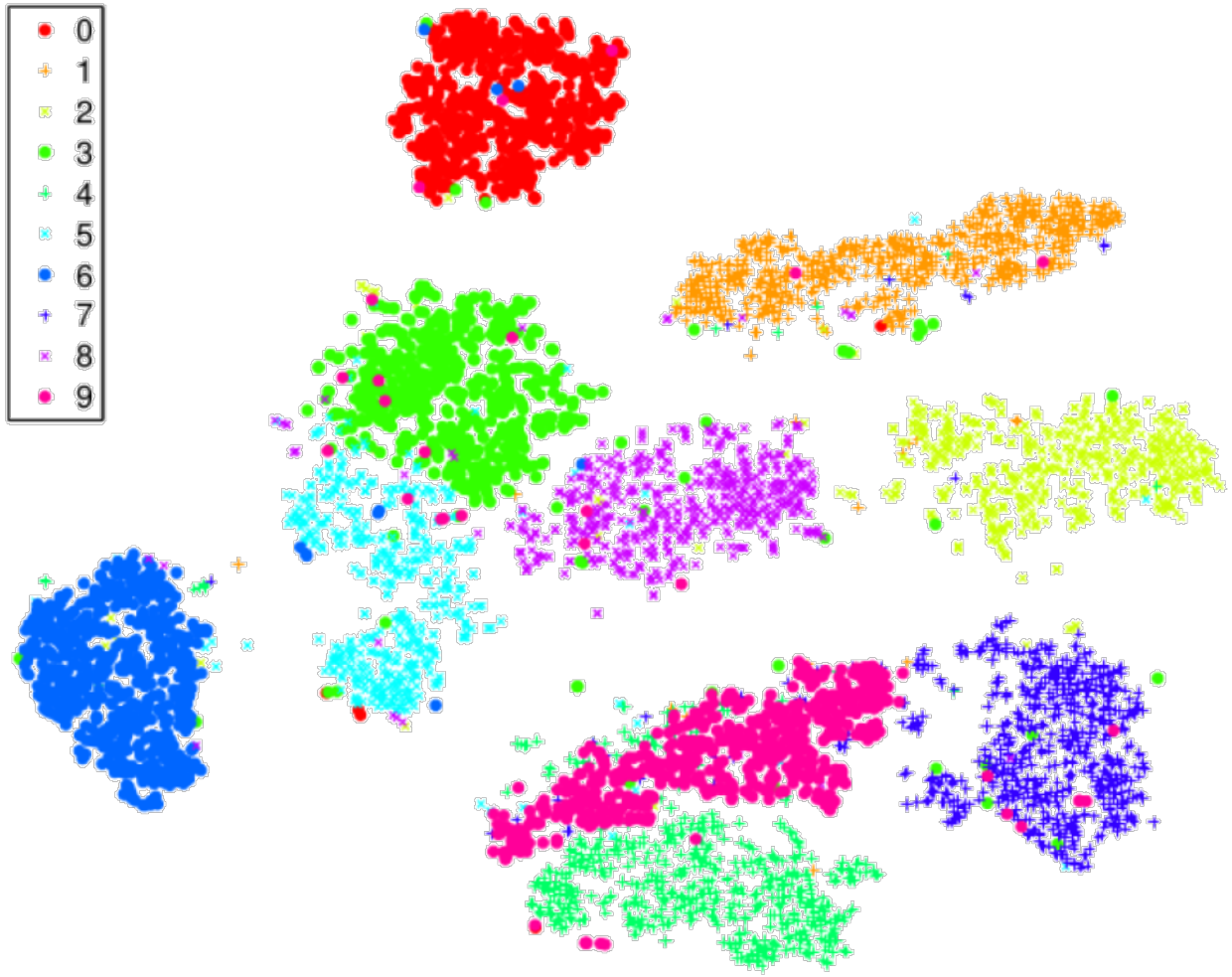
- Similarity of datapoints in High Dimension

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

- Similarity of datapoints in Low Dimension

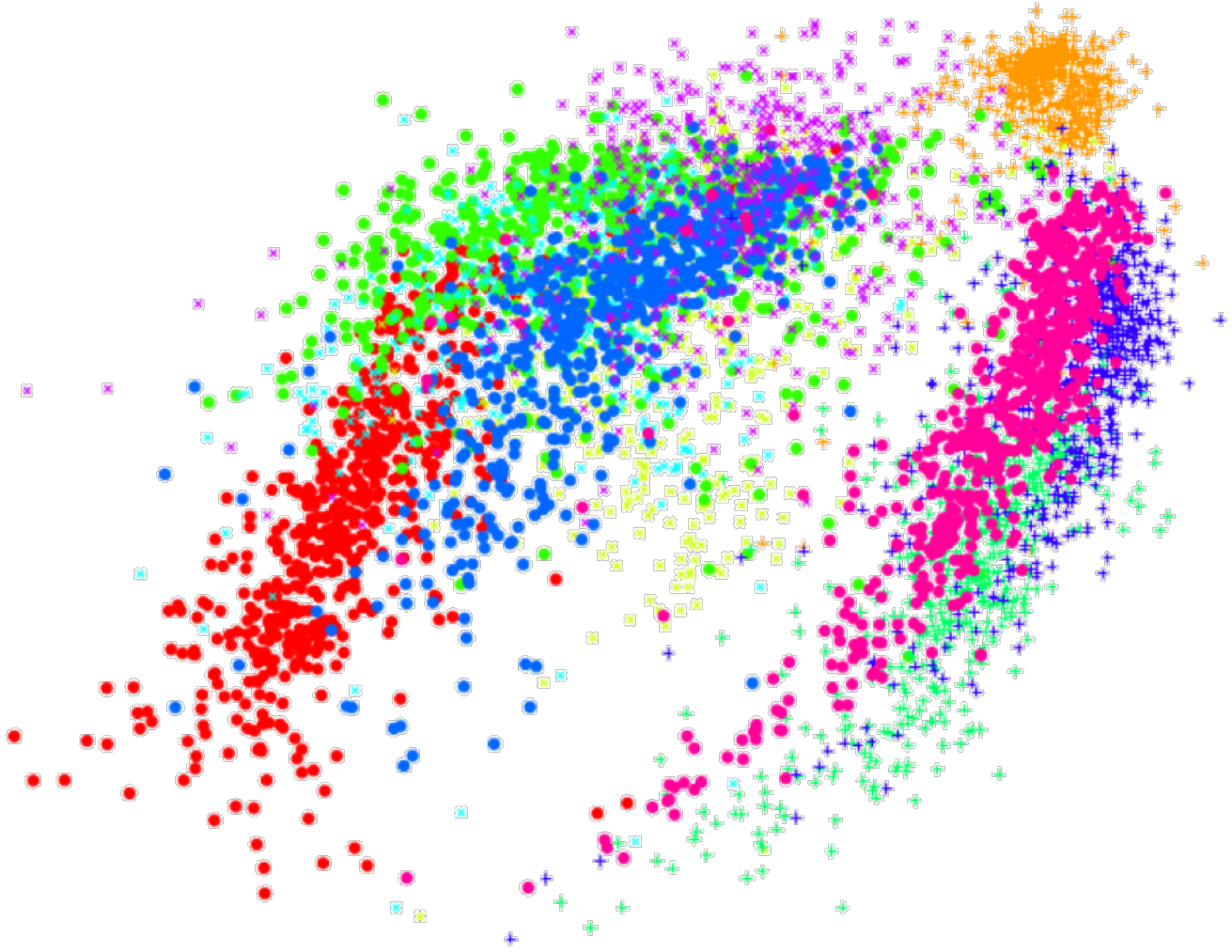
$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

MNIST t-SNE

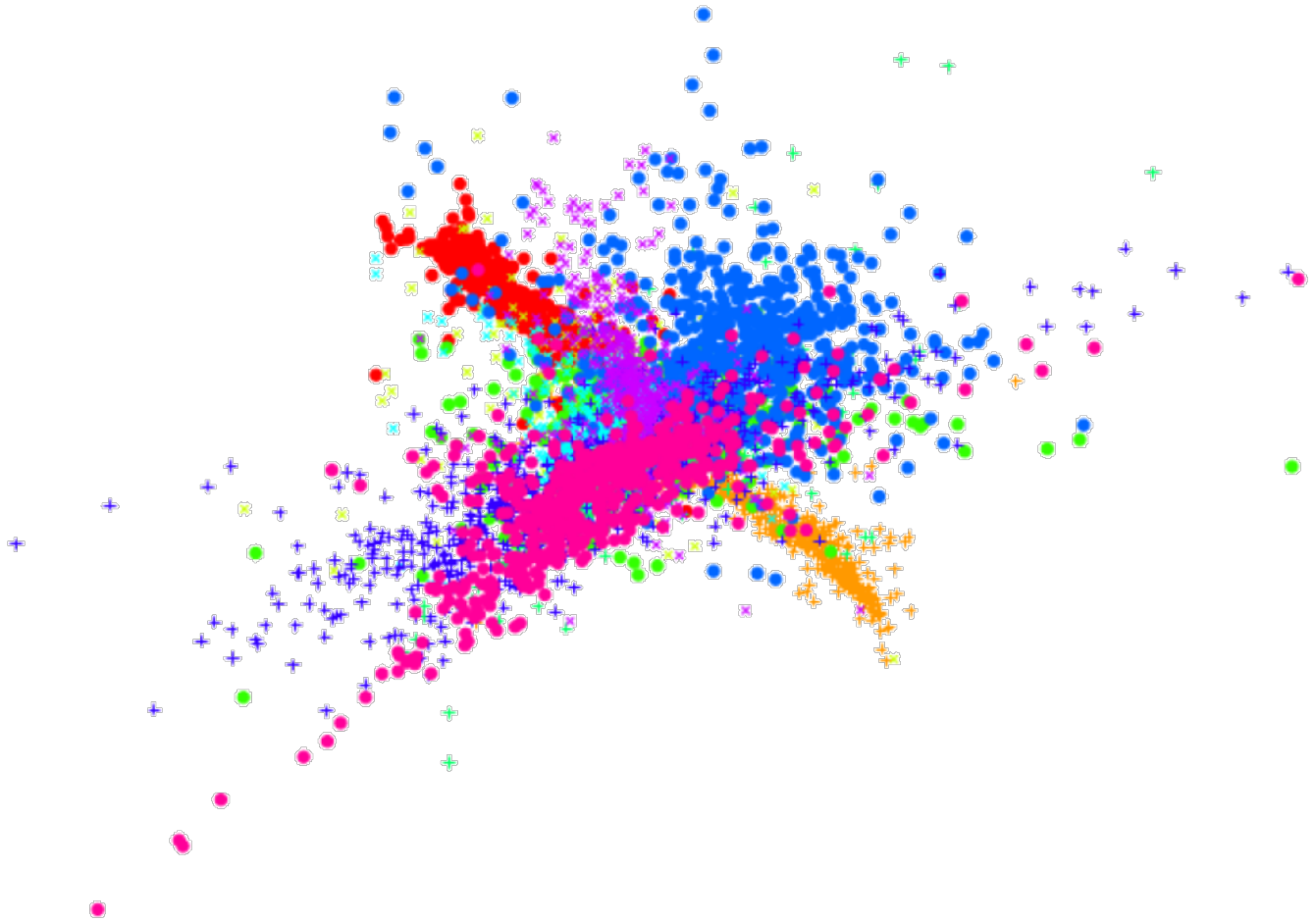


MNIST MDS w. Sammon Mapping

MNIST Isomap



MNIST LLE



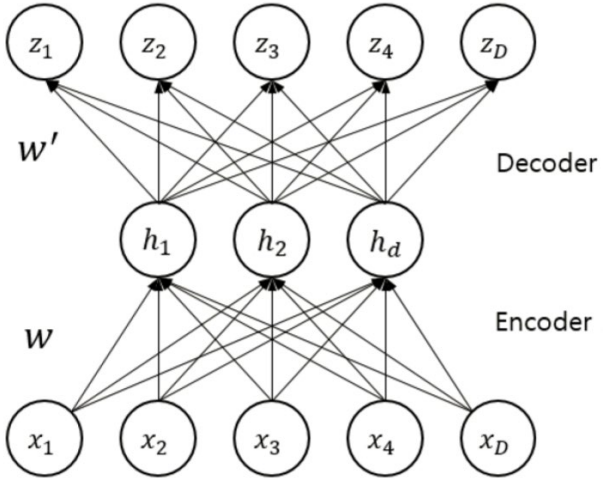
Source Codes

- t-SNE (Matlab, CUDA, Binary, Python, Torch, Julia, R and JavaScript)
- Parametric t-SNE (Matlab)
- Barnes-Hut-SNE (with C++, Matlab, Python, Torch, and R wrappers)

Autoencoder

Autoencoder

- An autoencoder is a feedforward neural network trained to reproduce its input at the output layer



- Encoder:

$$\mathbf{h} = f_\theta(\mathbf{x}) = a(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Decoder:

$$\mathbf{z} = g_{\theta'}(\mathbf{h}) = a(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

$$\theta, \theta' : \{\mathbf{W}, \mathbf{b}\}, \{\mathbf{W}', \mathbf{b}'\}$$

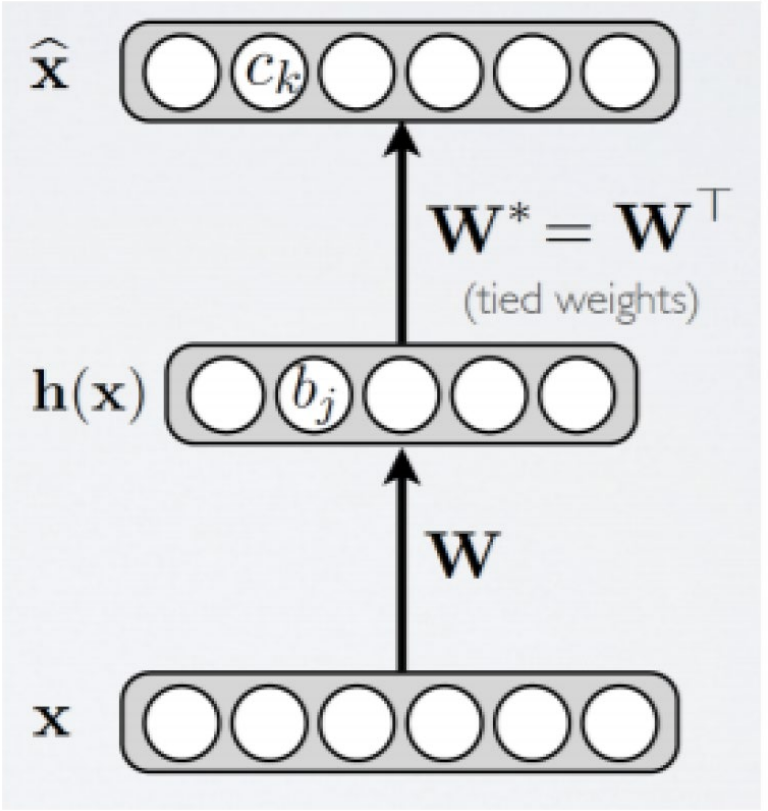
- Parameter estimation
- Back-propagation

$$\theta^*, \theta'^* = \underset{\theta,\theta'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}\right)$$

$$= \underset{\theta,\theta'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L\left(\mathbf{x}^{(i)}, g_{\theta'}(f_\theta(\mathbf{x}^{(i)}))\right)$$

Tied Weights

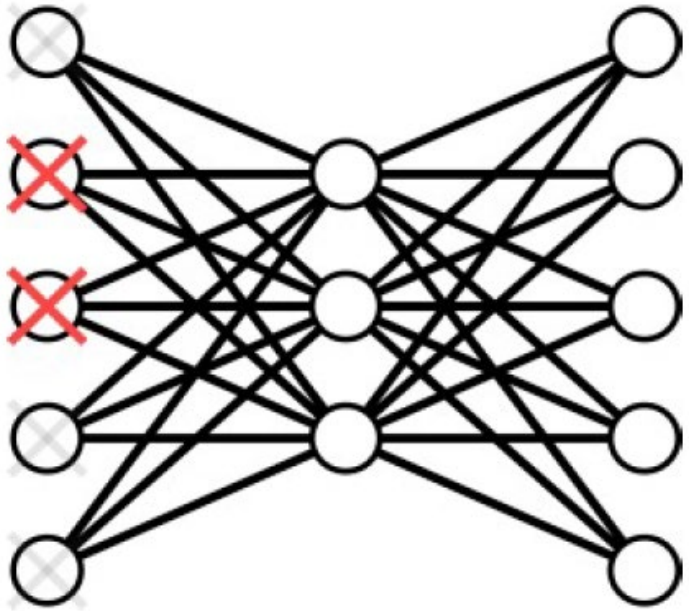
- Optionally, the reverse mapping W' may be constrained to be the transpose of the forward mapping, referred to as tied weights
  - Easy to learn
  - In linear case it approximates PCA
  - Tied weights are sort of regularization



Denoising Autoencoder

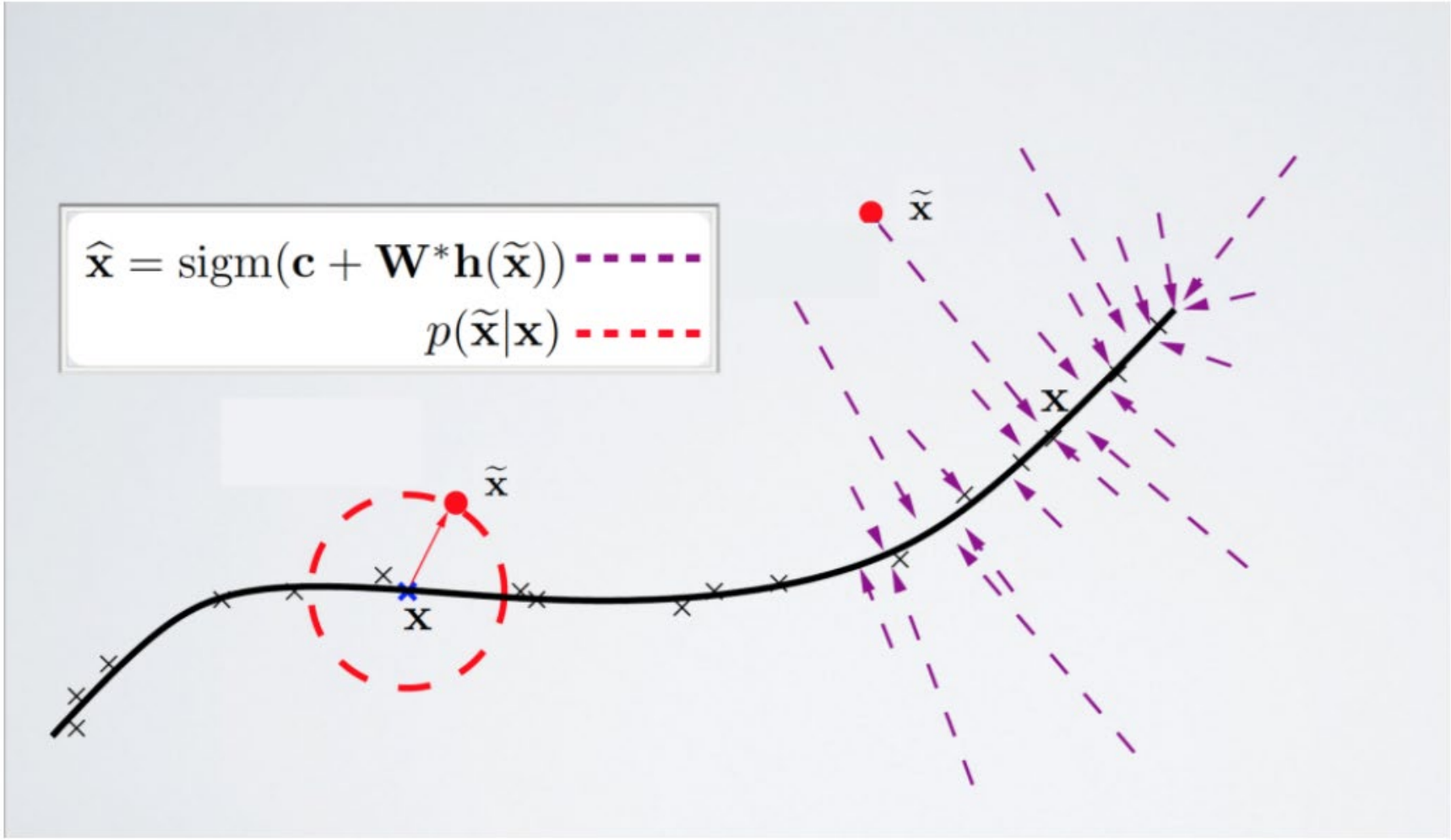
- Should be robust to the introduction of noise
  - Random assignment of inputs to 0, with probability
  - Gaussian noise

$$\mathbf{y} = f_\theta(\tilde{\mathbf{x}}) = f(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b})$$



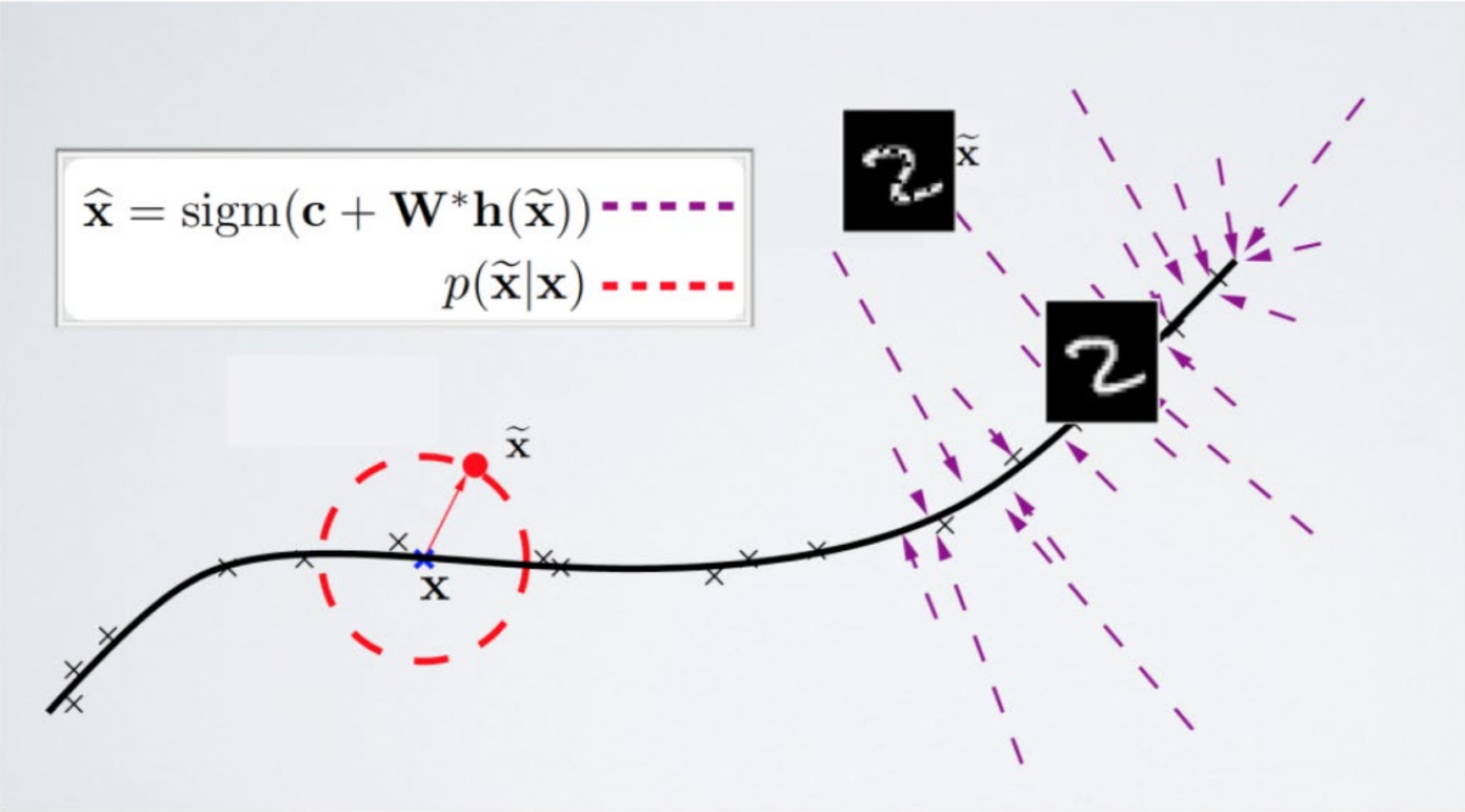
Denoising Autoencoder

- Manifold learning perspective



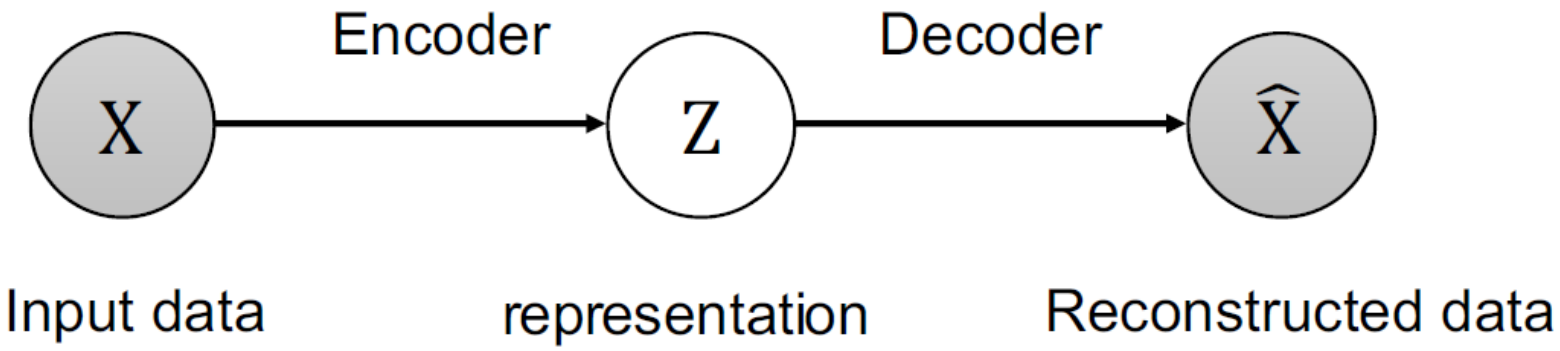
Denoising Autoencoder

- Manifold learning perspective



Variational Autoencoder

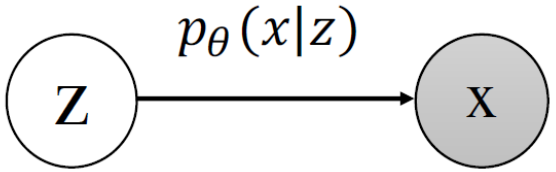
- Autoencoders can reconstruct data from the learned features
- Can we generate new data from an Autoencoder?



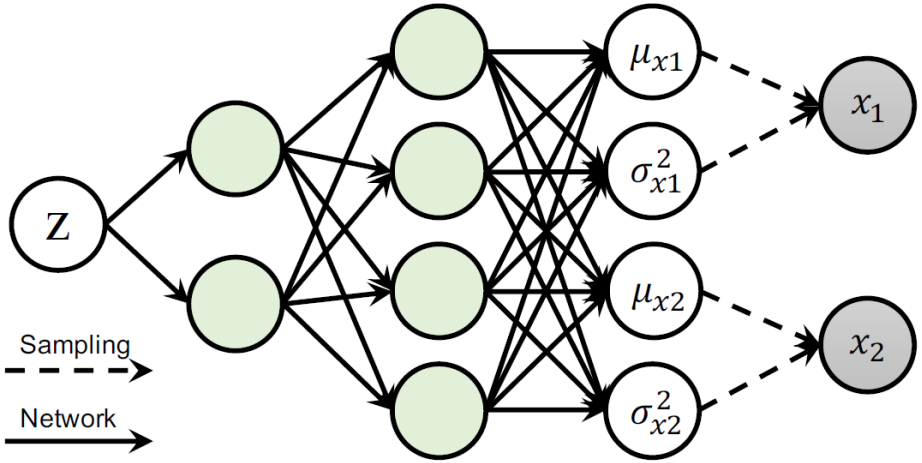
Variational Autoencoder

- Probabilistic spin on Autoencoder – will let us sample from the model to generate data
- Assume training data is generated from underlying latent (unobserved) representation

$z \sim p(z)$



- We wish to estimate the true parameters θ from the training data x
- Modeling the conditional distribution by a neural network (decoder)

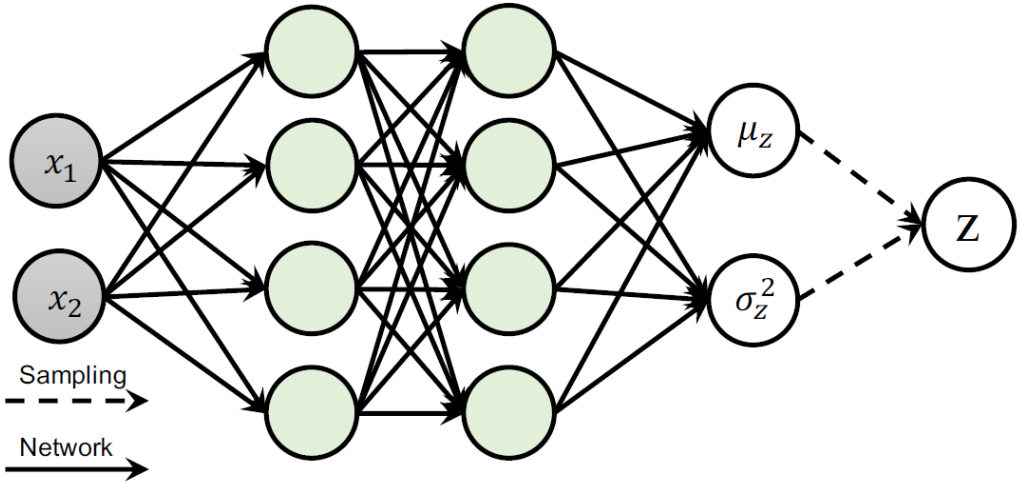


Variational Autoencoder

- For generative model, the training objective is to maximize likelihood of training data:

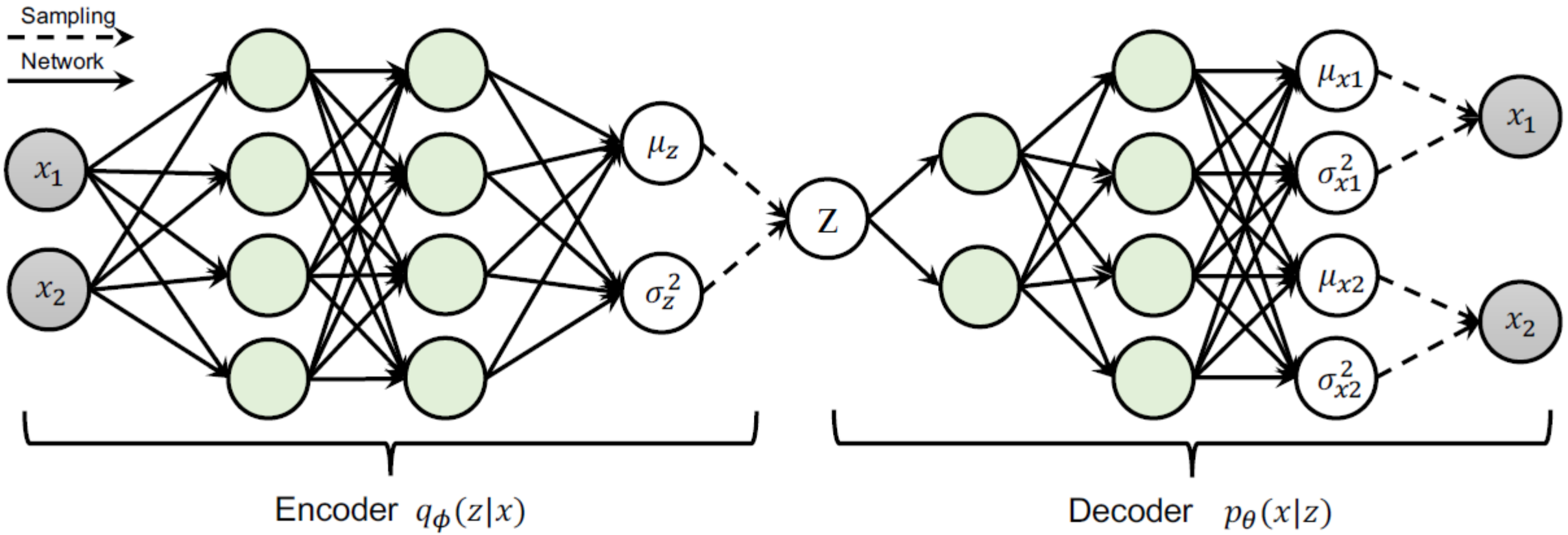
$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- Problem:
  - Intractable to compute p(x|z) for every z
  - The posterior distribution p(z|x) is also intractable
- Solution: In addition to decoder network, define an additional encoder network that approximates p(z|x)

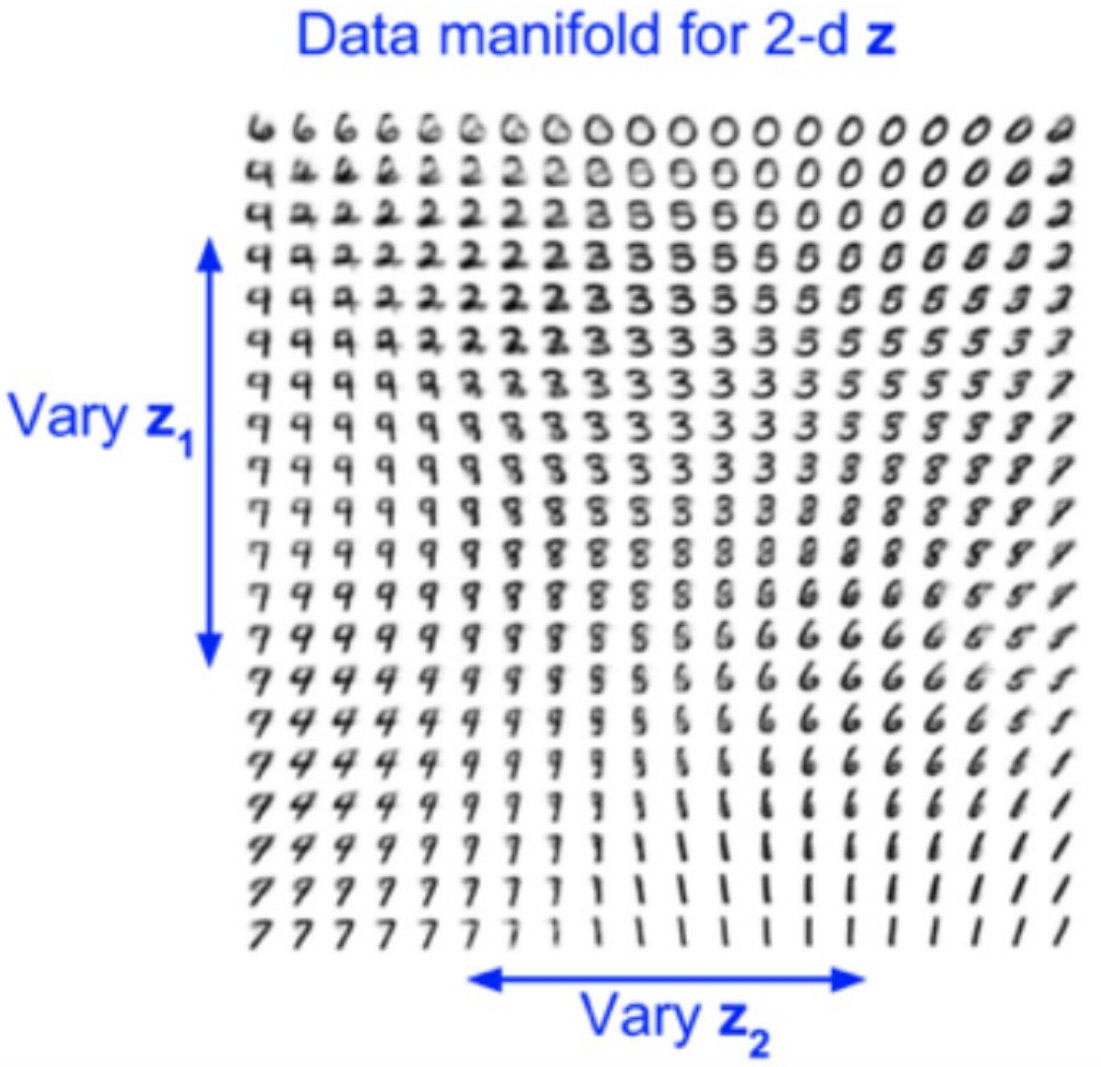


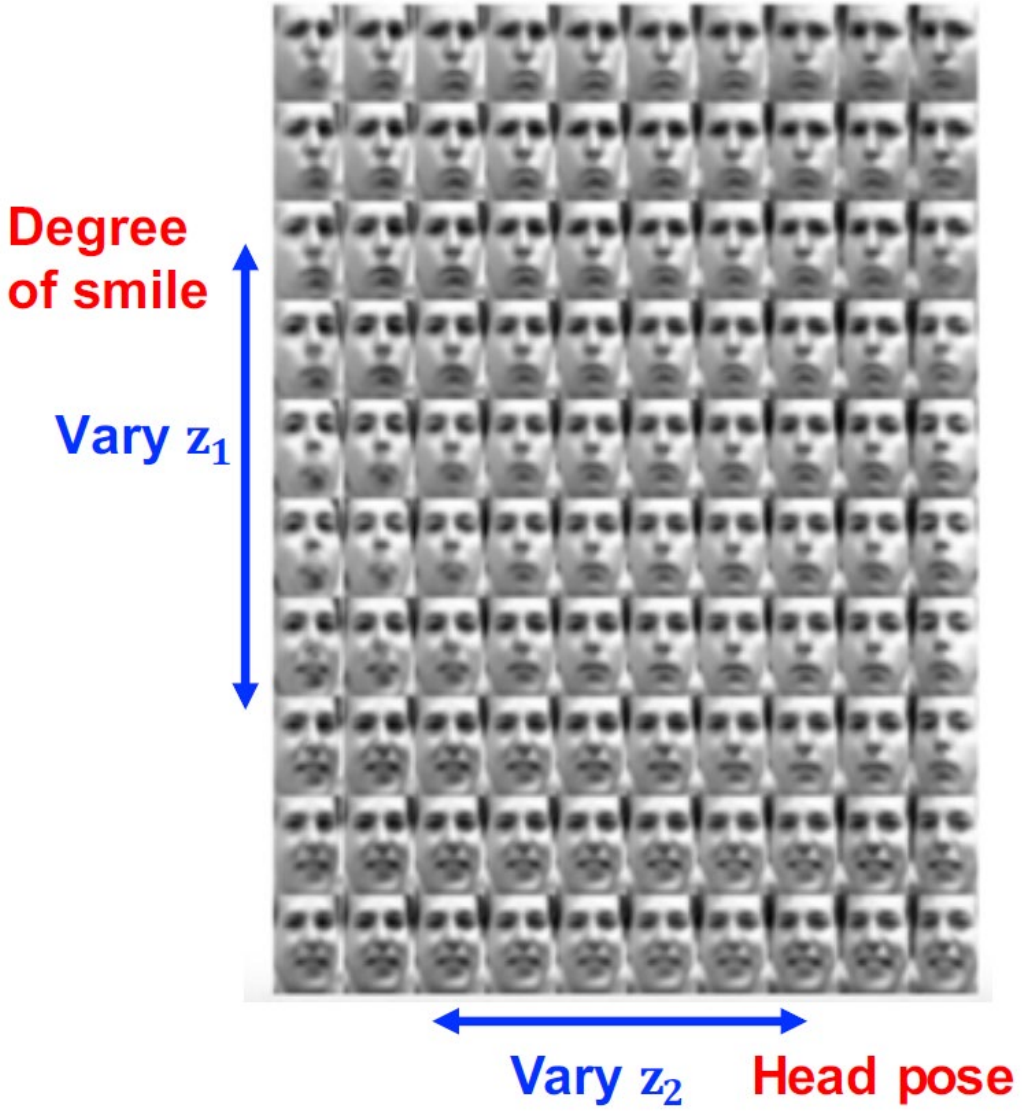
Variational Autoencoder

- The complete Variational Autoencoder



Variational Autoencoder





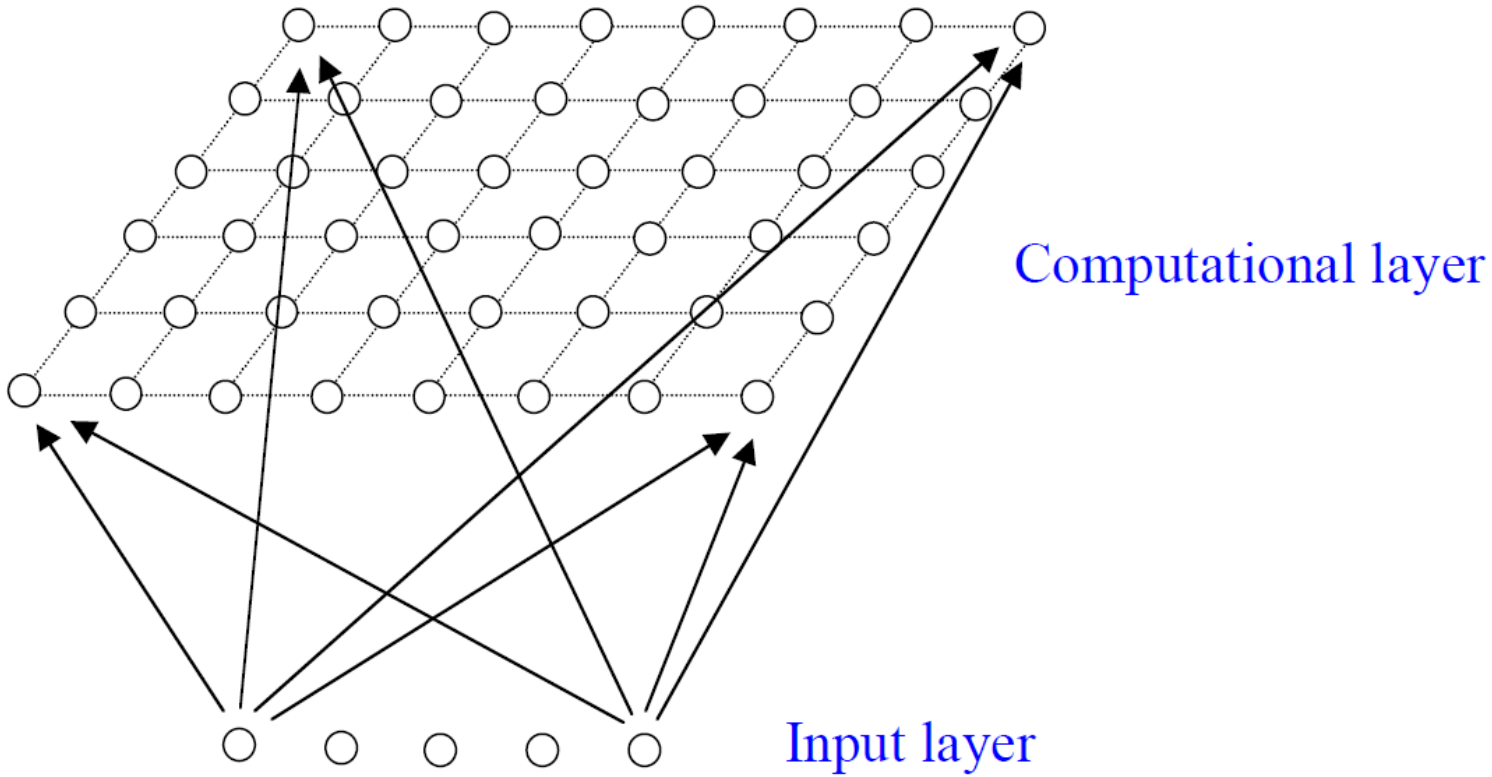
More Extended Autoencoders

- Sparse Autoencoder
- Contractive Autoencoder
- Convolutional Autoencoder
- …

Self Organizing Maps

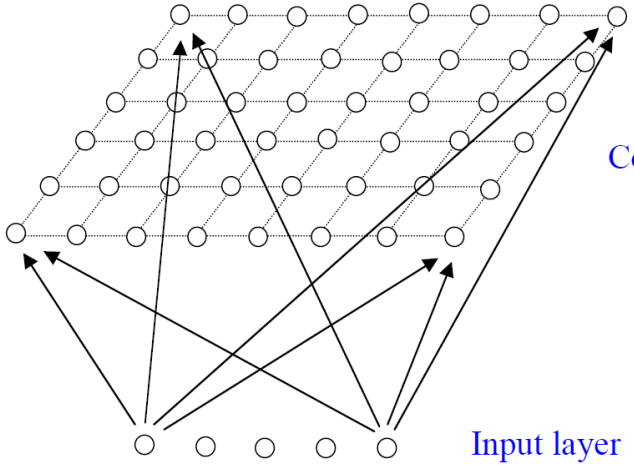
The SOM Algorithm

- To learn a feature map from the spatially continuous input space, to the low dimensional spatially discrete output space
- A one-dimensional map will just have a single row or column

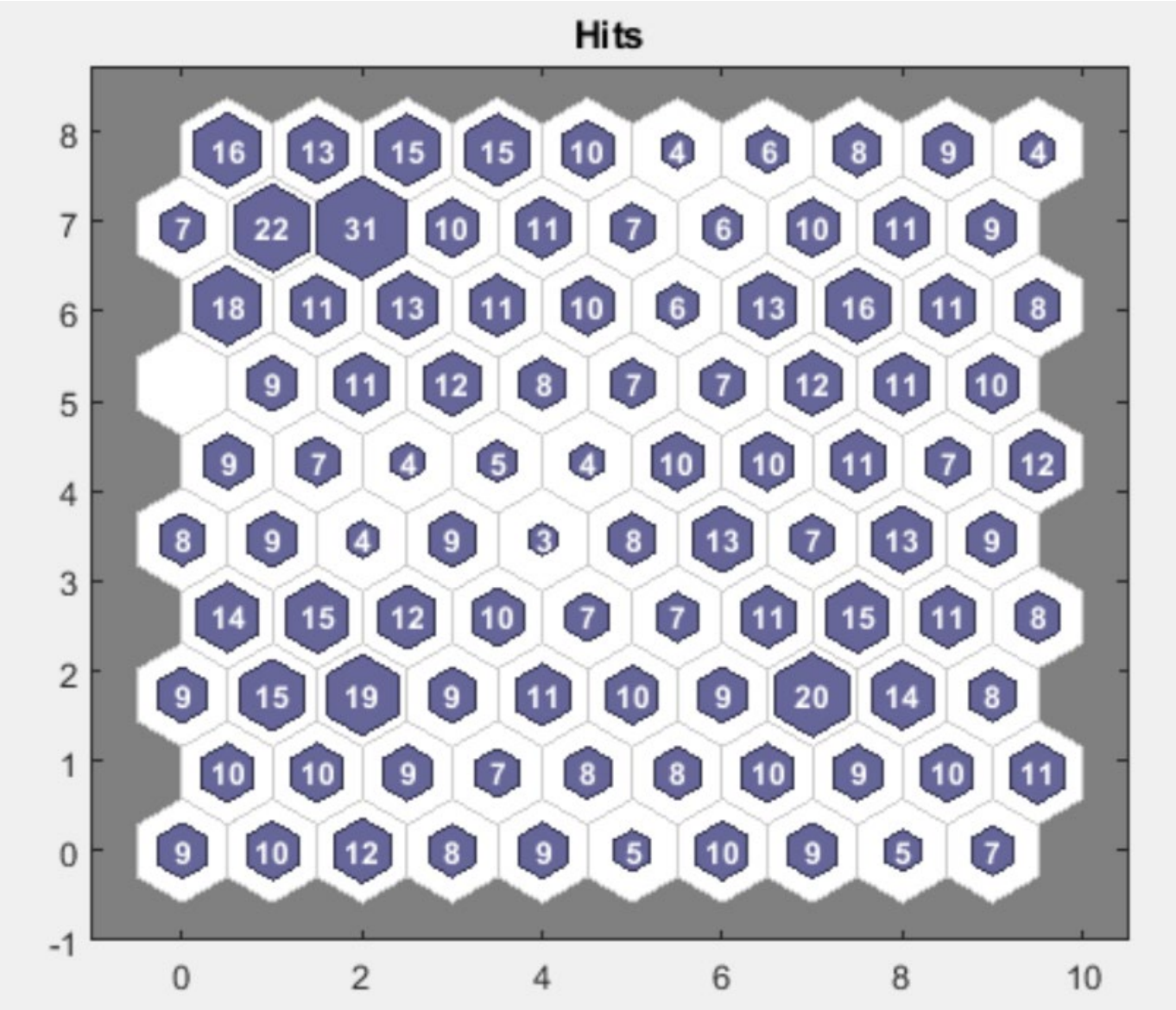


The SOM Algorithm

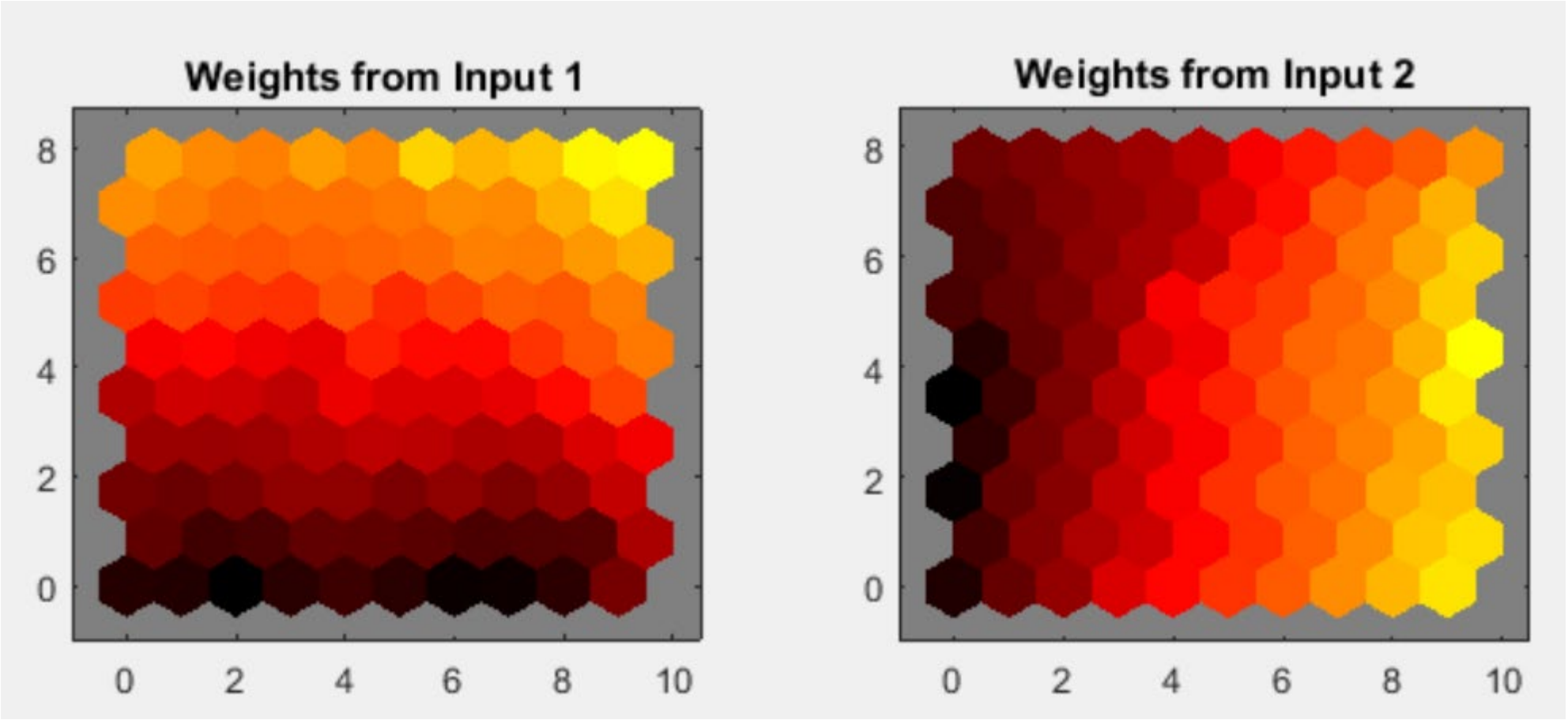
1. ***Initialization*** – Choose random values for the initial weight vectors $\mathbf{w}_j$.
2. ***Sampling*** – Draw a sample training input vector $\mathbf{x}$ from the input space.
3. ***Matching*** – Find the winning neuron $I(\mathbf{x})$ that has weight vector closest to the input vector, i.e. the minimum value of $d_j(\mathbf{x}) = \sum_{i=1}^{D}(x_i - w_{ji})^2$.
4. ***Updating*** – Apply the weight update equation $\Delta w_{ji} = \eta(t)\, T_{j,I(\mathbf{x})}(t)\, (x_i - w_{ji})$ where $T_{j,I(\mathbf{x})}(t)$ is a Gaussian neighbourhood and $\eta(t)$ is the learning rate.
5. ***Continuation*** – keep returning to step 2 until the feature map stops changing.



Experiments

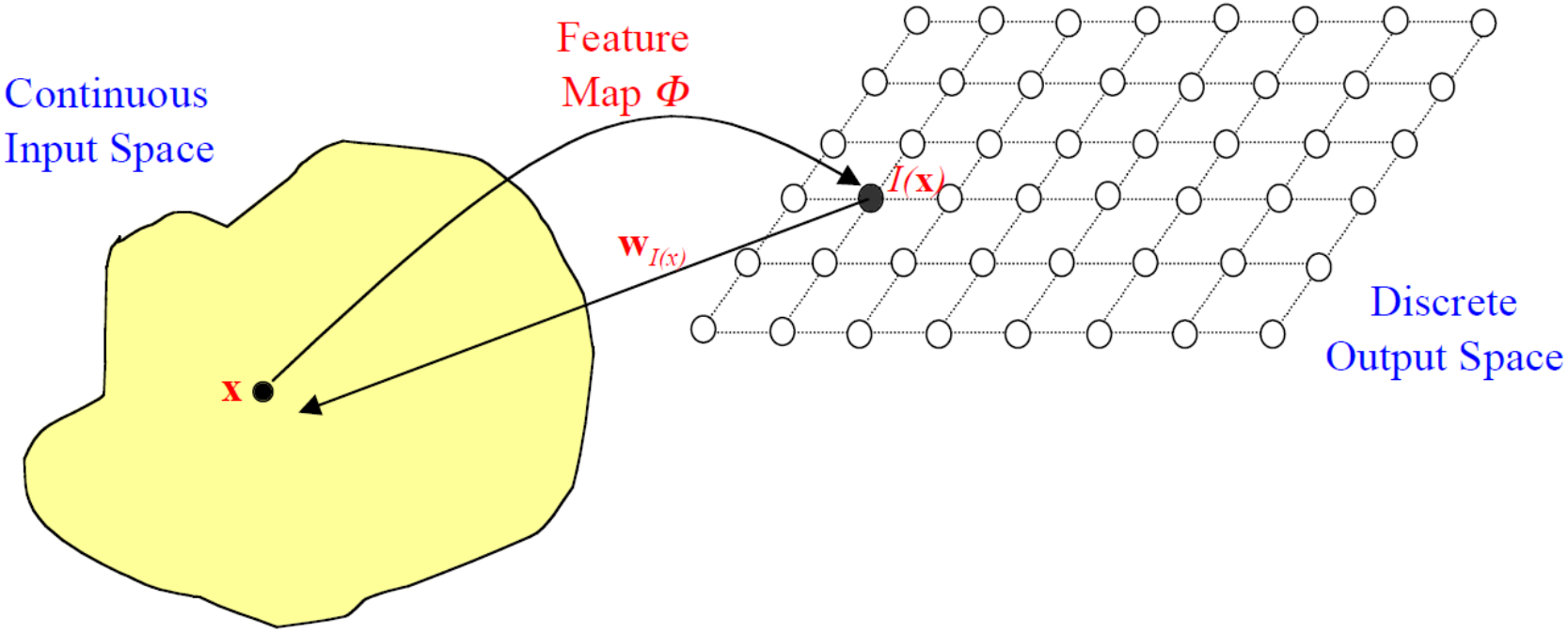


Experiments



Properties

- Approximation of the input space
- Topological ordering
- Density matching
- Feature selection

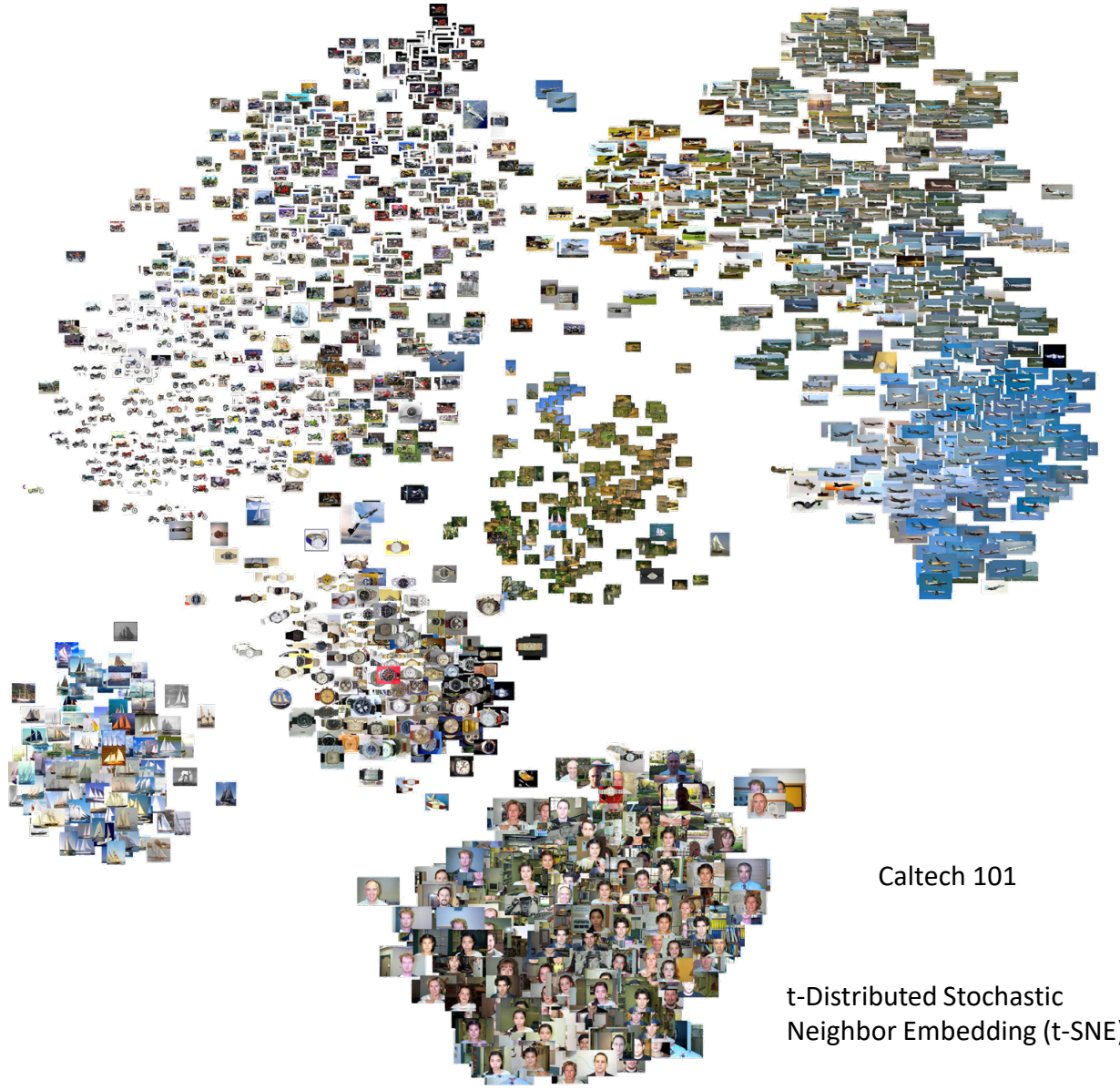


Many More Methods

Many NLDR Methods

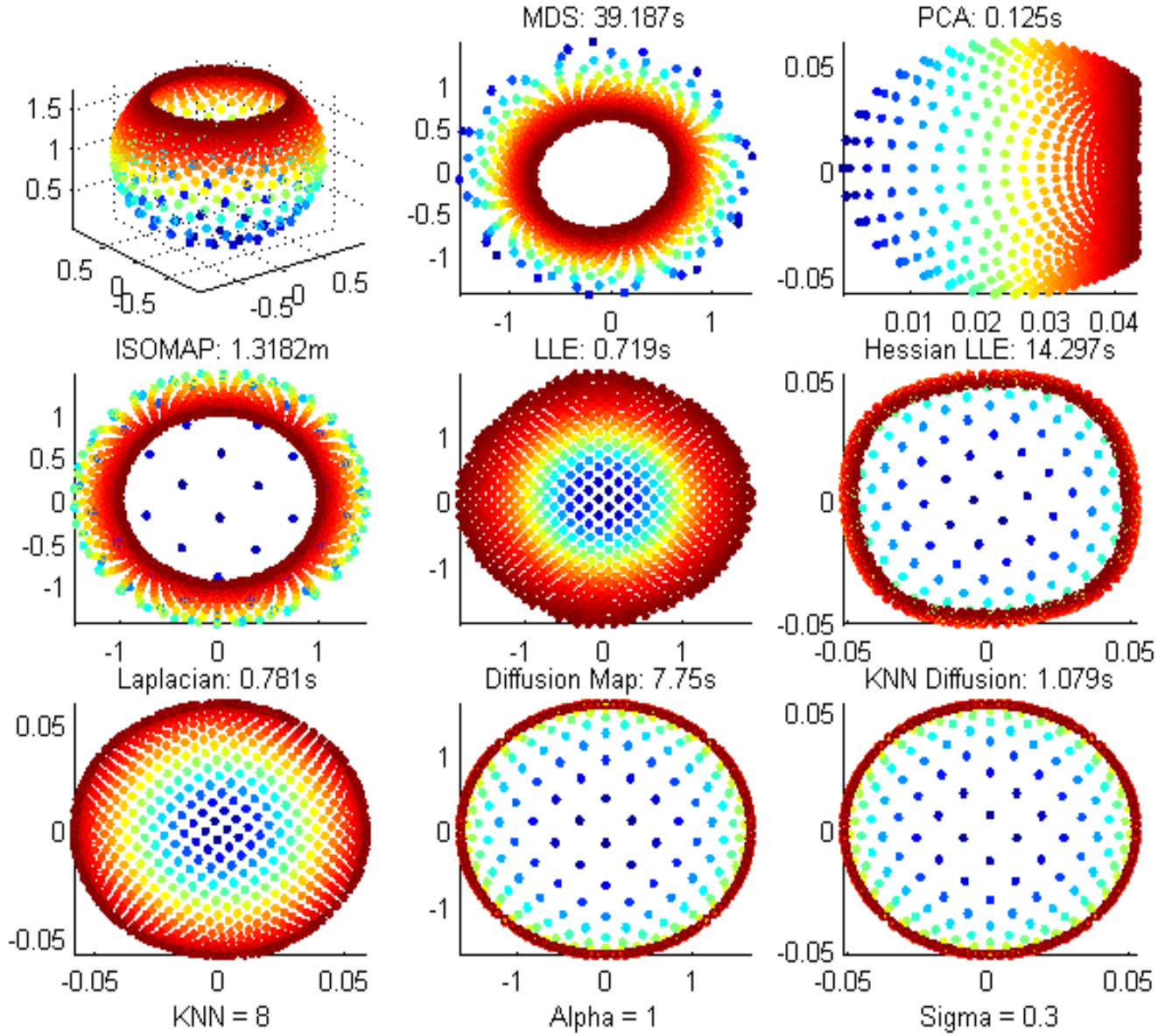
Contents [hide]

From Wikipedia



Caltech 101

t-Distributed Stochastic Neighbor Embedding (t-SNE)


MDS: 39.187s
PCA: 0.125s
ISOMAP: 1.3182m
LLE: 0.719s
Hessian LLE: 14.297s
Laplacian: 0.781s
Diffusion Map: 7.75s
KNN Diffusion: 1.079s
KNN = 8
Alpha = 1
Sigma = 0.3

	MDS	PCA	ISOMAP	LLE	Laplacian	Diffusion Map	KNN Diffusion	Hessian
Speed	Very slow	Extremely fast	Extremely slow	Fast	Fast	Fast	Fast	Slow
Infers geometry?	NO	NO	YES	YES	YES	MAYBE	MAYBE	YES
Handles non-convex?	NO	NO	NO	MAYBE	MAYBE	MAYBE	MAYBE	YES
Handles non-uniform sampling?	YES	YES	YES	YES	NO	YES	YES	MAYBE
Handles curvature?	NO	NO	YES	MAYBE	YES	YES	YES	YES
Handles corners?	NO	NO	YES	YES	YES	YES	YES	NO
Clusters?	YES	YES	YES	YES	NO	YES	YES	NO
Handles noise?	YES	YES	MAYBE	NO	YES	YES	YES	YES
Handles sparsity?	YES	YES	YES	YES	YES	NO	NO	NO may crash
Sensitive to parameters?	NO	NO	YES	YES	YES	VERY	VERY	YES

That’s All