Graphs for data science and ML

Machine Learning for graphs and with graphs

P. Borgnat, CNRS, LP ENSL (4)
Acknowledgements: some slides taken from P. Vandergheynst (EPFL), and from R. Cazabet (Univ Lyon 1)

Exploit the properties of the matrices of graphs

Fourth: try to visualise the graphs

- **Graph drawing, or Graph visualisation**, is an old problem
  - -> It was to find the best layout of a graph, to capture will its structure
  - historically: with low dimensional spaces (2D, 3D) => Gephi, Graphviz,…
- Then came the age of **Representation Learning:**
  - Find features, or latent space, in which the data is represented
  - At the heart of ML with Neural Networks for graphs: learn features to code best for the inner structures of the graph (or node) (& its attributes)

Pre-CNN methods of embeddings

- Use “physical models” for graph layout (e.g., Force layout, kamada-kawai)
  - -> Principle: put connected nodes close, non-connected nodes far away
- Use the properties the Laplacian to create a smooth embedding of the nodes
  - -> Laplacian eigenmaps
- LLE: Locally Linear Embedding
- Random Walked-based embeddings: DeepWalk, Node2Vec
  - -> Welcome to a brand new world: **learn a high-dimensional representation**

1) Physical models of graph layout

- Implemented in data/graph vizualization packages or softwares
- Often for practitioners in network science, and valid!

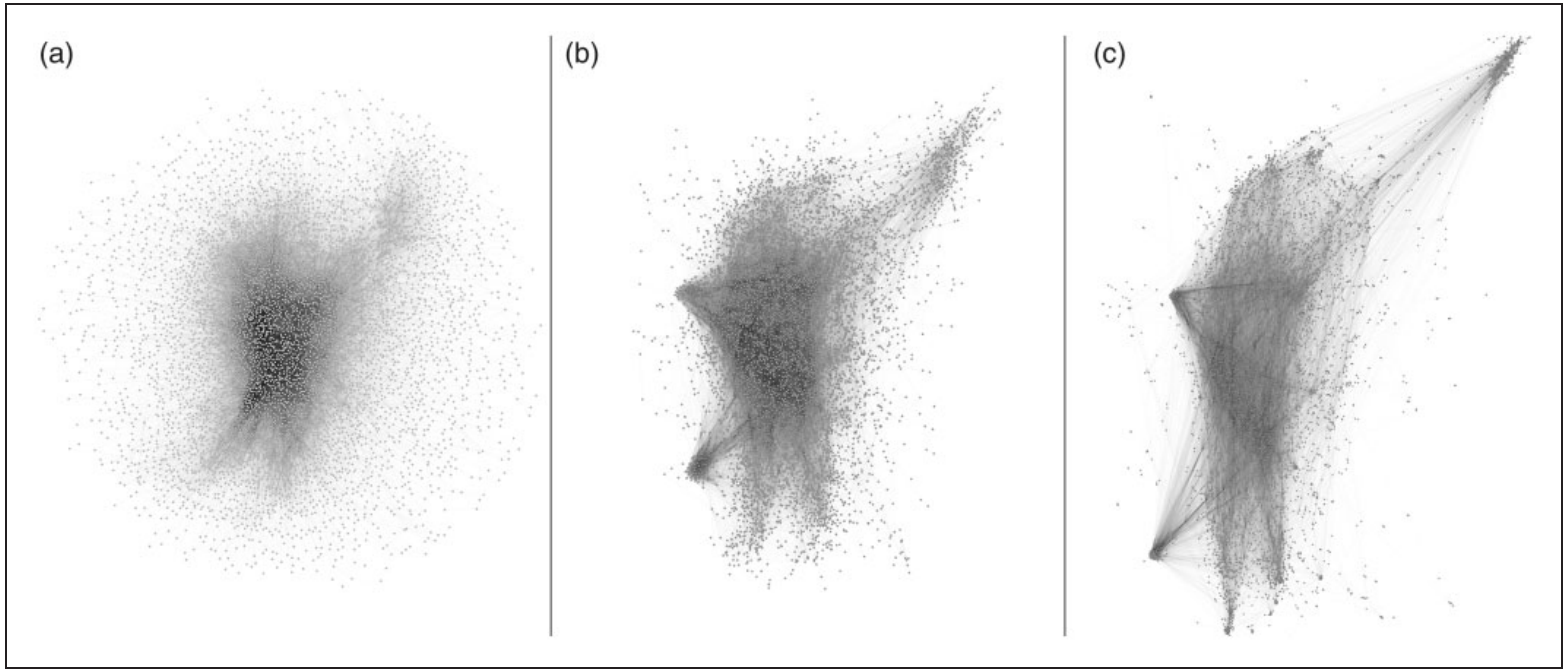


Figure 3. The "jazz network" spatialized (a) with the algorithm proposed by Fruchterman and Reingold (1991), (b) with ForceAtlas2 (with default parameters) and (c) with ForceAtlas2 with tweaked parameters for LinLog mode and gravity. This and all images created for this paper are available at: https://github.com/tommv/ForceDirectedLayouts.

What do we see when we look at networks: Visual network analysis, relational ambiguity, and force-directed layouts

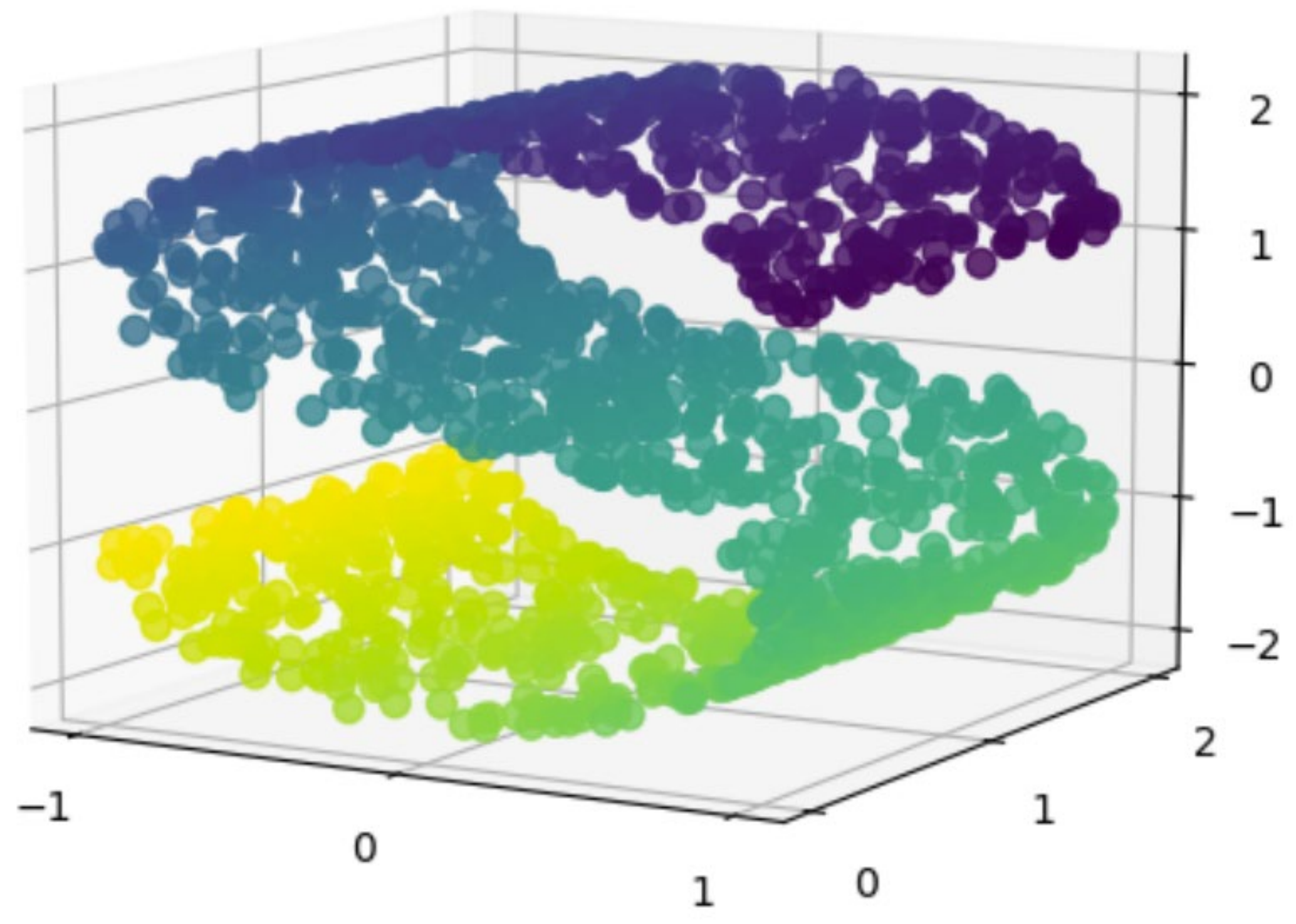
Tommaso Venturini[1], Mathieu Jacomy[2] and Pablo Jensen[3,4]

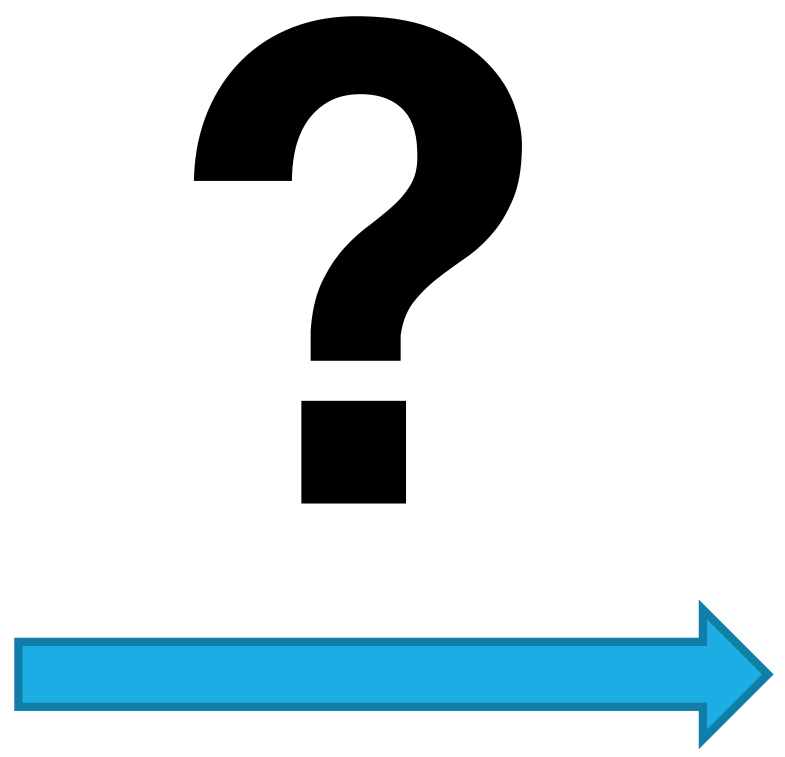
Big Data & Society
January–June: 1–16
© The Author(s) 2021
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/20539517211018488
journals.sagepub.com/home/bds
SAGE

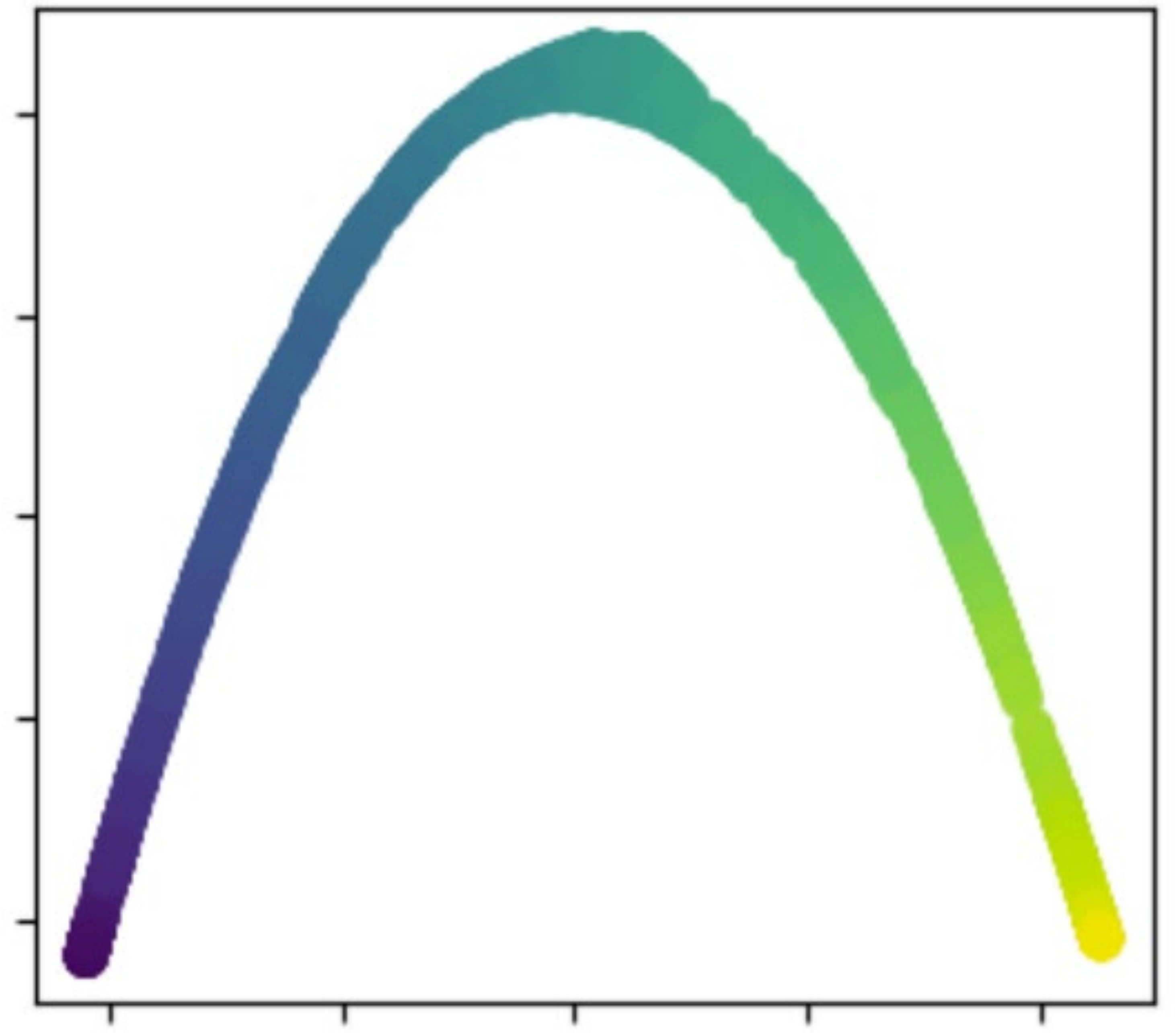
Embeddings of graphs in low dimension

Objective: find new coordinates

Examples from sklearn







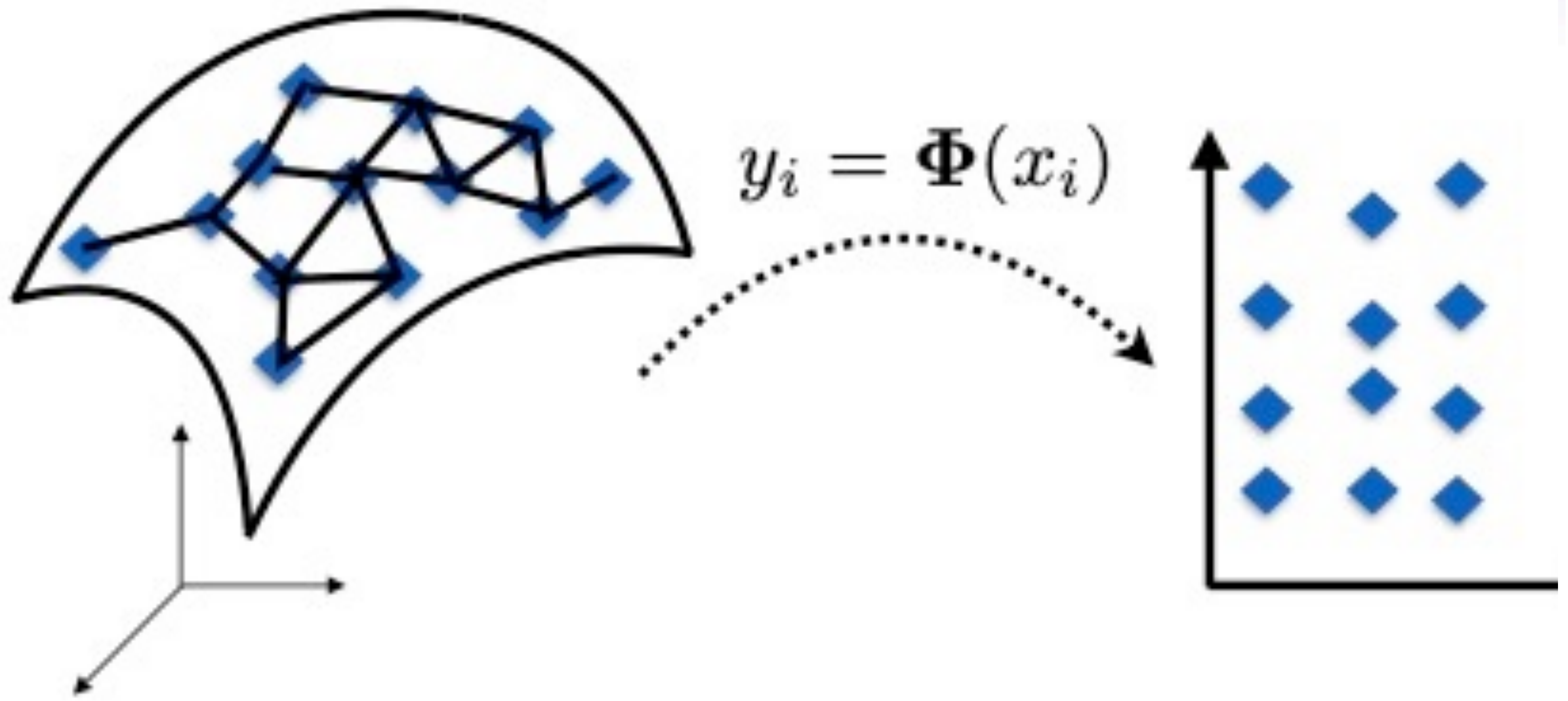
Q: can we learn a low-dimensional embedding (a latent vector for each data point) that preserves the original structure of X ?

2) Laplacian eigenmaps

Objective: embeddings of graphs from spectral features

- Objective of embedding: embed vertices in low dimensional space, so as to discover geometry

$$x_i \in \mathbb{R}^d \rightarrow y_i \in \mathbb{R}^k \text{ with } k < d$$



2) Laplacian eigenmaps

Objective: embeddings of graphs from spectral features

- Two starting points:
  - 1) you already have a graph, whose (weighted) adjacency matrix $\mathbf{A}$ or $\mathbf{W}$ captures (*sparse ?)* similarities between nodes,
  - 2) you have data points in high dimension, with coordinates $X \in \mathbb{R}^{N \times L}$,
    - N is the number of data points (= nodes) and L the dim. of features (=coord.)
    - => build a similarity graph, then you are back to point 1)

Create a graph to represent the data

Objective: capture similarities between data points

- This is a standard step in classification / clustering!
- Hence, several manners to code these similarities in a graph:

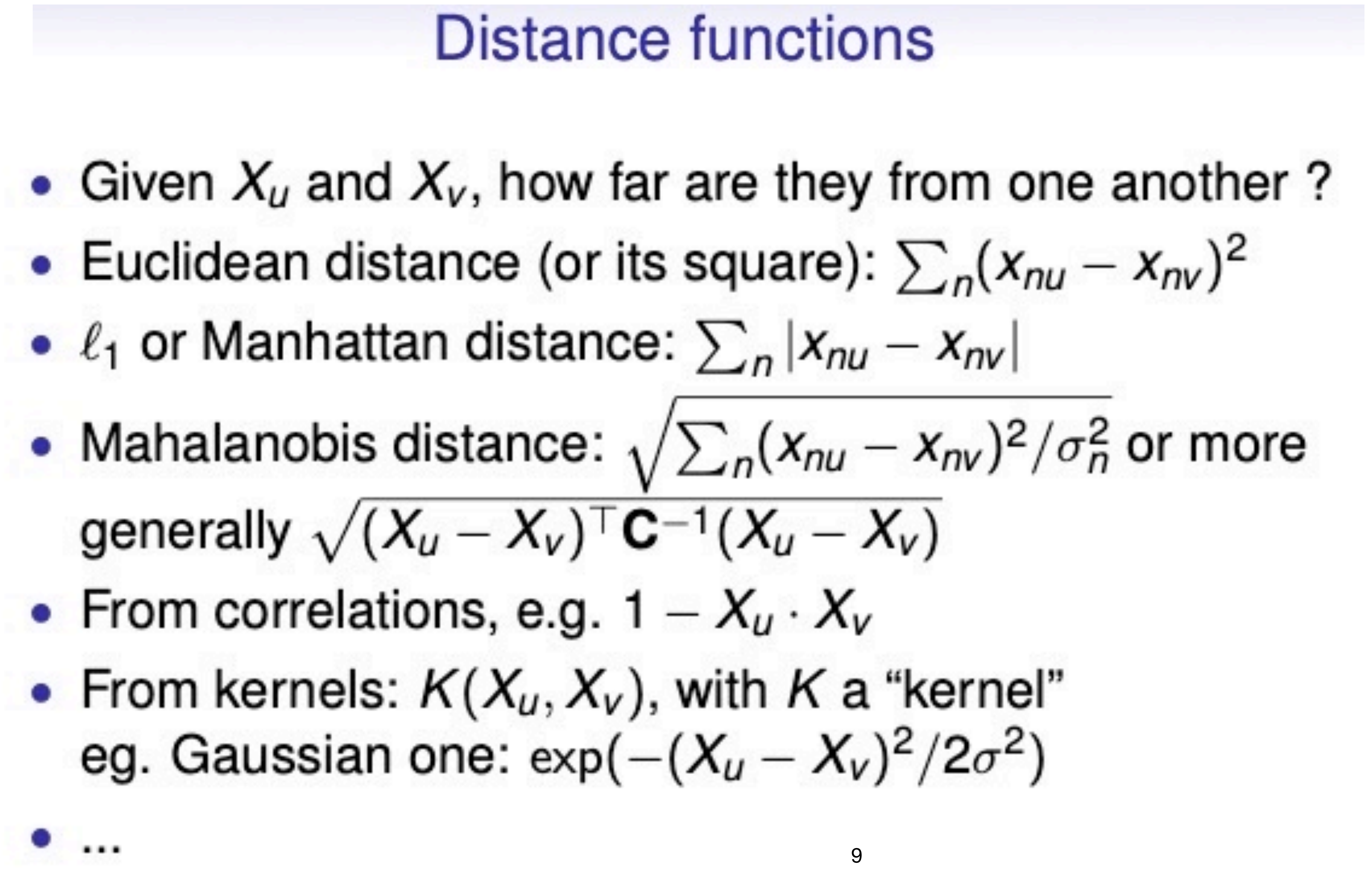
$$\text{selecting k-nearest neighbours of each point with distance } d(x_i, x_j)$$

OR

$$\text{selecting all points in a neighbourhood } d(x_i, x_j) \leqslant \epsilon$$

Create a graph to represent the data

Objective: capture similarities between data points



Distance functions

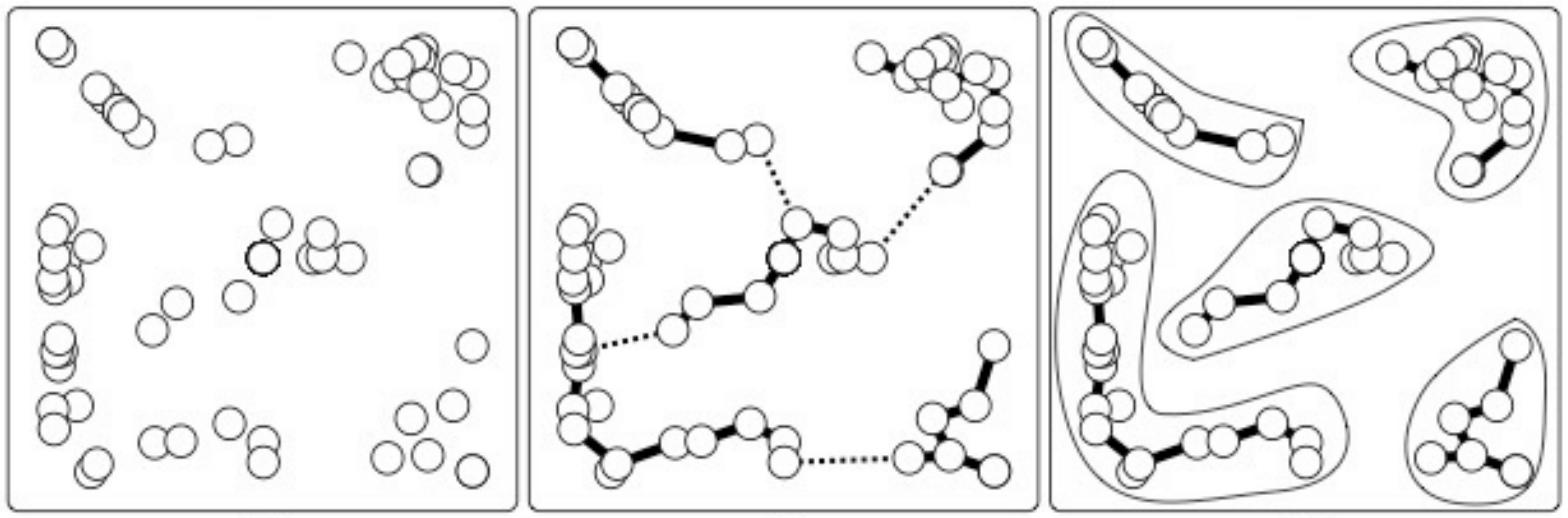
- Given X_u and X_v, how far are they from one another ?
- Euclidean distance (or its square): $\sum_n (x_{nu} - x_{nv})^2$
- ℓ_1 or Manhattan distance: $\sum_n |x_{nu} - x_{nv}|$
- Mahalanobis distance: $\sqrt{\sum_n (x_{nu} - x_{nv})^2 / \sigma_n^2}$ or more generally $\sqrt{(X_u - X_v)^\top \mathbf{C}^{-1} (X_u - X_v)}$
- From correlations, e.g. $1 - X_u \cdot X_v$
- From kernels: $K(X_u, X_v)$, with K a “kernel”
  eg. Gaussian one: $\exp(-(X_u - X_v)^2 / 2\sigma^2)$
- ...

Create a graph to represent the data

Objective: keep strong similarities (only) between data points

Great a graph "connecting the dots", i.e. find edges
to connect data points.
Several possibilities:

- **Mininimal Spanning Tree:** the tree with smallest sum of edge lengths connecting all nodes



(1) (2) (3)

MST + Cuts

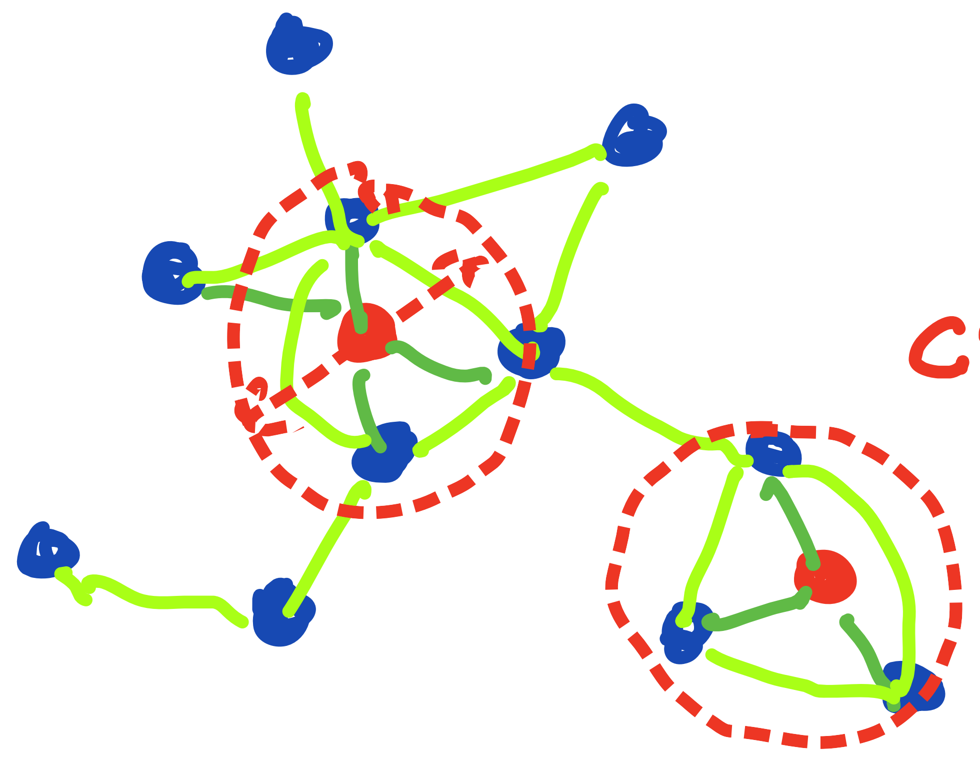
Create a graph to represent the data

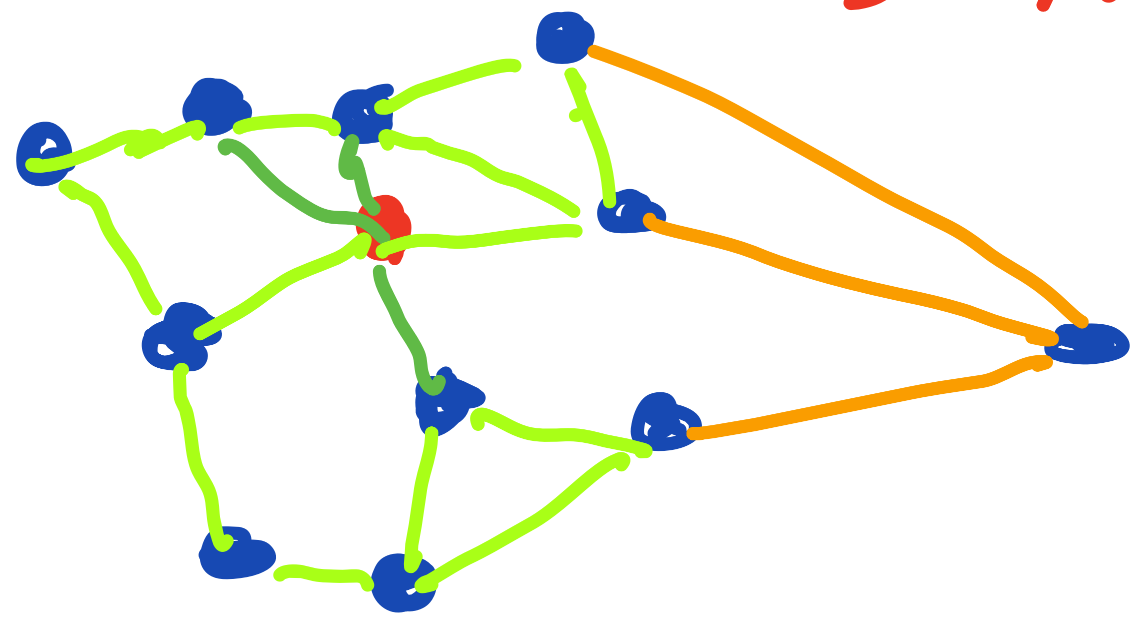
Objective: keep strong similarities (only) between data points

Great a graph "connecting the dots", i.e. find edges
to connect data points.
Several possibilities:

✗ **The ε-neighborhood graph:** $d(x_i, x_j) \leqslant \epsilon$

(1)





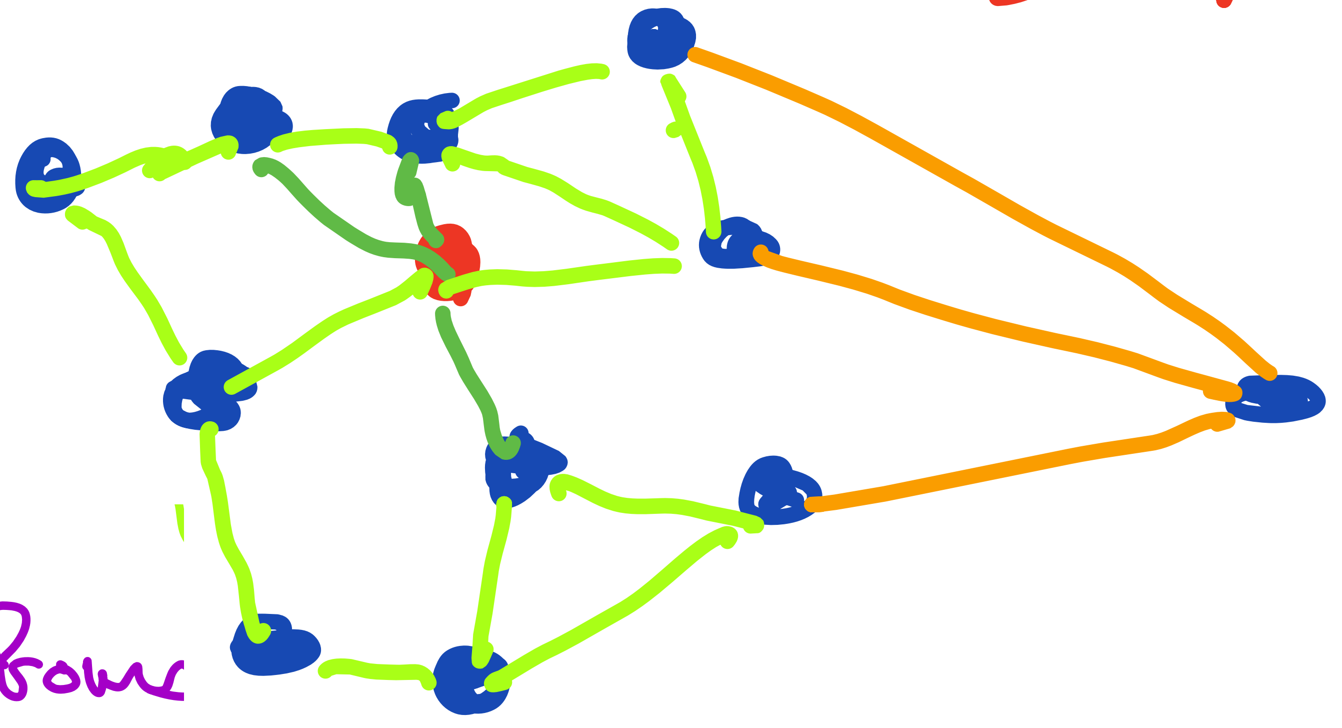
Create a graph to represent the data

Objective: keep strong similarities (only) between data points

Great a graph "connecting the dots", i.e. find edges to connect data points.
Several possibilities:

k-nearest neighbor graphs: with distance $d(x_i, x_j)$

3-NN graph



⚠ with outliers

Note :, $d \gtrsim k$ if symmetrized
. Very sparse if kNN <u>mutual</u> neighbours

Create a graph to represent the data

Objective: keep strong similarities (only?) between data points

Great a graph "connecting the dots", i.e. find edges
to connect data points.
Several possibilities:

The fully connected graph:

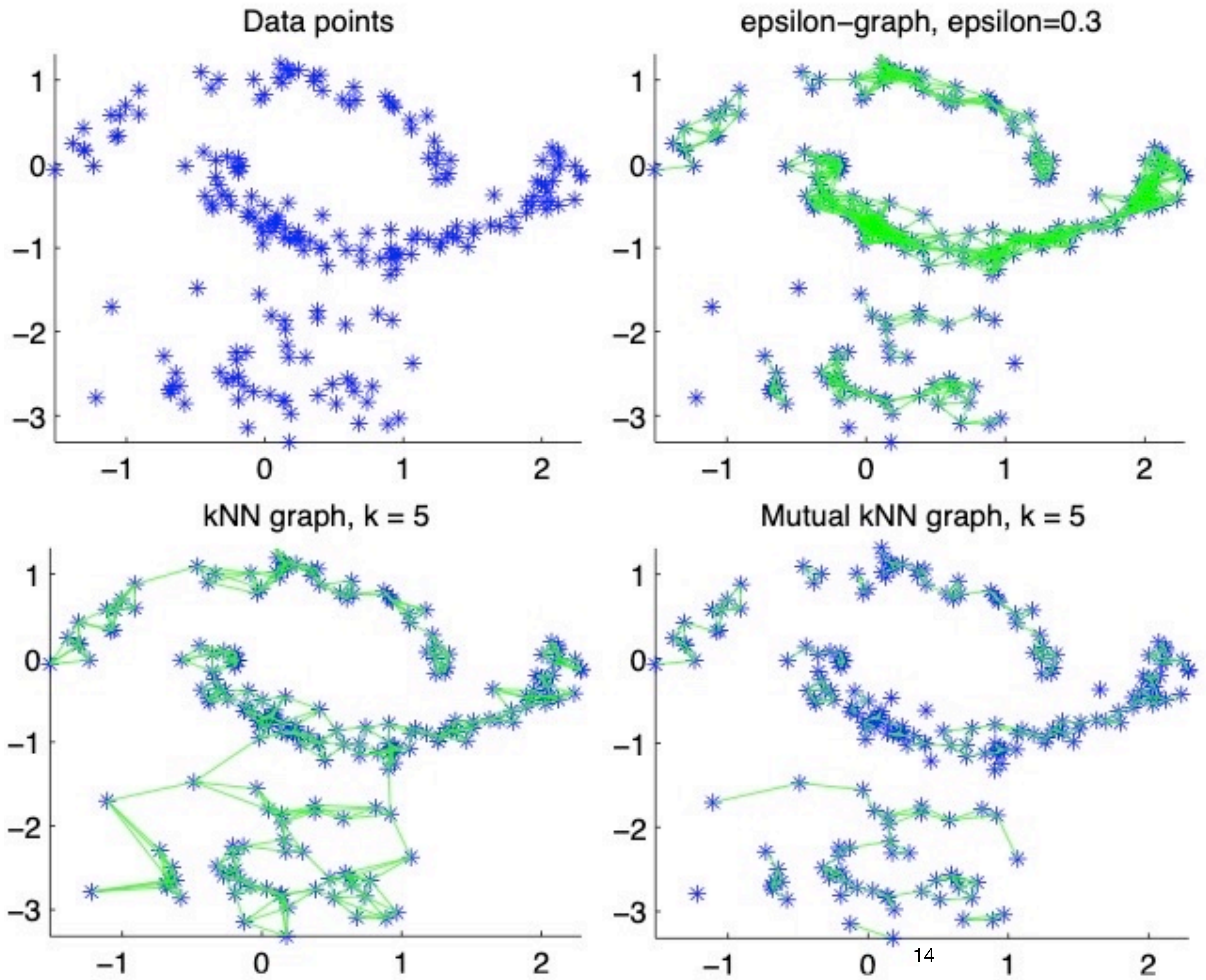
connect all nodes with all other nodes, but with a weight on each
edge, derived from some similarity function, going to 0 if distance goes to infinity

Example: Gaussian similarity function $s(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$

interest of the 3 previous solutions: **sparse graphs** !

for complete graph with similarity kernel: use **thresholding** to increase sparsity of the graph.

Create a graph to represent the data



Examples

A Tutorial on Spectral Clustering, Ulrike von Luxburg,

Create a graph to represent the data

Interlude: you know other methods!

2-a) model the local neighbourhood relationships between the data points

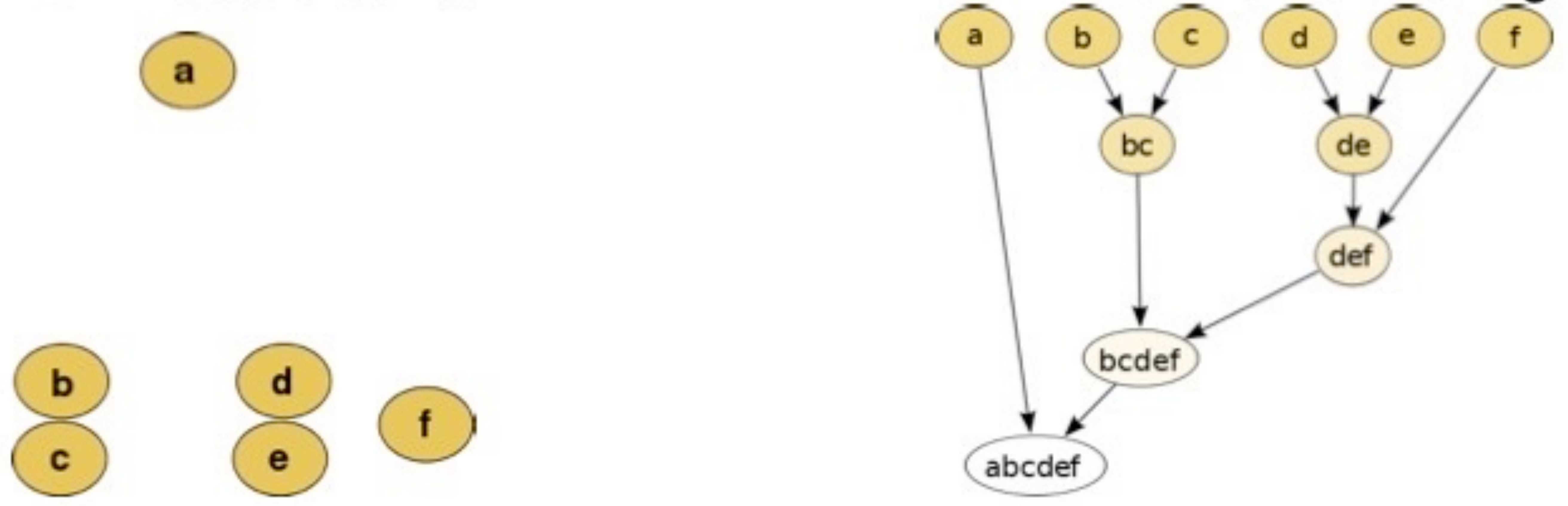
Create a graph to represent the data

Interlude: you know other methods!

2-b) Create a graph that clusters (or classifies) data points
A possible solution: **Hierarchical clustering**

- Main idea: group together closest points

Fom freature domain: to Tree clustering



with two broad strategies: Agglomerative (a "bottom-up" approach) vs. Divisive (a "top-down" approach)

Create a graph to represent the data

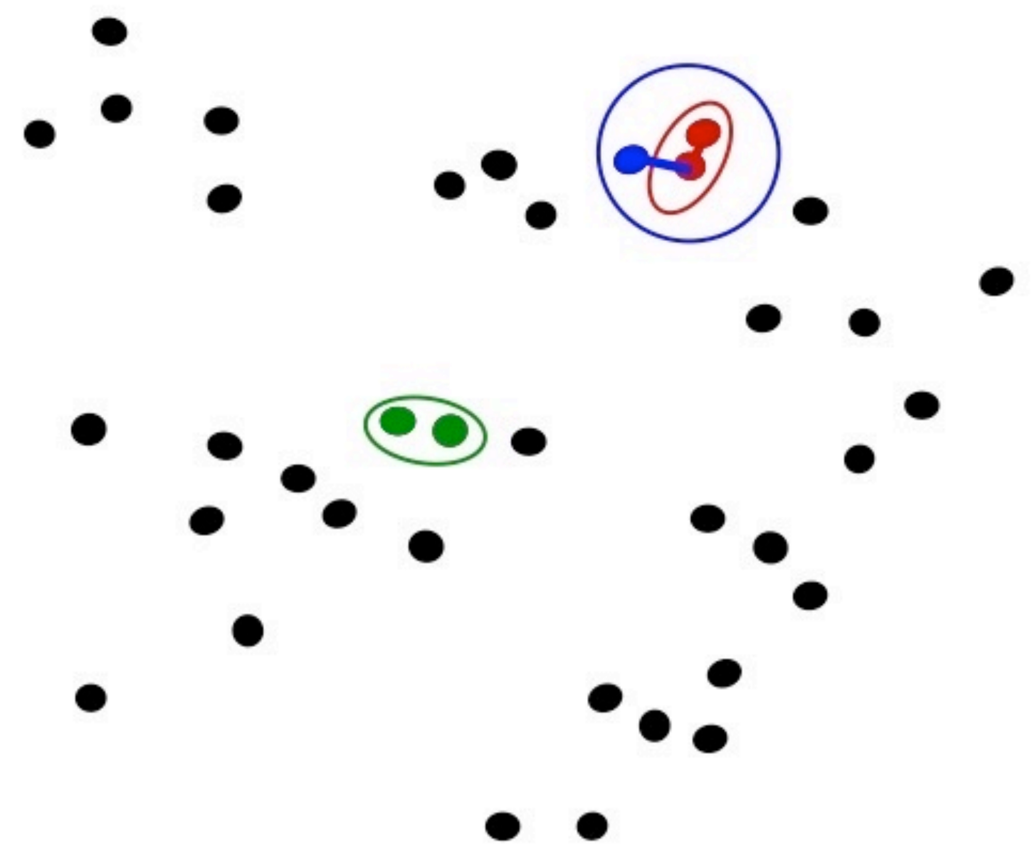
Interlude: you know other methods!

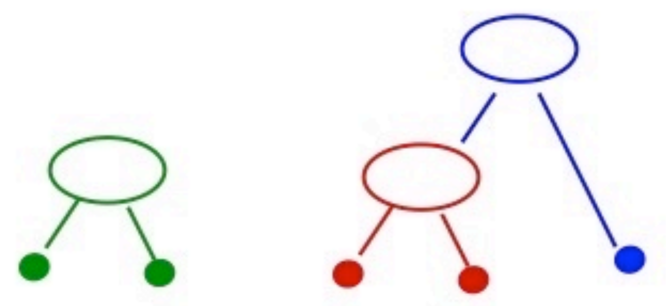
2-b) Create a graph that clusters (or classifies) data points
A possible solution: **Hierarchical clustering**

- Agglomerative clustering:
  - First merge very similar instances
  - Incrementally build larger clusters out of smaller clusters

- Algorithm:
  - Maintain a set of clusters
  - Initially, each instance in its own cluster
  - Repeat:
    - Pick the two closest clusters
    - Merge them into a new cluster
    - Stop when there's only one cluster left

- Produces not one clustering, but a family of clusterings represented by a dendrogram





Create a graph to represent the data

Interlude: you know other methods!

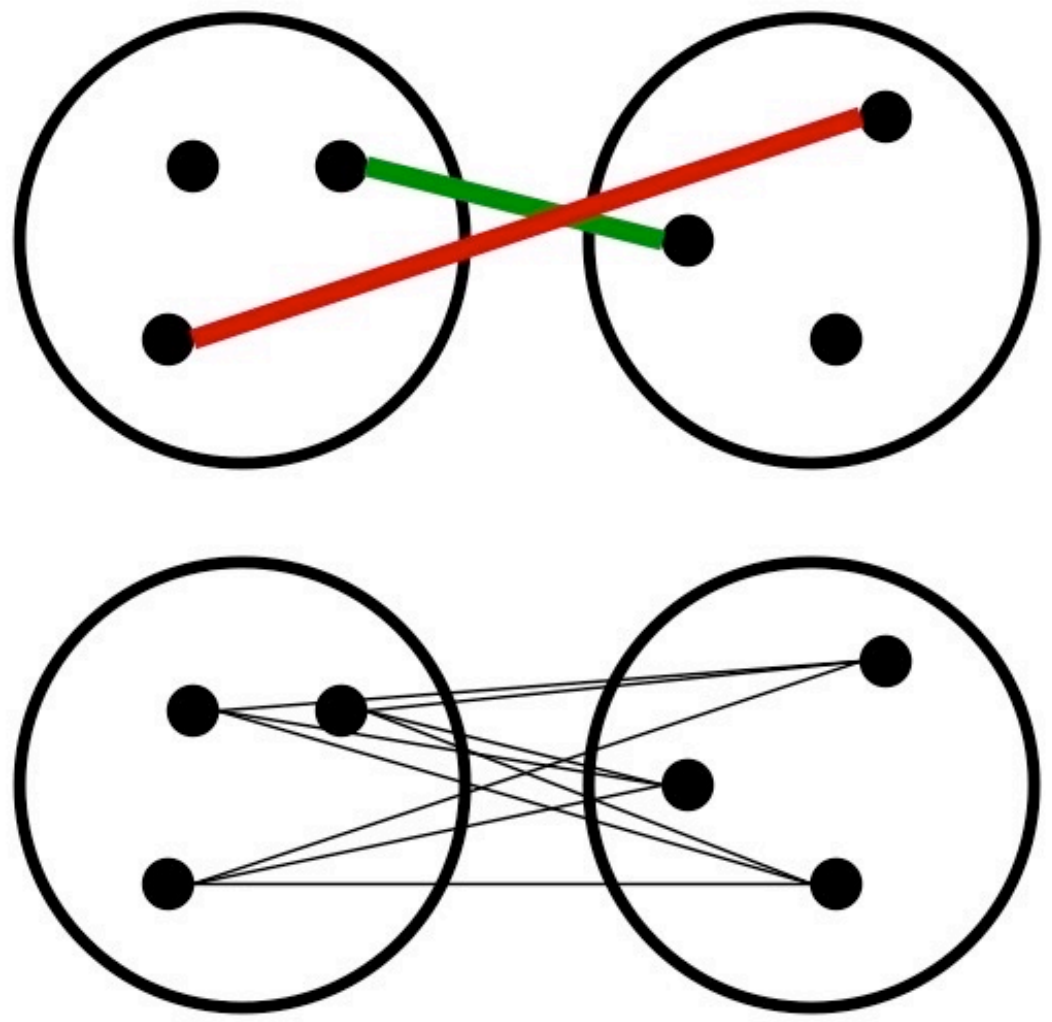
2-b) Create a graph that clusters (or classifies) data points
A possible solution: **Hierarchical clustering**

An issue involved in Agglomerative clustering

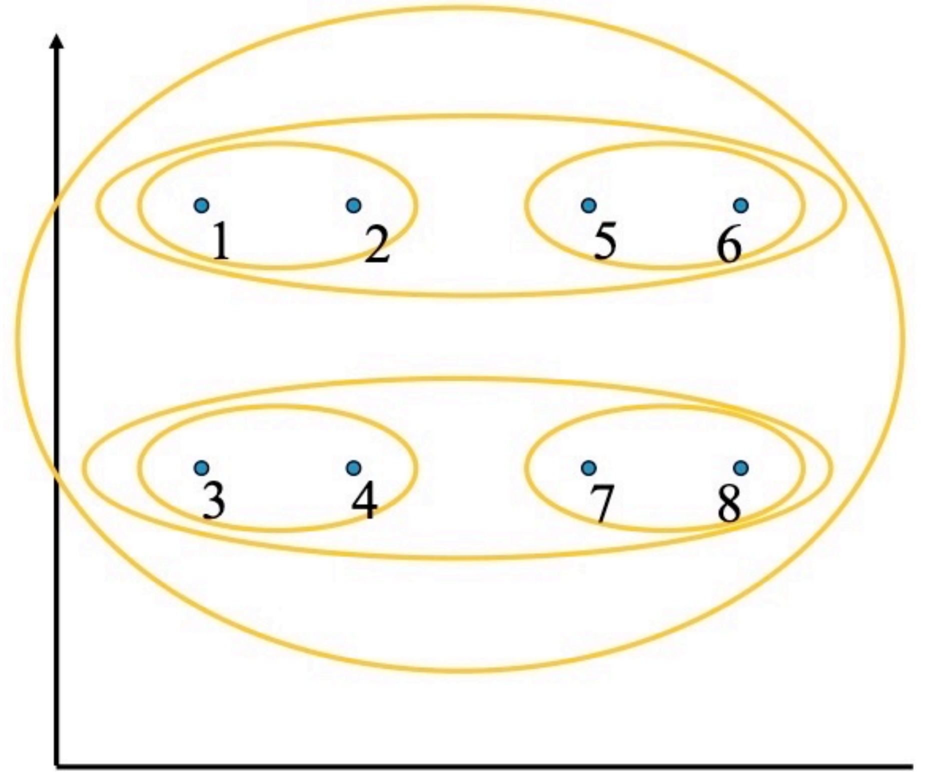
- How should we define "closest" for clusters with multiple elements?

- Many options:
  - Closest pair (single-link clustering)
  - Farthest pair (complete-link clustering)
  - Average of all pairs

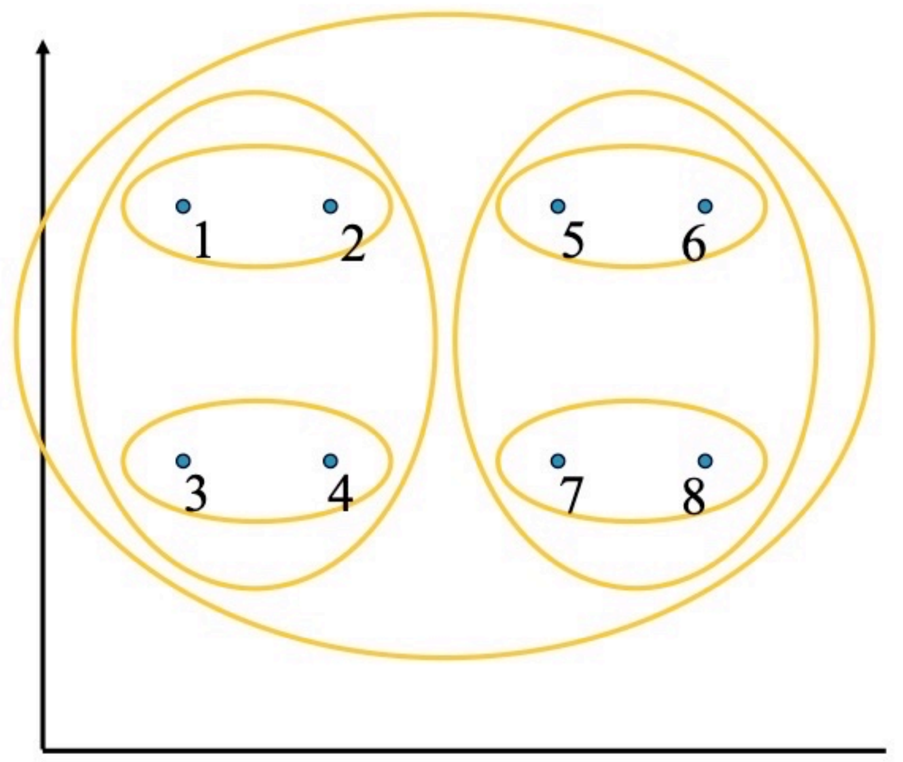
- Different choices create different clustering behaviors



Closest pair
(single-link clustering)



Farthest pair
(complete-link clustering)



[Pictures from Thorsten Joachims]

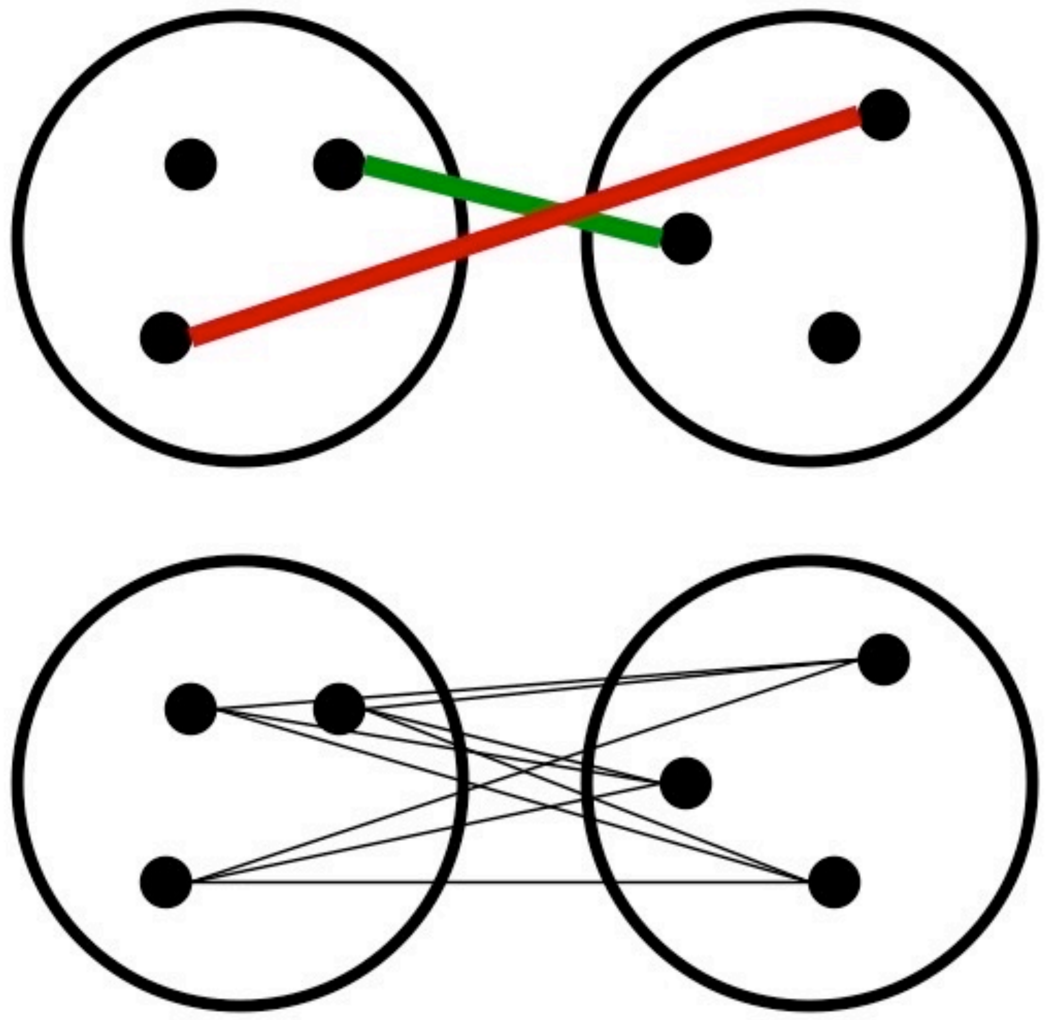
Create a graph to represent the data

Interlude: you know other methods!

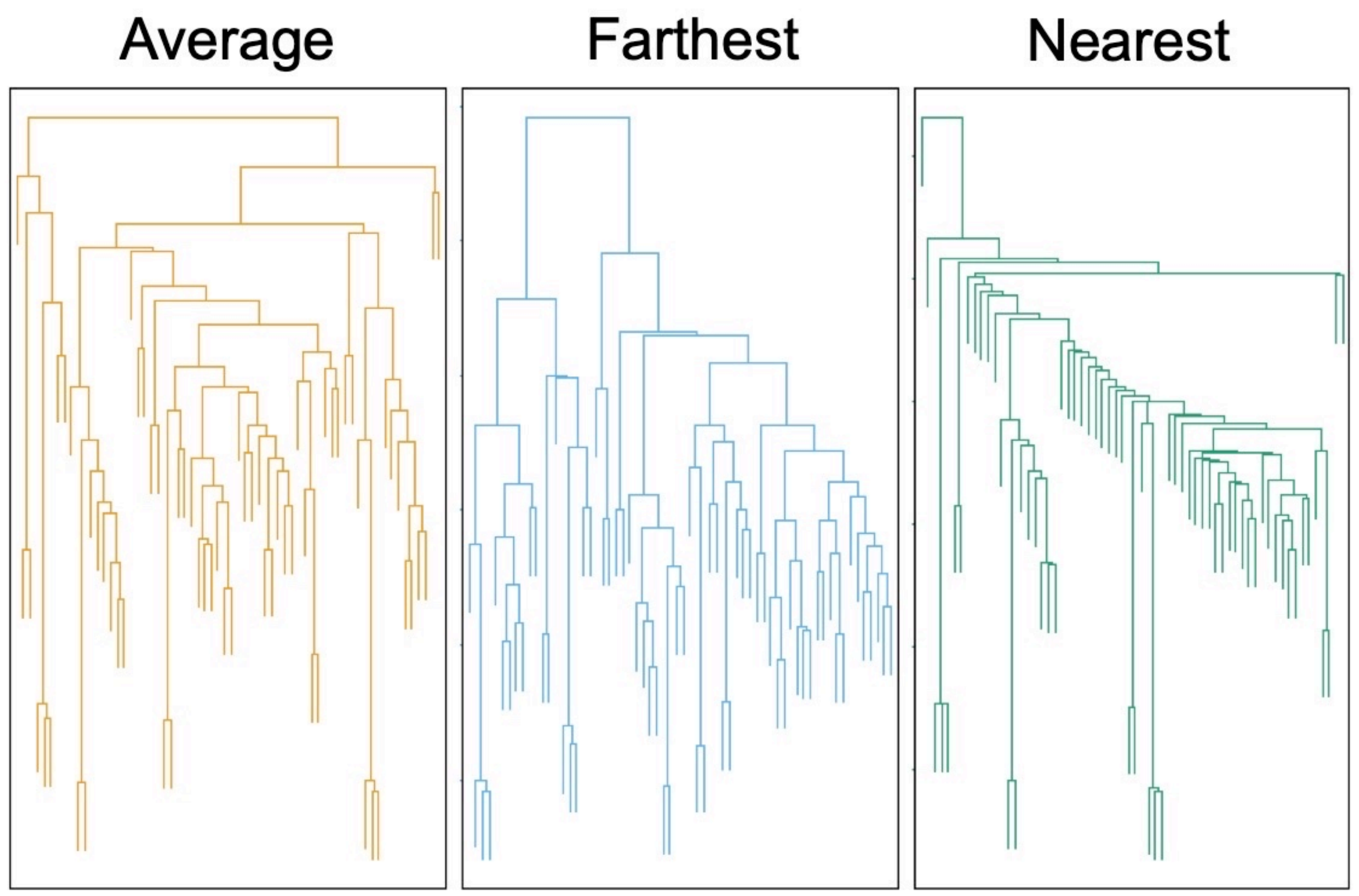
2-b) Create a graph that clusters (or classifies) data points
A possible solution: **Hierarchical clustering**

An issue involved in Agglomerative clustering

- How should we define "closest" for clusters with multiple elements?
- Many options:
  - Closest pair (single-link clustering)
  - Farthest pair (complete-link clustering)
  - Average of all pairs
- Different choices create different clustering behaviors



Clustering Behavior



Mouse tumor data from [Hastie *et al.*]

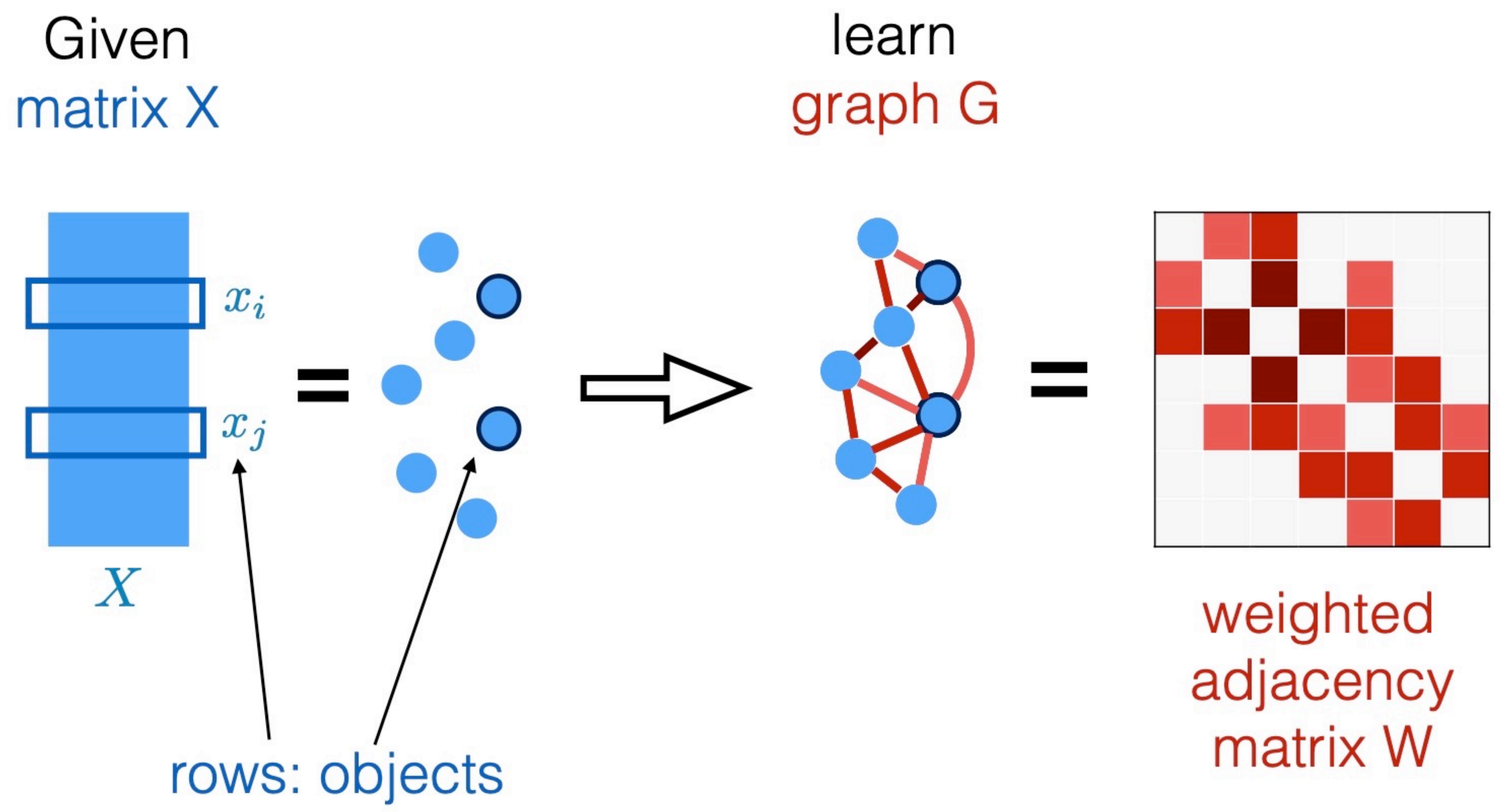
Create a graph to represent the data

Interlude: you know other methods!

- **2-c) Learn a graph that captures things from the data**

- **The general setting:**

from observations.... ...find a graph.... ...that models well the data



here: some model is useful
e.g.: Gaussian model

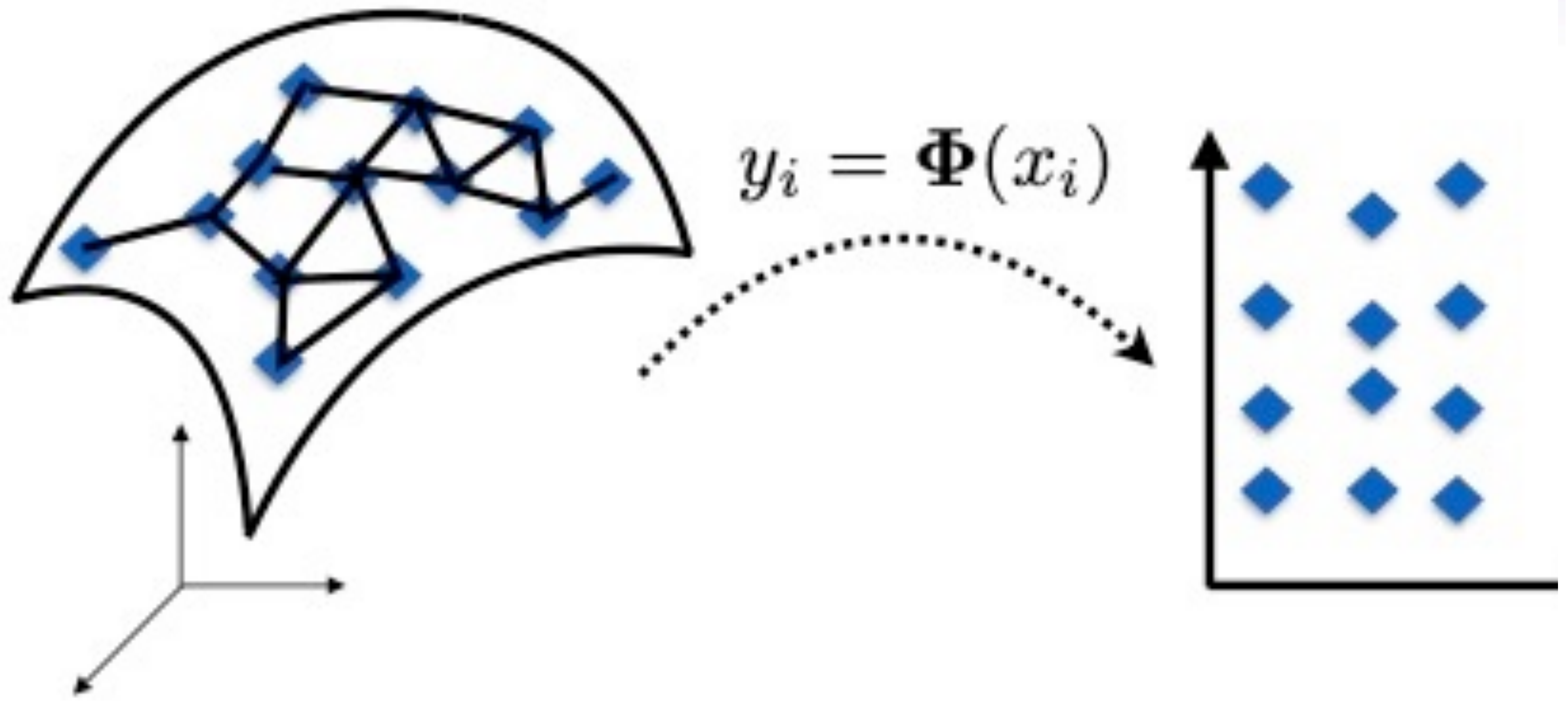
- Gaussian Graphical Models
- Bayesian Networks
- Methods with optimization inside!

Back to 2) Laplacian eigenmaps

Objective: embeddings of graphs from spectral features

- Objective of embedding: embed vertices in low dimensional space, so as to discover geometry

$$x_i \in \mathbb{R}^d \rightarrow y_i \in \mathbb{R}^k \text{ with } k < d$$



Laplacian eigenmaps

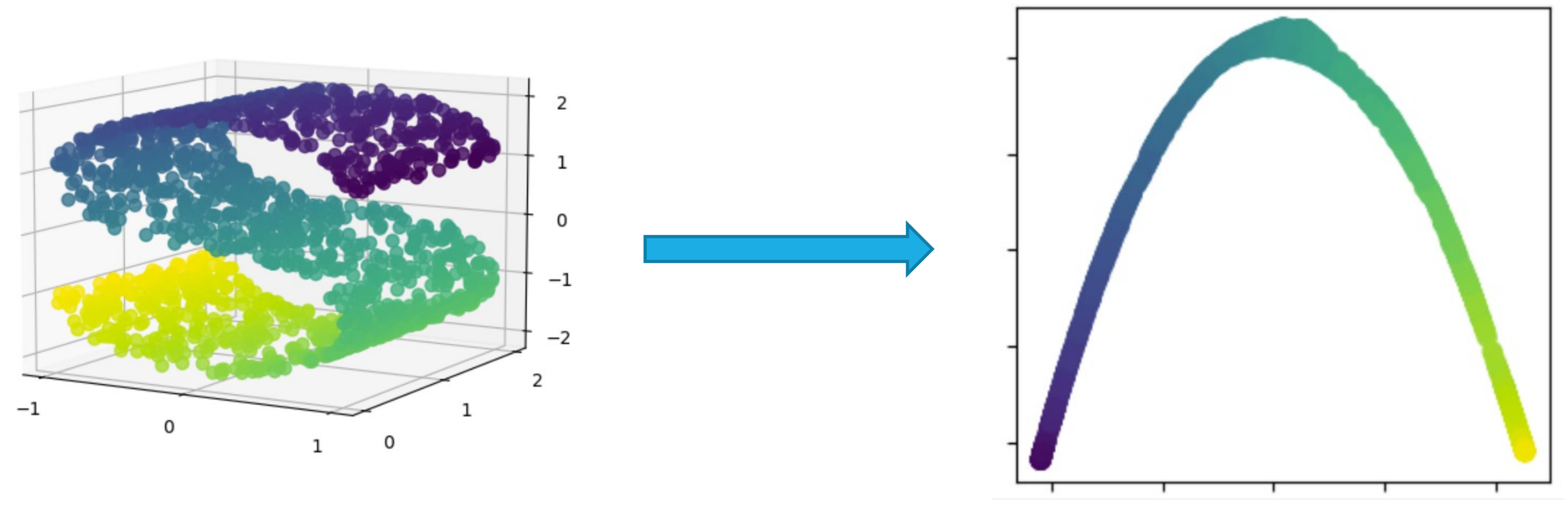
Formulation

W captures similarities among data points $x_i \in \mathbb{R}^L$

Suppose we embed in 1 dimension (P=1)

$$\arg\min_{y_1,\dots,y_N} \sum_{i\sim j} \mathbf{W}(i,j)(y_i - y_j)^2 \Rightarrow \arg\min_{y\in\mathbb{R}^N} y^T\mathbf{L}y$$

Add a constraint to avoid collapse y=0: $y^T\mathbf{D}y = 1$



Avoid trivial eigenvector: $y^T\mathbf{D1} = 0$

$$\Rightarrow \arg\min_{\substack{y\in\mathbb{R}^N \\ y^T\mathbf{D}y = 1 \\ y^T\mathbf{D1} = 0}} y^T\mathbf{L}y$$

Laplacian eigenmaps

Full problem

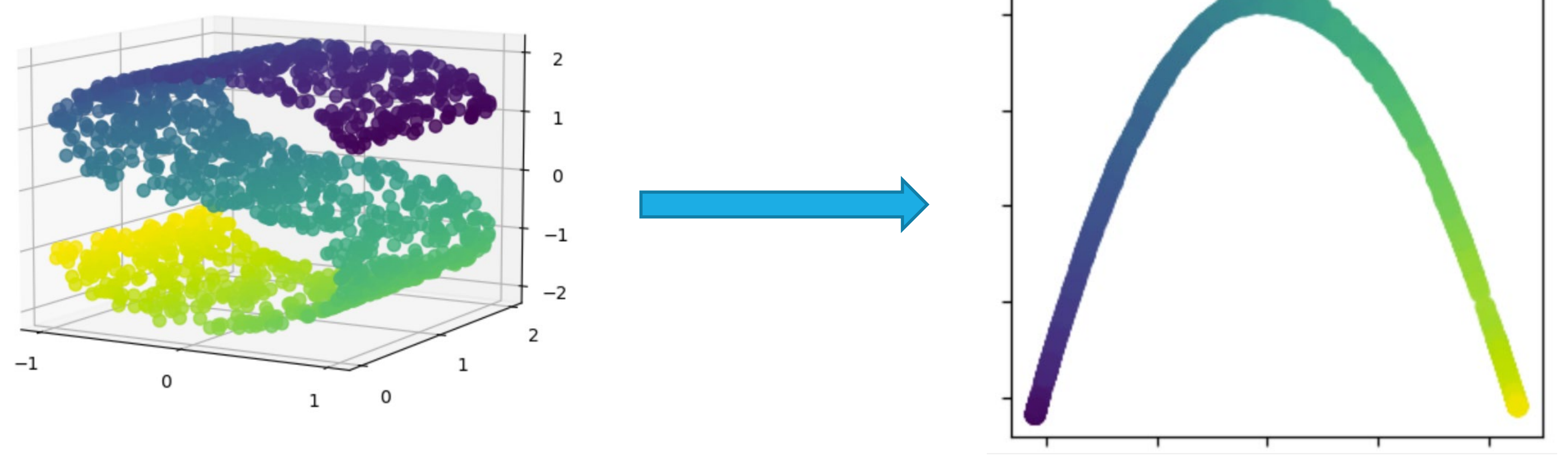
When we embed in P dimension $(P > 1)$

$$\arg\min_{y_1,\ldots,y_N} \sum_{i\sim j} \mathbf{W}(i,j)\|y_i - y_j\|_2^2$$

Algorithm: Laplacian Eigenmaps

Collect the coordinates of embedded points as lines of matrix Y

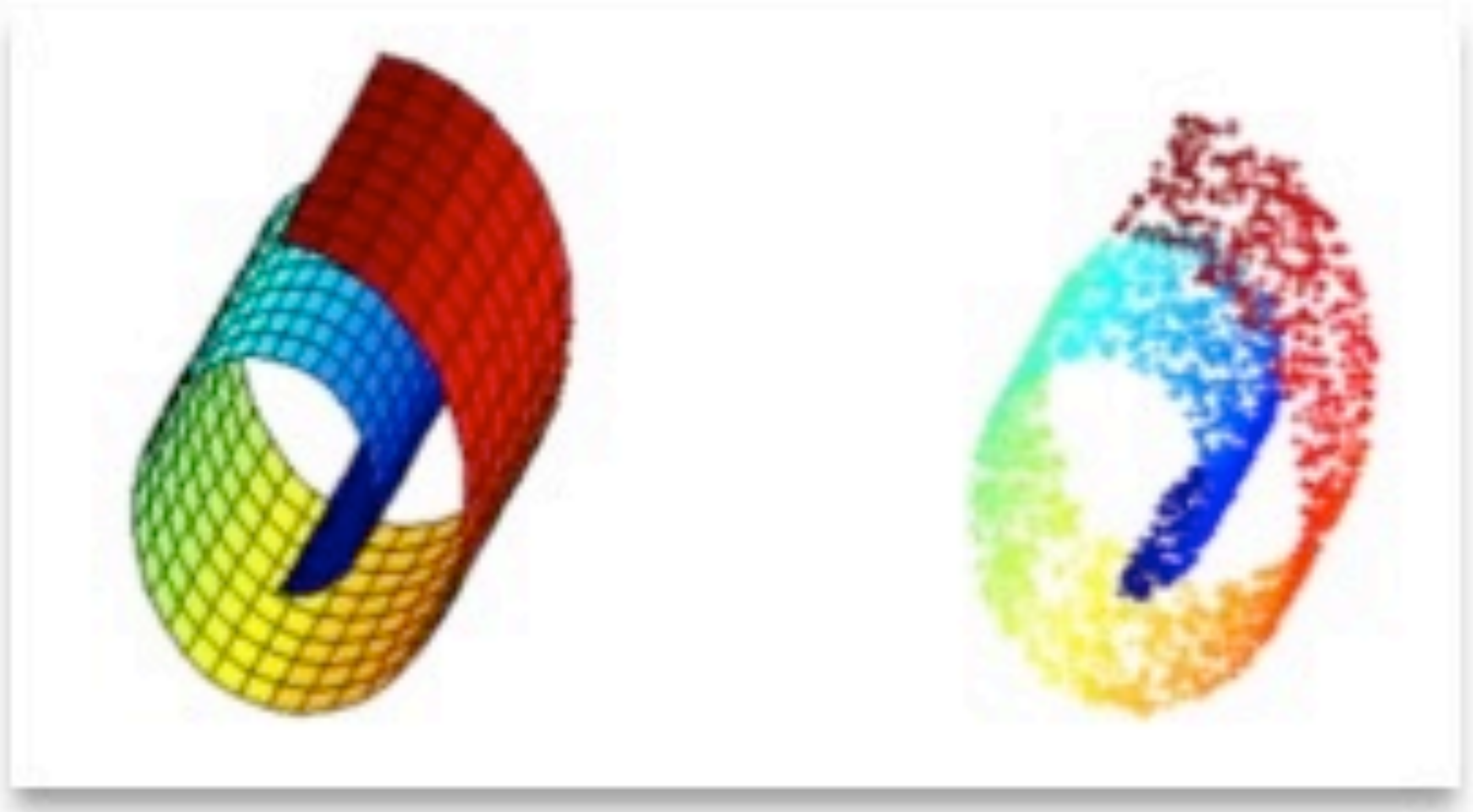
$$\arg\min_{\substack{Y \in \mathbb{R}^{N\times P} \\ Y^T\mathbf{D}Y = \mathbb{I}}} \operatorname{tr}(Y^T\mathbf{L}Y)$$

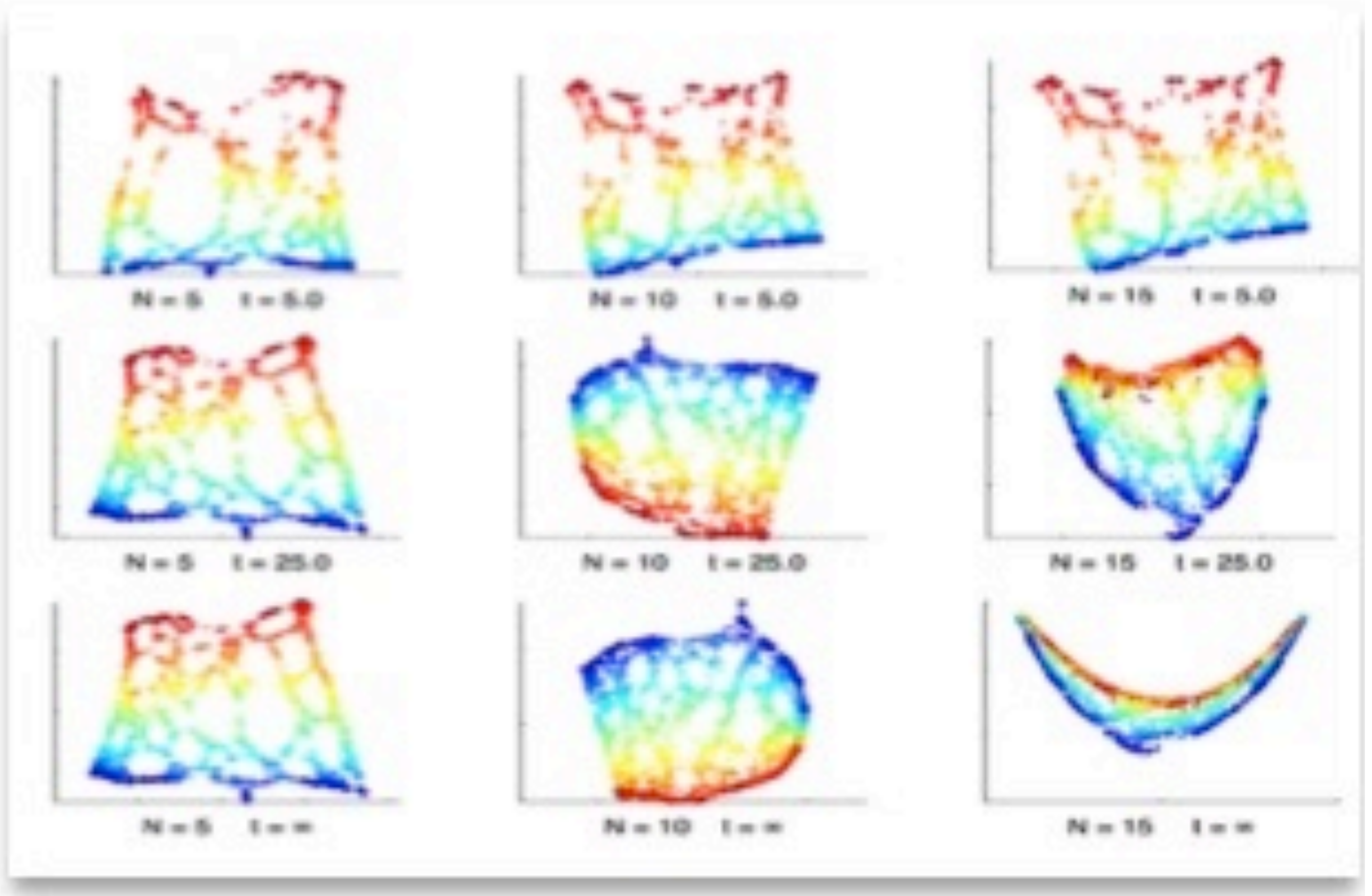


Laplacian Eigenmaps produces coordinate maps that are smooth functions/signals over the original graph.

Laplacian eigenmaps

- Some examples

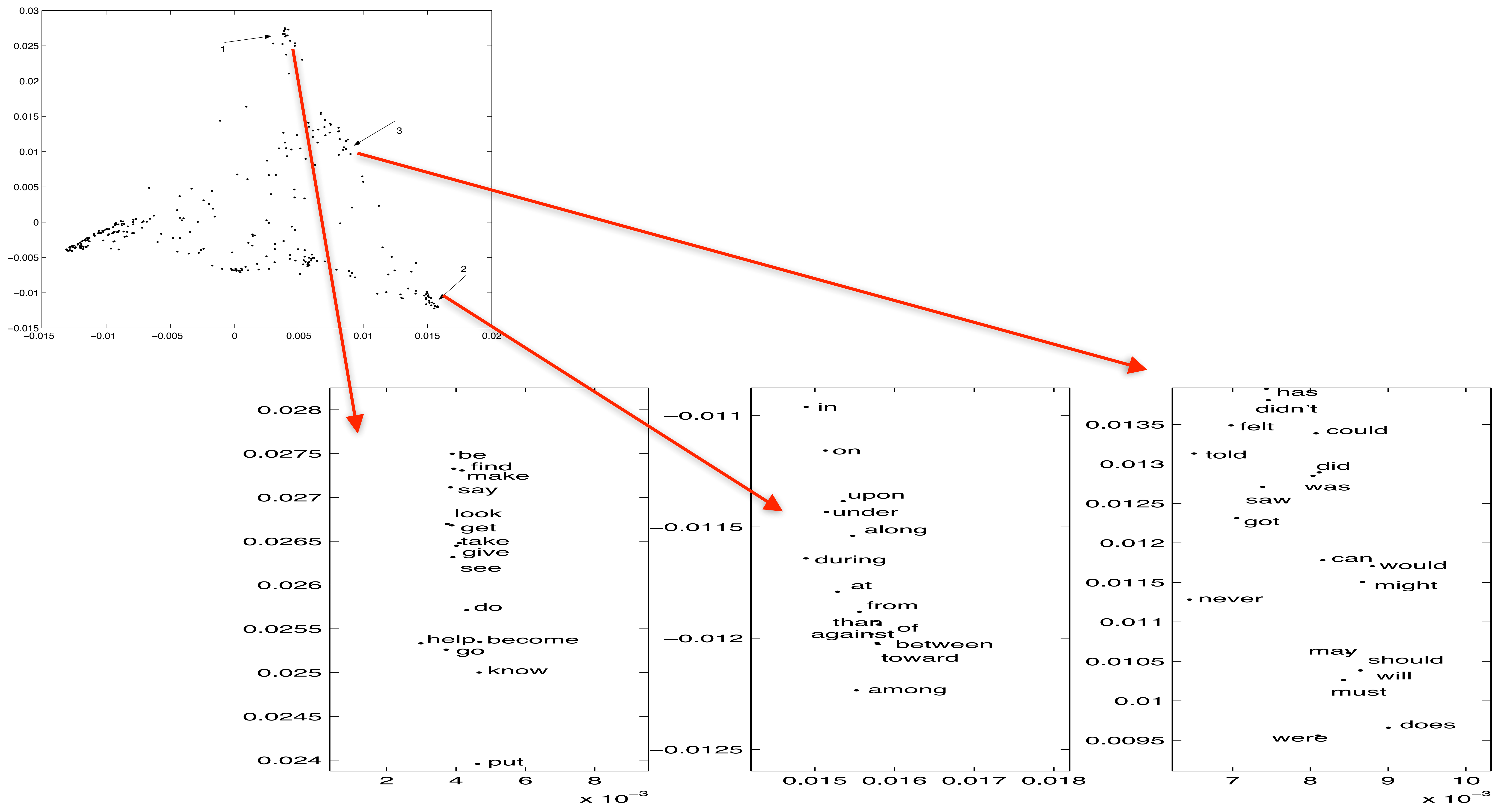




[Belkin, Niyogi, 2003]

Laplacian eigenmaps

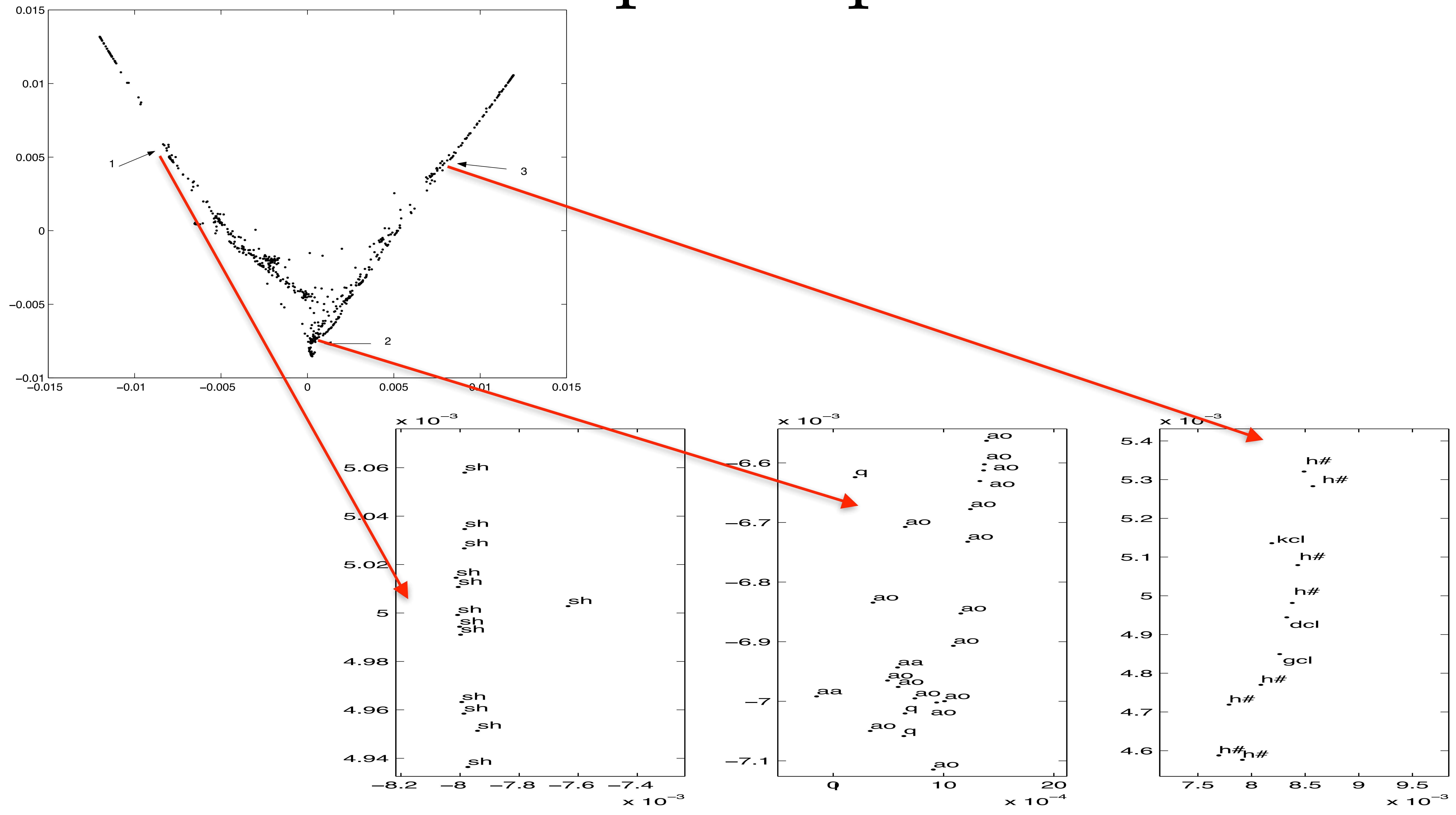
Examples: text



M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Comput*, vol. 15, no. 6, pp. 1373–1396, 2003.

Laplacian eigenmaps

Examples: speech

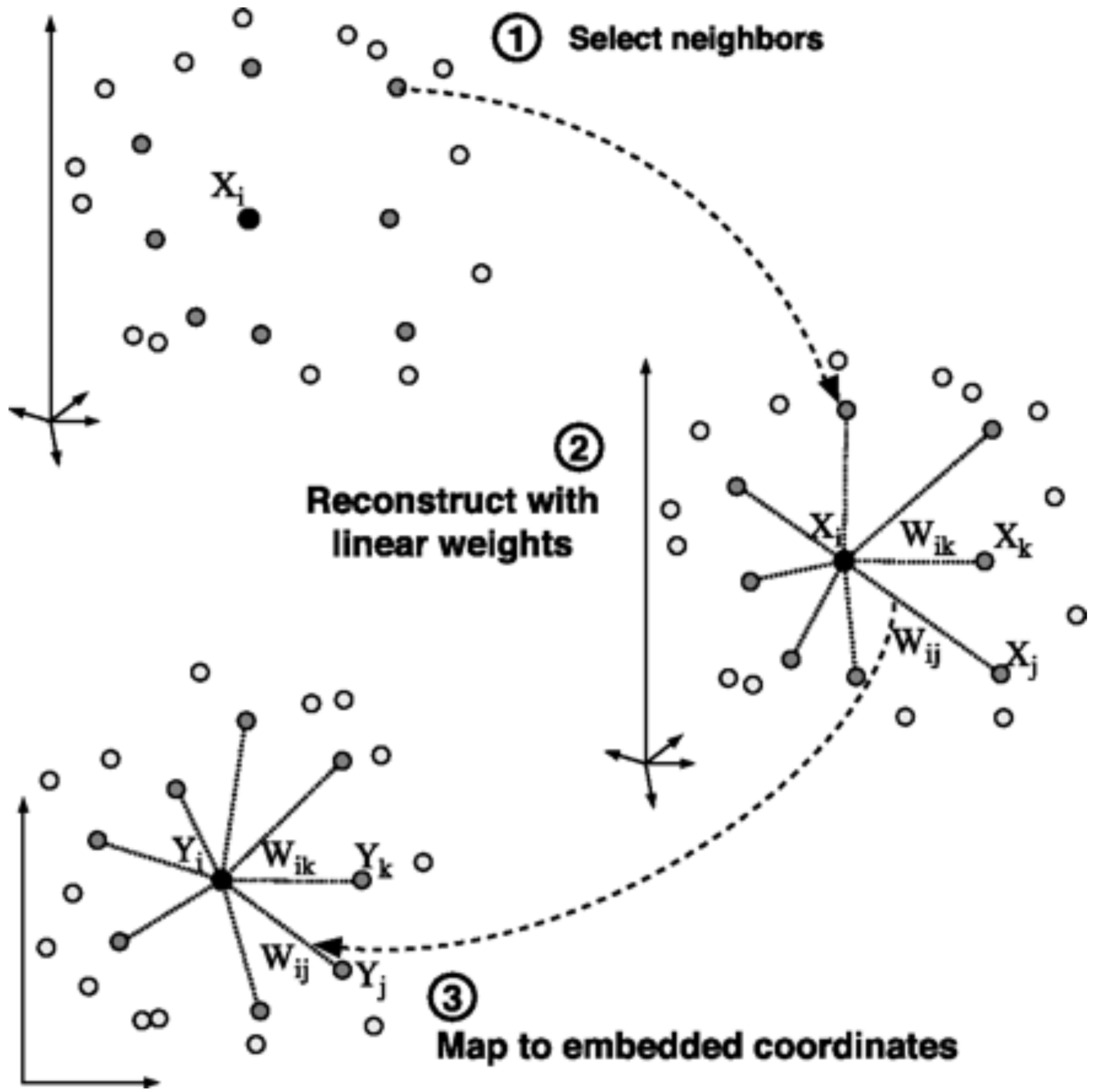


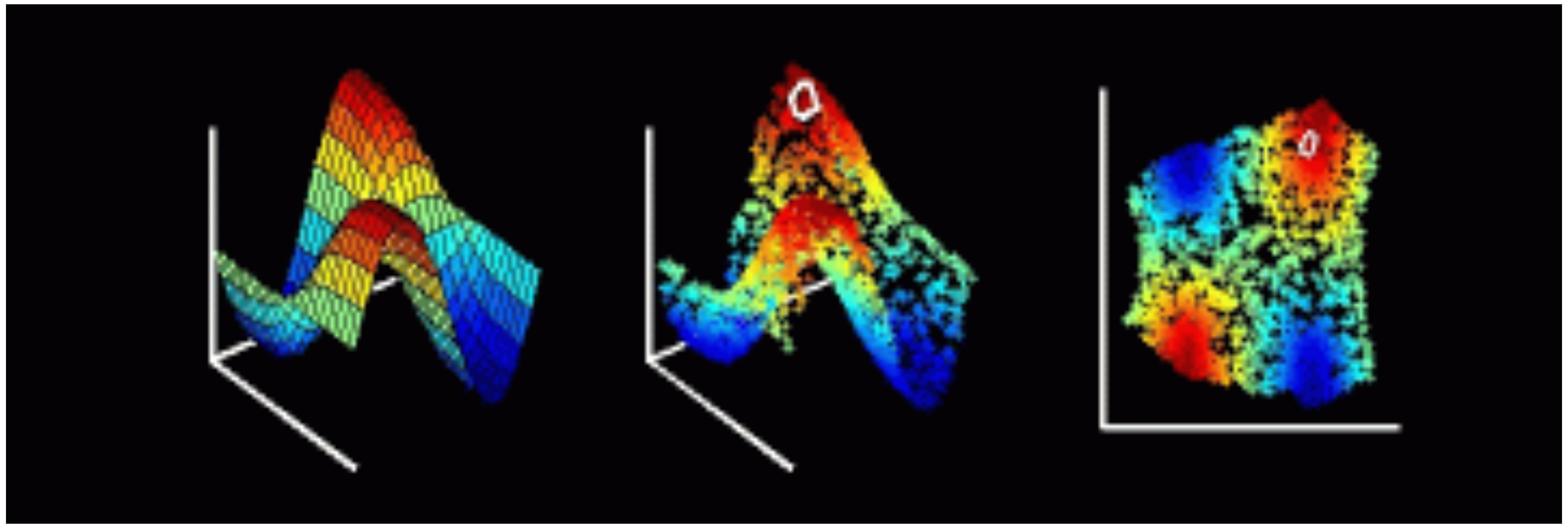
3) Locally Linear Embeddings

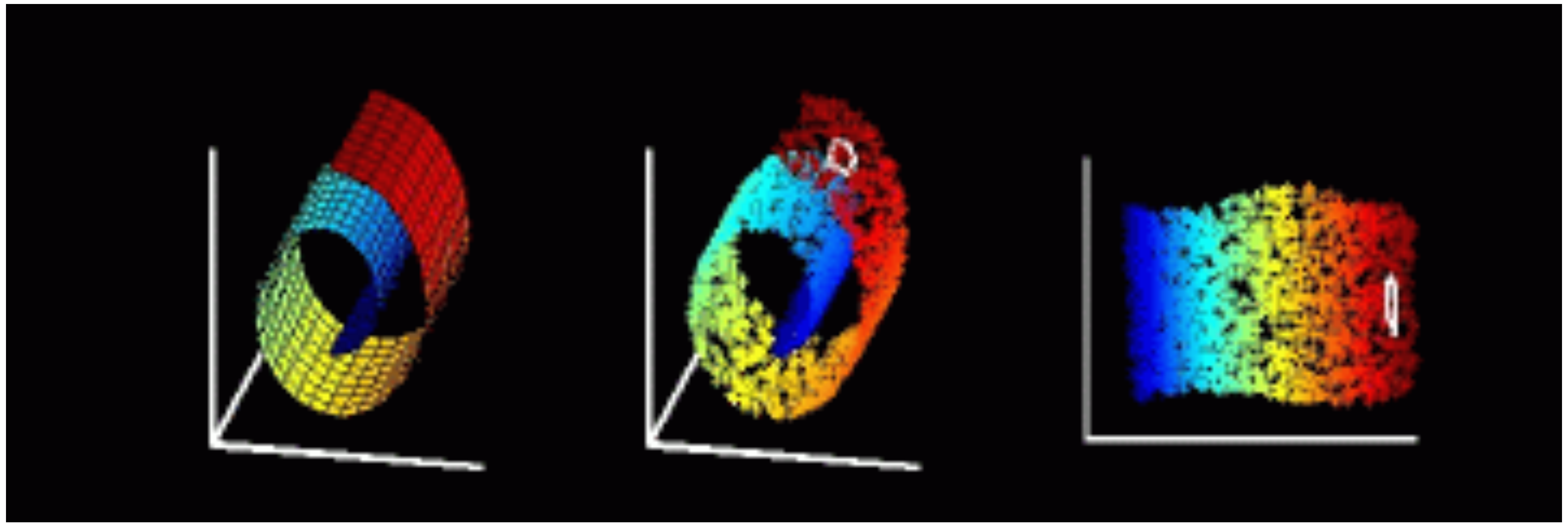
- Introduced 2000
- A node features can be represented as a linear combination of its neighbors'
  - $Y_i = \sum_j A_{ij} Y_j$
- Objective function:
  - $y^* = \min \sum_i \| Y_i - \sum_j A_{ij} Y_j \|^2$

Sam T. Roweis & Lawrence K. Saul (2000) “Nonlinear Dimensionality Reduction by Locally Linear Embedding”.

3) Locally Linear Embeddings







Sam T. Roweis & Lawrence K. Saul (2000) “Nonlinear Dimensionality Reduction by Locally Linear Embedding”.

4) Random-Walk based Representations

DeepWalk, Node2Vec,…

=> in truth, instances of encoder / decoder framework

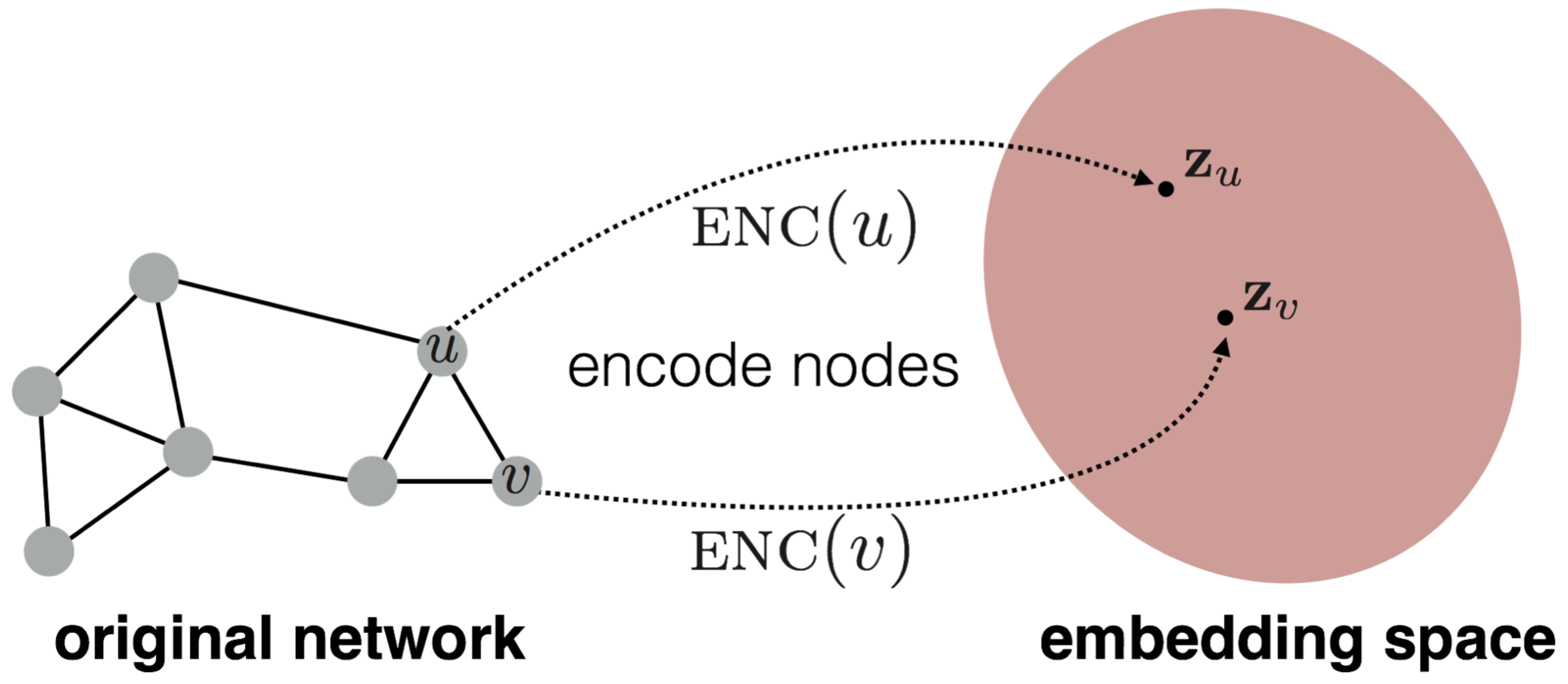


Figure 3.1: Illustration of the node embedding problem. Our goal is to learn an encoder (ENC), which maps nodes to a low-dimensional embedding space. These embeddings are optimized so that distances in the embedding space reflect the relative positions of the nodes in the original graph.

- W. Hamilton: Articles in 2017
- Book: *Graph Representation Learning*, 2020

4) Node Embedding by Encoder/Decoder

$$\text{ENC} : \mathcal{V} \rightarrow \mathbb{R}^d, \qquad \text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+.$$

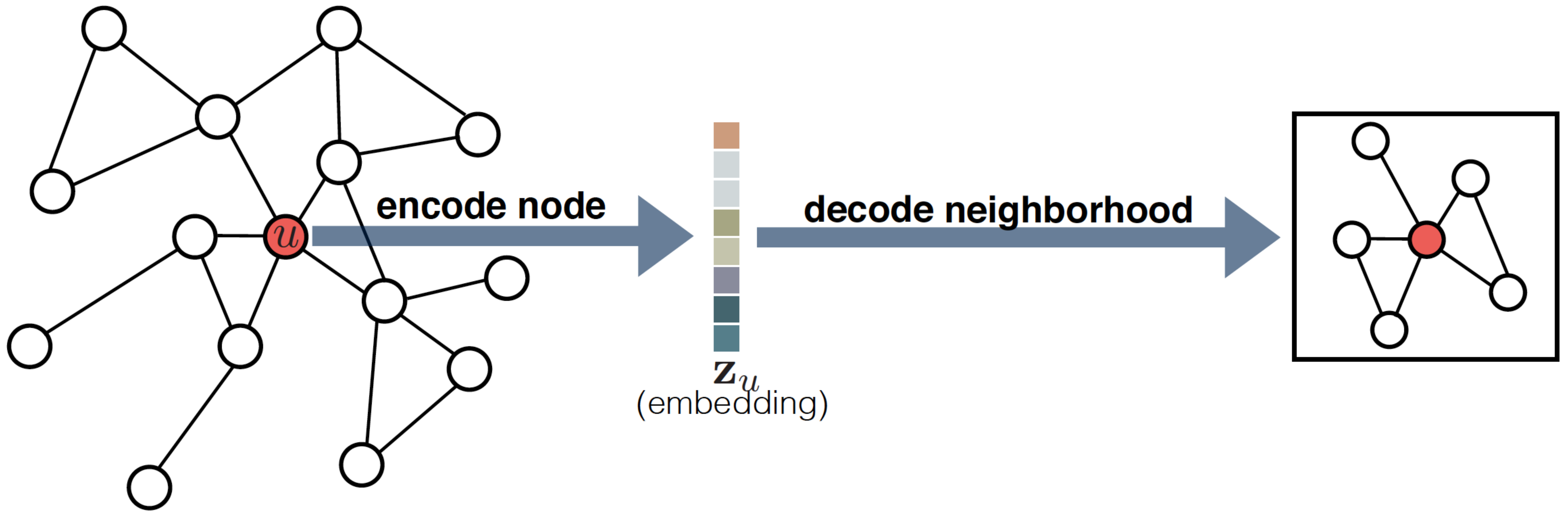
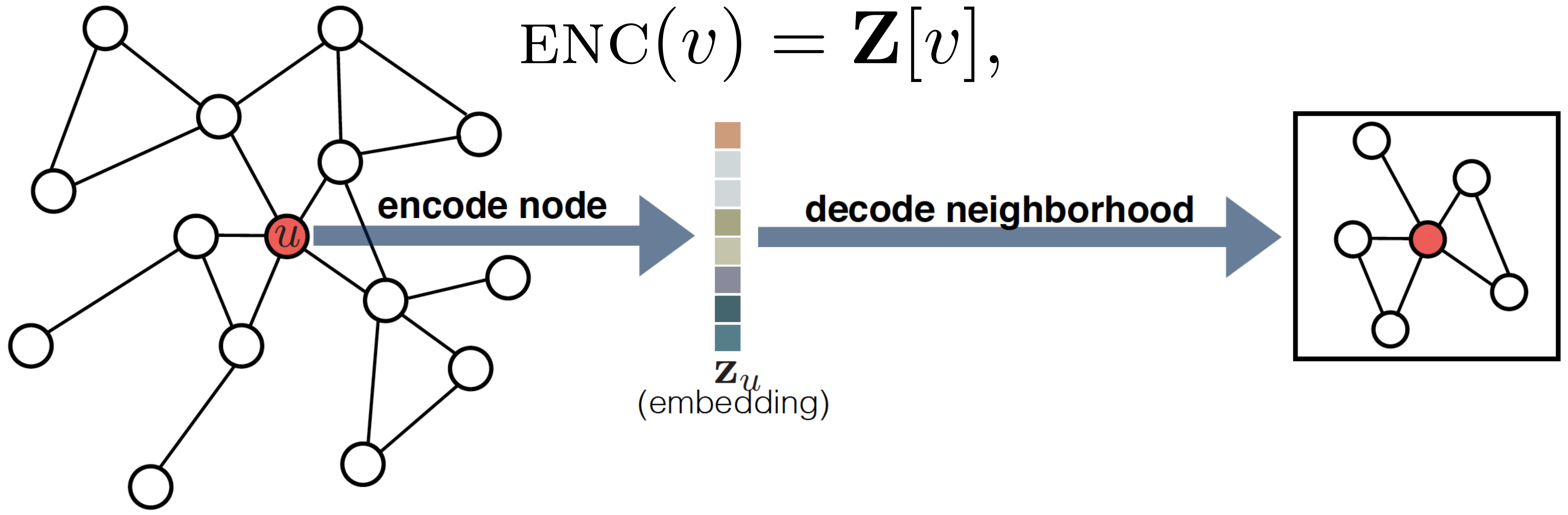


Figure 3.2: Overview of the encoder-decoder approach. The encoder maps the node u to a low-dimensional embedding $\mathbf{z}_u$. The decoder then uses $\mathbf{z}_u$ to reconstruct u's local neighborhood information.

- W. Hamilton, *2017*

4) Node Embedding by Encoder/Decoder



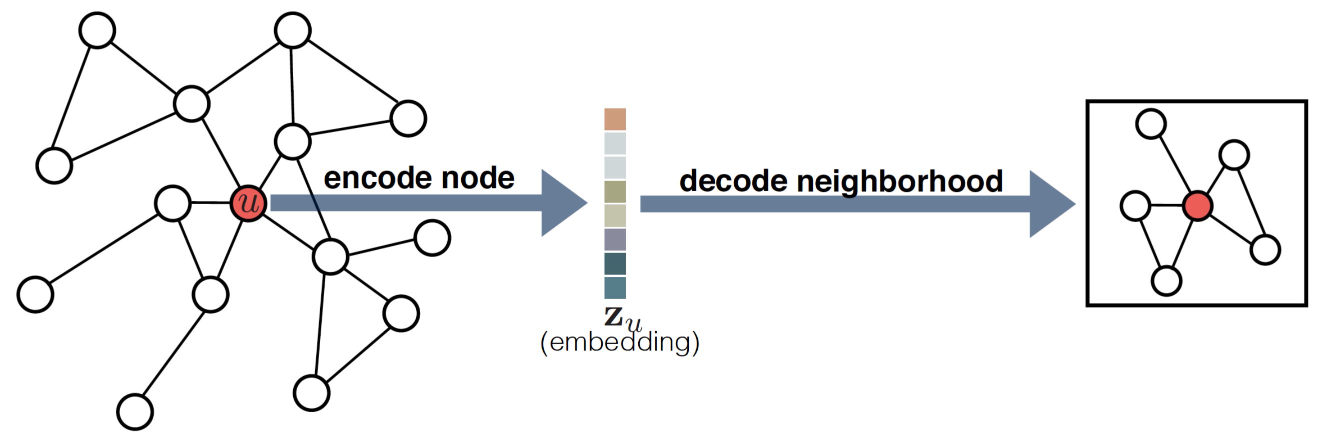
$$\mathrm{DEC}(\mathrm{ENC}(u), \mathrm{ENC}(v)) = \mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) \approx \mathbf{S}[u, v].$$

To train the representation, use a global loss for Auto-Encoding

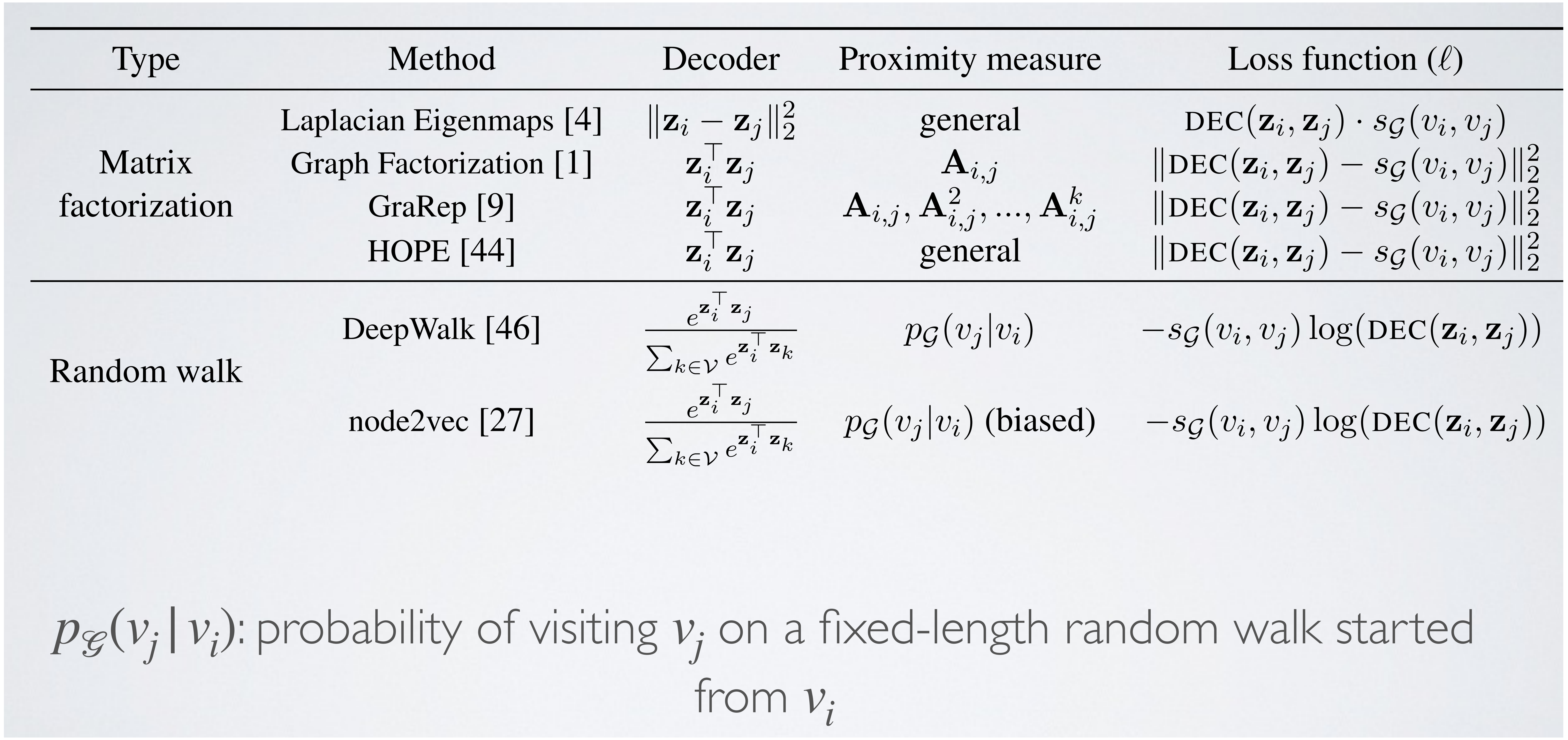
$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \ell\left(\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v), \mathbf{S}[u, v]\right)$$

- W. Hamilton, *2017*

4) Node Embedding by Encoder/Decoder



One finds known methods:

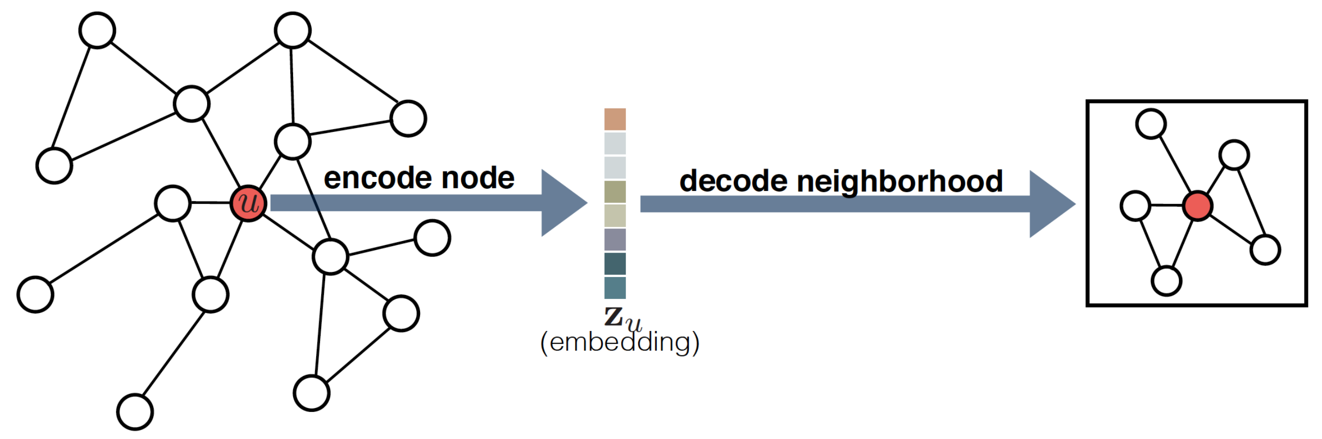


Type	Method	Decoder	Proximity measure	Loss function (ℓ)
Matrix factorization	Laplacian Eigenmaps [4]	$\\|\mathbf{z}_i - \mathbf{z}_j\\|_2^2$	general	$\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j)$
	Graph Factorization [1]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}$	$\\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\\|_2^2$
	GraRep [9]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}, \mathbf{A}_{i,j}^2, ..., \mathbf{A}_{i,j}^k$	$\\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\\|_2^2$
	HOPE [44]	$\mathbf{z}_i^\top \mathbf{z}_j$	general	$\\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\\|_2^2$
Random walk	DeepWalk [46]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \| v_i)$	$-s_{\mathcal{G}}(v_i, v_j) \log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j))$
	node2vec [27]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \| v_i)$ (biased)	$-s_{\mathcal{G}}(v_i, v_j) \log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j))$

$p_{\mathcal{G}}(v_j \,|\, v_i)$: probability of visiting v_j on a fixed-length random walk started from v_i

- W. Hamilton, *2017*

4) Node Embedding by Encoder/Decoder



One finds known methods:

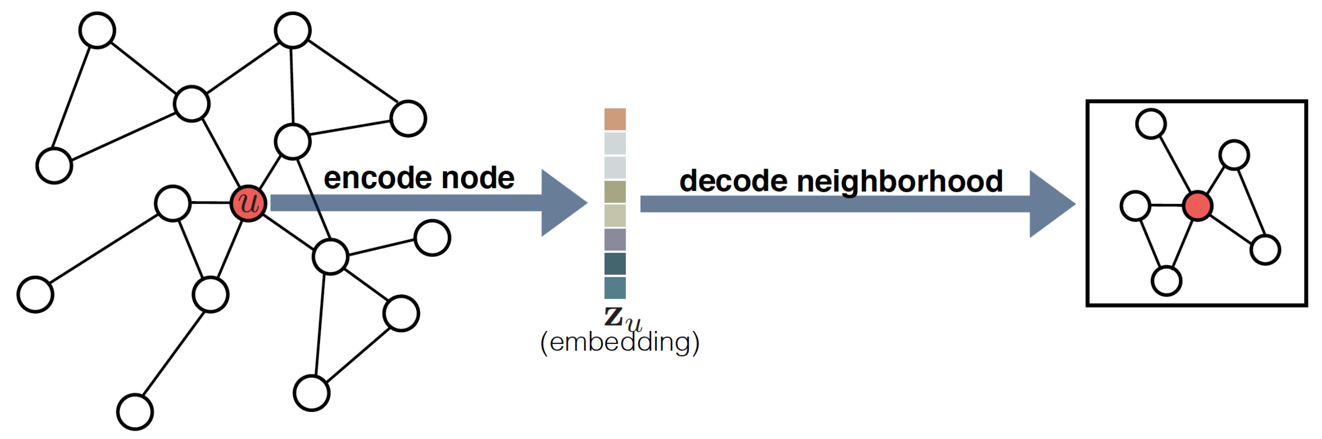
Two examples

1. Distributed large scale natural graph factorisation ($S = A$):

$$Z^* = \arg\min_{Z \in \mathbb{R}^{d \times n}} \|Z^T Z - S\|_F^2 + \frac{\lambda}{2}\|Z\|_F^2$$
$$= \sum_{(i,j)} \left(z_i^T z_j - S_{ij}\right)^2 + \frac{\lambda}{2} \sum_i \|z_i\|_2^2$$

solved with SGD in p with vertex partitioning for large graphs

4) Node Embedding by Encoder/Decoder



One finds known methods:

Two examples

2. GraRep: models k-hops relationships ($S = D^{-1}A$) $\quad p_k(x_i|x_j) = S_{ij}^k$

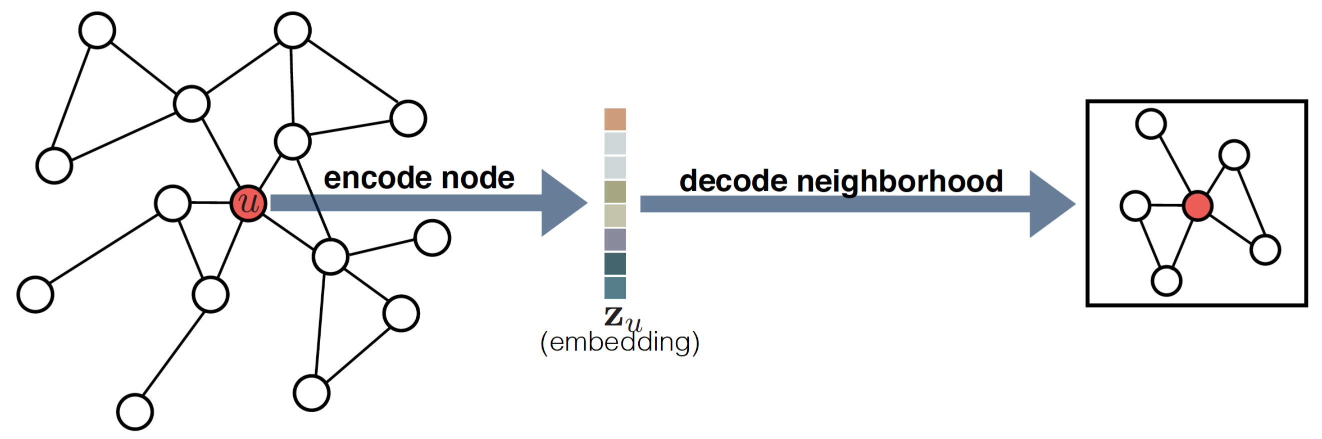
Inner product decoder: $\sigma(w_j^T c_i) \approx p_k(x_i|x_j)$ target and context latent vectors

k-hop Cross entropy loss: $L_k = \sum_{j \in V} L_k(w_j)$

$$L_k(w_j) = \sum_{i \in V} p_k(x_i|x_j) \log \sigma(w_j^T c_i) + \lambda \mathbb{E}_{c' \sim p_k(V)} \{\log \sigma(-w_j^T c')\}$$

<u>noise contrastive sampling:</u> choose c' from a noise distribution (here: at random independently of target w) and maximise probability that it is **not** a context of w

4) Node Embedding by Encoder/Decoder



One finds known methods:

Two examples

2. GraRep:

Choice of negative sampling distribution allows a factorization-based solution for the product $W_{(k)}^T C_{(k)} = Y_{(k)}$

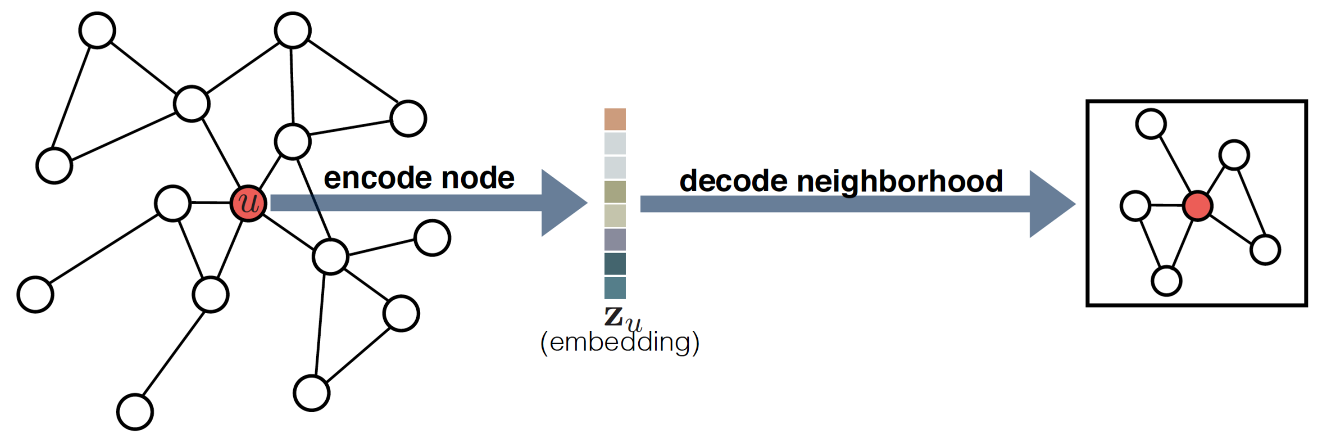
$$Y_{(k)ij} = \log\left(\frac{A_{ij}^k}{\sum_{\ell \in V} A_{\ell,j}^k}\right)$$

Solve for $W_{(k)}$ by SVD

and concatenate k=1,…,K

$$Y_{(k)} \approx U_{(k)}^{(d)} \Sigma_{(k)}^{(d)} \left(V_{(k)}^{(d)}\right)^T$$

4) Node Embedding by Encoder/Decoder

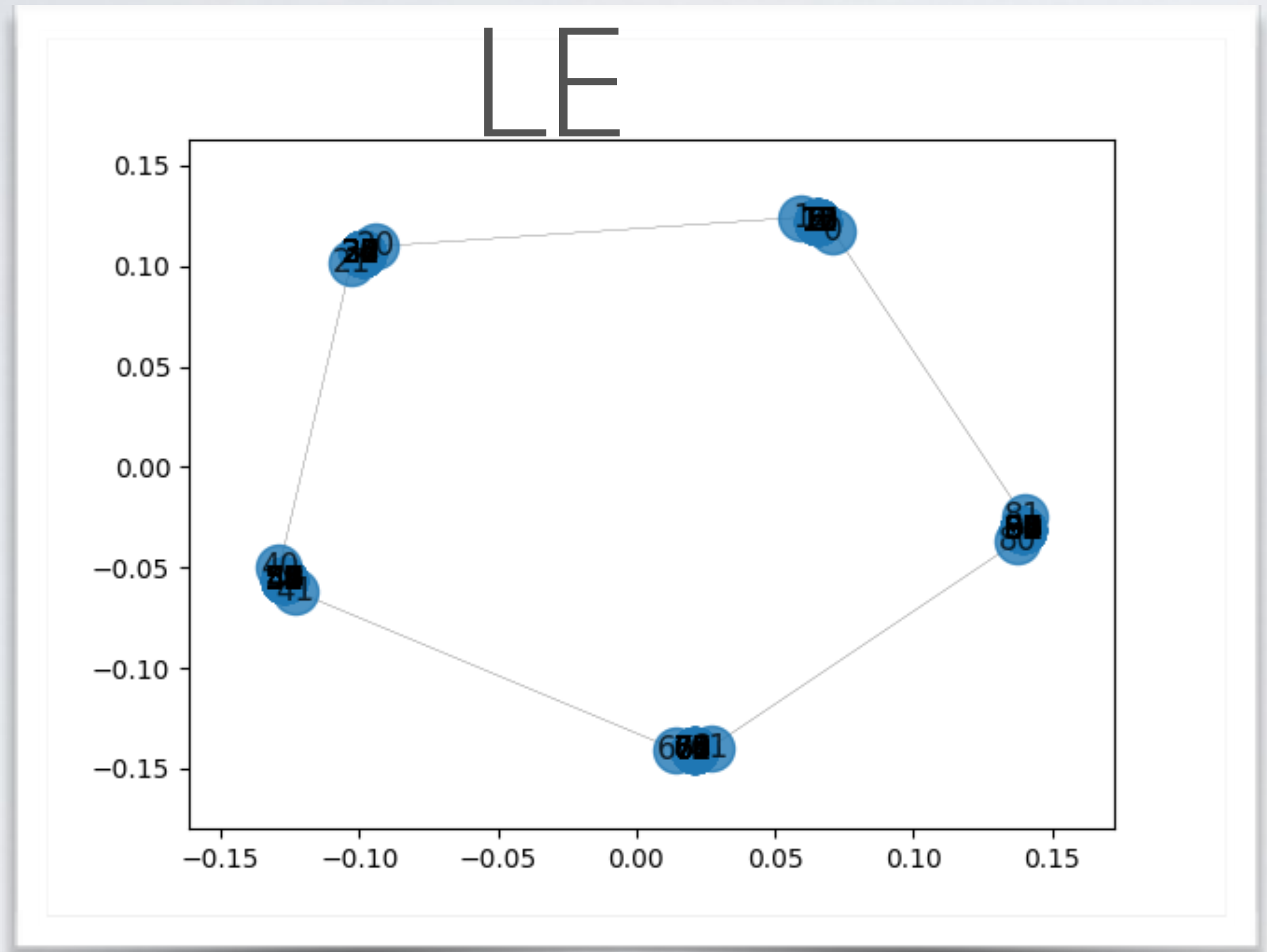


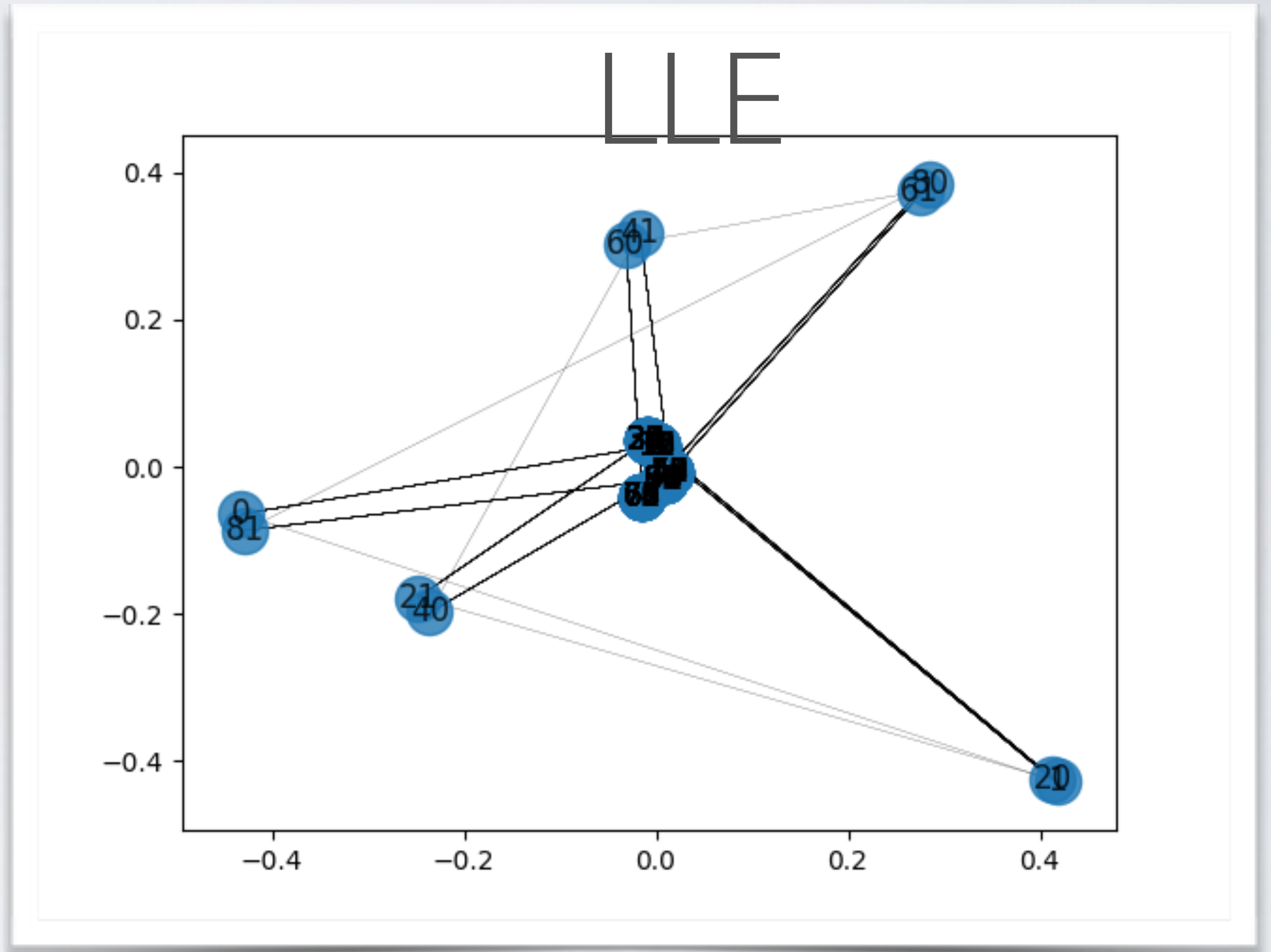
Limitations

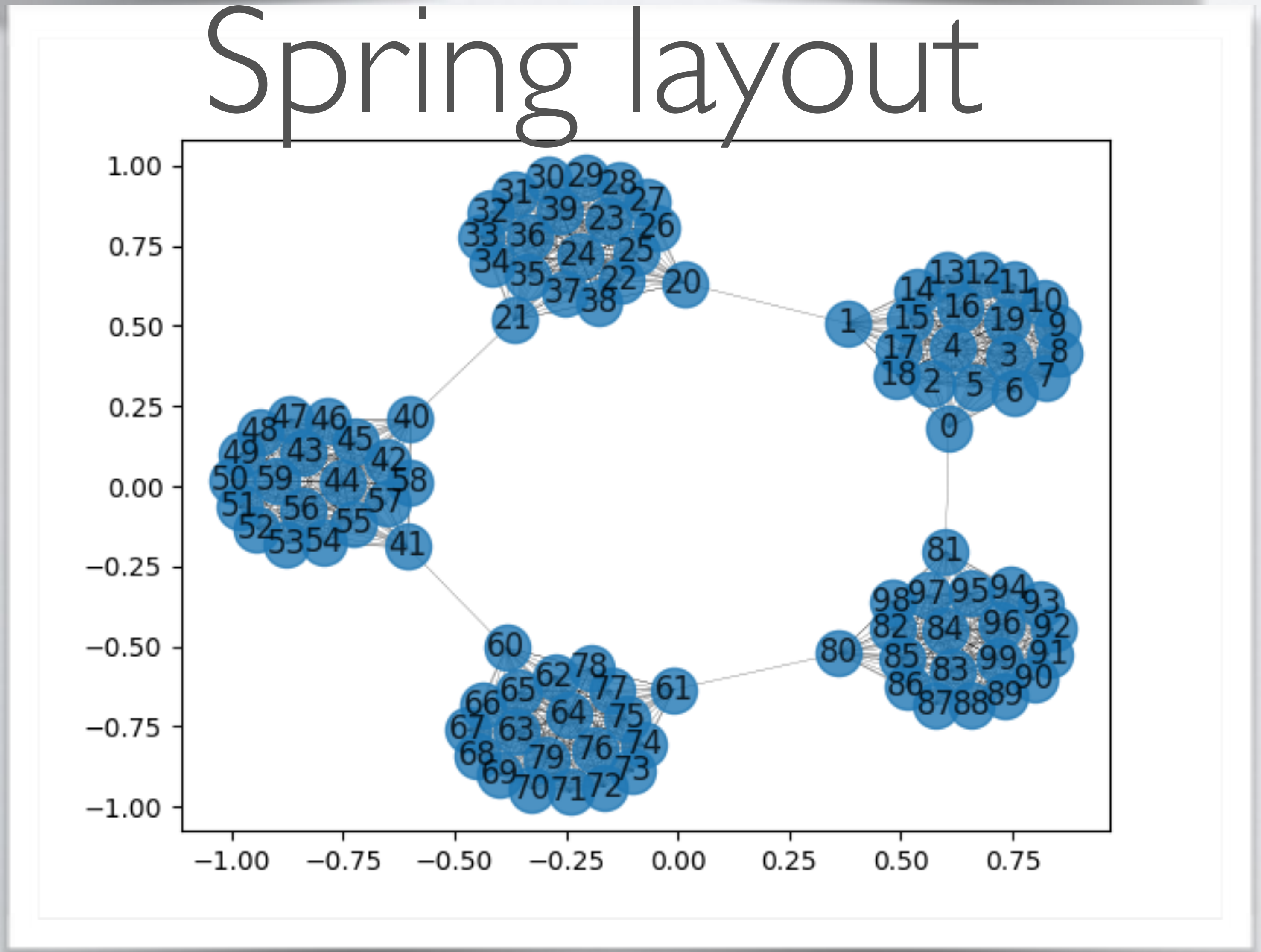
These techniques are transductive: you learn embeddings of observed nodes but you don't obtain a way to directly compute embeddings to unseen nodes. They don't easily leverage node features. No parameter sharing among nodes.

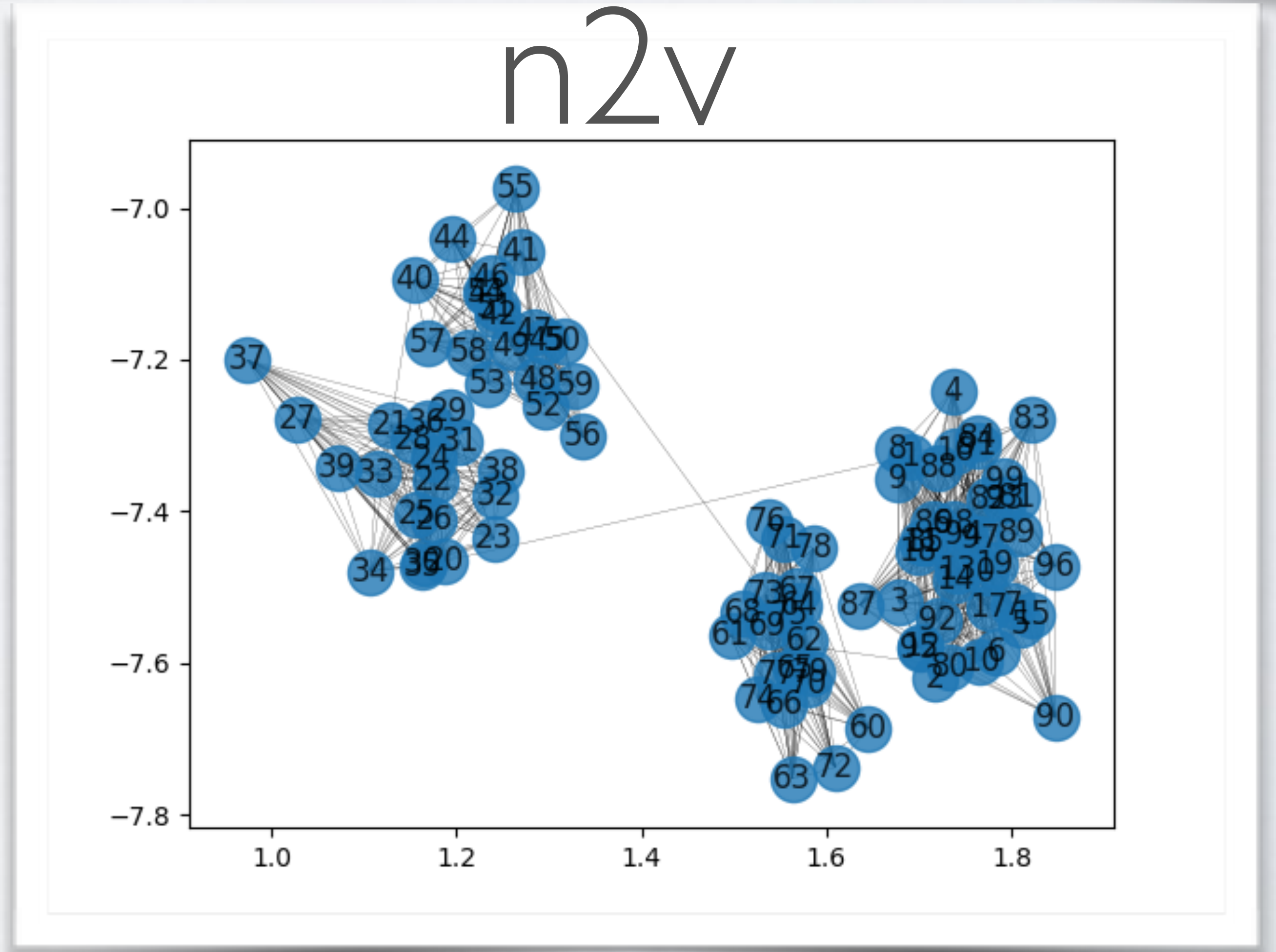
The lack of an encoder - a direct way to map a single (attributed) node to its latent code - is a weakness

- **Exemple on a Clique Ring: 5 cliques of size 20 connected by an edge as if on a ring**



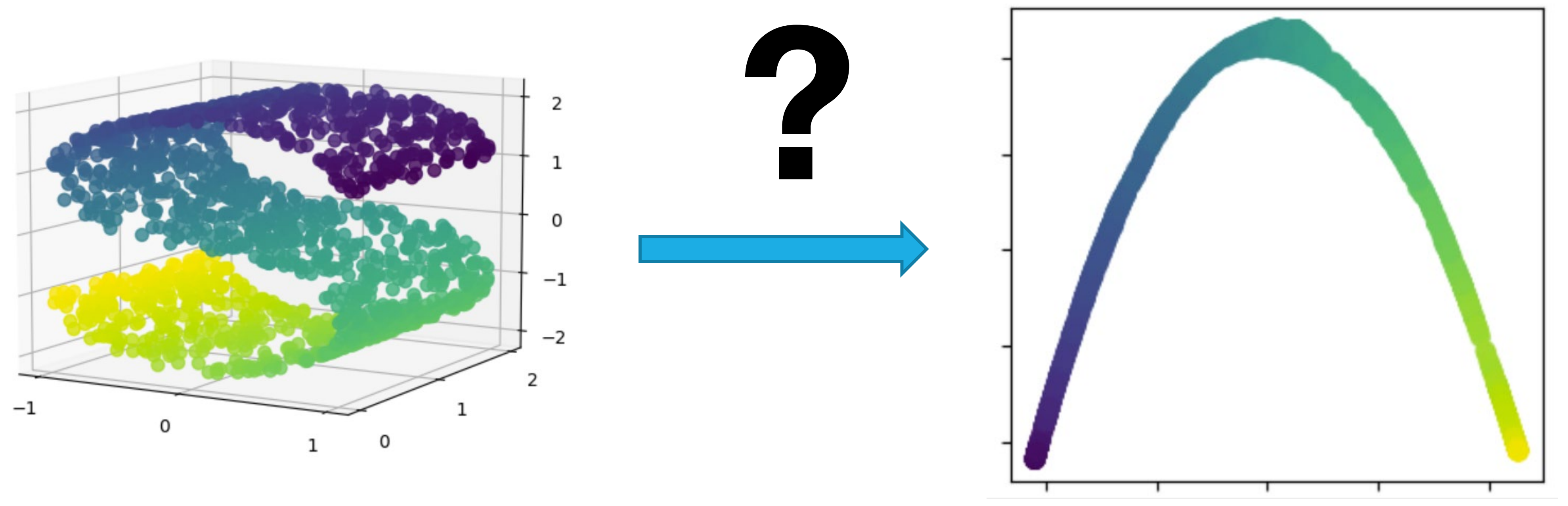


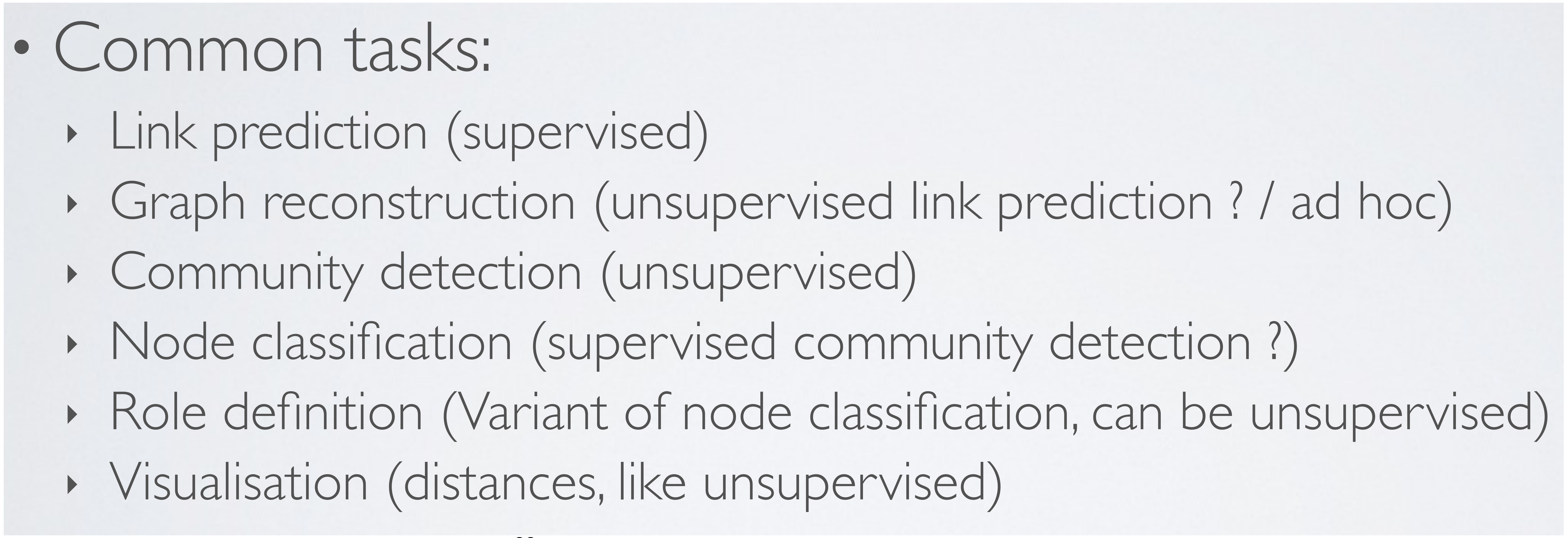




Embeddings of graphs in low dimension

Objective: find new coordinates **=> What for ?**





- Common tasks:
  - Link prediction (supervised)
  - Graph reconstruction (unsupervised link prediction ? / ad hoc)
  - Community detection (unsupervised)
  - Node classification (supervised community detection ?)
  - Role definition (Variant of node classification, can be unsupervised)
  - Visualisation (distances, like unsupervised)

Conclusion

Of Graph Embeddings and (Shallow) Representation Learning

- Efficient methods for Vizualization
    - (see also t-SNE, UMAP)
- Good to see / display structures in the graphs (and possibly explore /use them)
- OK for some representation learning (Lapl. maps, LLE, ENC/DEC)
- Less OK: not inductive; could use Deep ReprLearn. => **see Graph Neural Networks**