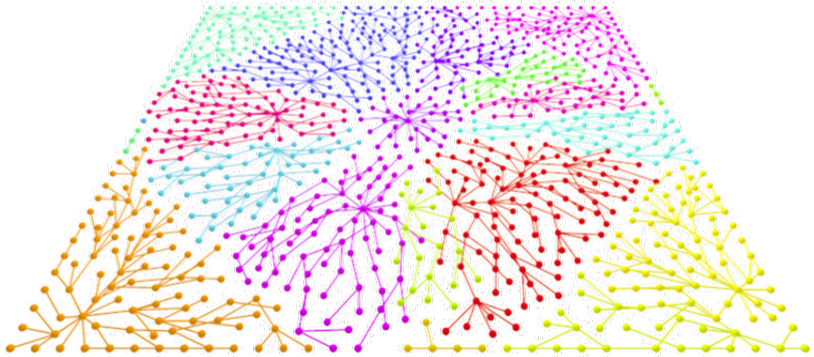
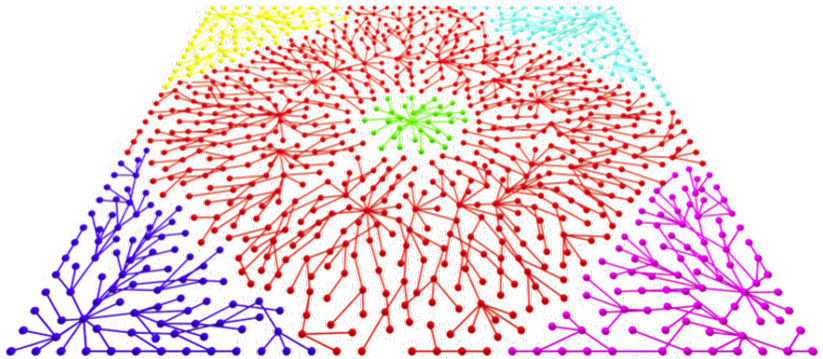
CS233, CME251: Geometric and Topological Data Analysis

Leonidas Guibas
Computer Science Department
Stanford University

Lecture 6
13 April 2022





Last Time:
The Graph View of Data

Graph Laplacians, Laplacian Embeddings, and Spectral Clustering

Graph Notations and Definitions

We consider *simple graphs* (no multiple edges or loops), $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$:

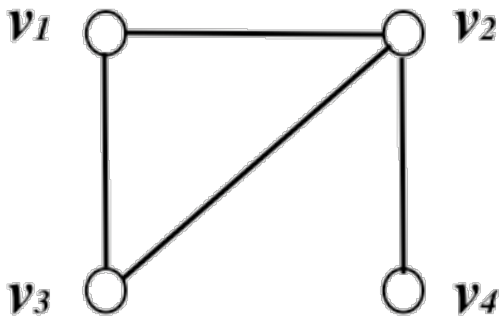
- $\mathcal{V}(\mathcal{G}) = \{v_1, \ldots, v_n\}$ is called the *vertex set* with $n = |\mathcal{V}|$;
- $\mathcal{E}(\mathcal{G}) = \{e_{ij}\}$ is called the *edge set* with $m = |\mathcal{E}|$;
- An edge e_{ij} connects vertices v_i and v_j if they are adjacent or neighbors. One possible notation for adjacency is $v_i \sim v_j$;
- The number of neighbors of a node v is called the *degree* of v and is denoted by $d(v)$, $d(v_i) = \sum_{v_i \sim v_j} e_{ij}$. If all the nodes of a graph have the same degree, the graph is *regular*; The nodes of an *Eulerian* graph have even degree.
- A graph is *complete* if there is an edge between every pair of vertices.

Adjacency Matrices

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

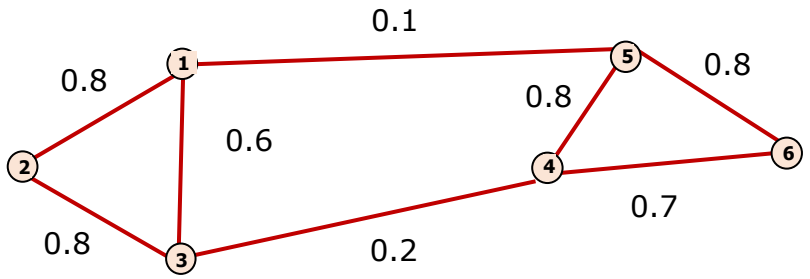
$$\mathbf{A} := \begin{cases} A_{ij} = 1 & \text{if there is an edge } e_{ij} \\ A_{ij} = 0 & \text{if there is no edge} \\ A_{ii} = 0 \end{cases}$$

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



Weighted Matrices

- Adjacency matrix (A)
  - $n \; x \; n$ matrix
  - $A = [w_{ij}]$ edge weight between vertex x_i and x_j



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0.8	0.6	0	0.1	0
x_2	0.8	0	0.8	0	0	0
x_3	0.6	0.8	0	0.2	0	0
x_4	0	0	0.2	0	0.8	0.7
x_5	0.1	0	0	0.8	0	0.8
x_6	0	0	0	0.7	0.8	0

- Important properties:
  - Symmetric matrix

  $\Rightarrow$ Eigenvalues are real

  $\Rightarrow$ Eigenvectors span orthogonal basis

Eigenvalues and Eigenvectors

- $\mathbf{A}$ is a real-symmetric matrix: it has n real eigenvalues and its n real eigenvectors form an orthonormal basis.
- Let $\{\lambda_1, \ldots, \lambda_i, \ldots, \lambda_r\}$ be the set of *distinct* eigenvalues.
- The eigenspace S_i contains the eigenvectors associated with λ_i:

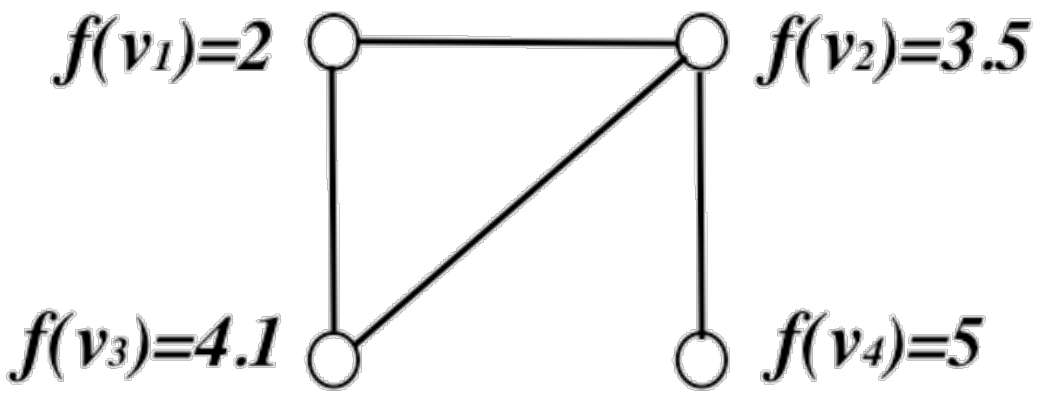
$$S_i = \{\boldsymbol{x} \in \mathbb{R}^n | \mathbf{A}\boldsymbol{x} = \lambda_i \boldsymbol{x}\}$$

- For real-symmetric matrices, the algebraic multiplicity is equal to the geometric multiplicity, for all the eigenvalues.
- The dimension of S_i (geometric multiplicity) is equal to the multiplicity of λ_i.
- If $\lambda_i \neq \lambda_j$ then S_i and S_j are mutually orthogonal.

Order the eigenvalues from small to large

Functions on Graphs

- We consider real-valued functions on the set of the graph's vertices, $\boldsymbol{f} : \mathcal{V} \longrightarrow \mathbb{R}$. Such a function assigns a real number to each graph node.
- $\boldsymbol{f}$ is a vector indexed by the graph's vertices, hence $\boldsymbol{f} \in \mathbb{R}^n$.
- Notation: $\boldsymbol{f} = (f(v_1), \ldots, f(v_n)) = (f(1), \ldots, f(n))$.
- The eigenvectors of the adjacency matrix, $\mathbf{A}\boldsymbol{x} = \lambda\boldsymbol{x}$, can be viewed as *eigenfunctions*.



Operators and Quadratic Forms

- The adjacency matrix can be viewed as an operator

$$\boldsymbol{g} = \mathbf{A}\boldsymbol{f}; g(i) = \sum_{i \sim j} f(j)$$

- It can also be viewed as a quadratic form:

$$\boldsymbol{f}^\top \mathbf{A} \boldsymbol{f} = \sum_{e_{ij}} f(i) f(j)$$

Incidence Matrices for Directed Graphs

- Let each edge in the graph have an arbitrary but fixed orientation;
- The incidence matrix of a graph is a $|\mathcal{E}| \times |\mathcal{V}|$ $(m \times n)$ matrix defined as follows:

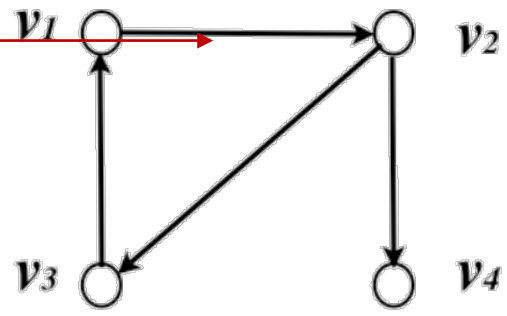
$$\nabla := \begin{cases} \nabla_{ev} = -1 & \text{if } v \text{ is the initial vertex of edge } e \\ \nabla_{ev} = 1 & \text{if } v \text{ is the terminal vertex of edge } e \\ \nabla_{ev} = 0 & \text{if } v \text{ is not in } e \end{cases}$$

Nabla

$$\nabla = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & +1 \end{bmatrix}$$

edges

vertices



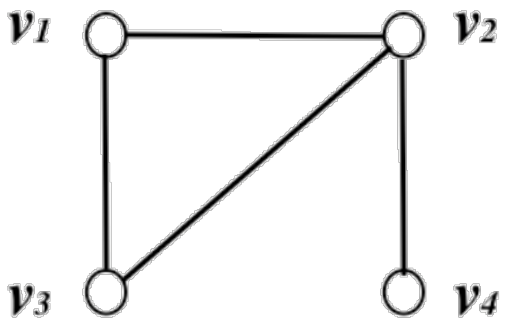
Graph (Unnormalized) Laplacian

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

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

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

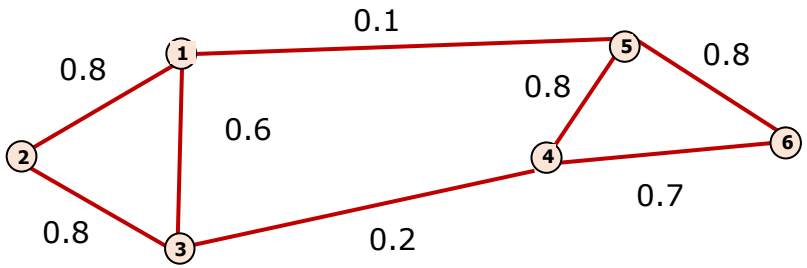
$$\mathbf{L} = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$



Laplacian Matrix for Weighted Graphs

- **Laplacian matrix (L)**
  - n x n symmetric matrix

$$L = D - A$$



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	-0.8	-0.6	0	-0.1	0
x_2	-0.8	1.6	-0.8	0	0	0
x_3	-0.6	-0.8	1.6	-0.2	0	0
x_4	0	0	-0.2	1.7	-0.8	-0.7
x_5	-0.1	0	0	-0.8	1.7	-0.8
x_6	0	0	0	-0.7	-0.8	1.5

- Important properties:
  - Eigenvalues are non-negative real numbers ← 0 always an eigenvalue
  - Eigenvectors are real and orthogonal
  - Eigenvalues and eigenvectors provide insight into the connectivity of the graph…

The Eigenspace of $\lambda_1 = 0$

- The eigenspace corresponding to $\lambda_1 = \ldots = \lambda_k = 0$ is spanned by the k mutually orthogonal vectors:

$$\begin{aligned} \boldsymbol{u}_1 &= \mathbf{1}_{L_1} \\ &\ldots \\ \boldsymbol{u}_k &= \mathbf{1}_{L_k} \end{aligned}$$

- with $\mathbf{1}_{L_i} = (0000111110000)^\top \in \mathbb{R}^n$
- These vectors are the *indicator vectors* of the graph's connected components.
- Notice that $\mathbf{1}_{L_1} + \ldots + \mathbf{1}_{L_k} = \mathbf{1}_n$

The Fiedler Vector

- The first non-null eigenvalue λ_{k+1} is called the Fiedler value.
- The corresponding eigenvector $\boldsymbol{u}_{k+1}$ is called the Fiedler vector.
- The multiplicity of the Fiedler eigenvalue is always equal to 1.
- The Fiedler value is the *algebraic connectivity of a graph*, the further from 0, the more connected.
- The Fidler vector has been extensively used for *spectral bi-partioning*
- Theoretical results are summarized in Spielman & Teng 2007: `http://cs-www.cs.yale.edu/homes/spielman/`

Laplacian Eigenvectors for Connected Graphs

- $\boldsymbol{u}_1 = \mathbf{1}_n, \mathbf{L}\mathbf{1}_n = \mathbf{0}$.
- $\boldsymbol{u}_2$ is the *the Fiedler vector* with multiplicity 1.
- The eigenvectors form an orthonormal basis: $\boldsymbol{u}_i^\top \boldsymbol{u}_j = \delta_{ij}$.
- For any eigenvector $\boldsymbol{u}_i = (\boldsymbol{u}_i(v_1) \dots \boldsymbol{u}_i(v_n))^\top$, $2 \leq i \leq n$:

$$\boldsymbol{u}_i^\top \mathbf{1}_n = 0$$

- Hence the components of $\boldsymbol{u}_i$, $2 \leq i \leq n$ satisfy:

$$\sum_{j=1}^{n} \boldsymbol{u}_i(v_j) = 0$$

- Each component is bounded by:

$$-1 < \boldsymbol{u}_i(v_j) < 1$$

λ_2 = algebraic connectivity, monotone under graph inclusion
The larger λ_2, the more connected the graph.

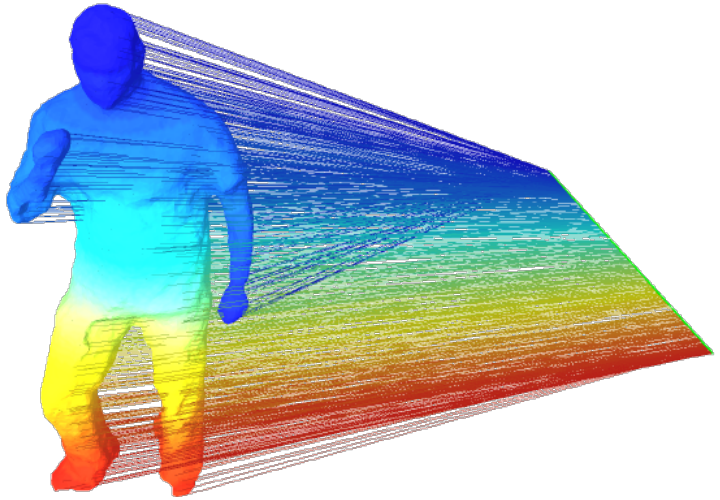
Higher-D Embeddings

- 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):

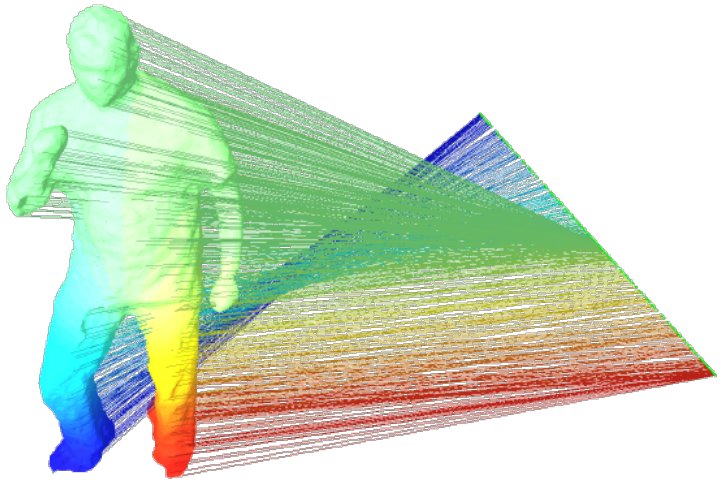
$$\arg \min_{\boldsymbol{f}_1 \cdots \boldsymbol{f}_k} \sum_{i,j=1}^{n} w_{ij} \|\boldsymbol{f}^{(i)} - \boldsymbol{f}^{(j)}\|^2 \text{ with: } \mathbf{F}^\top \mathbf{F} = \mathbf{I}.$$

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

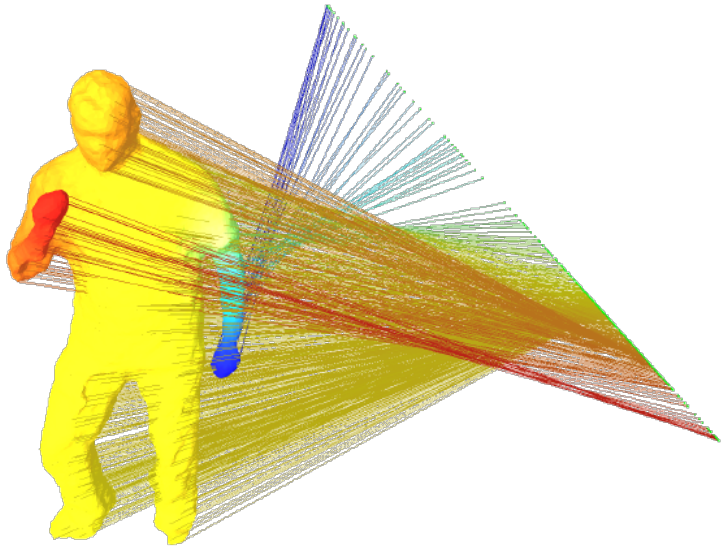
More Eigenvectors, More 1-D Embeddings



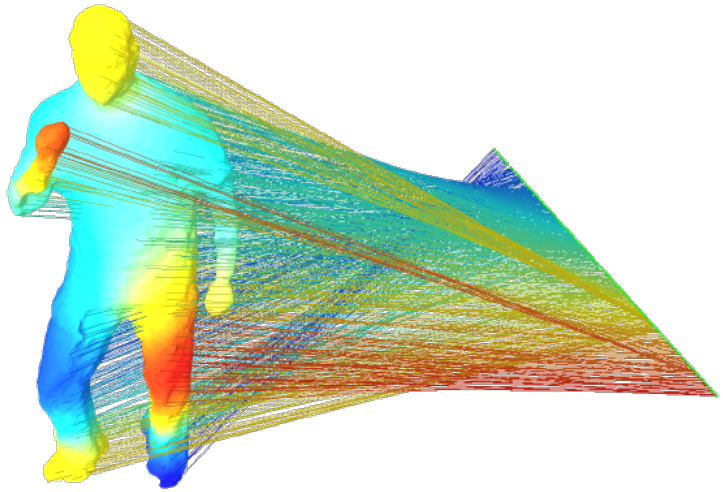
$\boldsymbol{u}_2$



$\boldsymbol{u}_3$



$\boldsymbol{u}_4$



$\boldsymbol{u}_8$

More 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:

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

Graph Partitioning

- **The graph-cut problem**: Partition the graph such that:
  1. Edges between groups have very low weight, and
  2. Edges within a group have high weight.

$$\text{cut}(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} W(A_i, \overline{A}_i) \text{ with } W(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

- **Ratio cut**: (Hagen & Kahng 1992)

$$\text{RatioCut}(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{|A_i|}$$

- **Normalized cut**: (Shi & Malik 2000)

$$\text{NCut}(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A}_i)}{\text{vol}(A_i)}$$

vol = sum of degrees

Spectral Clustering

- Both ratio-cut and normalized-cut minimizations are NP-hard problems
- Spectral clustering is a way to solve relaxed versions of these problems:
  1. The smallest non-null eigenvectors of the *unnormalized Laplacian* approximate the RatioCut minimization criterion, and
  2. The smallest non-null eigenvectors of the *random-walk Laplacian* approximate the NCut criterion.

k-Means Clustering

See Bishop'2006 (pages 424–428) for more details.

- What is a cluster: a group of points whose inter-point distance are small compared to distances to points outside the cluster.
- Cluster centers: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$.
- Goal: find an assignment of points to clusters as well as a set of vectors $\boldsymbol{\mu}_i$.
- Notations: For each point $\boldsymbol{y}_j$ there is a *binary indicator variable* $r_{ji} \in \{0, 1\}$.
- Objective: minimize the following *distorsion measure*:

$$J = \sum_{j=1}^{n} \sum_{i=1}^{k} r_{ji} \|\boldsymbol{y}_j - \boldsymbol{\mu}_i\|^2$$

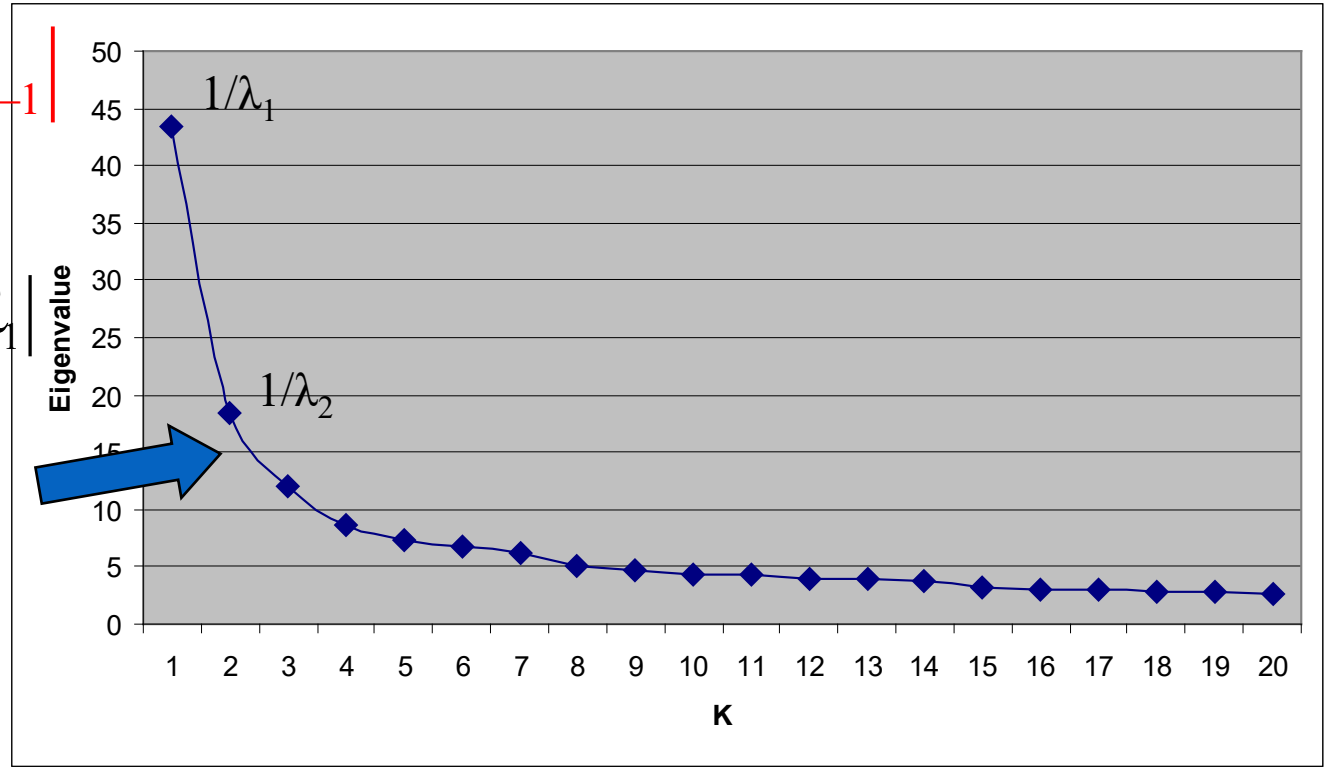
Spectral Gap: Selecting *k*

- *Eigengap*: the difference between two consecutive eigenvalues.
- Most stable clustering is generally given by the value k that maximizes the expression

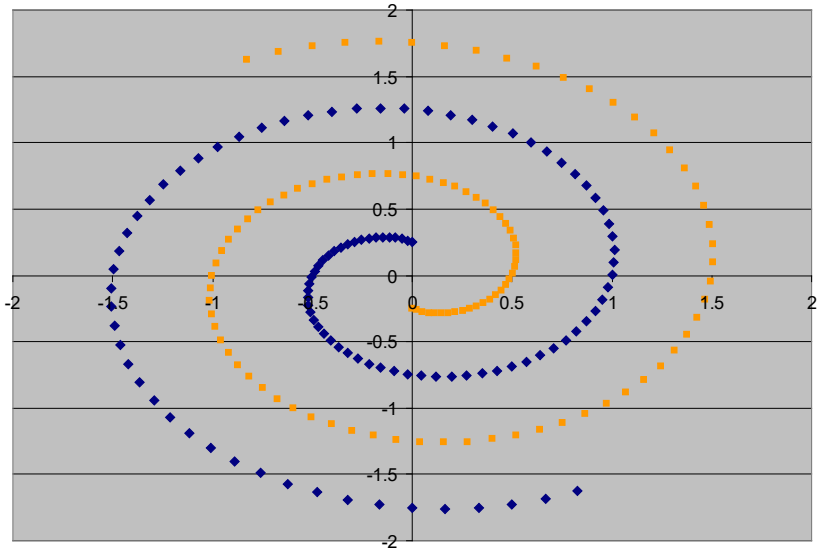
$$\Delta_k = \left| 1/\lambda_k - 1/\lambda_{k-1} \right|$$

$$\max \Delta_k = \left| 1/\lambda_2 - 1/\lambda_1 \right|$$

$\Rightarrow$ Choose $k=2$

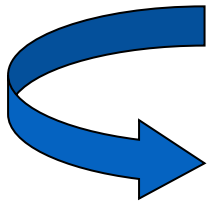


Spirals Again



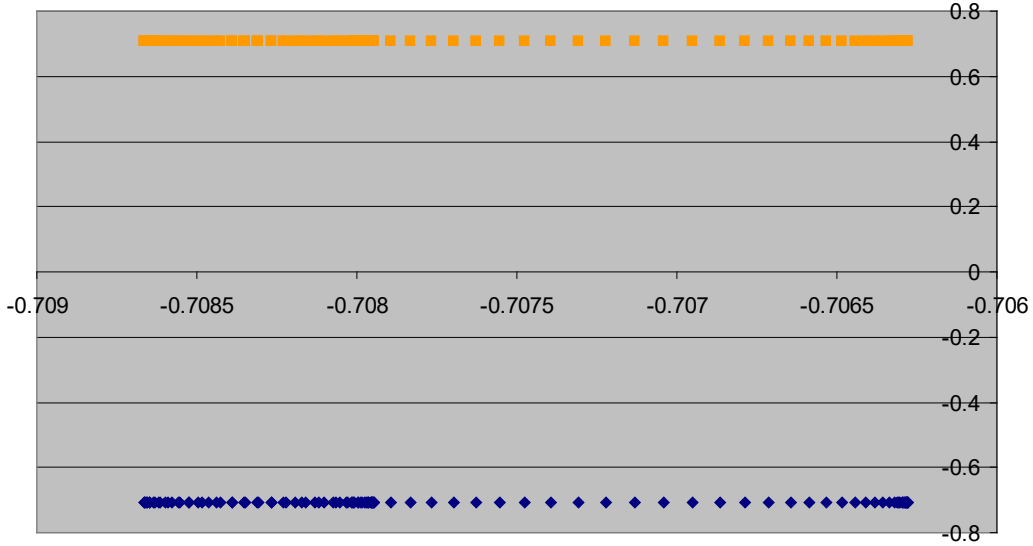
Dataset exhibits complex cluster shapes

$\Rightarrow$ 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



Today: Non-Linear Dimensionality Reduction

Locally Linear Embeddings (LLE)

Laplacian Eigenmaps

Isomap

t-distributed Stochastic Neighbor Embedding (t-SNE)

Dimensionality Reduction

Dimensionality Reduction is the process of **transforming data** from a **high-dimensional space into a low-dimensional space** so that **the low-dimensional representation retains some meaningful properties of the original data.**

Source: Wikipedia on Dimensionality Reduction

Dimensionality Reduction

Dimensionality Reduction is the process of **transforming data** from a **high-dimensional space into a low-dimensional space** so that **the low-dimensional representation retains some meaningful properties of the original data.**

Source: Wikipedia on Dimensionality Reduction

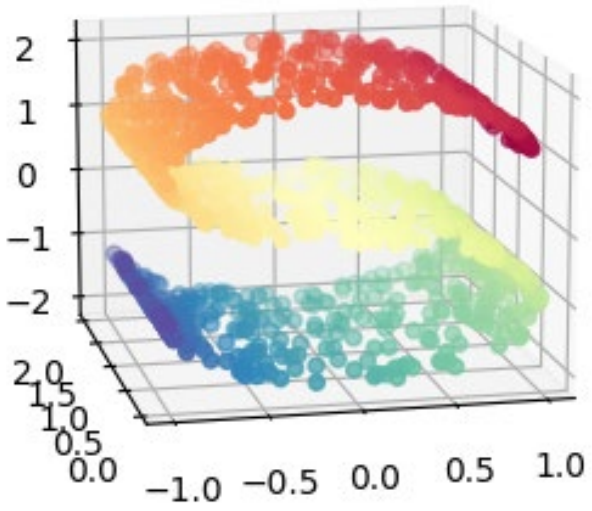
The reduction can be performed by:

- **Selection**, namely only some of the existing features are preserved.
- **Extraction**, namely **a reduced number of new features are created** based on the old features.

Dimensionality Reduction

Dimensionality Reduction is the process of **transforming data** from a **high-dimensional space into a low-dimensional space** so that **the low-dimensional representation retains some meaningful properties of the original data.**

Source: Wikipedia on Dimensionality Reduction



S-curve

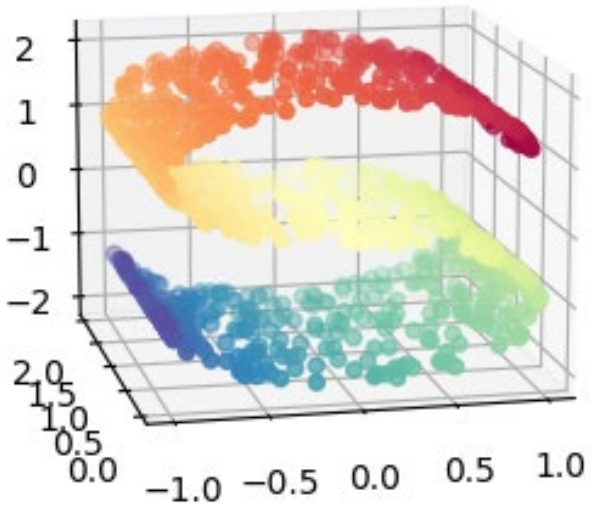
Key Assumptions:

- Represent each data point by a point in a lower dimensional space.

Dimensionality Reduction

Dimensionality Reduction is the process of **transforming data** from a **high-dimensional space into a low-dimensional space** so that **the low-dimensional representation retains some meaningful properties of the original data.**

Source: Wikipedia on Dimensionality Reduction



S-curve

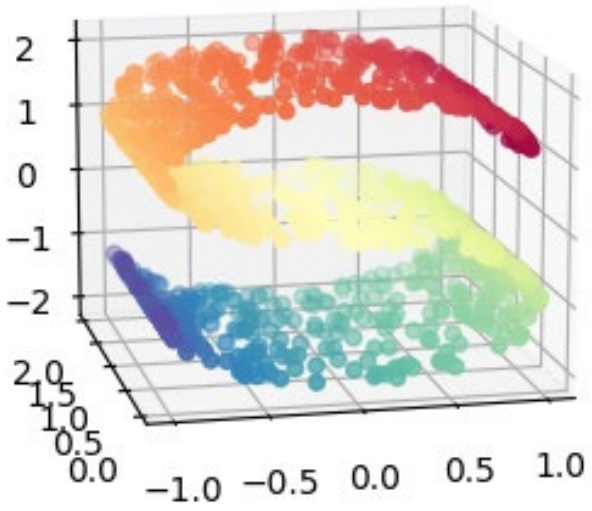
Key Assumptions:

- Represent each data point by a point in a lower dimensional space.
- Choose **the low-dimensional points so that they can represent some properties of the data points in the original space**, such as the pairwise distances between points, namely two close points in the original high-dimensional space should also be close in the low-dimensional space.

Dimensionality Reduction

Dimensionality Reduction is the process of **transforming data** from a **high-dimensional space into a low-dimensional space** so that **the low-dimensional representation retains some meaningful properties of the original data.**

Source: Wikipedia on Dimensionality Reduction



S-curve

Key Assumptions:

- Represent each data point by a point in a lower dimensional space.
- Choose **the low-dimensional points so that they can represent some properties of the data points in the original space**, such as the pairwise distances between points, namely two close points in the original high-dimensional space should also be close in the low-dimensional space.
- **We do not necessarily want to learn either a decoding function** that can reconstruct a 3D point from its low dimensional representative **or an encoding function that maps each high-dimensional point to its low representative.**

Dimensionality Reduction

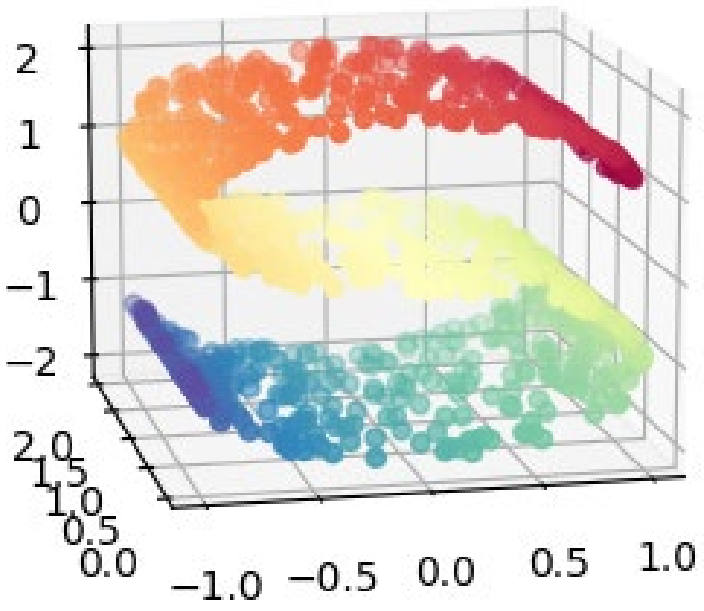
Dimensionality Reduction is the process of **transforming data** from a **high-dimensional space into a low-dimensional space** so that **the low-dimensional representation retains some meaningful properties of the original data.**

Source: Wikipedia on Dimensionality Reduction

Based on the type of the transformation function, we have:

- **Linear Dimensionality Reduction:** Data points are assumed to lie on a linear subspace, i.e. on a line, plane, hyperplane .
  - PCA
  - MDS (Multi-Dimensional Scaling)
  - LDA (Linear Discriminant Analysis)
  - ….
- **Non-Linear Dimensionality Reduction:** Data points are assumed to lie on a curved subspace, or more generally a manifold.
  - Kernel PCA
  - Focus of today's lecture

Dimensionality Reduction - Examples

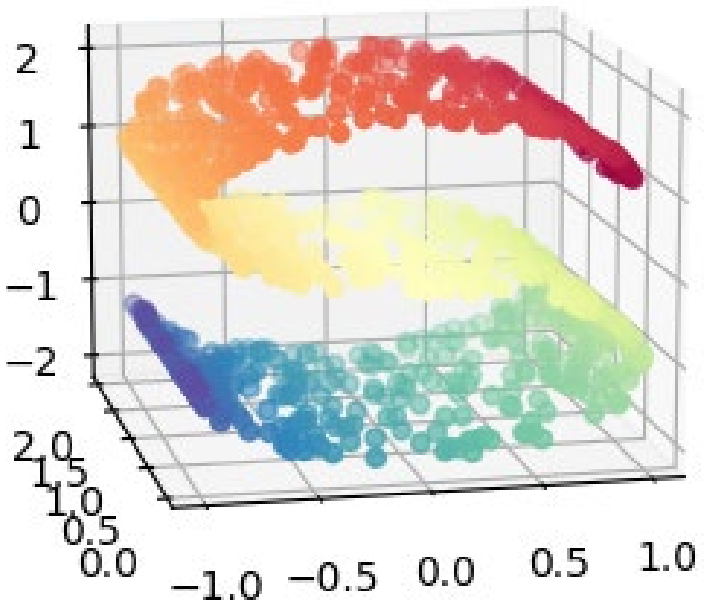


S-curve

Why do we need both linear and non-linear projection/transformations?

- Linear subspaces may be inefficient for some datasets, namely **many datasets contain essential non-linear structures** that are invisible to linear projection methods.

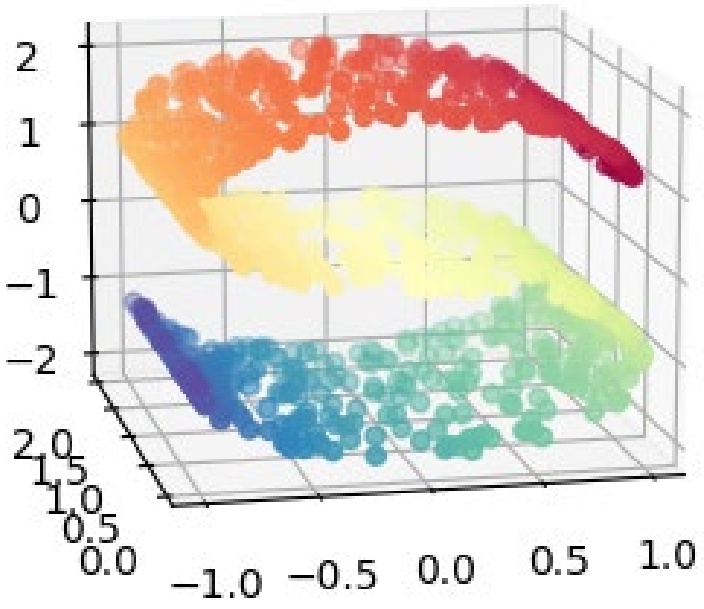
Dimensionality Reduction - Examples



S-curve

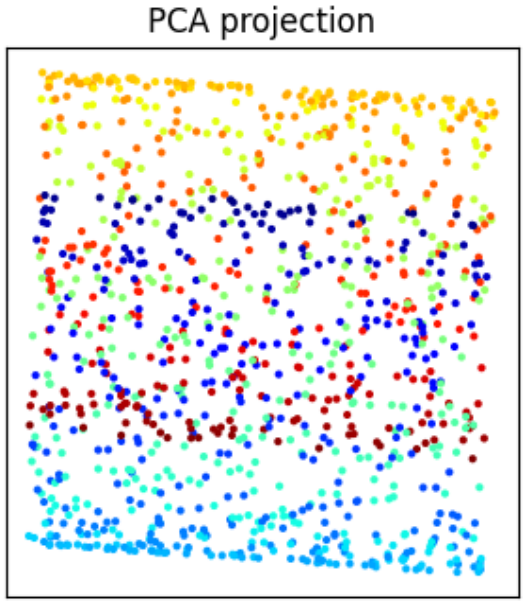
For the S-curve, which is a curved manifold, if we were to perform linear dimensionality reduction that would amount to finding a hyperplane cutting through the data and projecting all the points into that hyperplane. **Can we find a hyperplane of any orientation that would allow us to recover the original points from the projected data?**

Dimensionality Reduction - Examples

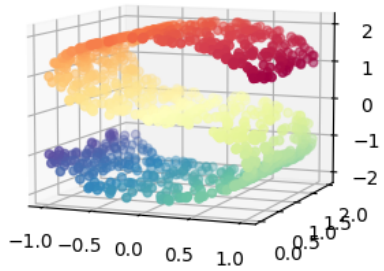


S-curve

For the S-curve, which is a curved manifold, if we were to perform linear dimensionality reduction that would amount to finding a hyperplane cutting through the data and projecting all the points into that hyperplane. **Can we find a hyperplane of any orientation that would allow us to recover the original points from the projected data?**



Dimensionality Reduction - Examples



S-curve

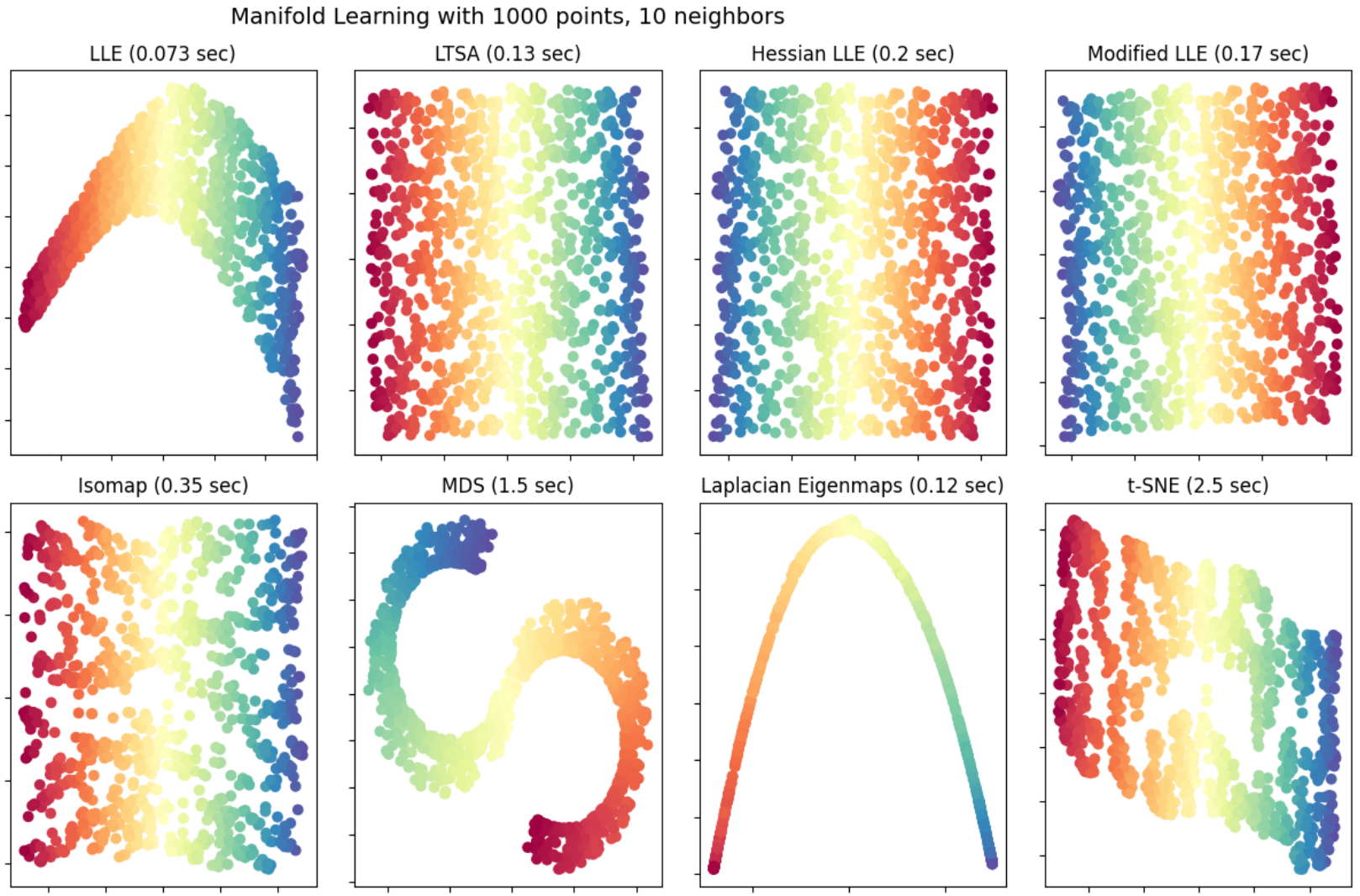
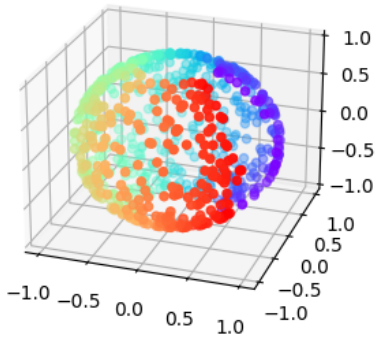


Image Generated running the scikit-learn demo for Manifold Learning Methods:
https://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html

Dimensionality Reduction - Examples



Punched Sphere

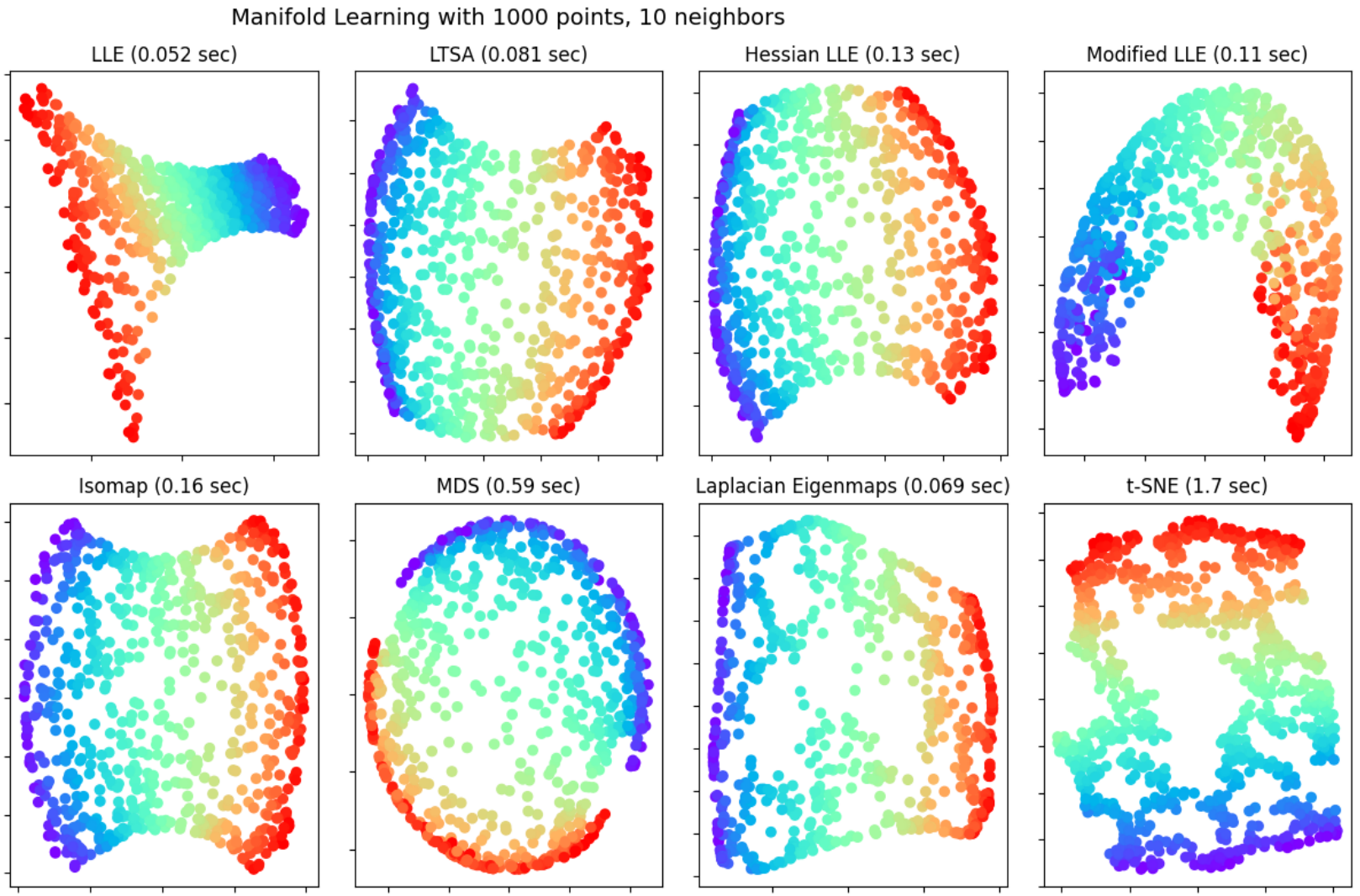


Image Generated running the scikit-learn demo for Manifold Learning Methods:
https://scikit-learn.org/stable/auto_examples/manifold/plot_manifold_sphere.html

Dimensionality Reduction - Examples

Collected data for **500,000 cells** from a mouse nervous system, and for each cell they measured **how strongly each gene (features) is expressed in each cell**.

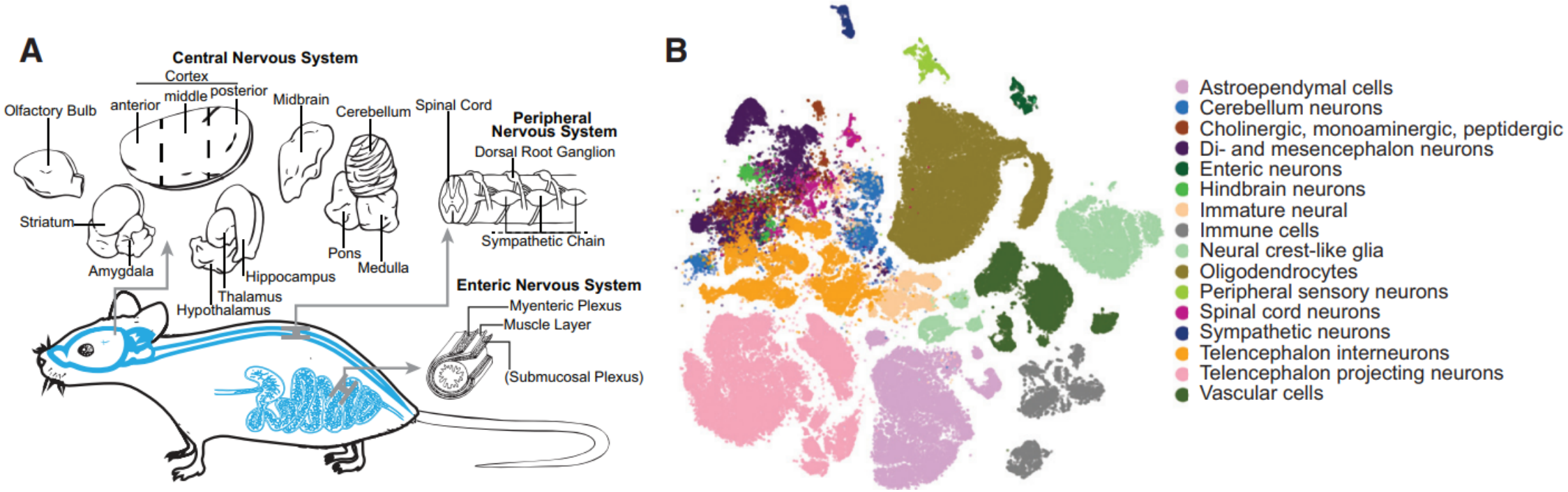


Image Source: Molecular Architecture of the Mouse Nervous System by Zeisel et. al.

Dimensionality Reduction - Examples

Collected data from approximately 500,000 people in the UK and tried to infer the people's ethnic origin using snips (single-neuclotide polymorphisms) as features. Snips tells as at which position in our genome, our genetic code is different from the average genome.

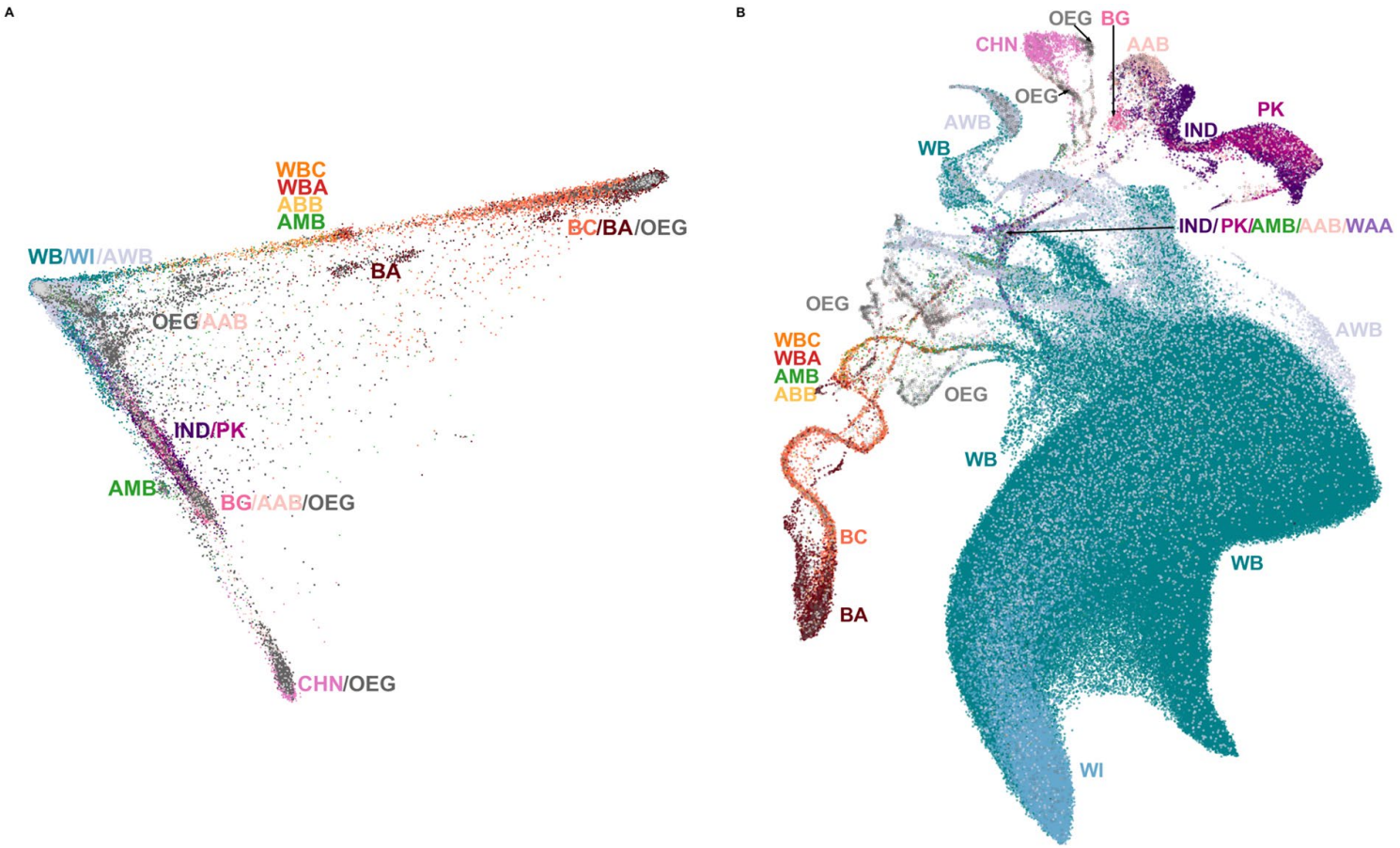


Image Source: UMAP reveals cryptic population structure and phenotype heterogeneity in large genomic cohorts, Papkovich et. al.

Dimensionality Reduction - Examples

The MNIST dataset contains 70,000 samples, each of which can be described with 64x64=768 pixel values. Here the pixels are our features.

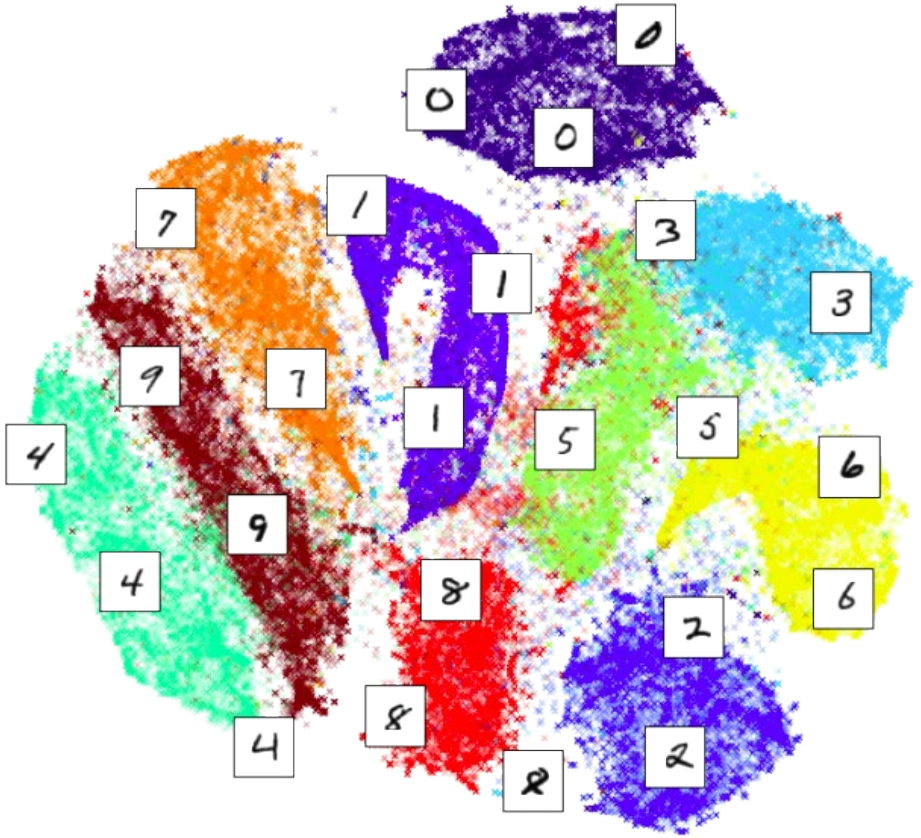


Image Source: t-SNE on MNIST from Kaggle

Methods for Non Linear Dimensionality Reduction

- Sammon's mapping, 1969
- Self-organizing map (SOM, aka Kohonen map, based on neural networks), 1982
- Principal curves and manifolds, 1984
- Autoencoders (some neural networks), 19xx
- Generative topographic map (GTM, probabilistic version of SOM), 1996
- Curvilinear component/distance analysis (CCA, CDA), 1997
- Kernel PCA (kPCA), 1998
- ISOMAP, 2000
- Locally-linear embedding (LLE), 2000
- Laplacian Eigenmaps, 2001
- Hessian LLE, 2003
- Gaussian process latent variable models (GPLVM), 2004
- Maximum variance unfolding (MVA, aka semidefinite embedding), 2004
- Maximum variance unfolding (MVA, aka semidefinite embedding), 2004
- Relational perspective map, 2004
- Nonlinear PCA (based on neural networks), 2005
- Local tangent space alignment (LTSA), 2005
- Modified LLE, 2006
- Diffusion maps, 2006
- Local multidimensional scaling, 2006
- Manifold alignment, 2008
- Manifold sculpting, 2008
- t-distributed stochastic neighbor embedding (t-SNE), 2008
- Diffeomorphic Dimensionality Reduction (Diffeomap), 2009
- Rank visu, 2009
- Topologically Constrained Isometric Embedding (TCIE), 2010

List Source: https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction

Methods for Non Linear Dimensionality Reduction

- **Sammon's mapping, 1969**
- Self-organizing map (SOM, aka Kohonen map, based on neural networks), 1982
- Principal curves and manifolds, 1984
- **Autoencoders (some neural networks), 19xx**
- Generative topographic map (GTM, probabilistic version of SOM), 1996
- Curvilinear component/distance analysis (CCA, CDA), 1997
- Kernel PCA (kPCA), 1998
- **ISOMAP, 2000**
- **Locally-linear embedding (LLE), 2000**
- **Laplacian Eigenmaps, 2001**
- Hessian LLE, 2003
- Gaussian process latent variable models (GPLVM), 2004
- Maximum variance unfolding (MVA, aka semidefinite embedding), 2004
- Maximum variance unfolding (MVA, aka semidefinite embedding), 2004
- Relational perspective map, 2004
- Nonlinear PCA (based on neural networks), 2005
- Local tangent space alignment (LTSA), 2005
- Modified LLE, 2006
- Diffusion maps, 2006
- Local multidimensional scaling, 2006
- Manifold alignment, 2008
- Manifold sculpting, 2008
- **t-distributed stochastic neighbor embedding (t-SNE), 2008**
- Diffeomorphic Dimensionality Reduction (Diffeomap), 2009
- Rank visu, 2009
- Topologically Constrained Isometric Embedding (TCIE), 2010

List Source: https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction

MDS:
Multi-Dimensional Scaling

Multi-Dimensional Scaling (MDS)

Multi-Dimensional Scaling (MDS) seeks to find a mapping between a set of points $\mathcal{X} = \mathbf{x}_1, \ldots, \mathbf{x}_M \in \mathbb{R}^N$ to a set of points $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\} \in \mathbb{R}^K$, where $K << N$, such that **the pairwise distances between points in the original higher-dimensional space and the projected lower-dimensional space is minimized.**

Multi-Dimensional Scaling (MDS)

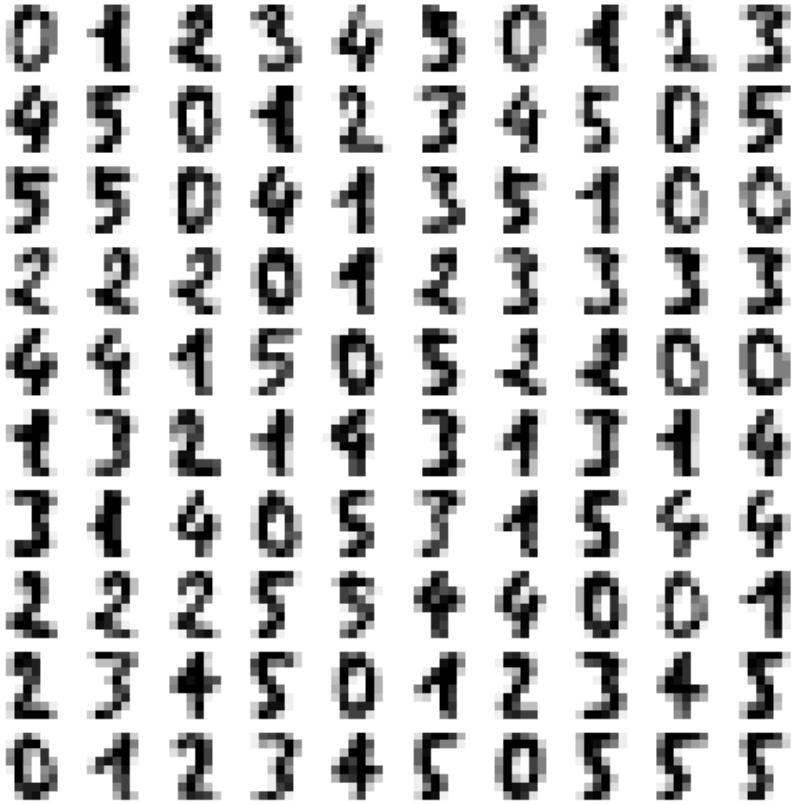
Multi-Dimensional Scaling (MDS) seeks to find a mapping between a set of points $\mathcal{X} = \mathbf{x}_1, \ldots, \mathbf{x}_M \in \mathbb{R}^N$ to a set of points $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\} \in \mathbb{R}^K$, where $K << N$, such that **the pairwise distances between points in the original higher-dimensional space and the projected lower-dimensional space is minimized.**

Usually MDS is formulated as the following optimization problem:

$$\underset{\mathcal{Y}=\{\mathbf{y}_1,\ldots,\mathbf{y}_M\}}{\operatorname{argmin}} \sum_{i<j} (|| \mathbf{x}_i - \mathbf{x}_j ||^2 - || \mathbf{y}_i - \mathbf{y}_j ||^2)^2$$

Multi-Dimensional Scaling (MDS) - MNIST

A selection from the 64-dimensional digits dataset



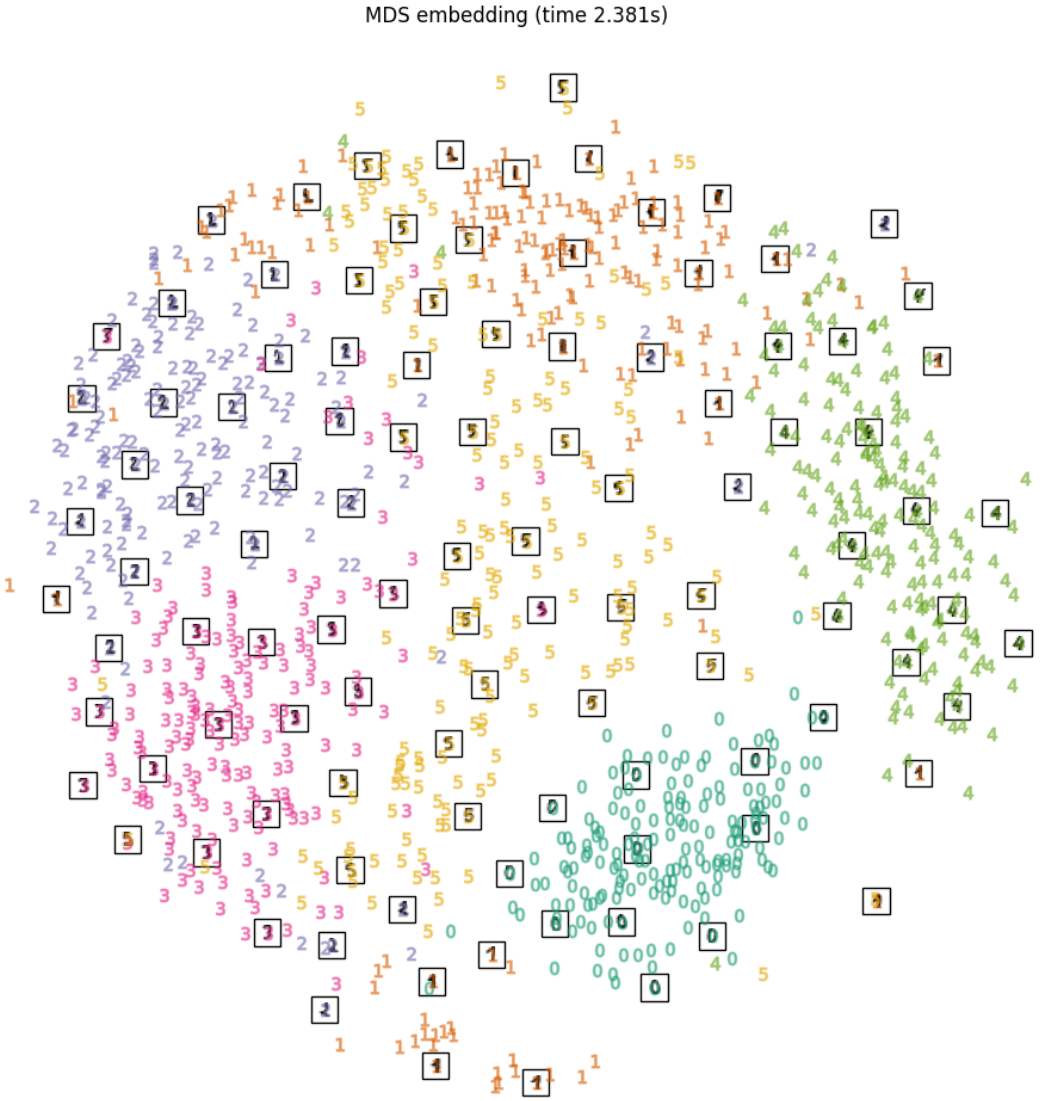
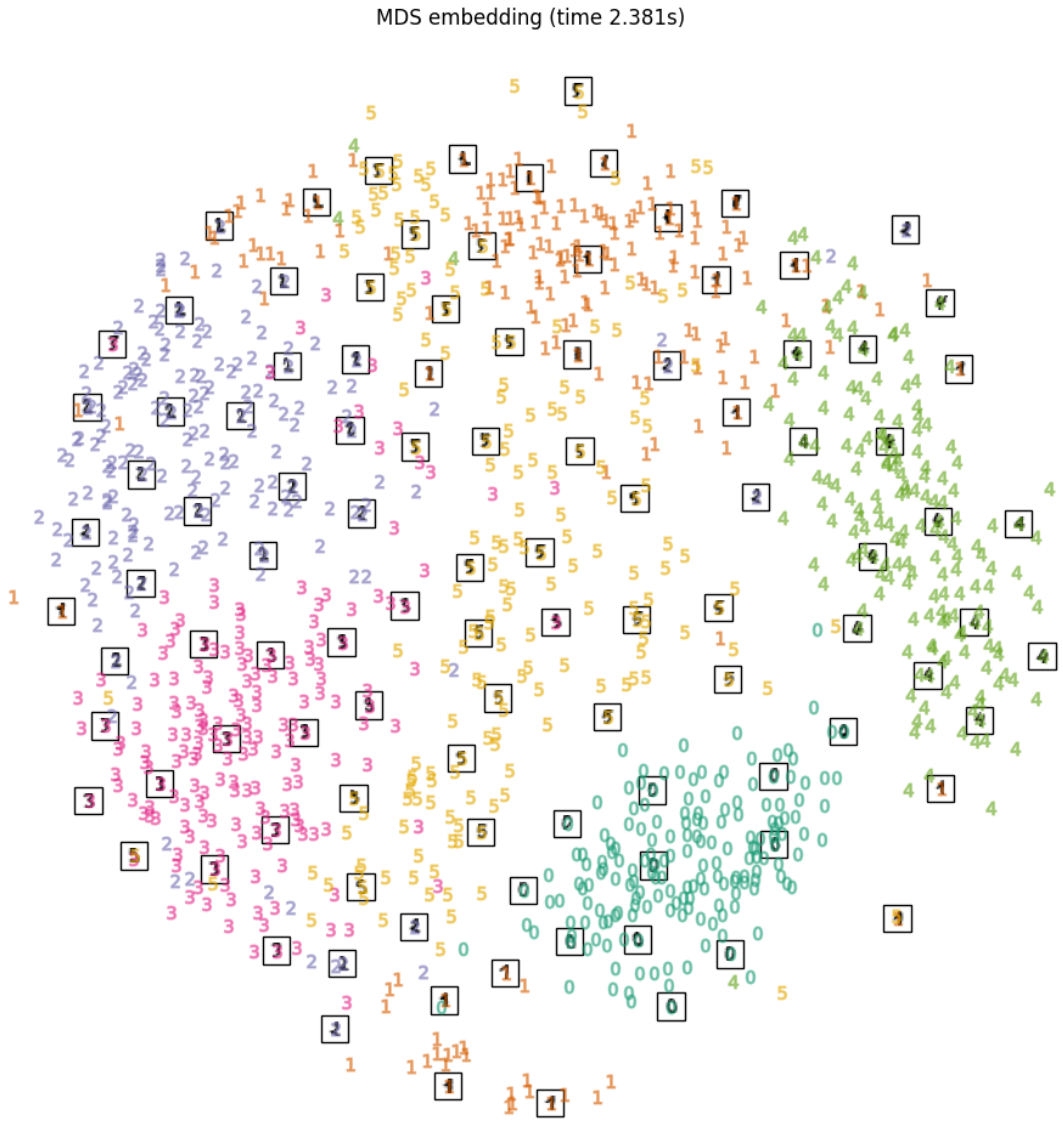


Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

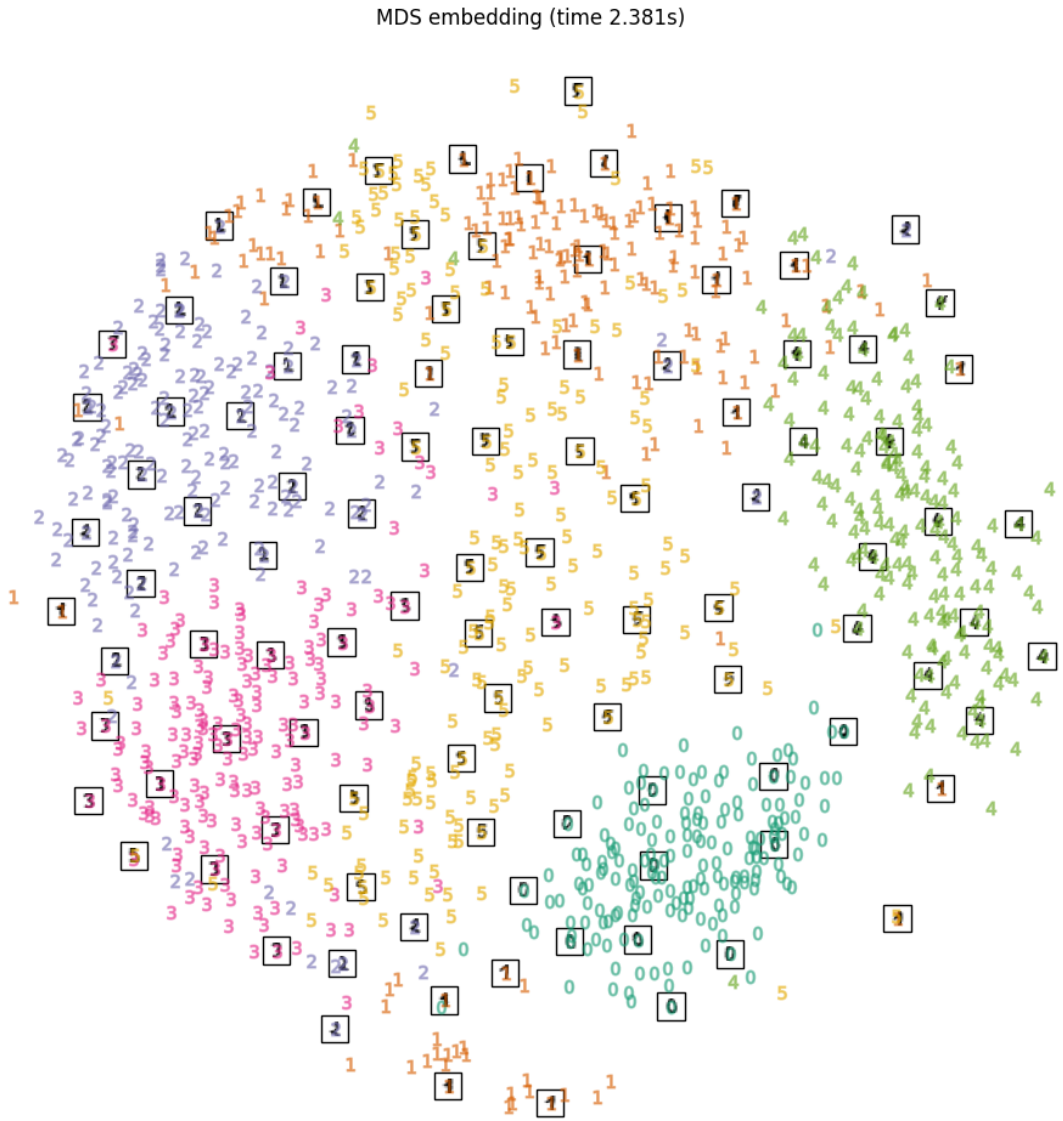
Why does MDS fail on MNIST?



It seems that MDS doesn't show the inherent structure in the data?

Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

Why does MDS fail on MNIST?

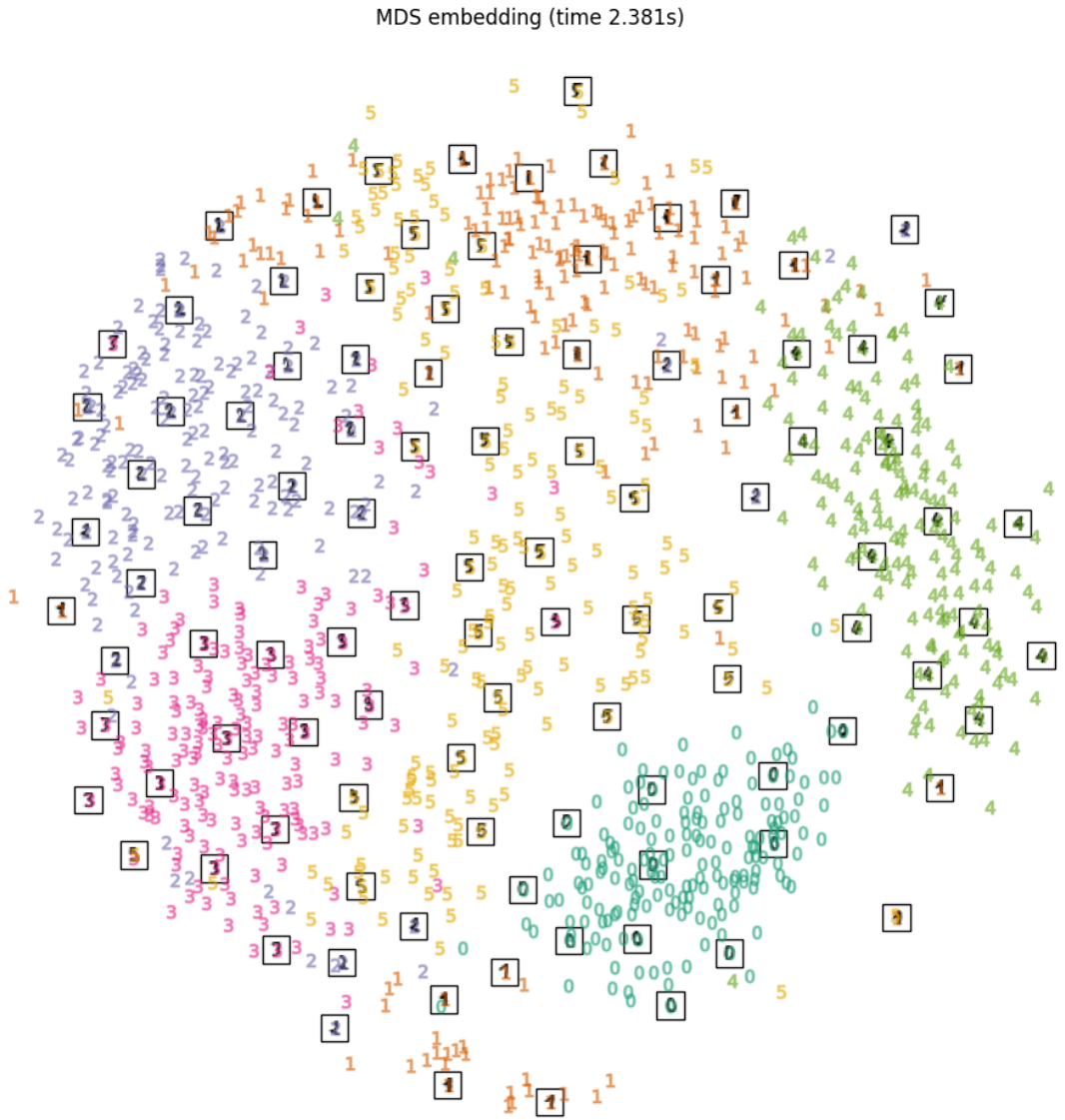


It seems that MDS doesn't show the inherent structure in the data?

- The idea of **preserving high-dimensional distances in low-dimensional embeddings is a bad idea.** (curse of dimensionality)

Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

Why does MDS fail on MNIST?

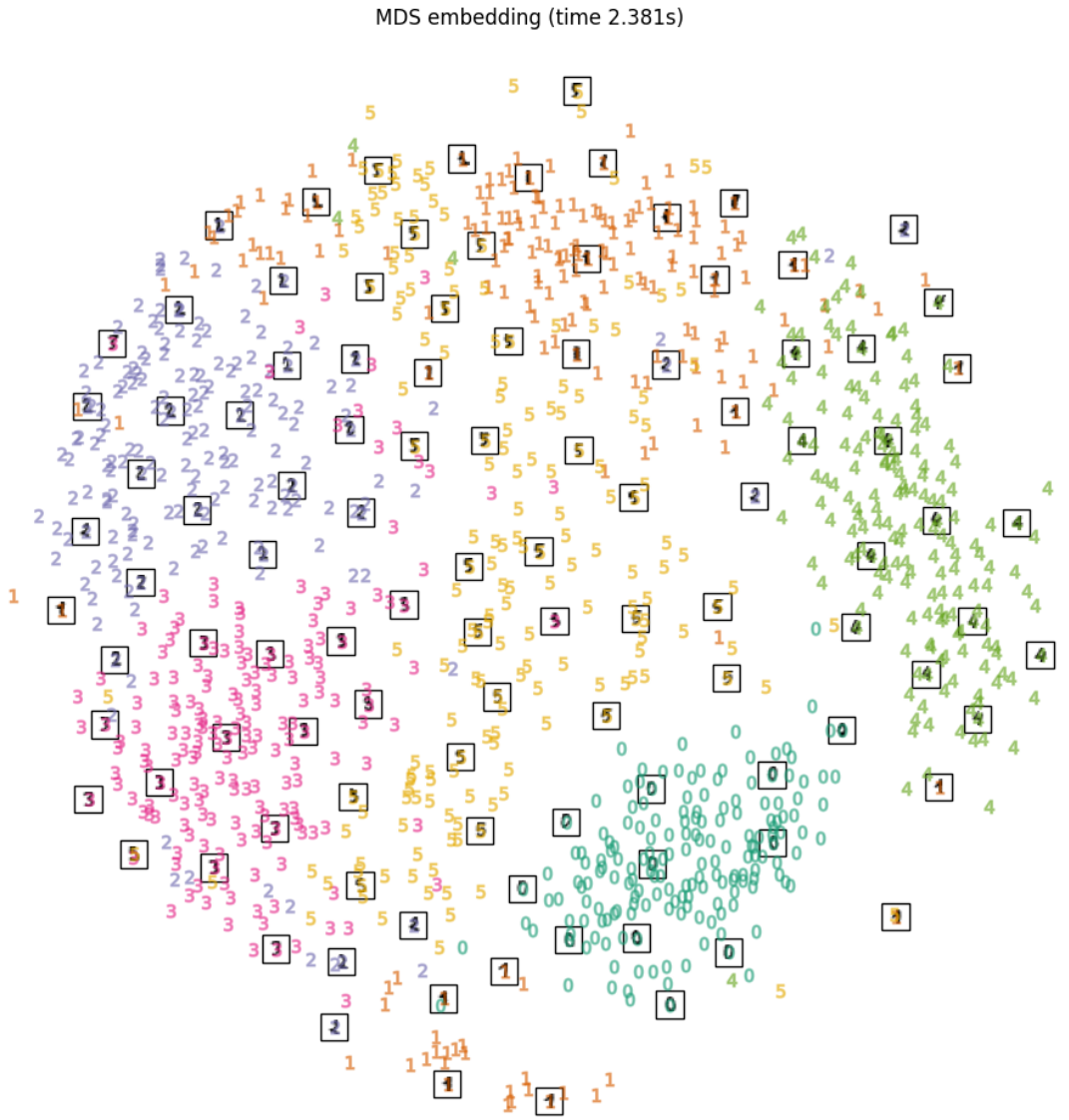


It seems that MDS doesn't show the inherent structure in the data?

- The idea of **preserving high-dimensional distances in low-dimensional embeddings is a bad idea.** (curse of dimensionality)
- Since it needs to compute all pairwise distances between points it takes a very long time to compute for a large number of points.

Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

Why does MDS fail on MNIST?



It seems that MDS doesn't show the inherent structure in the data?

- The idea of **preserving high-dimensional distances in low-dimensional embeddings is a bad idea.** (curse of dimensionality)
- Since it needs to compute all pairwise distances between points it takes a very long time to compute for a large number of points.
- Can we do better? **Preserve nearest neighbors instead of preserving distances.**

Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

Multi-Dimensional Scaling (MDS) - Sammon Mapping

MDS can be converted to a non-linear dimensionality reduction technique, by **putting more importance on small distances**. A variant of this is the **Sammon mapping** introduced by John W. Sammon in 1969.

The **Sammon mapping** can be formulated as the following optimization problem:

$$\underset{\mathcal{Y}=\{\mathbf{y}_1,\ldots,\mathbf{y}_M\}}{\operatorname{argmin}} \frac{1}{\sum_{i<j} || \mathbf{x}_i - \mathbf{x}_j ||^2} \sum_{i<j} \frac{(|| \mathbf{x}_i - \mathbf{x}_j ||^2 - || \mathbf{y}_i - \mathbf{y}_j ||^2)^2}{|| \mathbf{x}_i - \mathbf{x}_j ||^2}$$

Isomap: Isometric Mapping

A Global Geometric Framework for Nonlinear Dimensionality Reduction

Joshua B. Tenenbaum and Vin de Silva and John C. Langford

2000

Isomap

- **In a nutshell: Isomap is a non-linear dimensionality reduction** method based on the **spectral theory which tries to preserve the geodesic distances** in the lower dimensional space.

Isomap

- **In a nutshell: Isomap is a non-linear dimensionality reduction** method based on the **spectral theory which tries to preserve the geodesic distances** in the lower dimensional space.
- **Why geodesic distances?**
  - **Geodesic distance** is defined as the **distance between two vertices in a graph**, which amounts to **the number of edges in the shortest path** connecting the two vertices.

Isomap

- **In a nutshell: Isomap is a non-linear dimensionality reduction** method based on the **spectral theory which tries to preserve the geodesic distances** in the lower dimensional space.
- **Why geodesic distances?**
  - **Geodesic distance** is defined as the **distance between two vertices in a graph**, which amounts to **the number of edges in the shortest path** connecting the two vertices.

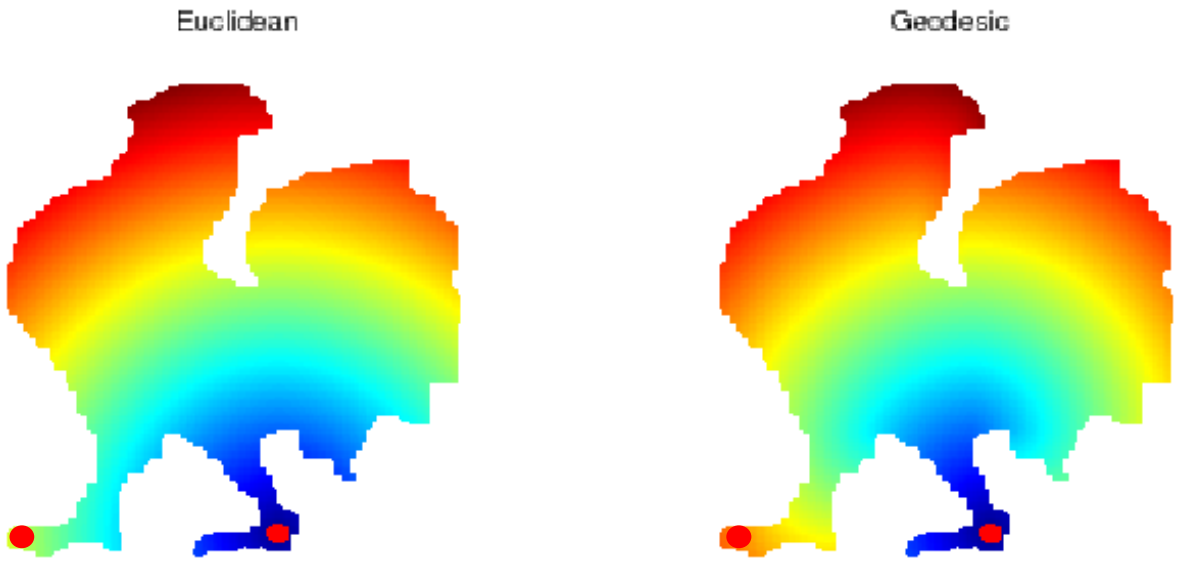


Image Source: https://medium.com/data-science-in-your-pocket/dimension-reduction-using-isomap-72ead0411dec

Isomap

- **In a nutshell: Isomap is a non-linear dimensionality reduction** method based on the **spectral theory which tries to preserve the geodesic distances** in the lower dimensional space.
- **Why geodesic distances?**
  - **Geodesic distance** is defined as the **distance between two vertices in a graph**, which amounts to **the number of edges in the shortest path** connecting the two vertices.

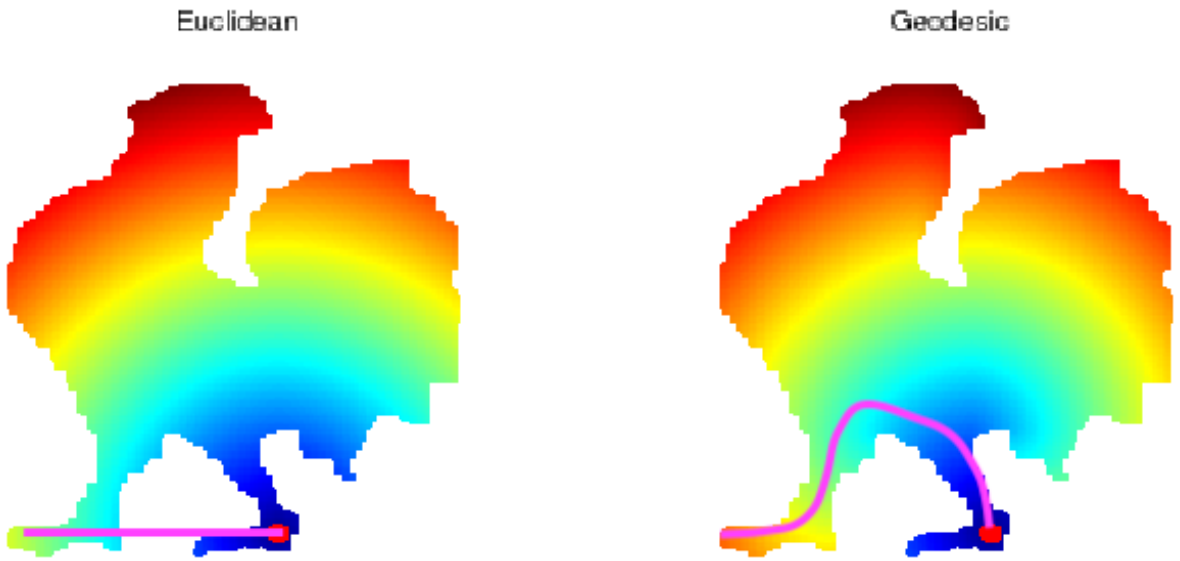
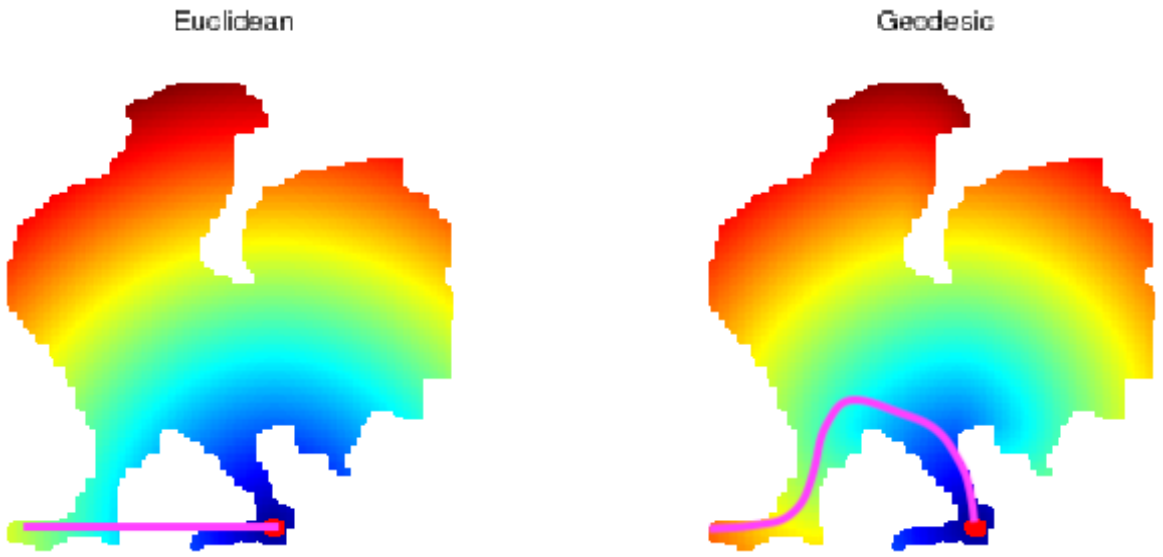


Image Source: https://medium.com/data-science-in-your-pocket/dimension-reduction-using-isomap-72ead0411dec

Isomap

- **In a nutshell: Isomap is a non-linear dimensionality reduction** method based on the **spectral theory which tries to preserve the geodesic distances** in the lower dimensional space.
- **Why geodesic distances?**
  - **Geodesic distance** is defined as the **distance between two vertices in a graph**, which amounts to **the number of edges in the shortest path** connecting the two vertices.



The Euclidean distance ignores the shape of the object when finding a path from one point to another.

Image Source: https://medium.com/data-science-in-your-pocket/dimension-reduction-using-isomap-72ead0411dec

Isomap – Algorithm Description

Isomap starts by creating a **neighborhood** graph. Using **this graph it estimates the geodesic distance between all pairs of points**. And then, t**hrough eigenvalue decomposition of the geodesic distance matrix**, it finds t**he low dimensional embedding of the dataset.**

Isomap – Algorithm Description

Isomap starts by creating a **neighborhood** graph. Using **this graph it estimates the geodesic distance between all pairs of points**. And then, t**hrough eigenvalue decomposition of the geodesic distance matrix**, it finds t**he low dimensional embedding of the dataset.**

Steps:

- **For each point in the dataset, determine its neighbors** (e.g using KNN) and form a neighborhood/adjacency graph.

Isomap – Algorithm Description

Isomap starts by creating a **neighborhood** graph. Using **this graph it estimates the geodesic distance between all pairs of points**. And then, t**hrough eigenvalue decomposition of the geodesic distance matrix**, it finds t**he low dimensional embedding of the dataset.**

Steps:

- **For each point in the dataset, determine its neighbors** (e.g using KNN) and form a neighborhood/adjacency graph.
- **Compute the geodesic distance**, namely the shortest path between two nodes in the graph (e.g. using Dijkstra's algorithm or Floyd–Warshall algorithm)

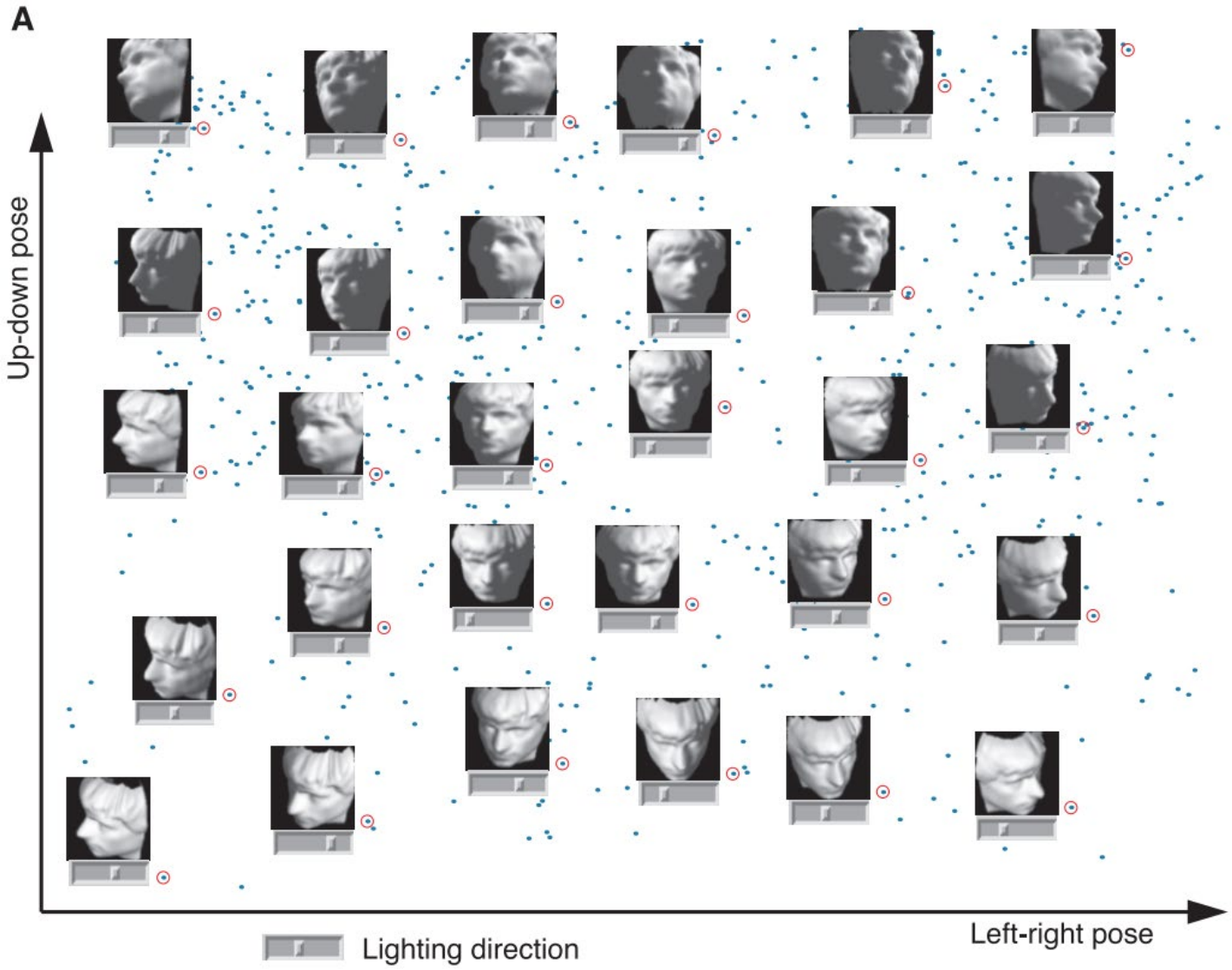
Isomap – Algorithm Description

Isomap starts by creating a **neighborhood** graph. Using **this graph it estimates the geodesic distance between all pairs of points**. And then, t**hrough eigenvalue decomposition of the geodesic distance matrix**, it finds t**he low dimensional embedding of the dataset.**

Steps:

- **For each point in the dataset, determine its neighbors** (e.g using KNN) and form a neighborhood/adjacency graph.
- **Compute the geodesic distance**, namely the shortest path between two nodes in the graph (e.g. using Dijkstra's algorithm or Floyd–Warshall algorithm)
- We **can now perform MDS** (Multidimensional Scaling) on the dissimilarity matrix formed from the above calculated geodesic distances to find a lower-dimensional manifold.

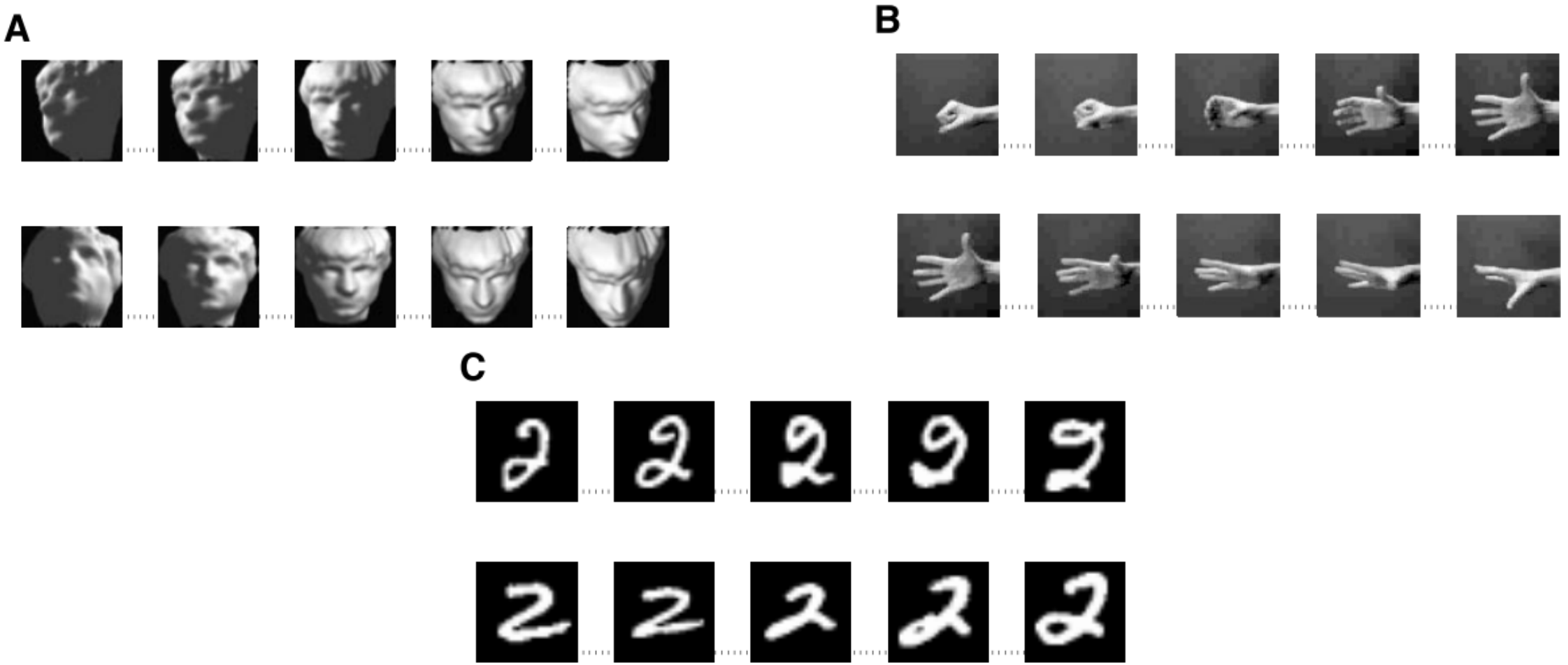
Isomap - Results



The input consists of **a sequence of 4096-D vectors, representing the brightness values** of 64x64 images of faces **rendered with different poses (two axis of variation) and lighting directions.**

Image Source: A Global Geometric Framework for Nonlinear Dimensionality Reduction, by Tenenbaum et al.

Isomap – Results



Interpolations across straight lines in the ISOMAP coordinate space.

Image Source: A Global Geometric Framework for Nonlinear Dimensionality Reduction, by Tenenbaum et al.

Isomap - MNIST

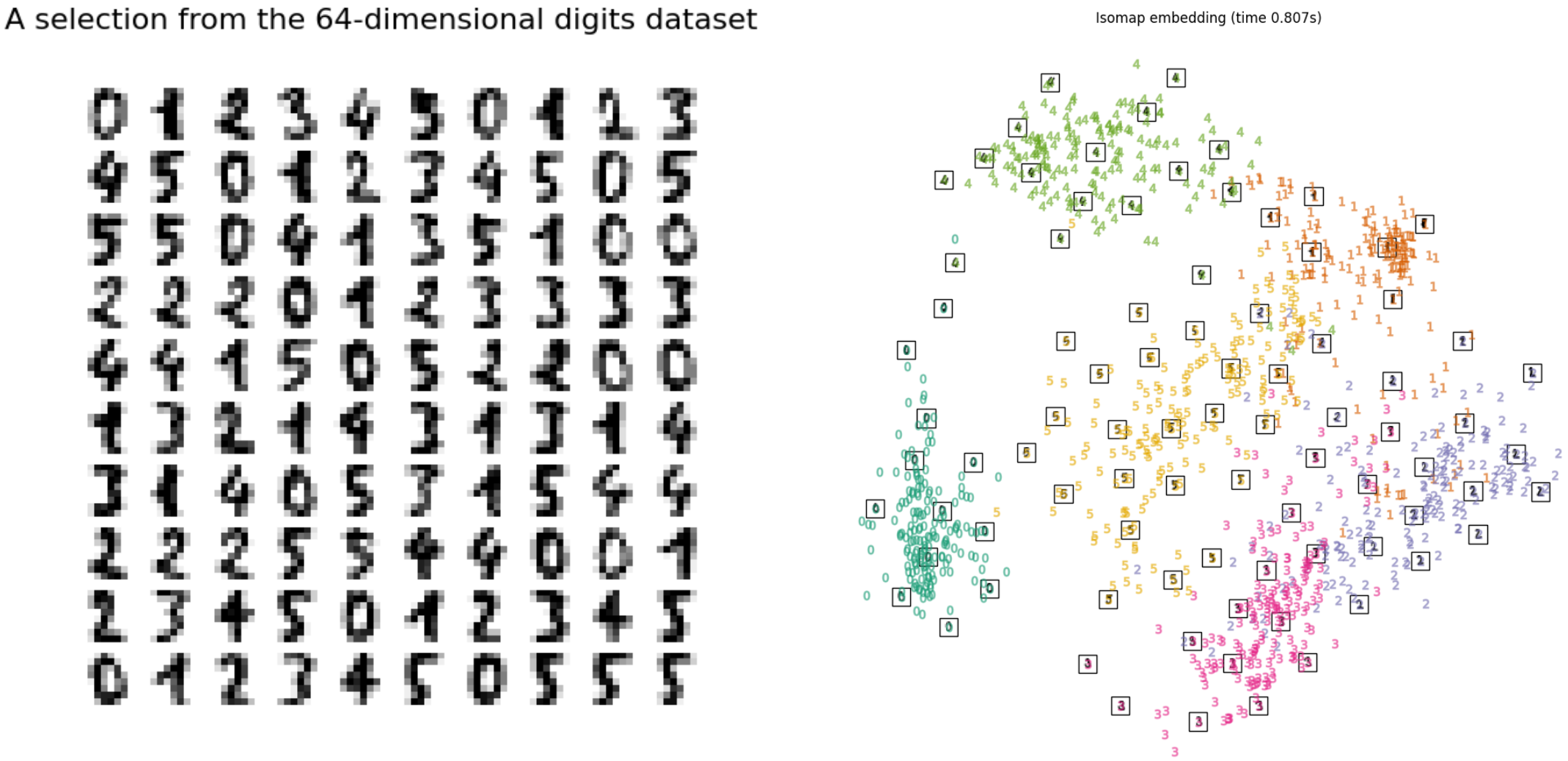


Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

LLE: Locally Linear Embeddings

Nonlinear Dimensionality Reduction by Locally Linear Embedding

Sam T. Roweis and Lawrence K. Saul

2000

Locally Linear Embeddings

- **In a nutshell: Previous methods based on Multi-Dimensional Scaling (MDS) seek to compute embeddings that attempt to preserve pairwise distances between data points**; these distances are measured along straight lines or, in more sophisticated usages of MDS such as Isomap, along shortest paths confined to the manifold of observed inputs. In contrast, locally linear embedding (LLE), **try to eliminate the need to estimate pairwise distances between data points**. Unlike previous methods, **LLE recovers global nonlinear structure from locally linear fits.**

Locally Linear Embeddings

Key Idea: Given a dataset that consist of N real-valued vectors $\mathcal{X} = \{\mathbf{x}_1, \ldots \mathbf{x}_N\}$, where $\mathbf{x}_i \in \mathbb{R}^D$, sampled from some underlying manifold. Provided there is sufficient data (such that the manifold is well-sampled), **we expect each data point and its neighbors to lie on or close to a locally linear patch of the manifold.** The local geometry of these patches can be characterized by the linear coefficients that reconstruct each data point from its neighbors.

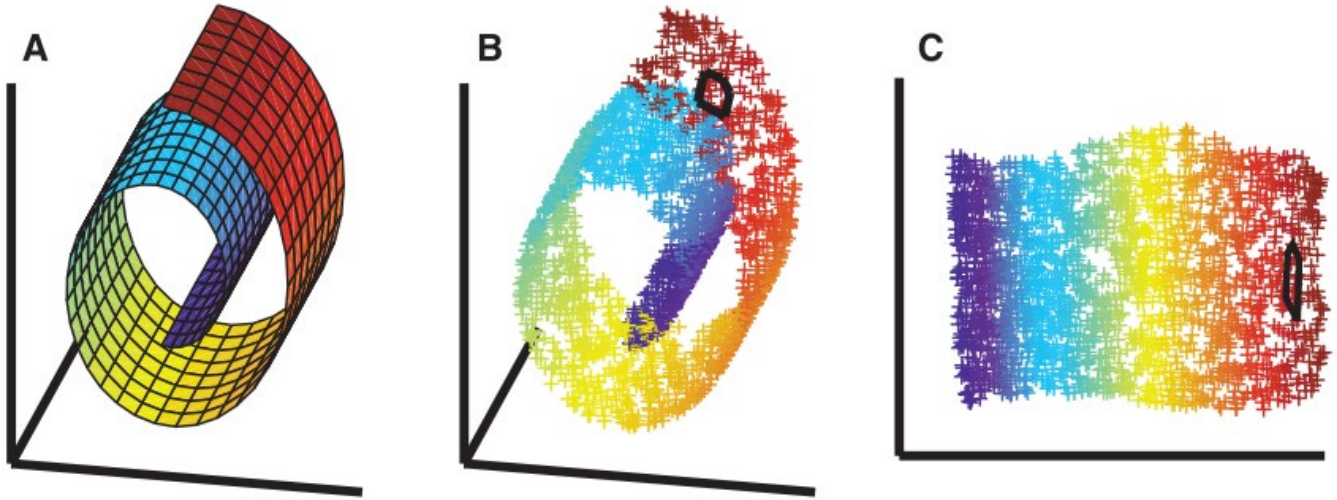
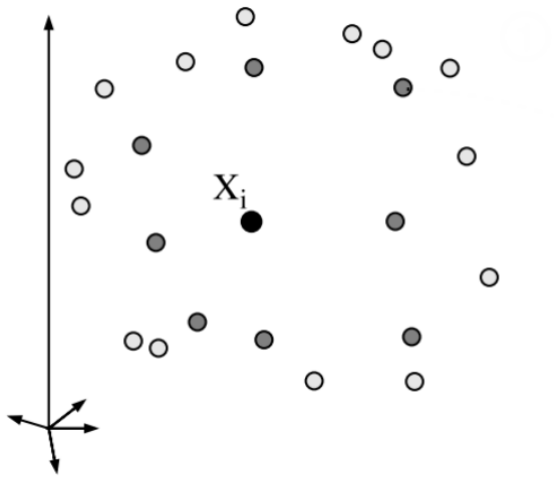


Image Source: Nonlinear Dimensionality Reduction by Locally Linear Embedding, by Roweis et al.

Locally Linear Embeddings – Algorithm Description

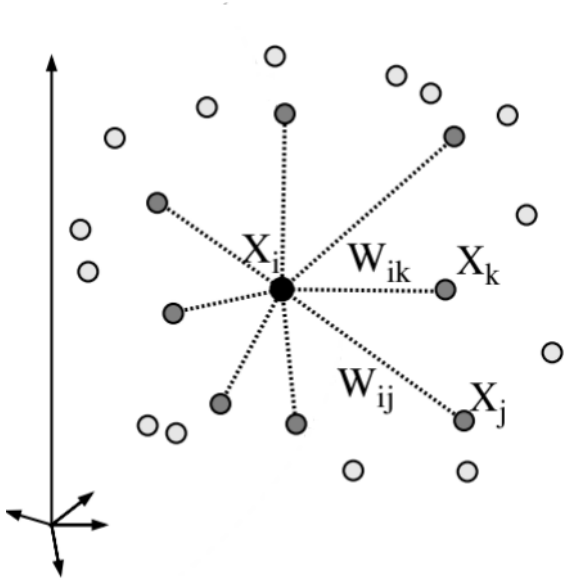
- For each datapoint $\mathbf{x}_i \in \mathcal{X}$ **find its neighbors** (for example using K-NN algorithm)



Locally Linear Embeddings – Algorithm Description

- For each datapoint $\mathbf{x}_i \in \mathcal{X}$ **find its neighbors** (for example using K-NN algorithm)
- **Compute the weights** $\mathbf{w}_{ij}$ that best linearly reconstruct $\mathbf{x}_i$ from its neighbors, by solving the constrained least-squares problem of

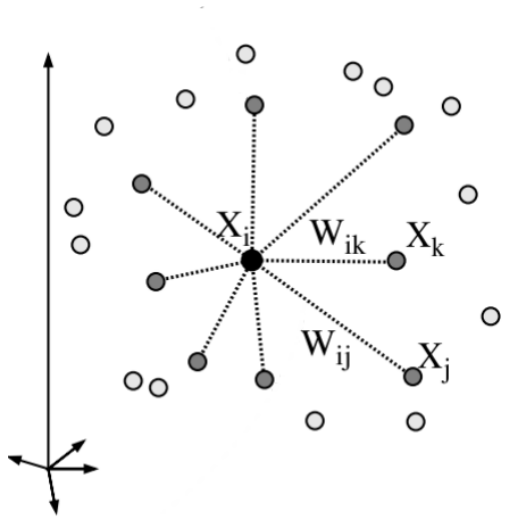
$$\epsilon(\mathbf{W}) = \sum_{i=1}^{N} || \mathbf{x}_i - \sum_{j=1}^{K} \mathbf{w}_{ij}\mathbf{x}_j ||^2$$



Locally Linear Embeddings – Finding the Weights

To compute the weights, we minimize the cost funcion $\epsilon(W)$

$$\epsilon(\mathbf{W}) = \sum_{i=1}^{N} \min_{\sum_j \mathbf{w}_{ij}=1} || \mathbf{x}_i - \sum_{j=1}^{K} \mathbf{w}_{ij}\mathbf{x}_j ||^2$$



subject to two constraints:

- Each data point $\mathbf{x}_i$ **should be reconstructed only from its neighbors**, namely $\mathbf{w}_{ij} = 0$, if $\mathbf{x}_j$ does not belong to the set of neighbors of $\mathbf{x}_i$.
- The rows of the weight matrix sum to 1, namely $\sum_j \mathbf{w}_{ij} = 1$

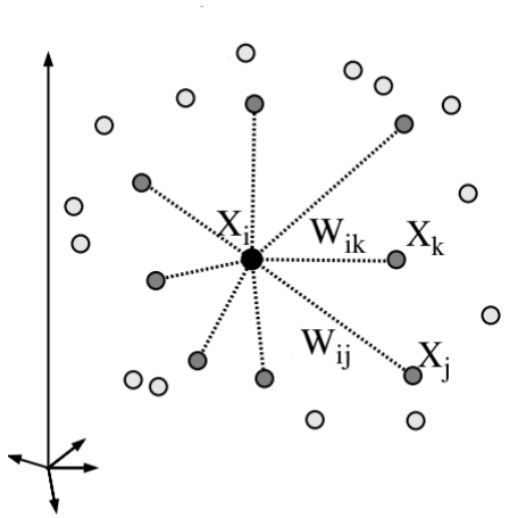
The optimal weights of these contraints can be found by **solving a least-squares problem**.

The weight $\mathbf{w}_ij$ expresses the contribution of the *j*-th point to the reconstruction of the *i*-th point.

Locally Linear Embeddings – Finding the Weights

On solving the least-squares problem: Let $\mathbf{x}$ be a data point with K neighbors $\mathbf{h}_j$, whose reconstruction weights sum to 1.0. The reconstruction error $|| \mathbf{x} - \sum_{j=1}^{K} \mathbf{w}_j \mathbf{h}_j ||^2$ can be minimized in three steps:

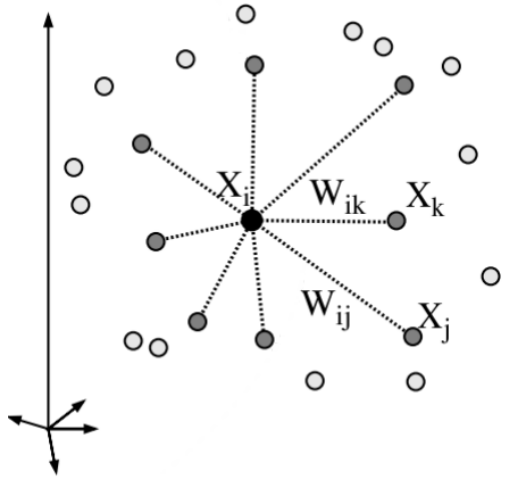
- **Compute the neighborhood correlation matrix** $\mathbf{C}_{jk} = \mathbf{h}_j \mathbf{h}_k$ and its inverse $\mathbf{C}_{jk}^{-1}$
- **Compute the Lagrange multiplier**, $\lambda = \alpha / \beta$, that enforces the sum-to-one constraint on the wieghts, where $\alpha = 1 - \sum_{jk} \mathbf{C}_{jk}^{-1} (\mathbf{x}\mathbf{h}_k)$ and $\beta = \sum_{jk} \mathbf{C}_{jk}^{-1}$.
- **Compute the reconstruction weights** $\mathbf{w}_j = \sum_k \mathbf{C}_{jk}^{-1} (\mathbf{x}\mathbf{h}_k + \lambda)$



Locally Linear Embeddings – Finding the Weights

On solving the least-squares problem: Let $\mathbf{x}$ be a data point with K neighbors $\mathbf{h}_j$, whose reconstruction weights sum to 1.0. The reconstruction error $|| \mathbf{x} - \sum_{j=1}^{K} \mathbf{w}_j \mathbf{h}_j ||^2$ can be minimized in three steps:

- **Compute the neighborhood correlation matrix** $\mathbf{C}_{jk} = \mathbf{h}_j \mathbf{h}_k$ and its inverse $\mathbf{C}_{jk}^{-1}$
- **Compute the Lagrange multiplier**, $\lambda = \alpha/\beta$, that enforces the sum-to-one constraint on the wieghts, where $\alpha = 1 - \sum_{jk} \mathbf{C}_{jk}^{-1}(\mathbf{x}\mathbf{h}_k)$ and $\beta = \sum_{jk} \mathbf{C}_{jk}^{-1}$.
- **Compute the reconstruction weights** $\mathbf{w}_j = \sum_k \mathbf{C}_{jk}^{-1}(\mathbf{x}\mathbf{h}_k + \lambda)$



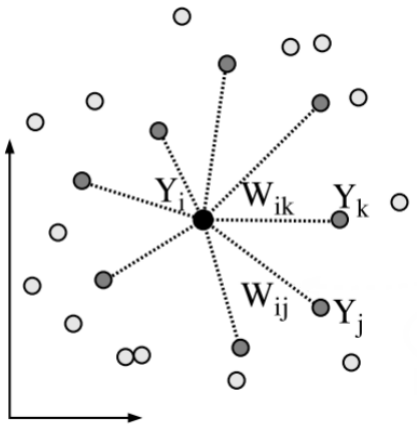
- For any data point, the weights that minimize the above reconstruction error **are invariant to rotations, rescalings and translations of that data point and its neighbors**.
- By design, **the reconstruction weights reflect intrinsic geometric properties** of the data that are invariant to the above transformations. Thus, **we expect their characterization of local geometry in the original data space to be equally valid for local patches on the manifold**.

Locally Linear Embeddings – Finding the Embeddings

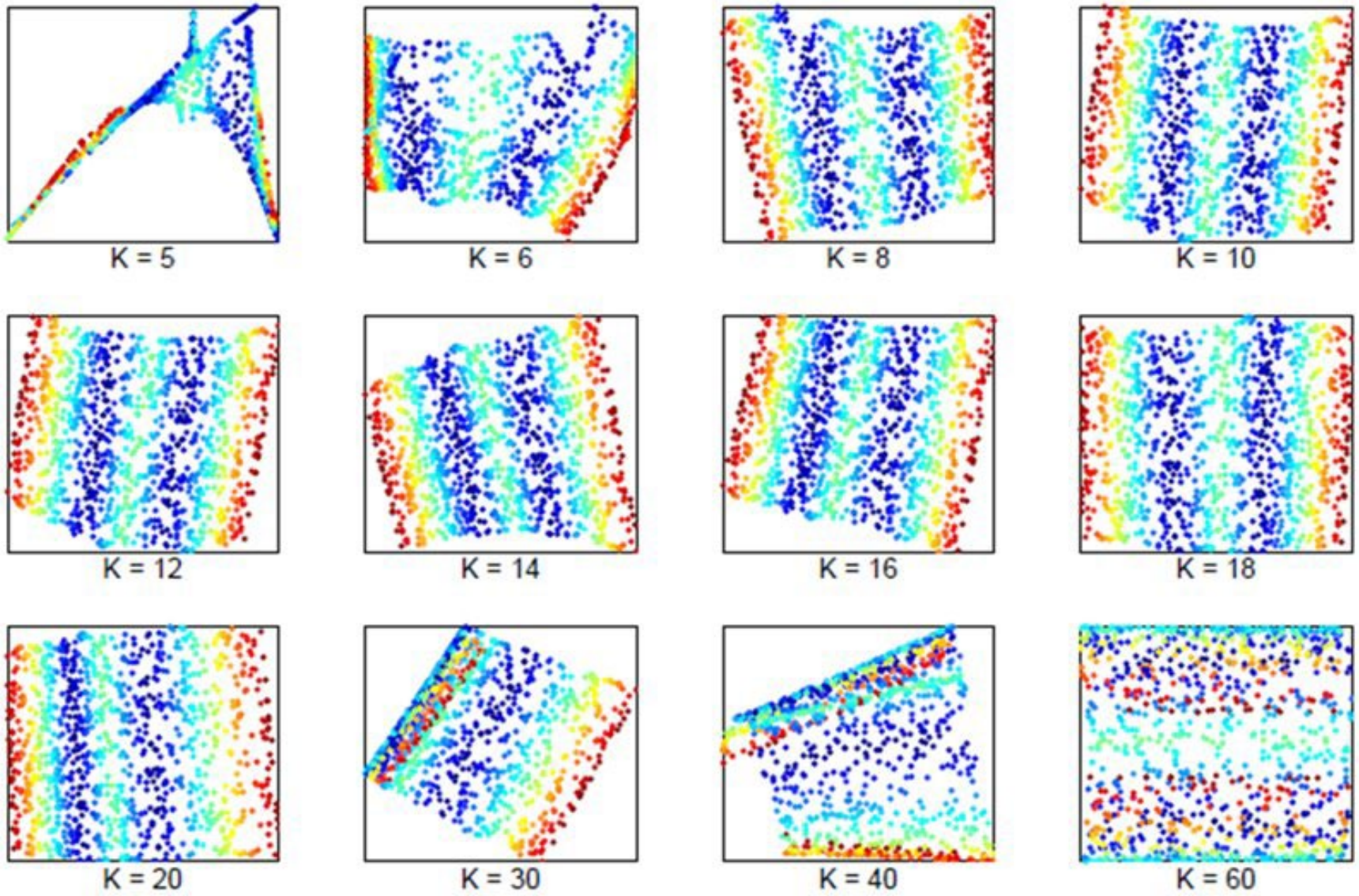
Each high-dimensional observation $\mathbf{x}_i$ is mapped to a low-dimensional vector $\mathbf{y}_i$ representing global internal coordinates on the manifold. This is done by choosing the d-dimensional coordinates $\mathbf{y}_i$ to minimize the embedding cost function

$$\min_{\mathbf{y}_1 \cdots \mathbf{y}_N} \sum_{i=1}^{N} || \mathbf{y}_i - \sum_{j=1}^{K} \mathbf{w}_{ij} \mathbf{y}_j ||^2$$

Note that in this cost function we fix the weights $\mathbf{w}_{ij}$ while optimizing the coordinates $\mathcal{Y}$. **The cost function can be minimized (subject to constraints) by solving a sparse $N \times N$ eigenvalue problem.**

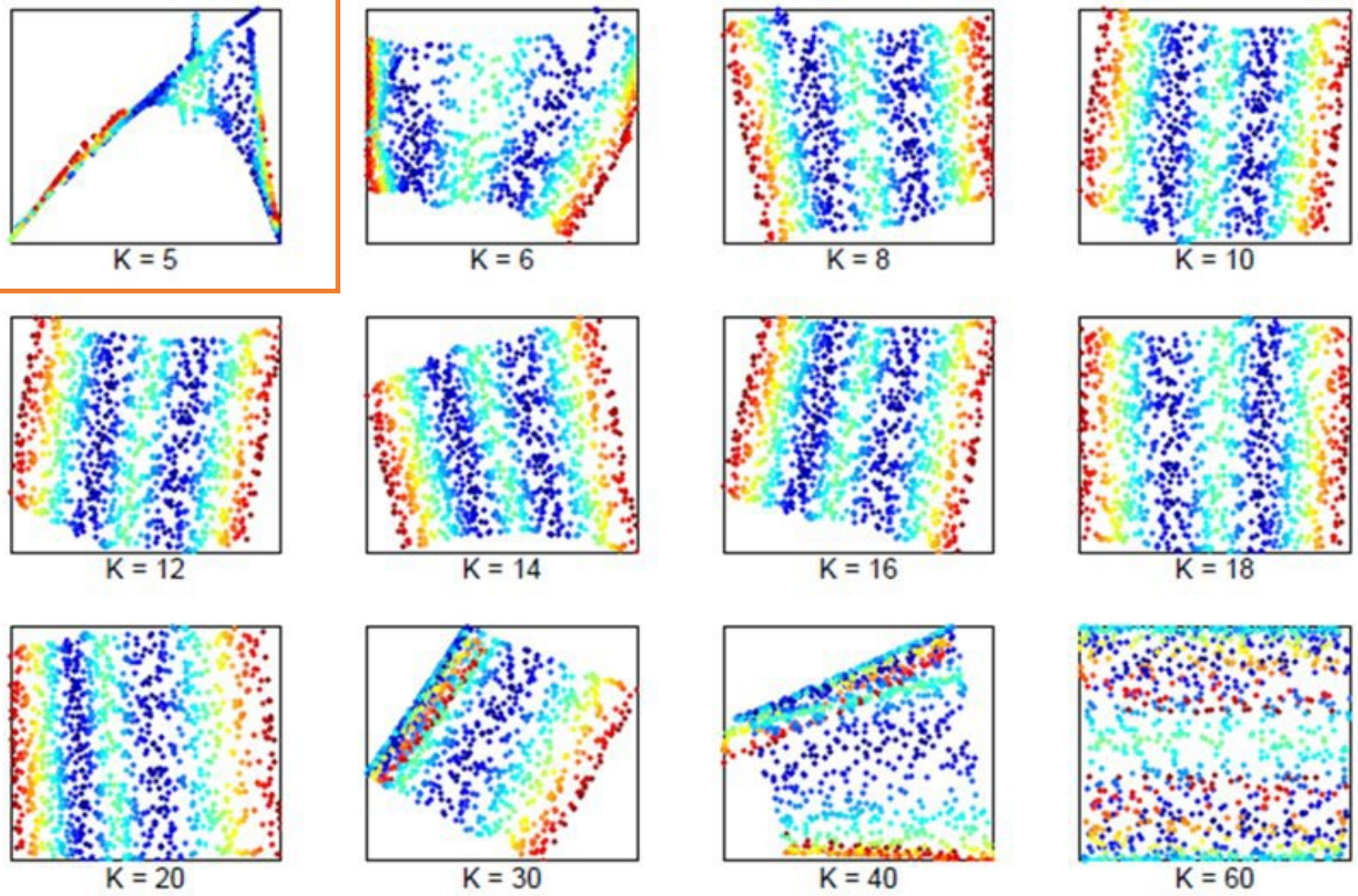


Locally Linear Embeddings – Neighboorhood size

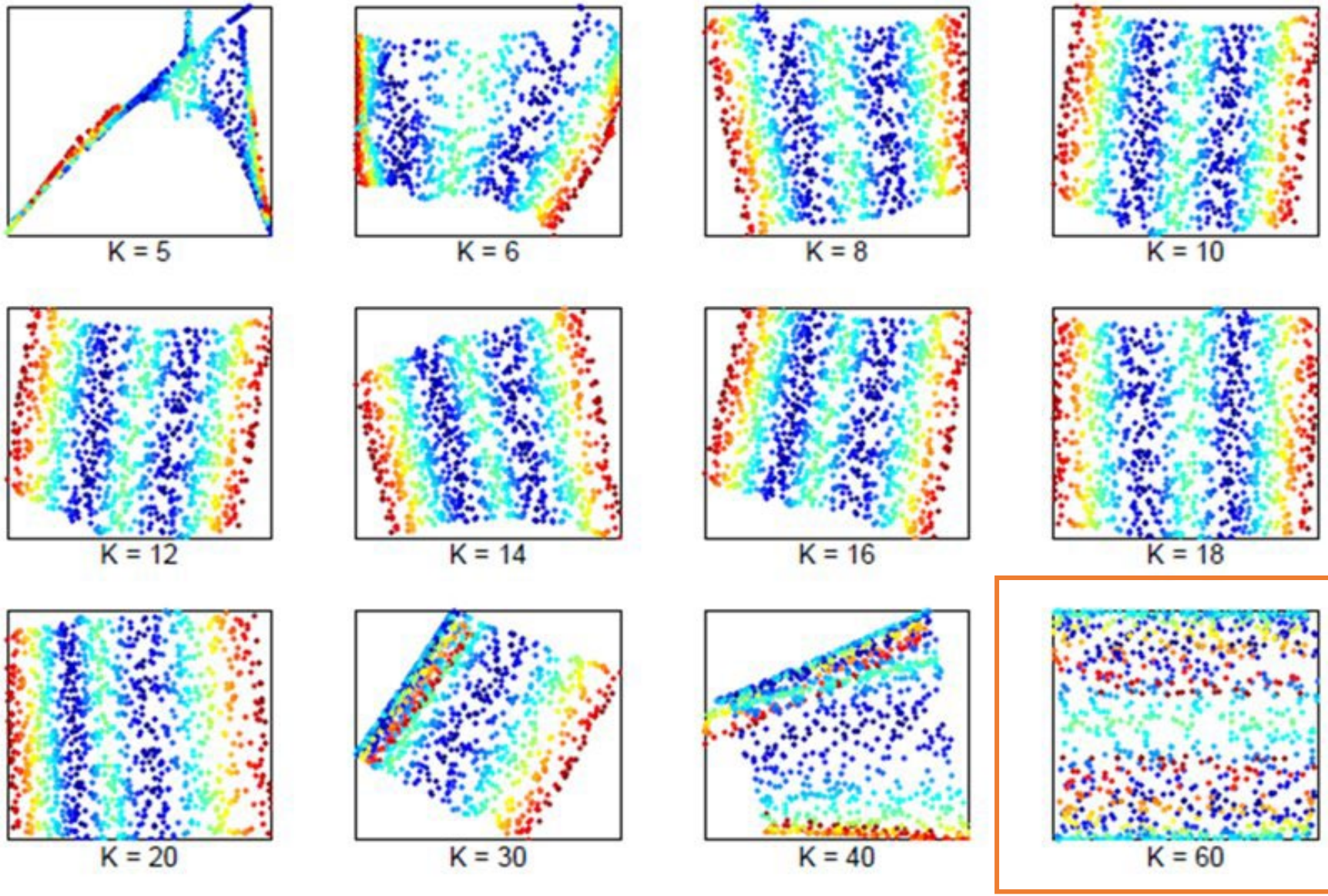


Locally Linear Embeddings – Neighboorhood size

Small number of neighbors can result in **large reconstruction errors**

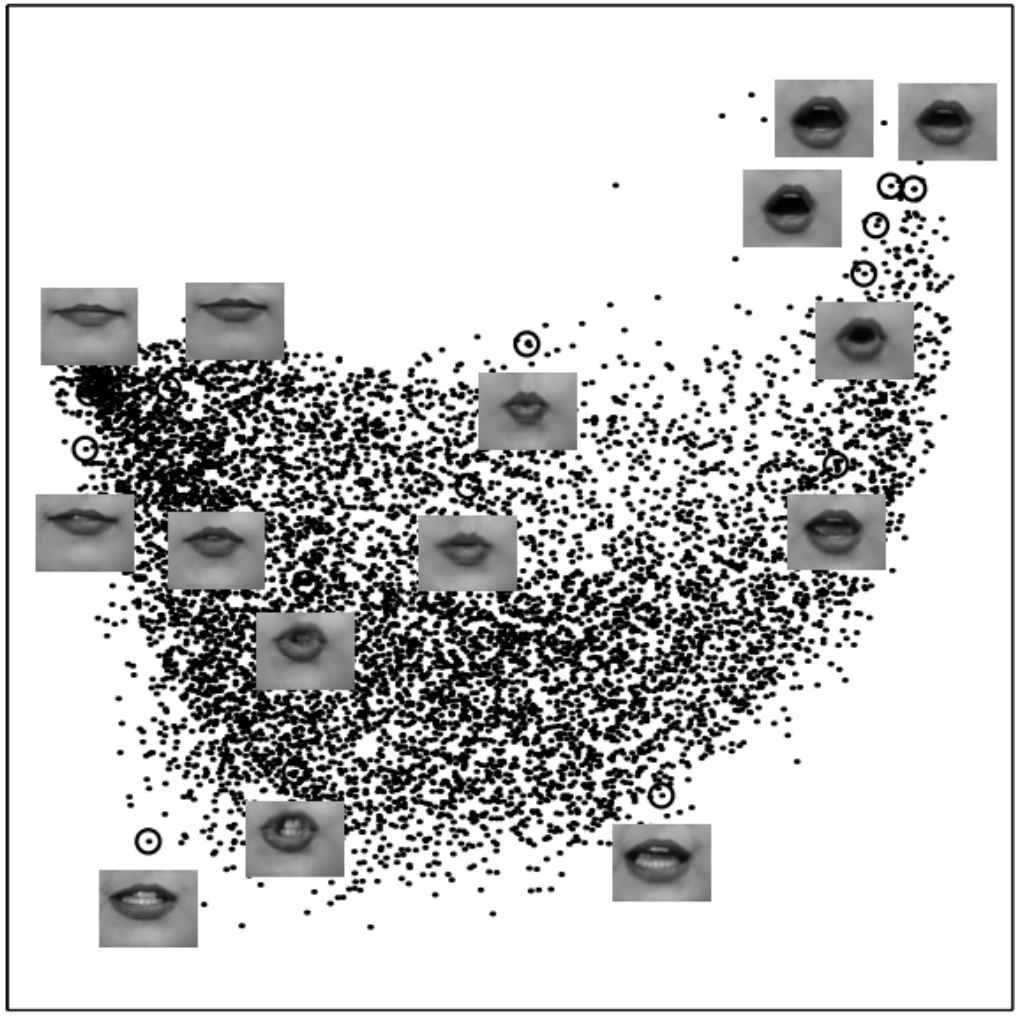


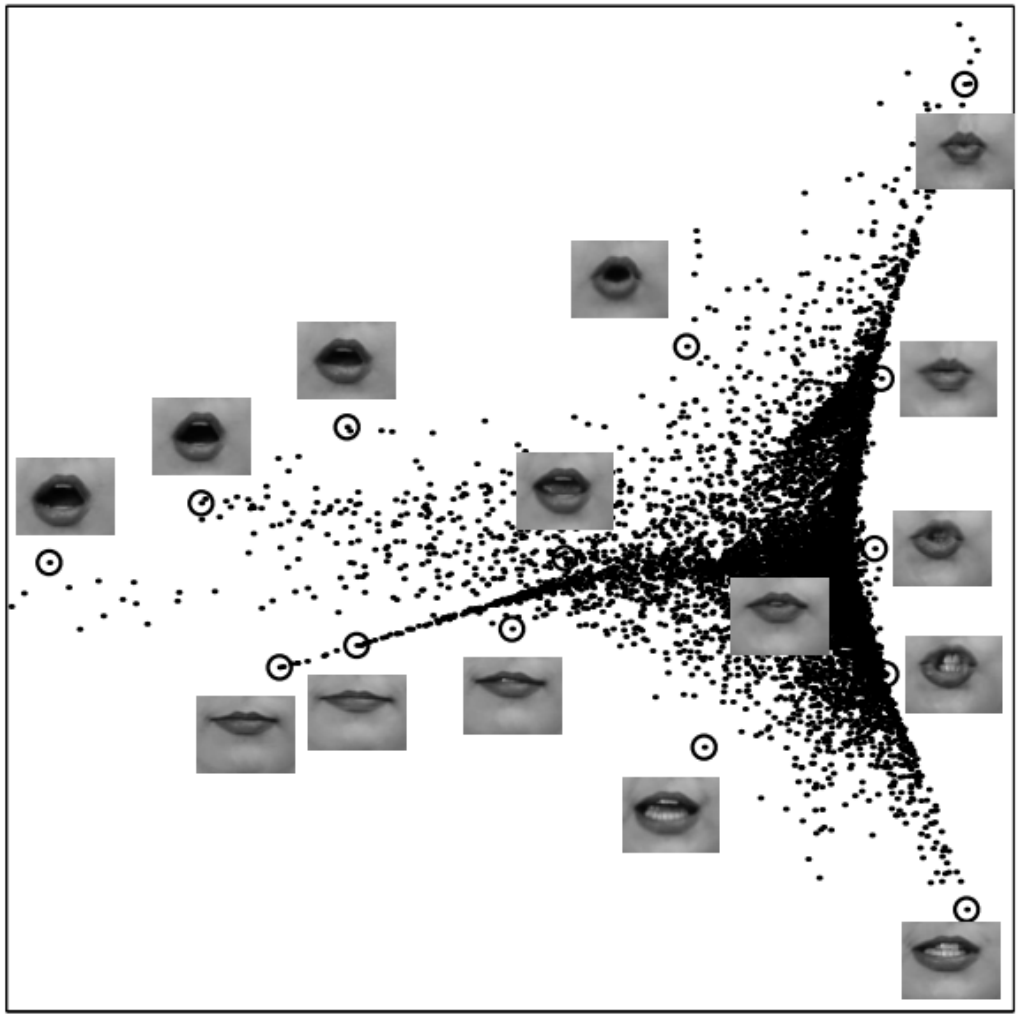
Locally Linear Embeddings – Neighboorhood size



Large number of neighbors results in the **Euclidean distance in the large neighborhood to be incorrect**

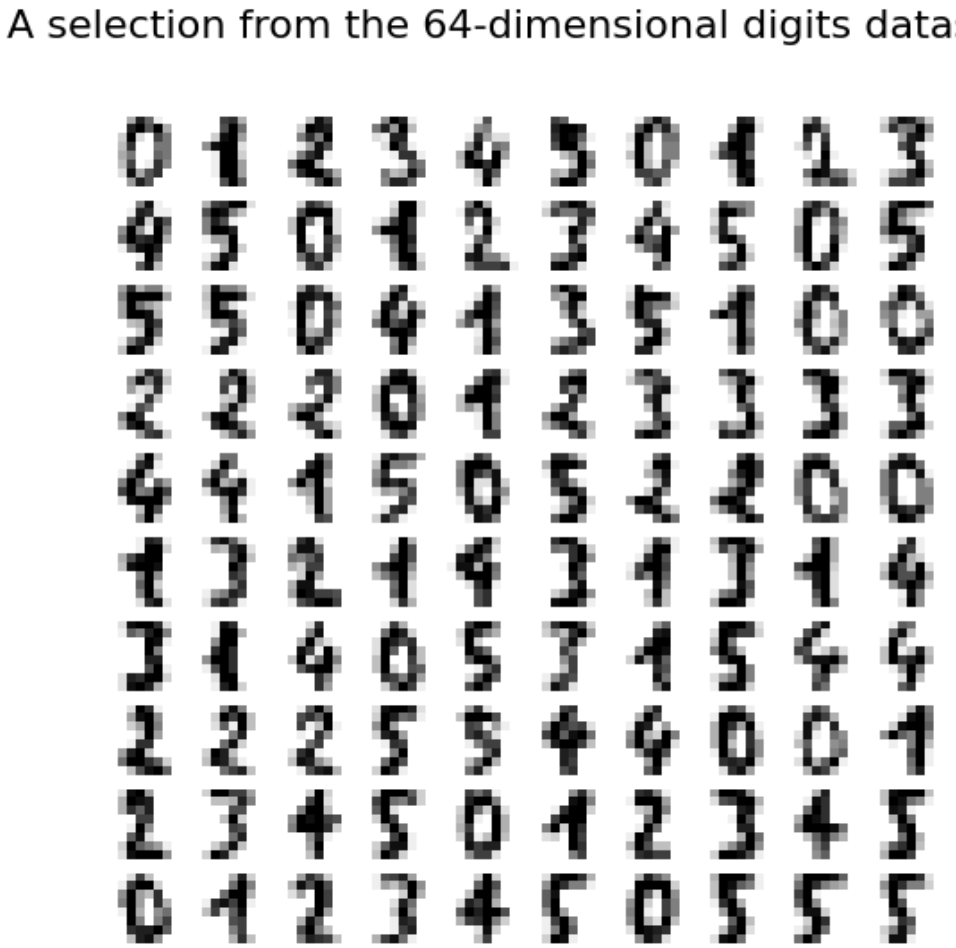
Locally Linear Embeddings – Results





Images of lips mapped into the embedding space described by the first two coordinates of **PCA (left)** and **LLE (right)**.

Locally Linear Embeddings - MNIST



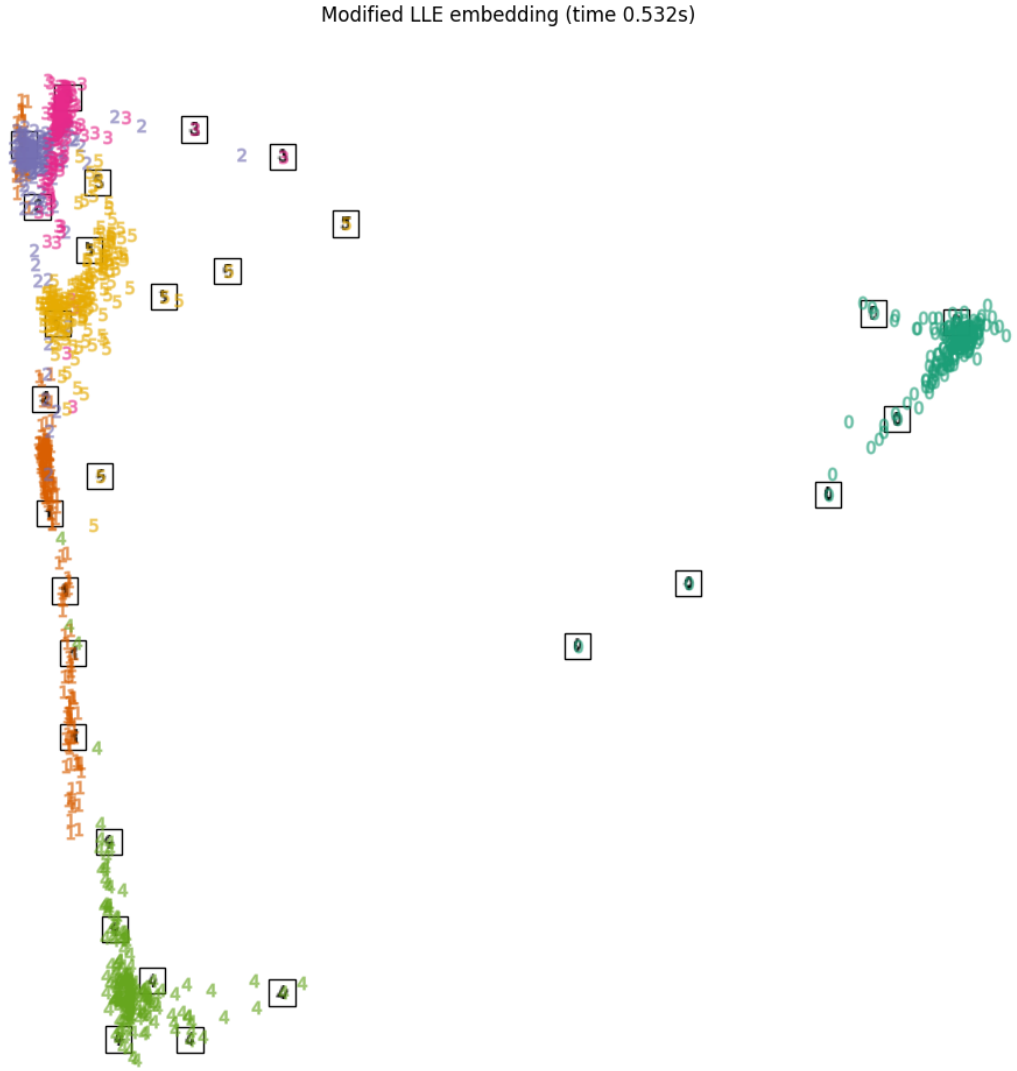


Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

Laplacian Eigenmaps

Laplacian Eigenmaps for Dimensionality Reduction and Data Representation

Mikhail Belkin and Partha Niyogi

2002

Laplacian Eigenmaps

- **In a nutshell:** Laplacian Eigenmaps is an approach that **builds a graph incorporating neighborhood information of the data set**. Using **the notion of the Laplacian of the graph**, it can compute a low-dimensional representation of the data set **that optimally preserves local neighborhood information.**

Laplacian Eigenmaps

- **In a nutshell:** Laplacian Eigenmaps is an approach that **builds a graph incorporating neighborhood information of the data set**. Using **the notion of the Laplacian of the graph**, it can compute a low-dimensional representation of the data set **that optimally preserves local neighborhood information.**

- The algorithm procedure consists of the three main steps:
  - **Build a neighborhood graph** from the given data
  - **Compute the shortest-path distances** along the graph
  - **Apply MDS to find a low-dimensional embedding**

Laplacian Eigenmaps

- **In a nutshell:** Laplacian Eigenmaps is an approach that **builds a graph incorporating neighborhood information of the data set**. Using **the notion of the Laplacian of the graph**, it can compute a low-dimensional representation of the data set **that optimally preserves local neighborhood information.**

- The algorithm procedure consists of the three main steps:
  - **Build a neighborhood graph** from the given data
  - **Compute the shortest-path distances** along the graph
  - **Apply MDS to find a low-dimensional embedding**

- The goal of Isomap is to directly preserve the global (nonlinear) geometry. In contrast, **Laplacian Eigenmaps focus on preserving the local geometry - nearby points in the original space remain nearby in the reduced space.**

Laplacian Eigenmaps – Algorithmic Overview

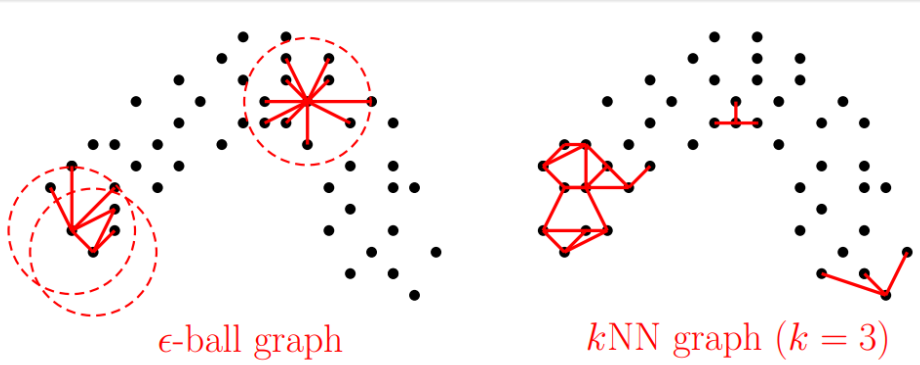
Given a set of points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_K\}$ in $\mathbb{R}^D$, construct a weighted graph with K nodes, one for each point, and a set of edges connecting neighboring points. The **embedding map can be provided by computing the eigenvectors of the graph Laplacian.**

Laplacian Eigenmaps – Algorithmic Overview

Given a set of points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_K\}$ in $\mathbb{R}^D$, construct a weighted graph with K nodes, one for each point, and a set of edges connecting neighboring points. The **embedding map can be provided by computing the eigenvectors of the graph Laplacian.**

Neighborhood Graph Contstruction: The first step of Laplacian Eigenmaps is to build a neighborhood graph G from the given data $\mathcal{X}$ **by connecting only "nearby" points, where nearby is defined in one of the following ways**:

- ϵ-**ball approach**: Two points $\mathbf{x}_i, \mathbf{x}_j$ are nearby if $|| \mathbf{x}_i - \mathbf{x}_j ||^2 \leq \epsilon$,
- **kNN approach**: Two points $\mathbf{x}_i, \mathbf{x}_j$ are nearby if one is among the k nearest neighbors of the other.



Laplacian Eigenmaps – Algorithmic Overview

Given a set of points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_K\}$ in $\mathbb{R}^D$, construct a weighted graph with K nodes, one for each point, and a set of edges connecting neighboring points. The **embedding map can be provided by computing the eigenvectors of the graph Laplacian.**

Neighborhood Graph Contstruction: The first step of Laplacian Eigenmaps is to build a neighborhood graph G from the given data $\mathcal{X}$ **by connecting only "nearby" points, where nearby is defined in one of the following ways**:

- ϵ-**ball approach**: Two points $\mathbf{x}_i, \mathbf{x}_j$ are nearby if $|| \mathbf{x}_i - \mathbf{x}_j ||^2 \leq \epsilon$,
- **kNN approach**: Two points $\mathbf{x}_i, \mathbf{x}_j$ are nearby if one is among the k nearest neighbors of the other.

Choosing the Weights: The Euclidean distances between nearby points are transformed to similarity scores (weights) in one of the following ways:

- **Simple-minded**: $\mathbf{w}_{ij} = 1$ if there is an edge between $\mathbf{x}_i$ and $\mathbf{x}_j$. Otherwise $\mathbf{w}_{ij} = 0$.
- **Gaussian Weights**: $\mathbf{w}_{ij} = exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{t})$ if there is an edge between $\mathbf{x}_i$ and $\mathbf{x}_j$. Otherwise $\mathbf{w}_{ij} = 0$

Laplacian Eigenmaps – Algorithmic Overview

Find the Eigenmaps: Compute eigenvalues and eigenvectors for the generalized eigenvector problem $L\mathbf{y} = \lambda D\mathbf{y}$.

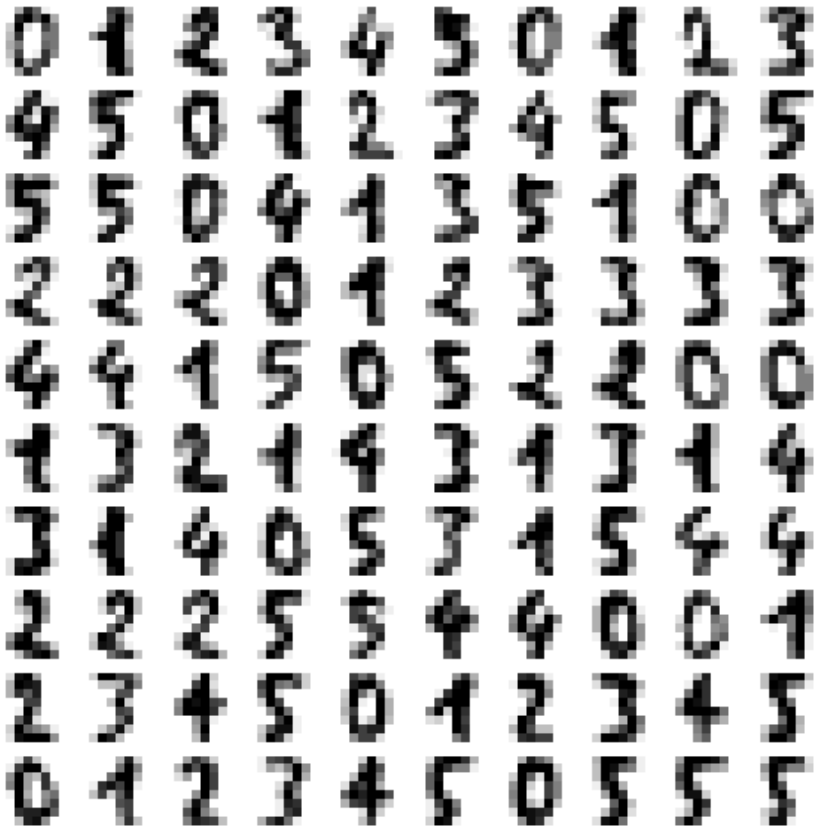
- Construct the **Laplacian matrix** $L = D - W$, where D is **diagonal weight matrix**, and its entries are column (or row, since W is symmetric) sums of W, $D_{ii} = \sum_j \mathbf{w}_{ji}$
- Consider the **problem of mapping weighted graph G into a line** so that the connected nodes stay as close as possible. To find $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$ we need to minimize $\sum_{ij} || \mathbf{y}_i - \mathbf{y}_j ||^2 \mathbf{w}_{ij}$, which turns out to be:

$$1/2 \sum_{ij} || \mathbf{y}_i - \mathbf{y}_j ||^2 \mathbf{w}_{ij} = \mathbf{y}^T L\mathbf{y}$$

- **The solution is provided by the matrix of eigenvectors corresponding to the lowest eigenvalues of the generalized eigenvalue problem.**

Laplacian Eigenmaps - MNIST

A selection from the 64-dimensional digits dataset



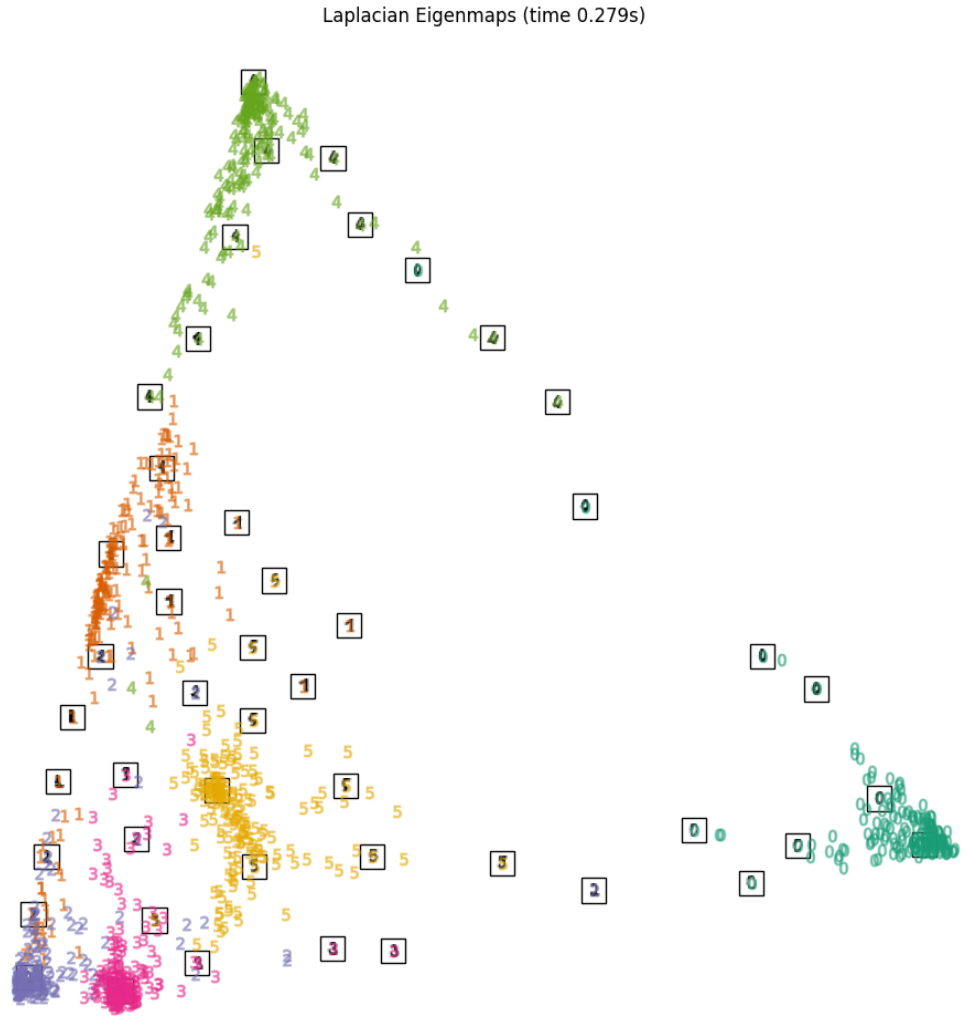


Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

t-SNE: t-distributed Stochastic Neighbor Embedding

Visualizing Data using t-SNE

Laurens van der Maaten and Geoffrey Hinton

2008

t-SNE

- **In a nutshell: Measure pairwise similarities between high-dimensional and low-dimensional data.**

t-SNE

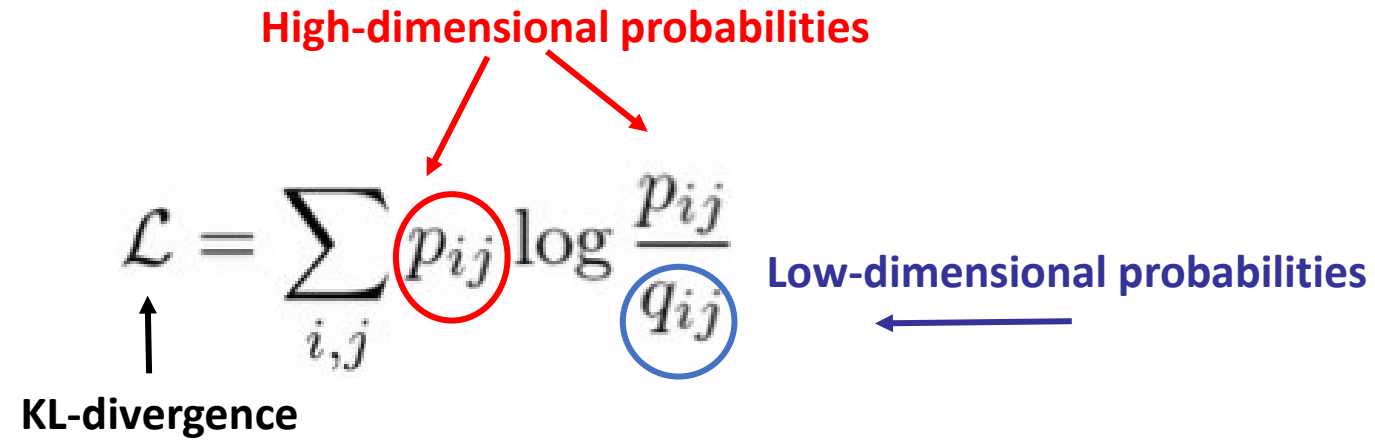
- **In a nutshell: Measure pairwise similarities between high-dimensional and low-dimensional data**.
- It first **defines a probability distribution** over pairs of high-dimensional data in such a way that **similar objects are assigned a higher probability, while dissimilar points are assigned a lower probability**.

t-SNE

- **In a nutshell: Measure pairwise similarities between high-dimensional and low-dimensional data**.
- It first **defines a probability distribution** over pairs of high-dimensional data in such a way that **similar objects are assigned a higher probability, while dissimilar points are assigned a lower probability**.
- Second, it **defines a similar probability distribution over the points in the low-dimensional space.**

t-SNE

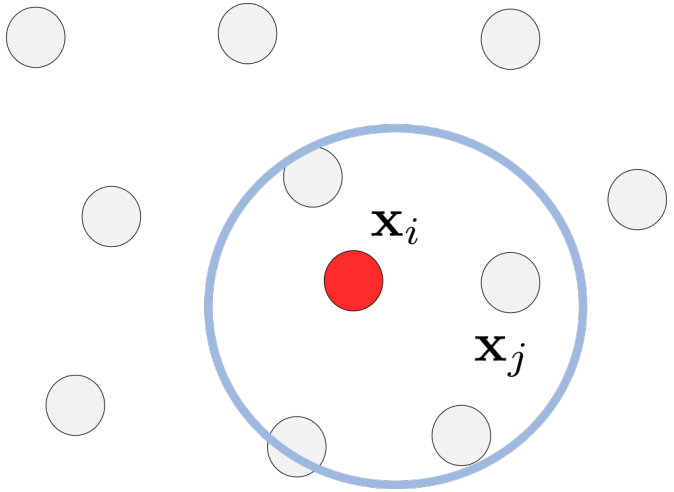
- **In a nutshell: Measure pairwise similarities between high-dimensional and low-dimensional data**.
- It first **defines a probability distribution** over pairs of high-dimensional data in such a way that **similar objects are assigned a higher probability, while dissimilar points are assigned a lower probability**.
- Second, it **defines a similar probability distribution over the points in the low-dimensional space.**
- Finally, **it minimizes the KL divergence between the two distributions with respect to the locations of the points in the map**.



Stochastic Neighborhood Embedding

Neighborhood Embedding Basics:

- Consider the neighborhood around an input data point $\mathbf{x}_i \in \mathbb{R}^d$
- Let as assume that we have a Gaussian distribution centered around $\mathbf{x}_i$
- Then the probability that $\mathbf{x}_i$ **chooses some other datapoint** $\mathbf{x}_j$ as its neighbor **is proportional to the density under this Gaussian**
- A point closer to $\mathbf{x}_i$ will be more likely than one further away



Stochastic Neighborhood Embedding

We want to convert high-dimensional data into conditional probabilities that represent similarities:

- **Similarities of points in high-dimensional space**:

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

Stochastic Neighborhood Embedding

We want to convert high-dimensional data into conditional probabilities that represent similarities:

- **Similarities of points in high-dimensional space**:

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

- **Similarities of points in low-dimensional space**:

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

Stochastic Neighborhood Embedding

We want to convert high-dimensional data into conditional probabilities that represent similarities:

- **Similarities of points in high-dimensional space**:

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

- **Similarities of points in low-dimensional space**:

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

- **Cost function** to be minimized using gradient descent (KL-Divergence):

$$KL(P \,||\, Q) = \sum_i \sum_j p_{j|i} \log\left(\frac{p_{j|i}}{q_{j|i}}\right)$$

Symmetric Stochastic Neighborhood Embedding

We want to minimize the KL-divergence between **joint probability distributions:**

- **Similarities of points in high-dimensional space**:

$$p_{ij} = \frac{exp(-\,||\,\mathbf{x}_i - \mathbf{x}_j\,||^2\,/2\sigma^2)}{\sum_{k \neq i} exp(-\,||\,\mathbf{x}_i - \mathbf{x}_k\,||^2\,/2\sigma^2)}$$

We can **symmetrize** the similarity as follows:

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

Symmetric Stochastic Neighborhood Embedding

We want to minimize the KL-divergence between **joint probability distributions:**

- **Similarities of points in high-dimensional space**:

$$p_{ij} = \frac{exp(-\,||\,\mathbf{x}_i - \mathbf{x}_j\,||^2\,/2\sigma^2)}{\sum_{k \neq i} exp(-\,||\,\mathbf{x}_i - \mathbf{x}_k\,||^2\,/2\sigma^2)}$$

- **Similarities of points in low-dimensional space**:

$$q_{ij} = \frac{exp(-\,||\,\mathbf{y}_i - \mathbf{y}_j\,||^2)}{\sum_{k \neq i} exp(-\,||\,\mathbf{y}_i - \mathbf{y}_k\,||^2)}$$

Symmetric Stochastic Neighborhood Embedding

We want to minimize the KL-divergence between **joint probability distributions:**

- **Similarities of points in high-dimensional space**:

$$p_{ij} = \frac{exp(-\,||\,\mathbf{x}_i - \mathbf{x}_j\,||^2\,/2\sigma^2)}{\sum_{k \neq i} exp(-\,||\,\mathbf{x}_i - \mathbf{x}_k\,||^2\,/2\sigma^2)}$$

- **Similarities of points in low-dimensional space**:

$$q_{ij} = \frac{exp(-\,||\,\mathbf{y}_i - \mathbf{y}_j\,||^2)}{\sum_{k \neq i} exp(-\,||\,\mathbf{y}_i - \mathbf{y}_k\,||^2)}$$

- **Cost function** to be minimized using gradient descent (KL-Divergence):

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

t-SNE

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

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

- **Similarities of points in high-dimensional space**:

$$p_{ij} = \frac{exp(- \| \mathbf{x}_i - \mathbf{x}_j \|^2 / 2\sigma^2)}{\sum_{k \neq i} exp(- \| \mathbf{x}_i - \mathbf{x}_k \|^2 / 2\sigma^2)}$$

- **Similarities of points in low-dimensional space**:

$$q_{ij} = \frac{(1+ \| \mathbf{y}_i - \mathbf{y}_j \|^2)^{-1})}{\sum_{l \neq k} (1+ \| \mathbf{y}_k - \mathbf{y}_i \|^2)^{-1}}$$

t-SNE - Impact of Perplexity

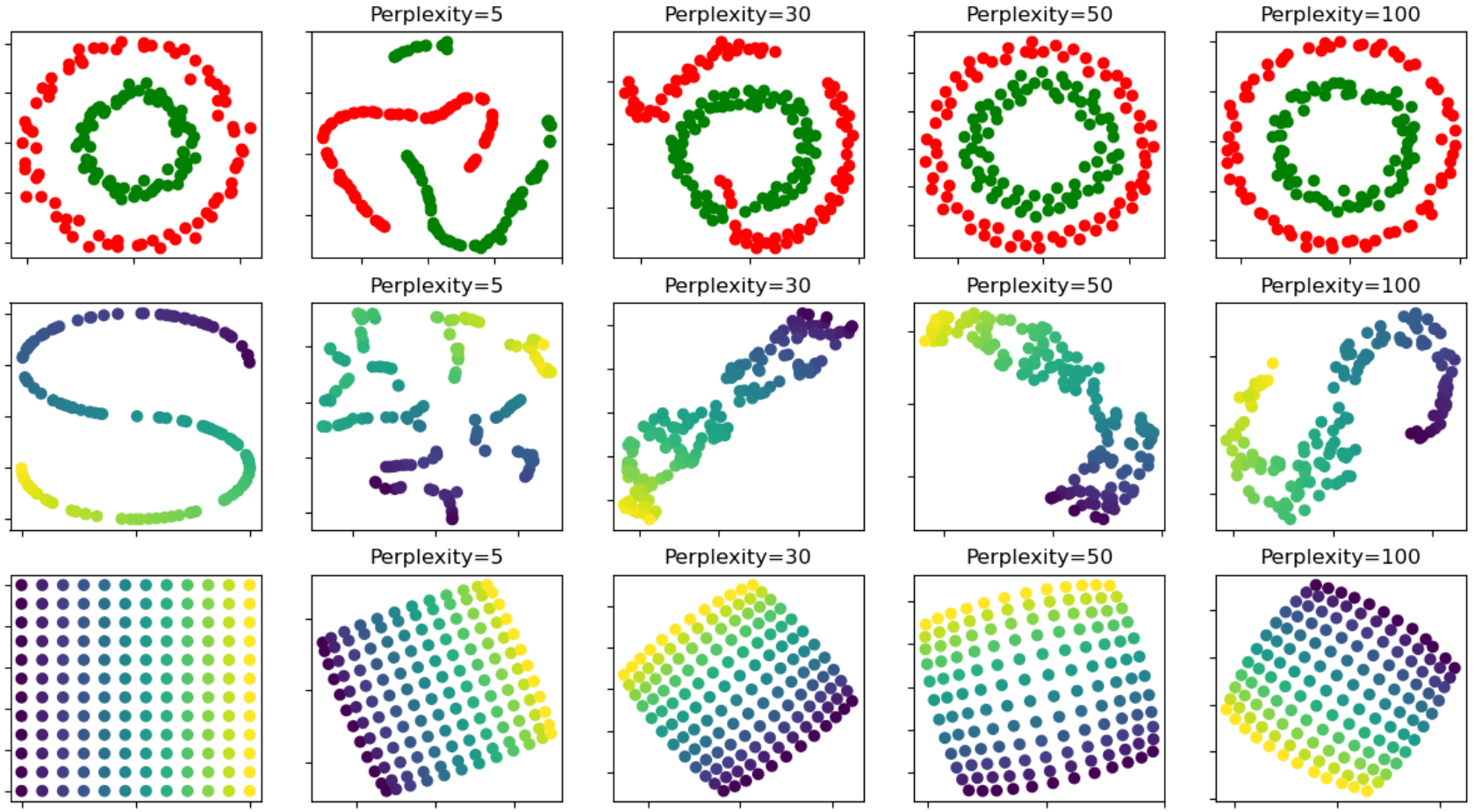
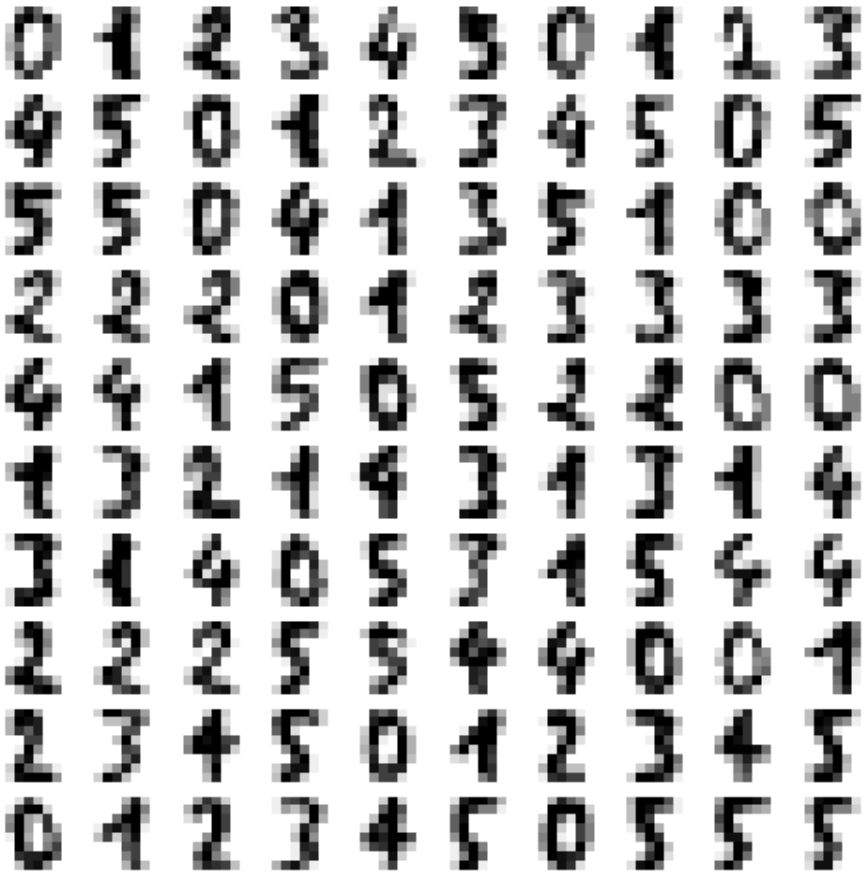


Image Generated running the code from:
https://scikit-learn.org/stable/auto_examples/manifold/plot_t_sne_perplexity.html

t-SNE - MNIST

A selection from the 64-dimensional digits dataset



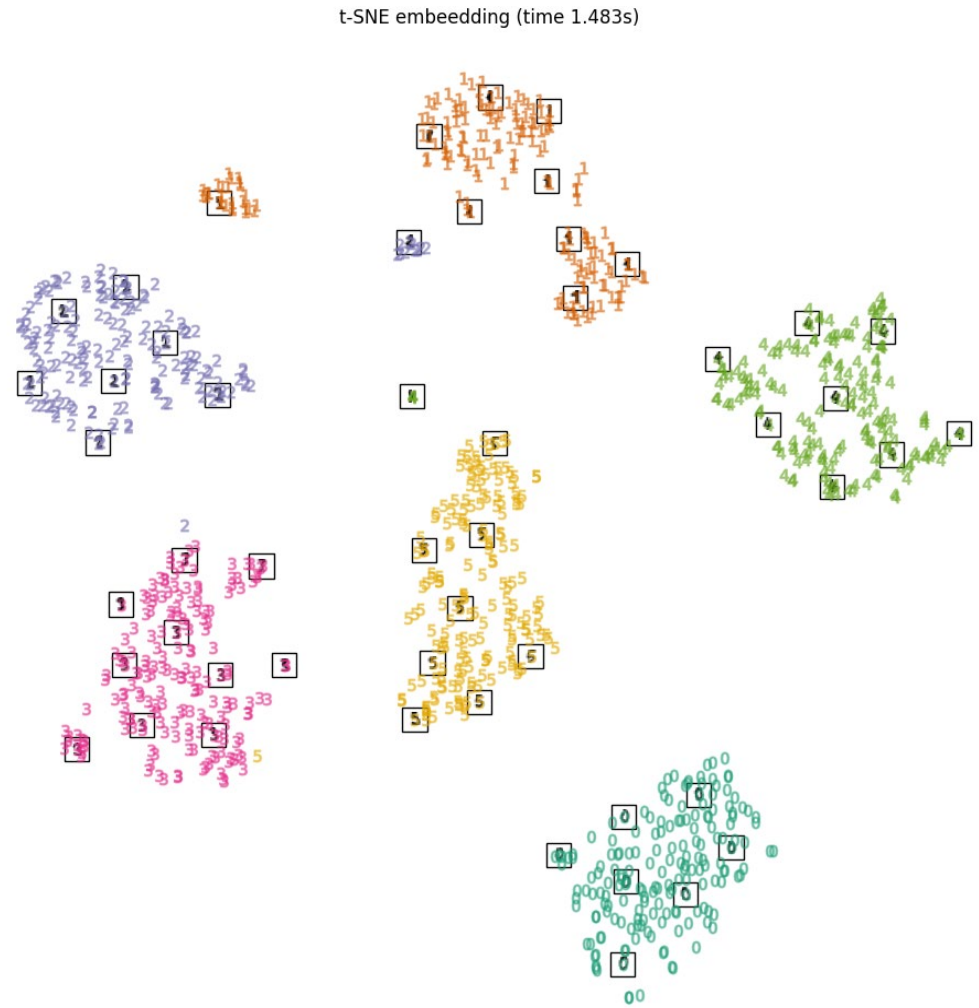


Image Generated running the scikit-learn demo for Manifold Learning on MNIST:
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html#sphx-glr-auto-examples-manifold-plot-lle-digits-py

t-SNE - Impact of Perplexity

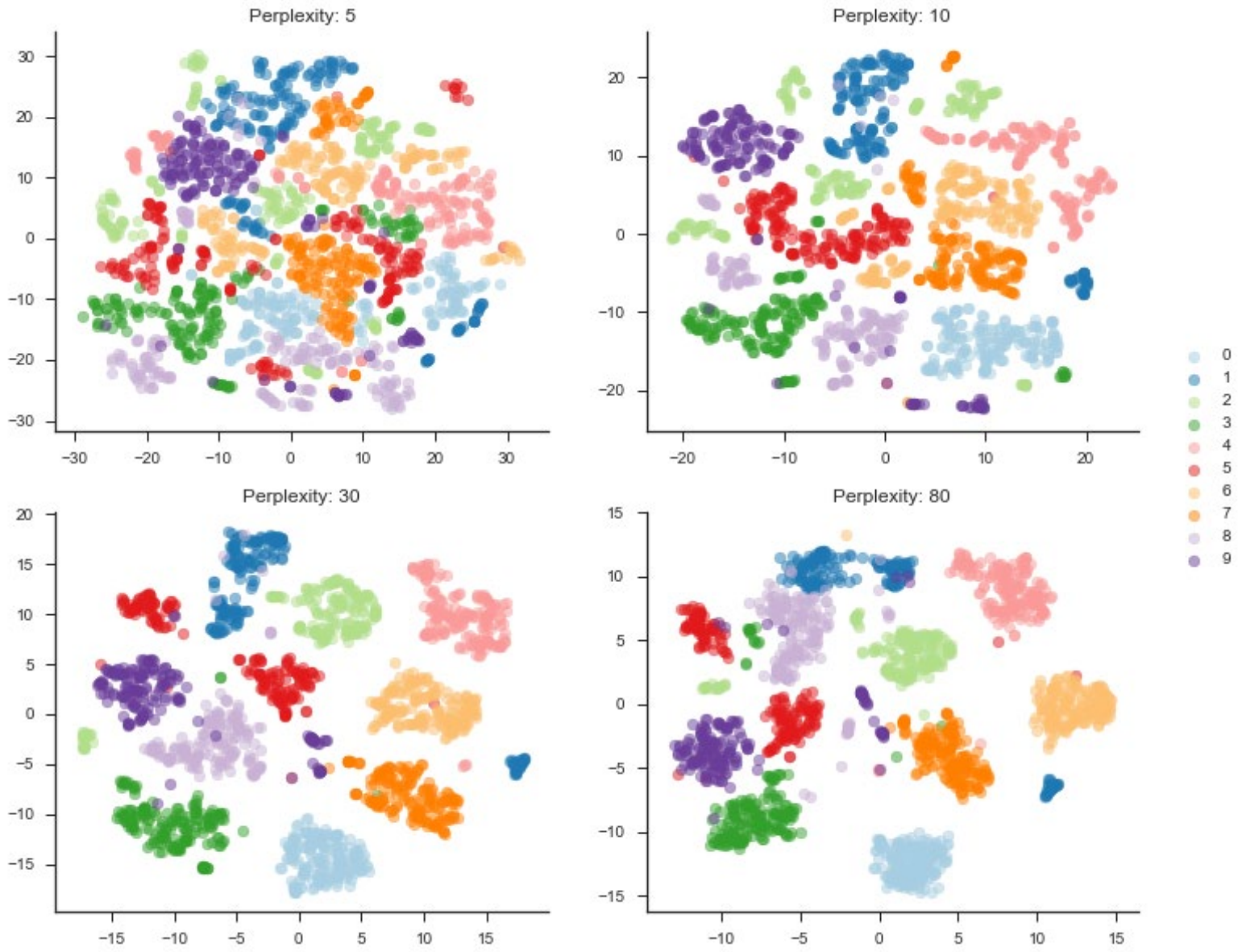


Image Generated running the code from:
https://nickc1.github.io/dimensionality/reduction/2017/11/04/exploring-tsne.html

Non-Linear Dimensionality Reduction Summary

Non-Linear Dimensionality Reduction - Summary

- **Multi-Dimensional Scaling** is used for analyzing the similarity between samples in the dataset based on the distances between data points in geometric spaces. **It tries to find a low dimensional representation of data that maintains the same distance between data points in original high dimensional data**.
  - MDS is time consuming and can take a lot of time when applied on large datasets

Non-Linear Dimensionality Reduction - Summary

- **Multi-Dimensional Scaling** is used for analyzing the similarity between samples in the dataset based on the distances between data points in geometric spaces. **It tries to find a low dimensional representation of data that maintains the same distance between data points in original high dimensional data**.
  - MDS is time consuming and can take a lot of time when applied on large datasets
- **Isomap** is an **extension of MDS or Kernel PCA.** It seeks to find lower dimensional embedding of the original dataset **while maintaining geodesic distances between all points in the original dataset**. Isomap tries to get a lower dimensional representation of the original dataset where points maintain geodesic distance between them like in the original representation.

Non-Linear Dimensionality Reduction - Summary

- **Multi-Dimensional Scaling** is used for analyzing the similarity between samples in the dataset based on the distances between data points in geometric spaces. **It tries to find a low dimensional representation of data that maintains the same distance between data points in original high dimensional data**.
  - MDS is time consuming and can take a lot of time when applied on large datasets
- **Isomap** is an **extension of MDS or Kernel PCA.** It seeks to find lower dimensional embedding of the original dataset **while maintaining geodesic distances between all points in the original dataset**. Isomap tries to get a lower dimensional representation of the original dataset where points maintain geodesic distance between them like in the original representation.
- **Locally Linear Embeddings (LLE)** seek to find a lower-dimensional projection of the original dataset, while **maintaining the distances within local neighborhood points.**
  - Very computational efficient.

Non-Linear Dimensionality Reduction - Summary

- **Laplacian Eigenmaps** is an extension of Isomap using **spectral decomposition of the graph's Laplacian, instead of MDS for finding the lower-dimensional space**.

Non-Linear Dimensionality Reduction - Summary

- **Laplacian Eigenmaps** is an extension of Isomap using **spectral decomposition of the graph's Laplacian, instead of MDS for finding the lower-dimensional space**.
- **t-SNE converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data**.

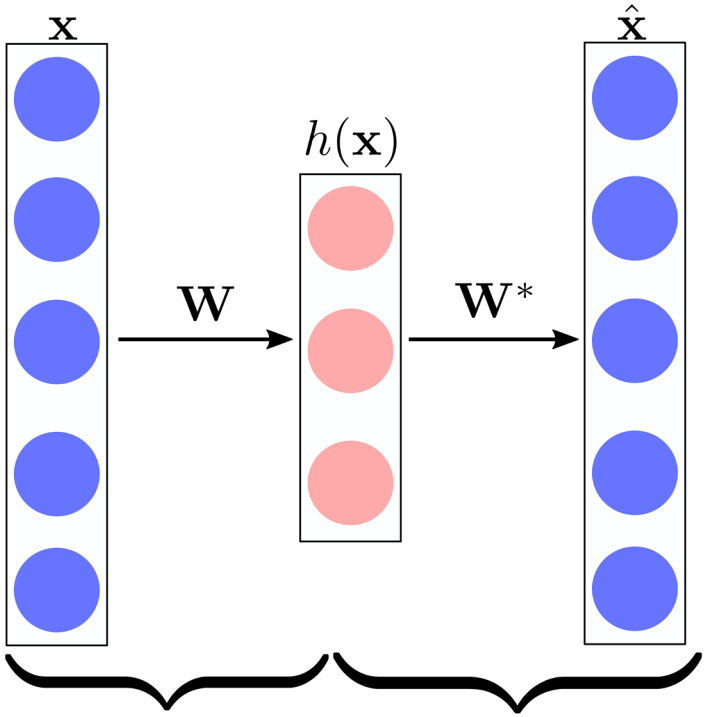
Non-Linear Dimensionality Reduction - Summary

	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

Autoencoders

Autoencoders

An autoencoder is a neural network trained to **reproduce its input** at the output layer. In other words, **an autoencoder maps a space to itself.**

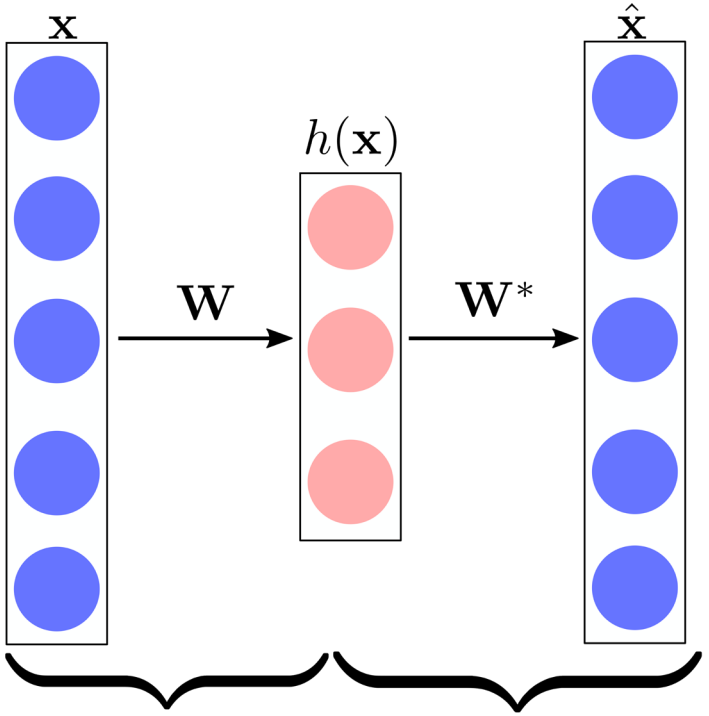


$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(\mathbf{b} + \mathbf{W}\mathbf{x})$$

$$\hat{\mathbf{x}} = o(\hat{a}(\mathbf{x})) = o(\mathbf{b}^* + \mathbf{W}^* h(\mathbf{x}))$$

Autoencoders

An autoencoder is a neural network trained to **reproduce its input** at the output layer. In other words, **an autoencoder maps a space to itself.**



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

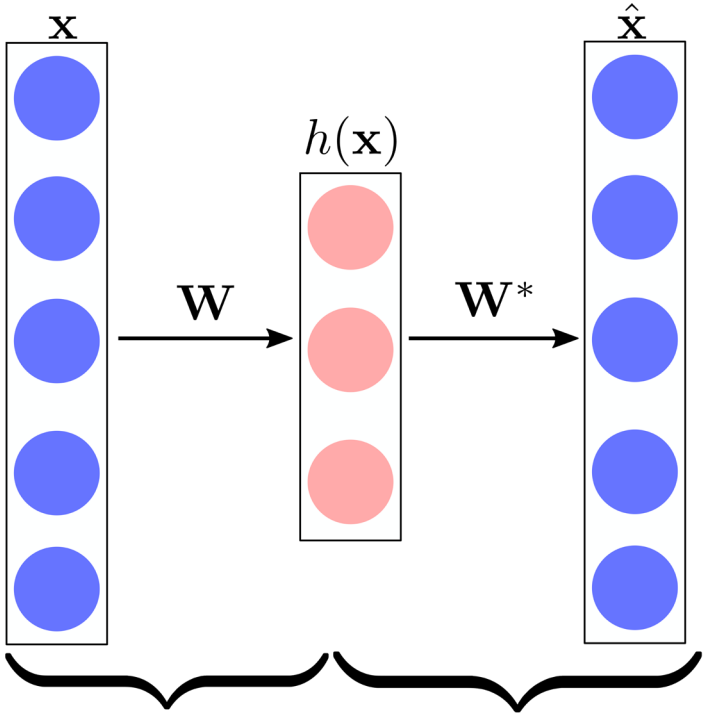
$$\hat{\mathbf{x}} = o(\hat{a}(\mathbf{x})) = o(\mathbf{b}^* + \mathbf{W}^* h(\mathbf{x}))$$

Network Parameters $\boldsymbol{\Theta} : \{\mathbf{b}, \mathbf{W}, \mathbf{b}^*, \mathbf{W}^*\}$

$$\boldsymbol{\Theta}^* = argmin_{\boldsymbol{\Theta}} \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$$

Autoencoders

An autoencoder is a neural network trained to **reproduce its input** at the output layer. In other words, **an autoencoder maps a space to itself.**



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

$$\hat{\mathbf{x}} = o(\hat{a}(\mathbf{x})) = o(\mathbf{b}^* + \mathbf{W}^* h(\mathbf{x}))$$

Network Parameters $\boldsymbol{\Theta} : \{\mathbf{b}, \mathbf{W}, \mathbf{b}^*, \mathbf{W}^*\}$

$$\boldsymbol{\Theta}^* = argmin_{\boldsymbol{\Theta}} \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$$

The loss function $L(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$

- For **binary outputs** can be:

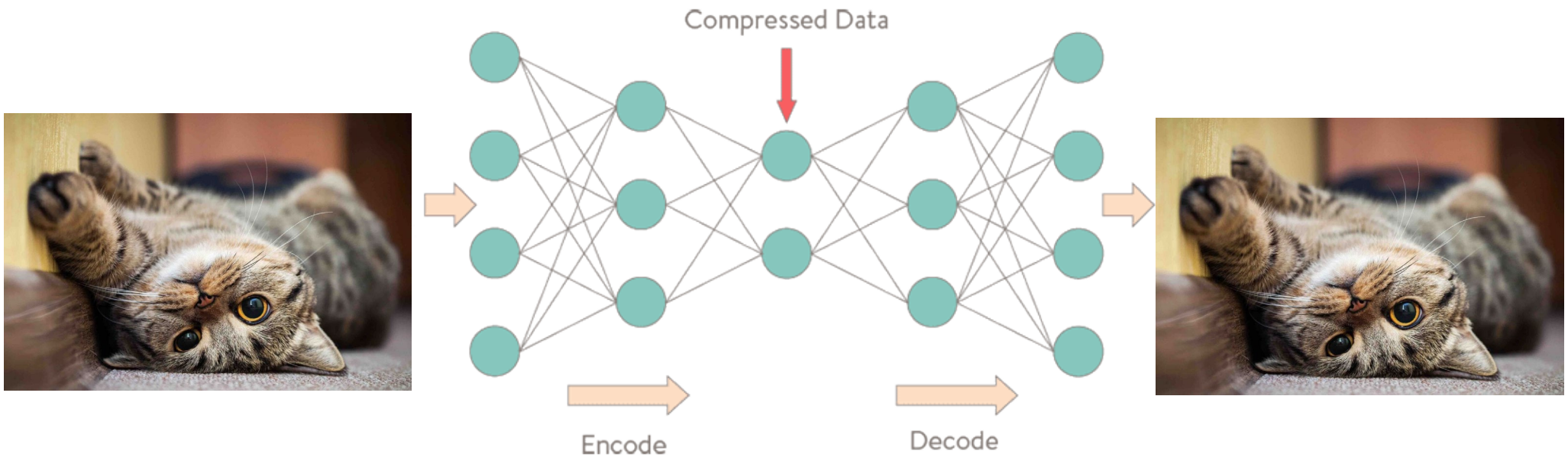
$$L(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)}) = \mathbf{x}^{(i)} \log(\hat{\mathbf{x}}^{(i)}) - (1 - \mathbf{x}^{(i)}) \log(1 - \hat{\mathbf{x}}^{(i)})$$

- For **real outputs** can be:

$$L(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)}) = || \mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}) ||^2$$

Autoencoders

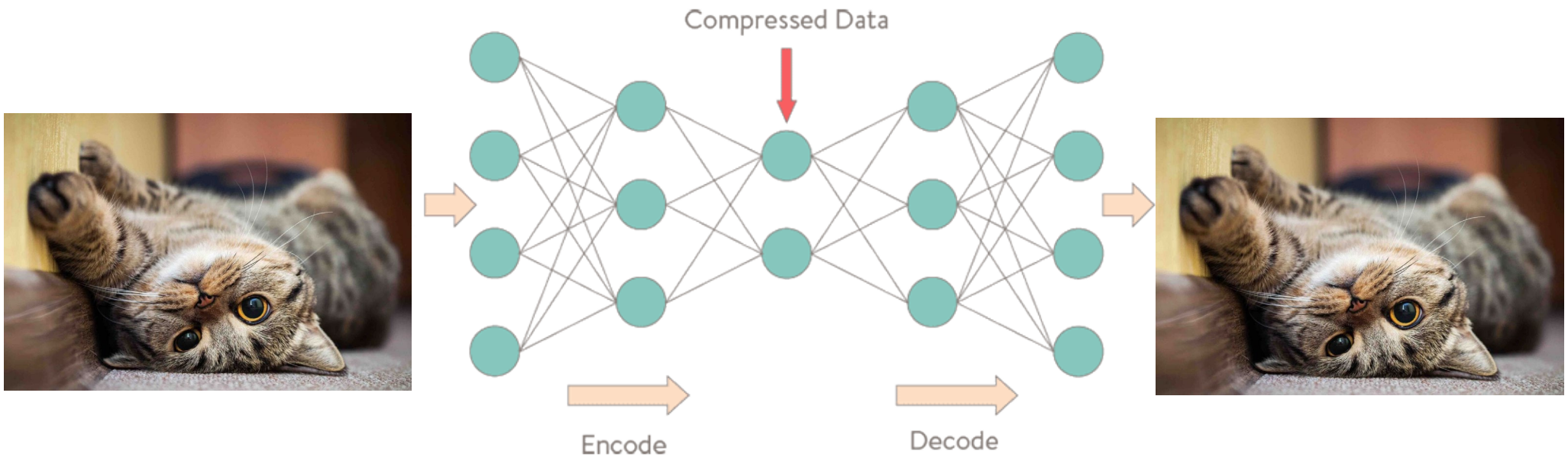
An autoencoder is a neural network trained to reproduce its input at the output layer. In other words, **an autoencoder maps a space to itself.**



Dimensionality reduction can be interpreted as **data compression** where the **encoder compresses the data** (from the original space to the latent space) **and the decoder decompresses them.**

Autoencoders

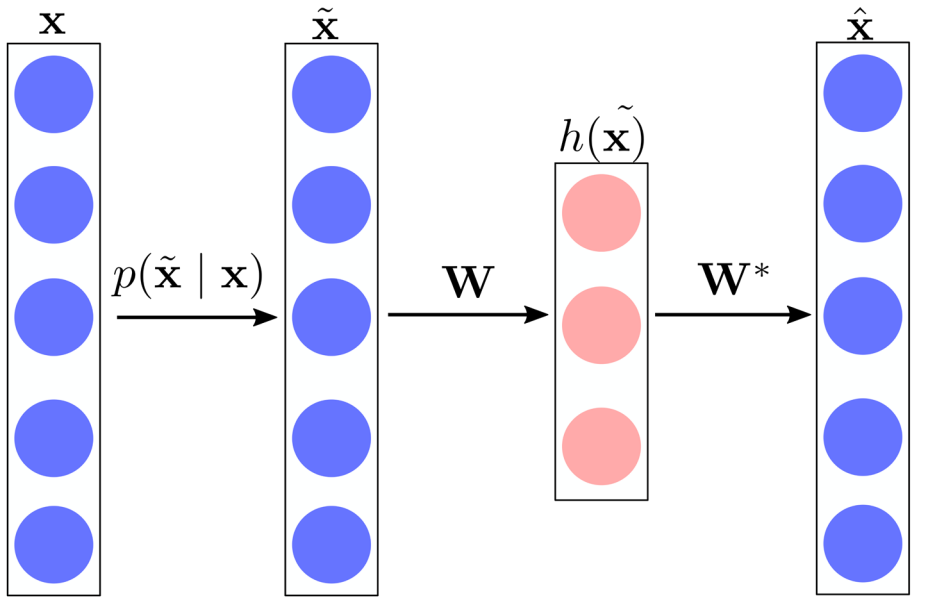
An autoencoder is a neural network trained to reproduce its input at the output layer. In other words, **an autoencoder maps a space to itself.**



The **main purpose** of a dimensionality reduction method is to f**ind the best encoder/decoder pair**. Namely, for a given set of possible encoders and decoders, **we are looking for the pair that keeps the maximum of information when encoding** and, so, **has the minimum of reconstruction error when decoding**.

Denoising Autoencoder

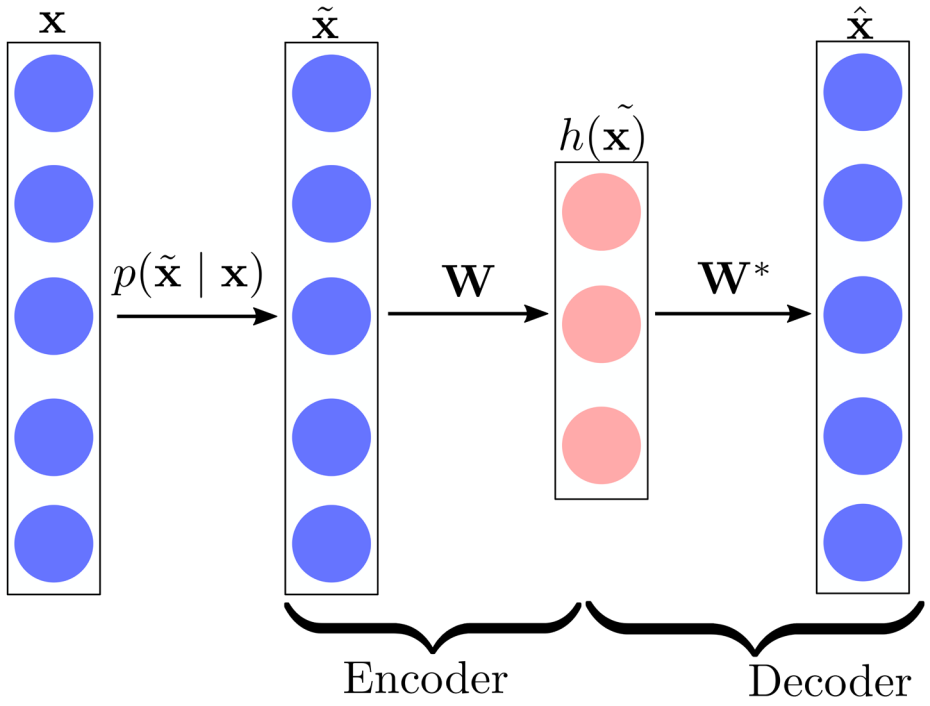
Denoising autoencoders **corrupt the data on purpose by randomly setting some of the input values to 0.0.** This prevents learning the identity function, namely simply copying individual elements of the input.



$$h(\mathbf{x}) = g(a(\tilde{\mathbf{x}})) \qquad \hat{\mathbf{x}} = o(\hat{a}(\mathbf{x}))$$
$$= g(\mathbf{b} + \mathbf{W}\tilde{\mathbf{x}}) \qquad = o(\mathbf{b}^* + \mathbf{W}^* h$$

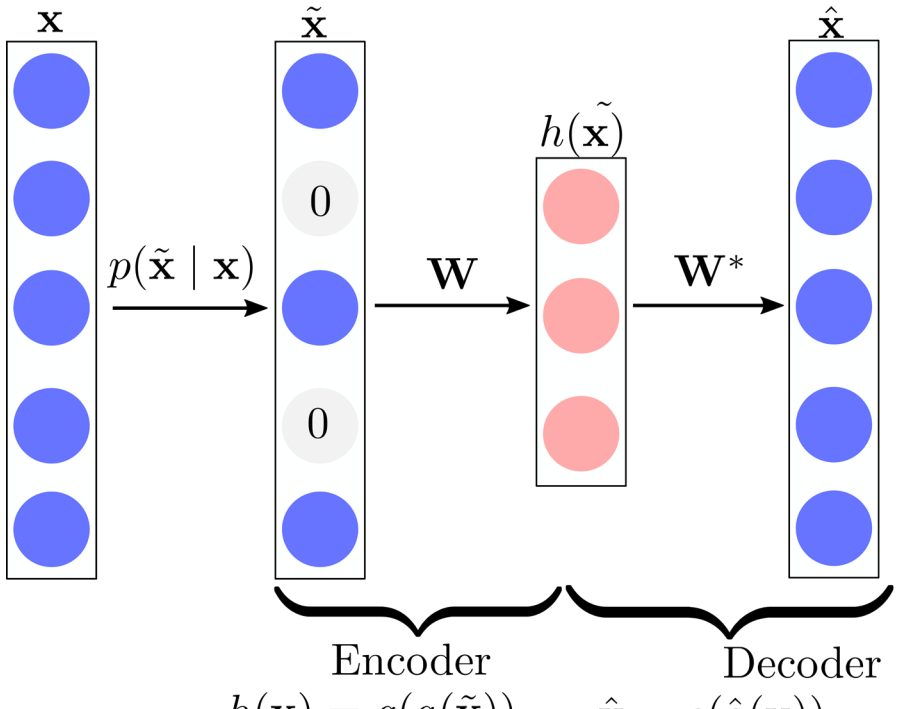
Denoising Autoencoder

Denoising autoencoders **corrupt the data on purpose by randomly setting some of the input values to 0.0.** This prevents learning the identity function, namely simply copying individual elements of the input.



$$h(\mathbf{x}) = g(a(\tilde{\mathbf{x}})) = g(\mathbf{b} + \mathbf{W}\tilde{\mathbf{x}}) \qquad \hat{\mathbf{x}} = o(\hat{a}(\mathbf{x})) = o(\mathbf{b}^* + \mathbf{W}^* h$$

Gaussian additive noise

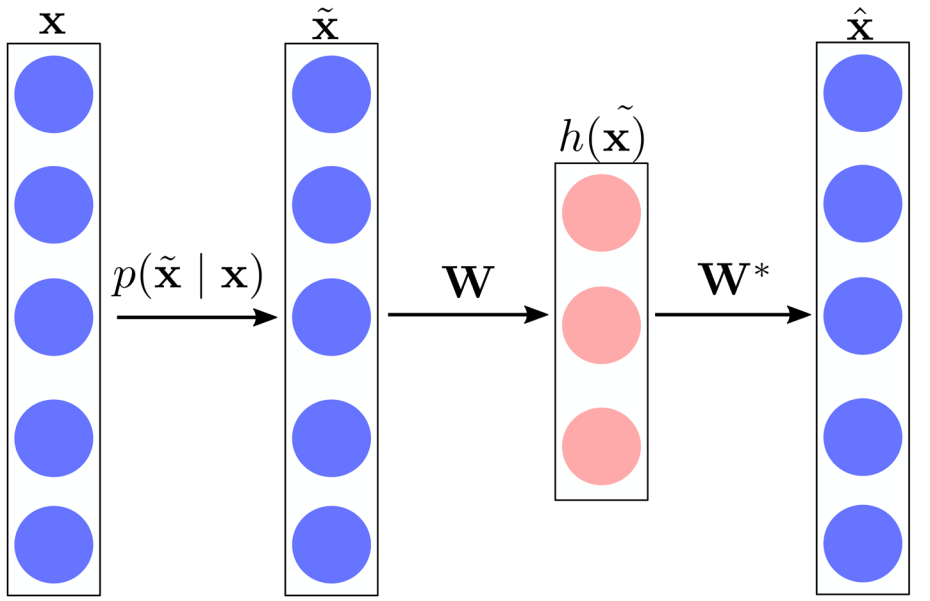


$$h(\mathbf{x}) = g(a(\tilde{\mathbf{x}})) = g(\mathbf{b} + \mathbf{W}\tilde{\mathbf{x}}) \qquad \hat{\mathbf{x}} = o(\hat{a}(\mathbf{x})) = o(\mathbf{b}^* + \mathbf{W}^* h(\tilde{\mathbf{x}}))$$

Random assignment of subset of inputs to 0

Denoising Autoencoder

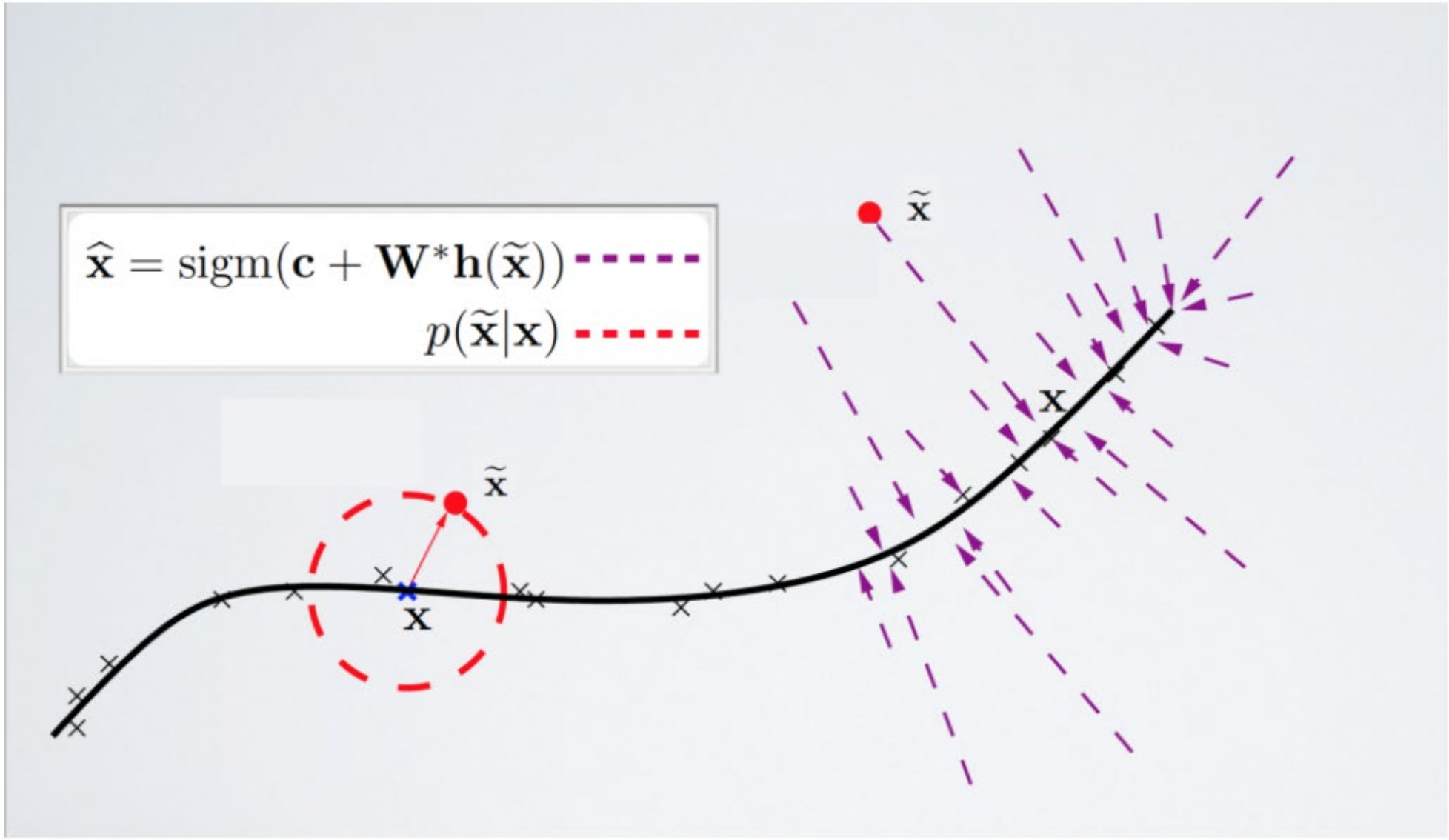
Denoising autoencoders **corrupt the data on purpose by randomly setting some of the input values to 0.0.** This prevents learning the identity function, namely simply copying individual elements of the input.



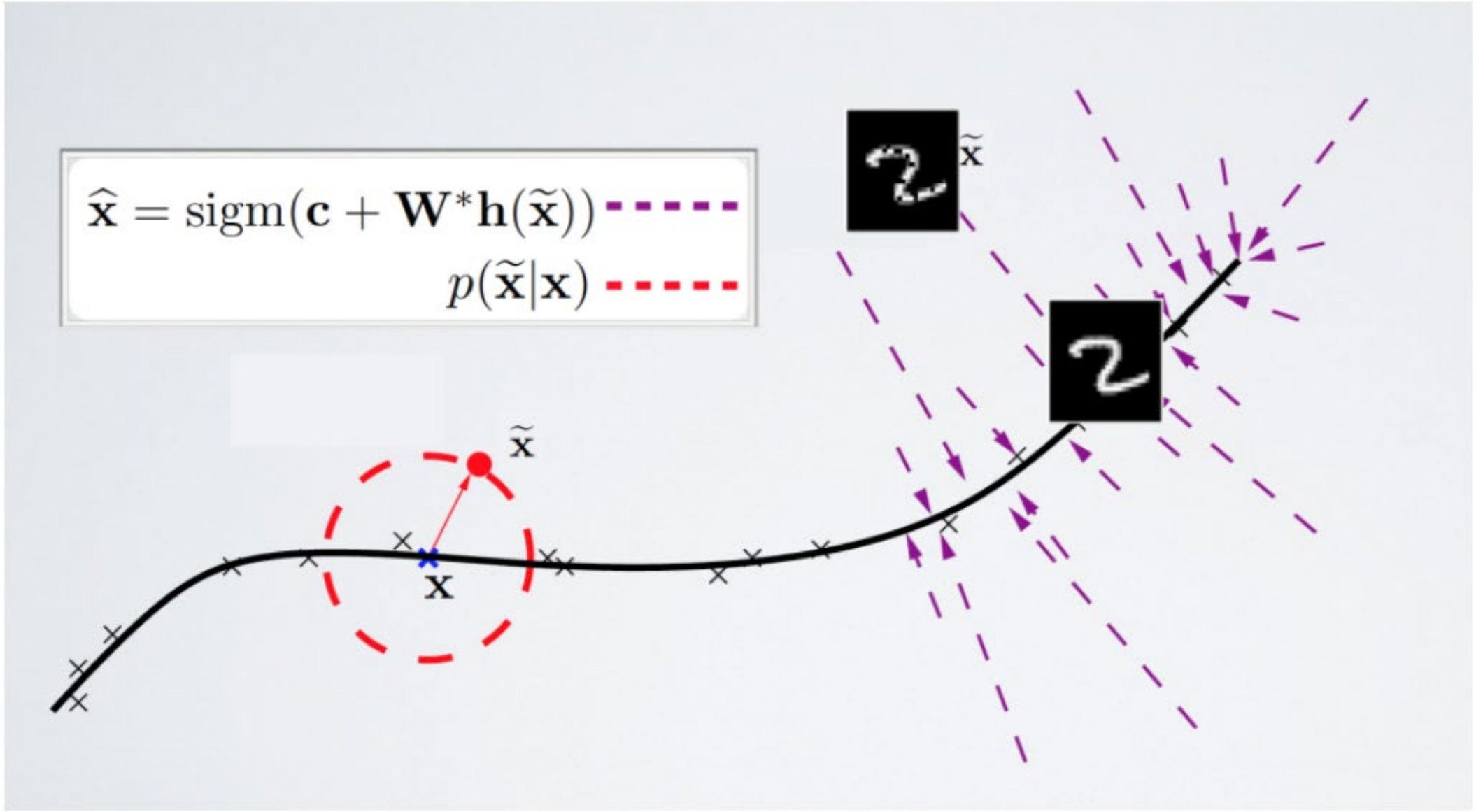
$$\text{Network Parameters } \boldsymbol{\Theta} : \{\mathbf{b}, \mathbf{W}, \mathbf{b}^*, \mathbf{W}^*\}$$

$$\boldsymbol{\Theta}^* = argmin_{\boldsymbol{\Theta}} \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$$

Denoising Autoencoder – Manifold Perspective



Denoising Autoencoder – Manifold Perspective



Is this all?

- Sammon's mapping, 1969
- Self-organizing map (SOM, aka Kohonen map, based on neural networks), 1982
- Principal curves and manifolds, 1984
- Autoencoders (some neural networks), 19xx
- Generative topographic map (GTM, probabilistic version of SOM), 1996
- Curvilinear component/distance analysis (CCA, CDA), 1997
- Kernel PCA (kPCA), 1998
- ISOMAP, 2000
- Locally-linear embedding (LLE), 2000
- Laplacian Eigenmaps, 2001
- Hessian LLE, 2003
- Gaussian process latent variable models (GPLVM), 2004
- Maximum variance unfolding (MVA, aka semidefinite embedding), 2004
- Maximum variance unfolding (MVA, aka semidefinite embedding), 2004
- Relational perspective map, 2004
- Nonlinear PCA (based on neural networks), 2005
- Local tangent space alignment (LTSA), 2005
- Modified LLE, 2006
- Diffusion maps, 2006
- Local multidimensional scaling, 2006
- Manifold alignment, 2008
- Manifold sculpting, 2008
- t-distributed stochastic neighbor embedding (t-SNE), 2008
- Diffeomorphic Dimensionality Reduction (Diffeomap), 2009
- Rank visu, 2009
- Topologically Constrained Isometric Embedding (TCIE), 2010

List Source: https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction

That’s All