
LEARNING AND PROCESSING OVER NETWORKS

Michaël DEFFERRARD

Rodrigo PENA



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Motivation

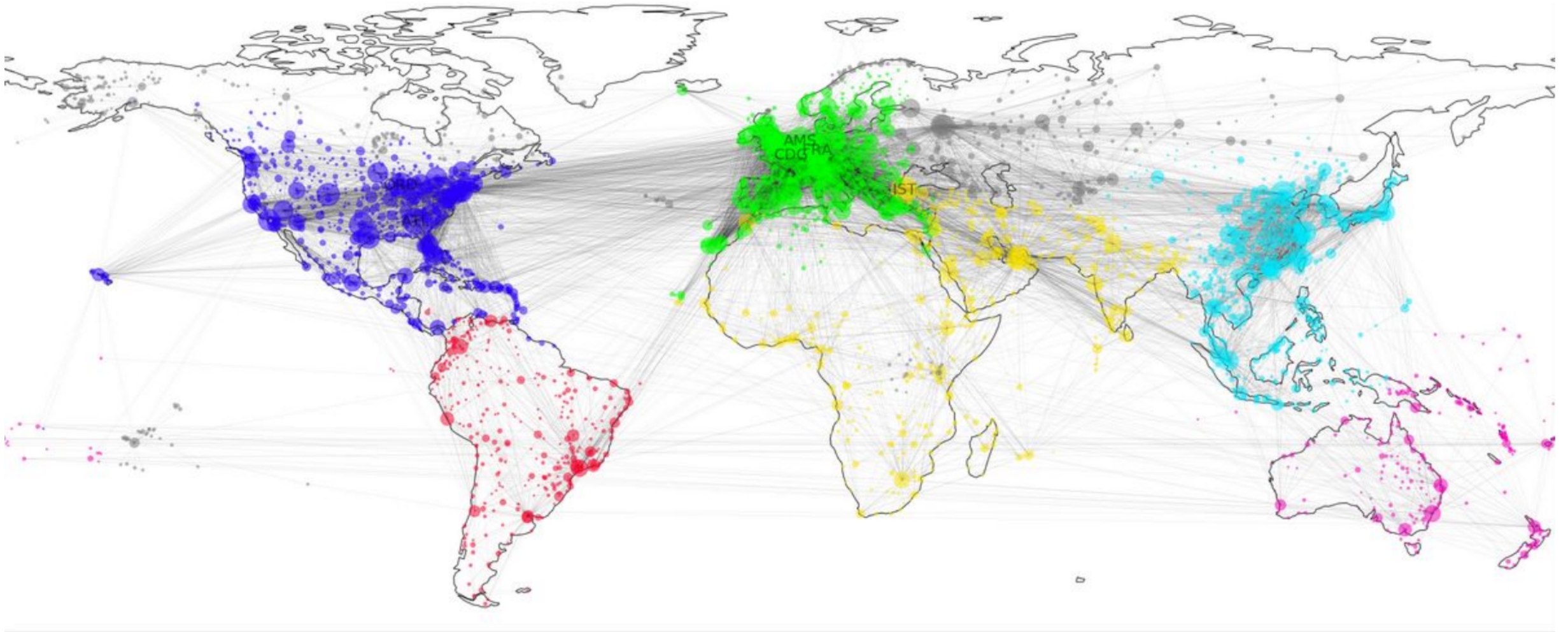


Image from https://github.com/mdeff/ntds_2018/blob/master/projects/slides/team_12.pdf

Motivation



Image from <https://www.openstreetmap.org>

Motivation

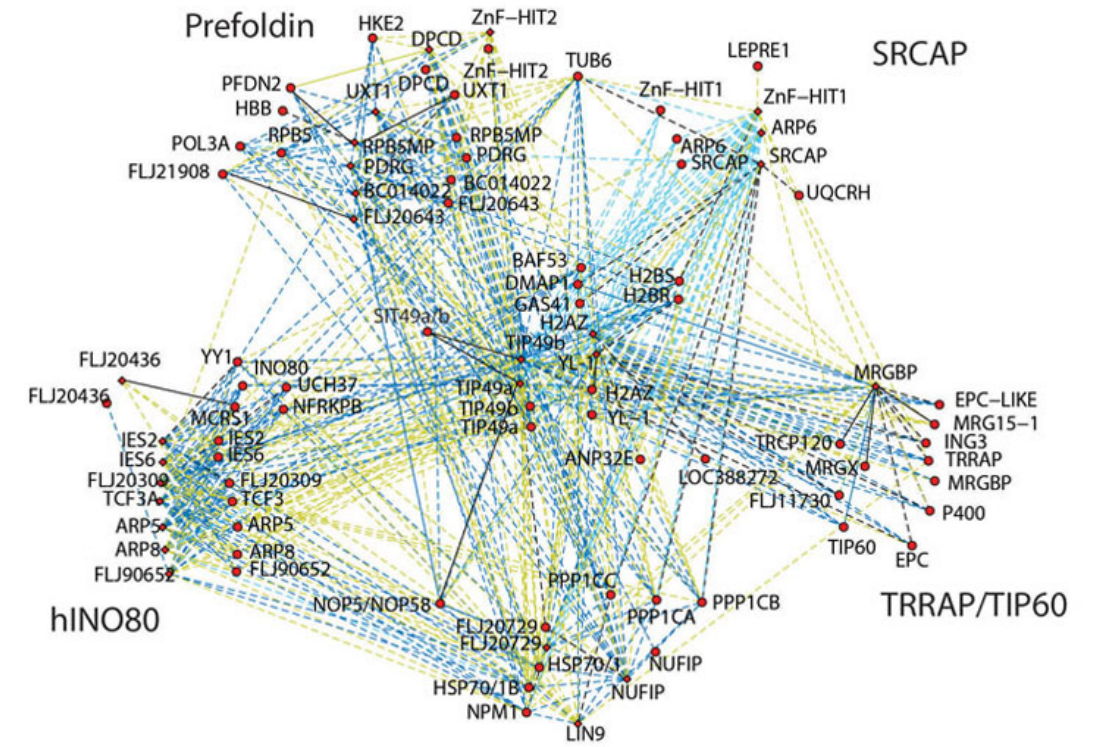
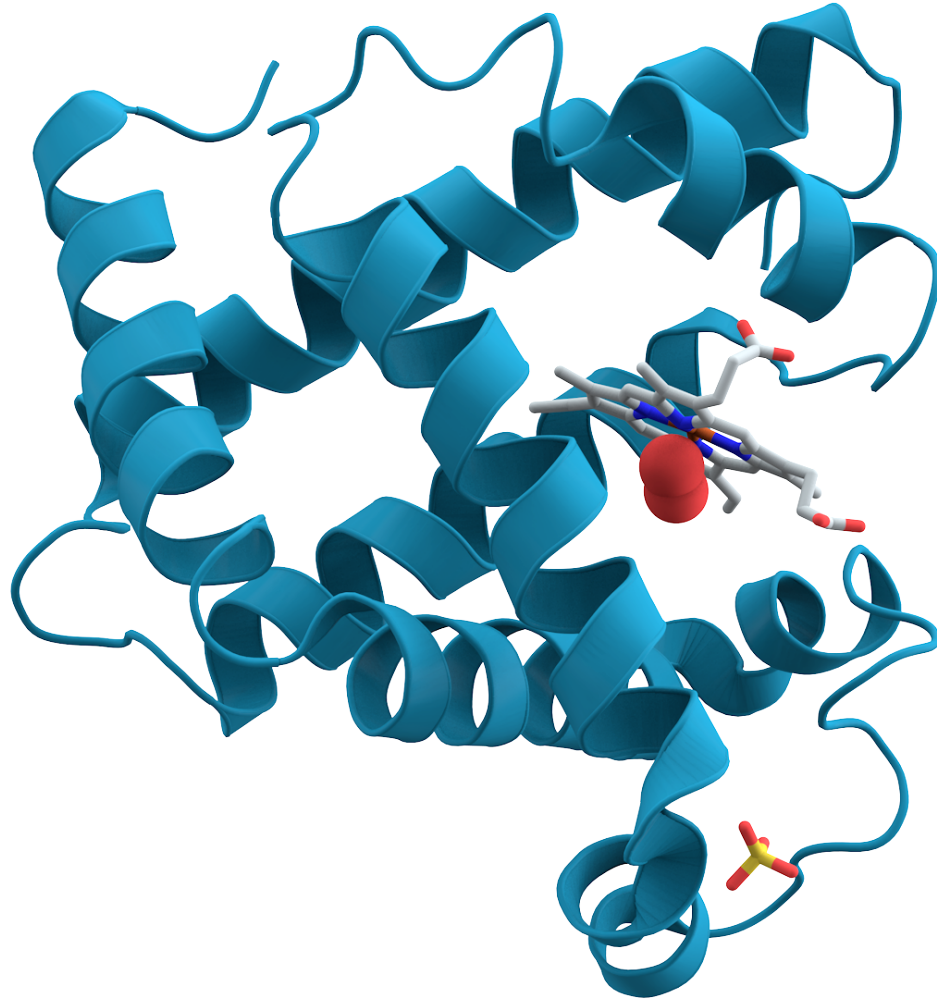


Image from <https://commons.wikimedia.org/wiki/File:Myoglobin.png>

Image from <http://research.stowers.org/proteomics/ProtNetAnal.html>

Motivation

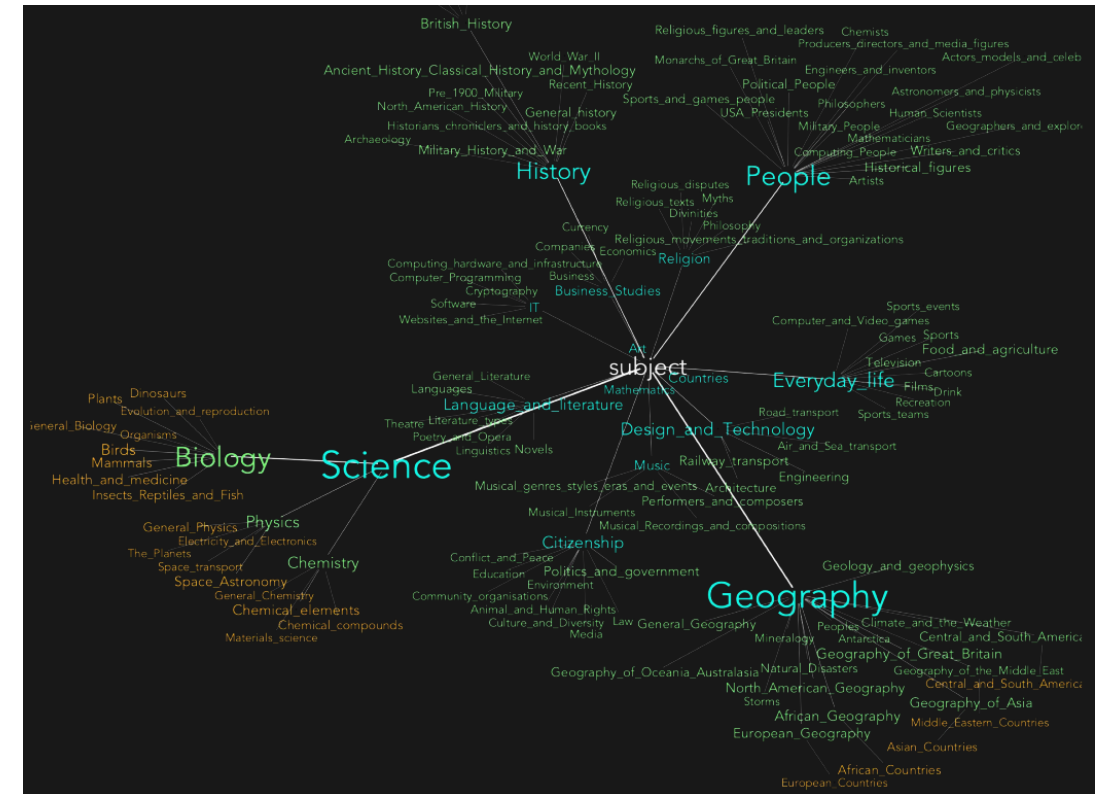
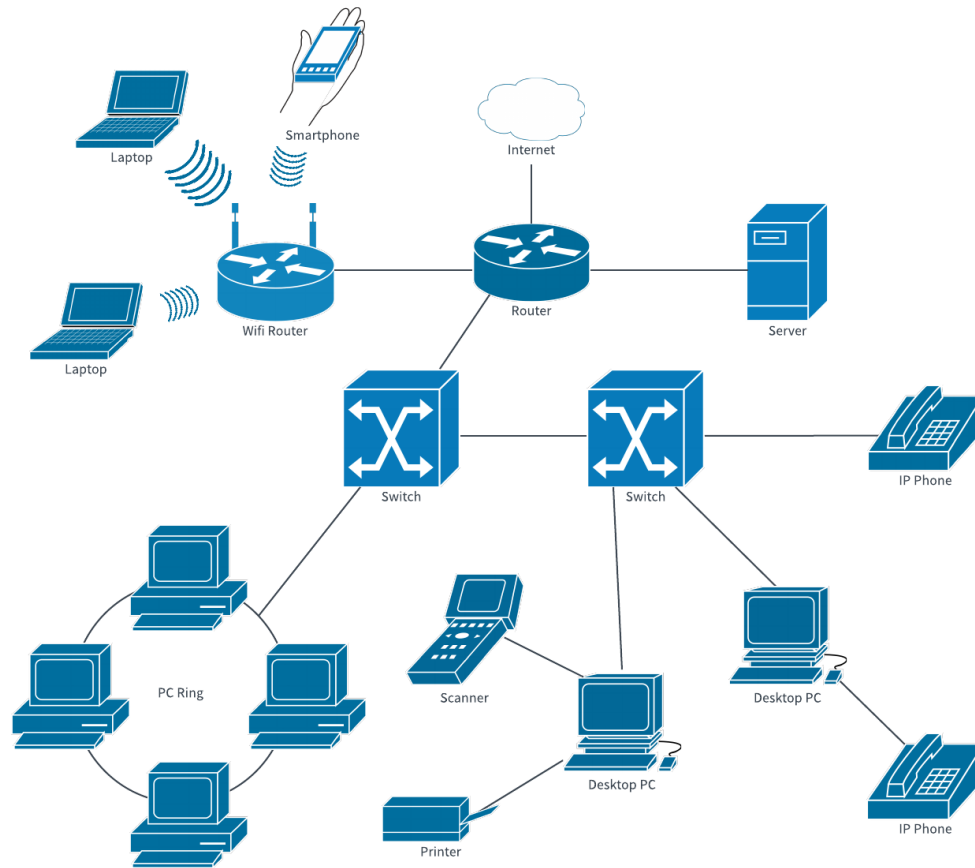


Image from <https://www.lucidchart.com/pages/templates/network-diagram/internet-network-diagram-template>

Image from https://github.com/mdeff/ntds_2018/blob/master/projects/slides/team_13.pdf

Outline

1. Basics
2. Network properties
3. Spectral methods
4. Spectral representation and filters
5. Learning on graphs

Graph

graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$

vertex set $\mathcal{V} = \{v_i\}$ with $|\mathcal{V}|$ vertices

edge set $\mathcal{E} = \{e_i\}$ with $|\mathcal{E}|$ edges

$e_k = (v_i, v_j)$ is an oriented edge from v_i to v_j

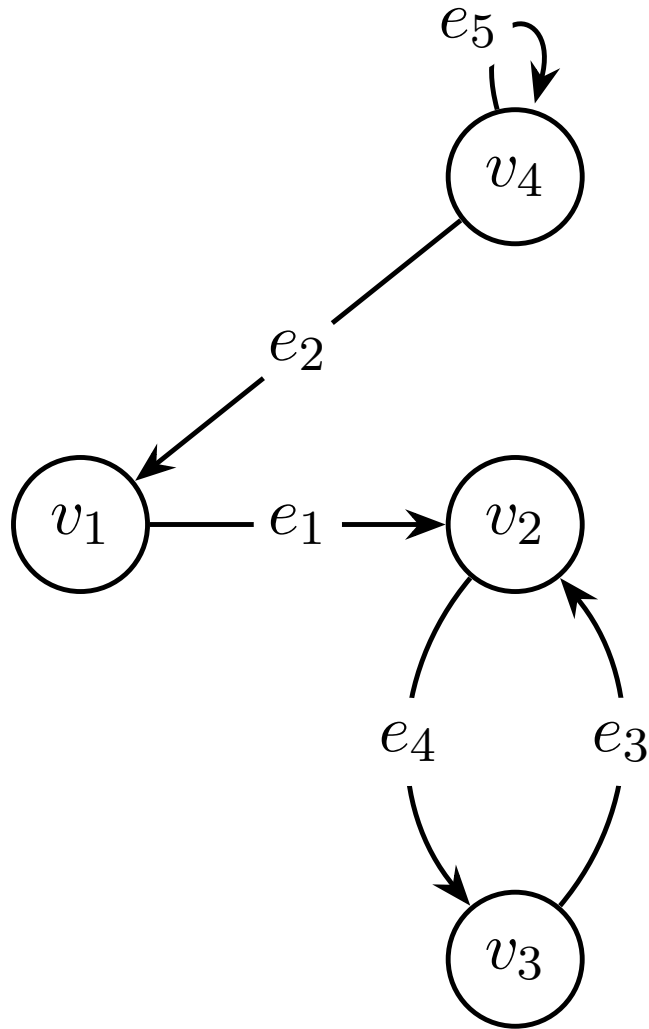
adjacency $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$

$A(i, j)$ is the weight of the edge (v_i, v_j)

$A(i, j) = 1$ if edges are not weighted

$A(i, j) = 0$ if $(v_i, v_j) \notin \mathcal{E}$

Example



$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$$

$$\mathcal{V} = \{v_1, v_2, v_3, v_4\}$$

$$\mathcal{E} = \left\{ \underbrace{(v_1, v_2)}_{e_1}, \underbrace{(v_4, v_1)}_{e_2}, \underbrace{(v_3, v_2)}_{e_3}, \underbrace{(v_2, v_3)}_{e_4}, \underbrace{(v_4, v_4)}_{e_5} \right\}$$

$$A = \begin{pmatrix} 0 & w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & w & 0 & 0 \\ w & 0 & 0 & w \end{pmatrix}, \text{ with edge weight } w$$

Degree

outdegree $D^{out} = \text{diag}(A1) = \text{diag}(d^{out})$

$d^{out}(i) = \sum_j A(i, j)$ is the (weighted) number of edges leaving v_i

indegree $D^{in} = \text{diag}(1A) = \text{diag}(d^{in})$

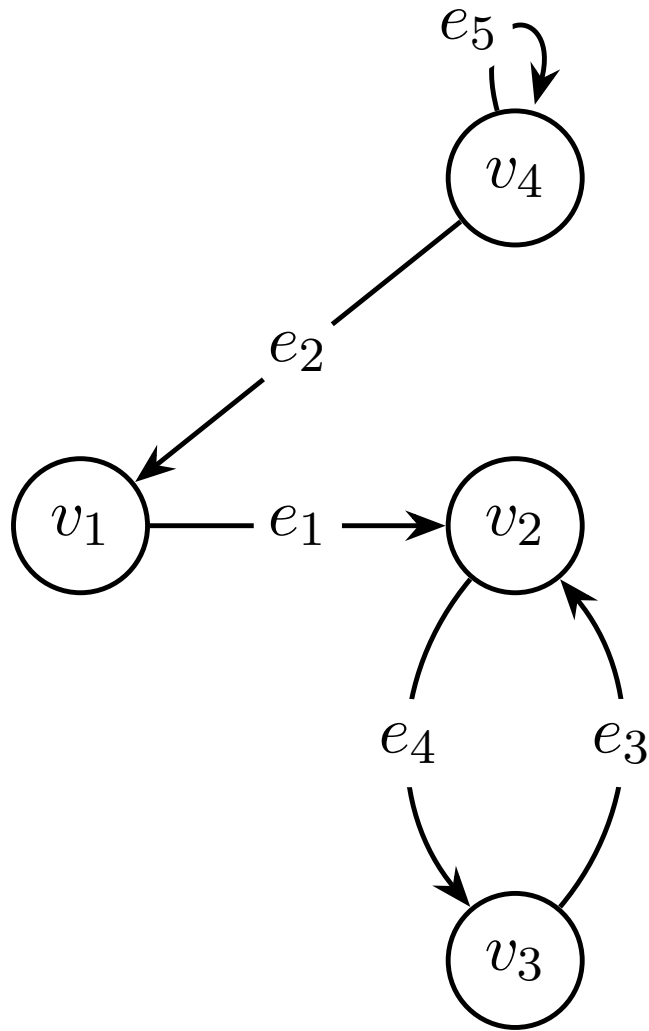
$d^{in}(j) = \sum_i A(i, j)$ is the (weighted) number of edges arriving at v_i

degree $D = \frac{1}{2}(D^{out} + D^{in}) = \text{diag}(d)$

$d(i)$ is the (weighted) number of edges connected to v_i

$(D = D^{out} = D^{in} \text{ for undirected graphs})$

Example



$$A = \begin{pmatrix} 0 & w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & w & 0 & 0 \\ w & 0 & 0 & w \end{pmatrix}$$

$$d^{out} = A1 = (w, w, w, 2w)^{\top}$$

$$d^{in} = 1A = (w, 2w, w, w)$$

$$D = \frac{1}{2}(D^{out} + D^{in}) = \begin{pmatrix} w & 0 & 0 & 0 \\ 0 & \frac{3}{2}w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & 0 & 0 & \frac{3}{2}w \end{pmatrix}$$

Signals

vertex signal a function $x : \mathcal{V} \rightarrow \mathbb{R}$ seen as a vector $x \in \mathbb{R}^{|\mathcal{V}|}$
 $x(i)$ is the value of x on vertex v_i

edge signal a function $y : \mathcal{E} \rightarrow \mathbb{R}$ seen as a vector $x \in \mathbb{R}^{|\mathcal{E}|}$
 $y(k)$ is the value of y on edge $e_k = (v_i, v_j)$

Signals are data about vertices and edges, such as features or labels.

Diffusion and random walks

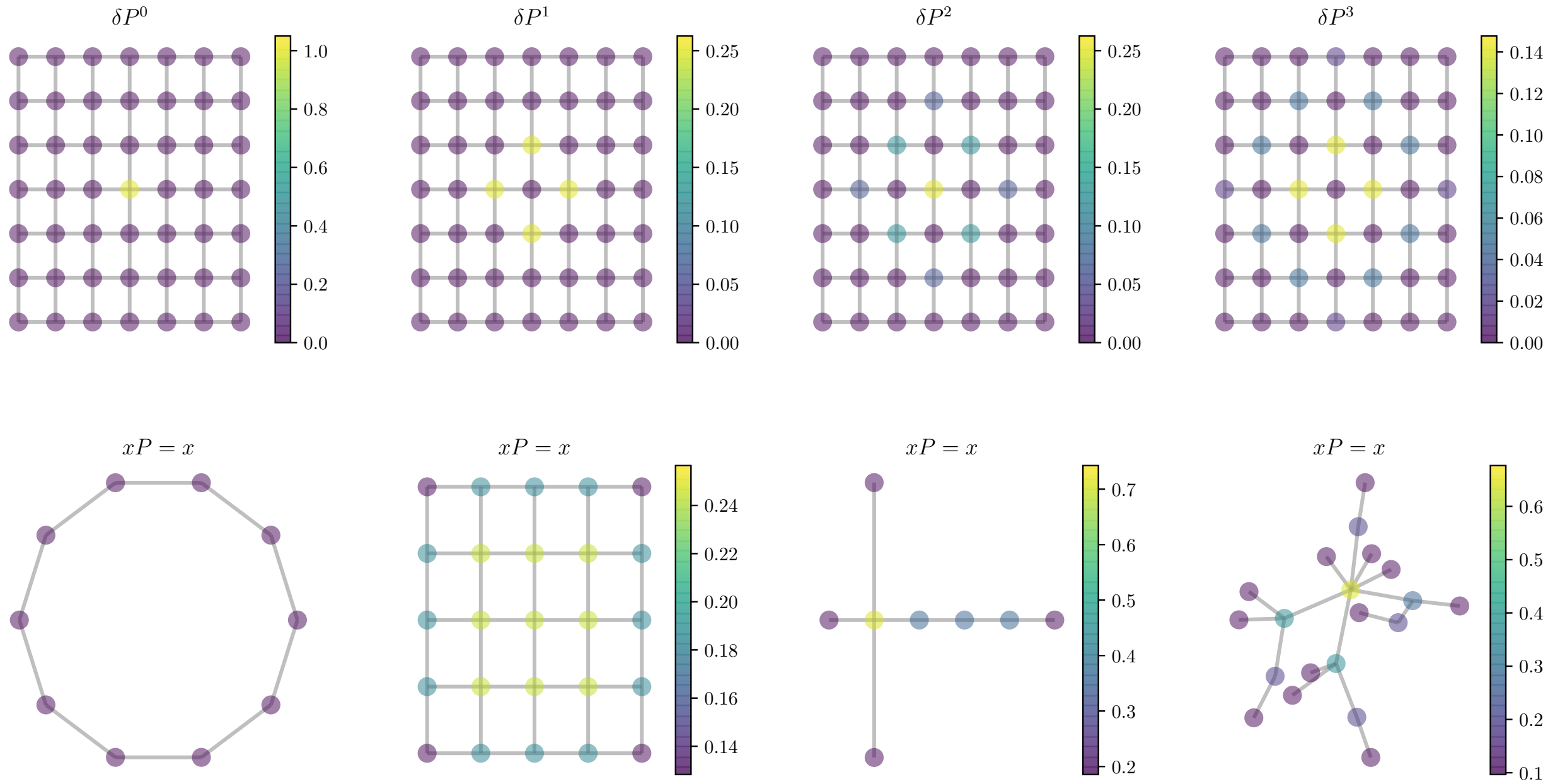
$P = D_{out}^{-1}A$ is the probability transition matrix of a Markov chain

Properties

- ▶ P is a right stochastic matrix, i.e., $P1 = 1$
- ▶ a random walker starting on v_i has probability $(\delta_i P^k)(j)$ to be on v_j after k steps¹
- ▶ there exists a *stationary* probability vector x such that $xP = x$

¹The Kronecker delta $\delta_i \in \mathbb{R}^{|\mathcal{V}|}$ has value zero at all vertices but v_i where $\delta_i(i) = 1$.

Example



Basic Network Properties

These slides have been based on **Pascal Frossard**'s classes for the EPFL course "A Network Tour of Data Science"

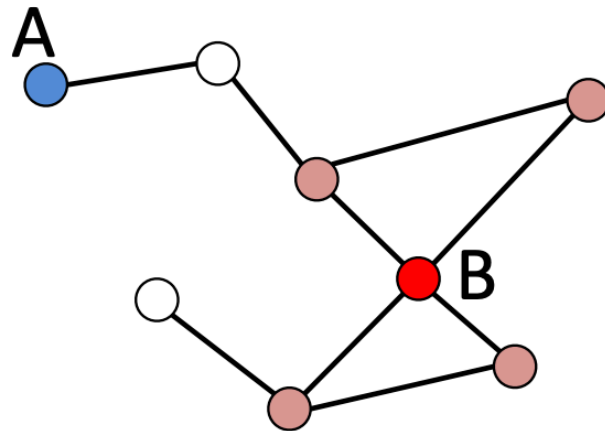


A M
L D

Outline

- Node degrees
- Adjacency matrix
- Edge density
- Connectedness
- Path lengths
- Clustering

Node degree



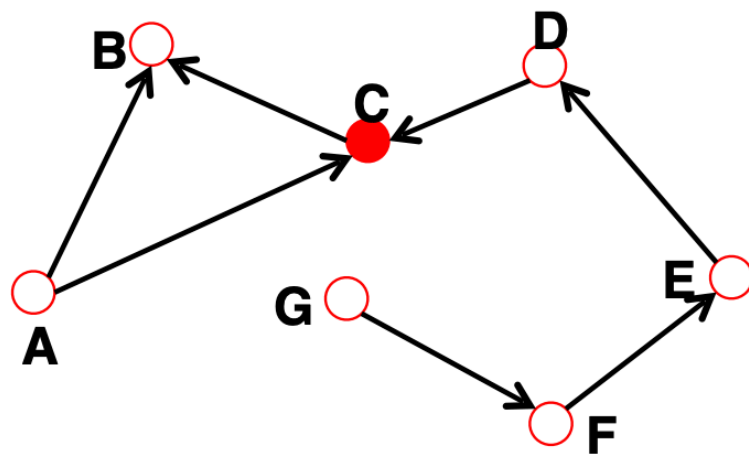
- Undirected graph

- degree k_i = number of connected edges

$$k_A = 1 \quad k_B = 4 \quad L = \frac{1}{2} \sum_{i=1}^N k_i$$

- Directed graph

- in-degree k_i^{in} = sum of incident edges
- out-degree k_i^{out} = sum of outgoing edges
- (total) degree $k_i = k_i^{in} + k_i^{out}$
- source = node with $k_i^{in} = 0$
- sink = node with $k_i^{out} = 0$

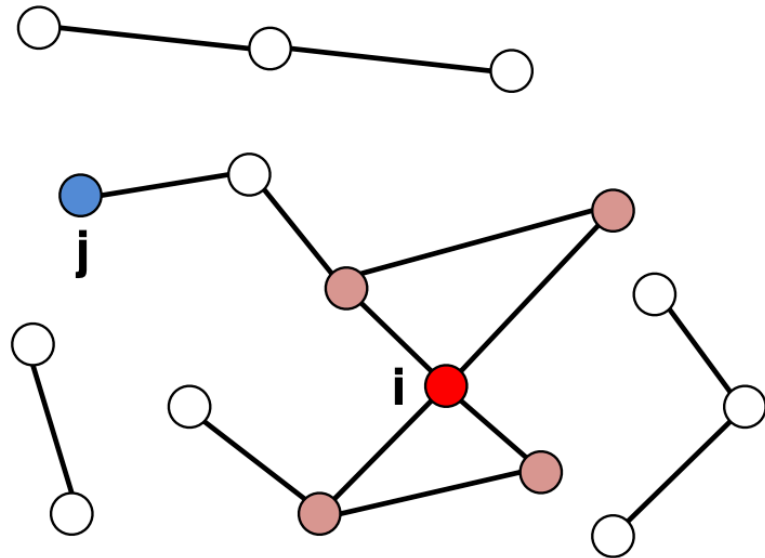


$$k_C^{in} = 2 \quad k_C^{out} = 1 \quad k_C = 3$$

For weighted graphs, the degree is the sum of the edge weights, and not the number of edges, i.e., $k_i = \sum_{j \in \mathcal{V}} w_{i,j}$

Average degree

Undirected graphs

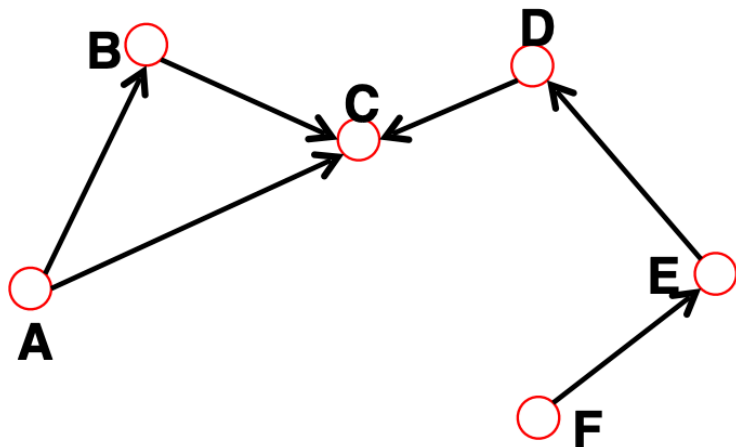


Average degree:

$$\langle k \rangle \equiv \frac{1}{N} \sum_{i=1}^N k_i$$

$$\langle k \rangle \equiv \frac{2L}{N}$$

Directed graphs



Average degrees:

$$\langle k^{in} \rangle \equiv \frac{1}{N} \sum_{i=1}^N k_i^{in}$$

$$\langle k^{out} \rangle \equiv \frac{1}{N} \sum_{i=1}^N k_i^{out}$$

$$\langle k^{in} \rangle \equiv \langle k^{out} \rangle = \frac{L}{N}$$

Average degree: I.R.L

Network	Nodes	Links	Directed / Undirected	N	L	$\langle K \rangle$
Internet	Routers	Internet connections	Undirected	192,244	609,066	6.34
WWW	Webpages	Links	Directed	325,729	1,497,134	4.60
Power Grid	Power plants, transformers	Cables	Undirected	4,941	6,594	2.67
Mobile-Phone Calls	Subscribers	Calls	Directed	36,595	91,826	2.51
Email	Email addresses	Emails	Directed	57,194	103,731	1.81
Science Collaboration	Scientists	Co-authorships	Undirected	23,133	93,437	8.08
Actor Network	Actors	Co-acting	Undirected	702,388	29,397,908	83.71
Citation Network	Papers	Citations	Directed	449,673	4,689,479	10.43
E. Coli Metabolism	Metabolites	Chemical reactions	Directed	1,039	5,802	5.58
Protein Interactions	Proteins	Binding interactions	Undirected	2,018	2,930	2.90

Degree distribution

- p_k : probability that a node has degree k

$$p_k = \frac{N_k}{N} \quad \langle k \rangle = \sum_{k=0}^{\infty} k p_k$$

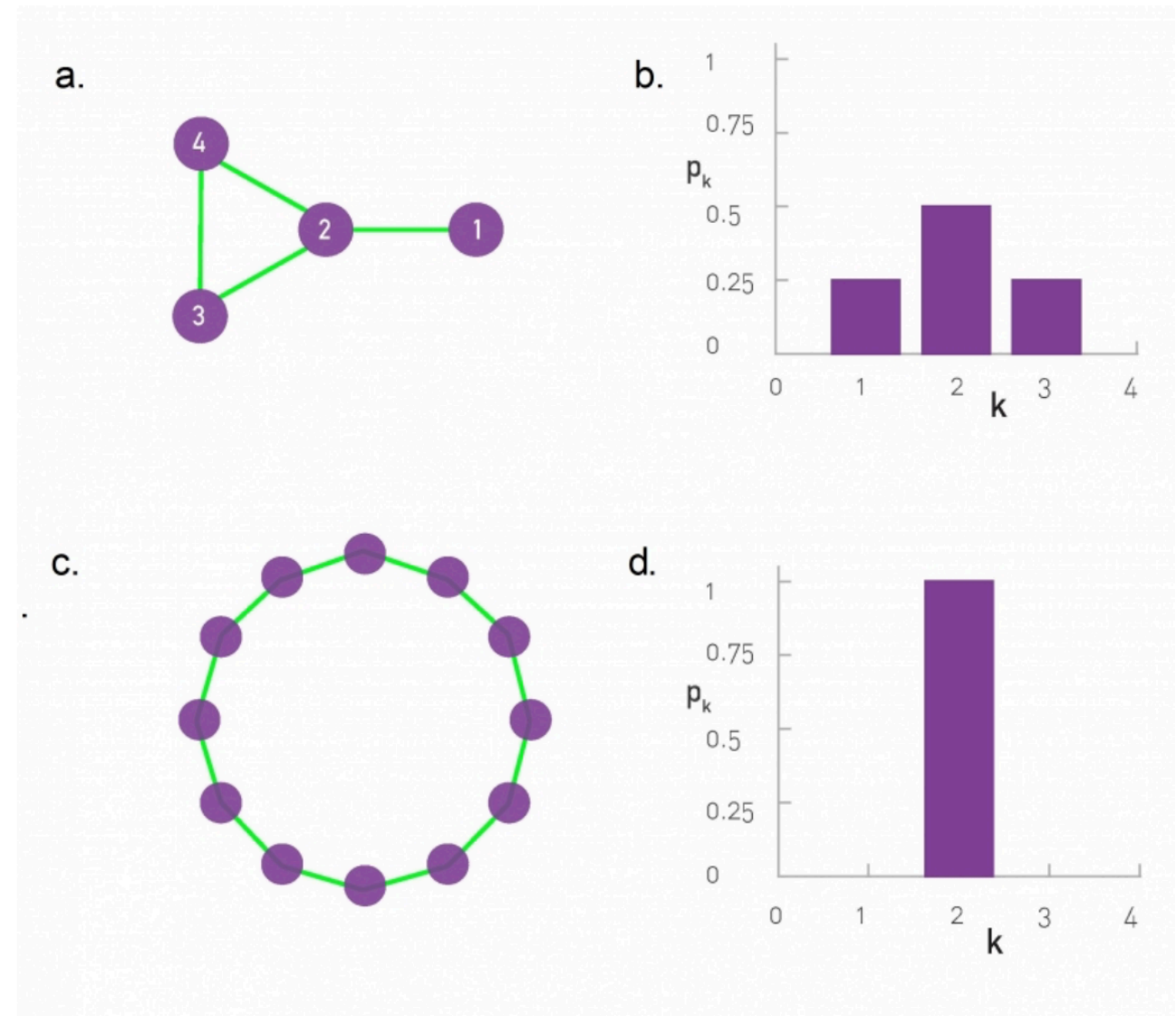
- with N_k the number of nodes of degree k

- $p(k)$: pdf of degrees

- probability of having nodes with degrees between k_1 and k_2 :

$$\int_{k_1}^{k_2} p(k) dk$$

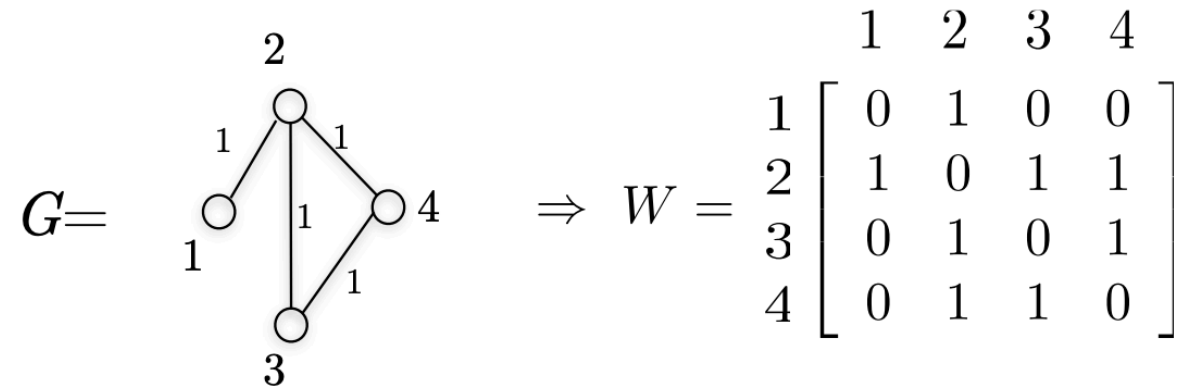
- Normalization conditions: $\sum_{k=0}^{\infty} p_k = 1$ resp., $\int_0^{\infty} p(k) dk = 1$



Adjacency matrix

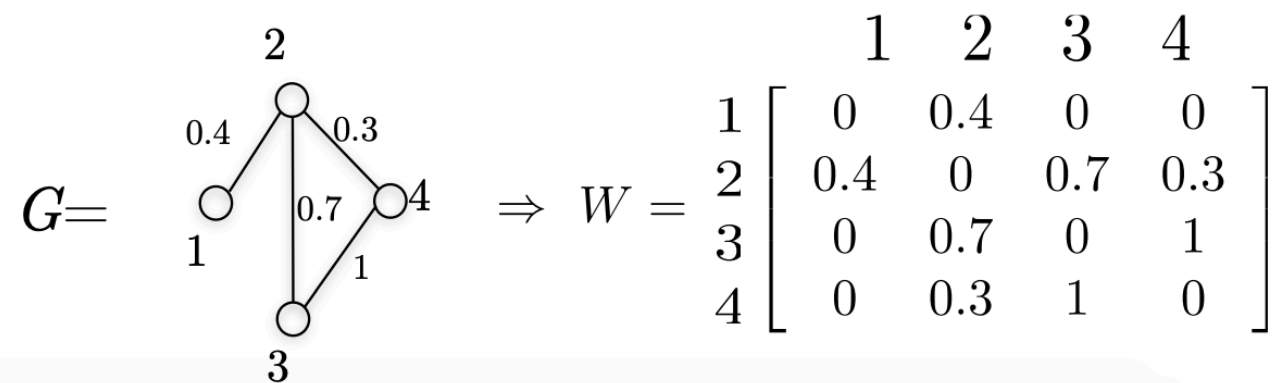
- The *adjacency* matrix A (aka *similarity*, or *weight* matrix W) captures all information about the network
- Binary matrix (unweighted graph): $w_{i,j} \in \{0, 1\}$

$$W_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$



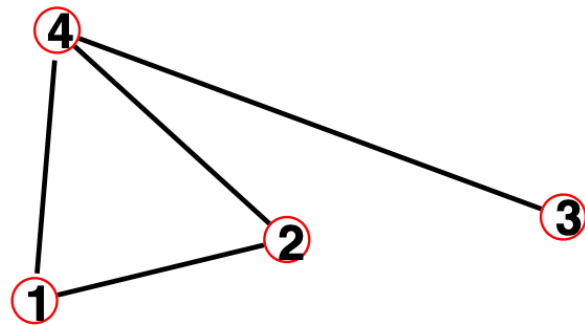
- Weighted matrix: $w_{i,j} \in [0, 1]$ (commonly normalised)

$$W_{i,j} = \begin{cases} w_{i,j} \in [0, 1], & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$



Adjacency matrix and degrees

Undirected graphs



$$W = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$W_{ij} = W_{ji}$$

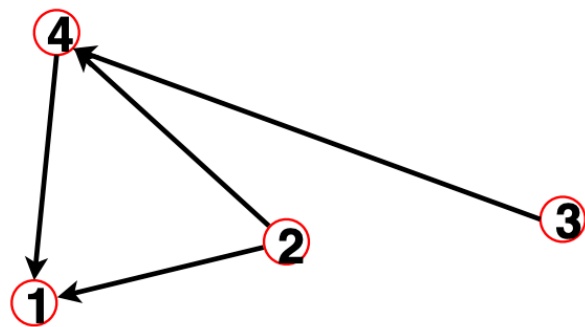
$$W_{ii} = 0$$

$$k_i = \sum_{j=1}^N W_{ij}$$

$$k_j = \sum_{i=1}^N W_{ij}$$

$$L = \frac{1}{2} \sum_{i=1}^N k_i = \frac{1}{2} \sum_{i,j} W_{ij}$$

Directed graphs



$$W = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$W_{ij} \neq W_{ji}$$

$$W_{ii} = 0$$

$$k_i^{out} = \sum_{j=1}^N W_{ij}$$

$$k_j^{in} = \sum_{i=1}^N W_{ij}$$

$$L = \sum_{i=1}^N k_i^{in} = \sum_{j=1}^N k_j^{out} = \sum_{i,j} W_{ij}$$

Density / Sparsity

- Maximum number of links in a network:

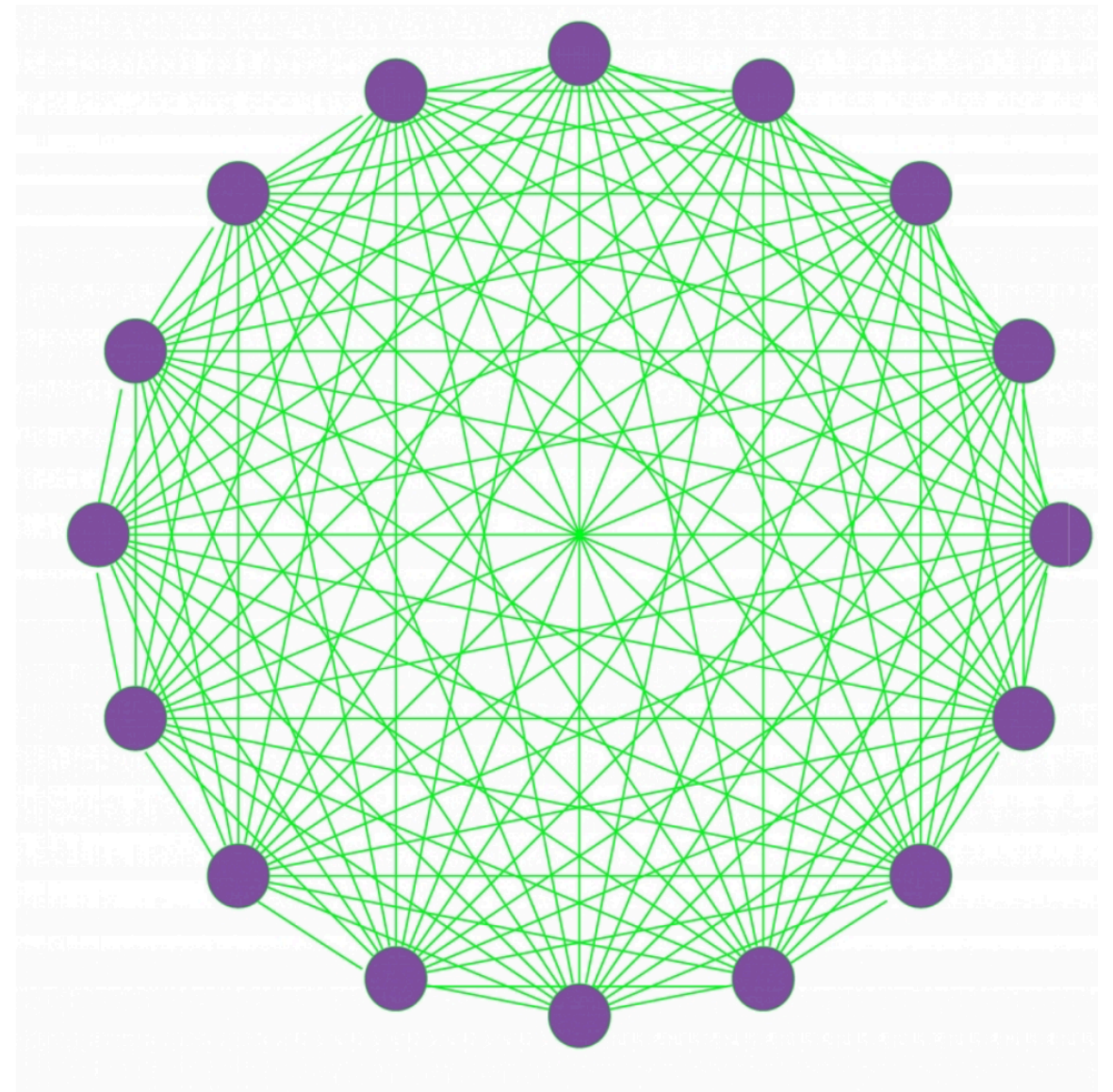
$$L_{\max} = \binom{N}{2} = \frac{N(N-1)}{2}$$

- a graph with $L = L_{\max}$ is a *complete graph*, and $L = \mathcal{O}(N^2)$
- a complete graph has an average degree of

$$\langle k \rangle = N - 1$$

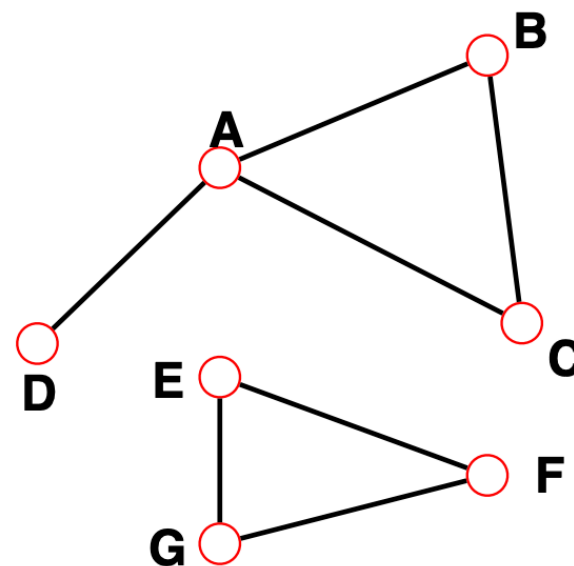
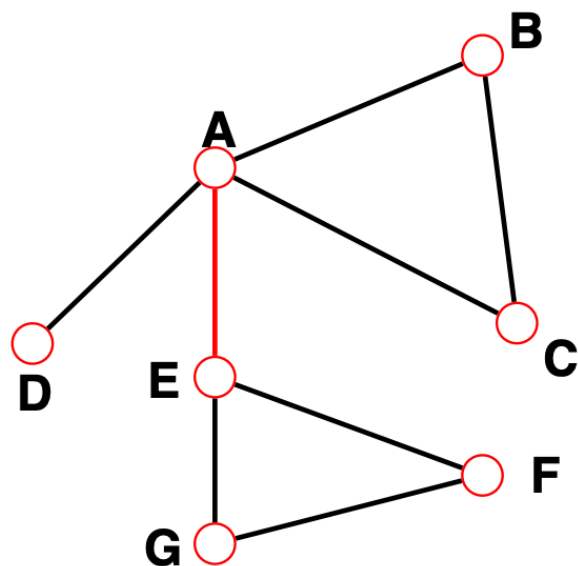
- Fortunately, most real networks are sparse:

$$\langle k \rangle \ll N - 1 \quad \text{and} \quad L \ll L_{\max}, \quad \text{typically} \quad L = \mathcal{O}(N)$$



Connectedness

- A connected (undirected) graph is a graph where any two vertices can be joined by a path.



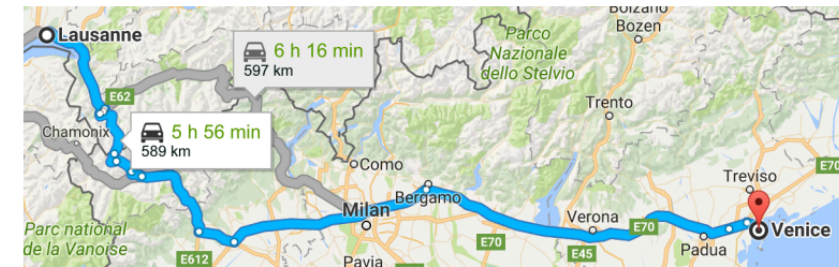
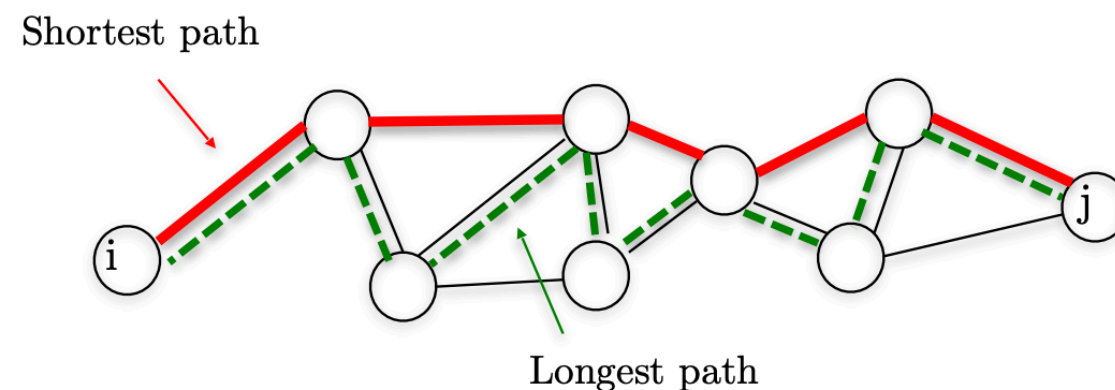
*Largest Component:
Giant Component*

*The rest: **Isolates***

- A disconnected graph is made up by two or more connected components.
 - A bridge is a link that, if erased, leads to a disconnected graph.

[Shortest] Paths

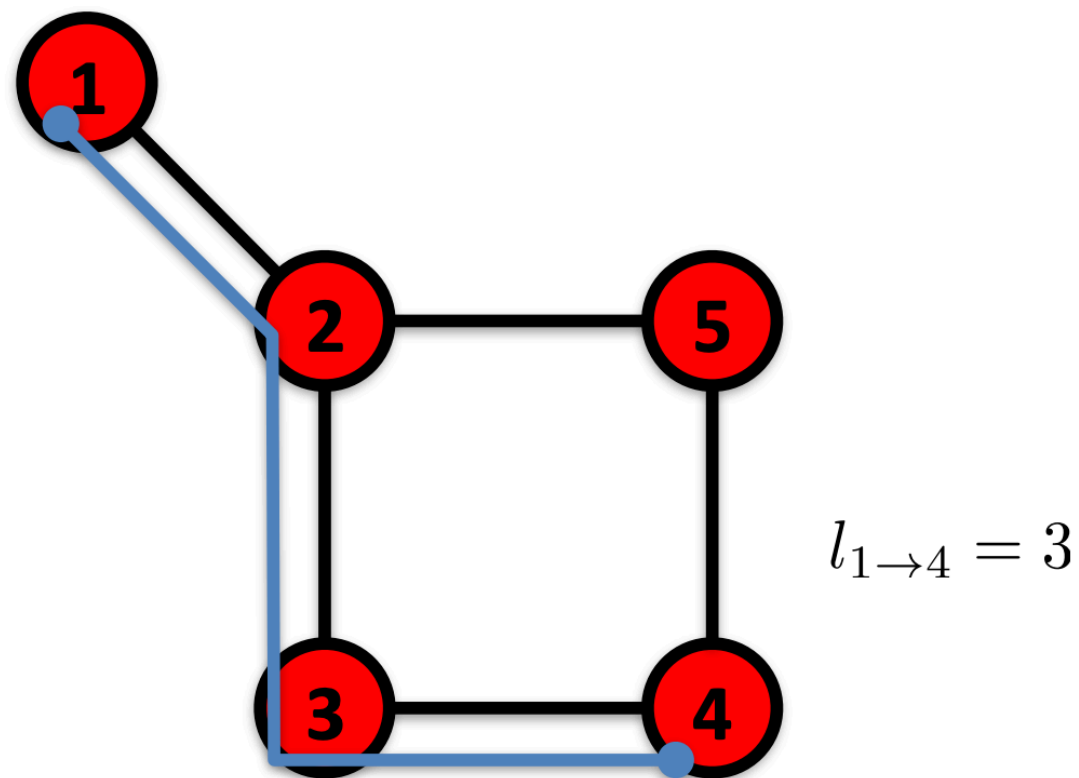
- Road navigation, e.g., Lausanne to Venice



- On weighted graphs, the distance can be defined as the sum of edge weights along the path, rather than the hop distance
 - Fast Algorithm: Dijkstra's algorithm (1956)

[Shortest] Paths: Diameter

- The network diameter d_{max} is the maximum distance between any pair of nodes in the graph (i.e., the longest shortest path).



[Average] Paths

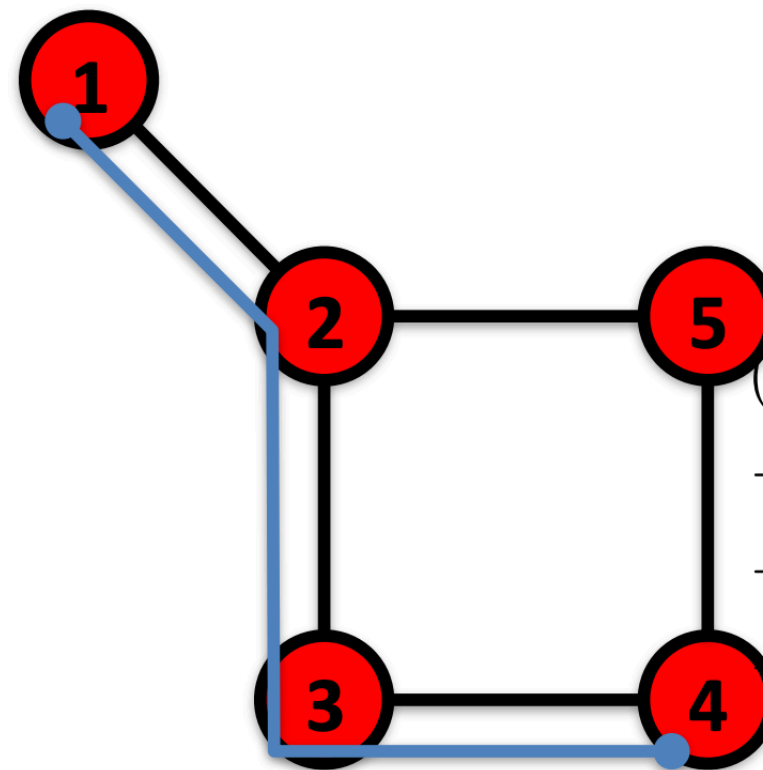
- The average path length (or average distance) for a connected graph is

$$\langle d \rangle \equiv \frac{1}{N(N-1)} \sum_{i,j \neq i} d_{ij} \text{ , or } \langle d \rangle \equiv \frac{2}{N(N-1)} \sum_{i,j > i} d_{ij} \text{ if } d_{ij} = d_{ji}$$

(undirected graphs)

- with d_{ij} the distance between node i and j

The average distance is the average of the shortest paths for all pairs of nodes.



$$(l_{1 \rightarrow 2} + l_{1 \rightarrow 3} + l_{1 \rightarrow 4} + l_{1 \rightarrow 5} + l_{2 \rightarrow 3} + l_{2 \rightarrow 4} + l_{2 \rightarrow 5} + l_{3 \rightarrow 4} + l_{3 \rightarrow 5} + l_{4 \rightarrow 5}) / 10 = 1.6$$

Clustering coefficients

- The clustering coefficient is an indication about the fraction of the node's neighbors that are connected:

$$C_i = \frac{2L_i}{k_i(k_i - 1)} \quad \text{for } k_i \notin [0, 1]$$

$C_i = 0$ otherwise

where L_i represents the number of links between the k_i neighbors of node i

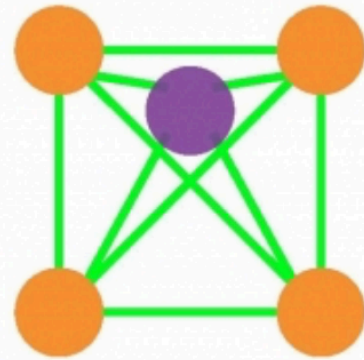
- It represents the probability that two neighbors of a node link to each other,

$$C_i \in [0, 1]$$

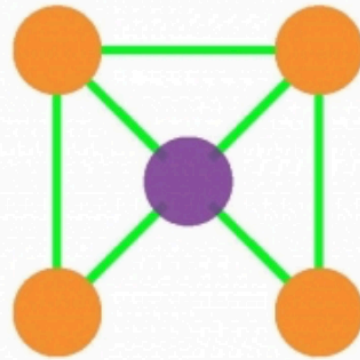
- The average clustering coefficient is given by $\langle C \rangle = \frac{1}{N} \sum_{i=1}^N C_i$

Clustering coefficients: example

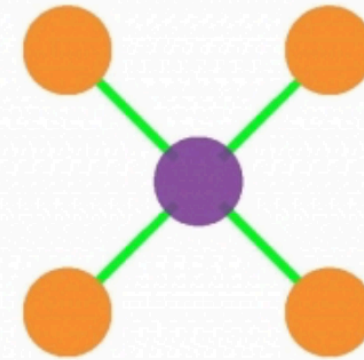
a.



$$C_i = 1$$

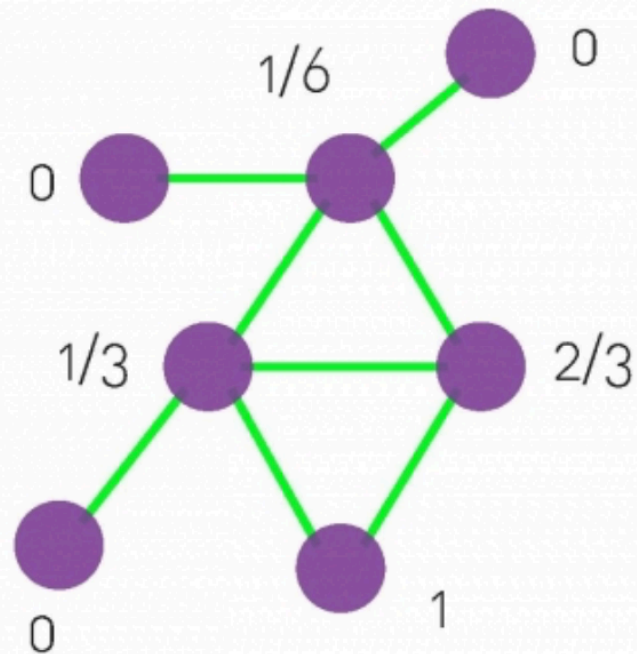


$$C_i = 1/2$$



$$C_i = 0$$

b.



$$\langle C \rangle = \frac{13}{42} \approx 0.310$$

$$C_{\Delta} = \frac{3}{8} = 0.375$$

Recap

- Important concepts:
 - Degree distribution
 - Connectedness
 - Sparsity
 - Path lengths
 - Clustering coefficient
- Reference:
 - Barabási, A. L. Network science. Cambridge university press, 2016.
URL: <http://networksciencebook.com/>

Laplacian Eigenmaps and Spectral Clustering

- References

- (Belkin and Niyogi 2003) Laplacian eigenmaps for dimensionality reduction and data representation.
- (Koren 2005) Drawing Graphs by Eigenvectors.
- (von Luxburg 2007) A tutorial on spectral clustering.

Outline

- Laplacian eigenmaps
 - From Laplacian eigenvectors to "smooth" coordinates
- Spectral clustering
 - From graph cuts to Laplacian eigenvectors
- Keep in mind: $f^\top L f := \sum_{i,j} W_{ij} |f(i) - f(j)|^2$ is **small** for which function?

$$L = D - A$$

[Cheatsheet]



$$L_{\text{norm}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

$$D = \text{diag} \left(\left(\sum_{j=1}^n W_{ij} \right)_i \right)$$

Laplacian Eigenmaps

... or how to place similar points close by,
while keeping other nodes as far away.

Dimensionality reduction

- Dataset: large matrix $X \in \mathbb{R}^{N \times d}$
 -  number of data points
 -  dimension of each data point
- Often, d is very large (e.g. images), so we need to reduce it
 - For computations
 - For visualization ($d = 2$ or 3)
- **Main question:** how to reduce d while preserving some properties (usually pairwise distances)

Formulation: spectral graph embedding

- **Guideline:** similar points are embedded close together

$$W(i, j) \text{ LARGE} \implies \|y_i - y_j\|_2 \text{ small}$$

- Proposal: $\min_{y_1, \dots, y_n} W(i, j) \|y_i - y_j\|_2$
- But also avoid collapse $y_1 = \dots = y_n = 0$
 - Solution: fix $\|y_i\|_2 = 1/\sqrt{\deg(i)}$
- And avoid trivial eigenvector $\langle y_i, D1 \rangle = 0$

Full problem

Algorithm: **Laplacian Eigenmaps**

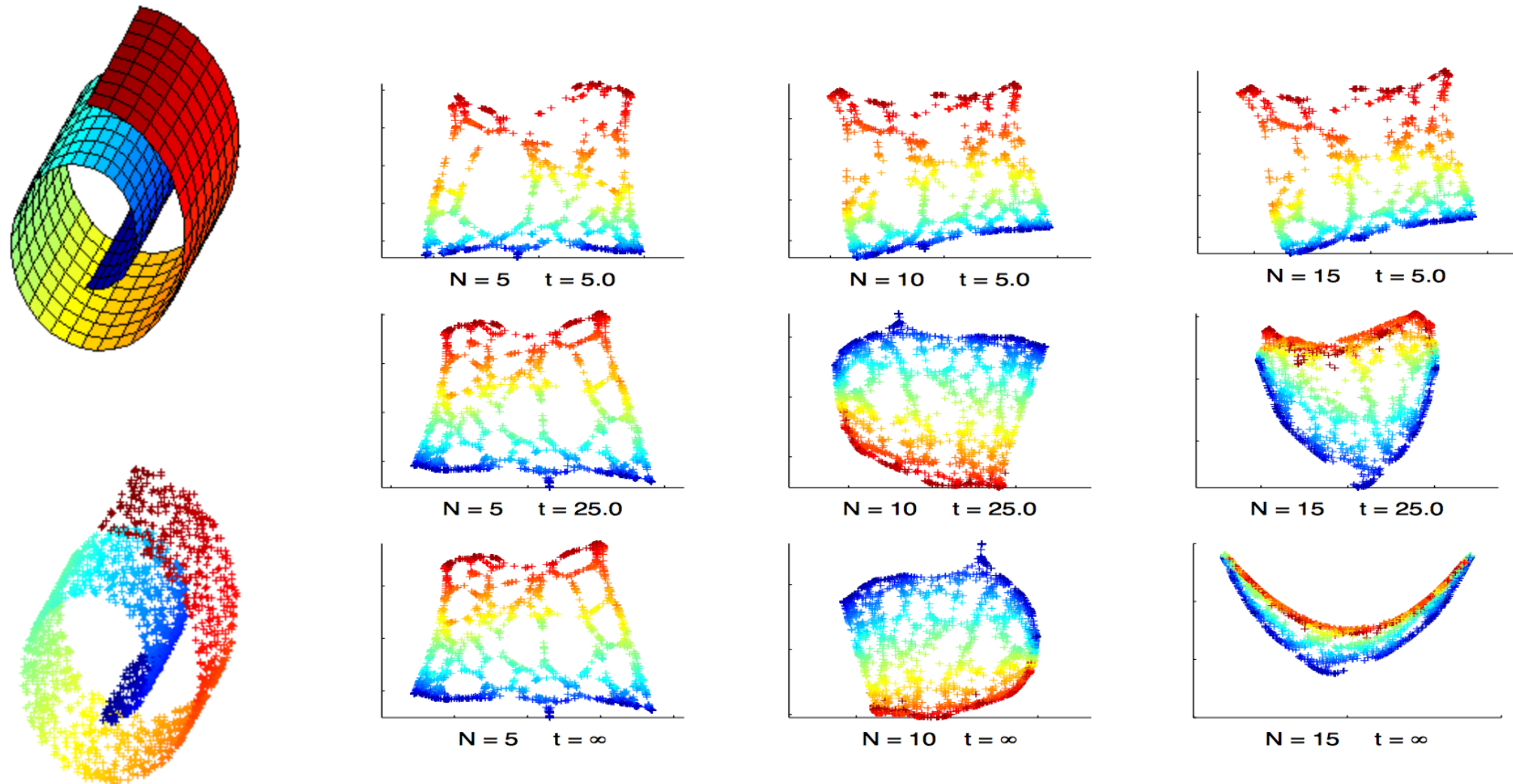
1. Solve $Y = \arg \min_{Z \in \mathbb{R}^{N \times p}} Z^\top L Z$ s.t. $Z^\top D Z = I_p$ and $Z^\top D \mathbf{1} = 0$

2. Collect embedding coordinates from the rows of Y

- Coordinate maps produced are "smooth" functions over the graph.

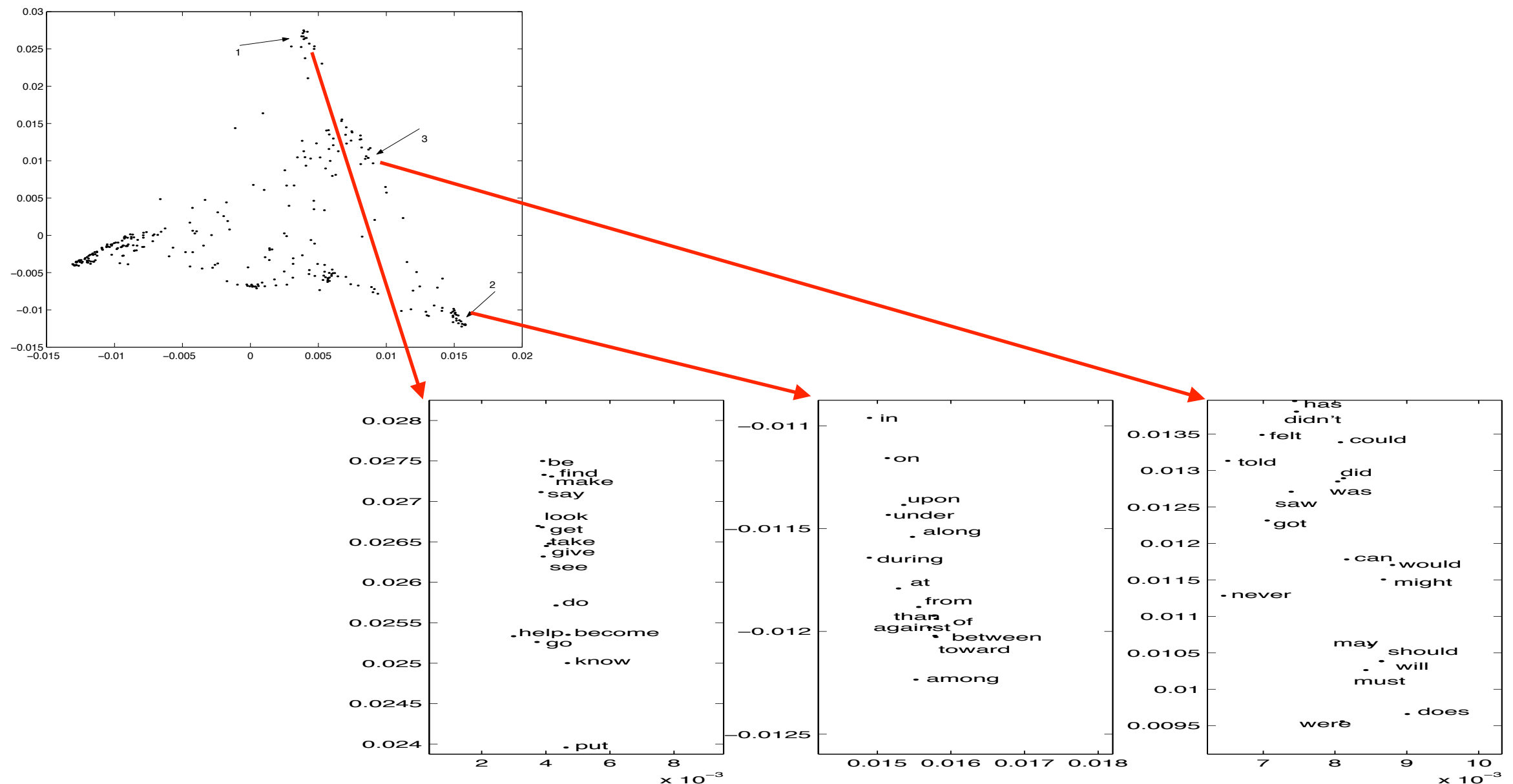
Example: Swiss-roll

- (Belkin and Niyogi 2003) Laplacian eigenmaps for dimensionality reduction and data representation Neural Computation, 15, 1373–1396.



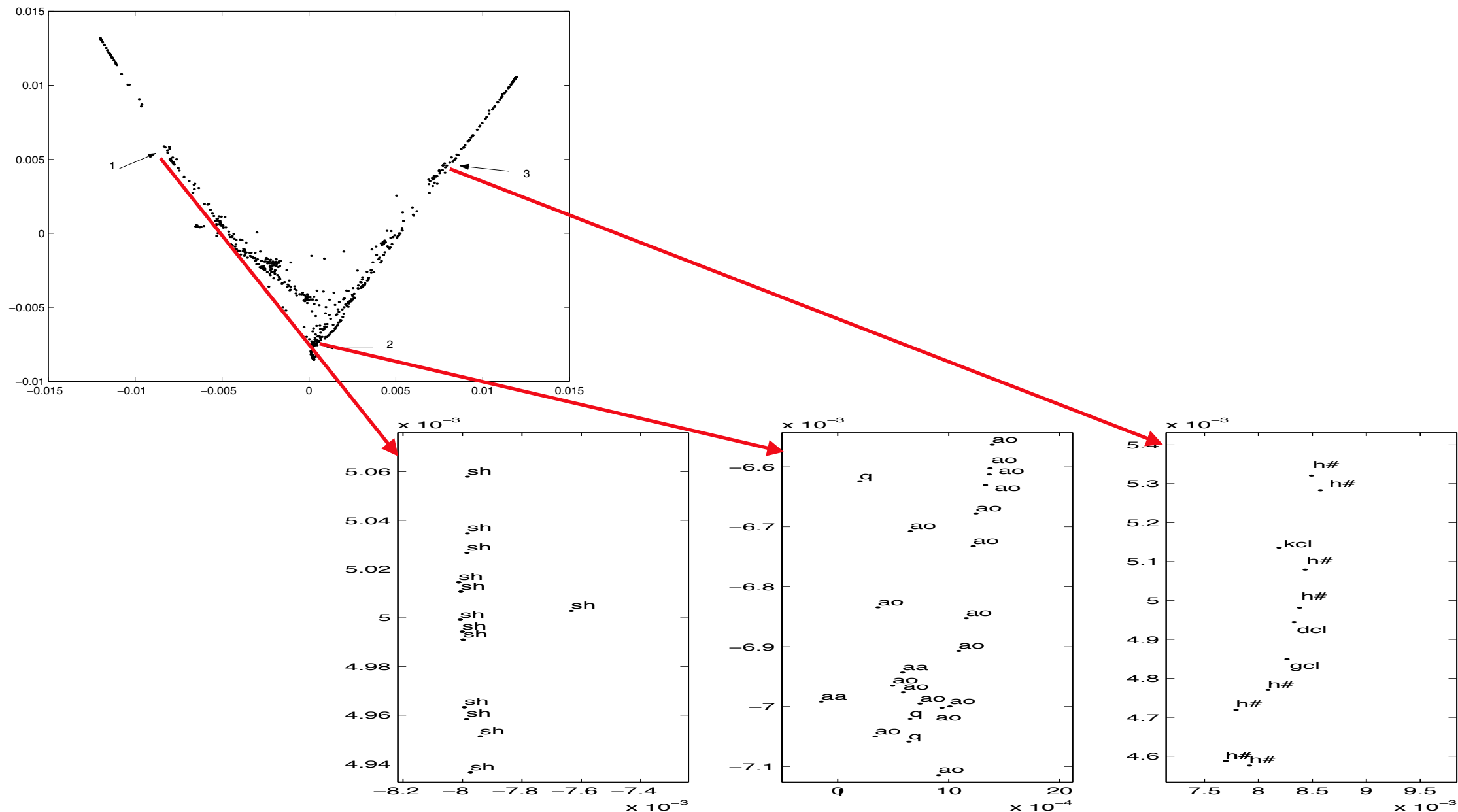
Example: text

- (Belkin and Niyogi 2003) Laplacian eigenmaps for dimensionality reduction and data representation Neural Computation, 15, 1373–1396.



Example: speech

- (Belkin and Niyogi 2003) Laplacian eigenmaps for dimensionality reduction and data representation Neural Computation, 15, 1373–1396.



Recap

- Laplacian eigenmaps
 - Embedding given by the first couple of Laplacian eigenvectors
 - Coordinate maps are "smooth" on the graph

$$L = D - A$$

[Cheatsheet]

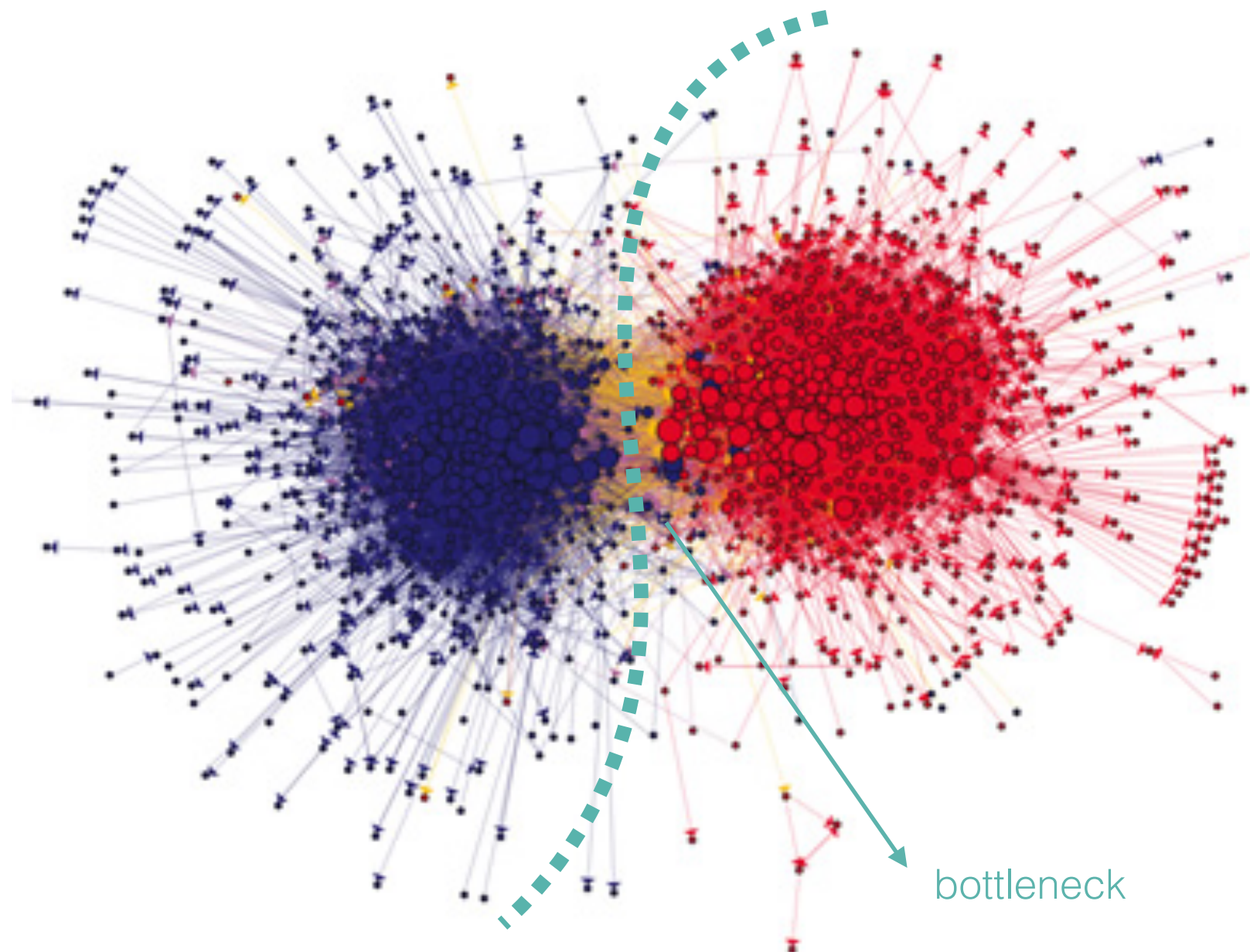
$$L_{\text{norm}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

$$D = \text{diag} \left(\left(\sum_{j=1}^n W_{ij} \right)_i \right)$$

Spectral Clustering

... or what do Laplacian eigenvectors have to do with cluster assignments?

Intuition: minimum cuts



Cost of cluster cuts

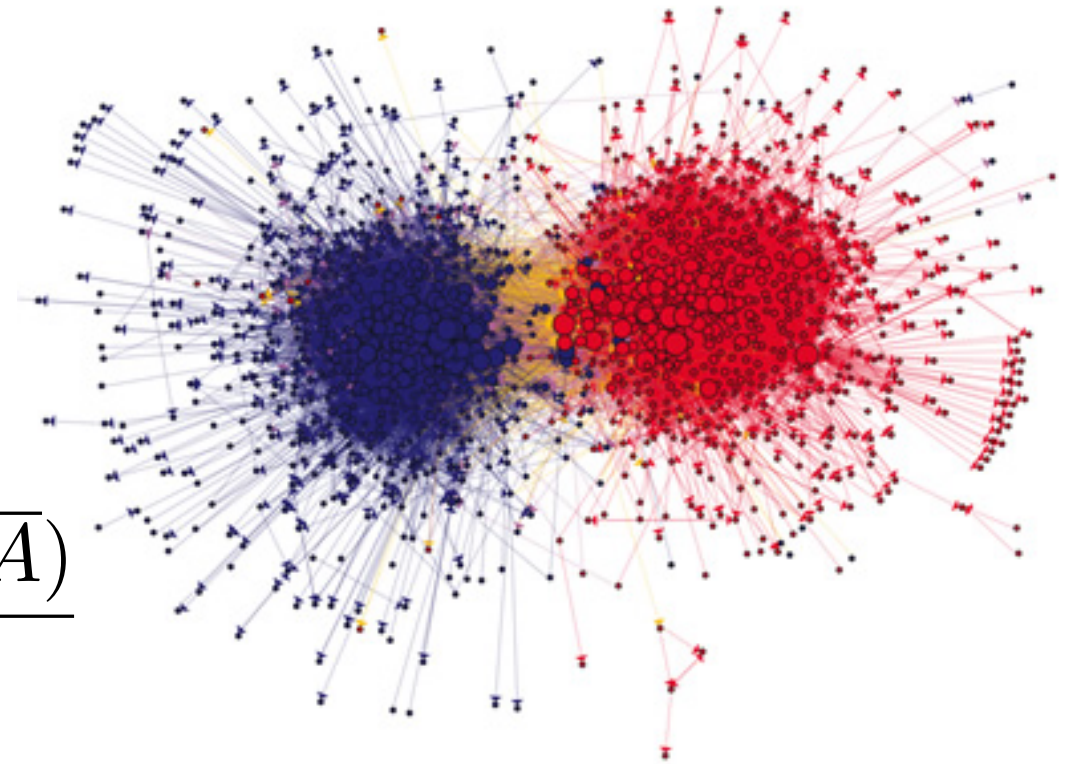
- Ratio Cut:

$$C(A, B) := \sum_{i \in A, j \in B} \mathbf{W}[i, j]$$

$$\text{RatioCut}(A, \bar{A}) := \frac{1}{2} \frac{C(A, \bar{A})}{|A|} + \frac{1}{2} \frac{C(A, \bar{A})}{|\bar{A}|}$$

$$f[i] = \begin{cases} \sqrt{|\bar{A}|/|A|} & \text{if } i \in A \\ -\sqrt{|A|/|\bar{A}|} & \text{if } i \in \bar{A} \end{cases}$$

- Cost equivalence: $f^T \mathbf{L} f = |V| \text{RatioCut}(A, \bar{A})$



Exposing RatioCut

- From a problem equivalent to RatioCut...

$$\arg \min_{f \text{ indicator of } A \subseteq V} f^\top L f \text{ s.t. } \|f\|_2 = \sqrt{N} \text{ and } \langle f, 1 \rangle = 0$$

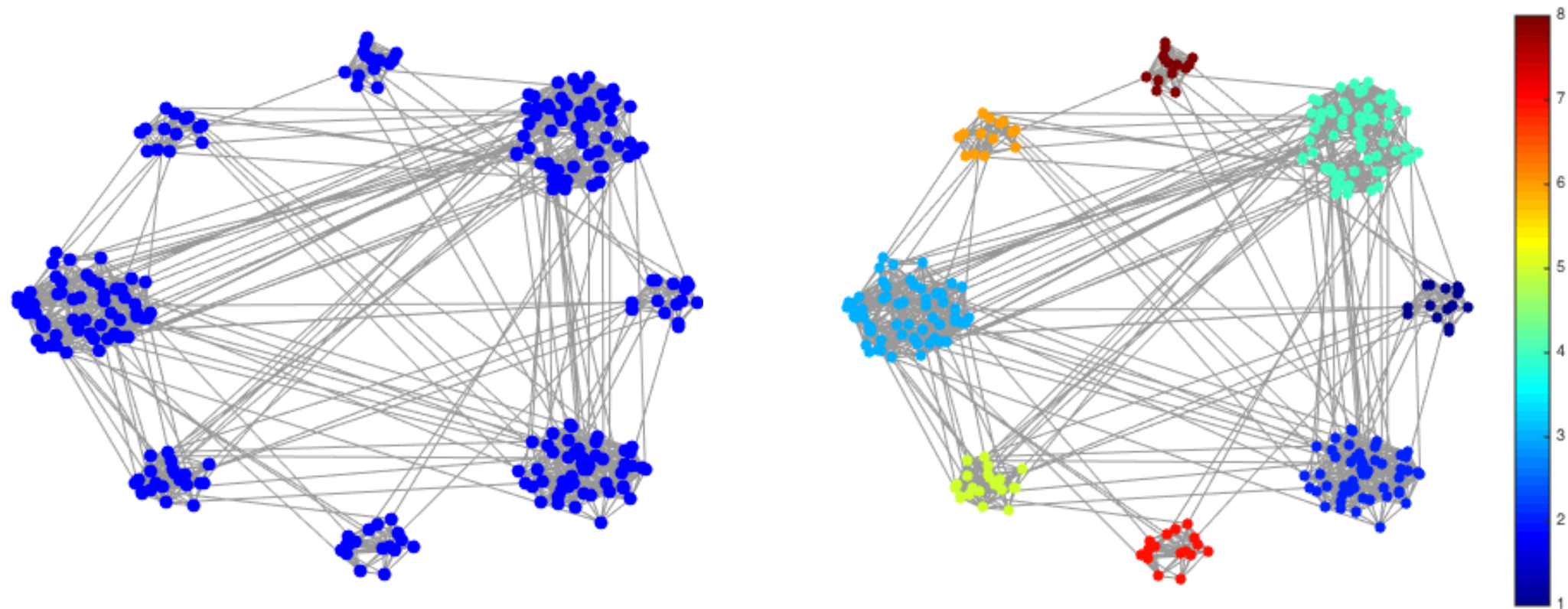
↓
NP-hard

- ... to its continuous relaxation

$$\arg \min_{f \in \mathbb{R}^N} f^\top L f \text{ s.t. } \|f\|_2 = \sqrt{N} \text{ and } \langle f, 1 \rangle = 0$$

↘ Solution: first non-trivial eigenvector of L (Fiedler vector)!

Spectral clustering algorithm



Algorithm: **Spectral clustering**

1. Compute U_k , the matrix containing the first k eigenvectors of L .
2. Compute cluster assignment via k-means from those eigenvectors.

Recap

- Spectral clustering
 - Cluster indicators are "smooth" on the graph
 - Laplacian eigenvectors provide centers for k-means

$$L = D - A$$

$$L_{\text{norm}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}$$

[Cheatsheet]

$$D = \text{diag} \left(\left(\sum_{j=1}^n W_{ij} \right)_i \right)$$

SPECTRAL REPRESENTATION AND FILTERING

Michaël DEFFERRARD

Signals

vertex signal a function $x : \mathcal{V} \rightarrow \mathbb{R}$ seen as a vector $x \in \mathbb{R}^{|\mathcal{V}|}$
 $x(i)$ is the value of x on vertex v_i

edge signal a function $y : \mathcal{E} \rightarrow \mathbb{R}$ seen as a vector $x \in \mathbb{R}^{|\mathcal{E}|}$
 $y(k)$ is the value of y on edge $e_k = (v_i, v_j)$

Signals are data about vertices and edges, such as features or labels.

Diffusion and random walks

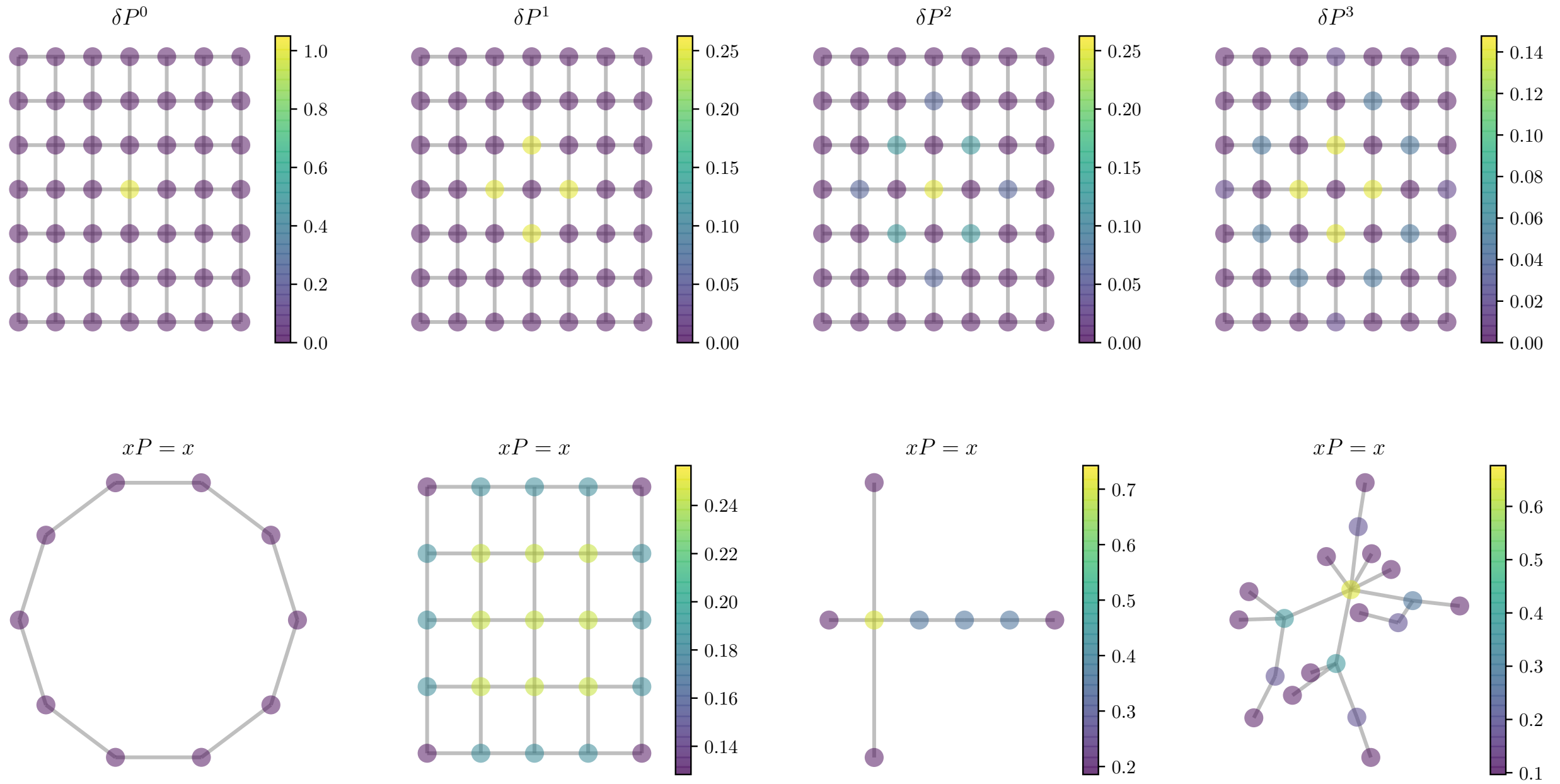
$P = D_{out}^{-1}A$ is the probability transition matrix of a Markov chain

Properties

- ▶ P is a right stochastic matrix, i.e., $P1 = 1$
- ▶ a random walker starting on v_i has probability $(\delta_i P^k)(j)$ to be on v_j after k steps¹
- ▶ there exists a *stationary* probability vector x such that $xP = x$

¹The Kronecker delta $\delta_i \in \mathbb{R}^{|\mathcal{V}|}$ has value zero at all vertices but v_i where $\delta_i(i) = 1$.

Example



Fourier transform

Introduced to study the heat equation. Why?

$$\frac{\partial f}{\partial x^2} = \frac{\partial f}{\partial t}$$



Joseph Fourier (1768 – 1830)

Fourier basis

Answer: it diagonalizes the Laplace operator.

$$L = U\Lambda U^\top \qquad u_k = \arg \min_{\substack{u \in \mathbb{R}^{|\mathcal{V}|} \\ \|u\|_2=1 \\ u \perp \{u_1, \dots, u_{k-1}\}}} u^\top L u$$

eigenvectors $U = (u_1, \dots, u_{|\mathcal{V}|}), U^\top U = I$

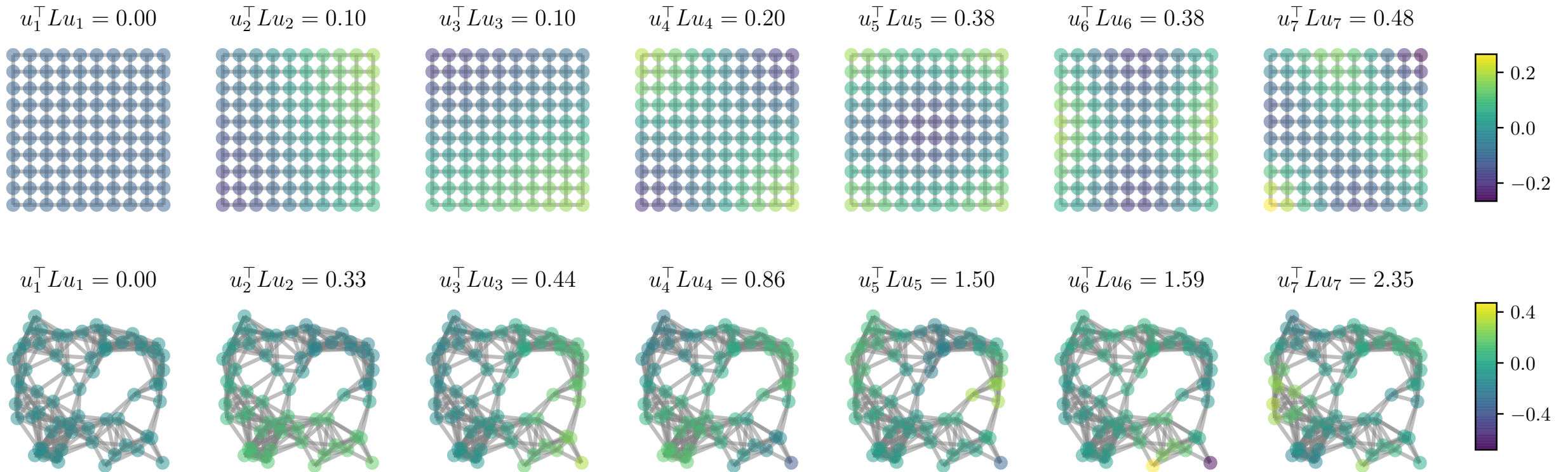
u_k is the k -th Fourier mode s.t. $Lu_k = \lambda_k u_k$

eigenvalues $\Lambda = \text{diag}((\lambda_1, \dots, \lambda_{|\mathcal{V}|})) = U^\top L U$

$\lambda_k = u_k^\top L u_k$ is the frequency associated to u_k

Example

Fourier mode u_k associated to frequency $\lambda_k = u_k^\top L u_k$.



Fourier transform

transform $\hat{x} = \mathcal{F}_{\mathcal{G}}\{x\} = U^{\top} x$

$\hat{x}(k) = \langle x, u_k \rangle$ measures how much frequency λ_k is present in x

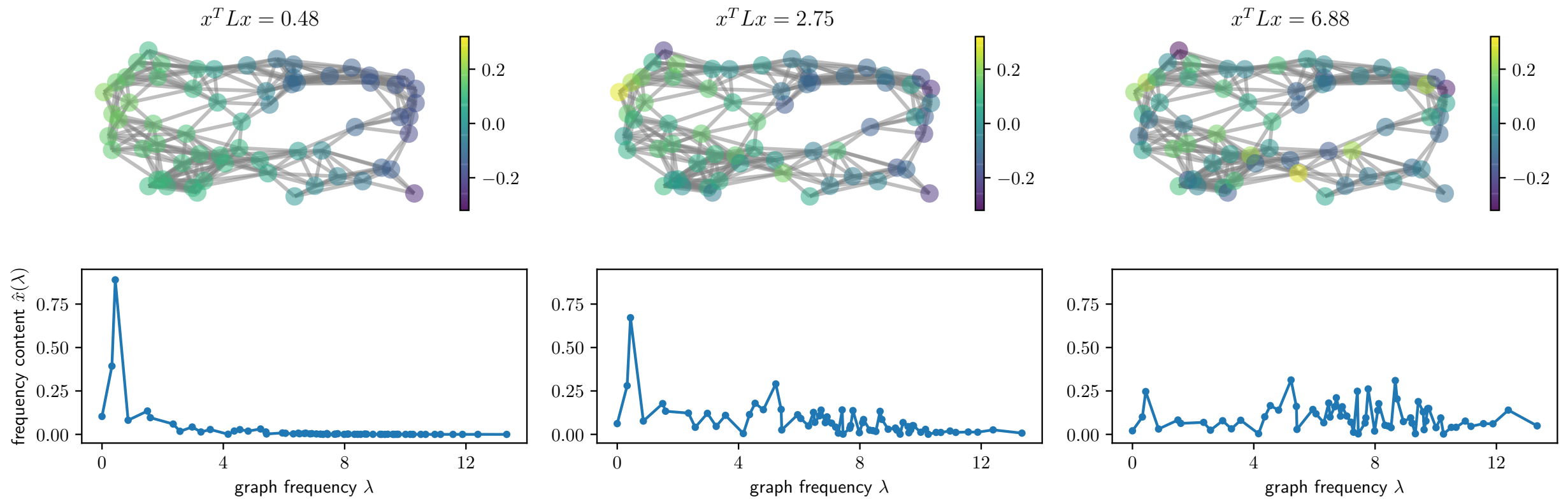
inverse $x = \mathcal{F}_{\mathcal{G}}^{-1}\{\hat{x}\} = U\hat{x} = UU^{\top}x = Ix$

Interpretation

- ▶ change of basis (from vertex to spectral): $x \Rightarrow \hat{x}$ and $L \Rightarrow \Lambda$
- ▶ projections of x on the Fourier modes u_k
- ▶ harmonic decomposition $x = \sum_k \hat{x}(k)u_k$

Example

Vertex domain representation x and spectral domain representation $\hat{x} = U^\top x$.



Filtering

kernel a function $g : \mathbb{R} \rightarrow \mathbb{R}$ that defines the action of the filter

filter an operator acting on signals represented by $g(L)$

A signal $x \in \mathbb{R}^{|\mathcal{V}|}$ is filtered by the kernel g as:

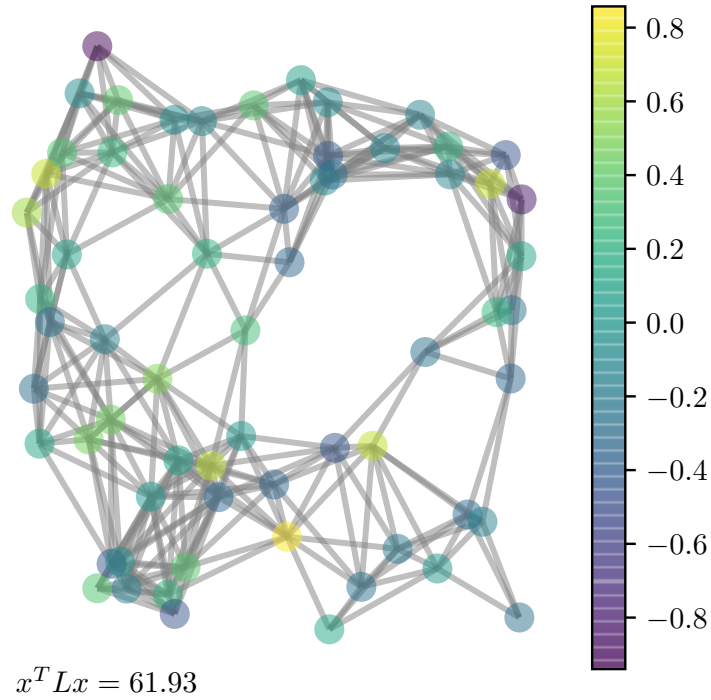
$$y = g(L)x = U g(\Lambda) U^\top x$$

Step by step

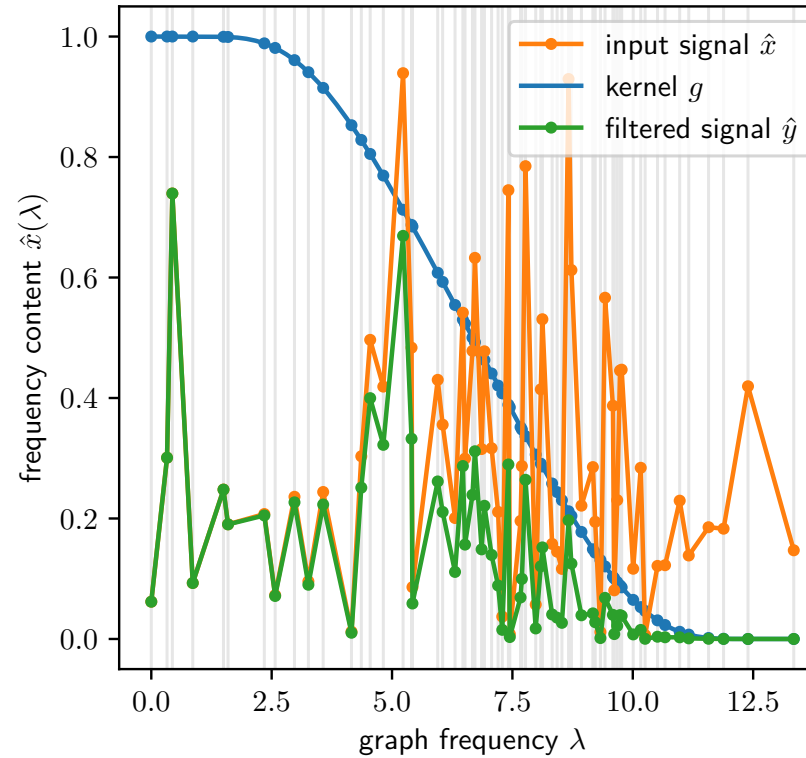
1. take the Fourier transform: $\hat{x} = U^\top x$
2. take an element-wise product with the kernel evaluated at the eigenvalues:
 $\hat{y} = (g(\lambda_1), \dots, g(\lambda_{|\mathcal{V}|})) \odot \hat{x}$
3. take the inverse Fourier transform: $y = U \hat{y}$

Example

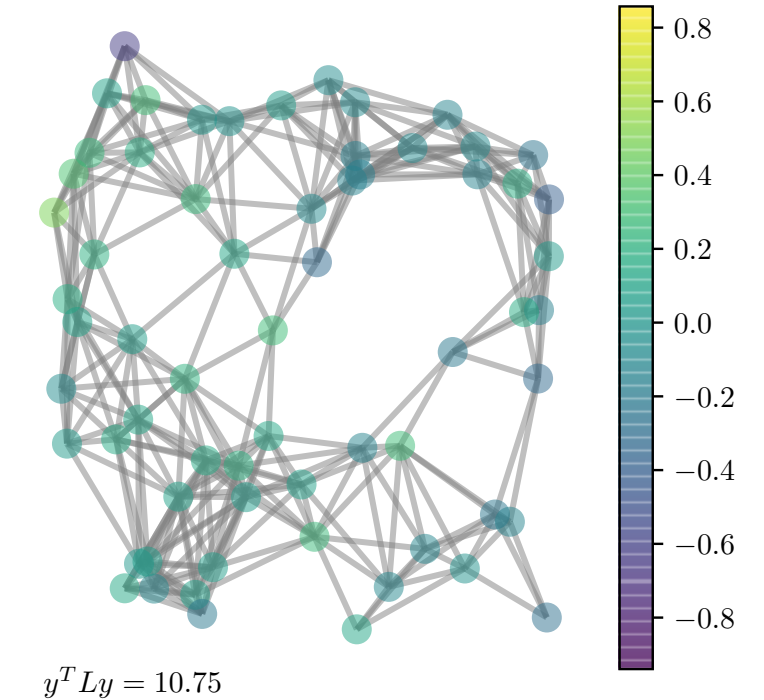
input signal x in the vertex domain



signals in the spectral domain



filtered signal y in the vertex domain



Observation: the *low-pass filtered* signal y is much smoother than x !

Convolution without translation?

1D Euclidean convolution:

$$(x * g)(i) = \sum_{j=-\infty}^{\infty} x(j)g(i-j) = \langle T_i g, x \rangle,$$

where $T_i g$ is a **translation** of the signal g by i steps.

Graph convolution:

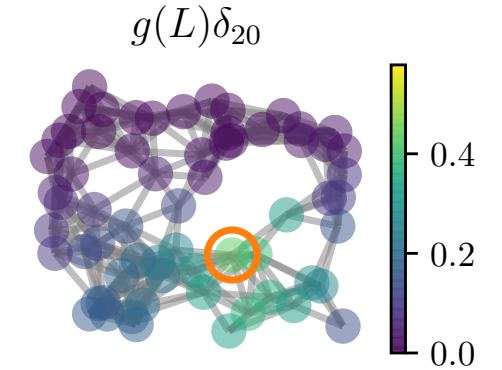
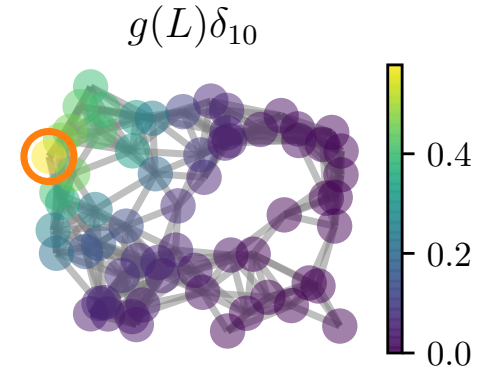
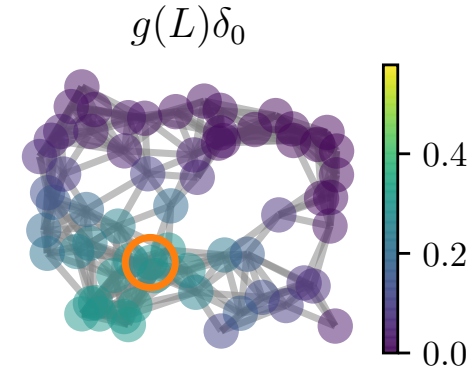
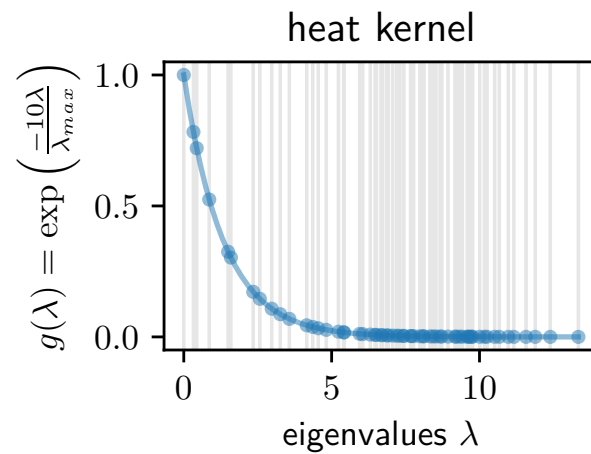
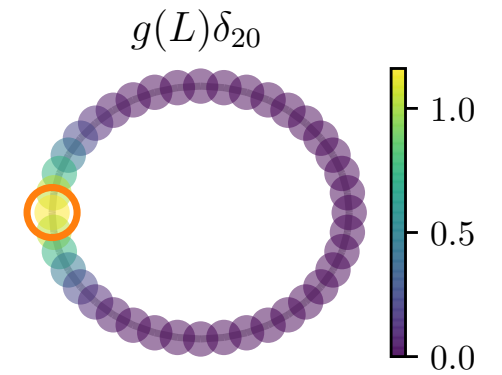
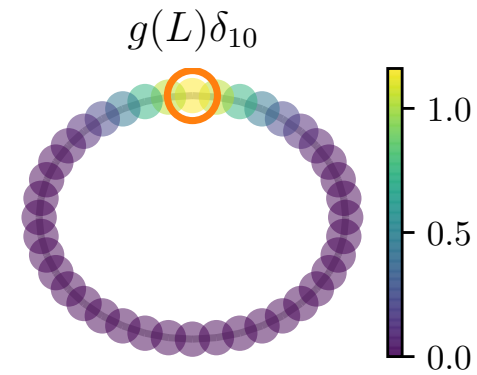
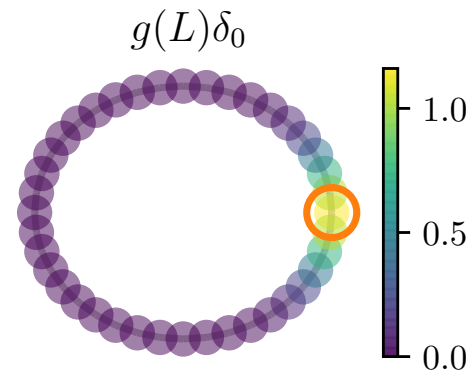
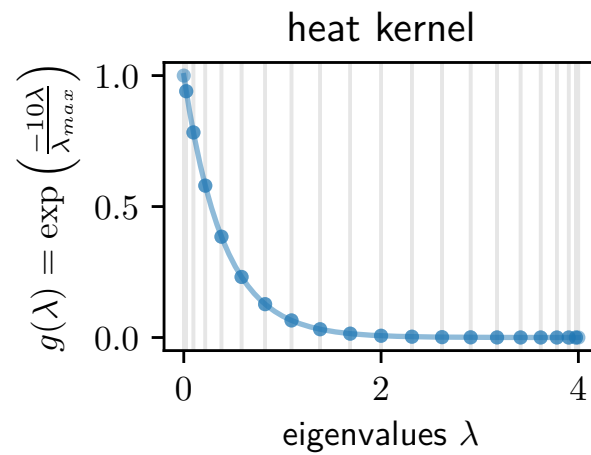
$$(x *_G g)(i) = (g(L)x)(i) = \langle \mathcal{T}_i g(L), x \rangle = \langle g(L)\delta_i, x \rangle,$$

where $\mathcal{T}_i g$ is the **localization** of the kernel g at node v_i .

We filter x with a kernel g . We cannot convolve x with another signal!

Example: localization vs translation

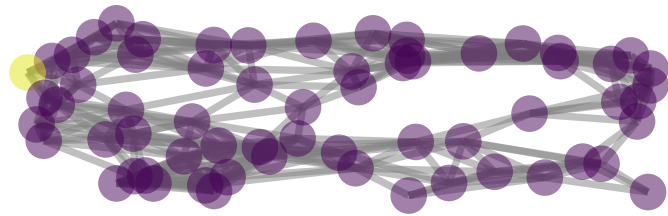
$$\mathcal{T}_i g(L) = g(L) \delta_i$$



Example: vertex domain kernel visualization

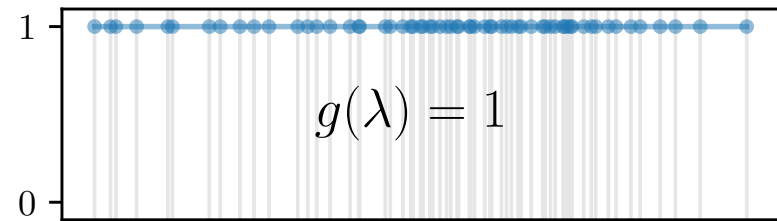
$$\mathcal{T}_i g(L) = g(L) \delta_i$$

localized $y = g(L) \delta_{10}$ (sensor)



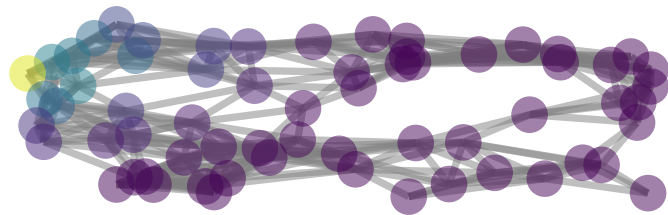
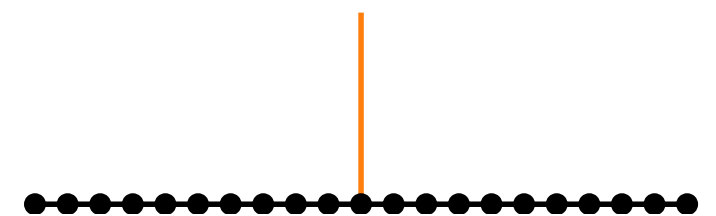
$$y^T L y = 367.18$$

kernel $g(\lambda)$ defined in the spectral domain

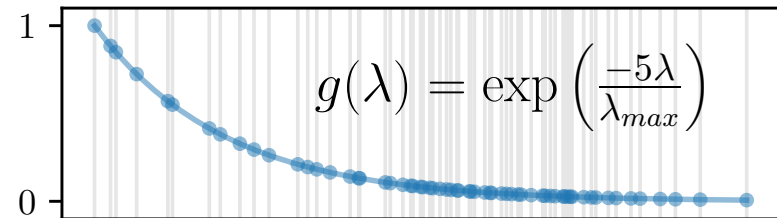


$$g(\lambda) = 1$$

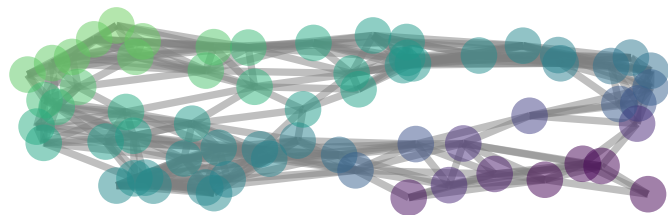
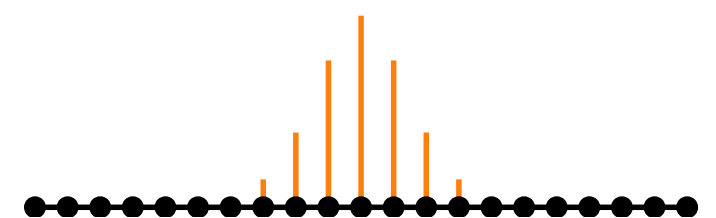
localized $y = g(L) \delta_{10}$ (path graph)



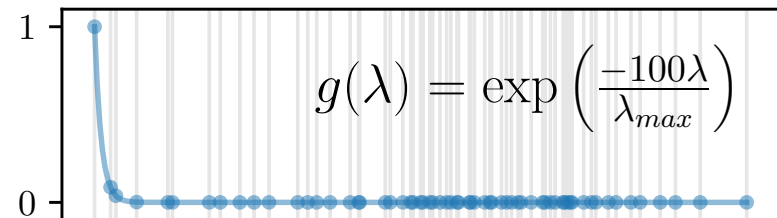
$$y^T L y = 6.83$$



$$g(\lambda) = \exp\left(\frac{-5\lambda}{\lambda_{max}}\right)$$

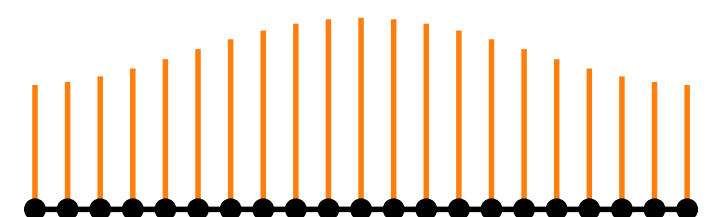


$$y^T L y = 0.00$$



$$g(\lambda) = \exp\left(\frac{-100\lambda}{\lambda_{max}}\right)$$

λ : laplacian's eigenvalues / graph frequencies



v_0 v_5 v_{10} v_{15} v_{20}

Summary so far

1. The adjacency matrix A fully describes a graph \mathcal{G} and acts as a diffusion operator.
2. The incidence matrix S acts as the gradient $S^\top x$ and divergence Sy .
The Laplacian $L = SS^\top$ is the divergence of the gradient.
3. The Laplacian $L = U\Lambda U^\top$ is diagonalized by the Fourier basis U .
4. The Fourier transform $\hat{x} = U^\top x$ shows the frequency content of the signal x .
5. L and Λ (x and \hat{x}) are the same operator (function) expressed in different bases.
6. The kernel g filters a signal x as $g(L)x$ with the operator $g(L) = Ug(\Lambda)U^\top$.
7. Kernel $g(\lambda)$ defined in the spectral domain. Localized on v_i as $\mathcal{T}_i g(L) = g(L)\delta_i$.

Filter design

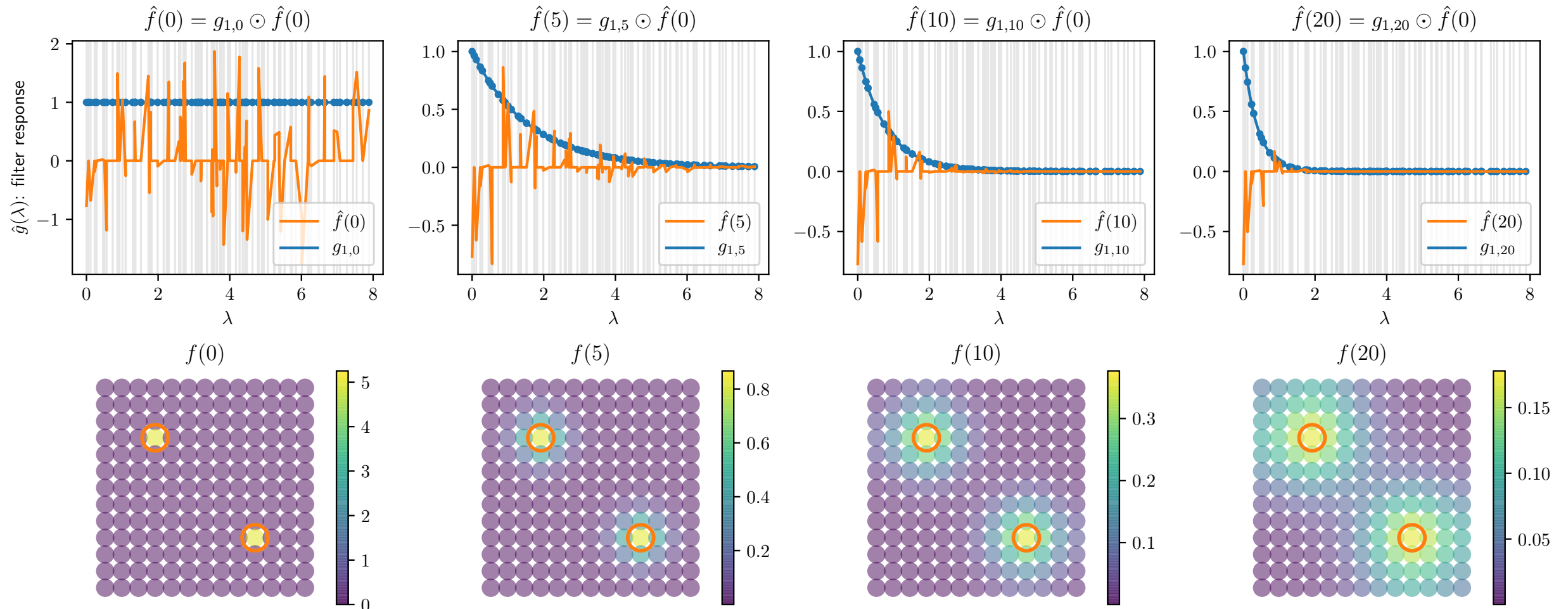
Task: design a kernel $g : \mathbb{R} \rightarrow \mathbb{R}$ such that $y = g(L)x$ is the solution of something interesting.

Examples

- ▶ Heat diffusion: $g_{\tau t}(\lambda) = \exp(-\tau t \lambda)$
- ▶ Wave propagation: $g_{\tau t}(\lambda) = \cos\left(t \arccos\left(1 - \frac{\tau^2}{2} \lambda\right)\right)$
- ▶ Projection on a subspace: $g(\lambda) = \begin{cases} 1 & \text{if } \lambda_{min} < \lambda < \lambda_{max}, \\ 0 & \text{otherwise.} \end{cases}$
- ▶ Denoising with $\arg \min_y \|y - x\|_2^2 + \tau y^\top L y$: $g(\lambda) = \frac{1}{1 + \tau \lambda}$

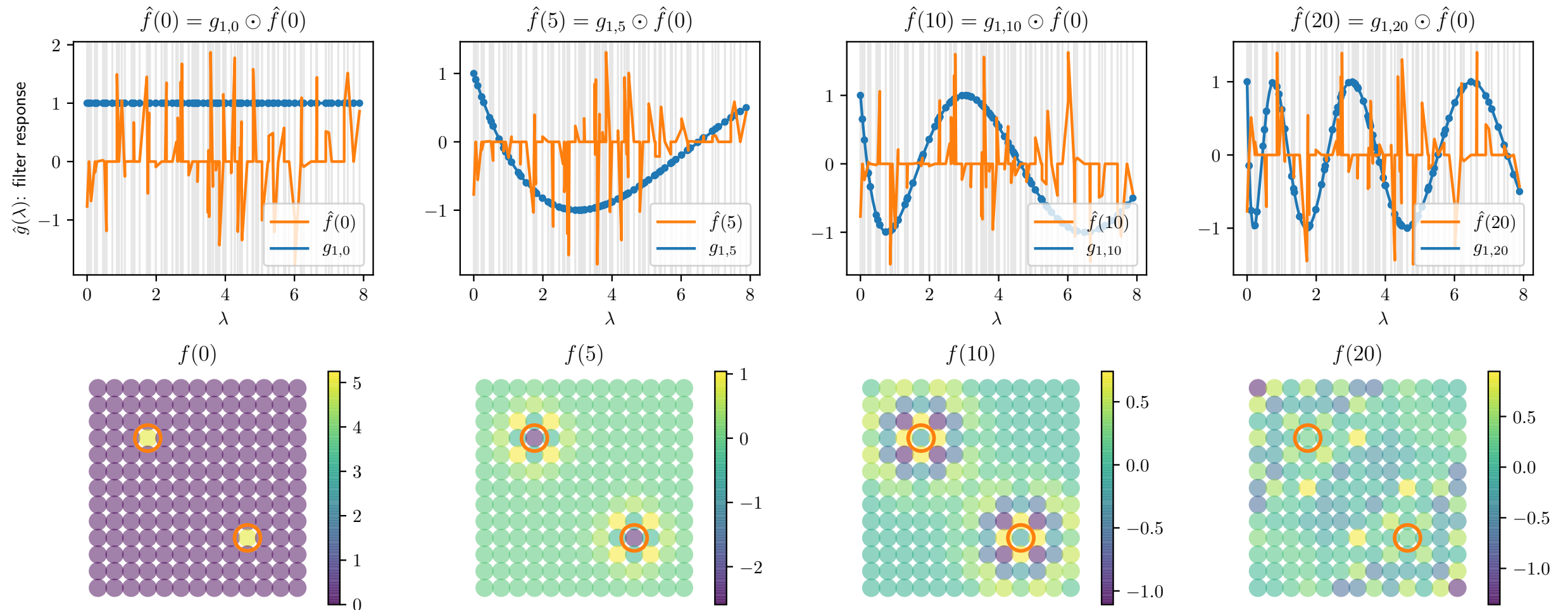
Example: heat diffusion

$$-\tau L f(t) = \partial_t f(t) \quad \Rightarrow \quad f(t) = g_{\tau t}(L) f(0) \text{ with } g_{\tau t}(\lambda) = \exp(-\tau t \lambda)$$



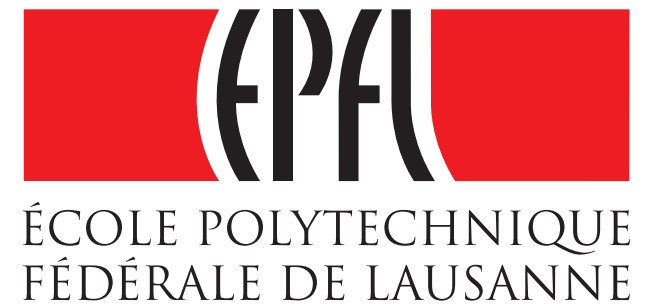
Example: wave propagation

$$-\tau^2 L f(t) = \partial_{tt} f(t) \quad \Rightarrow \quad f(t) = g_{\tau t}(L) f(0) \text{ with } g_{\tau t}(\lambda) = \cos \left(t \arccos \left(1 - \frac{\tau^2}{2} \lambda \right) \right)$$

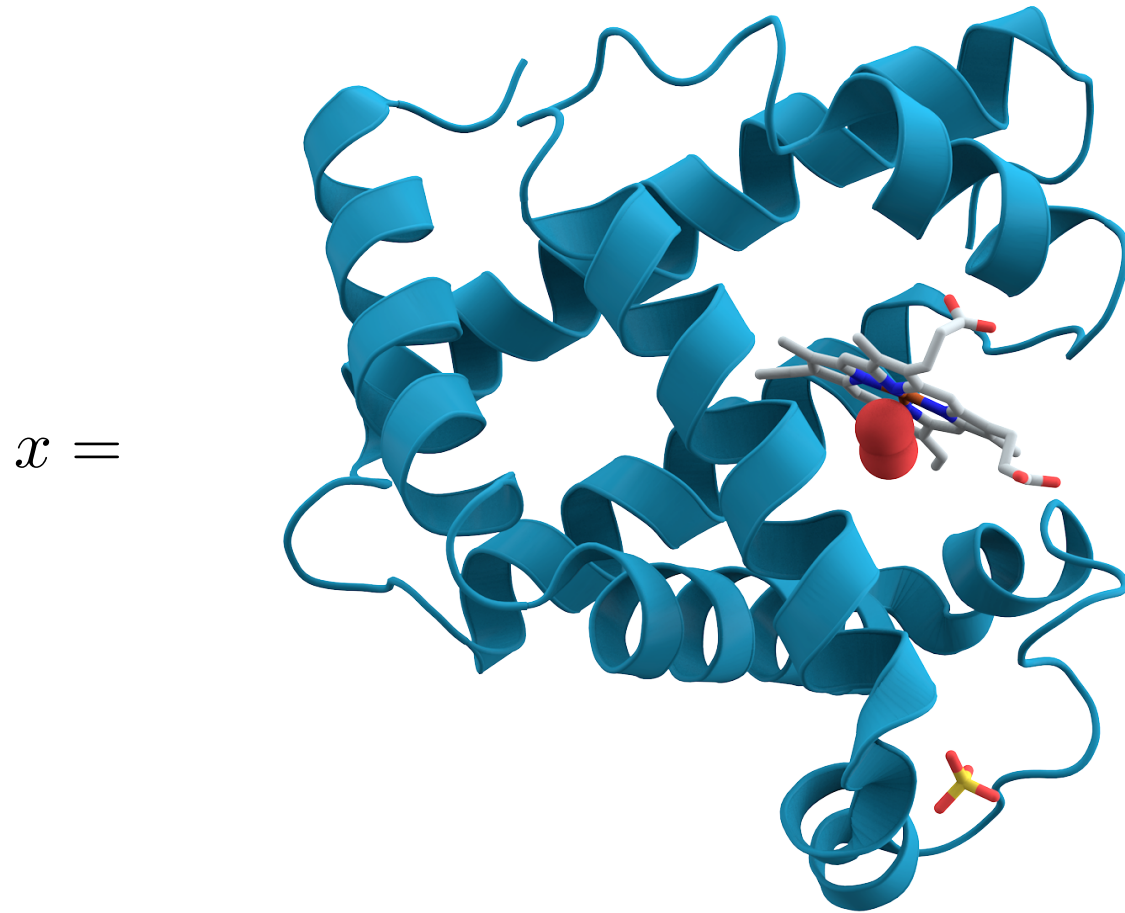


LEARNING ON GRAPHS

Michaël DEFFERRARD



Motivation



$$y = f(x) = \begin{cases} \text{"toxic"} \\ \text{"non-toxic"} \end{cases}$$

$$y = f(x) = 80\% \text{ toxic}$$

Goal: learn the **unknown** function f , using both **structure** and **features**.

Filter design: design a kernel g such that $y = g(L)x$ is the solution of something interesting. But what if we don't know the process by which y depends on x , and can't derive g ? Answer: learn the kernel from examples.

Task: approximate the optimal unknown mapping $y = g(L)x$ by a parameterized approximation $y \approx \tilde{y} = g_\theta(L)x$, where θ are the parameters to be learned.

We got:

- ▶ a set of examples $\{(x_n, y_n)\}_{n=1}^N$, hopefully large enough
- ▶ a cost function to measure how good our approximation is, for example $c(\tilde{y}, y) = \|\tilde{y} - y\|_2^2$

The goal is to minimize the expected cost $\mathbf{E}_{(x,y)}[c(g_\theta(L)x, y)]$.

The expectation cannot be computed as the distribution $P(x, y)$ is unknown. However, we can compute the empirical risk, an approximation that is the average cost over our training data: $R(g_\theta) = \frac{1}{N} \sum_n c(g_\theta(L)x_n, y_n)$.

Solution: $\hat{\theta} = \arg \min_{\theta} R(g_\theta)$

Training

How to find $\hat{\theta} = \arg \min_{\theta} R(g_{\theta})$?

A popular optimization algorithm is (stochastic) gradient descent, an iterative algorithm that updates the parameters as

$$\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} c(g_{\theta}(L)x_i, y_i)$$

upon seeing the example (x_i, y_i) .

All the computations must be differentiable w.r.t. θ !

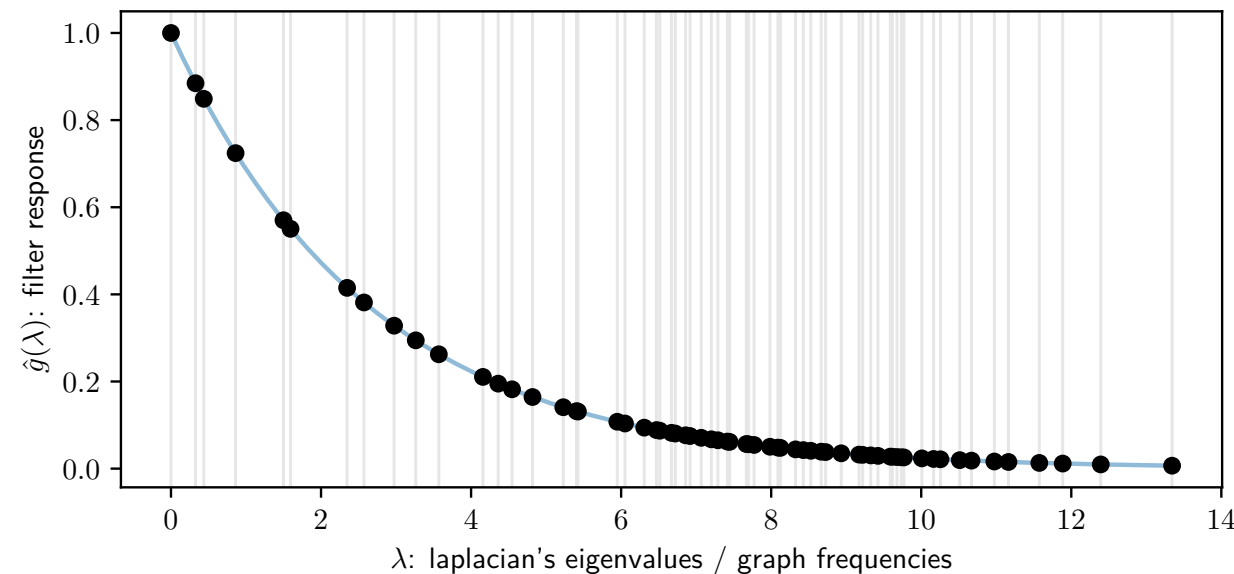
In practice, gradients are computed through back-propagation.

Kernel parameterization

Defferrard, Bresson, and Vandergheynst 2016

Non-parametric filter, can learn any filter (n degrees of freedom):

$$g_{\theta}(\Lambda) = \text{diag}(\theta), \quad \theta \in \mathbb{R}^n \Rightarrow y = U \text{diag}(\theta) U^{\top} x$$



- ▶ Learning complexity is $\mathcal{O}(n)$
- ▶ Computational complexity is $\mathcal{O}(n^2)$ (& memory)
- ▶ Non-localized in vertex domain

Polynomial parametrization

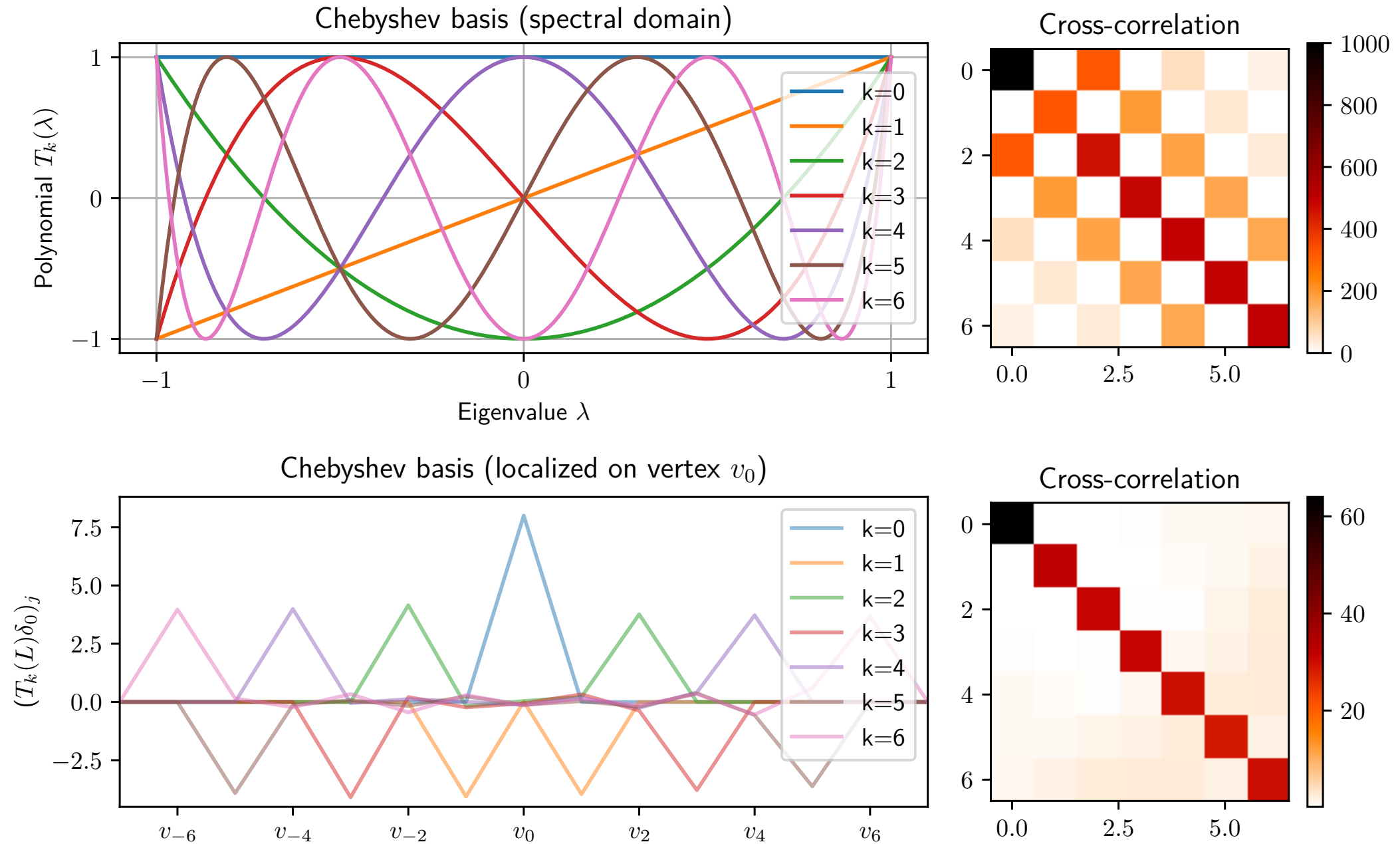
Defferrard, Bresson, and Vandergheynst 2016

$$g_{\theta}(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k = \sum_{k=0}^{K-1} \tilde{\theta}_k T_k(\tilde{\Lambda}), \quad \tilde{\Lambda} = \frac{2}{\lambda_n} \Lambda - I_n$$

Chebyshev polynomials: $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$
with $T_0 = 1$ and $T_1 = x$

- ▶ Can learn any K -localized filter.
- ▶ Allows a distributed implementation: only accesses the K -neighborhood.
- ▶ K -localized
- ▶ Learning complexity is $\mathcal{O}(K)$
- ▶ Computational complexity is $\mathcal{O}(K|\mathcal{E}|)$ (same as classical ConvNets!)

Chebyshev polynomials



Fast implementation by recursion

Defferrard, Bresson, and Vandergheynst 2016

$$y = g_{\theta}(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x = \sum_{k=0}^{K-1} \theta_k \bar{x}_k, \quad \tilde{L} = \frac{2}{\lambda_n} L - I_n$$

Recurrence: $\bar{x}_0 = x$

$$\bar{x}_1 = \tilde{L}x$$

$$\bar{x}_k = T_k(\tilde{L})x = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$$

- ▶ Any polynomial can be used. They all have the same representative power. Optimization difficulty might vary.
- ▶ Any matrix can be used instead of the Laplacian L , including the adjacency matrix, or even a non-symmetric adjacency or “Laplacian”.
- ▶ The learned filter parameters θ can be transferred across graphs, i.e., used with different L .

Spatial vs Spectral

Defferrard, Bresson, and Vandergheynst 2016

Convolution on graphs can be **spectrally motivated**.

$$y = U g_{\theta}(\Lambda) U^{\top} x$$

In the absence of an $O(n \log n)$ Fast Fourier Transform (FFT), which only exists for specific domains, that is however too expensive. $\mathcal{O}(n^3)$ operations for the EVD, plus $\mathcal{O}(n^2)$ operations per forward and backward pass.

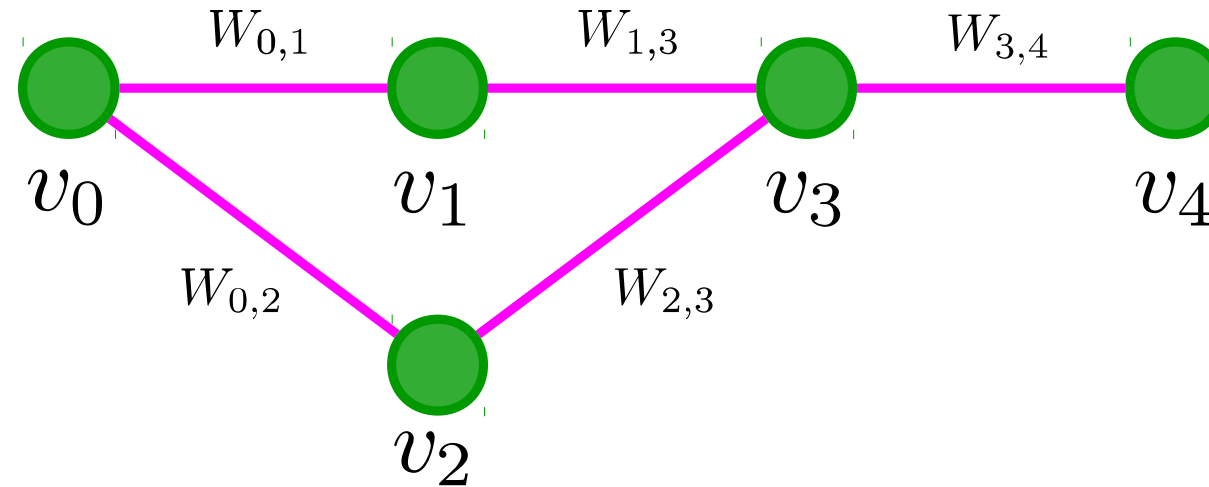
With polynomials, the convolution is however **spatially implemented**.

$$y = g_{\theta}(L)x = \sum_k \theta_k L^k x = \sum_k \tilde{\theta}_k T_k(\tilde{L})x$$

Leading to many other interpretations: message-passing between nodes, local tangent planes, permutation invariant aggregation, etc.

Weights of paths

$(W^k)_{ij}$ is the sum of all weighted paths of length k between v_i and v_j .

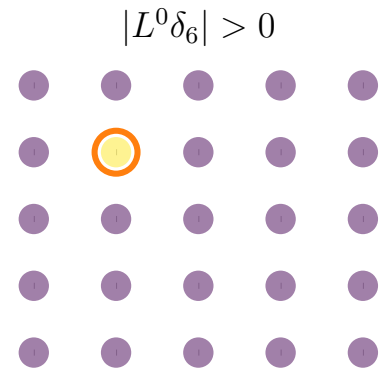
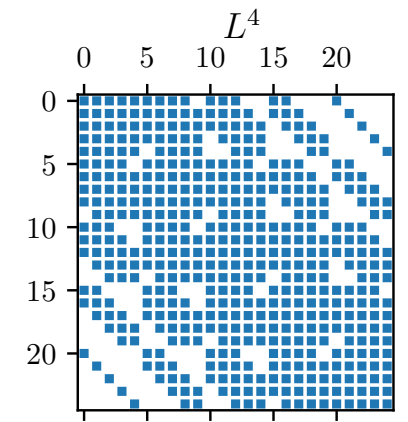
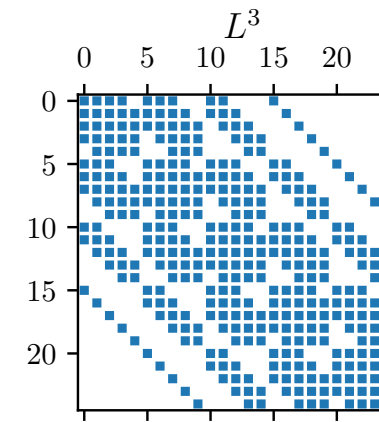
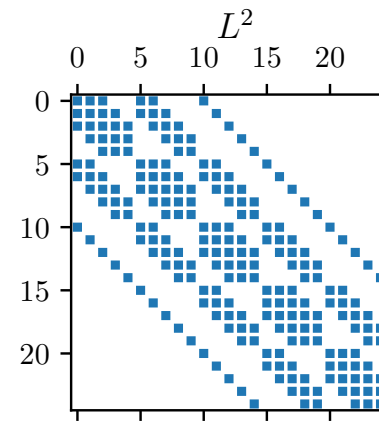
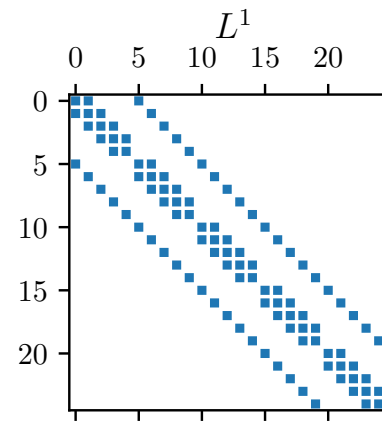
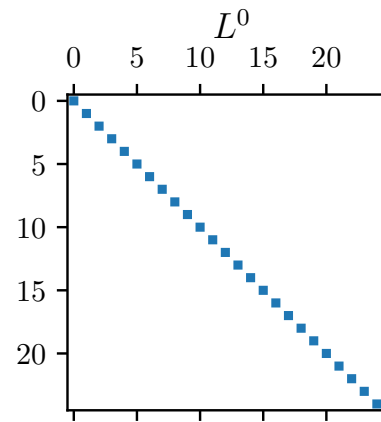


- ▶ A path is an ordered set of nodes. Example: (v_2, v_3, v_4) .
- ▶ $p_{ij}^k = \{(v_i, \dots, v_j), \dots, (v_i, \dots, v_j)\}$ is the set of all paths of length k between v_i and v_j . Example: $p_{0,3}^2 = \{(v_0, v_1, v_3), (v_0, v_2, v_3)\}$.
- ▶ Path weight $(W^k)_{ij} = \text{weight}(p_{ij}^k) = \sum_{\text{paths}} \prod_{\text{edges } (v_k, v_l)} W_{kl}$.
Example: $(W^2)_{0,3} = (W_{0,1} \cdot W_{1,3}) + (W_{0,2} \cdot W_{2,3})$.

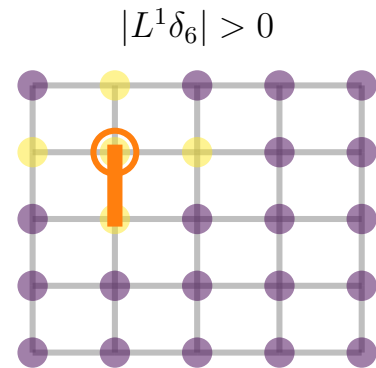
Neighborhoods

L^k defines the k -neighborhood

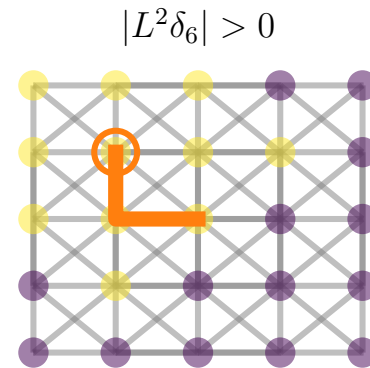
Localization: $d_G(v_i, v_j) > K$ implies $(L^K)_{ij} = 0$



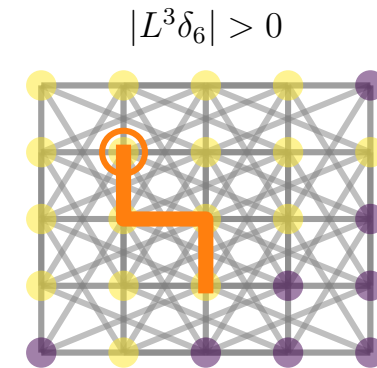
$||W^0||_0 = 0$ edges



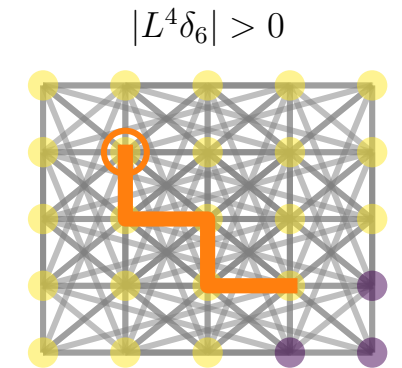
$||W^1||_0 = 40$ edges



$||W^2||_0 = 62$ edges



$||W^3||_0 = 108$ edges



$||W^4||_0 = 122$ edges

Learned combination of neighboring values

$y = \sum_k \theta_k L^k x$ is a linear transformation, where the coefficients are:

- ▶ the learned parameter θ_k ,
- ▶ the k -neighborhood encoded by L^k .

Weighted sum of neighborhoods:

$$y_i = \sum_k \theta_k \bar{x}_k = \underbrace{\theta_0 x}_{\text{own value}} + \underbrace{\theta_1 \bar{x}_1}_{\text{1-neighborhood}} + \underbrace{\theta_2 \bar{x}_2}_{\text{2-neighborhood}} + \cdots + \underbrace{\theta_K \bar{x}_K}_{K\text{-neighborhood}}$$

- ▶ Monomials in L : $\bar{x}_k = L^k x$
- ▶ Monomials in A : $\bar{x}_k = A^k x$
- ▶ Chebyshev polynomials in L : $\bar{x}_0 = x, \bar{x}_1 = \tilde{L}x, \bar{x}_k = T_k(\tilde{L})x = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$

Aggregation function

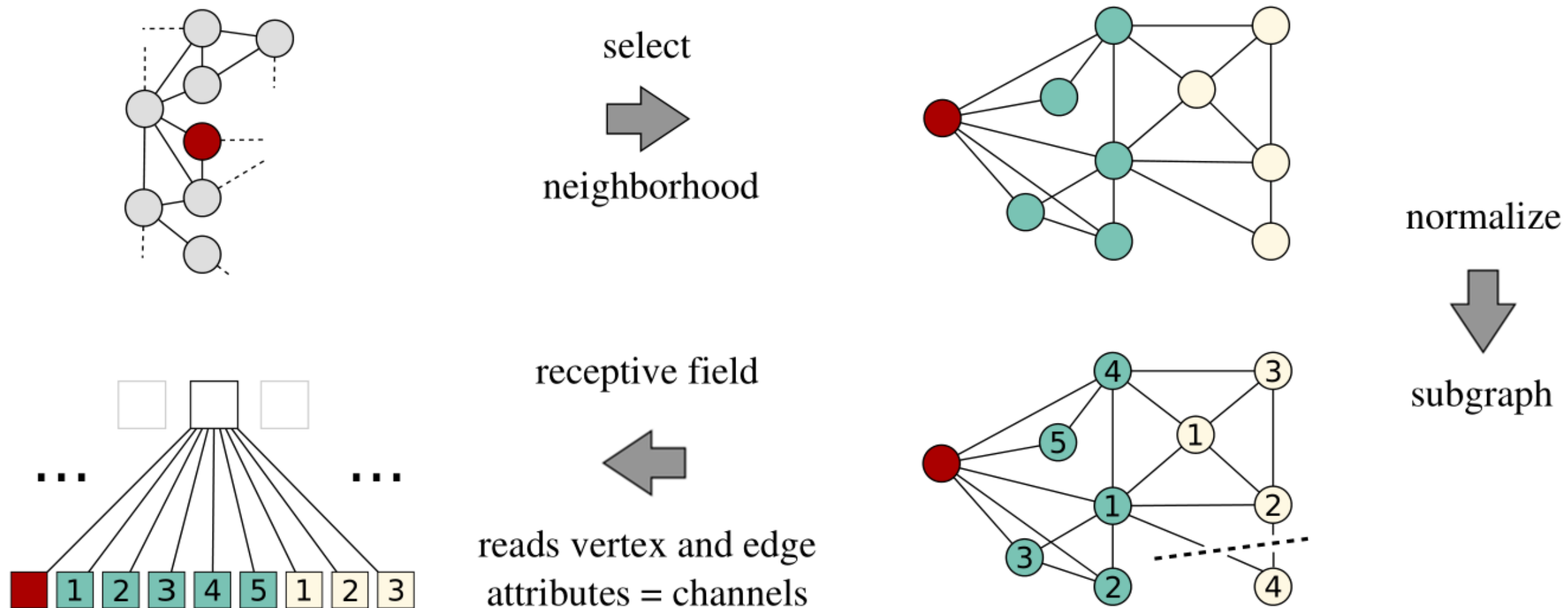
$y = f(x) = \sum_k \bar{x}_k$ is learning how to combine the values \bar{x}_k from the k -neighborhood.
The *basic unit* is the neighborhoods, not the nodes.

What else can be done? Any function f that is invariant to the number of neighbors and their permutation.

Goal: map a varying-length representation to a length K representation for $\mathcal{O}(K)$ learning complexity.

Spatial approach: node ordering

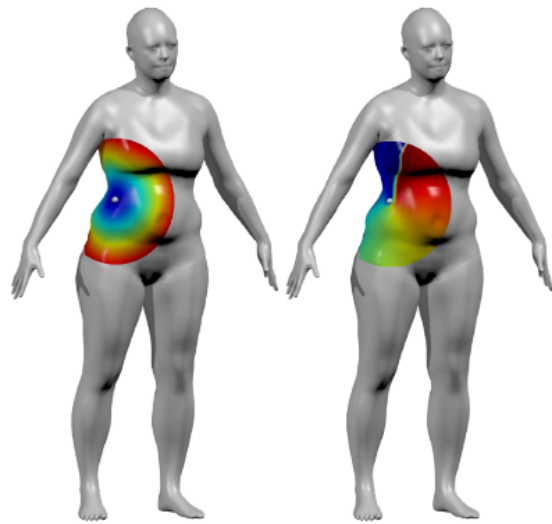
Niepert, Ahmed, and Kutzkov 2016



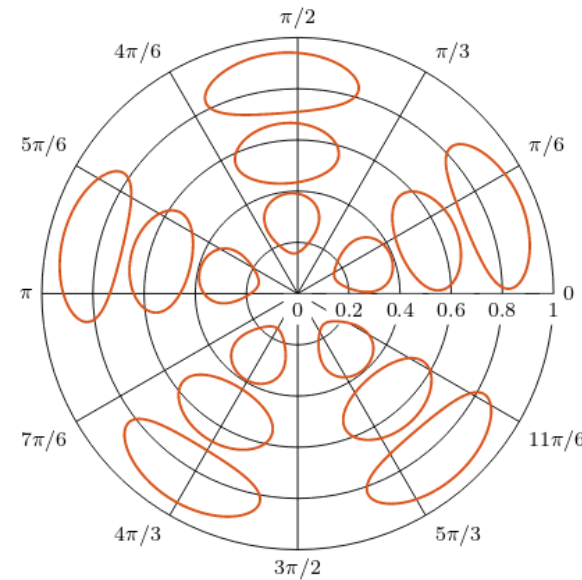
- anisotropic filters
- require an ordering of the nodes

Spatial approach: patches on the manifold's tangent plane

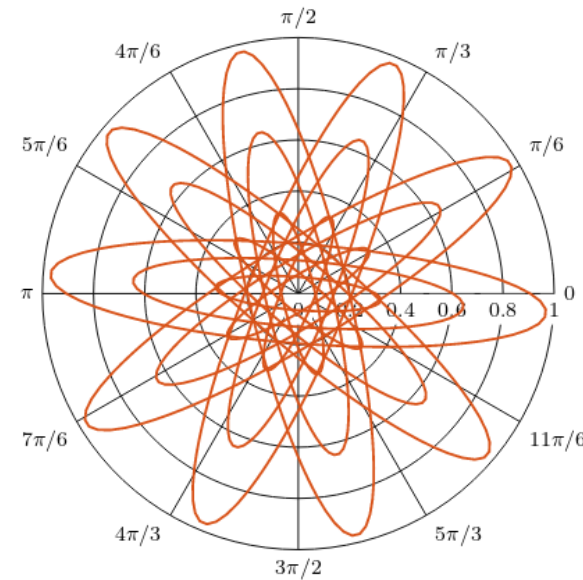
Monti, Boscaini, Masci, Rodola, Svoboda, and Bronstein 2017



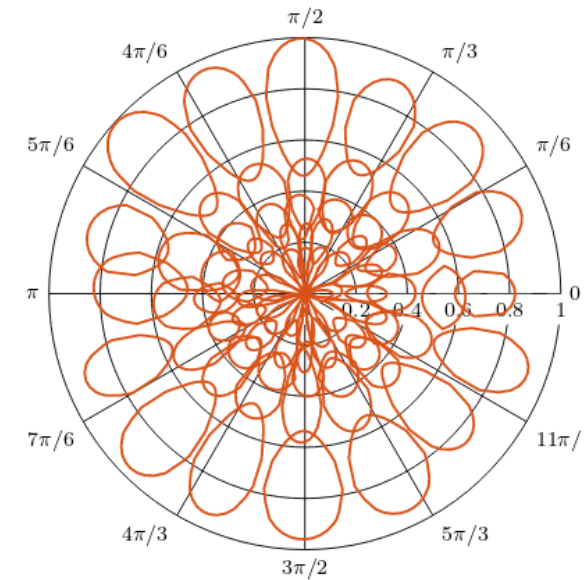
Polar coordinates ρ, θ



GCNN



ACNN



MoNet

- anisotropic filters
- manifolds only

The need to consider multiple scales

Most data on large graphs exhibit **patterns at multiple scales**.

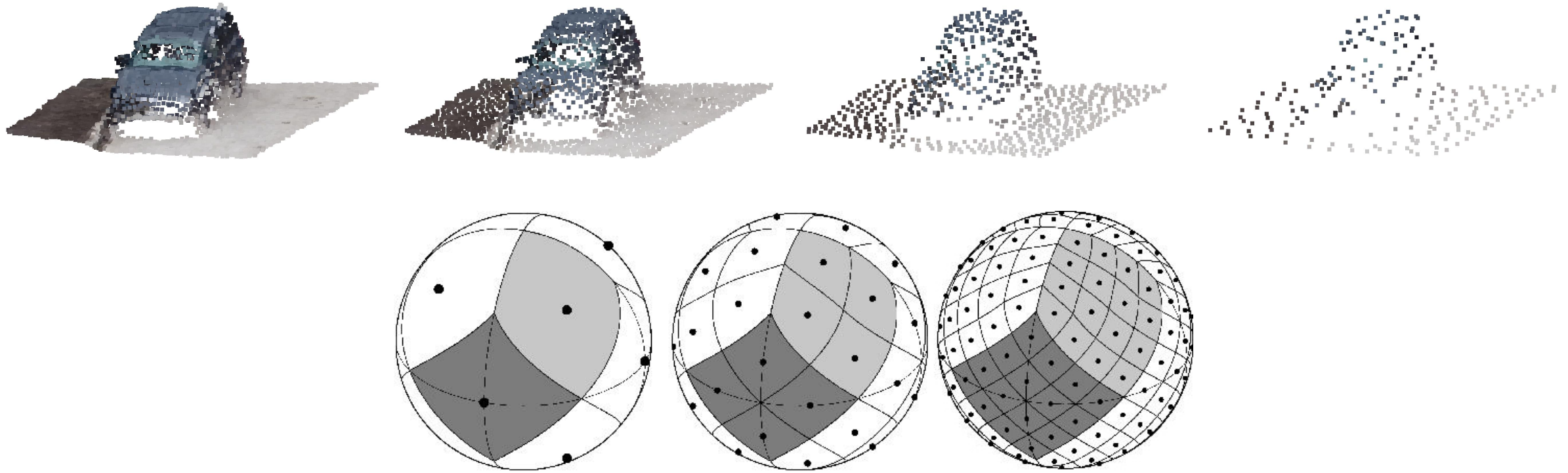
Some filters thus need to have larger receptive fields to capture longer-range dependencies. This can be achieved by:

1. increasing the size of the filters (the polynomial order),
2. increasing the number of layers,
3. down-sampling the domain (pooling).

While we can easily do (1) and (2), it can drastically increase the number of parameters to learn. For now, we don't yet have a generic and functional approach to (3).

Coarsening: hierarchical representation

Graph coarsening is certainly an answer to the down-sampling problem.

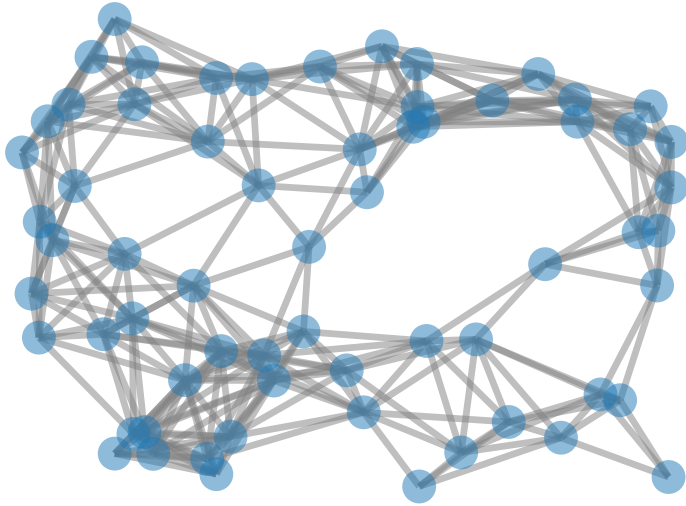


- Easy and well-defined when the domain has a hierarchical structure.

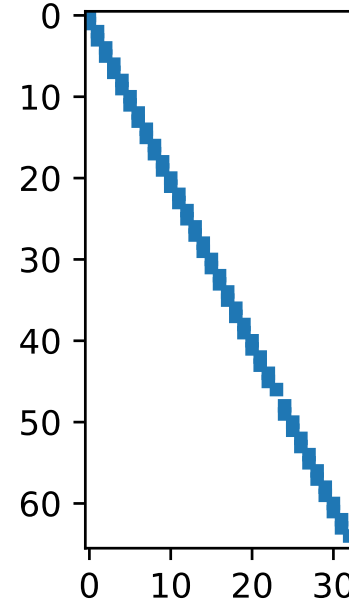
Coarsening: greedy local approach

Defferrard, Bresson, and Vandergheynst 2016

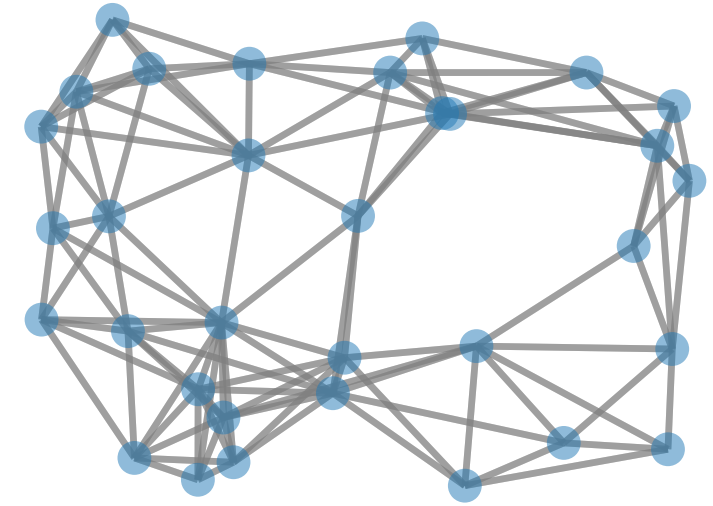
Input graph: $|V| = 64$, $|E| = 303$



Coarsening matrix $C \in \mathbb{R}^{66 \times 33}$



Coarsened graph: $|V| = 33$, $|E| = 230$

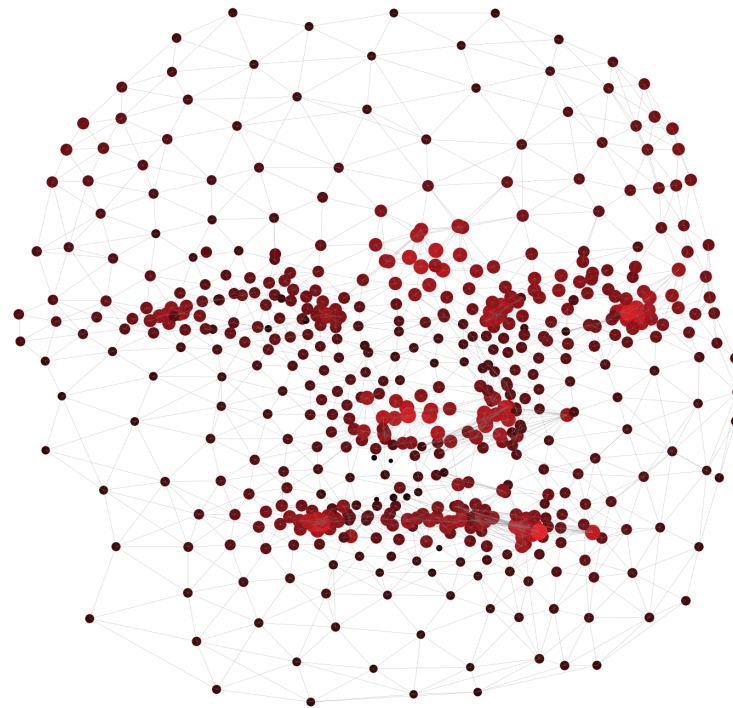
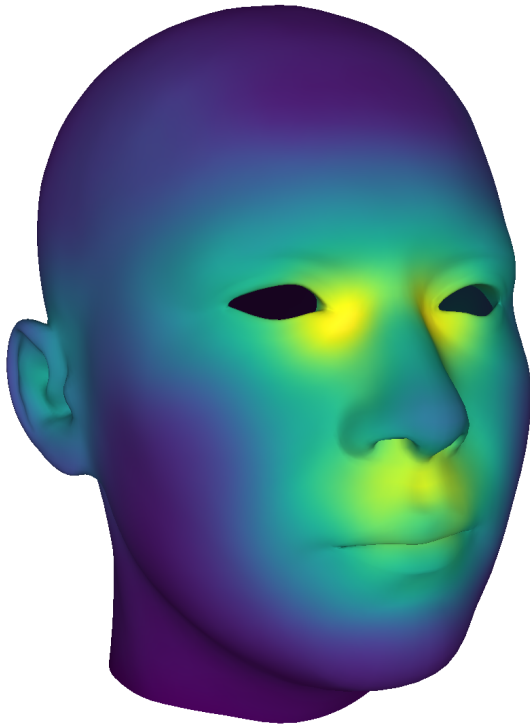


- ▶ Greedy node merging (e.g., Graclus, Metis) works well for regular graphs.
- ▶ Can be done as pre-processing.
- ▶ Conditioned on the structure only.
- ▶ Much harder on non-regular graphs.

Learned coarsening: an attention mechanism

Defferrard and Loukas 2018

hard combinatorial problem \Rightarrow learn a **continuous relaxation** of the operation



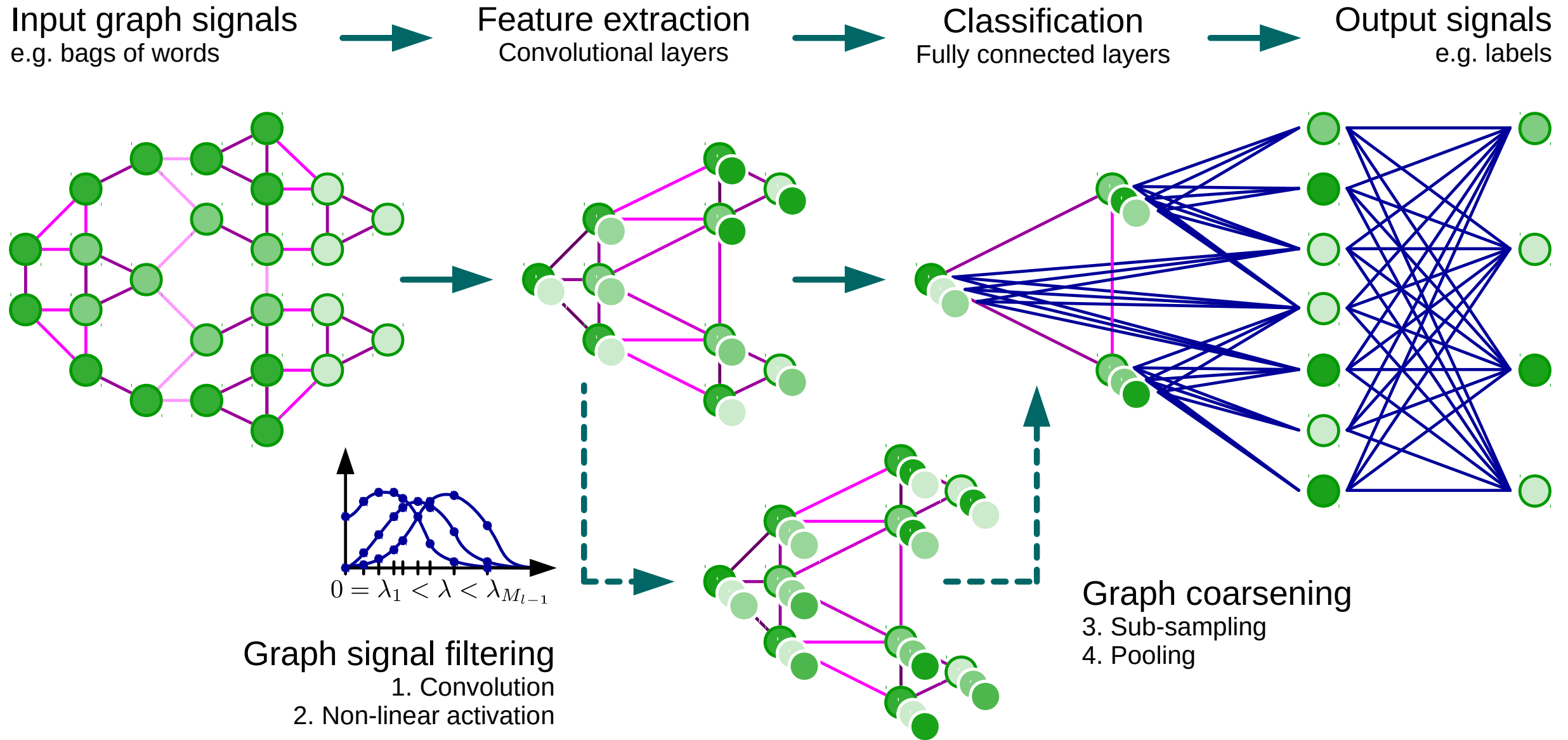
Conditioned on:

1. the structure
2. the features
3. the task

introspection!

Graph ConvNet architecture

Defferrard, Bresson, and Vandergheynst 2016



Multiple kinds of problems: combination of data and tasks

Graphs that model discrete relations

- ▶ Social networks
- ▶ Graph of citations or hyperlinks
- ▶ Molecules (proteins)
- ▶ Knowledge graphs

Graphs that represent sampled manifolds

- ▶ Meshes (shapes, surfaces)
- ▶ Point clouds
- ▶ Data on spheres (planets, sky)
- ▶ Traffic on roads

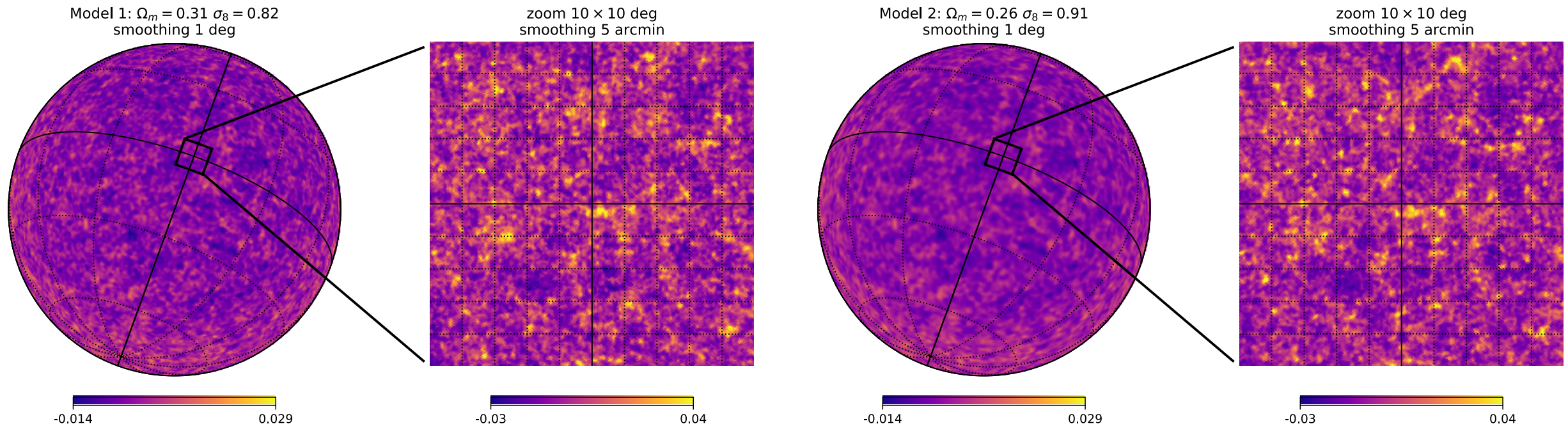
Tasks:

- ▶ Node classification or regression (semi-supervised learning)
- ▶ Graph classification or regression
- ▶ Signal classification or regression

Cosmological application: data & problem

Perraudin, Defferrard, Kacprzak, and Sgier 2018

- ▶ Cosmologists devise models of how the universe works.
- ▶ We only get to observe one real universe.
- ▶ Problem: which simulation is closest to the real thing? A signal classification task.

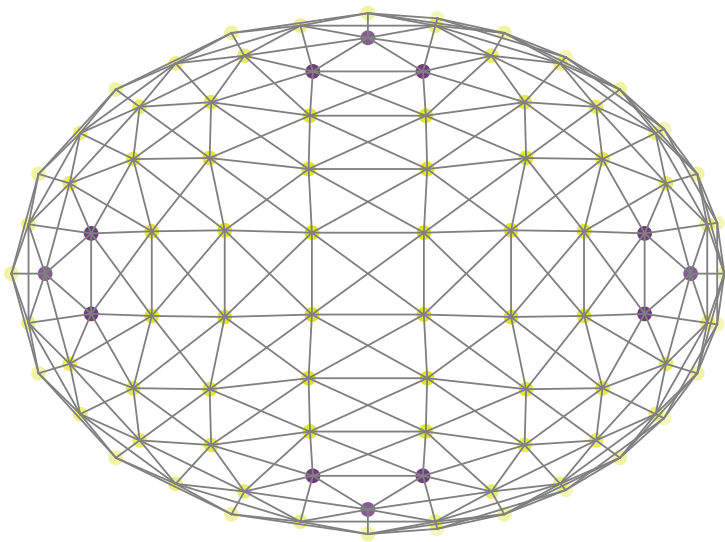


Two mass maps generated from different cosmological parameters.

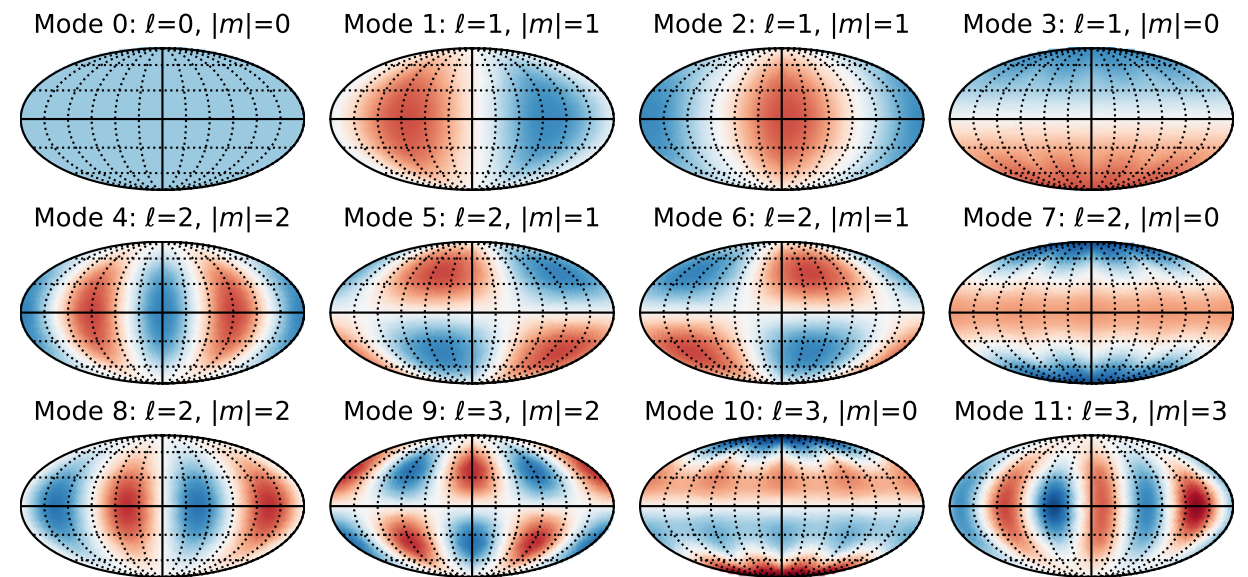
Cosmology: graph

Perraudin, Defferrard, Kacprzak, and Sgier 2018

- ▶ Data lives on the sky, a sphere.
- ▶ The sphere is discretized, and can be represented by a graph.
- ▶ Numerous kind of spherical sky maps in cosmology and astrophysics.
Cosmic microwave background, galaxy clustering, gravitational lensing.



Sphere discretized by graph.

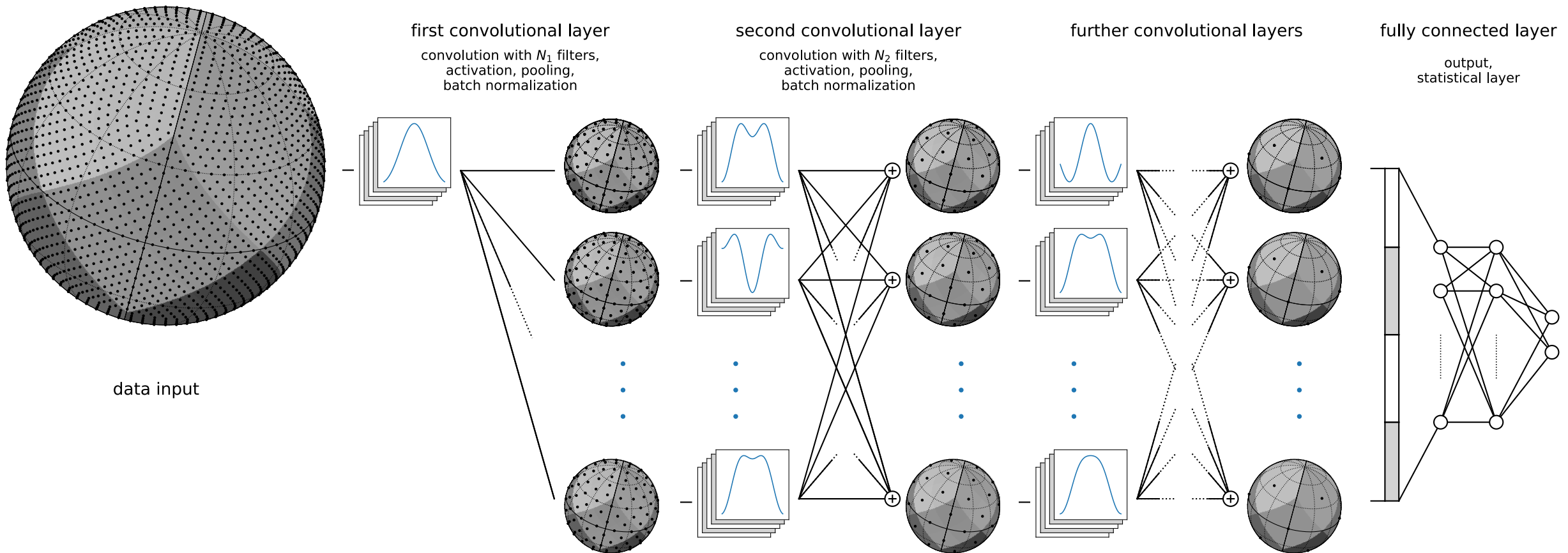


Fourier modes resemble spherical harmonics.

Cosmology: model

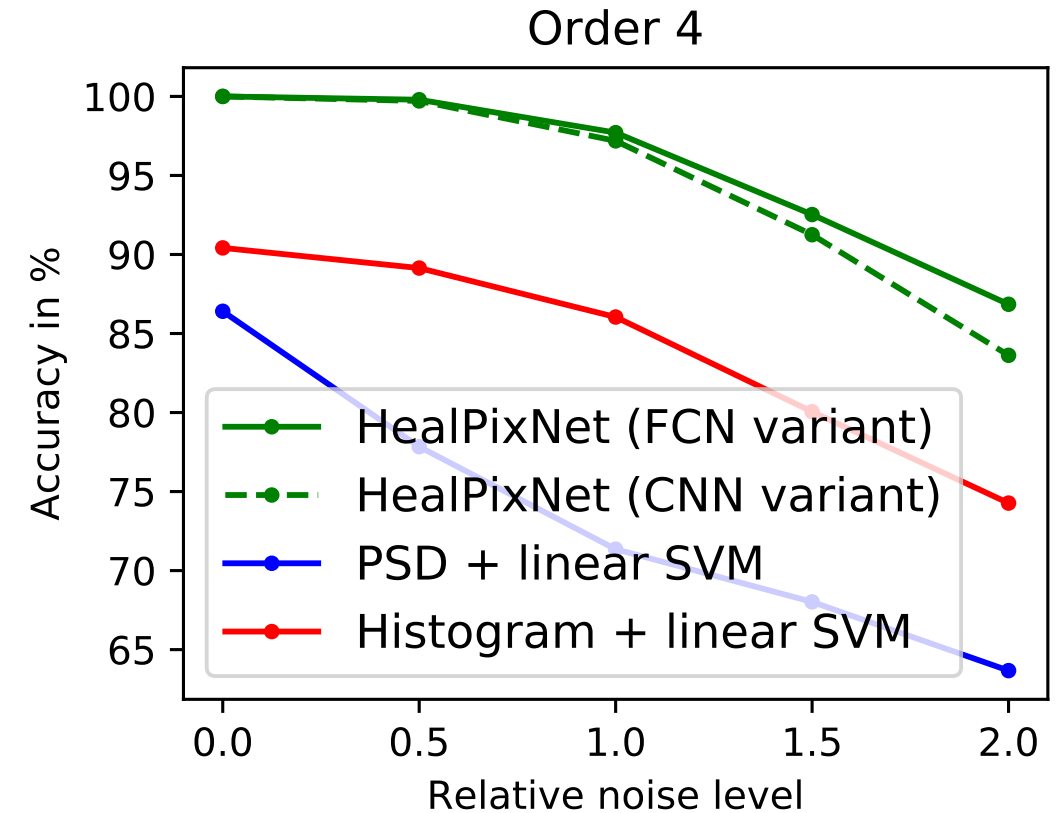
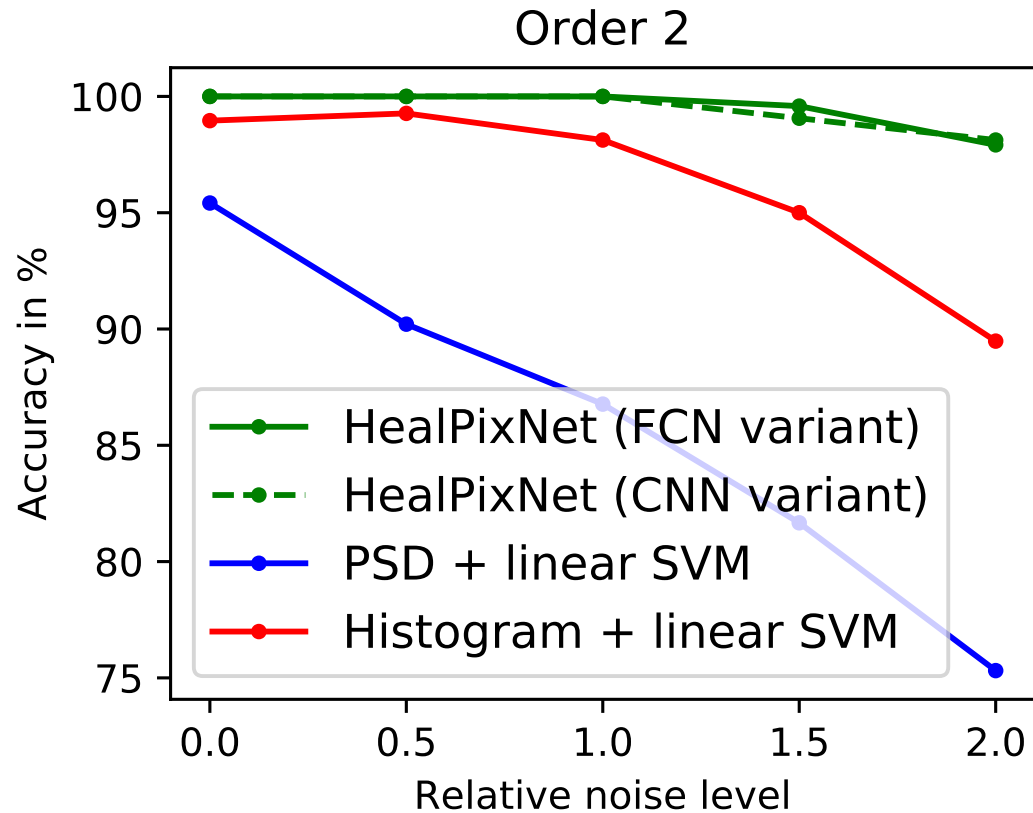
Perraudin, Defferrard, Kacprzak, and Sgier 2018

A classical CNN or FCN architecture, but on the sphere, which is modeled by a graph.



Cosmology: results

Perraudin, Defferrard, Kacprzak, and Sgier 2018

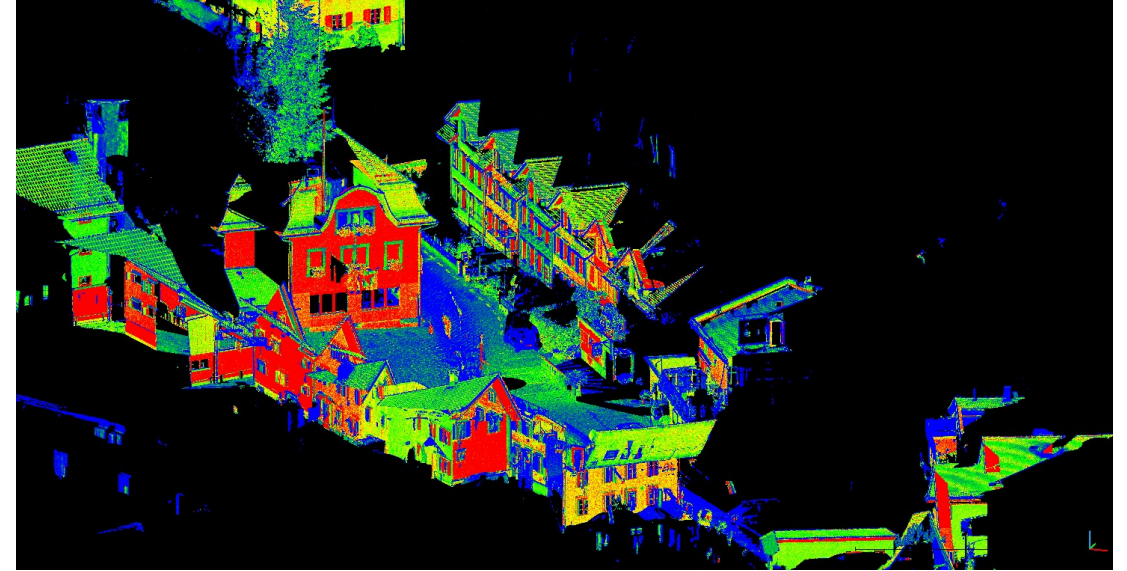


Significantly better than two standard benchmarks used in cosmology.

Segmentation of point clouds



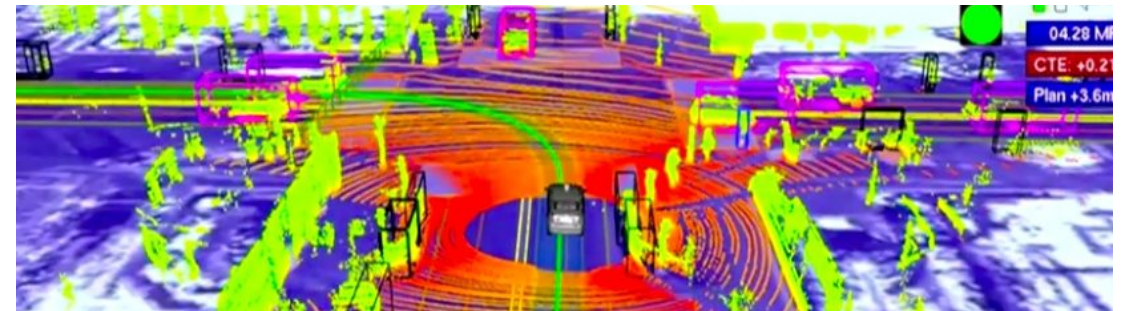
remote sensing / surveying



outdoor mapping



indoor mapping



autonomous driving

Different classification problems

Goal: assign class labels.

- ▶ granularity
- ▶ class vs instance



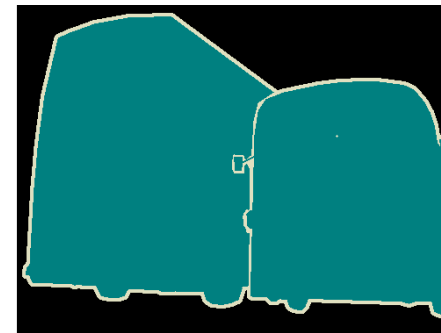
input¹



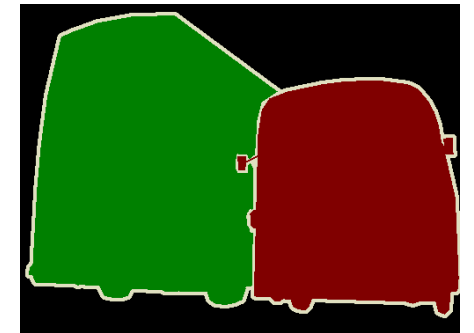
classification



object recognition



semantic seg.



instance seg.

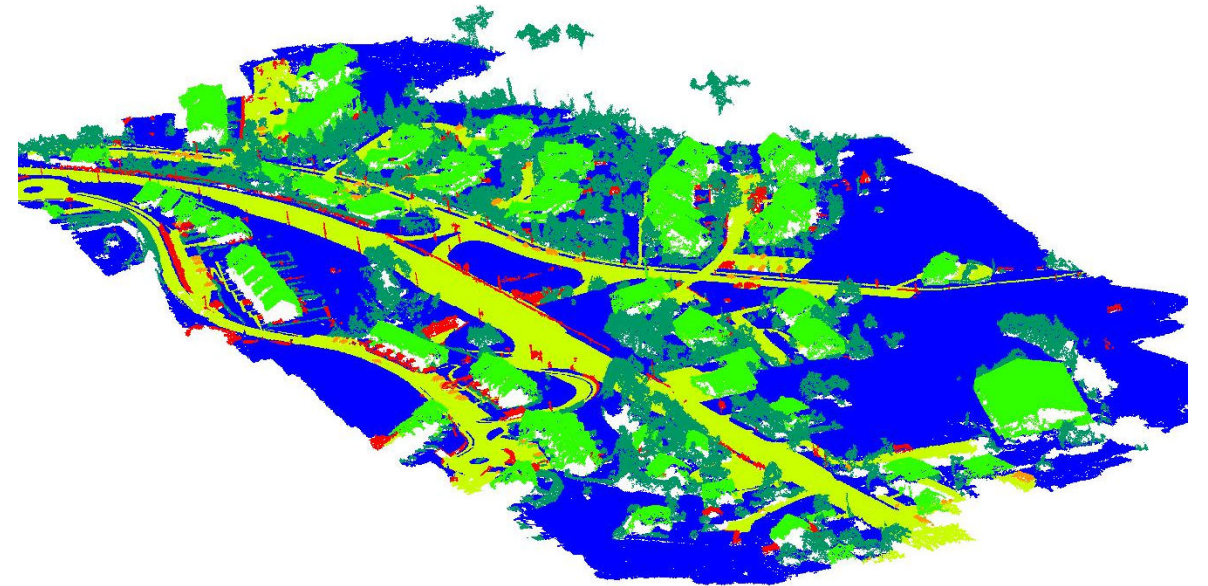
¹ Image source: https://sthalles.github.io/assets/deep_segmentation_network/object_class_segmentation.png

Data

input a set of features associated to a set of points
output a label associated to each point

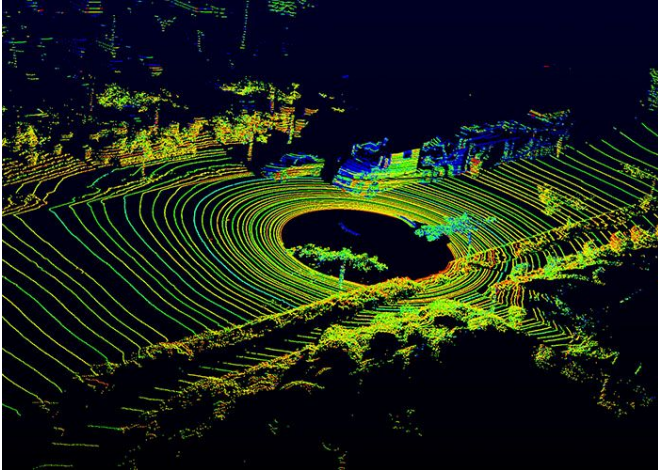


x,y,z coordinates with RGB colors

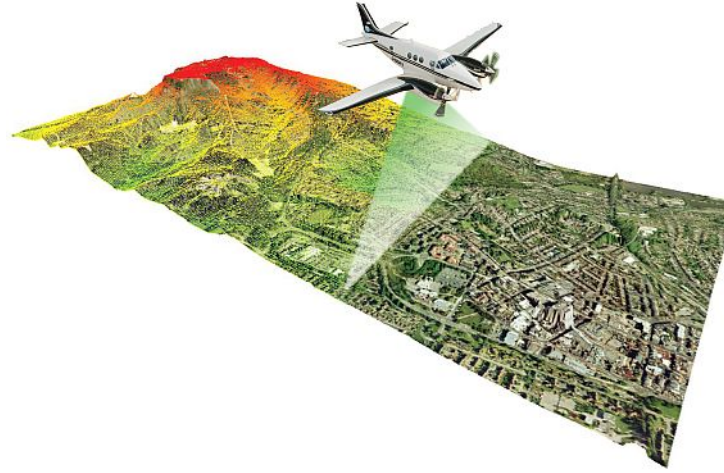


class labels

Data acquisition



ground LIDAR



aerial LIDAR



aerial images

Our case, aerial images:

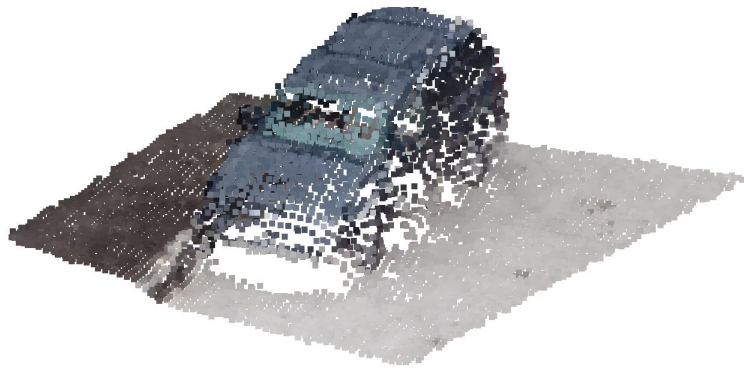
- ▶ Drones take aerial pictures of the ground.
- ▶ Each point is photographed multiple times from different point-of-views.
- ▶ Point cloud constructed by photogrammetry.

Graph

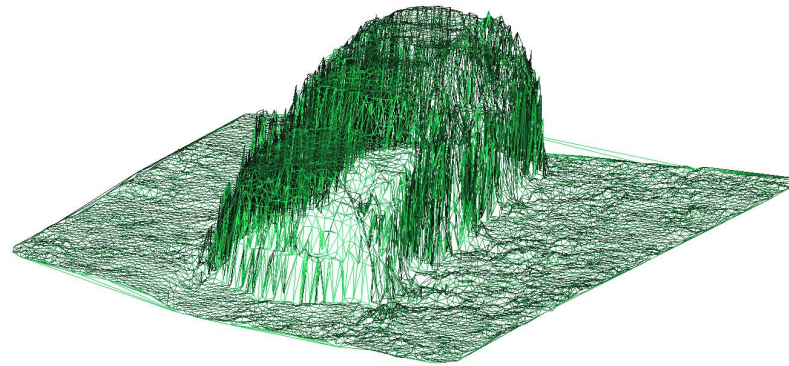
Cherqui, Morsier, and Defferrard 2018

A graph gives:

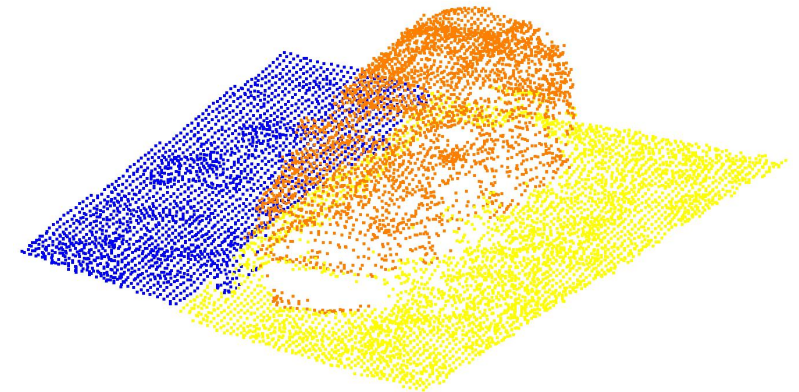
- ▶ Neighborhood information, needed for consistent labeling.
- ▶ A support, needed for efficient computation.



RGB features



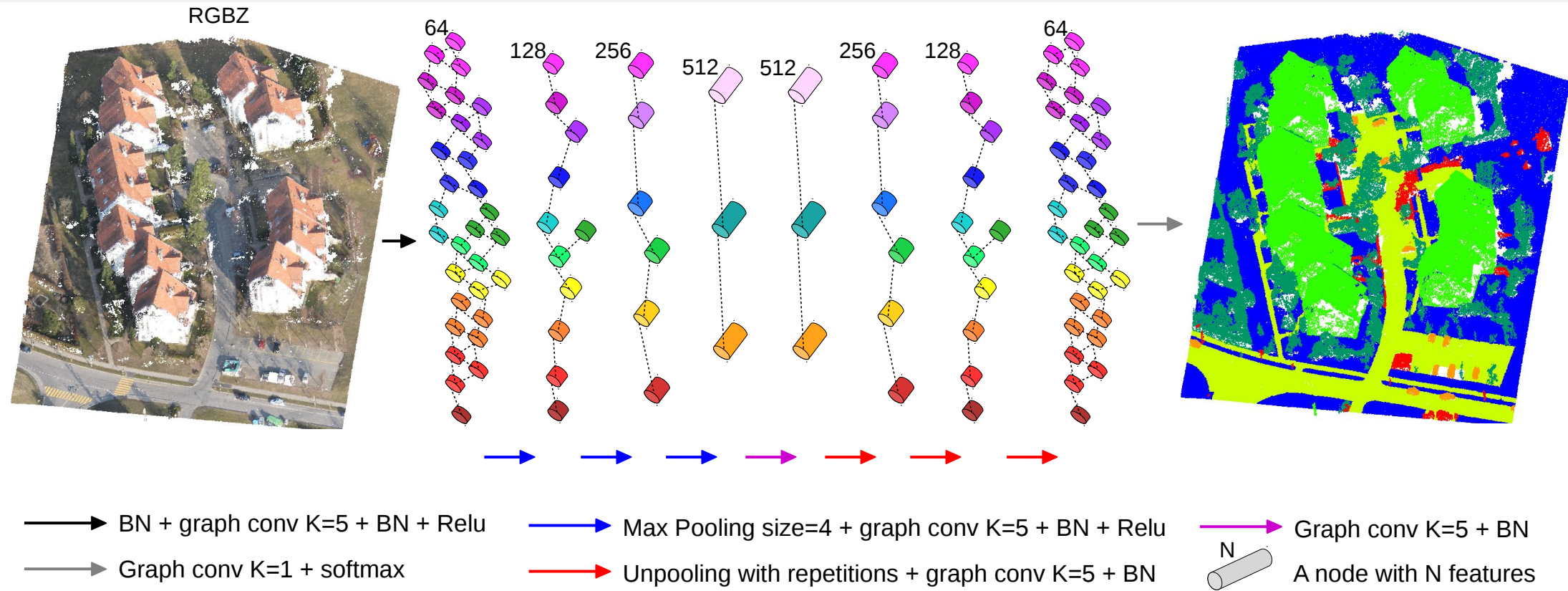
graph



labels

Model

Cherqui, Morsier, and Defferrard 2018



Characteristics:

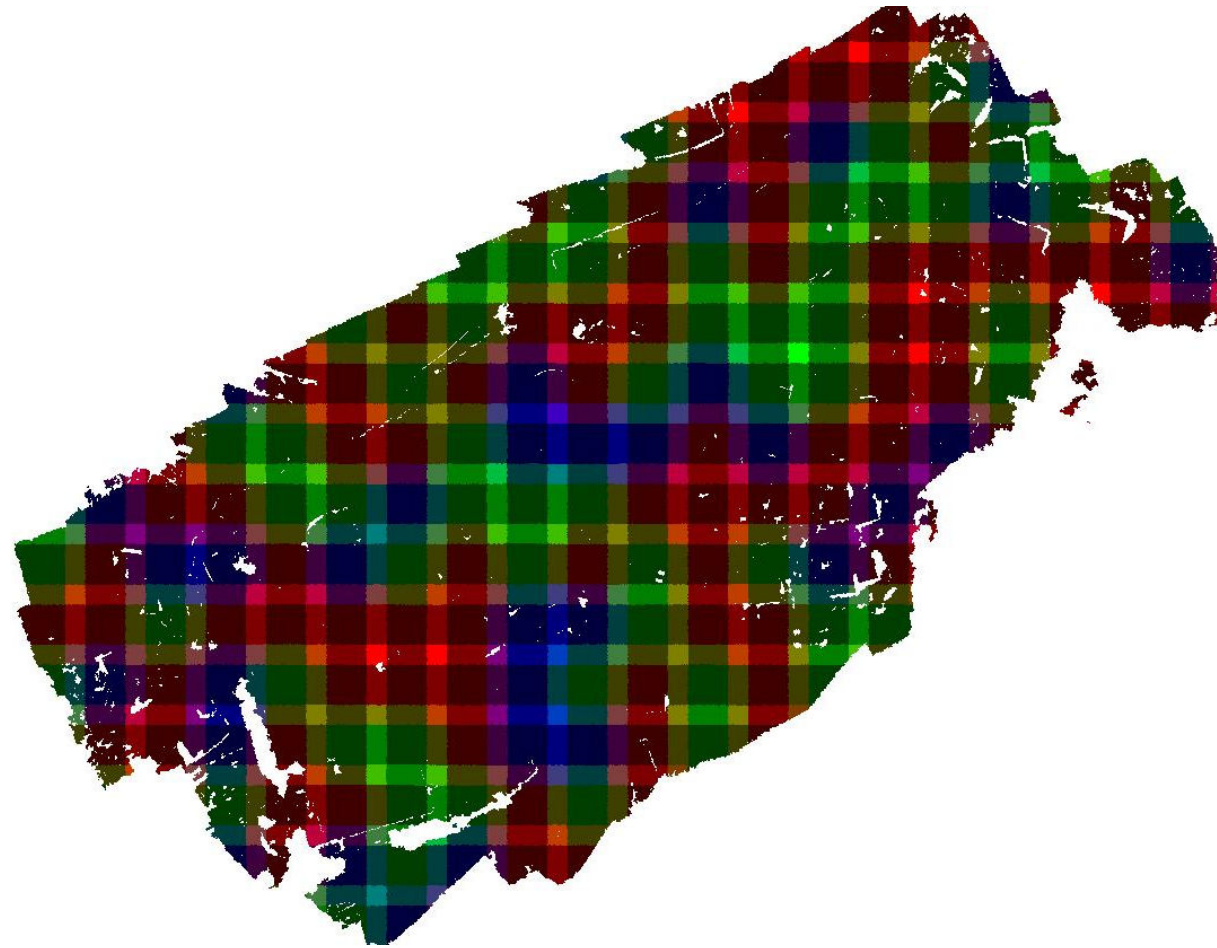
- ▶ Dense prediction.
- ▶ *Reason* at multiple scales.
- ▶ Local decisions.

Main difficulties:

- ▶ Large number of points.
- ▶ Training samples are of varying sizes.

Data preparation

Cherqui, Morsier, and Defferrard 2018

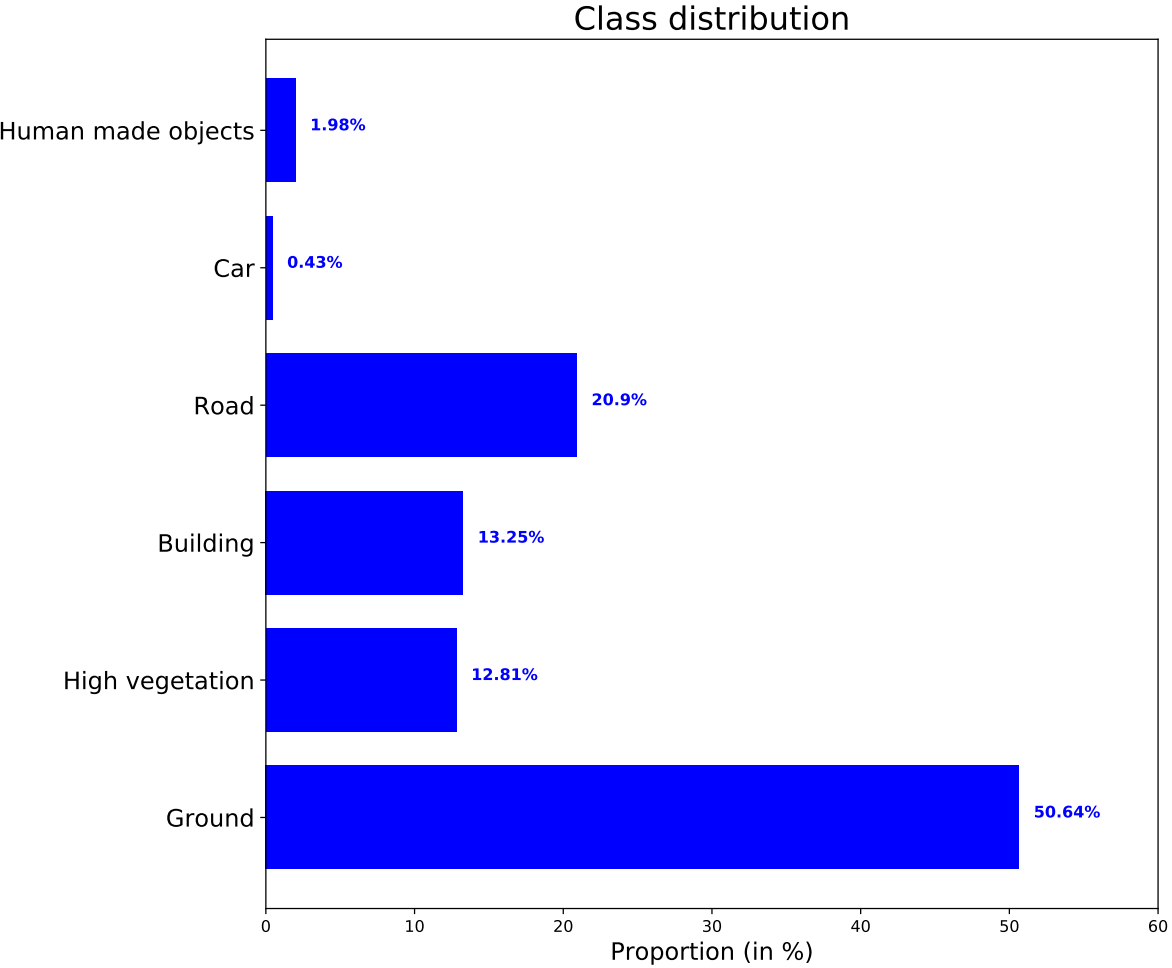


- ▶ tiling: $36m \times 36m$ ($48m \times 48m$ with context)
- ▶ split: 50% training tiles (green), 16% validation tiles (blue), 35% test tiles (red)

Results with RGBZ

Cherqui, Morsier, and Defferrard 2018

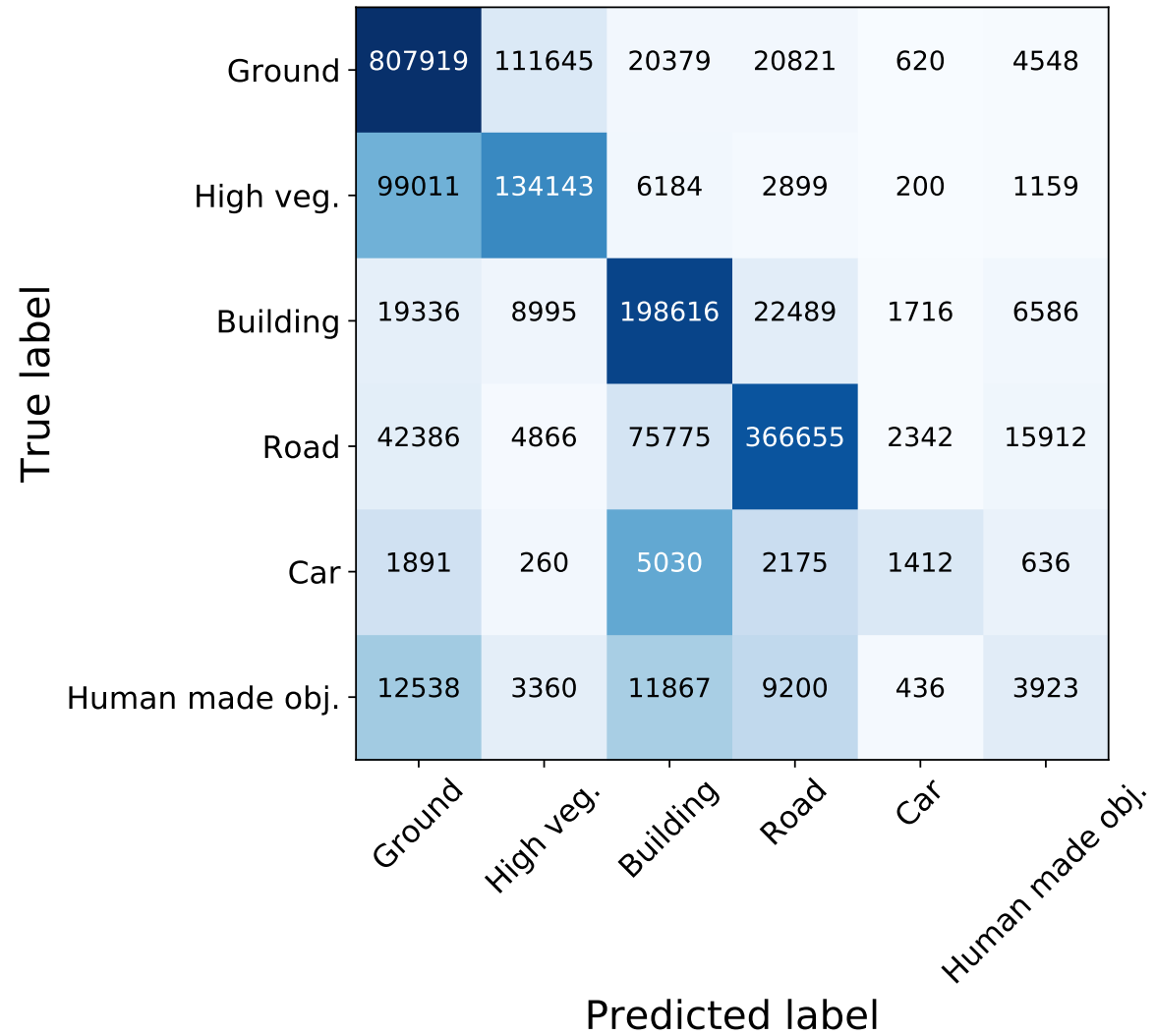
Model	Accuracy	
	Overall (micro)	Mean (macro)
Random Forest	75%	53%
Graph ConvNet	86%	68%



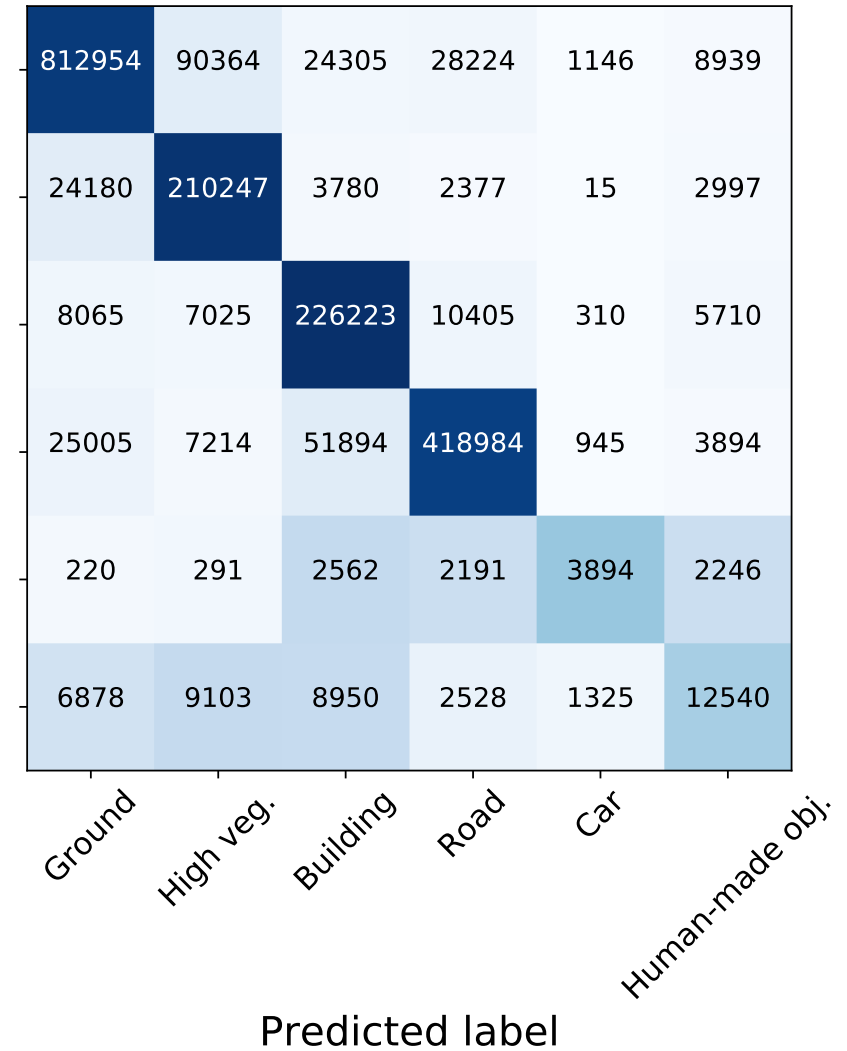
Results

Cherqui, Morsier, and Defferrard 2018

Random forest baseline



Graph ConvNet



Take-home message

Filters can be **designed** to solve known problems.

If the transformation is unknown, **learn** filters from examples.

PS: to practice, try the PyGSP from <https://github.com/epfl-lts2/pygsp>.