Introduction to Machine Learning.
CSCI-UA 9473, Lecture 10.

Augustin Cosse

Ecole Normale Supérieure, DMA & NYU
Fondation Sciences Mathématiques de Paris.

2018

Manifold Learning

- With an increase in the volume of data, learning algorithms are facing new challenges.
- Irrelevant and correlated features add to the computational complexity
- Understanding large amounts of data thus requires extracting information out of them. Otherwise such data is useless

Manifold Learning

- Many examples of high dimensional data exhibiting such redundancy arise in computer vision.
- As a particular instance, consider the problem of estimating where a person is looking. This information can be particularly useful to an automated agent as it often gives a clear indication on where objects of interest are located.
- The main difficulty lies in the raw (i.e pixel) information that the agent receives
- Most of the time, the agent "only gets to see a few samples from which it needs to interpolate and generalize the various scenarios"

source: Nakul Verma, Mathematical Advances in Manifold Learning

Manifold Learning

- The agent is thus faced with the problem of finding an appropriate representation of the data based on which it can complete and/or interpolate the samples.
- Manifold learning is interested in inferring some global properties of high dimensional objects (i.e the manifold) from a few samples.

source: Nakul Verma, Mathematical Advances in Manifold Learning

Manifold Learning: the basics

- We say that a function $f : U \mapsto V$ is a diffeomorphism if it is smooth (all partial derivatives exist and are continuous) and invertible with a smooth inverse.

- A subset $\mathcal{M} \subset \mathbb{R}^d$ is said to be a smooth n-manifold if $\mathcal{M}$ is diffeomorphic to $\mathbb{R}^n$. That is to say at each $p \in \mathcal{M}$, one can find an open neighborhood $U \subset \mathbb{R}^D$ such that there exists a diffeomorphic map between $U \cap \mathcal{M}$ and $\mathbb{R}^n$.

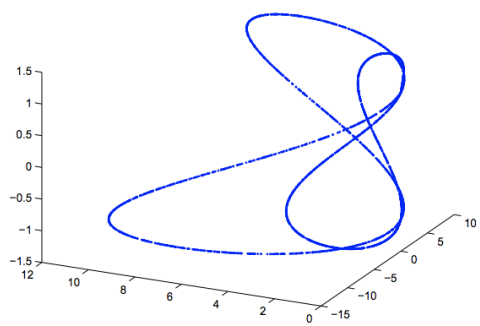


Figure 1: A 1-manifold in $\mathbb{R}^3$

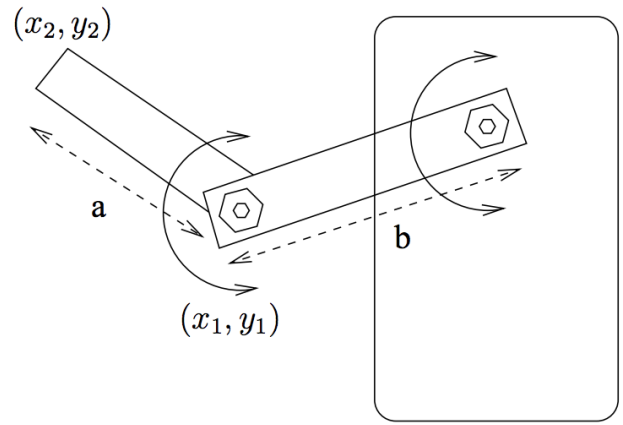


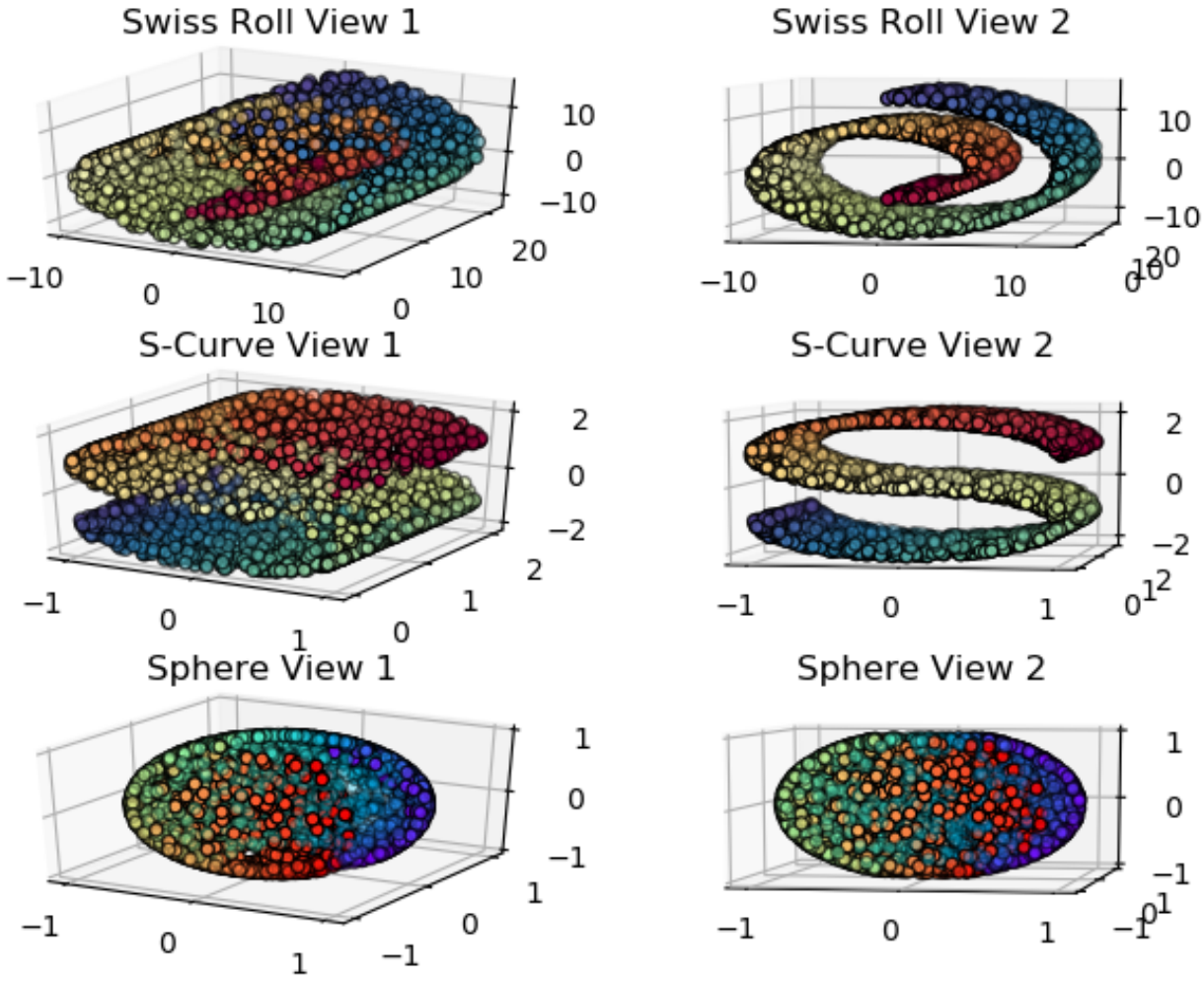
Figure 2: Movement of a robot's arm traces out a 2-manifold in $\mathbb{R}^4$

source: Nakul Verma, Mathematical Advances in Manifold Learning

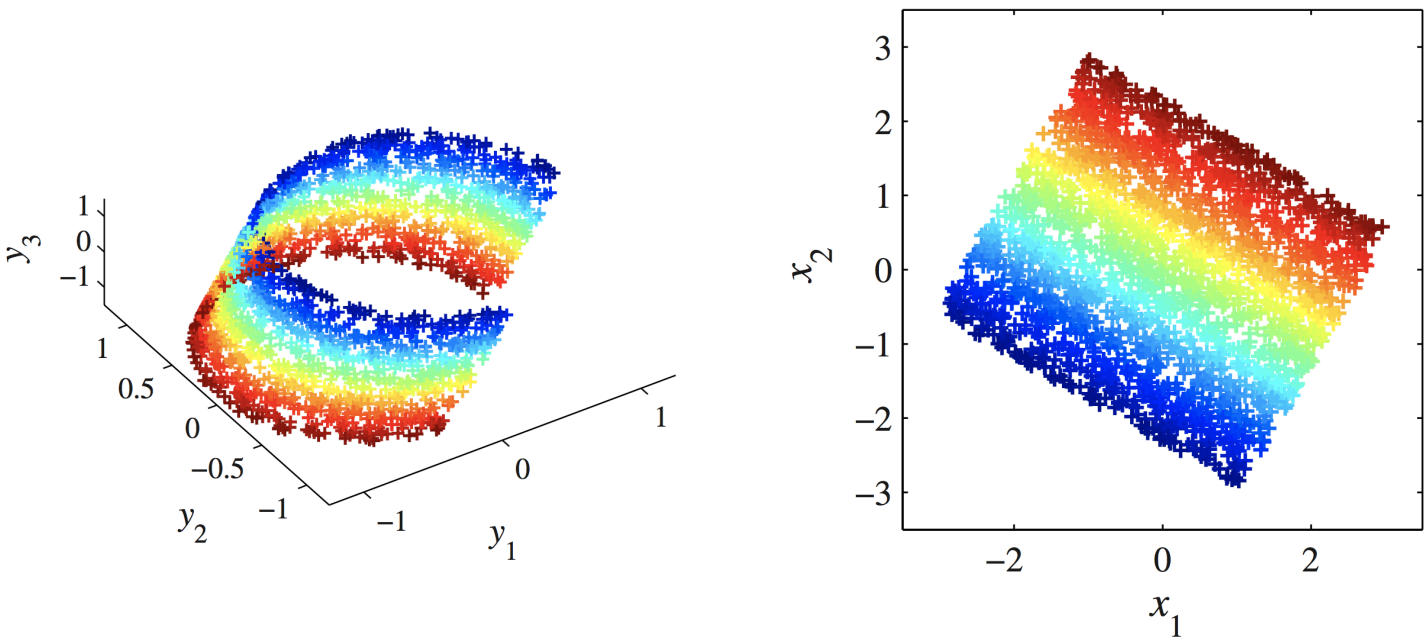
Manifold Learning: the basics

- An embedding is a representation of a topological object (e.g. graph, manifold) in a certain space ($\mathbb{R}^D$) in such a way that the topological properties are preserved
- As an example, the embedding of a manifold preserves open sets
- We will call intrinsic dimension of a random vector $\boldsymbol{y}$, the minimal number of parameters or latent variables needed to describe $\boldsymbol{y}$. (The intrinsic dimension is also the topological dimension of the support of the distribution of the vector $\boldsymbol{y}$)

Manifold Learning



Manifold Learning. From 3D to 2D embedding



source: Lee and Verleysen, Non Linear Dimensionality reduction

Manifold Learning: Some applications

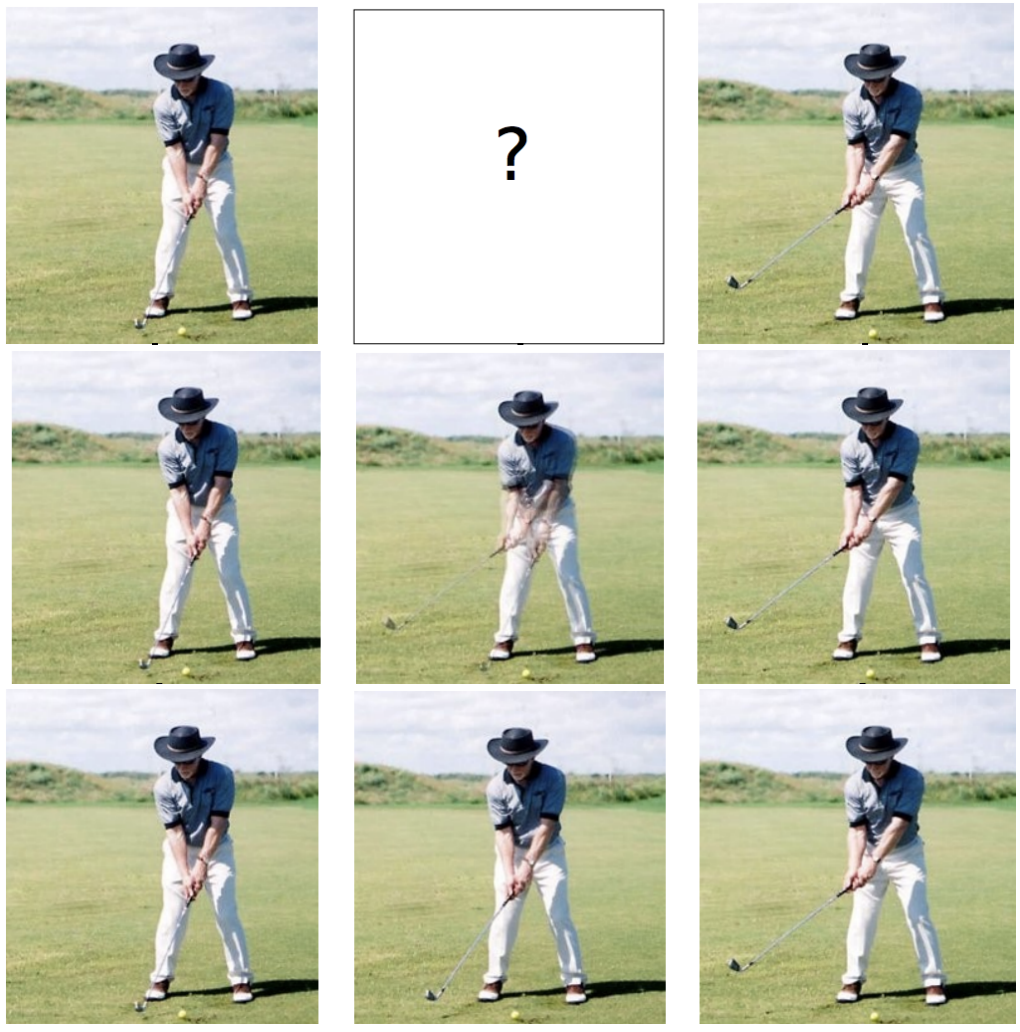
- Manifold learning methods have played an important role in the prevention and treatment of diseases (Golchin et al. 2014) including
  - Diagnosis of cancer and brain tumor progression
  - Abnormalities in the walking cycle
  - Abnormalities on the left ventricle in cardiac ECG images
- Other applications include
  - Face recognition
  - Motion tracking in video surveillance
  - Pose estimation

Manifold Learning: Main interests

- Data compression
- Denoising, deblurring
- Visualization and curse of dimensionality
- Image interpolation for high resolution movies (reasonable distance metric definition)

Source: Advanced Perception, UCBerkeley, David R. Thompson

Manifold Learning: Image interpolation



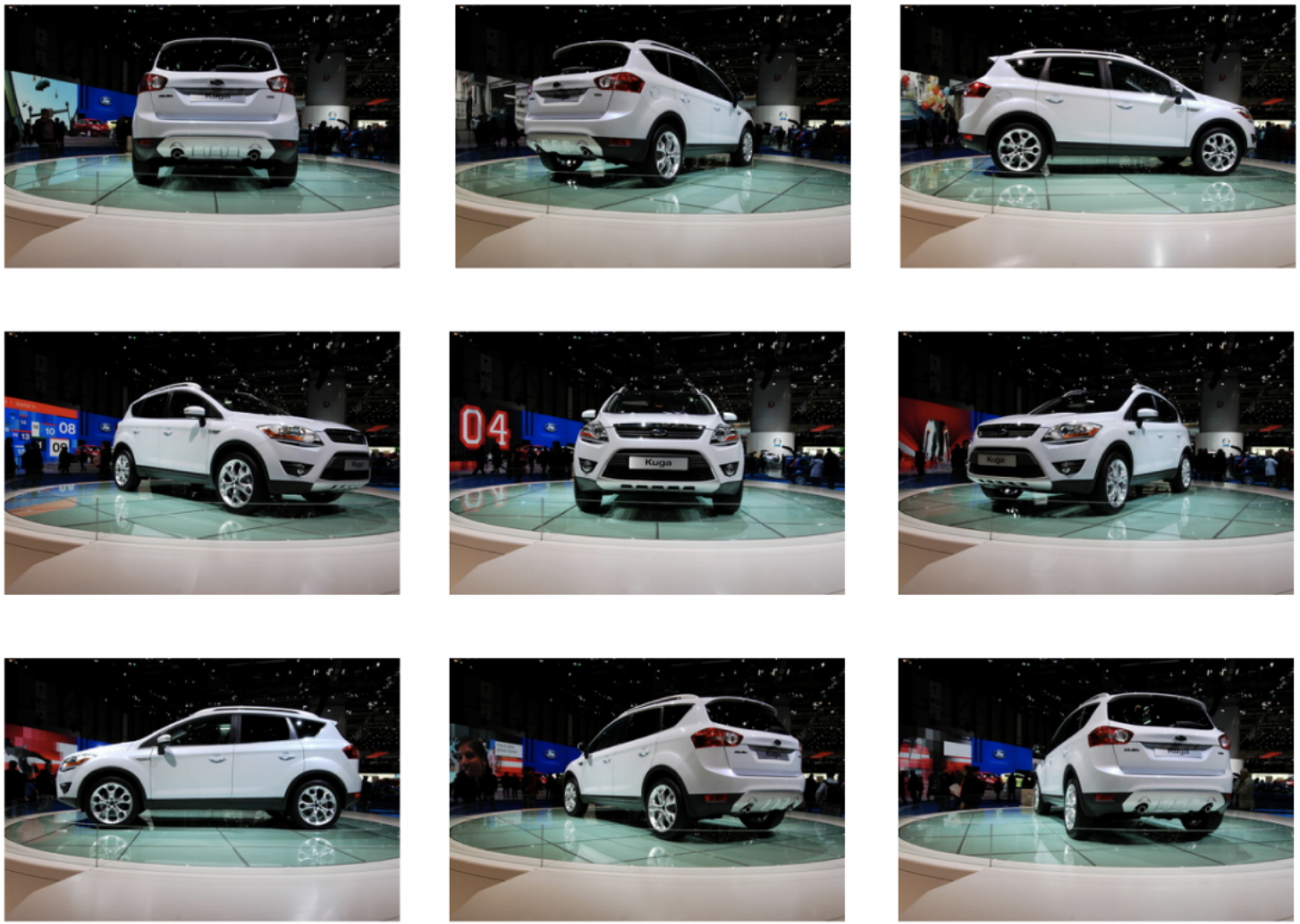
Source: Advanced Perception, UCBerkeley, David R. Thompson

Manifold Learning: Pose estimation

- In computer vision and robotics, a typical task is to identify objects and to determine each object's position.
- The combination of position and orientation of an object is known as the pose.
- The ability to accurately estimate the pose of generic object categories from videos or images is crucial in many applications such as robotic manipulation, human-object interaction, image indexing,...

Source: Mei et al., Robust Object Pose Estimation via Statistical Manifold Modeling D. Yang, Hierarchical regression learning for car pose estimation.

Manifold Learning: Car pose estimation (II)



D. Yang, Hierarchical regression learning for car pose estimation.

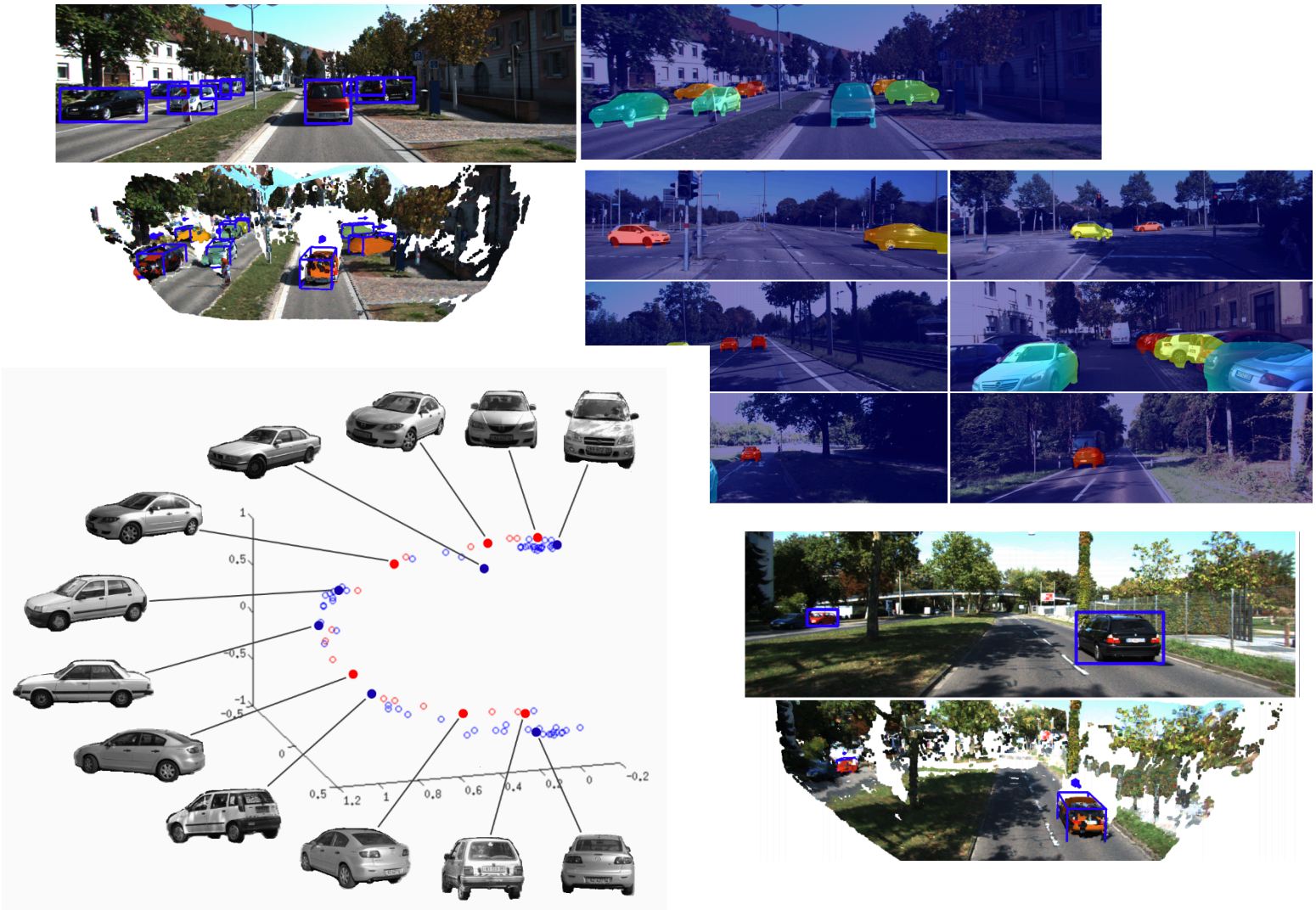
Manifold Learning for Pose estimation

- Pose estimation can be considered as a classification problem (discrete set of angles), or as a regression problem (continuous motion)
- Another important application of pose estimation is autonomous driving (AD). In AD, the pose (a.k.a. viewing angles) of a vehicle indicates the front direction of the car
- Car pose estimation is important problem in intelligent transportation systems. "[...]The viewing angles of surrounding vehicles of an autonomous driving car imply its possible past and future paths." [D. Yang, Hierarchical regression learning for car pose estimation.]

Manifold Learning: Car pose estimation (III)

D. Yang, Hierarchical regression learning for car pose estimation.

Manifold Learning: Car pose estimation (IV)



M. A.-Nachimson et al., *Implicit 3D Shape Models for Pose Estimation*

F. Engelmann et al. Joint Object Pose Est. and Shape Rec. in Urban Street Scenes Using 3D Shape Priors

Manifold Learning: Car pose estimation (V)

Figure 1. Example images from the initial set.

Figure 2. Example images from the remaining training set.

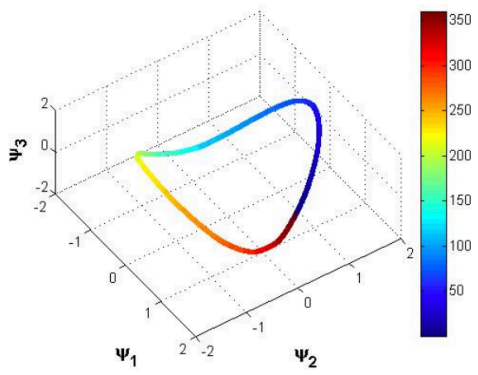
Figure 10. Part labels propagated from the model based on the voting.

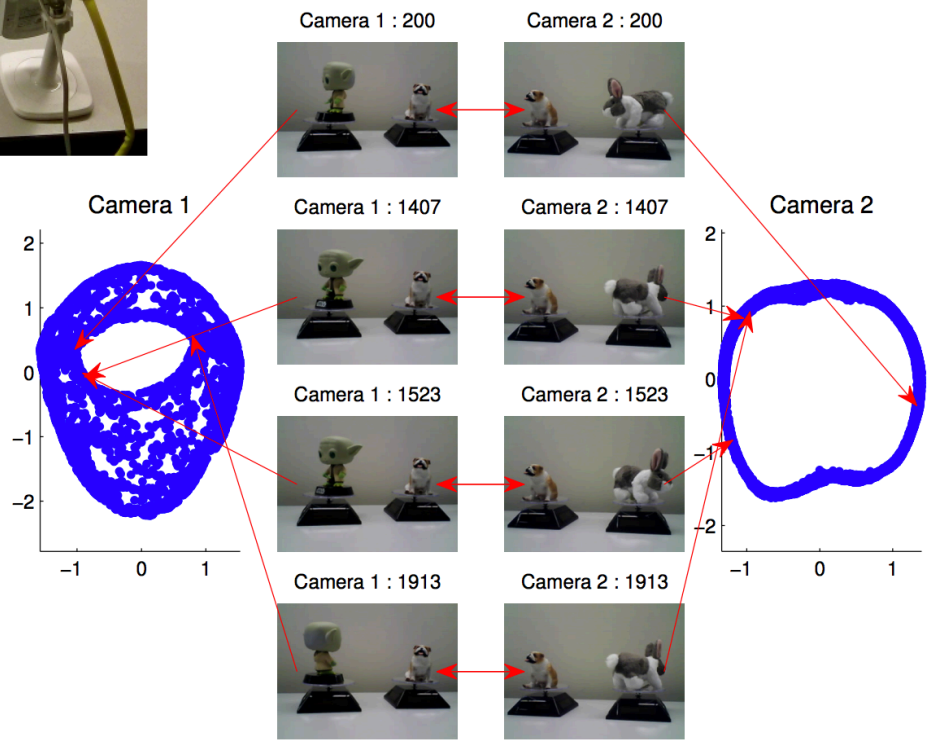
Source: M. Arie-Nachimson, R. BasriConstructing, *Implicit 3D Shape Models for Pose Estimation*

Manifold learning for car pose estimation

- "Large variation of the low level input imagery feature usually makes the mapping challenging" D. Yang, Hierarchical regr. learn. for car pose est.
- "The existing regression methods for vehicle viewpoint direction incorporate manifold locality into either explicit feature representation [13, 14] or implicit regression model training [4]" D. Yang, Hierarchical regr. learn. for car pose est.

Manifold learning: Different sources of variability





Source: Roy R. Lederman et al.

Manifold learning: Body pose estimation

Source: N. Sarafianos et al., 3D Human pose estimation: A review of the literature and analysis of covariates

Manifold learning: Body pose estimation

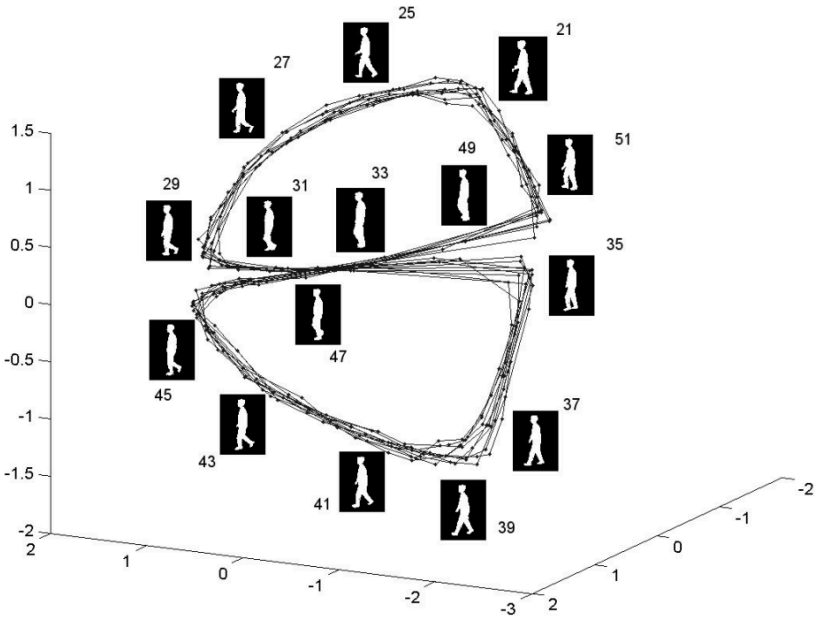
- The recovery of a 3D body pose is a fundamental aspect of human motion analysis
- The human body is an articulated object moving through the 3D space and is constrained by 3D body kinematics and dynamics as well as the dynamics of the activity being performed.

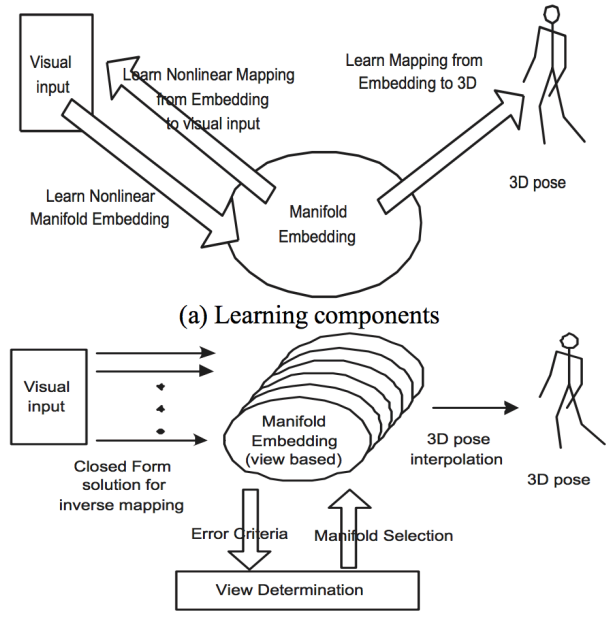
Source: A. Elgammal and C. Lee, Inferring 3D Body Pose from Silhouettes using Activity Manifold Learning

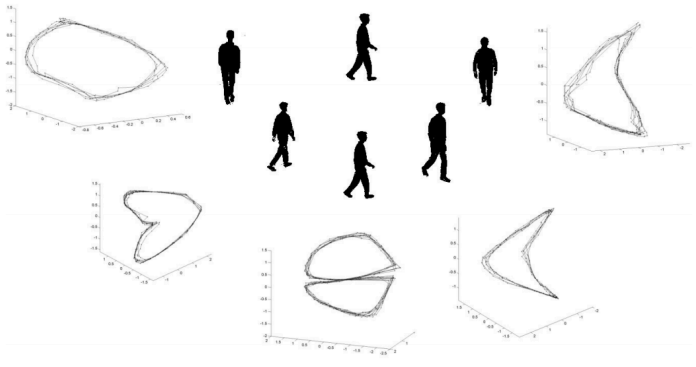
Manifold learning: Body pose estimation

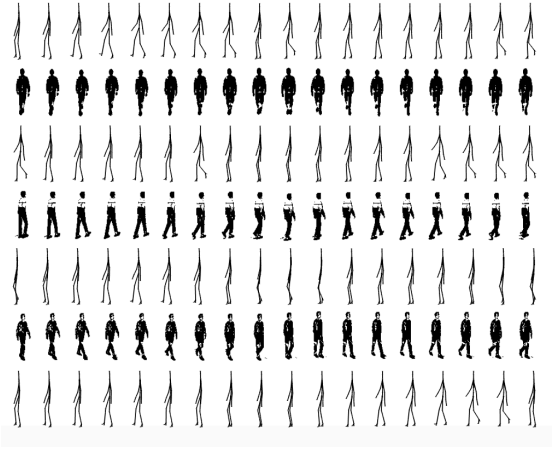
- The problem appears in a variety of fields including
  - Human computer interaction (computers can be controled by human gestures or can recognize sign languages)
  - Human robot interaction. Domestic robots should be able to perceive human body pose
  - Video surveillance (in smart surveillance systems, human motion conveys the action of the subject. Manual monitoring is impossible and a system should assist focusing on events of interest)
  - Gaming and other VR applications (Microsoft Kinect)
  - Sport performance analysis (analyzing the actions of athletes from multiple views),
  - ...

Source: Sarafianos et al.



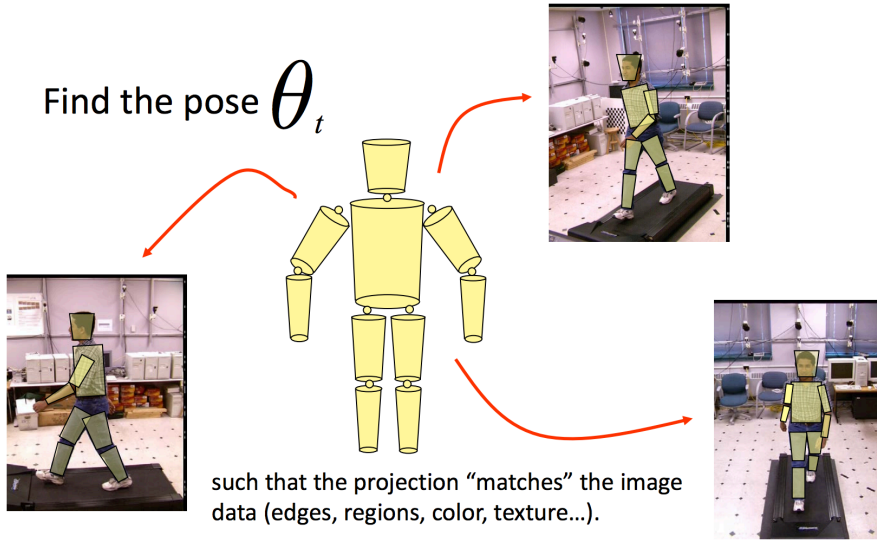


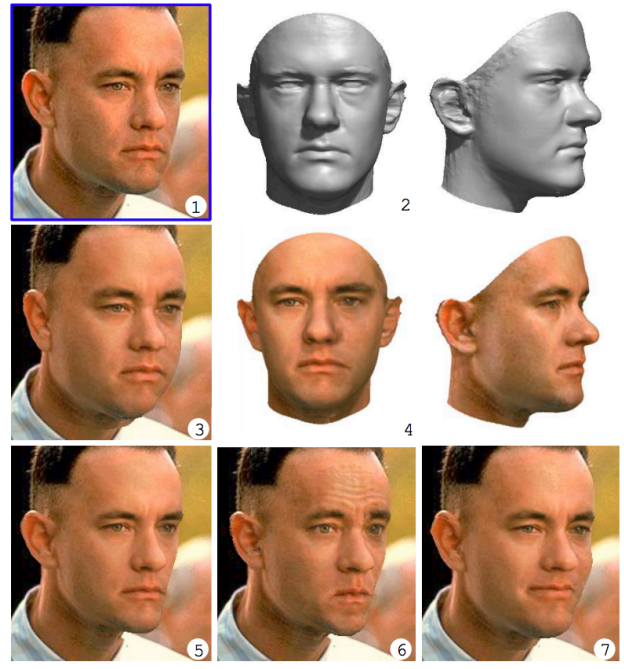


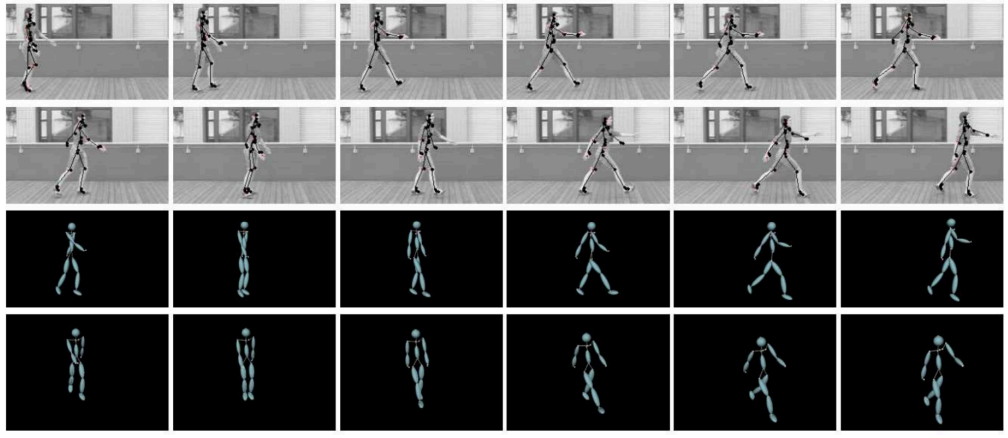


source: Elgammal et al.

The generative approach







Sources: V. Blanz, T. Vetter, SIGGRAPH '99
R. Urtasu, D. Fleet, P. Fua, CVPR 2006
M.J. Black, Estimating Human Motion: Past, Present, and Future, october 2018.

Manifold learning methods

- Multidimensional Scaling (MDS)
- Isometricfeaturemapping (ISOMAP)
- Locally linear embedding (LLE)
- Self Organizing maps (SOM)
- Laplacian eigenmaps
- Diffusion maps
- ...

Multidimensional scaling (MDS)

- Multidimensional scaling refers to a family of methods which construct a lower dimensional representation of the dataset from information on interpoint distances.
- MDS has been widely used in the humanities such as psychology, economics or sociology in which it is particularly interesting because it can be used with a notion of similarity between points.
- MDS does not require interpoint distances but can be applied from a similarity measure only.

Multidimensional scaling (MDS)

- The classical multidimensional scaling approach preserves inner products instead of distances. I.e in classical multidimensional scaling, we start with similarities s_{ij}. If not given similarities explicitely, we use pairwise inner products.
- The classical stress function is thus defined as

$$E_C(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N) = \sum_{i,j} (s_{i,j} - \langle \boldsymbol{z}_i - \overline{\boldsymbol{z}}, \boldsymbol{z}_j - \overline{\boldsymbol{z}} \rangle)$$

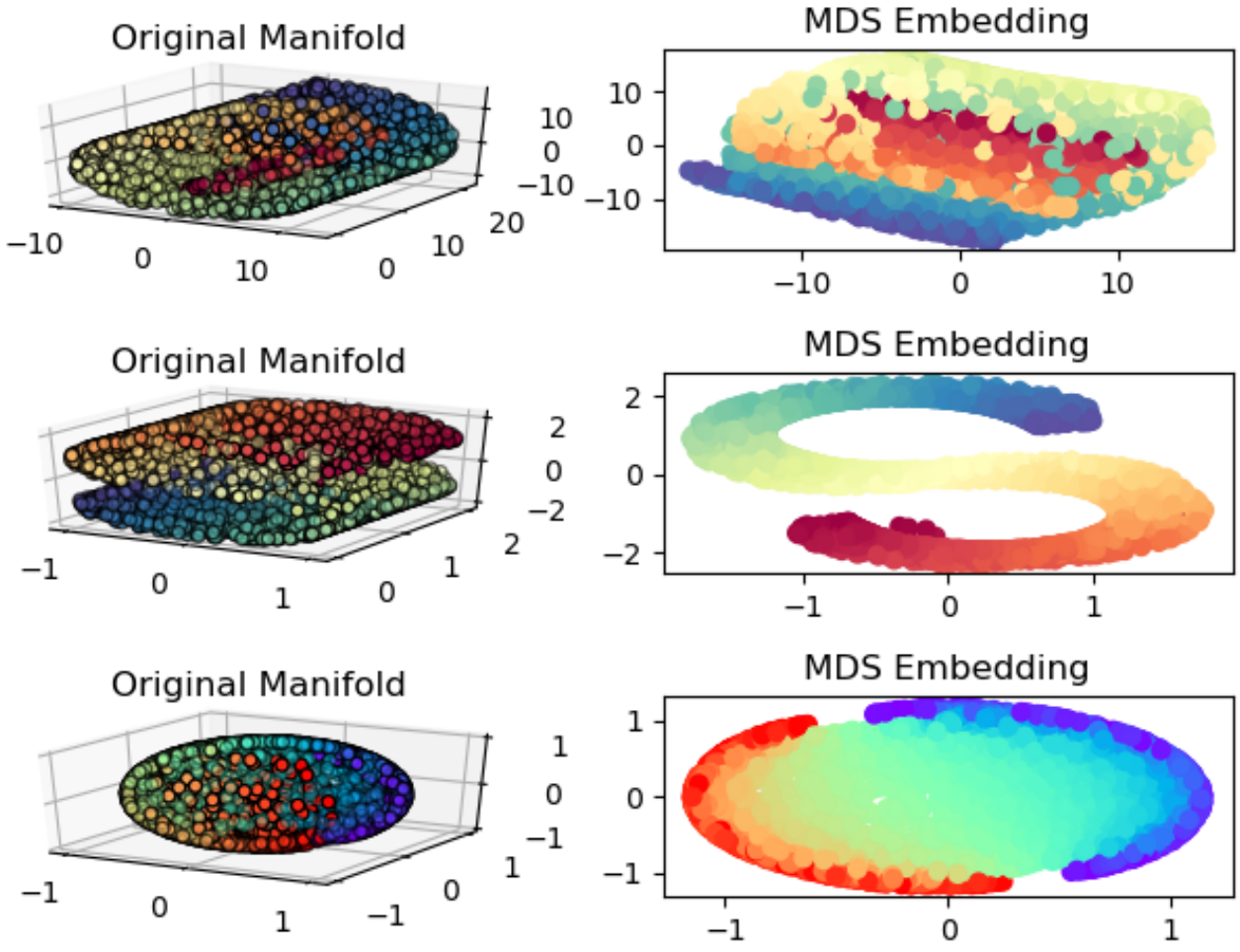
Where the similarity is thus often given by the centered inner products $s_{ij} \equiv \langle \boldsymbol{x}_i - \overline{\boldsymbol{x}}, \boldsymbol{x}_j - \overline{\boldsymbol{x}} \rangle$.

1. If available data consists of points $\boldsymbol{x}_i$, stack them into a matrix $\boldsymbol{Y}$ then center them, compute the pairwise inner products $\boldsymbol{S} = \boldsymbol{Y}^T\boldsymbol{Y}$ and go to third step
2. If available data consists of pairwise Euclidean distances, transform them to inner products.
   Square the distances and build the matrix $\boldsymbol{B}$
   Perform double centering of $\boldsymbol{B}$ to get $\tilde{\boldsymbol{S}}$.

$$\tilde{\boldsymbol{S}} = (\boldsymbol{I} - \boldsymbol{M})\boldsymbol{B}(\boldsymbol{I} - \boldsymbol{M})$$

3. Compute the eigenvalue decomposition $\tilde{\boldsymbol{S}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$
4. The p-dimensional representation is obtained by computing the product

$$\hat{\boldsymbol{Z}} = \boldsymbol{I}_{p\times N}\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^*$$

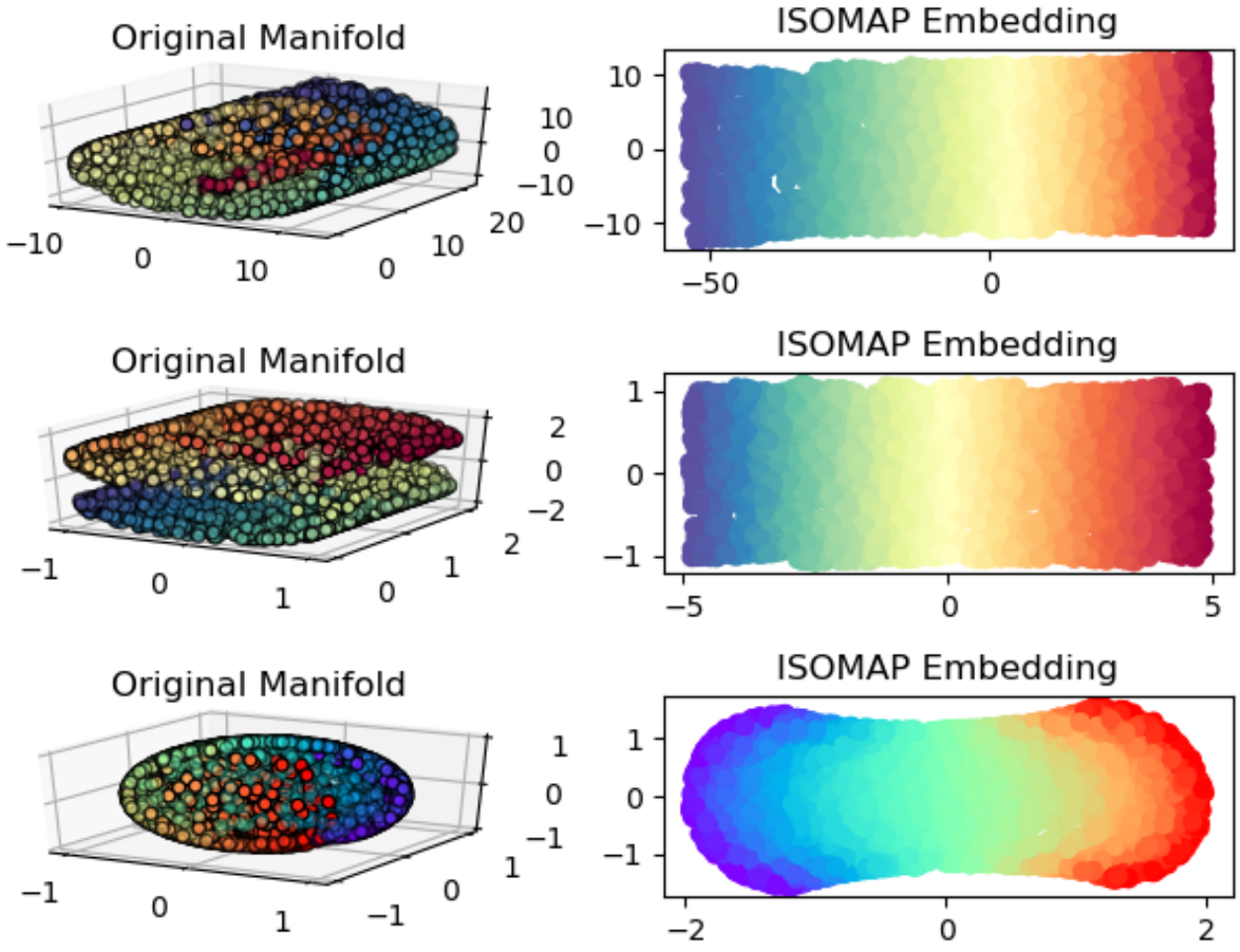

Original Manifold
MDS Embedding
Original Manifold
MDS Embedding
Original Manifold
MDS Embedding

Isometric feature mapping I (ISOMAP)

- Isometric feature mappings (ISOMAP) is the simplest non linear dimensionality reduction approach that uses graph distances as an approximation to geodesic distances.
- The original ISOMAP follows from replacing the Euclidean distance in Classical multidimensional scaling by a graph distance.
- During the first step, the method determines which points are neighbors on the manifold based on pairwise distances between points in the original/input space.

Isometric feature mapping II (ISOMAP)

- The two most common approaches are to consider the points lying in a given Euclidean ball around each point $\boldsymbol{x}_i$, or to simply consider the neighborhood to be the K nearest neighbors.
- The method then builds a graph $\mathcal{G}$ by connecting neighbouring points with edges of weights $w_{ij} = d(i, j)$.
- The second step estimates the pairwise geodesic distances $d_{\mathcal{M}}(i, j)$ of the prototypes by computing the shortest path distances in the graph $\mathcal{G}$.
- Finally, the last step applies classical Multidimensional scaling to the matrix of graph distances $D_{i,j} = (d_{\mathcal{G}}(i, j))$


Original Manifold
ISOMAP Embedding
Original Manifold
ISOMAP Embedding
Original Manifold
ISOMAP Embedding

Locally linear embedding (LLE)

- Locally linear embedding is a topology preserving method which is originally due to Roweis and Saul
- In LLE the idea is to define a lower dimensional embedding but keep the local affine structure of the data
- Locally linear embedding then relies on the assumption that the manifold underlying the data is locally (i.e at the level of the local K-neighborhood) linear (which seems a reasonable assumption when the dataset is large enough and not too noisy).

Locally linear embedding (LLE)

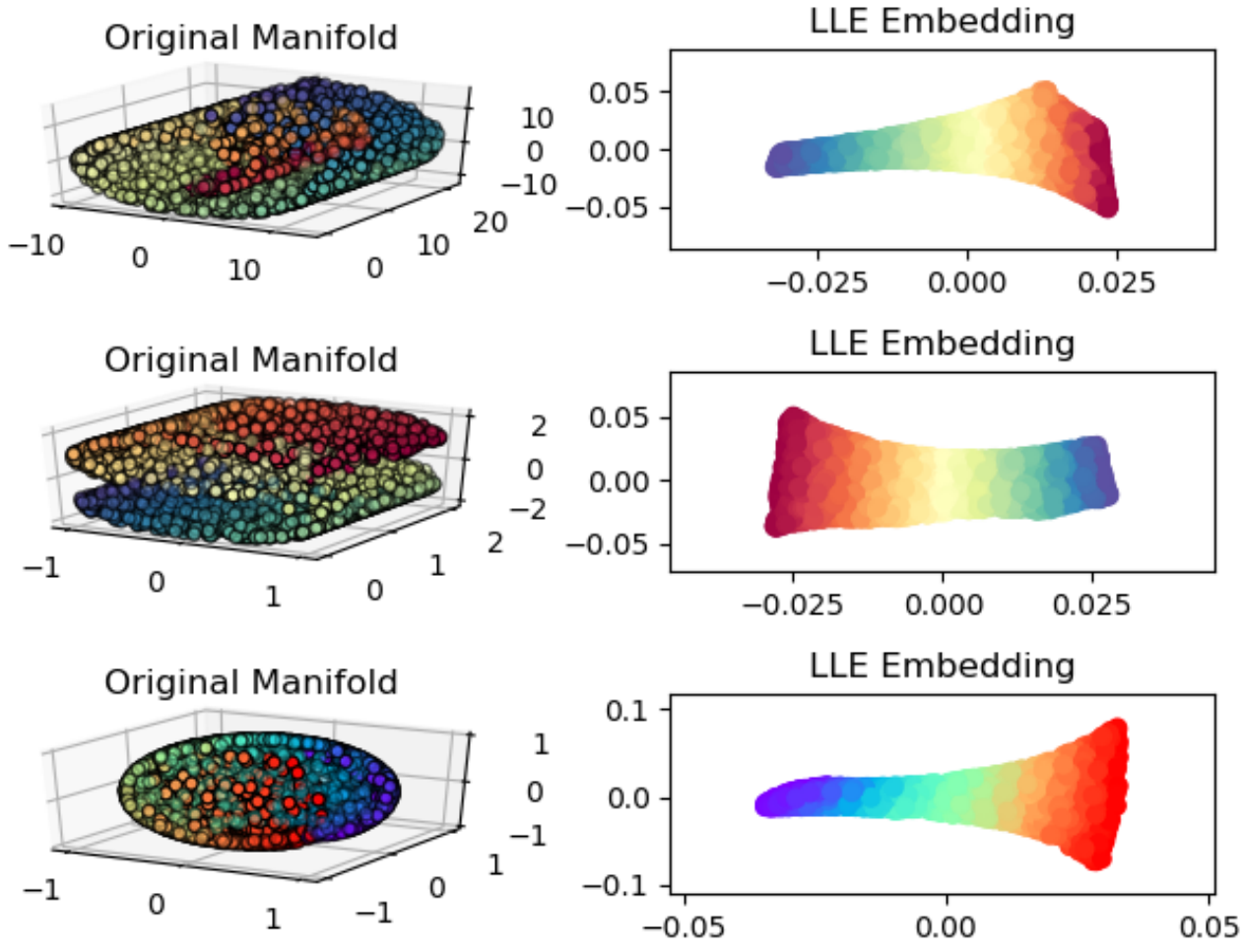
- After computing the neighborhoods, the second step in LLE then computes the linear combination of the K neighbours that best approximates every given point $\boldsymbol{x}_i$ from the dataset,

$$\min_{w} \|\boldsymbol{x}_i - \sum_{k \in \mathcal{N}(\boldsymbol{x}_i)} w_{i,k} \boldsymbol{x}_k\|^2.$$

  Here we let $\mathcal{N}(\boldsymbol{x}_i)$ to denote the k nearest neighbors of $\boldsymbol{x}_i$.

- The low dimensional representation is then given by

$$\min_{\boldsymbol{z}} \|\boldsymbol{z}_i - \sum_{k \in \mathcal{N}(\boldsymbol{z}_i)} w_{i,k} \boldsymbol{z}_k\|^2.$$


Original Manifold
LLE Embedding
Original Manifold
LLE Embedding
Original Manifold
LLE Embedding

Self organizing maps I (SOM)

- The idea underlying Self Organizing Maps goes back to the work of von der Malsburg in 1973.
- The idea which was then developed as a way to better understand the visual cortex then remained under the radar until the 80's, when Kohonen came with a simplification of the ideas of von der Malsburg and provided an easy implementation.
- Because of their relative simplicity, SOMs are used in a variety of areas including time series prediction and data visualization.

Self organizing maps II (SOM)

- In classical quantization, a smaller set of prototypes is used to approximate (i.e to fit) a larger set of points from the original space. However, the prototypes are free to move independently of each other.
- The idea of SOMs is similar except that in SOM all prototypes are connected to each other through a lattice (i.e the prototypes are living on a N-D lattice) also called Constrained topological map.
- Each time a prototype is updated and moved, its neighbors on the lattice move accordingly.
- By doing this, we ensure that the points in the dataset corresponding to points that are close on the lattice, will be close to each other as well.

Self organizing maps III (SOM)

- Self organizing maps start from a two dimensional rectangular lattice of prototypes $\boldsymbol{z}_j \in \mathbb{R}^N$.
- The prototypes can be initialized for example as lying on the two dimensional principal component plane of the data (the prototypes are then often chosen to be equispaced on that plane). The SOM procedure then slowly bends the plane/lattice to match it as much as possible with the original data.
- In the case of SOM, the lattice thus plays the role of the embedding space. SOM runs through the dataset multiple times (performing multiple *epochs*).

Self organizing maps IV (SOM)

- For any given point $\boldsymbol{x}_i$ from the original dataset, the first step determines the index of the closest point on the lattice,

$$\gamma^* = \underset{\gamma}{\text{argmin}} d(\boldsymbol{x}_i, \boldsymbol{z}(\gamma))$$

  Here d is the Euclidean distance.

- The second step then updates the prototypes $\boldsymbol{z}(\gamma)$ as

$$\boldsymbol{z}(\gamma) \leftarrow \boldsymbol{z}(\gamma) + \alpha \mathcal{N}(\gamma, \zeta)(\boldsymbol{x}_i - \boldsymbol{z}(\gamma))$$

- $\mathcal{N}(\gamma, \zeta)$ is called the neighborhood function. This function is defined by a (threshold) parameter $\mathcal{N} = \mathcal{N}_\theta$ and was defined in the original von der Malsburg paper to be the *Bubble function*,

$$\mathcal{N}(\gamma_1, \gamma_2; \theta) = \begin{cases} 0 & \text{when } d_{\mathcal{G}}(\gamma_1, \gamma_2) > \theta \\ 1 & \text{when } d_{\mathcal{G}}(\gamma_1, \gamma_2) \leq \theta \end{cases}$$

Self organizing maps V (SOM)

- The *neighborhood function* is also sometimes defined from the gaussian kernel, as

$$\mathcal{N}_\theta(\gamma_1, \gamma_2) = \exp\left(-\frac{d^2_{\mathcal{G}(\gamma_1,\gamma_2)}}{2\theta^2}\right)$$

  The distance $d_{\mathcal{G}}$ is often chosen to be any distance defined on the lower dimensional space, i.e., $d_{\mathcal{G}}(\gamma_1, \gamma_2) = d(\gamma_1, \gamma_2)$. The threshold parameter θ is also often chosen to decrease with the epochs from a starting value θ_0 to 1 (over about a thousand iterations).

Self organizing maps IV (SOM)

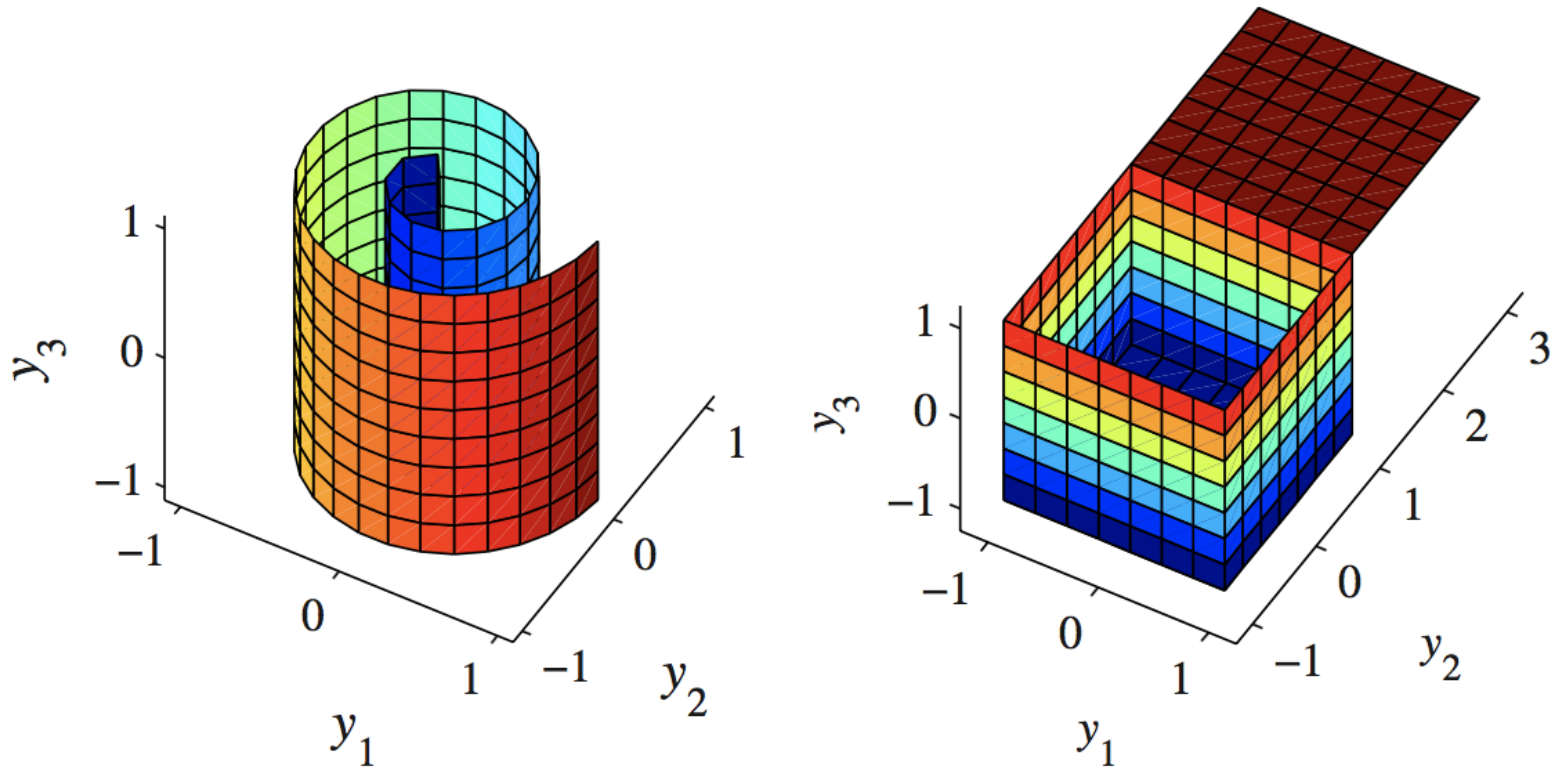
- At the end of the iterations, the mapping

$$f \ : \ \boldsymbol{x} \mapsto \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{z}(\gamma_{1,\boldsymbol{x}}, \ldots, \gamma_{P,\boldsymbol{x}}) = \boldsymbol{z}(\boldsymbol{\gamma}_{\boldsymbol{x}})$$

  from any point of the original dataset to the points on the lattice is obtained by simply taking the nearest prototype on the lattice, i.e.

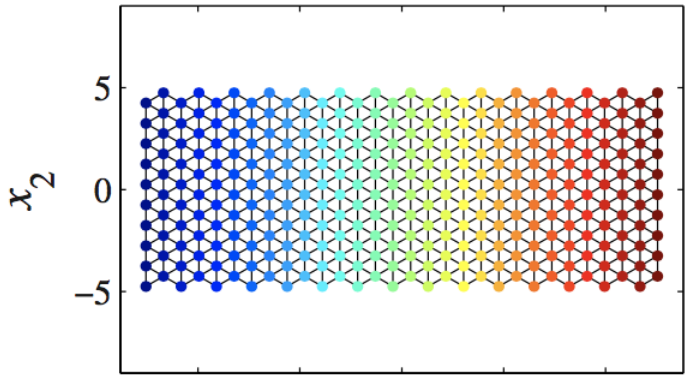
$$f(\boldsymbol{x}_i) = \boldsymbol{z}(\boldsymbol{\gamma}) \quad \text{with} \quad \boldsymbol{\gamma} = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} d(\boldsymbol{x}_i, \boldsymbol{z}(\boldsymbol{\gamma})).$$

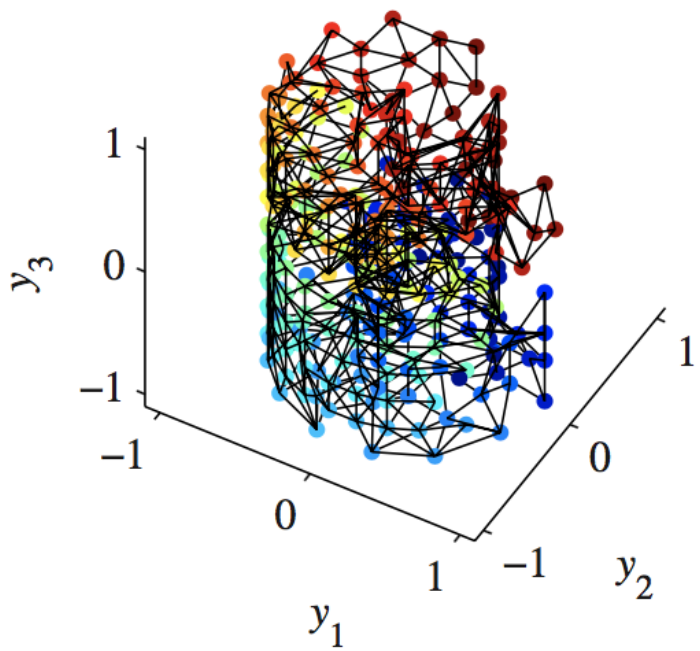
SOM: Original manifolds

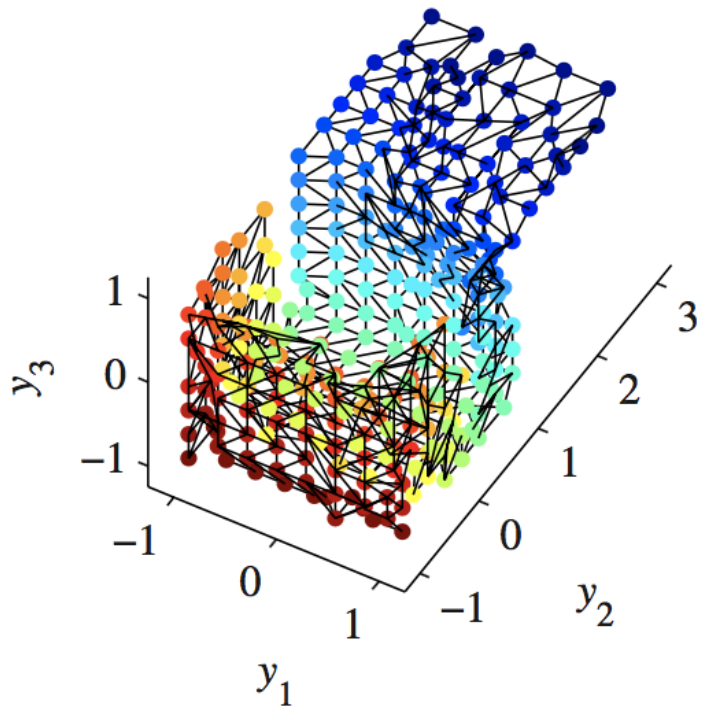


source: Lee and Verleysen, Non Linear Dimensionality reduction

SOM: lattice definition and embedding







source: Lee and Verleysen, Non Linear Dimensionality reduction

Laplacian eigenmaps

- The notion of Laplacian eigenmap was first introduced by Belkin (2003). Just as in Isomap, the method starts by building a graph representation of the data
- The difference is that the algorithm builds the lower dimensional embedding of the data by using the Laplacian of the graph.
- Moreover it is based on the minimization of the local distances. In that sense the algorithm can be considered as belonging to the class of *spectral decomposition* NLDR techniques.
- Laplacian eigenmaps can in fact be considered as an intermediate between ISOMAP (which uses graph distances but in a global framework) and locally linear embeddings which use Euclidean distances but locally.

Laplacian eigenmaps: constraints

- To compute the lower dimensional embedding, the method can be understood as minimizing a criterion of the form

$$\ell(\boldsymbol{z}) = \frac{1}{2} \sum_{i,j=1}^{m} \|\boldsymbol{y}_i - \boldsymbol{y}_j\| w_{ij}$$

under appropriate constraints.

- When the w_{ij} represent the edges of the similarity graph, we get the following decomposition

$$\begin{aligned} \sum_{i,j} (\boldsymbol{y}_i - \boldsymbol{y}_j)^2 W_{ij} &= \sum_i y_i^2 D_{ii} + \sum_j y_j^2 D_{jj} - 2 \sum_{i,j} y_i y_j W_{ij} \\ &= 2\boldsymbol{y}^T \boldsymbol{L} \boldsymbol{y} \end{aligned}$$

which is based on the graph Laplacian $L = D - W$, $D_{ii} = \sum_j D_{ij}$.

Laplacian eigenmaps: constraints

$$\min_{\boldsymbol{y}^T \boldsymbol{D} \boldsymbol{y} = 1} \ell(\boldsymbol{z}) = \min_{\boldsymbol{y}^T \boldsymbol{D} \boldsymbol{y} = 1} \frac{1}{2} \sum_{i,j=1}^{m} \|\boldsymbol{y}_i - \boldsymbol{y}_j\| w_{ij} = \min_{\boldsymbol{y}^T \boldsymbol{D} \boldsymbol{y} = 1} 2 \boldsymbol{y}^T \boldsymbol{L} \boldsymbol{y}$$

- The first constraint $\boldsymbol{y}^T \boldsymbol{D} \boldsymbol{y}$ removes arbitrary scalings (including $\boldsymbol{y} = \mathbf{0}$) by requiring the vectors $\boldsymbol{y}$ to be normalized.
- Finding the optimal embedding in the above framework is thus equivalent to finding the eigenvectors corresponding to the smallest eigenvalues of the graph Laplacian.
- Any graph always admits $\mathbf{1}$ as a trivial eigenvector of the Laplacian (i.e $\boldsymbol{L} \cdot \mathbf{1} = 0$). If the graph is connected, $\mathbf{1}$ is the only trivial eigenvector.
- We can thus remove the trivial solution by only retaining the eigenvectors that are orthogonal to $\mathbf{1}$, $\langle \boldsymbol{y}, \mathbf{1} \rangle = 0$.

Laplacian eigenmaps: general algorithm (Part I)

- Build a graph by putting an edge between points i and j if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are close. There are two approaches at defining distances

  1. ε-neighborhoods. Two nodes i and j are connected by an edge if $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 < \varepsilon$ (for the usual Euclidean norm)
  2. K nearest neighbors. Here two nodes i and j are connected by an edge if i is among the K nearest neighbors of j.

- Weighting of the edges. There are again two possible variations

  1. Heat kernel. Whenever the nodes i and j are connected, set the distance as

  $$W_{ij} = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/t)$$

  Where $t \in \mathbb{R}$ is a parameter. Otherwise, set $W_{ij} = 0$.

  2. Hard thresholding. Simply set $W_{ij} = 1$ if points i and j are connected by an edge and $W_{ij} = 0$ otherwise.

Laplacian eigenmaps: general algorithm (Part II)

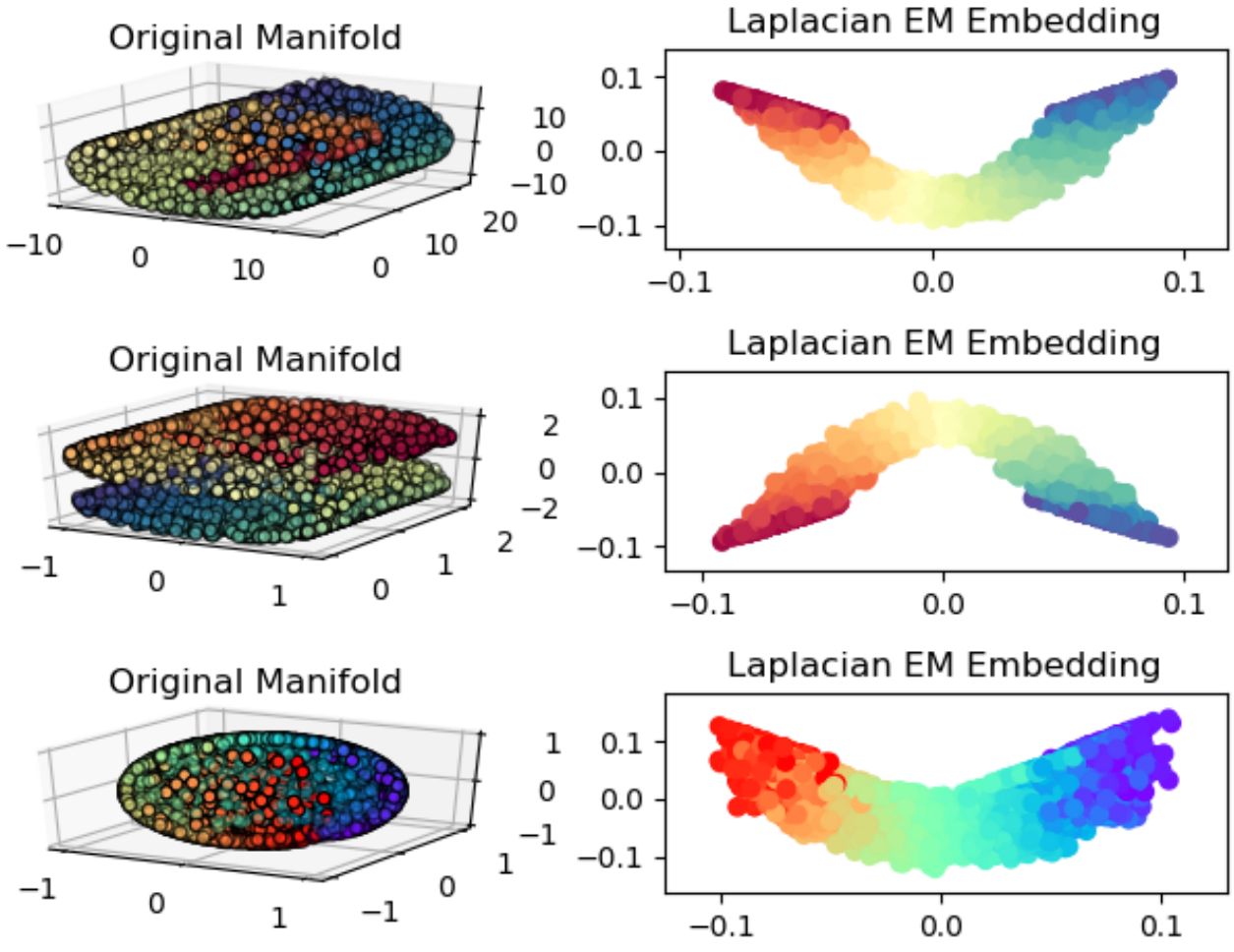
- Assume the graph $\mathcal{G}$ constructed above is connected, otherwise proceed with each connected component.
- Compute eigenvalues and eigenvectors of the generalized eigenvector problem,

$$L\boldsymbol{v} = \lambda D\boldsymbol{v}$$

  - D is the diagonal weight matrix given by the sum $D_{ii} = \sum_j W_{ji}$,
  - L is the Laplacian matrix $L = D - W$ (positive semidefinite) which can be thought of as an operator on functions defined on vertices of G.

- Let $0 = \lambda_0 \leq \lambda_1, \leq \ldots, \lambda_m$ to denote the solutions to the eigenvalue problem above so that $L\boldsymbol{v}_0 = \lambda_0 D\boldsymbol{v}_0$,... $L\boldsymbol{v}_m = \lambda_m D\boldsymbol{v}_m$. We then define the embedding as

$$\boldsymbol{x}_i = (\boldsymbol{v}_1[i], \ldots, \boldsymbol{v}_m[i])$$

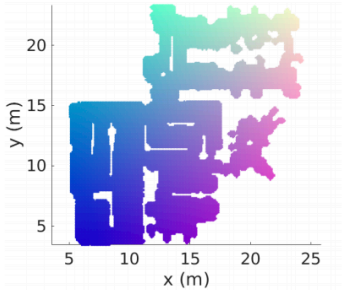

Original Manifold
Laplacian EM Embedding
Original Manifold
Laplacian EM Embedding
Original Manifold
Laplacian EM Embedding

Diffusion maps

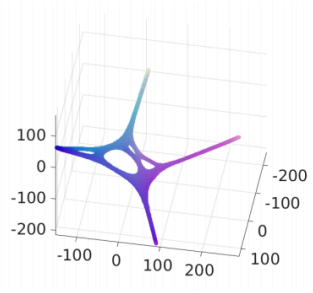
- Many robotic applications require repeated on demand motion planning in mapped environments in the presence of other dynamic agents
- Having a function encoding pairwise cost-to-go is important for finding feasible paths
- Computing and storing pairwise potentials can be impractical as it requires computing every query to any given new goal
- To reduce this computational complexity, it is possible to learn the map's geometry and develop a memory efficient parametrization
- Each state in the map is turned into a diffusion coordinate and the states are then compared through traditional Euclidean distance.

source: YF Chen et al., Motion Planning with Diffusion Maps.

Diffusion maps



(a) 2D gridmap



(b) diffusion space

Fig. 1: A robotic vehicle navigates autonomously in a dynamic indoor environment. The vehicle uses a SICK Lidar for localization, and a Velodyne for obstacle detection.

source: YF Chen et al., Motion Planning with Diffusion Maps.

Diffusion maps: Theory

- Diffusion maps (Coifman 2006) rely on a combination of graph representation and Markov chain sampling (or equivalently random walks on the graph)
- A diffusion maps embeds data into a lower dimensional space where Euclidean distance between points approximates the diffusion distance in the original feature space
- Given a graph representation of a manifold and standing at one of the vertices, it is more likely to go to one of the neighbors of this vertex than to one that is far away
- We thus interpret connectivity (and similarity between features) as a measure of probability

Source: Coifman and Lafon, Diffusion maps

de la Porte et al. An Introduction to Diffusion Maps

Diffusion maps: Theory

- We thus interpret connectivity (and similarity between features) as a measure of probability,

$$p(x, y) \propto \text{connectivity}(x, y) \propto k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{\alpha})$$

- Here we decided to use the Gaussian kernel, although any other kernel would appropriate

- To keep a probability measure, we normalize the connectivity as

$$\begin{aligned} p(x, y) &= \text{connectivity}(x, y) \\ &= \frac{1}{\sum_{y \in X} k(x, y)} k(\boldsymbol{x}, \boldsymbol{y}) \\ &= \frac{1}{\sum_{y \in X} k(x, y)} \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{\alpha}) \end{aligned}$$

Source: Coifman and Lafon, Diffusion maps

de la Porte et al. An Introduction to Diffusion Maps

Diffusion maps: Theory

- Given the probability $p(\boldsymbol{x}, \boldsymbol{y})$ we introduce the row normalized (Markov) transition/diffusion matrix $\boldsymbol{\Pi}$,

$$\boldsymbol{\Pi}_{ij} = p(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

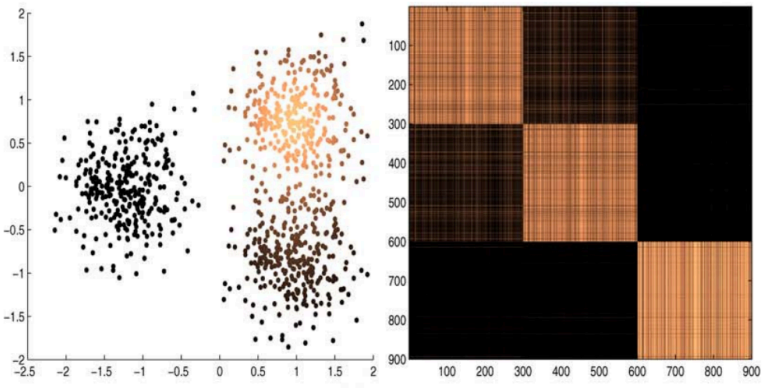
- The matrix $\boldsymbol{\Pi}$ encodes some local information on how likely it is to move from $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$ when following a random walk of length 1.

- Similarly, $(\boldsymbol{\Pi})^t_{ij}$ gives the probability to go from points $\boldsymbol{x}_i$ to point $\boldsymbol{x}_j$ when following a random walk of length t.

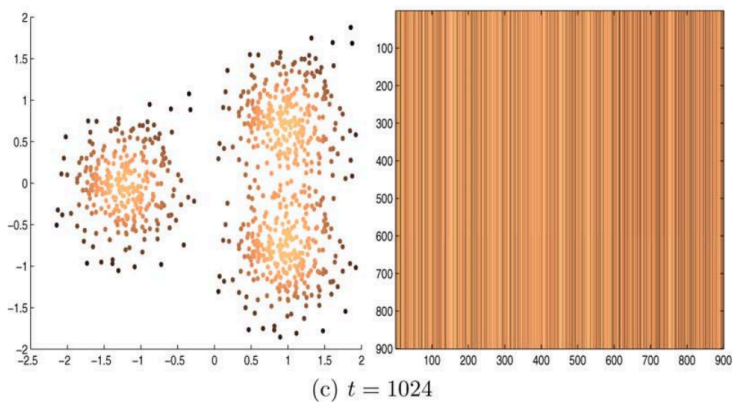
- Diffusion maps use t as a scale parameter. Running the chain forward in time (i.e. taking larger powers of $\boldsymbol{\Pi}$) will reveal geometric features of the dataset at different scales.

Diffusion maps: Theory

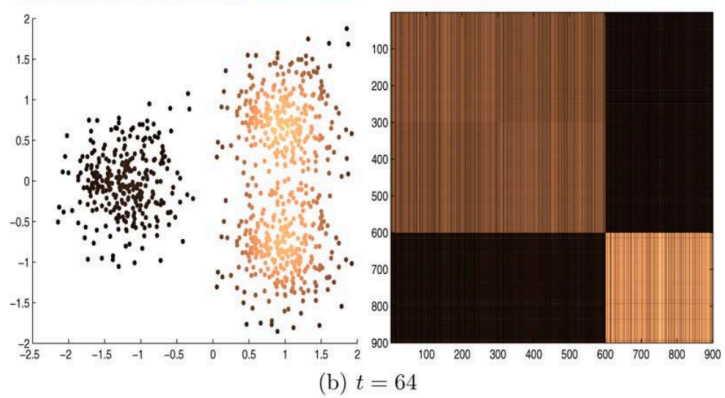
- The multiscale nature of diffusion maps is illustrated through the following example taken from Coifman and Lafon, 2006. In this example, the transition matrix at times $t = 8$, $t = 64$, $t = 1024$. Note how the local structure of the dataset completely vanishes at large t.



(a) $t = 8$



(c) $t = 1024$



(b) $t = 64$

Source: Coifman and Lafon, Diffusion maps

Diffusion maps: Theory

- So far we have spoken about random walks and paths in the graph representing the data but we haven't discussed the embedding.

- Given the transition matrix, we define the diffusion distance as

$$\begin{aligned} D_t(x,y) &\equiv \|p_t(x,\cdot) - p_t(y,\cdot)\|_{L_2}^2 \\ &= \int_X (p_t(x,u) - p_t(y,u))^2 \frac{d\mu(u)}{\pi(u)} \end{aligned}$$

- $D_t(x,y)$ will be small when there is a large number of paths between $\boldsymbol{x}$ and $\boldsymbol{y}$. That is when for a large number of vertices u in the graph it is very likely to go from $\boldsymbol{x}$ to $\boldsymbol{u}$ and similarly likely to go from $\boldsymbol{u}$ to $\boldsymbol{y}$. (I.e there is a community)

- On the other hand, if it is very likely to go from $\boldsymbol{x}$ to a whole subset of $\boldsymbol{u}$ at which it is very unlikely to reach $\boldsymbol{y}$, that probably means $\boldsymbol{x}$ and $\boldsymbol{y}$ are in different communities.

Diffusion maps: The Algorithm

- Choose a kernel (e.g. $k_\varepsilon = e^{-\|x-y\|^2/\varepsilon}$)
- Set $d(x) = \int_X k(x,y)\, d\mu(y)$ or $d(x) = \sum_{y \in X} k(x,y)$ (discrete) and define the transition matrix of the chain as

$$\mathbf{\Pi}_{i,j} = p(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{k(\boldsymbol{x}_i, \boldsymbol{x}_j)}{d(\boldsymbol{x})}$$

- Compute the eigenvalue decomposition of the transition matrix $\mathbf{\Pi}$ (let us label the eigenvalues and eigenvectors as (λ_k, ψ_k))
- Finally define the embedding $\Psi(t)$, at each scale t, as

$$\Psi_t(x) = \begin{pmatrix} \lambda_1^t \psi_1(x) \\ \lambda_2^t \psi_2(x) \\ \vdots \\ \lambda_s^t \psi(s)(x) \end{pmatrix}$$

- x encodes the Position/index in the input space.

Diffusion maps: wrapping up

- By choosing a particular value of t, Diffusion maps thus enable to select a particular scale for our analysis of the data.
- Diffusion maps can be understood as going one step further than other manifold learning methods as they as they also require us to specify which scale we want the embedding to represent
- In practice the diffusion distance has a discrete form

$$\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 \equiv \sum_{u \in \mathcal{D}} |p_t(\boldsymbol{x}_i, \boldsymbol{u}) - p_t(\boldsymbol{x}_j, \boldsymbol{u})|^2$$