

# Recent Advances in Learning with Graphs

Xiaowen Dong

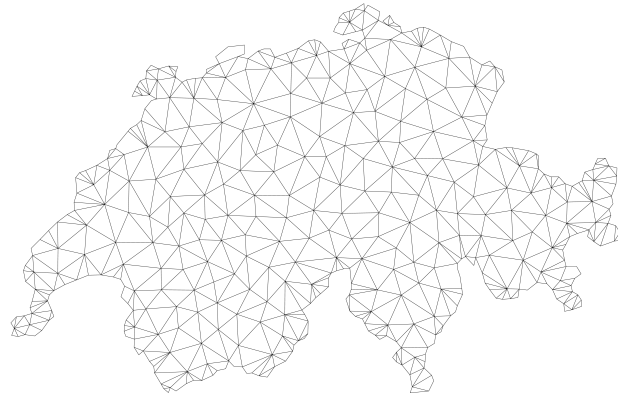
Department of Engineering Science

University of Oxford

Clarkson University, March 2021



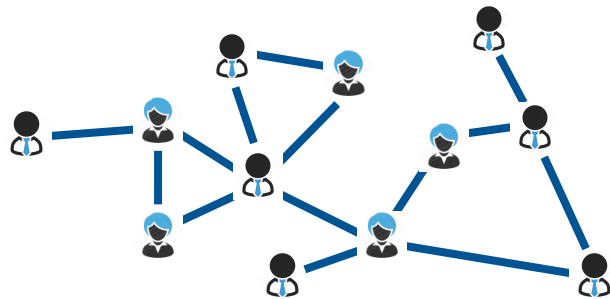
# Networks are pervasive



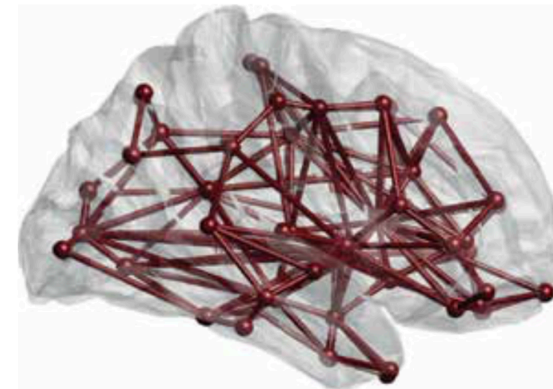
**geographical network**



**traffic network**

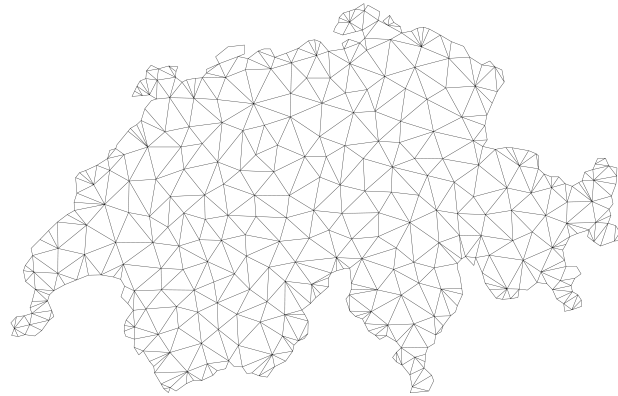


**social network**



**brain network**

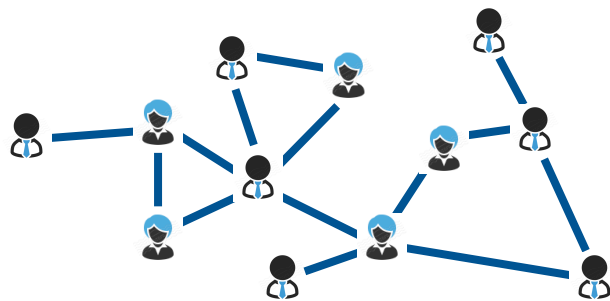
# Networks are pervasive



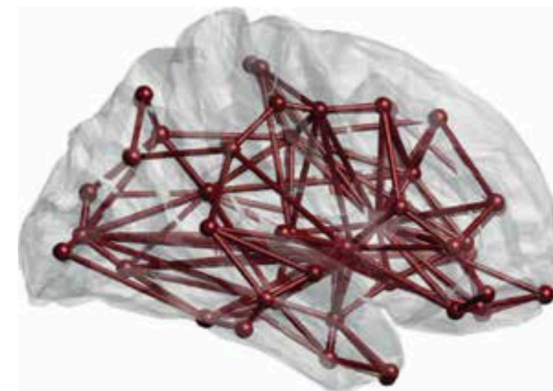
**geographical network**



**traffic network**



**social network**

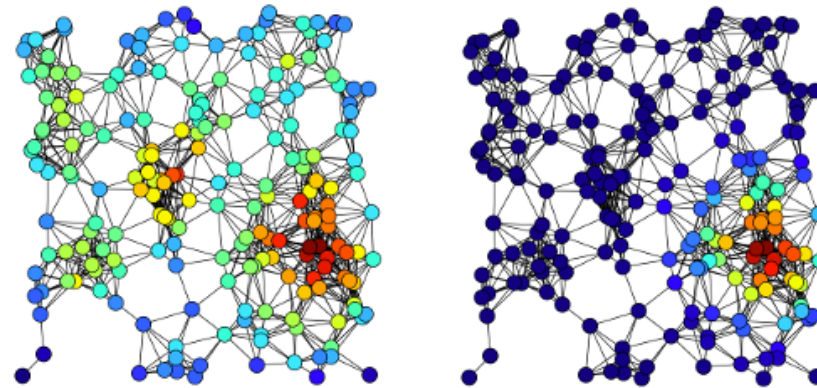


**brain network**

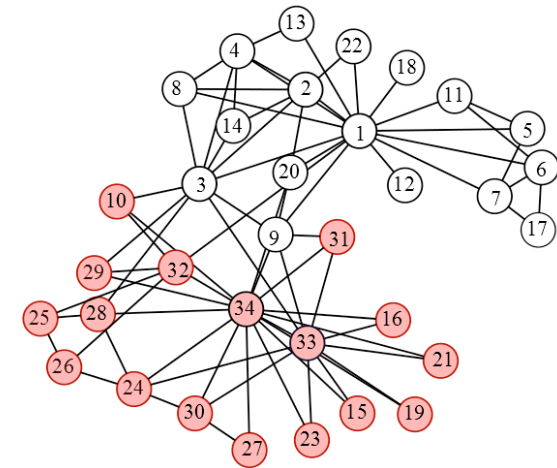
**graphs provide mathematical representation of networks**

# The field of network science

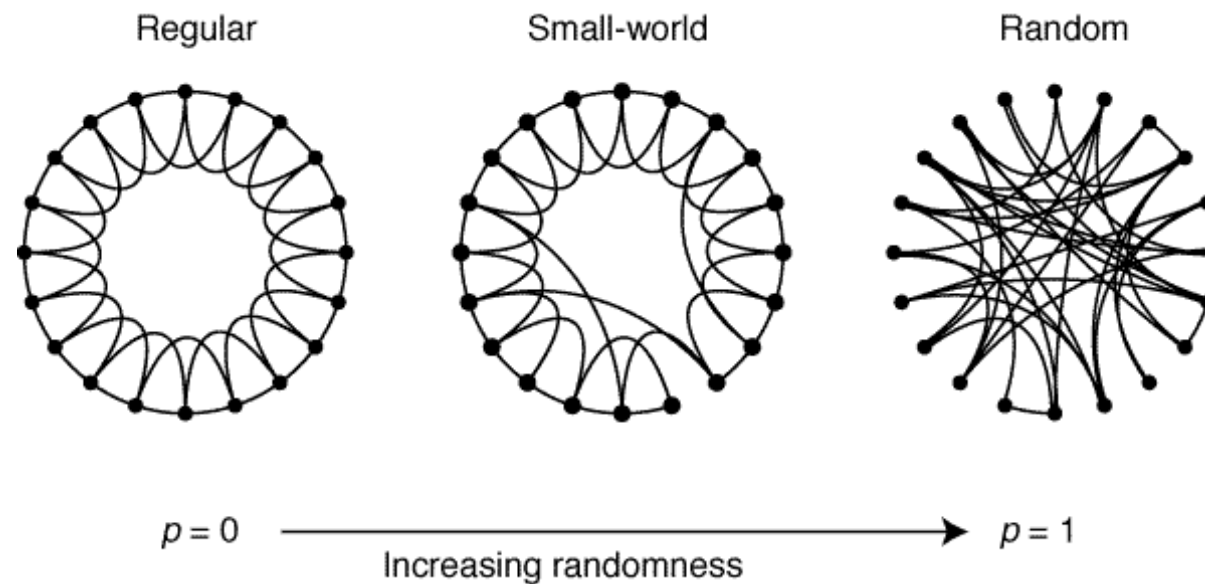
network  
centrality



community  
detection



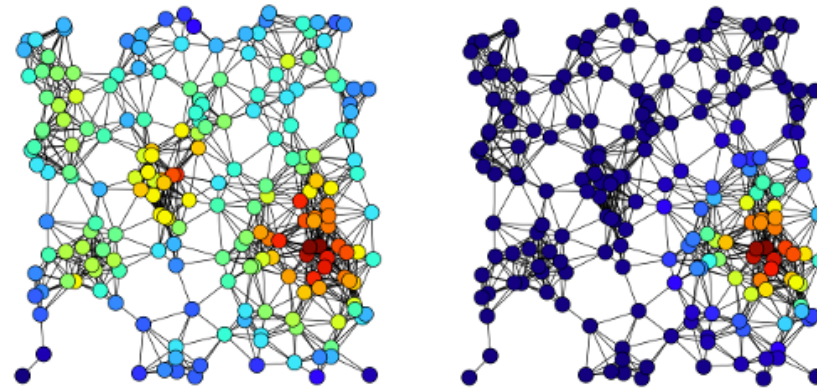
random graph  
models



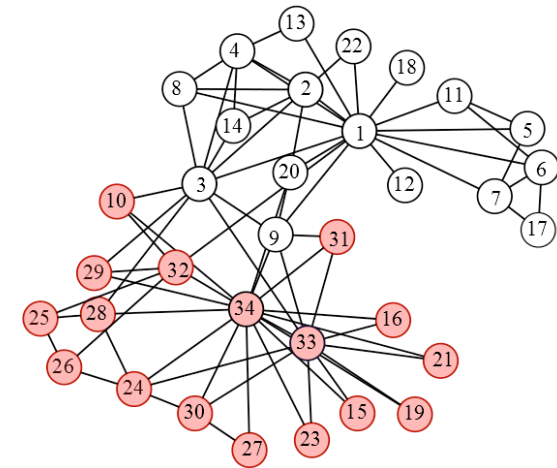


# The field of network science

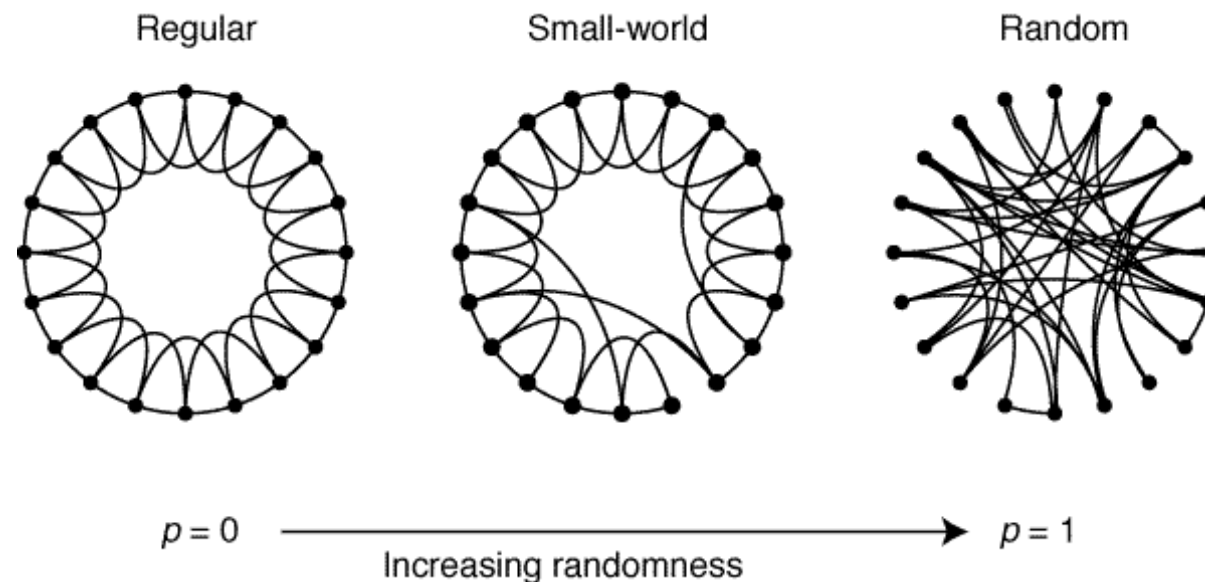
network  
centrality



community  
detection

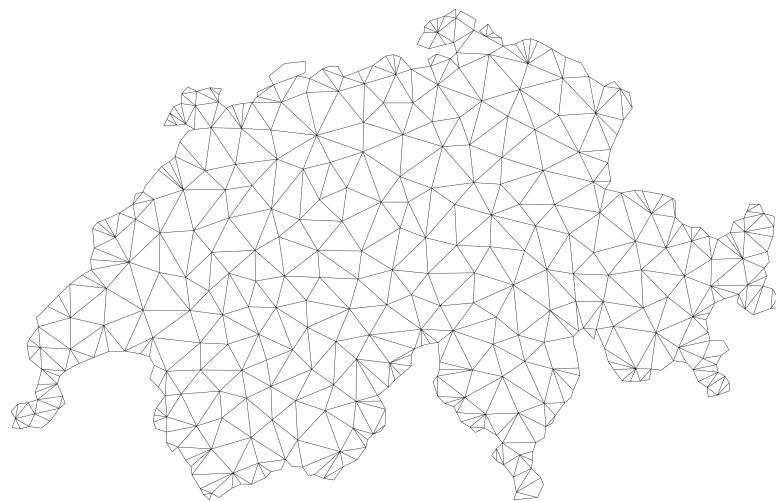


random graph  
models



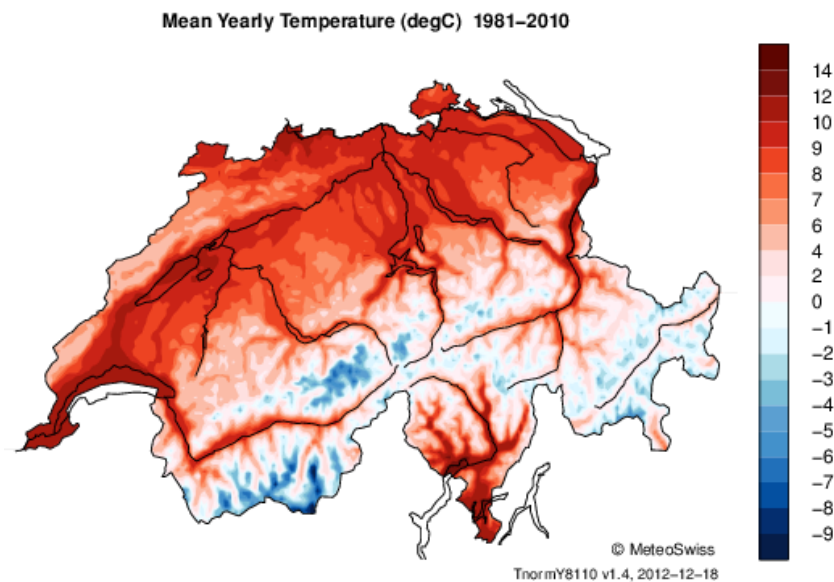
from **edge attributes** to **node attributes**  
from **graphs** to **graph-structured data**

# Graph-structured data are pervasive



- vertices
  - geographical regions
- edges
  - geographical proximity between regions

# Graph-structured data are pervasive



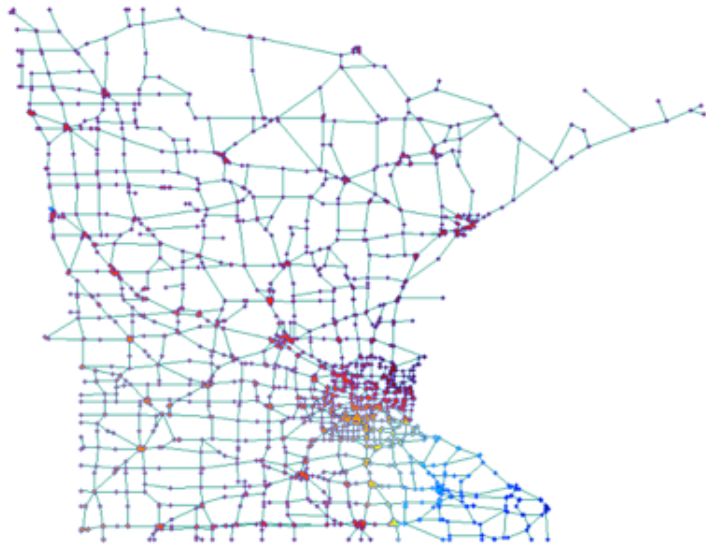
- vertices
  - geographical regions
- edges
  - geographical proximity between regions
- signal
  - temperature records in these regions

# Graph-structured data are pervasive



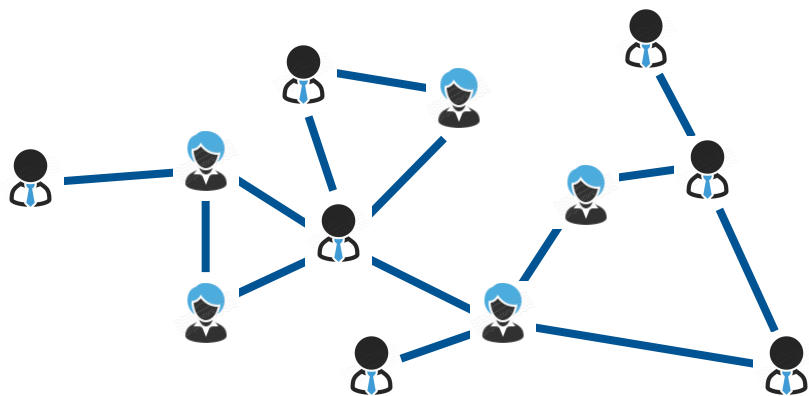
- nodes
  - road junctions
- edges
  - road connections

# Graph-structured data are pervasive



- nodes
  - road junctions
- edges
  - road connections
- signal
  - traffic congestion at junctions

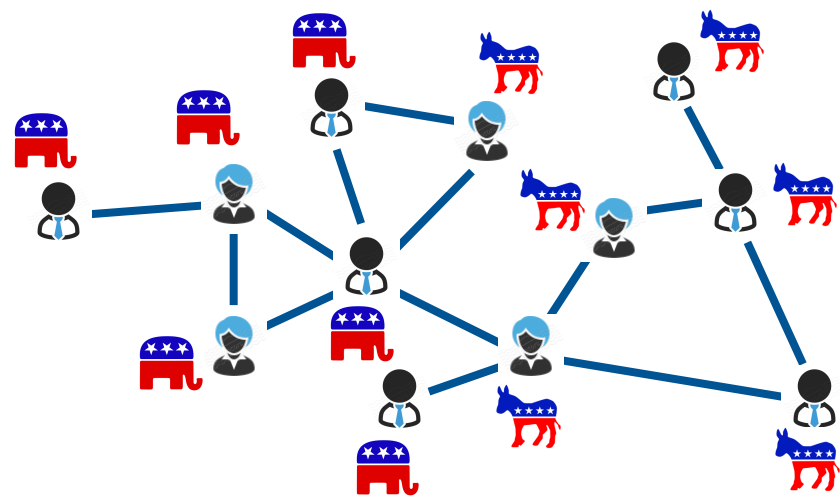
# Graph-structured data are pervasive



- nodes
  - individuals
- edges
  - friendship between individuals



# Graph-structured data are pervasive



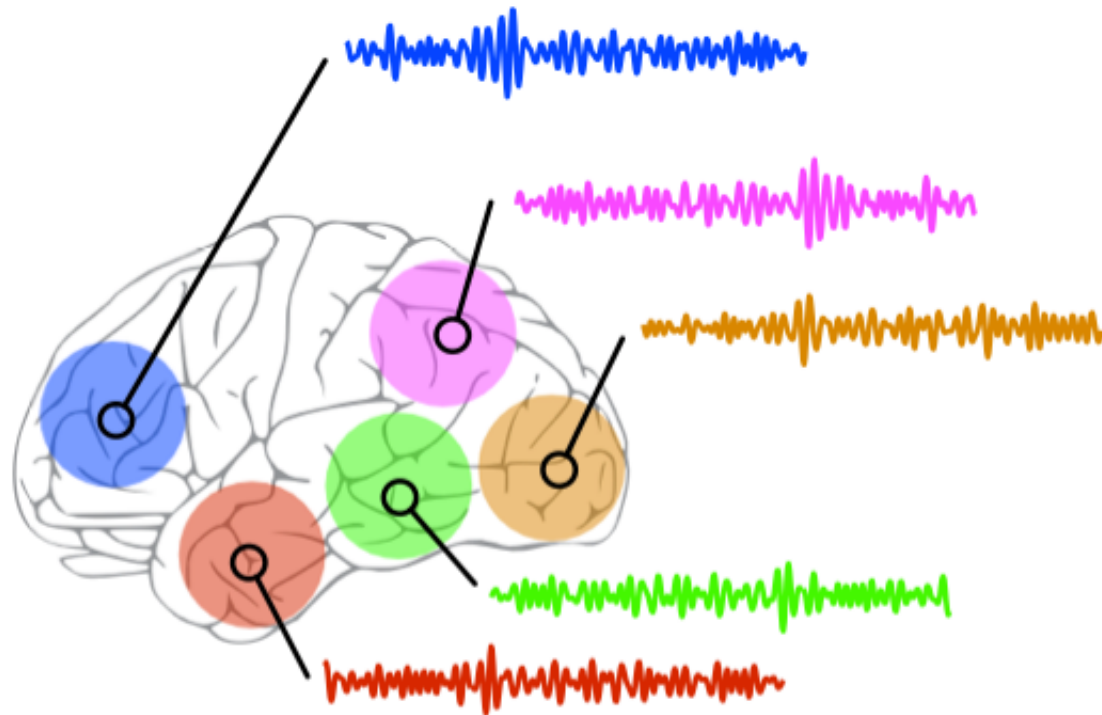
- nodes
  - individuals
- edges
  - friendship between individuals
- signal
  - political view

# Graph-structured data are pervasive



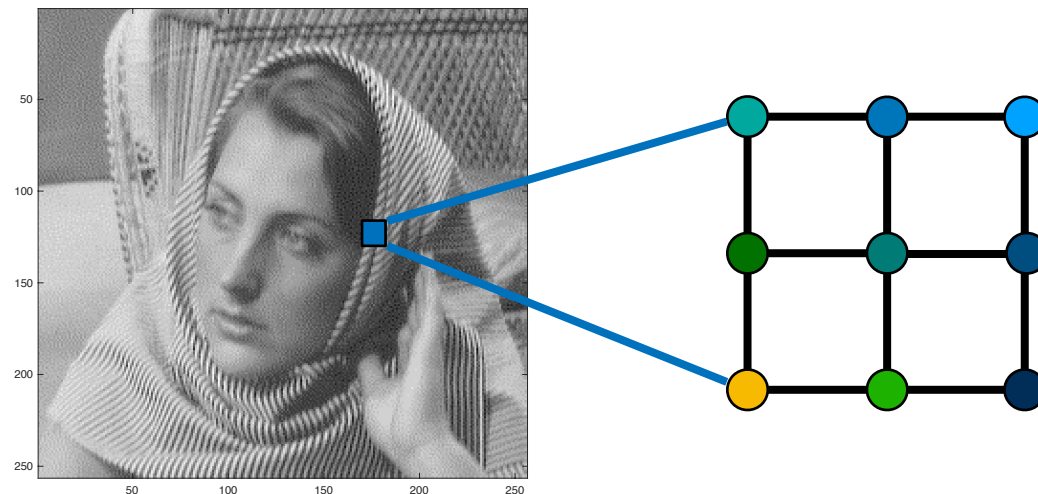
- nodes
  - brain regions
- edges
  - structural connectivity between brain regions

# Graph-structured data are pervasive



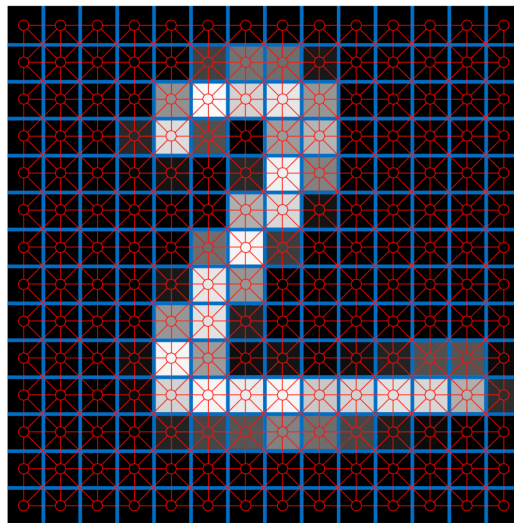
- nodes
  - brain regions
- edges
  - structural connectivity between brain regions
- signal
  - blood-oxygen-level-dependent (BOLD) time series

# Graph-structured data are pervasive

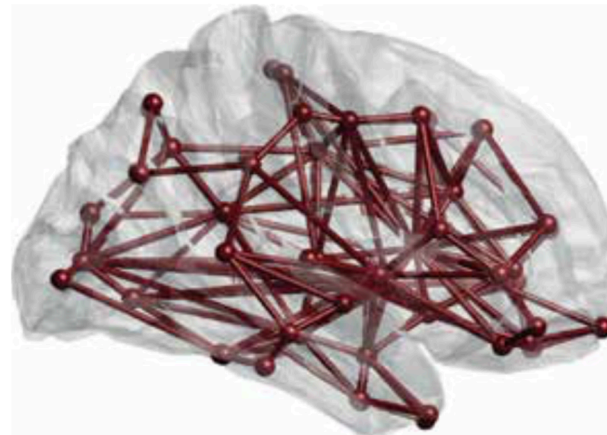


- nodes
  - pixels
- edges
  - spatial proximity between pixels
- signal
  - pixel values

# Learning with graph-structured data



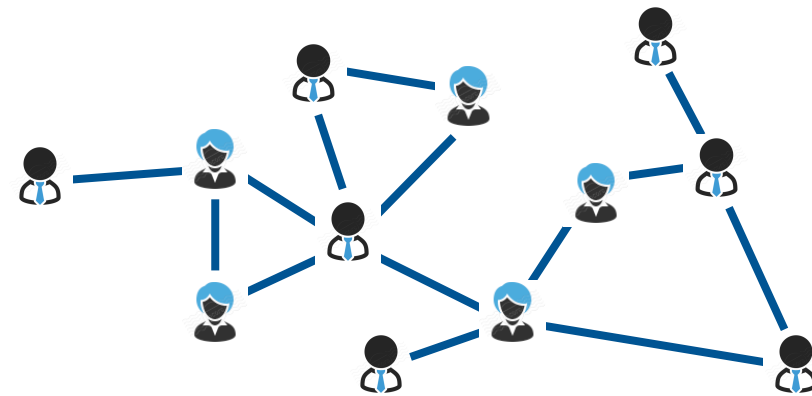
is it a 2?  
is it a 4?



condition?  
no condition?

**(supervised) graph-wise classification**

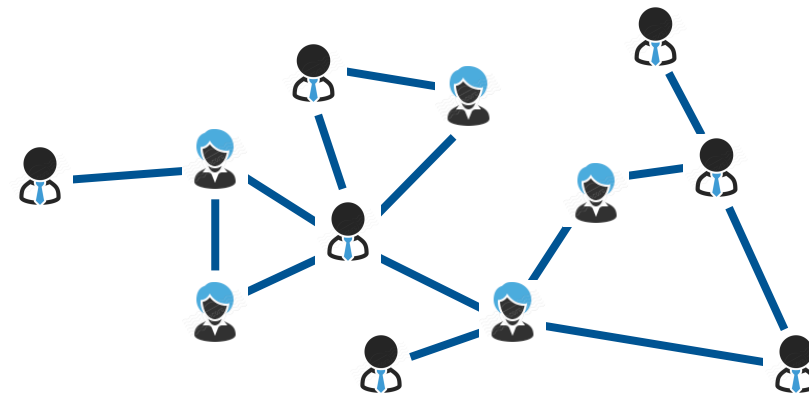
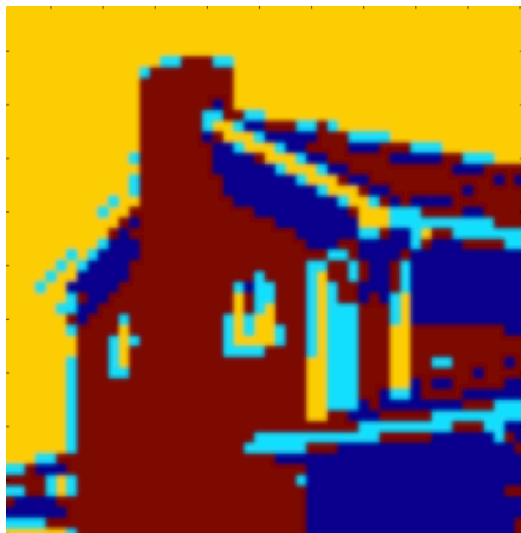
# Learning with graph-structured data



**(semi-supervised) node-wise classification**

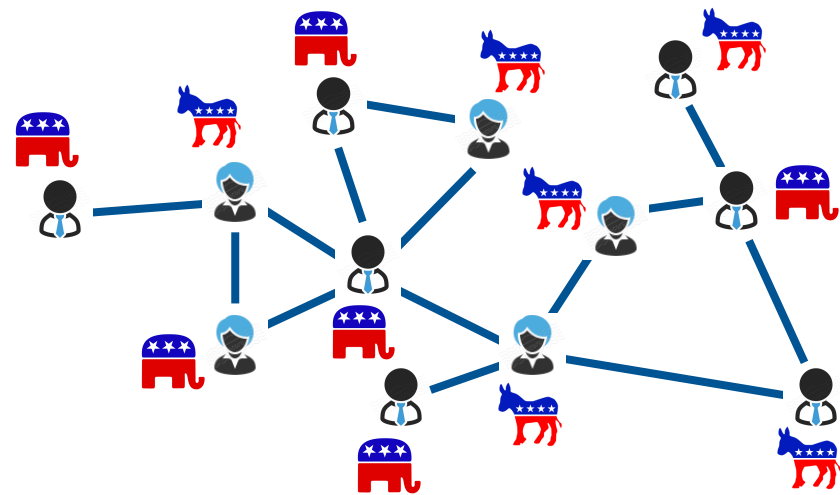
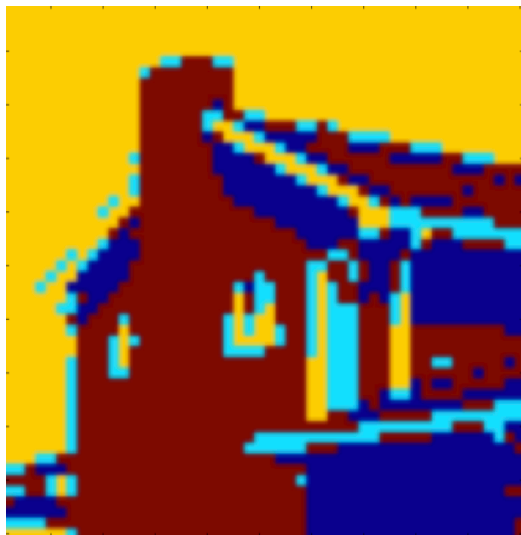


# Learning with graph-structured data



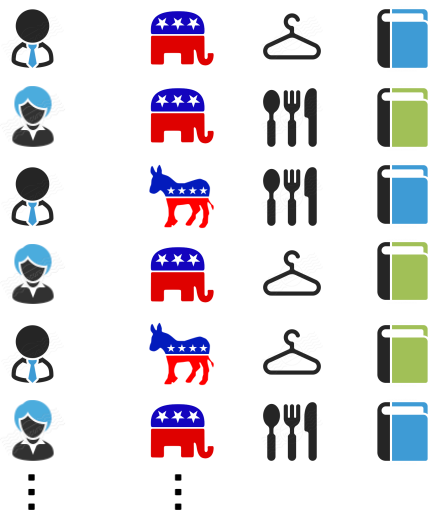
**(semi-supervised) node-wise classification**

# Learning with graph-structured data



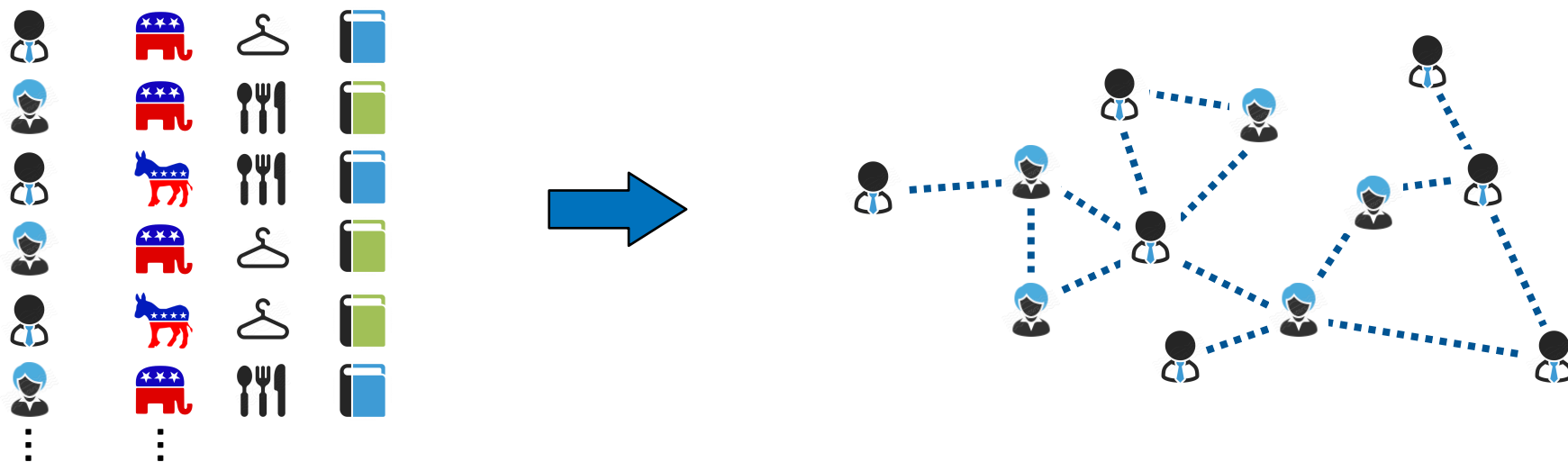
(semi-supervised) node-wise classification

# Learning with graph-structured data



learning graph structure from data

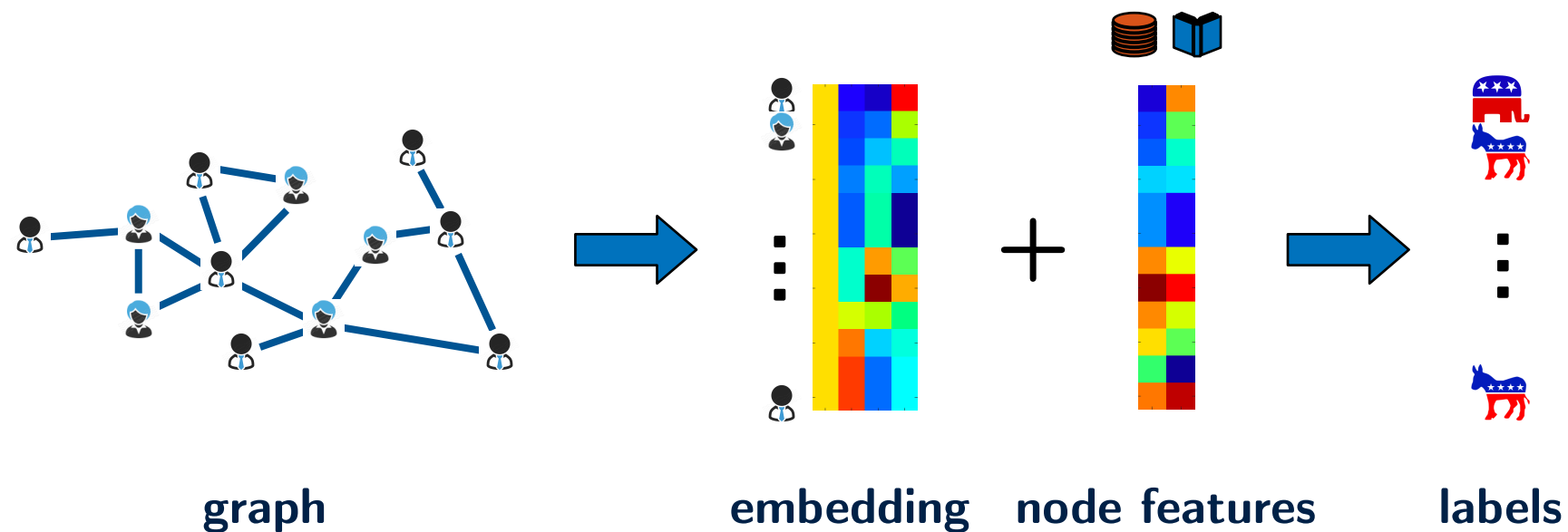
# Learning with graph-structured data



learning graph structure from data

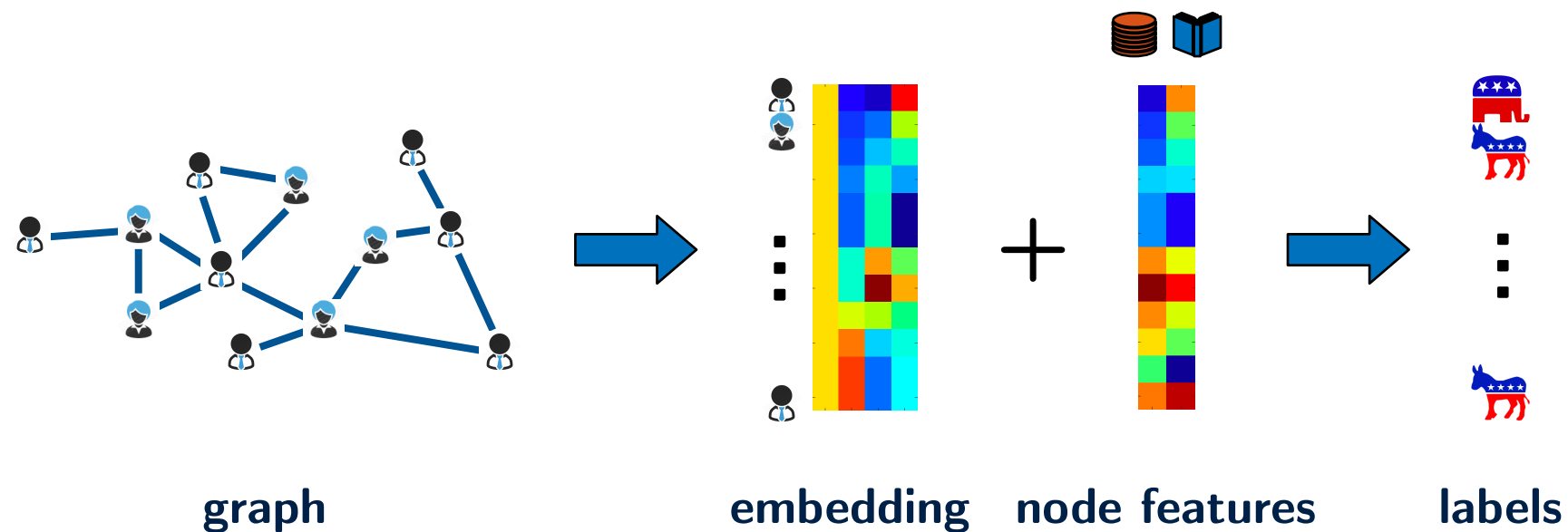
# How to incorporate graph into learning?

- Embedding of graph structure leads to information loss



# How to incorporate graph into learning?

- Embedding of graph structure leads to information loss



- Need for new models that directly incorporate structure in data analysis
  - graph signal processing (GSP)
  - graph neural networks (GNN)



# Outline

- Graph signal processing (GSP): Basic concepts
- Graph spectral filtering: Basic tools of GSP
- Graph neural networks (GNNs)
- Applications

# Outline

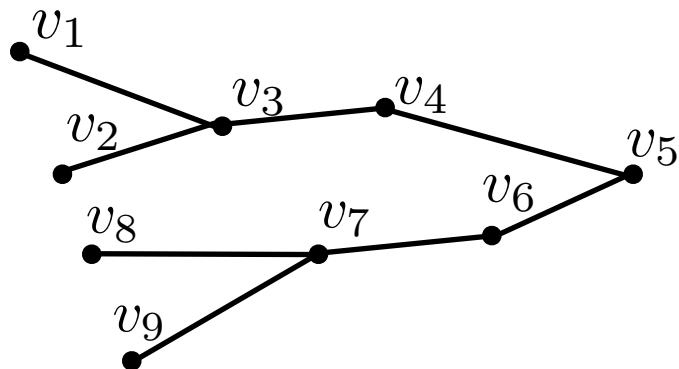
- Graph signal processing (GSP): Basic concepts
- Graph spectral filtering: Basic tools of GSP
- Graph neural networks (GNNs)
- Applications

# Graph signal processing (GSP)

- Graph-structured data can be represented by signals defined on graphs or **graph signals**

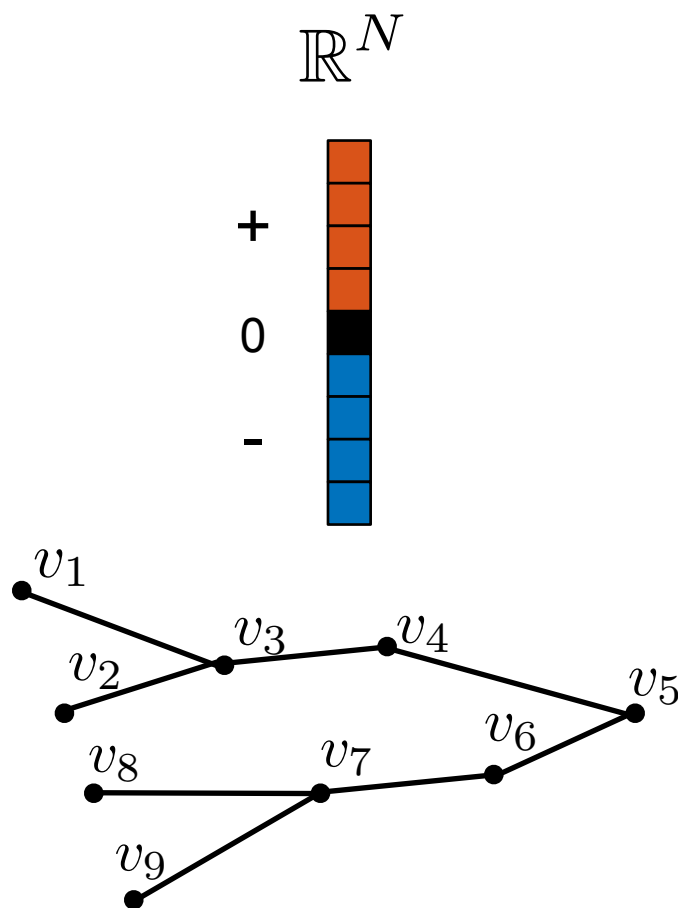
# Graph signal processing (GSP)

- Graph-structured data can be represented by signals defined on graphs or **graph signals**



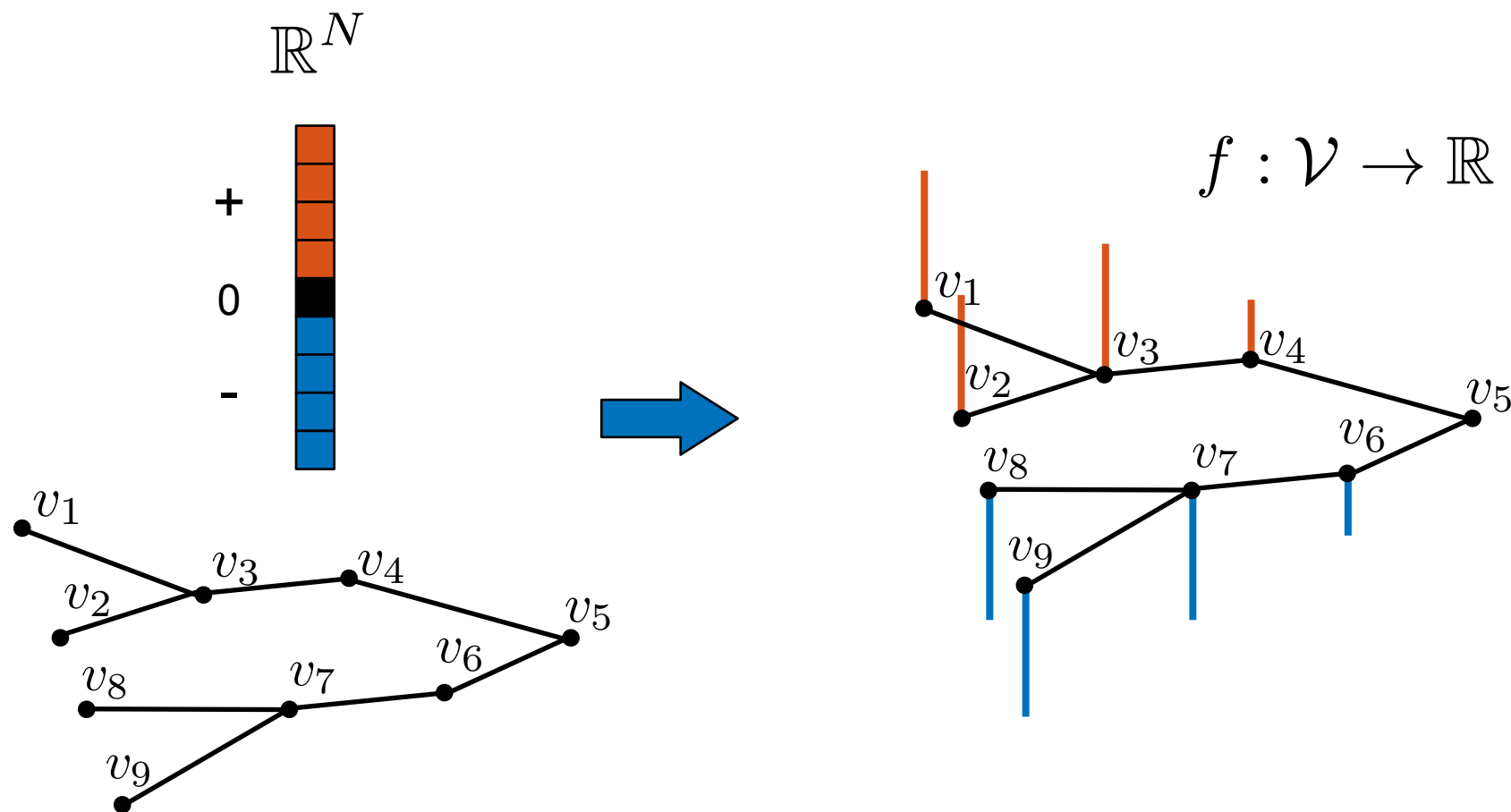
# Graph signal processing (GSP)

- Graph-structured data can be represented by signals defined on graphs or **graph signals**



# Graph signal processing (GSP)

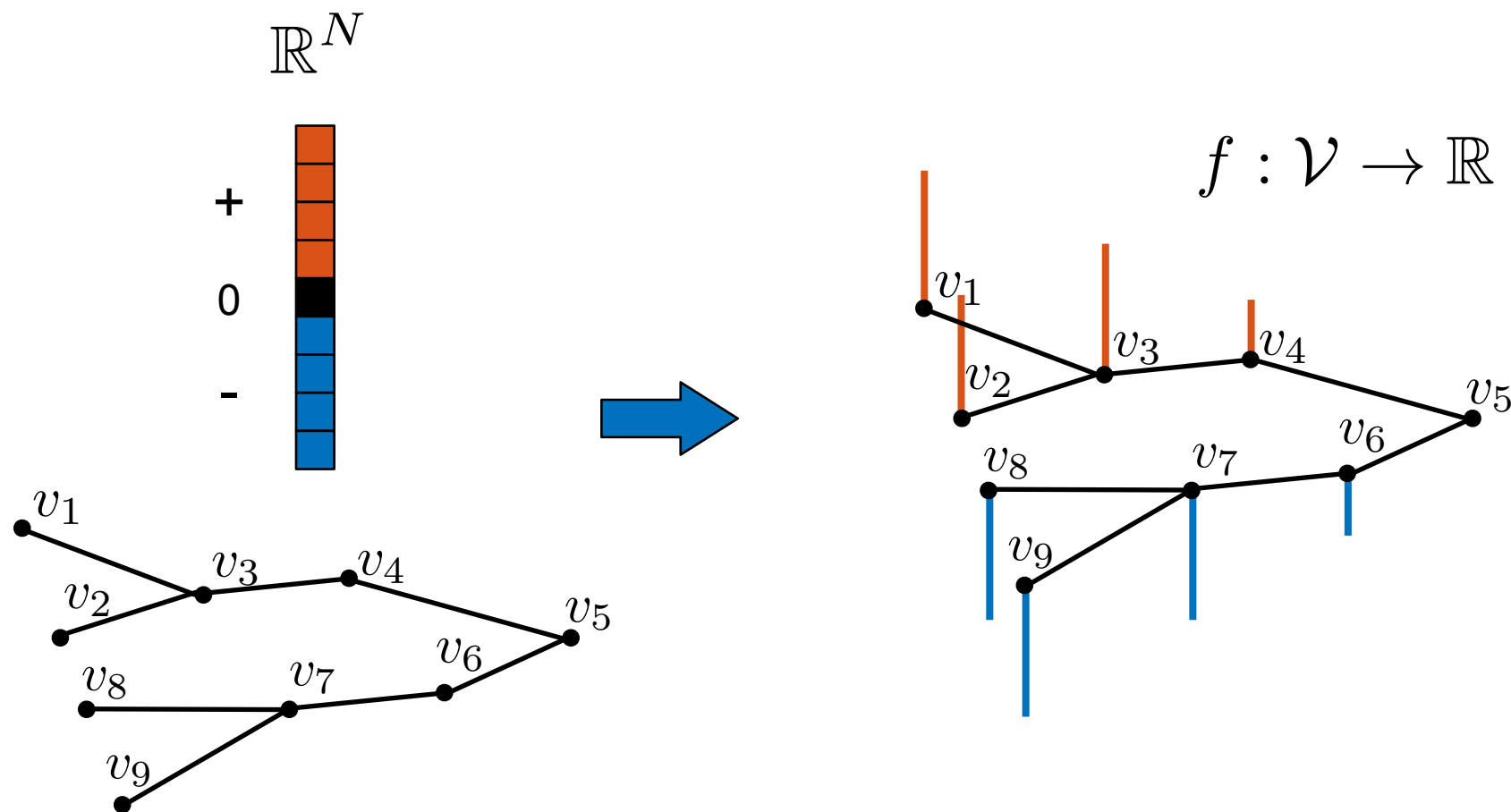
- Graph-structured data can be represented by signals defined on graphs or **graph signals**





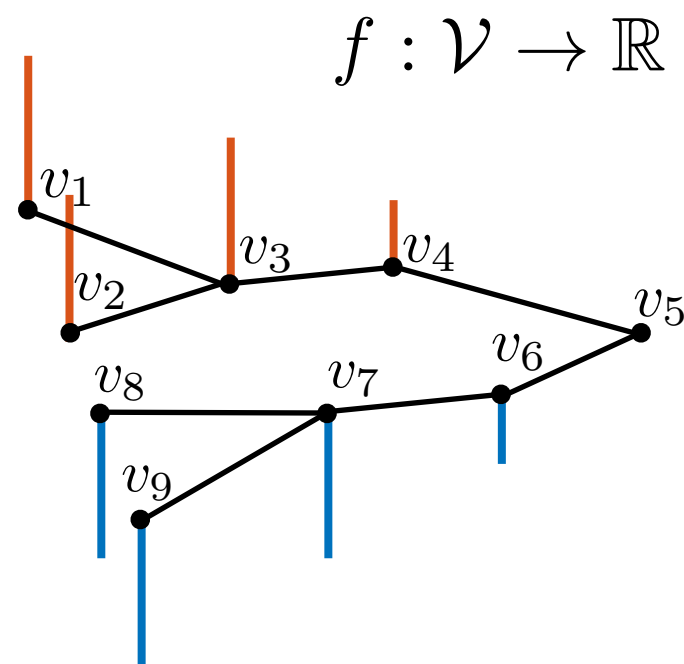
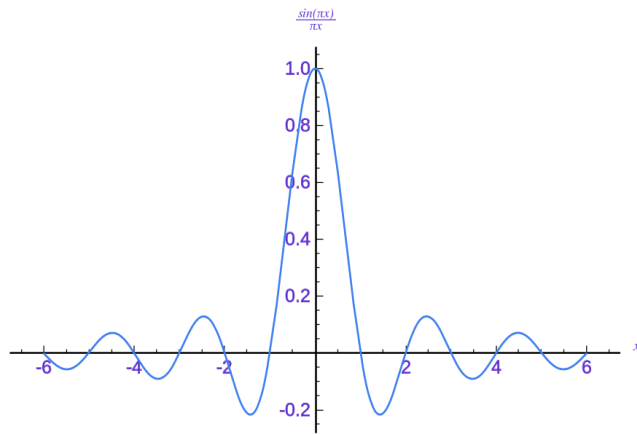
# Graph signal processing (GSP)

- Graph-structured data can be represented by signals defined on graphs or **graph signals**



takes into account both structure (edges) and data (values at vertices)

# Graph signal processing (GSP)



how to generalise classical signal processing tools  
on irregular domains such as graphs?

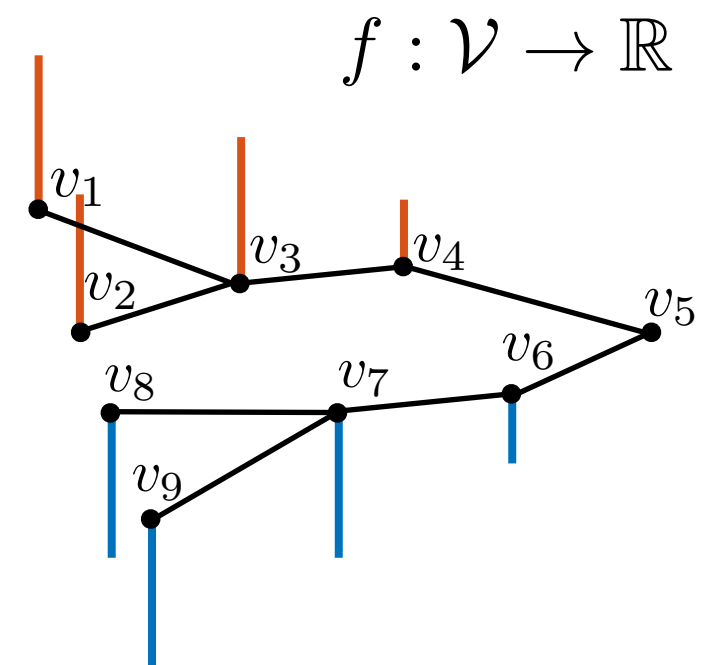
# Graph signal processing (GSP)

- Main GSP approaches can be categorised into two families:
  - vertex (spatial) domain designs
  - frequency (graph spectral) domain designs

# Graph signal processing (GSP)

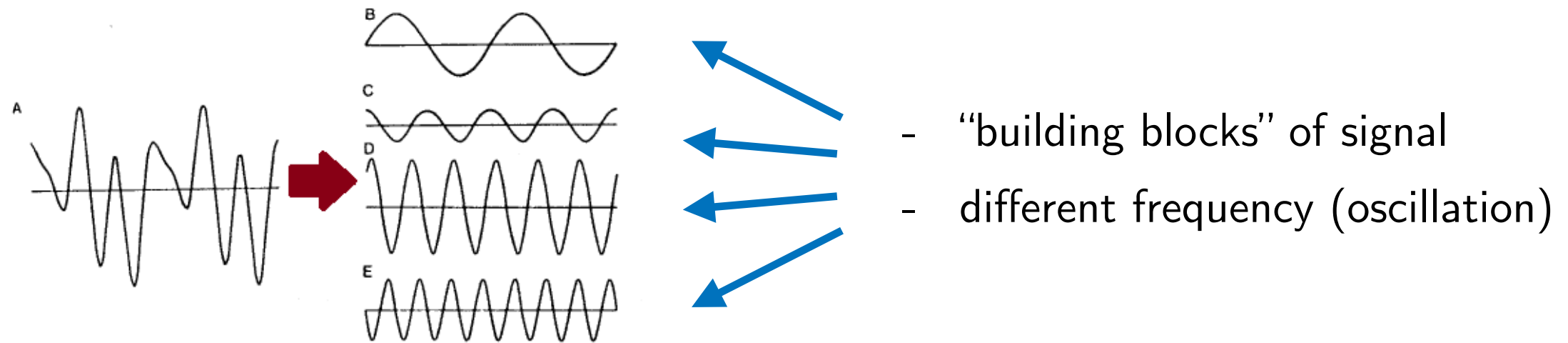
- Main GSP approaches can be categorised into two families:
  - vertex (spatial) domain designs
  - frequency (graph spectral) domain designs

**important for analysis of signal properties**



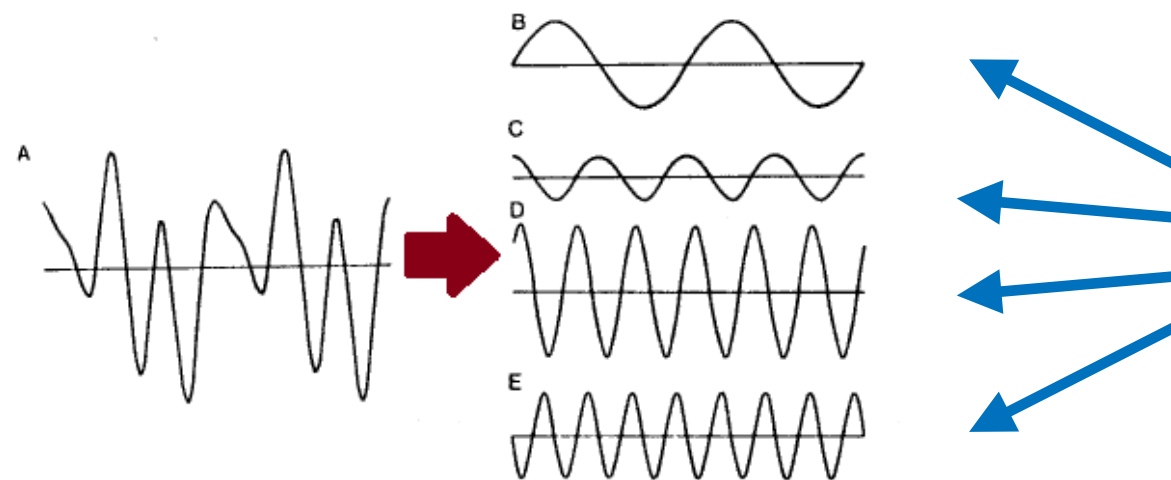
# Graph signal processing (GSP)

- Classical Fourier transform provides frequency domain representation of signals



# Graph signal processing (GSP)

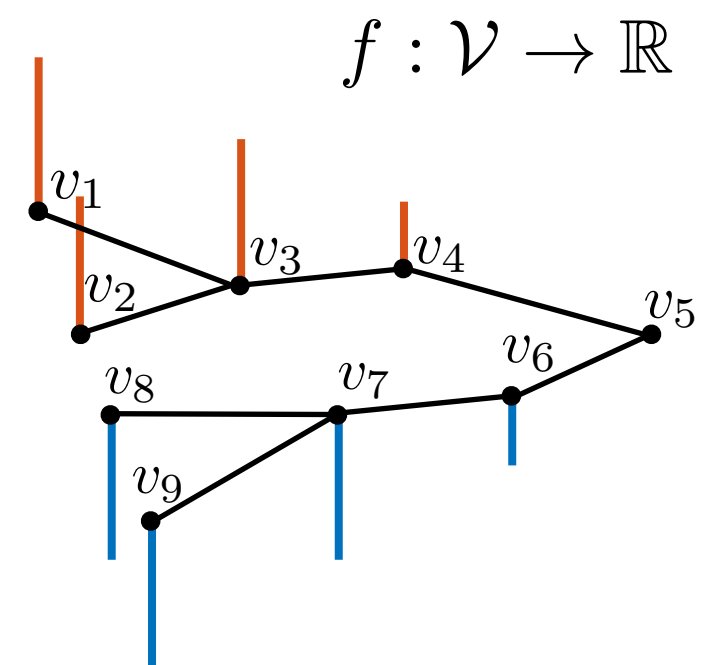
- Classical Fourier transform provides frequency domain representation of signals



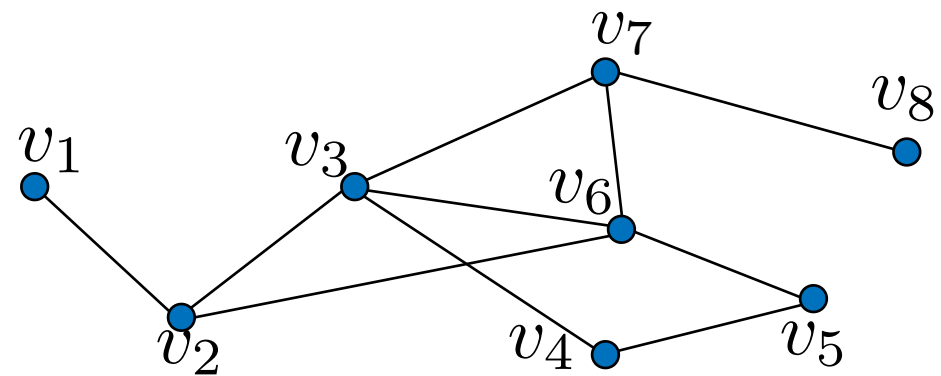
- “building blocks” of signal
- different frequency (oscillation)

- What about a notion of frequency for graph signals?

**we need the graph Laplacian matrix**



# Graph Laplacian



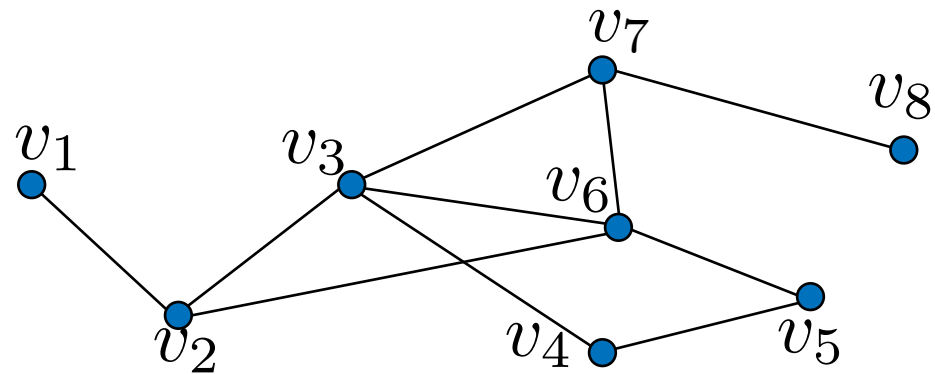
weighted and undirected graph:

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

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

$W$

# Graph Laplacian



weighted and undirected graph:

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

$$D = \text{diag}(d(v_1), \dots, d(v_N))$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

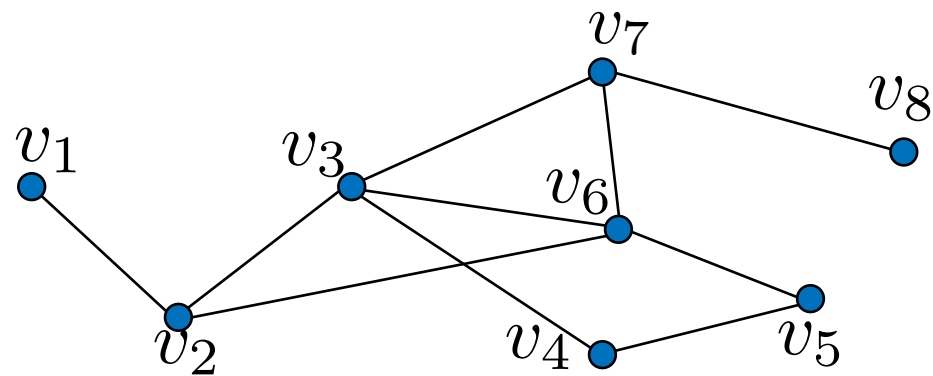
$D$

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

$W$



# Graph Laplacian



weighted and undirected graph:

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

$$D = \text{diag}(d(v_1), \dots, d(v_N))$$

$$L = D - W \quad \text{equivalent to } G!$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

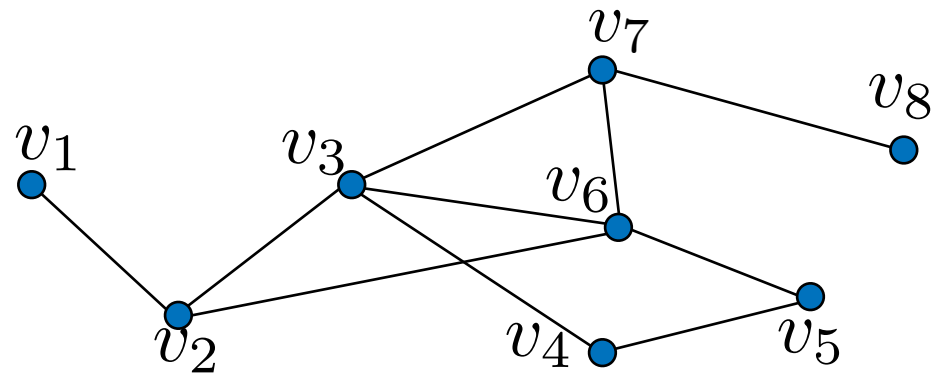
$D$

$W$

$L$

- symmetric
- off-diagonal entries non-positive
- rows sum up to zero

# Graph Laplacian



weighted and undirected graph:

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

$$D = \text{diag}(d(v_1), \dots, d(v_N))$$

$$L = D - W \quad \text{equivalent to } \mathbf{G}!$$

$$L_{\text{norm}} = D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

$D$

$W$

$L$

- symmetric
- off-diagonal entries non-positive
- rows sum up to zero

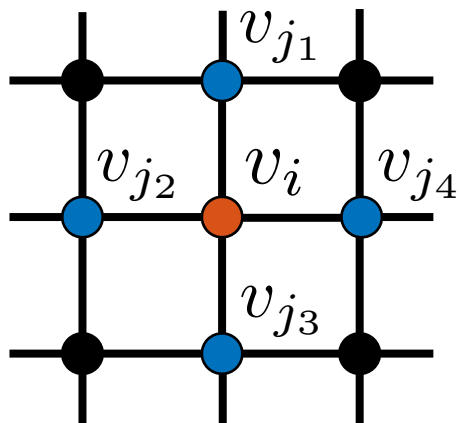
# Graph Laplacian

**Why graph Laplacian?**

# Graph Laplacian

## Why graph Laplacian?

- approximation of the Laplace operator



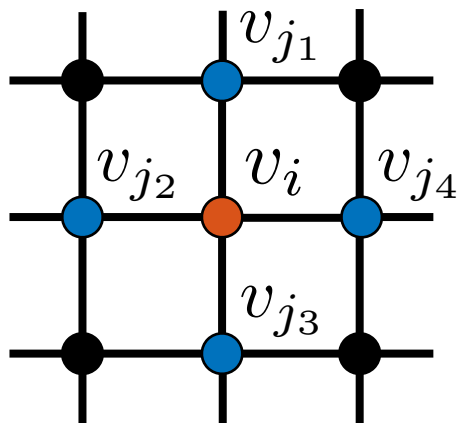
$$(Lf)(i) = 4f(i) - [f(j_1) + f(j_2) + f(j_3) + f(j_4)]$$

standard 5-point stencil for approximating  $-\nabla^2 f$

# Graph Laplacian

## Why graph Laplacian?

- approximation of the Laplace operator

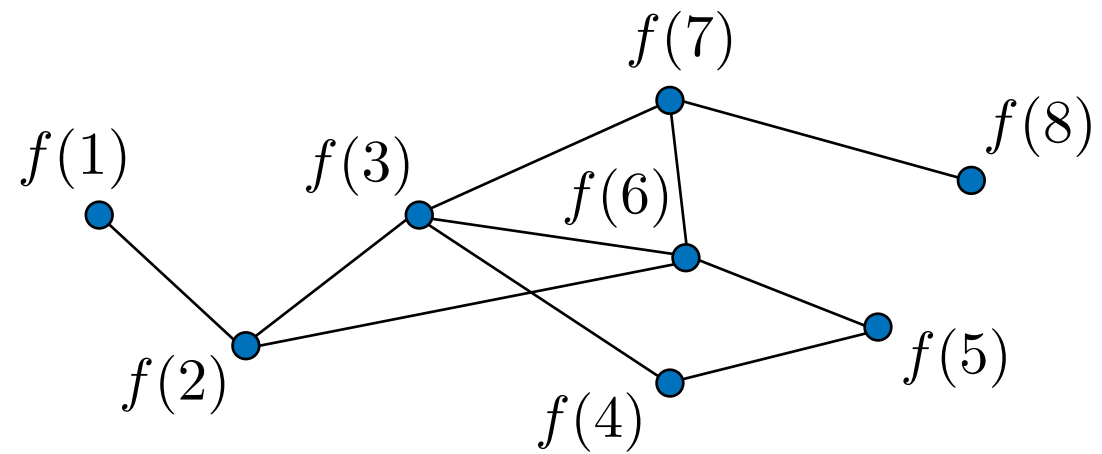


$$(Lf)(i) = 4f(i) - [f(j_1) + f(j_2) + f(j_3) + f(j_4)]$$

standard 5-point stencil for approximating  $-\nabla^2 f$

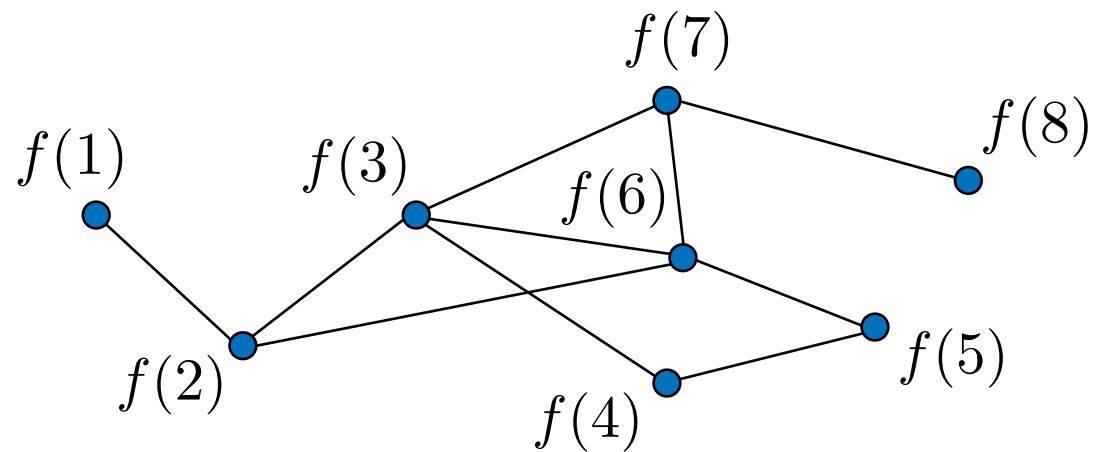
- converges to the Laplace-Beltrami operator (given certain conditions)
- provides a notion of “frequency” on graphs

# Graph Laplacian



graph signal  $f : \mathcal{V} \rightarrow \mathbb{R}^N$

# Graph Laplacian

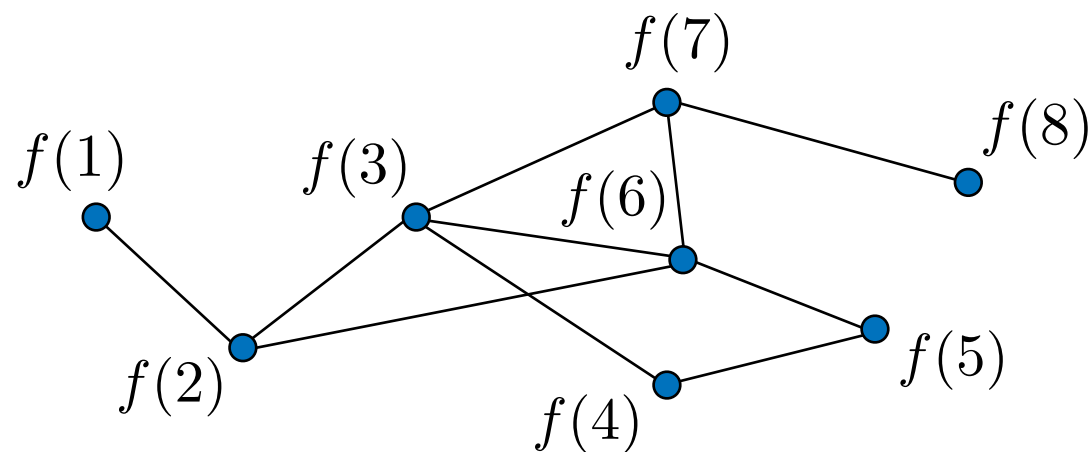


graph signal  $f : \mathcal{V} \rightarrow \mathbb{R}^N$

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{pmatrix}$$

$$Lf(i) = \sum_{j=1}^N W_{ij}(f(i) - f(j))$$

# Graph Laplacian



graph signal  $f : \mathcal{V} \rightarrow \mathbb{R}^N$

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{pmatrix}$$

$$\begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{pmatrix}^T \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{pmatrix}$$

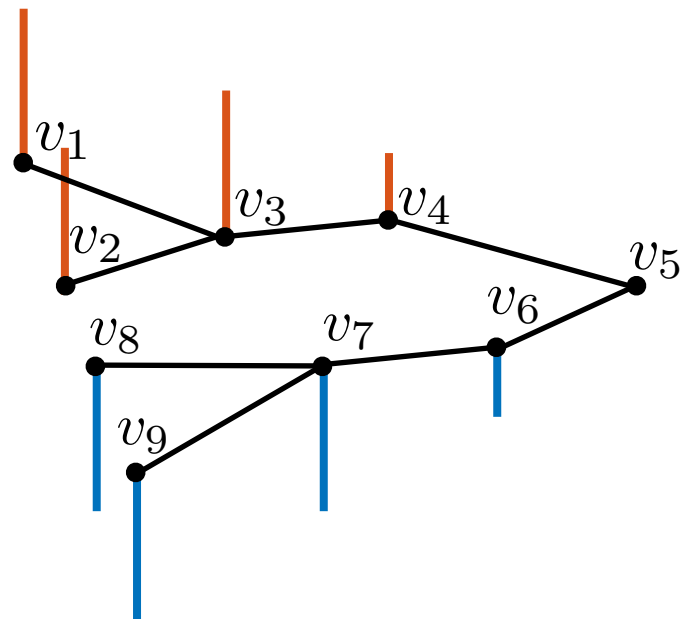
$$Lf(i) = \sum_{j=1}^N W_{ij} (f(i) - f(j))$$

$$f^T Lf = \frac{1}{2} \sum_{i,j=1}^N W_{ij} (f(i) - f(j))^2$$

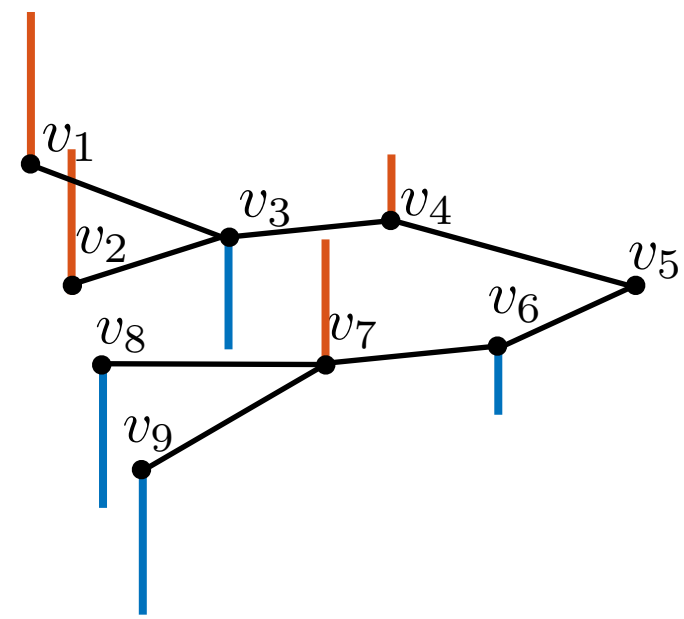
a measure of “smoothness”



# Graph Laplacian



$$f^T L f = 1$$



$$f^T L f = 21$$

# Graph Laplacian

- $L$  has a complete set of orthonormal eigenvectors:  $L = \chi \Lambda \chi^T$

$$L = \underbrace{\begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}}_{\chi} \underbrace{\begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} \text{---} \chi_0^T \text{---} \\ \cdots \\ \text{---} \chi_{N-1}^T \text{---} \end{bmatrix}}_{\chi^T}$$

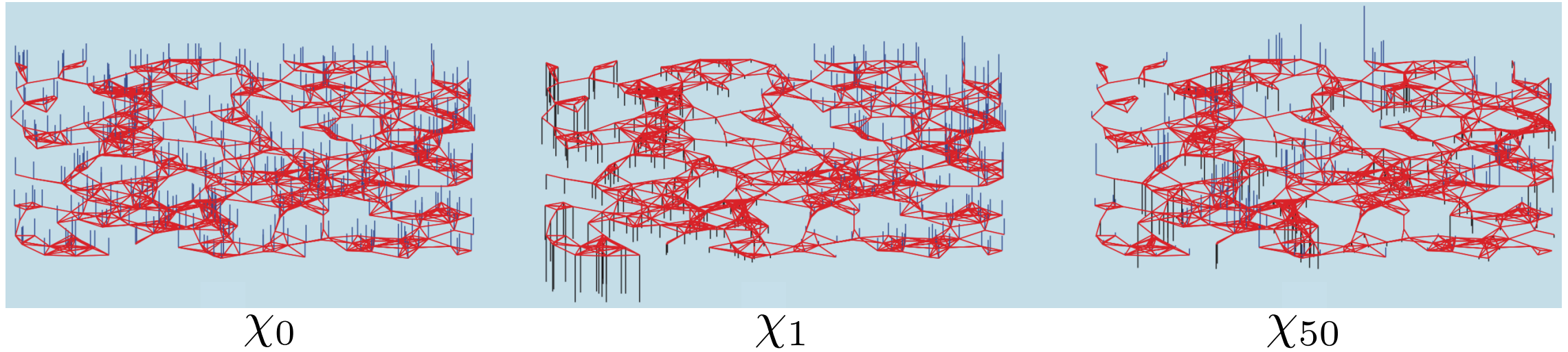
# Graph Laplacian

- $L$  has a complete set of orthonormal eigenvectors:  $L = \chi \Lambda \chi^T$

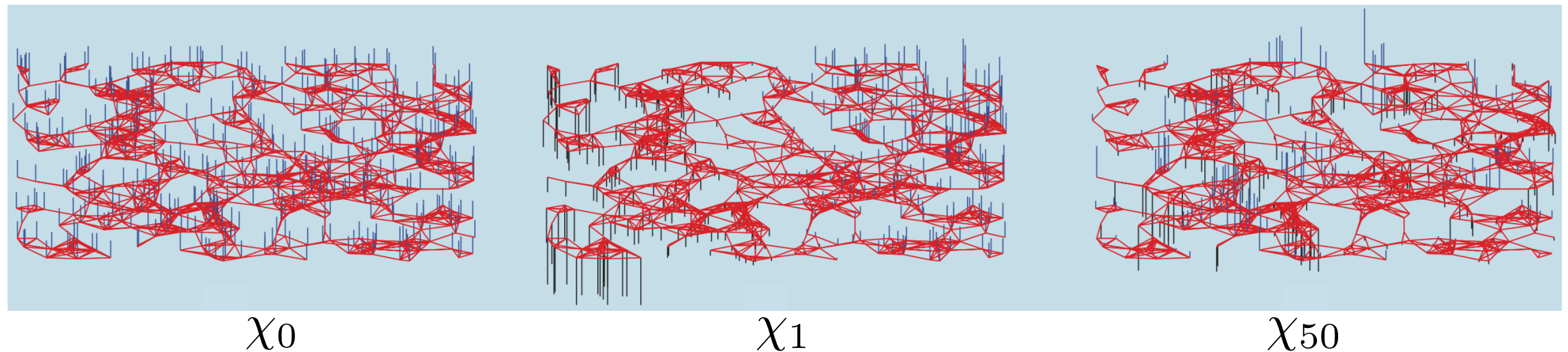
$$L = \underbrace{\begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}}_{\chi} \underbrace{\begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} \text{---} \chi_0^T \text{---} \\ \cdots \\ \text{---} \chi_{N-1}^T \text{---} \end{bmatrix}}_{\chi^T}$$

- Eigenvalues are usually sorted increasingly:  $0 = \lambda_0 < \lambda_1 \leq \dots \leq \lambda_{N-1}$

# Graph Fourier transform



# Graph Fourier transform



low frequency

high frequency

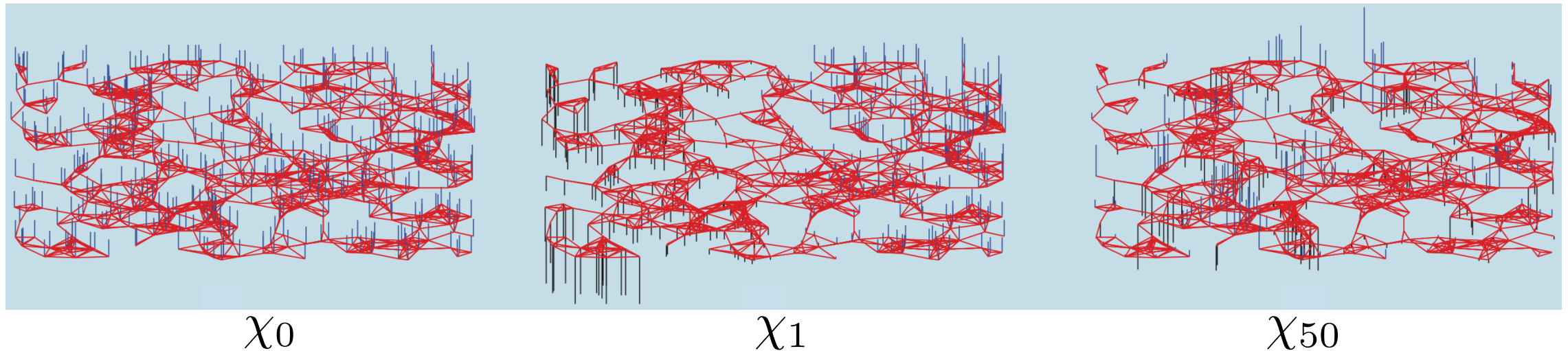
$$L = \chi \Lambda \chi^T$$

$$\chi_0^T L \chi_0 = \lambda_0 = 0$$

$$\chi_{50}^T L \chi_{50} = \lambda_{50}$$

- Eigenvectors associated with smaller eigenvalues have values that vary less rapidly along the edges

# Graph Fourier transform



$$L = \chi \Lambda \chi^T$$

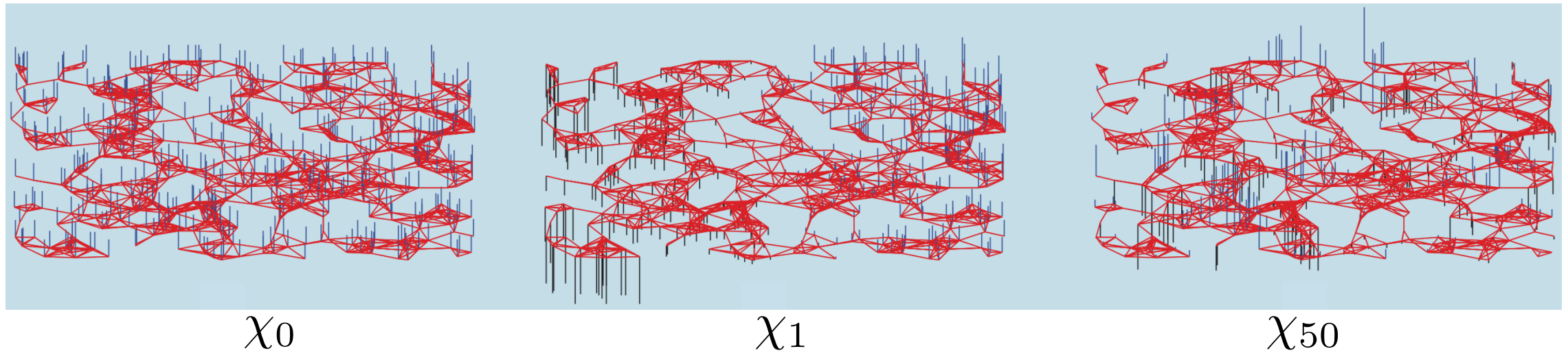
$$\chi_0^T L \chi_0 = \lambda_0 = 0$$

$$\chi_{50}^T L \chi_{50} = \lambda_{50}$$

graph Fourier transform:  
[Hammond11]

$$\hat{f}(\ell) = \langle \chi_\ell, f \rangle : \begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}^T \begin{bmatrix} | \\ f \\ | \end{bmatrix}$$

# Graph Fourier transform



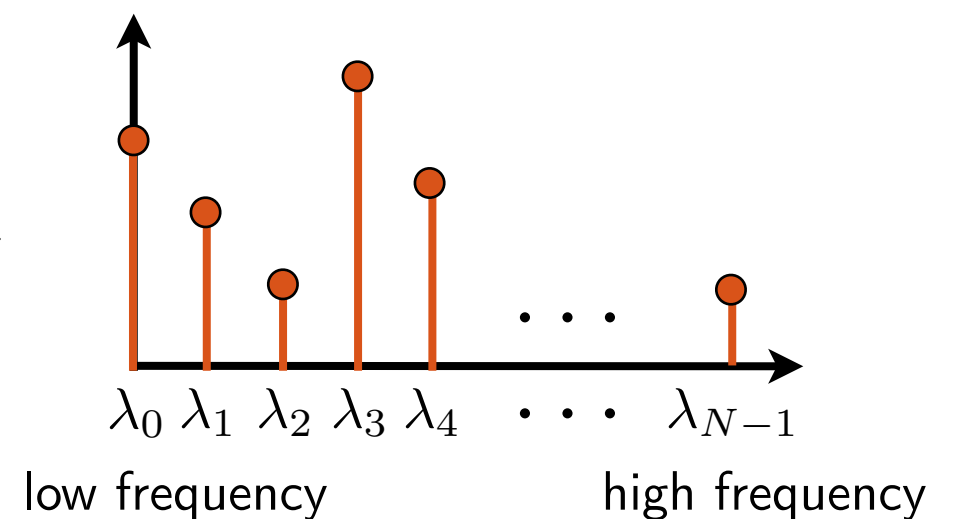
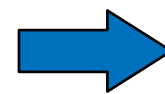
$$L = \chi \Lambda \chi^T$$

$$\chi_0^T L \chi_0 = \lambda_0 = 0$$

$$\chi_{50}^T L \chi_{50} = \lambda_{50}$$

graph Fourier transform:  
[Hammond11]

$$\hat{f}(\ell) = \langle \chi_\ell, f \rangle : \begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}^T \begin{bmatrix} | \\ f \\ | \end{bmatrix}$$



# Graph Fourier transform

- The Laplacian  $L$  admits the following eigendecomposition:  $L\chi_\ell = \lambda_\ell\chi_\ell$



# Graph Fourier transform

- The Laplacian  $L$  admits the following eigendecomposition:  $L\chi_\ell = \lambda_\ell\chi_\ell$

one-dimensional Laplace operator:  $-\nabla^2$



eigenfunctions:  $e^{j\omega x}$



Classical FT:  $\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx$

$$f(x) = \frac{1}{2\pi} \int \hat{f}(\omega) e^{j\omega x} d\omega$$

# Graph Fourier transform

- The Laplacian  $L$  admits the following eigendecomposition:  $L\chi_\ell = \lambda_\ell\chi_\ell$

one-dimensional Laplace operator:  $-\nabla^2$



eigenfunctions:  $e^{j\omega x}$



Classical FT:  $\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx$

$$f(x) = \frac{1}{2\pi} \int \hat{f}(\omega) e^{j\omega x} d\omega$$

graph Laplacian:  $L$



eigenvectors:  $\chi_\ell$



$f : V \rightarrow \mathbb{R}^N$

Graph FT:  $\hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i)$

$$f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$

# Graph Fourier transform

- The Laplacian  $L$  admits the following eigendecomposition:  $L\chi_\ell = \lambda_\ell\chi_\ell$

one-dimensional Laplace operator:  $-\nabla^2$



eigenfunctions:  $e^{j\omega x}$



Classical FT:  $\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx$

$$f(x) = \frac{1}{2\pi} \int \hat{f}(\omega) e^{j\omega x} d\omega$$

graph Laplacian:  $L$



eigenvectors:  $\chi_\ell$

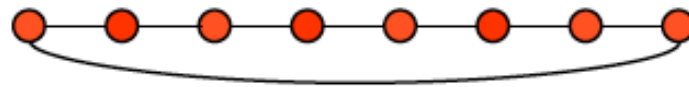


$f : V \rightarrow \mathbb{R}^N$

Graph FT:  $\hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i)$

$$f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$

# Two special cases



- (Unordered) Laplacian eigenvalues:  $\lambda_\ell = 2 - 2 \cos \left( \frac{2\ell\pi}{N} \right)$

- One possible choice of orthogonal Laplacian eigenvectors:

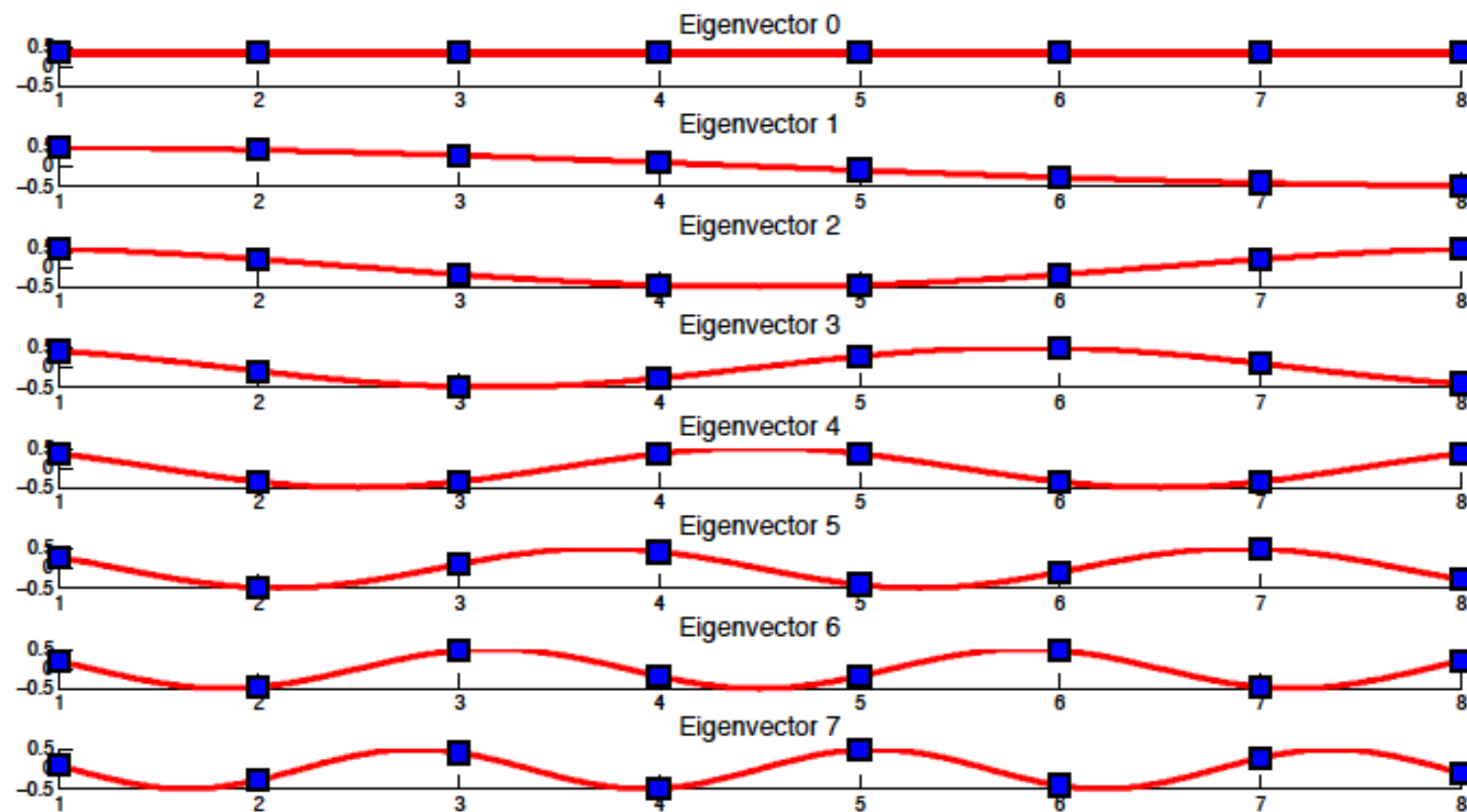
$$\chi_\ell = \left[ 1, \omega^\ell, \omega^{2\ell}, \dots, \omega^{(N-1)\ell} \right], \text{ where } \omega = e^{\frac{2\pi j}{N}}$$

- $\begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}$  is the Discrete Fourier Transform (DFT) matrix

# Two special cases



$$\lambda_\ell = 2 - 2 \cos\left(\frac{\pi \ell}{N}\right) \quad \chi_0(i) = \frac{1}{\sqrt{N}}, \quad \chi_\ell(i) = \sqrt{\frac{2}{N}} \cos\left(\frac{\pi \ell (i-0.5)}{N}\right), \quad \ell = 1, 2, \dots, N-1$$

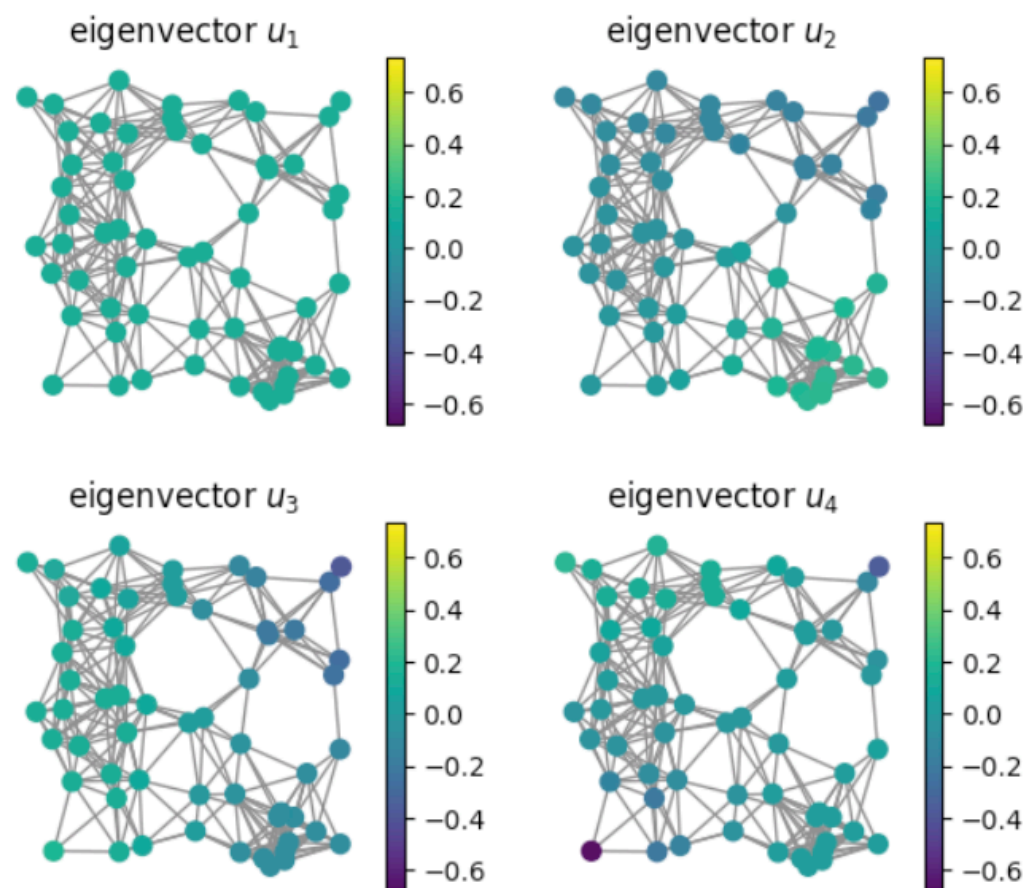


$$\begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}$$
 is the Discrete Cosine Transform matrix (DCT-II, Strang, 1999), which is used in JPEG image compression

# Graph Fourier transform

- Example on a simple graph

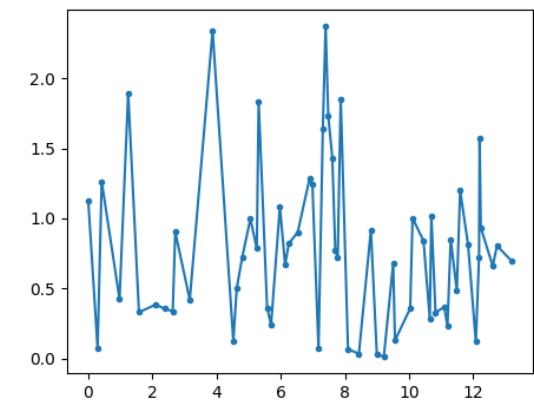
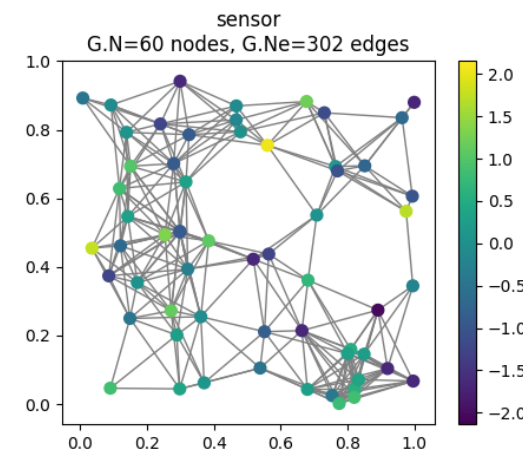
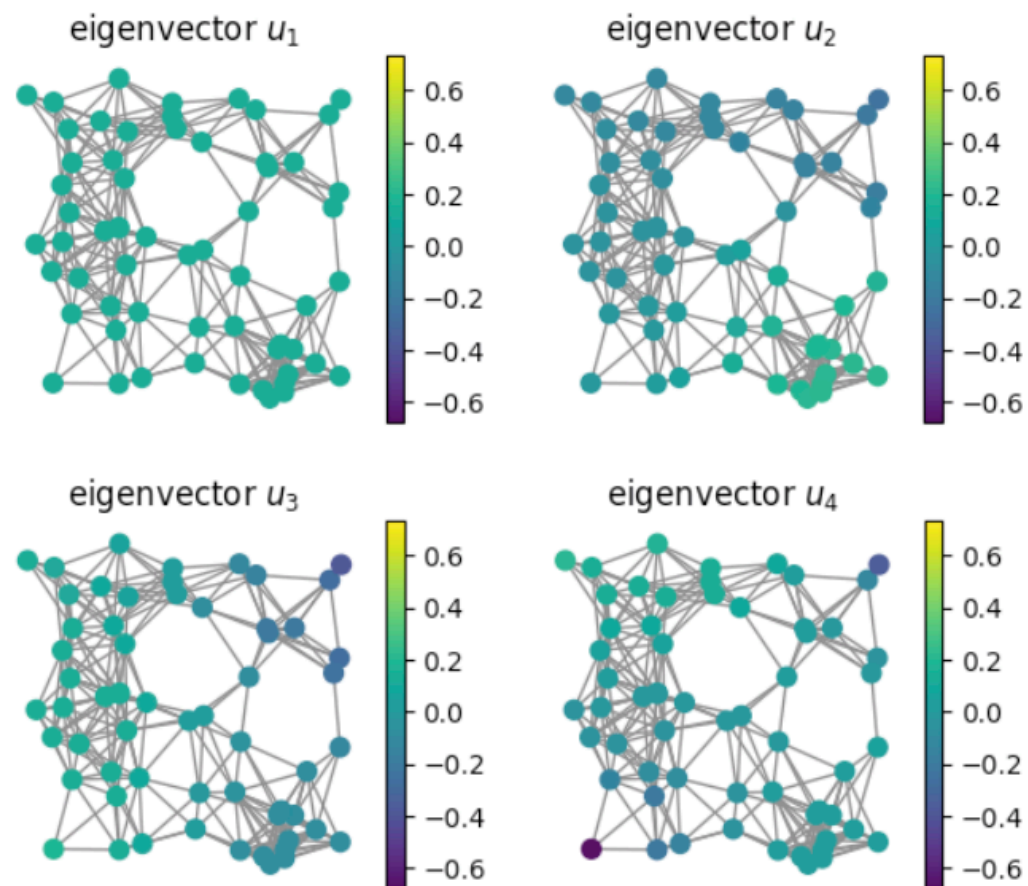
$$\hat{f}(\ell) = \langle \chi_\ell, f \rangle : \begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}^T \begin{bmatrix} | \\ f \\ | \end{bmatrix}$$



# Graph Fourier transform

- Example on a simple graph

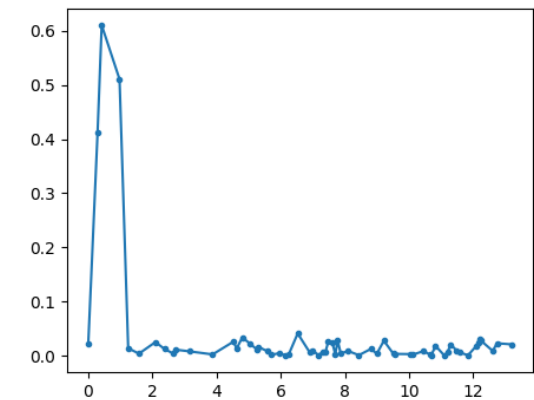
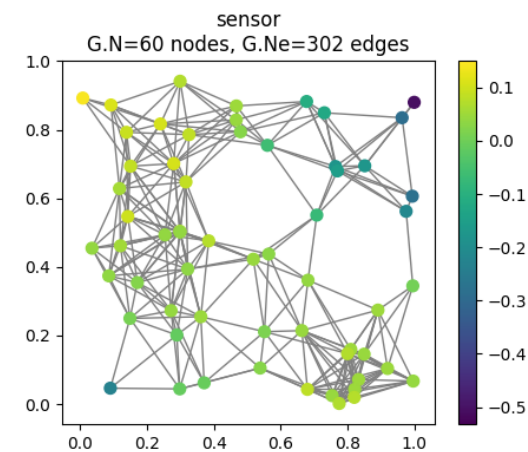
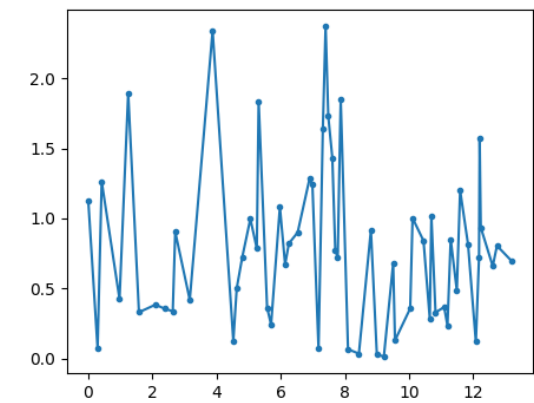
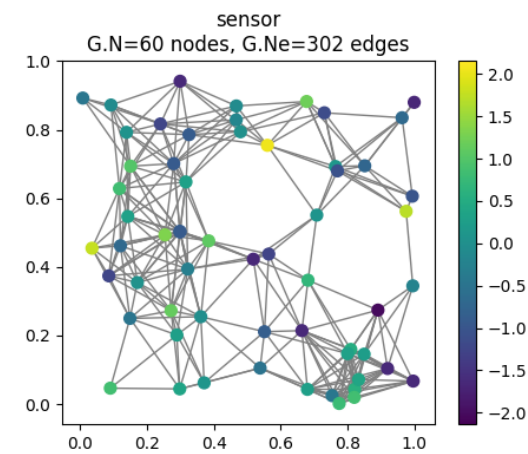
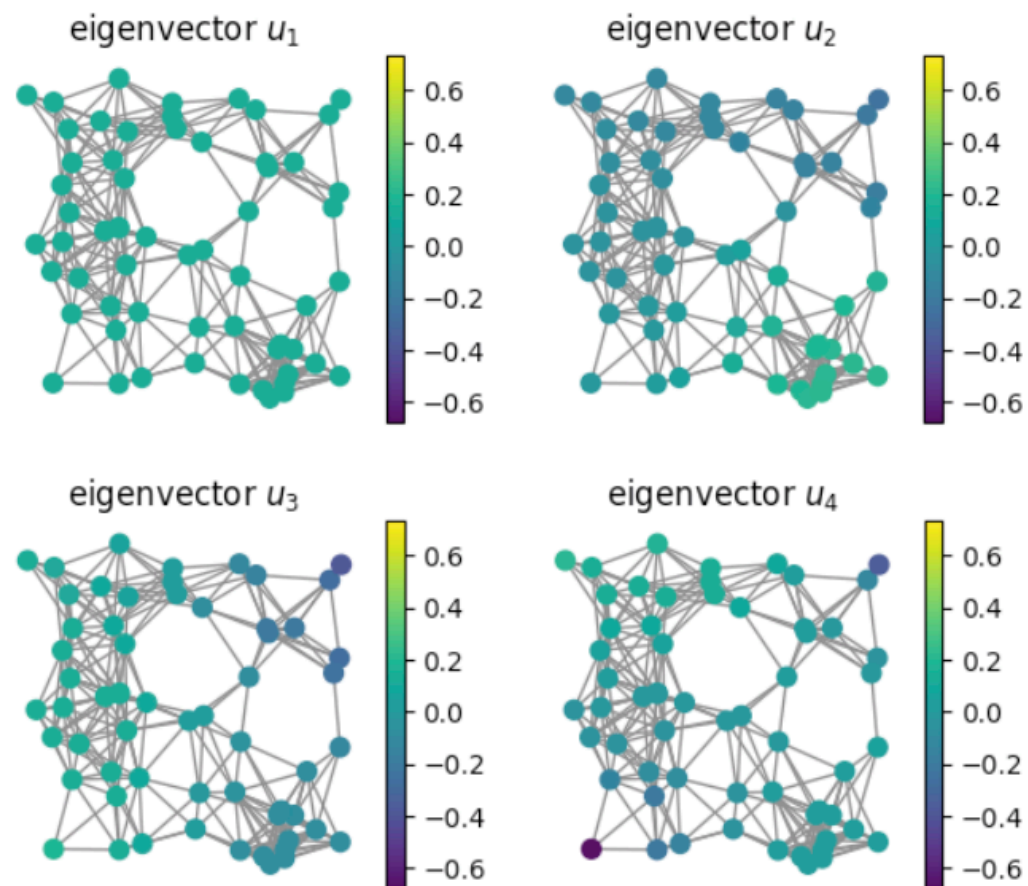
$$\hat{f}(\ell) = \langle \chi_\ell, f \rangle : \begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}^T \begin{bmatrix} | \\ f \\ | \end{bmatrix}$$



# Graph Fourier transform

- Example on a simple graph

$$\hat{f}(\ell) = \langle \chi_\ell, f \rangle : \begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}^T \begin{bmatrix} | \\ f \\ | \end{bmatrix}$$





# Outline

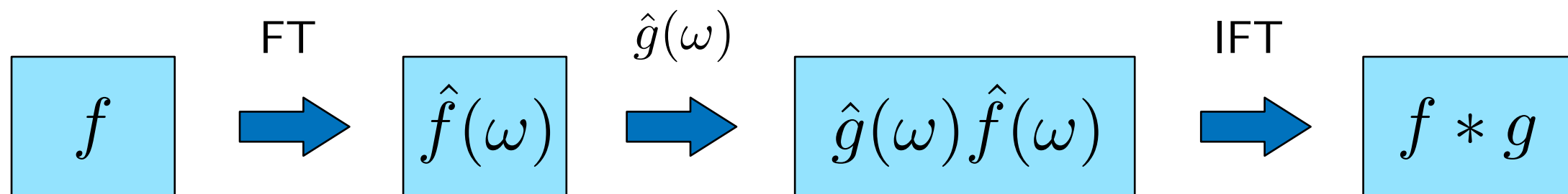
- Graph signal processing (GSP): Basic concepts
- Graph spectral filtering: Basic tool of GSP
- Graph neural networks (GNNs)
- Applications

# Classical frequency filtering

Classical FT:  $\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx \quad f(x) = \frac{1}{2\pi} \int \hat{f}(\omega) e^{j\omega x} d\omega$

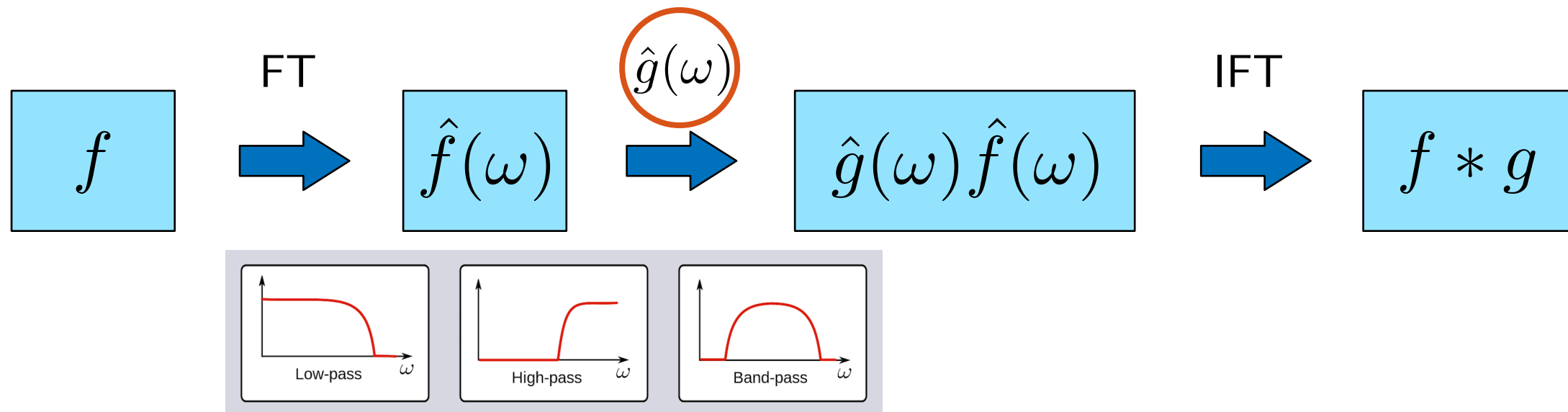
# Classical frequency filtering

Classical FT:  $\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx$      $f(x) = \frac{1}{2\pi} \int \hat{f}(\omega) e^{j\omega x} d\omega$



# Classical frequency filtering

Classical FT:  $\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx$      $f(x) = \frac{1}{2\pi} \int \hat{f}(\omega) e^{j\omega x} d\omega$

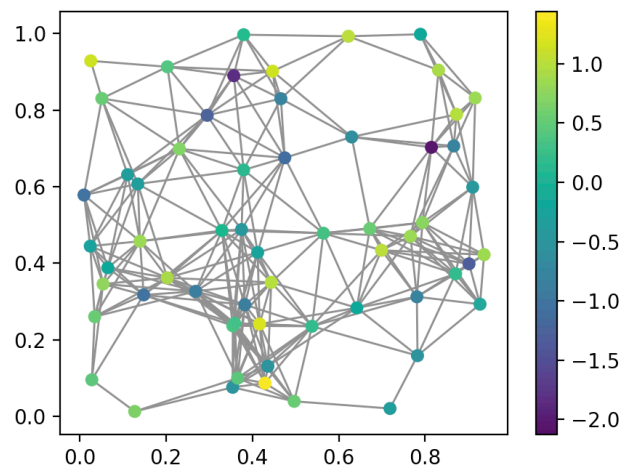
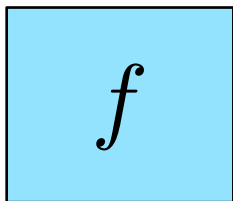


# Graph spectral filtering

$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$

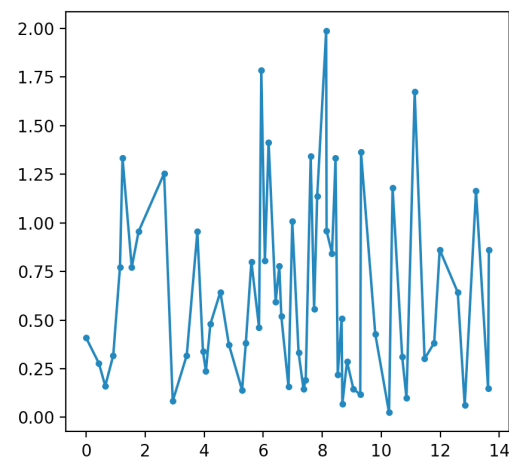
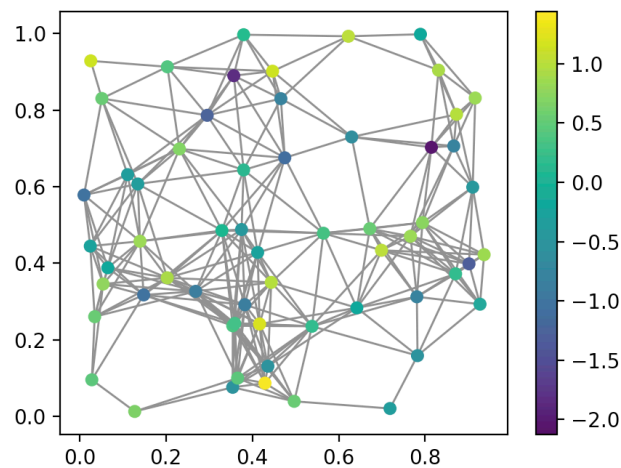
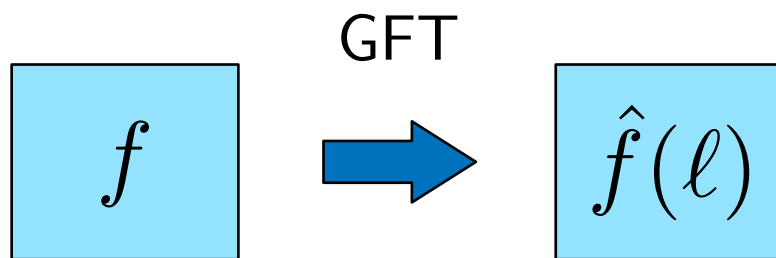
# Graph spectral filtering

$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$



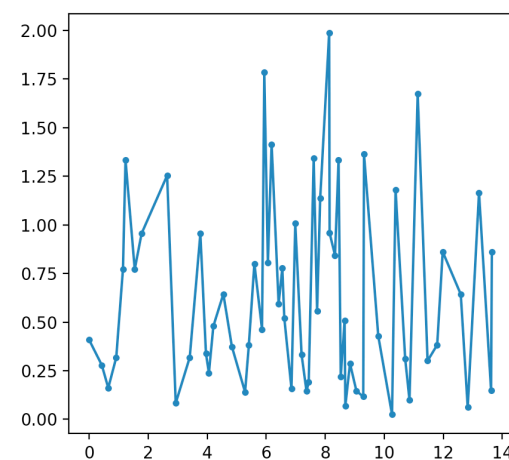
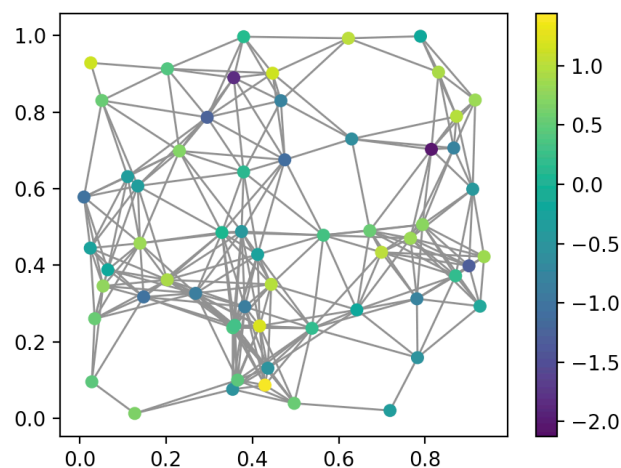
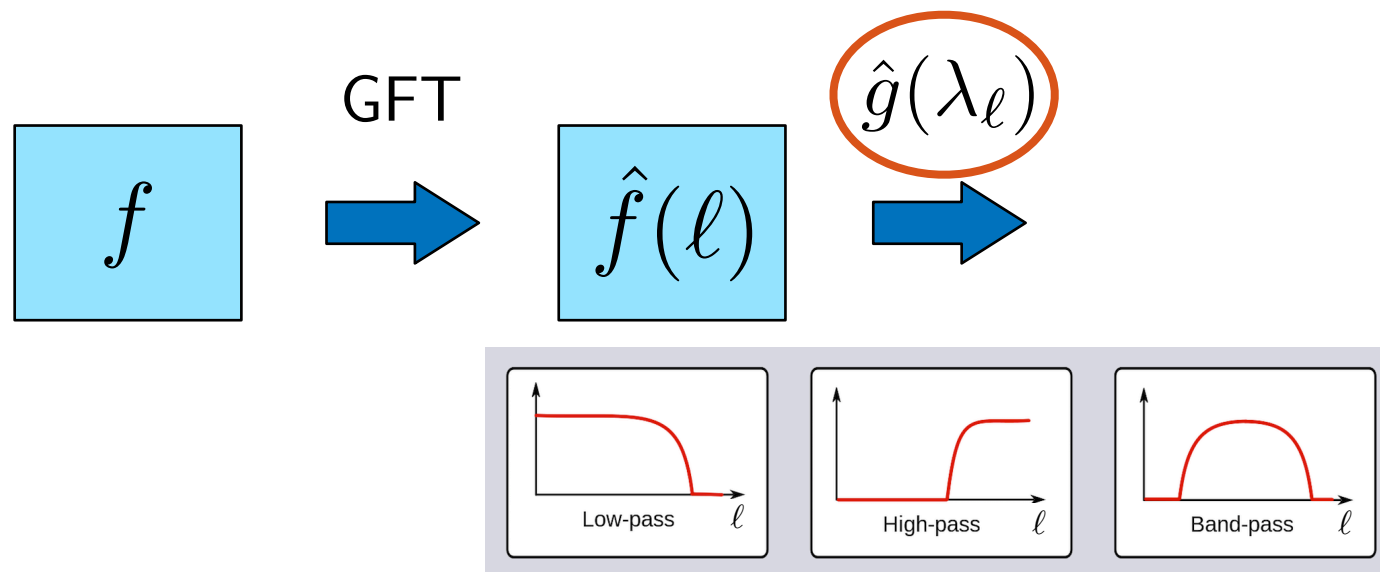
# Graph spectral filtering

$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$



# Graph spectral filtering

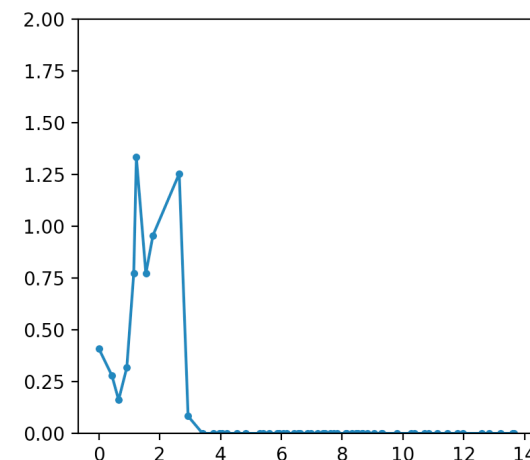
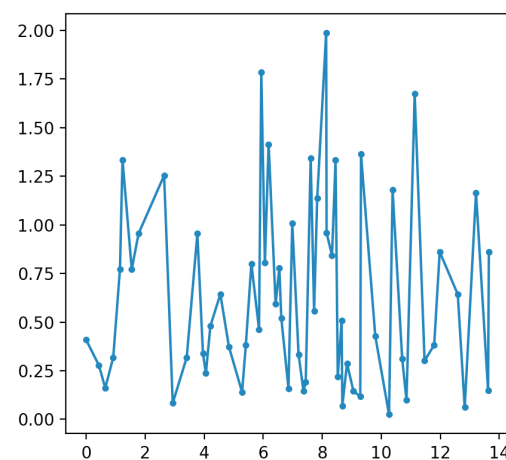
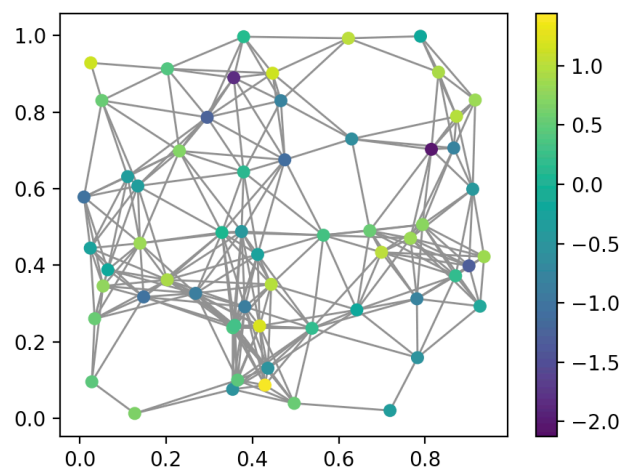
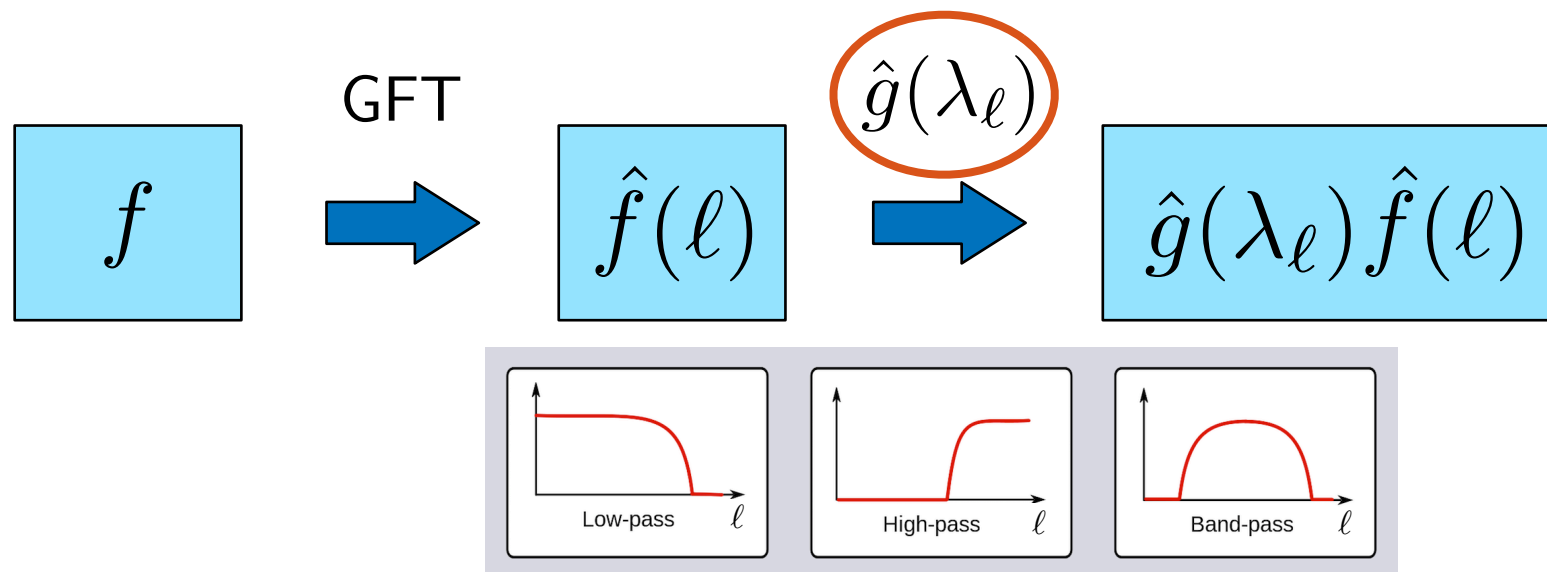
$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$





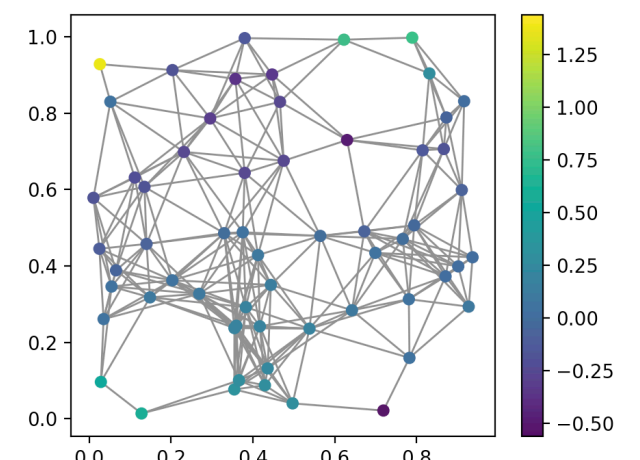
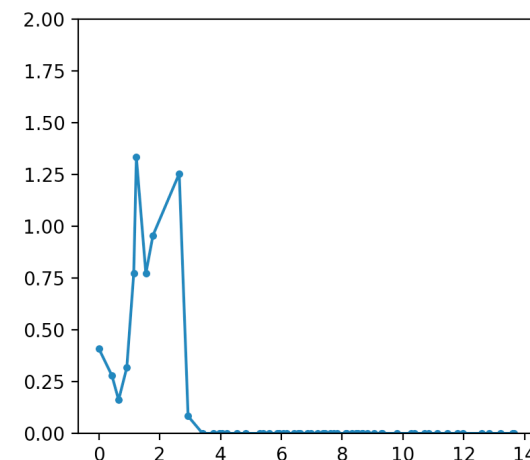
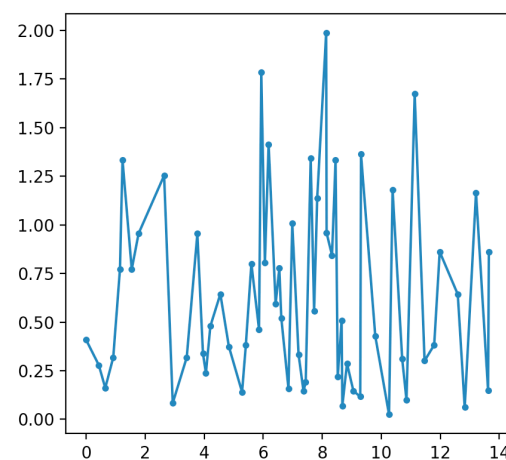
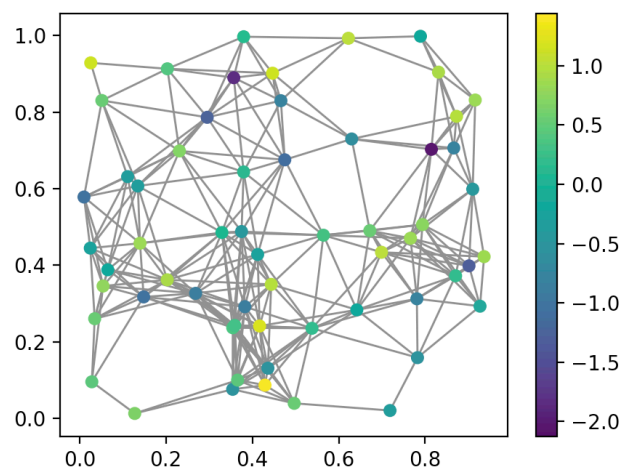
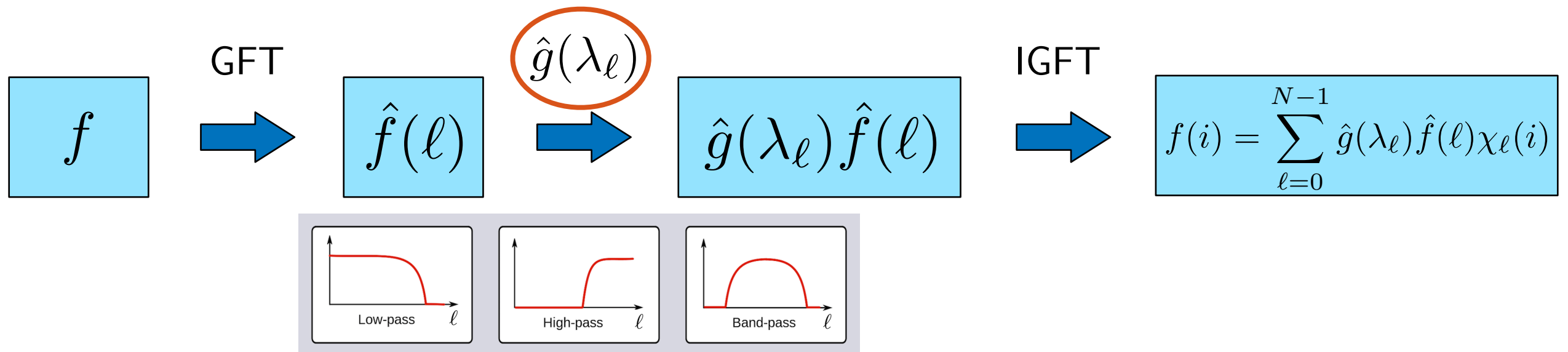
# Graph spectral filtering

$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$



# Graph spectral filtering

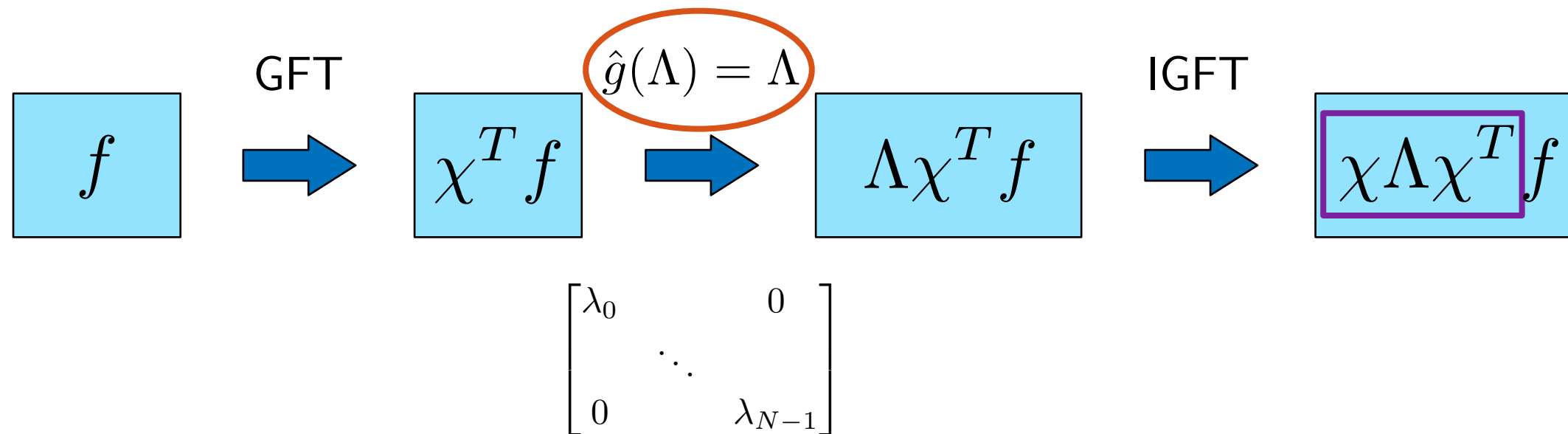
$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$



# Graph Laplacian revisited

$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$

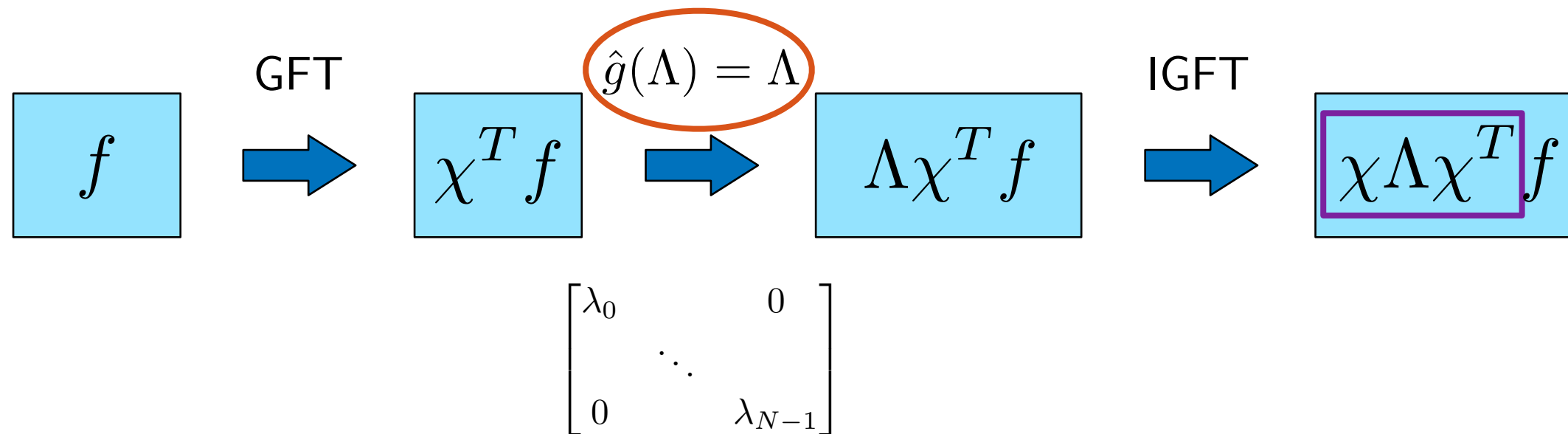
The Laplacian  $L$  is a difference operator:  $Lf = \chi \Lambda \chi^T f$



# Graph Laplacian revisited

$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$

The Laplacian  $L$  is a difference operator:  $Lf = \chi \Lambda \chi^T f$

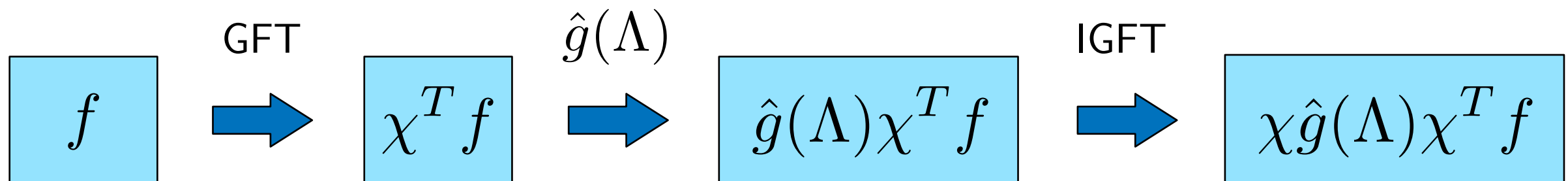


The Laplacian operator filters the signal in the spectral domain by its eigenvalues!

The Laplacian quadratic form:  $f^T L f = \|L^{\frac{1}{2}} f\|_2 = \|\chi \Lambda^{\frac{1}{2}} \chi^T f\|_2$

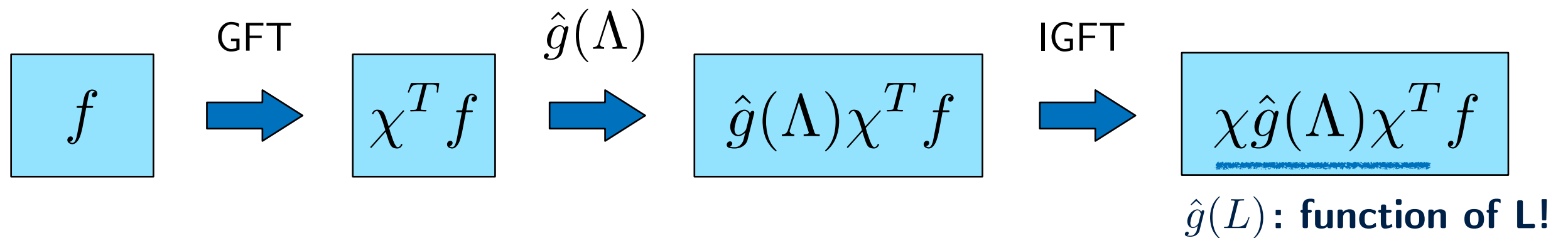
# Graph transform/dictionary design

- Transforms and dictionaries can be designed through graph spectral filtering: Functions of graph Laplacian!



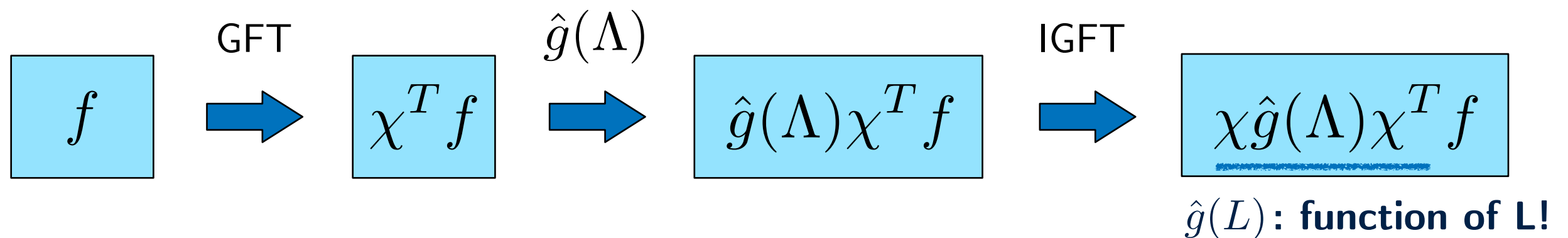
# Graph transform/dictionary design

- Transforms and dictionaries can be designed through graph spectral filtering: Functions of graph Laplacian!



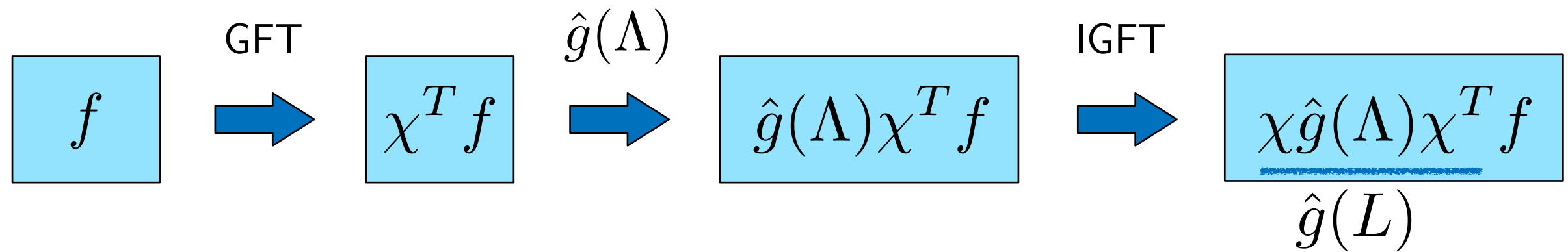
# Graph transform/dictionary design

- Transforms and dictionaries can be designed through graph spectral filtering: Functions of graph Laplacian!



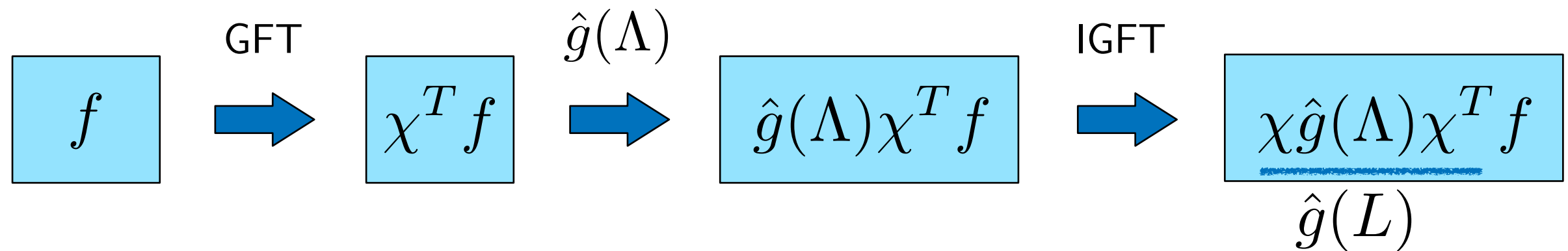
- Important properties can be achieved by properly defining  $\hat{g}(L)$ , such as localisation of atoms
- Closely related to kernels and regularisation on graphs

# A practical example





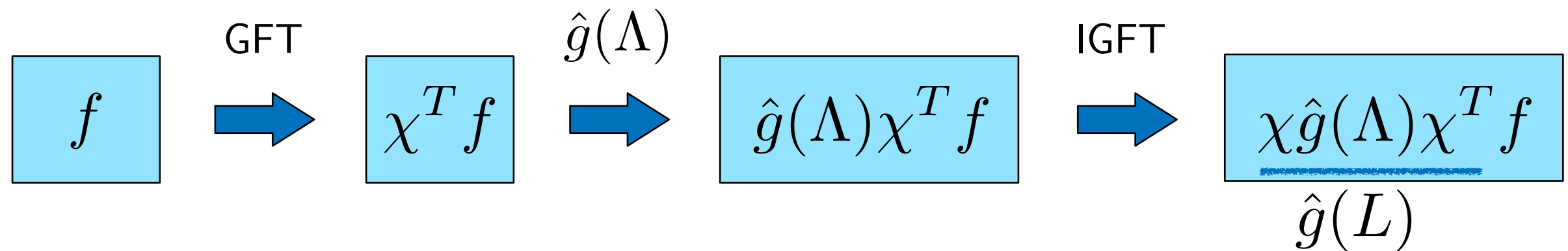
# A practical example



problem: we observe a noisy graph signal  $f = y_0 + \eta$  and wish to recover  $y_0$

$$y^* = \arg \min_y \{ \|y - f\|_2^2 + \gamma y^T L y \}$$

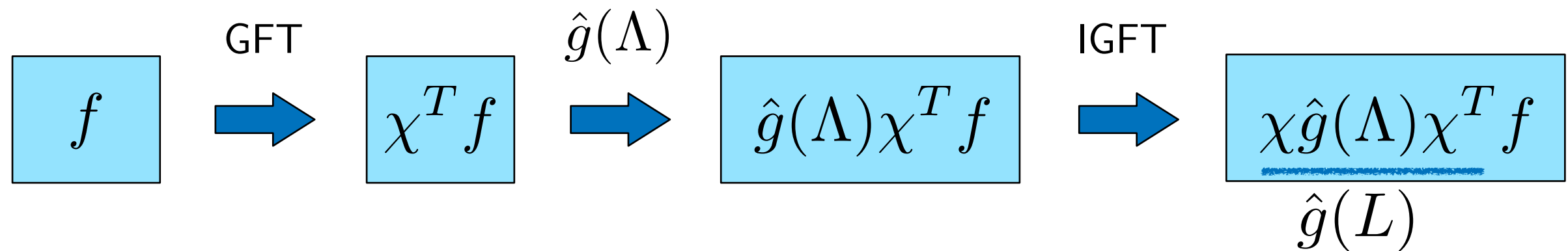
# A practical example



problem: we observe a noisy graph signal  $f = y_0 + \eta$  and wish to recover  $y_0$

$$y^* = \arg \min_y \{ \underbrace{\|y - f\|_2^2}_{\text{data fitting term}} + \underbrace{\gamma y^T L y}_{\text{"smoothness" assumption}} \}$$

# A practical example

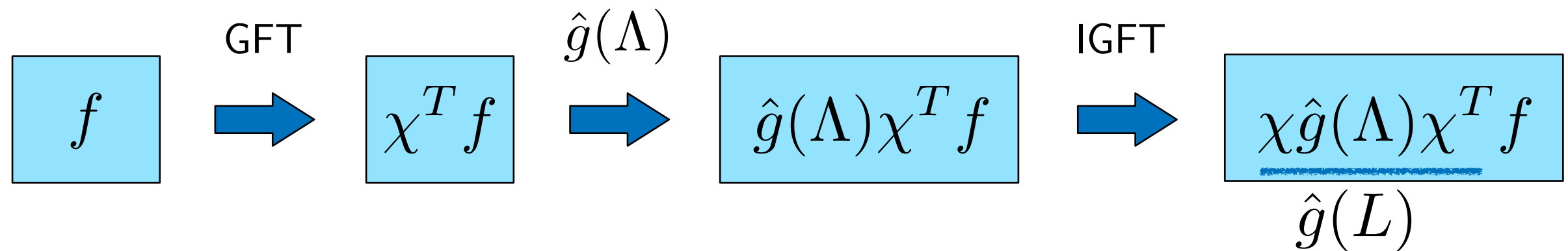


problem: we observe a noisy graph signal  $f = y_0 + \eta$  and wish to recover  $y_0$

$$y^* = \arg \min_y \{ \underbrace{\|y - f\|_2^2}_{\text{data fitting term}} + \gamma \underbrace{y^T L y}_{\text{"smoothness" assumption}} \}$$

$\downarrow$   
 $y^* = \underbrace{(I + \gamma L)^{-1} f}_{\hat{g}(L)}$

# A practical example



problem: we observe a noisy graph signal  $f = y_0 + \eta$  and wish to recover  $y_0$

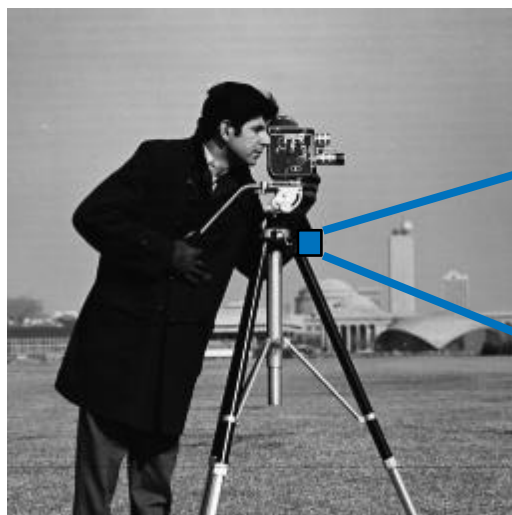
$$y^* = \arg \min_y \{ \underbrace{\|y - f\|_2^2}_{\text{data fitting term}} + \underbrace{\gamma y^T L y}_{\text{"smoothness" assumption}} \}$$

$$y^* = \underbrace{(I + \gamma L)^{-1} f}_{\hat{g}(L)} = \chi (1 + \gamma \Lambda)^{-1} \chi^T f$$

**remove noise by low-pass filtering  
in graph spectral domain!**

# A practical example

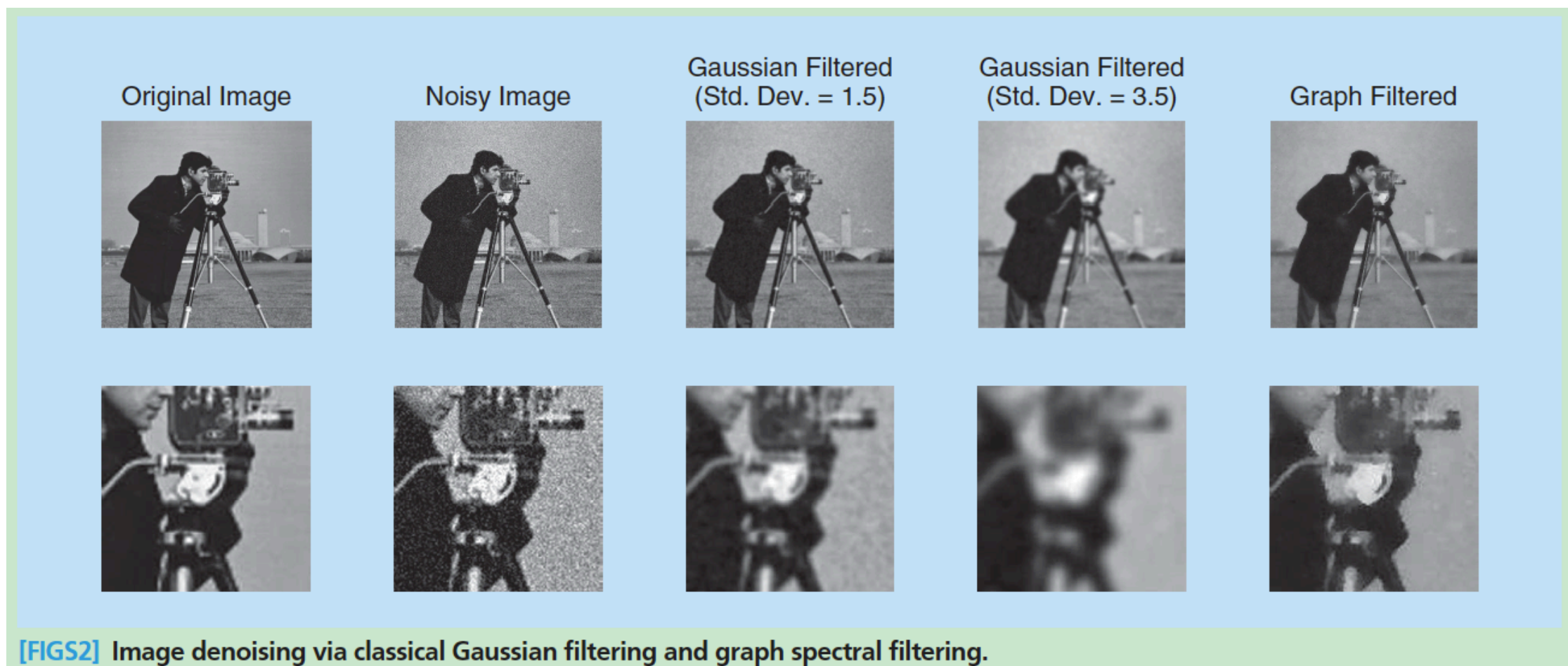
- noisy image as observed noisy graph signal
- regular grid graph (weights inversely proportional to pixel value difference)



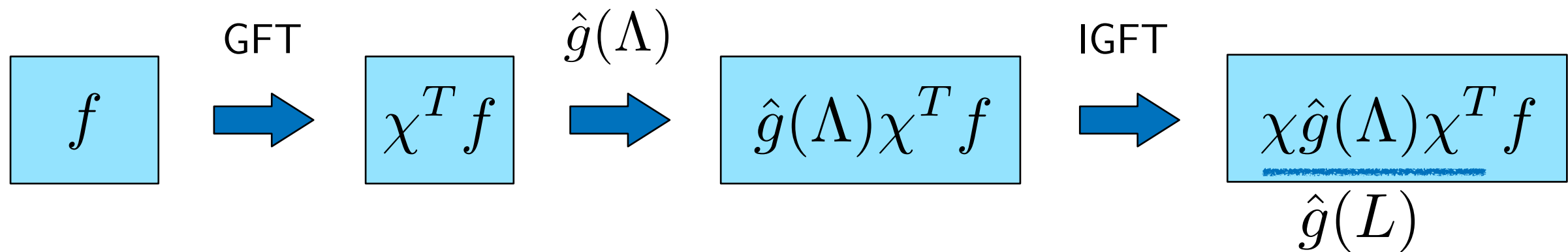
$$w_{ij} = \frac{1}{|f(i) - f(j)|}$$

# A practical example

- noisy image as observed noisy graph signal
- regular grid graph (weights inversely proportional to pixel value difference)



# More filtering operations



low-pass filters:  $\hat{g}(L) = (I + \gamma L)^{-1} = \chi(I + \gamma \Lambda)^{-1} \chi^T$

window kernel: windowed graph Fourier transform

shifted and dilated band-pass filters: spectral graph wavelets  $\hat{g}(sL)$

adapted kernels: learn values of  $\hat{g}(L)$  directly from data

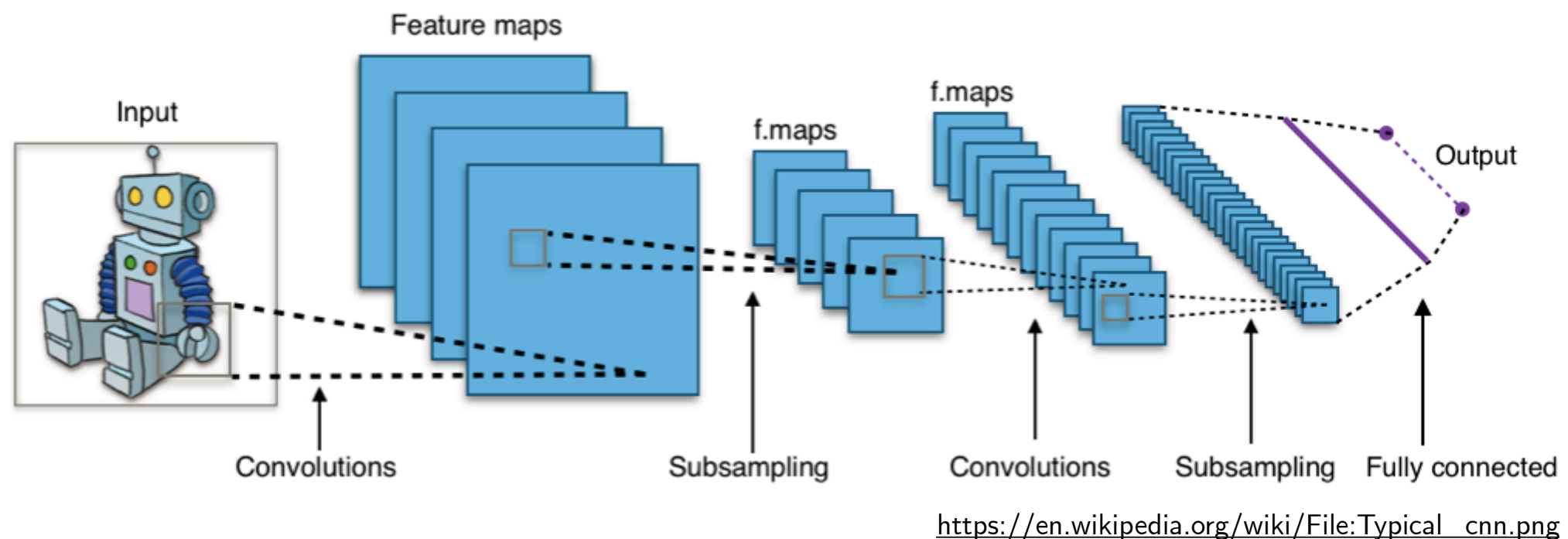
parametric polynomials:  $\hat{g}_s(L) = \sum_{k=0}^K \alpha_{sk} L^k = \chi \left( \sum_{k=0}^K \alpha_{sk} \Lambda^k \right) \chi^T$

# Outline

- Graph signal processing (GSP): Basic concepts
- Graph spectral filtering: Basic tool of GSP
- Graph neural networks (GNNs)
- Applications

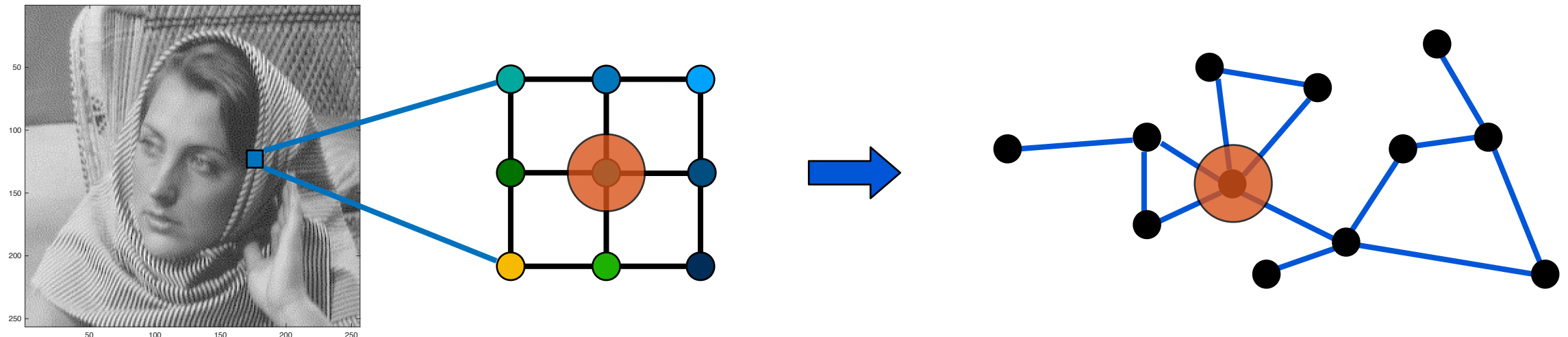


# CNNs exploit structure within data



- checklist
  - **convolution:** translation equivariance
  - **localisation:** compact filters (independent of sample dimension)
  - **multi-scale:** compositionality
  - **efficiency:**  $\mathcal{O}(N)$  computational complexity

# CNNs on graphs?



- checklist
  - **convolution:** how to do it on graphs?
  - **localisation:** what's the notion of locality?
  - **multi-scale:** how to down-sample on graphs?
  - **efficiency:** how to keep the computational complexity low?

# Convolution on graphs

classical convolution

time domain

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau$$

frequency domain

$$\widehat{(f * g)}(\omega) = \hat{f}(\omega) \cdot \hat{g}(\omega)$$

convolution on graphs

?

# Convolution on graphs

## classical convolution

time domain

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau$$

frequency domain

$$\widehat{(f * g)}(\omega) = \hat{f}(\omega) \cdot \hat{g}(\omega)$$

## convolution on graphs

graph spectral domain

$$\widehat{(f * g)}(\lambda) = ((\chi^T f) \circ \hat{g})(\lambda)$$



# Convolution on graphs

## classical convolution

time domain

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau$$

frequency domain

$$\widehat{(f * g)}(\omega) = \hat{f}(\omega) \cdot \hat{g}(\omega)$$

## convolution on graphs

node domain

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



graph spectral domain

$$\widehat{(f * g)}(\lambda) = ((\chi^T f) \circ \hat{g})(\lambda)$$



# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



parametric filter as polynomial of eigenvalues

$$\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \quad \theta \in \mathbb{R}^{K+1} \quad \longrightarrow \quad \hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$$

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



parametric filter as polynomial of eigenvalues

$$\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \quad \theta \in \mathbb{R}^{K+1}$$



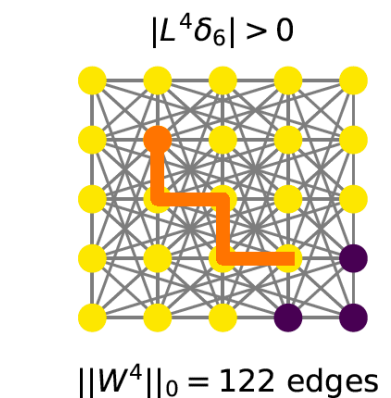
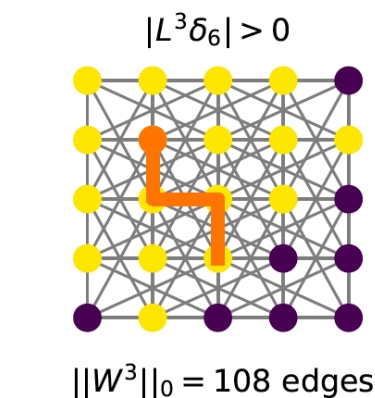
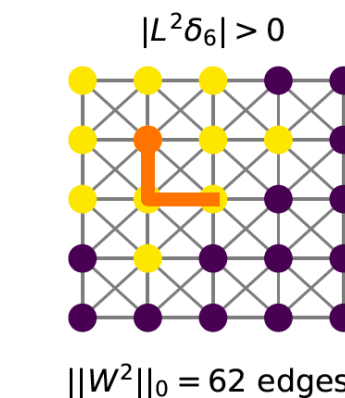
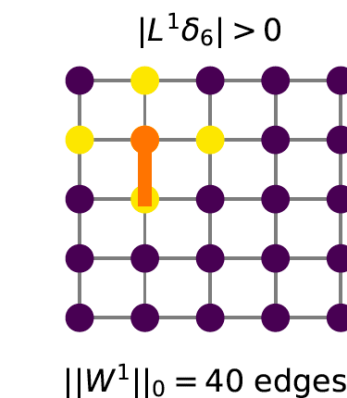
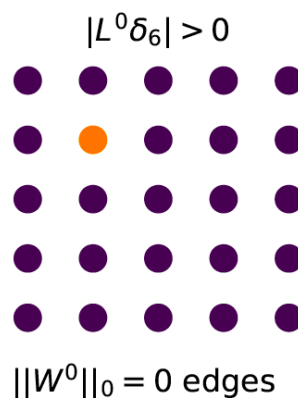
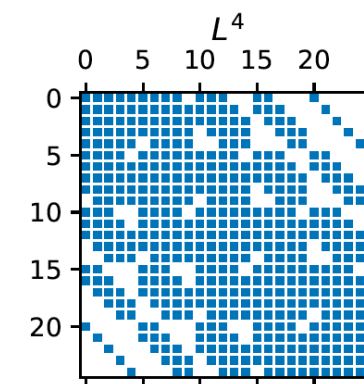
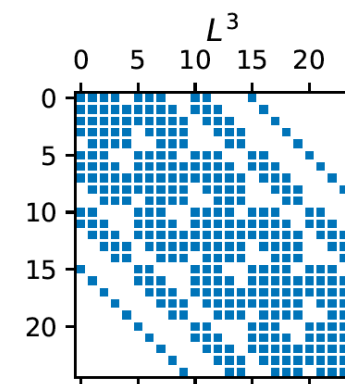
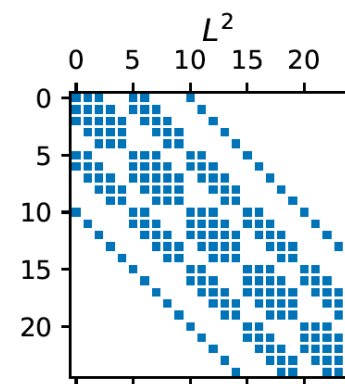
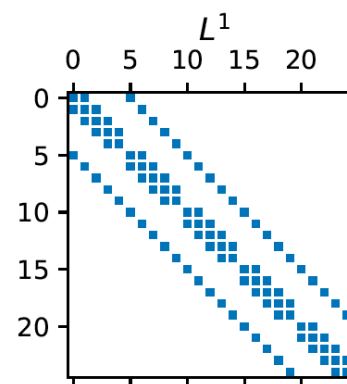
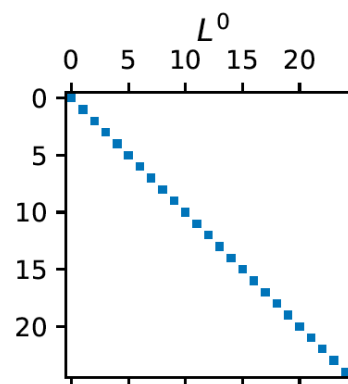
$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$$

what do powers of graph Laplacian capture?



# Powers of graph Laplacian

$L^k$  defines the  $k$ -neighborhood



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

(slides by Michaël Deferrard)

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

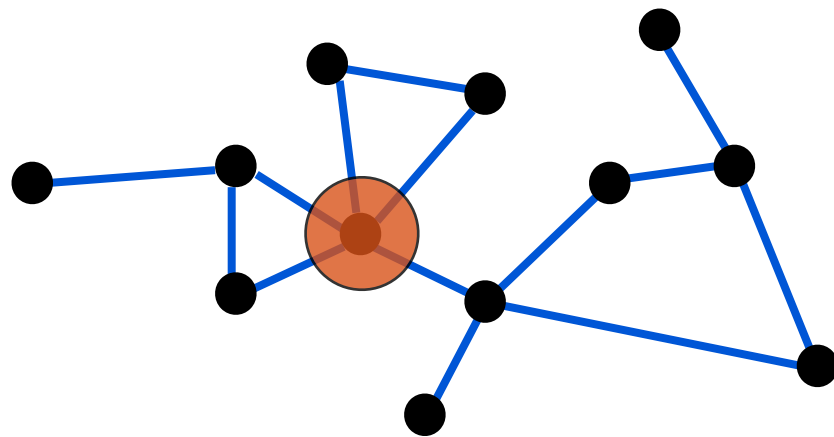


parametric filter as polynomial of eigenvalues

$$\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \quad \theta \in \mathbb{R}^{K+1}$$



$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$$



(localisation within **K-hop** neighbourhood)

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



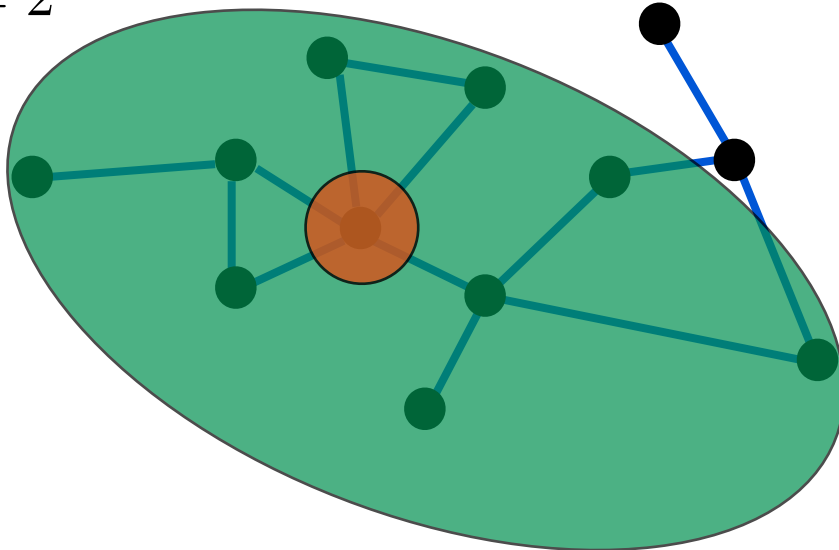
parametric filter as polynomial of eigenvalues

$$\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \quad \theta \in \mathbb{R}^{K+1}$$



$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$$

$K = 2$



(localisation within **K-hop** neighbourhood)

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



parametric filter as polynomial of eigenvalues

$$\hat{g}_\theta(\lambda) = \sum_{j=0}^K \theta_j \lambda^j, \quad \theta \in \mathbb{R}^{K+1} \quad \longrightarrow \quad \hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$$

- checklist
  - **convolution:** expressed in the graph spectral domain
  - **localisation:** within K-hop neighbourhood
  - **learning complexity:**  $\mathcal{O}(K)$
  - **computational complexity:**  $\mathcal{O}(K|\mathcal{E}|)$  , no need for GFT

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



fast and stable implementation by Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L}) \quad \text{with a scaled Laplacian} \quad \tilde{L} = \frac{2}{\lambda_{N-1}} L - I$$

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



fast and stable implementation by Chebyshev approximation

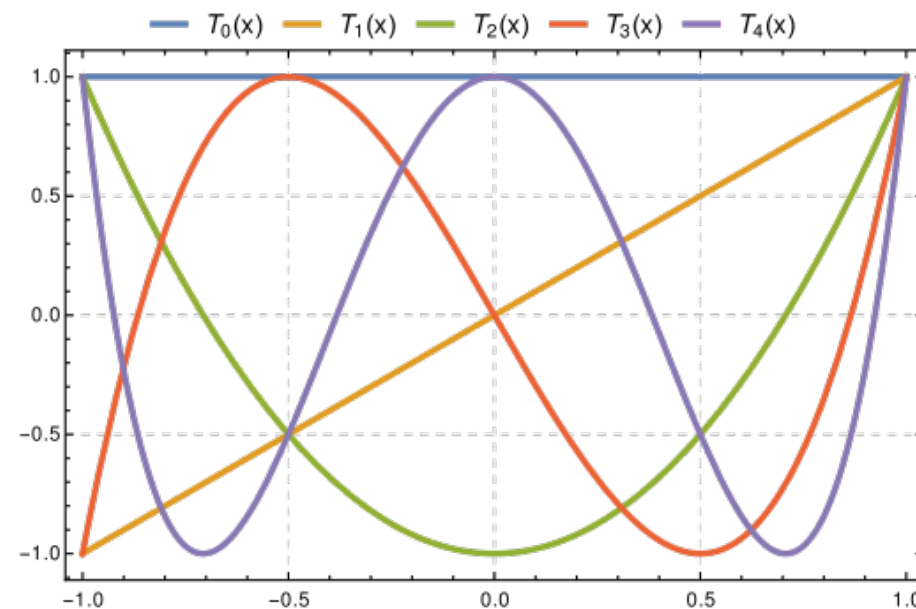
$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L}) \quad \text{with a scaled Laplacian} \quad \tilde{L} = \frac{2}{\lambda_{N-1}} L - I$$

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

- recursively defined
- orthogonal basis for  $L^2([-1, 1], dy/\sqrt{1-y^2})$



# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



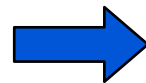
fast and stable implementation by Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L}) \quad \text{with a scaled Laplacian} \quad \tilde{L} = \frac{2}{\lambda_{N-1}} L - I$$

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$



$$T_0(\tilde{L}) = I$$

$$T_1(\tilde{L}) = \tilde{L}$$

$$T_k(\tilde{L}) = 2\tilde{L} T_{k-1}(\tilde{L}) - T_{k-2}(\tilde{L})$$

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



simplified Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L})$$



# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



simplified Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L}) \quad \xrightarrow{K=1} \quad \theta_0 I + \theta_1 (\tilde{L} - I)$$

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



simplified Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L})$$

$$\xrightarrow{K=1} \theta_0 I + \theta_1 (\tilde{L} - I)$$

$$\xrightarrow{\lambda_{N-1} \approx 2} = \theta_0 I - \theta_1 (D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$

(localisation within **1-hop** neighbourhood)

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



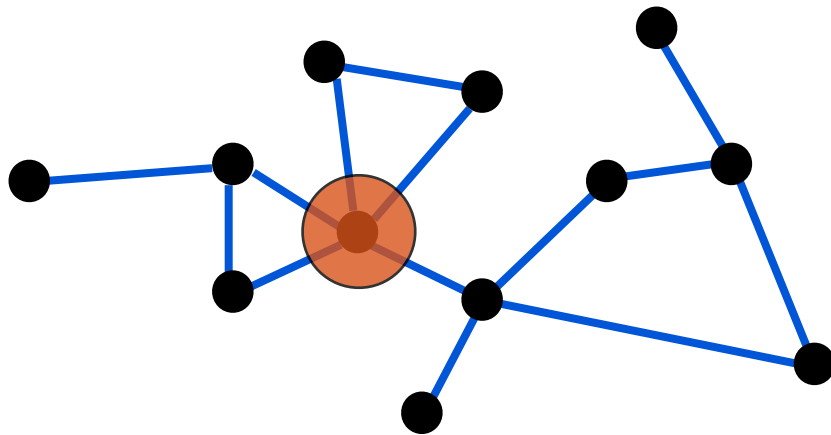
simplified Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L})$$

$$\xrightarrow{K=1} \theta_0 I + \theta_1 (\tilde{L} - I)$$

$$\xrightarrow{\lambda_{N-1} \approx 2} = \theta_0 I - \theta_1 (D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$

(localisation within **1-hop** neighbourhood)



# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



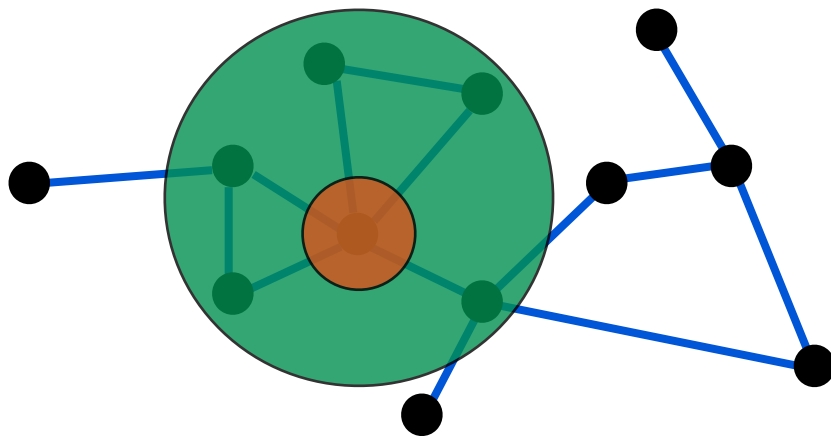
simplified Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L})$$

$$\xrightarrow{K=1} \theta_0 I + \theta_1 (\tilde{L} - I)$$

$$\xrightarrow{\lambda_{N-1} \approx 2} = \theta_0 I - \theta_1 (D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$

(localisation within **1-hop** neighbourhood)



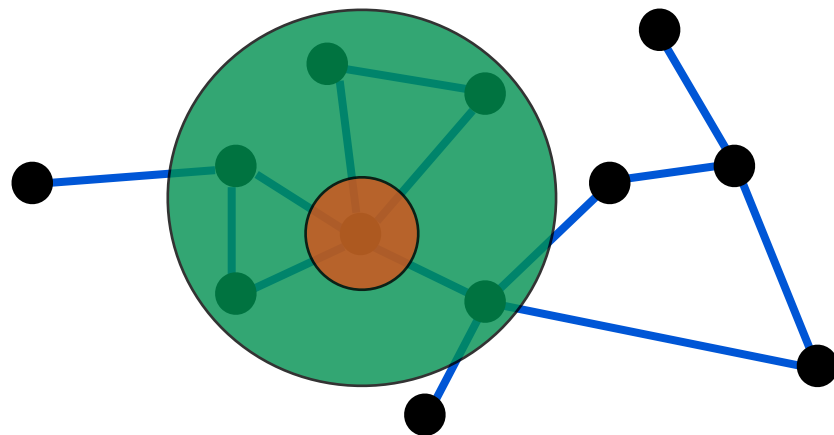
# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



simplified Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L})$$



$$\begin{aligned} &\xrightarrow{K=1} \theta_0 I + \theta_1 (\tilde{L} - I) \\ &\xrightarrow{\lambda_{N-1} \approx 2} = \theta_0 I - \theta_1 (D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) \end{aligned}$$

(localisation within **1-hop** neighbourhood)

$$\begin{aligned} &\alpha = \theta_0 = -\theta_1 \\ &\xrightarrow{\quad} = \alpha (I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) \end{aligned}$$

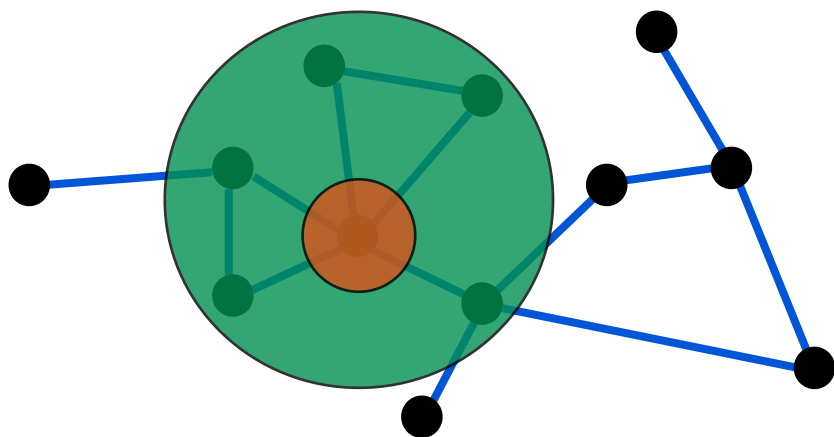
# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



simplified Chebyshev approximation

$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j T_j(\tilde{L})$$



$$\begin{aligned} &\xrightarrow{K=1} \theta_0 I + \theta_1 (\tilde{L} - I) \\ &\xrightarrow{\lambda_{N-1} \approx 2} = \theta_0 I - \theta_1 (D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) \end{aligned}$$

(localisation within **1-hop** neighbourhood)

$$\begin{aligned} &\alpha = \theta_0 = -\theta_1 \\ &\xrightarrow{\quad} = \alpha (I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) \end{aligned}$$

$$\begin{aligned} &\text{renormalisation} \\ &\xrightarrow{\quad} = \alpha (\tilde{D}^{-\frac{1}{2}} \tilde{W} \tilde{D}^{-\frac{1}{2}}) \end{aligned}$$

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



simplified Chebyshev approximation: too simple?

$$\hat{g}_\alpha(L) = \alpha(I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$

# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

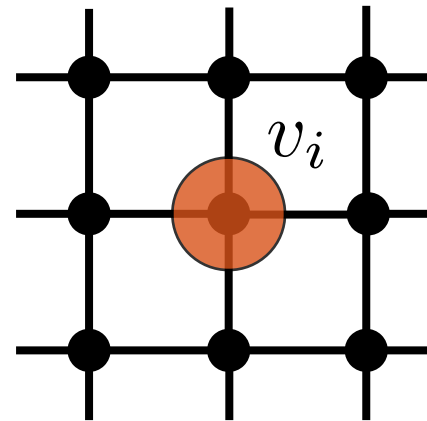


simplified Chebyshev approximation: too simple?

$$\hat{g}_\alpha(L) = \alpha(I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$



$$y_i = \alpha f_i + \alpha \frac{1}{\sqrt{d_i}} \sum_{j:(i,j) \in \mathcal{E}} w_{ij} \frac{1}{\sqrt{d_j}} f_j$$





# Convolution on graphs

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$



simplified Chebyshev approximation: too simple?

$$\hat{g}_\alpha(L) = \alpha(I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$

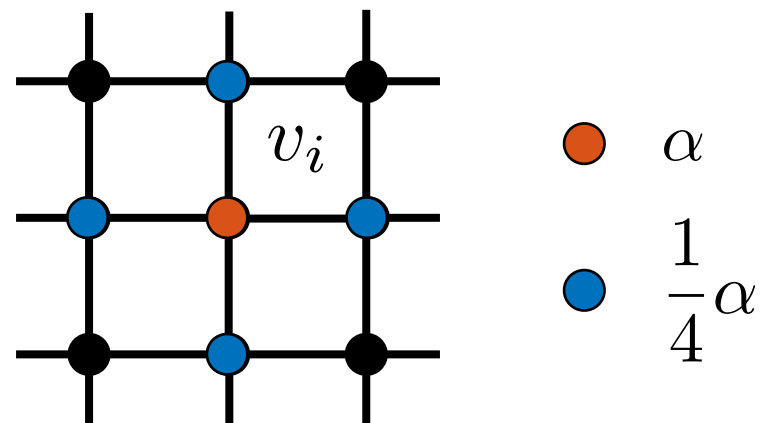


$$y_i = \alpha f_i + \alpha \frac{1}{\sqrt{d_i}} \sum_{j:(i,j) \in \mathcal{E}} w_{ij} \frac{1}{\sqrt{d_j}} f_j$$



unitary edge weights

$$y_i = \alpha f_i + \frac{1}{4} \alpha \sum_{j:(i,j) \in \mathcal{E}} f_j$$



# Convolution on graphs - Summary

- Convolution is defined via the **graph spectral** domain...

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

# Convolution on graphs - Summary

- Convolution is defined via the **graph spectral** domain...

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

- ...but can be implemented in the **spatial (vertex)** domain

$$y = \hat{g}_\theta(L) f = \sum_{j=0}^K \theta_j T_j(\tilde{L}) f$$

# Convolution on graphs - Summary

- Convolution is defined via the **graph spectral** domain...

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

- ...but can be implemented in the **spatial (vertex)** domain

$$y = \hat{g}_\theta(L) f = \sum_{j=0}^K \theta_j T_j(\tilde{L}) f \quad \rightarrow \quad \text{simple averaging in Kipf and Welling 2017}$$

# Convolution on graphs - Summary

- Convolution is defined via the **graph spectral** domain...

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

- ...but can be implemented in the **spatial (vertex)** domain

$$y = \hat{g}_\theta(L) f = \sum_{j=0}^K \theta_j T_j(\tilde{L}) f \quad \rightarrow \quad \text{simple averaging in Kipf and Welling 2017}$$

- Forward pass in spectral graph CNNs

$$\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L) f) \right)$$

# Convolution on graphs - Summary

- Convolution is defined via the **graph spectral** domain...

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

- ...but can be implemented in the **spatial (vertex)** domain

$$y = \hat{g}_\theta(L) f = \sum_{j=0}^K \theta_j T_j(\tilde{L}) f \quad \rightarrow \quad \text{simple averaging in Kipf and Welling 2017}$$

- Forward pass in spectral graph CNNs

$$\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L) f) \right)$$

# Convolution on graphs - Summary

- Convolution is defined via the **graph spectral** domain...

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

- ...but can be implemented in the **spatial (vertex)** domain

$$y = \hat{g}_\theta(L) f = \sum_{j=0}^K \theta_j T_j(\tilde{L}) f \quad \rightarrow \quad \text{simple averaging in Kipf and Welling 2017}$$

- Forward pass in spectral graph CNNs

$$\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L) f) \right)$$

# Convolution on graphs - Summary

- Convolution is defined via the **graph spectral** domain...

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

- ...but can be implemented in the **spatial (vertex)** domain

$$y = \hat{g}_\theta(L) f = \sum_{j=0}^K \theta_j T_j(\tilde{L}) f \quad \rightarrow \quad \text{simple averaging in Kipf and Welling 2017}$$

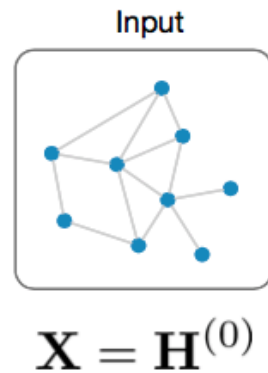
- Forward pass in spectral graph CNNs

$$\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L) f) \right)$$



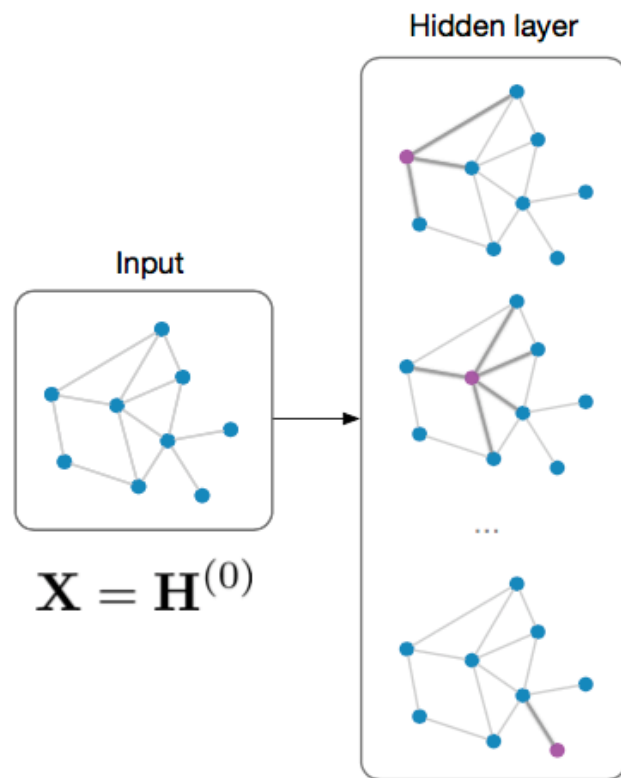
# Graph convolutional networks (GCN)

- Forward pass:  $\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L)f) \right)$



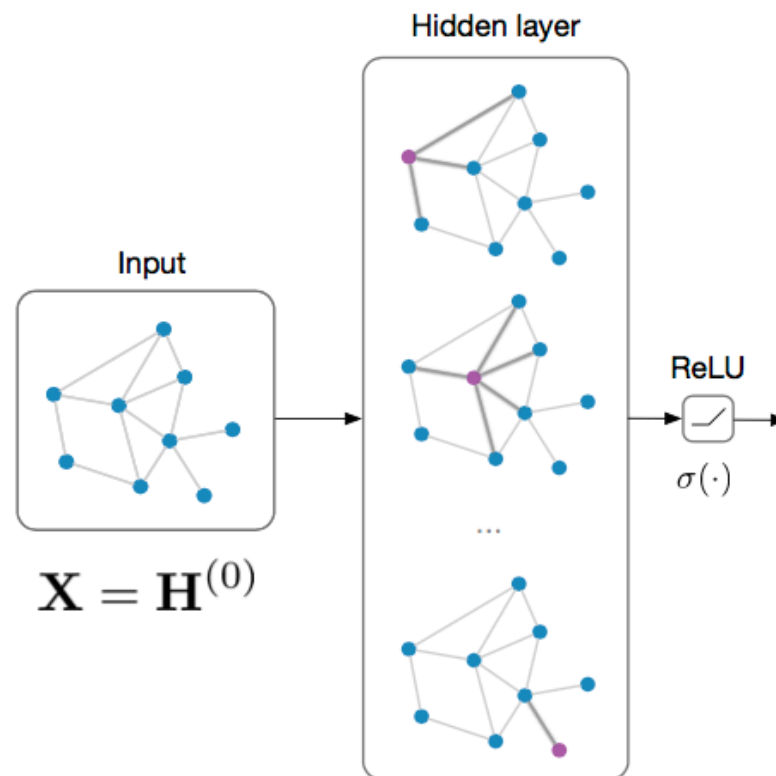
# Graph convolutional networks (GCN)

- Forward pass:  $\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L)f) \right)$



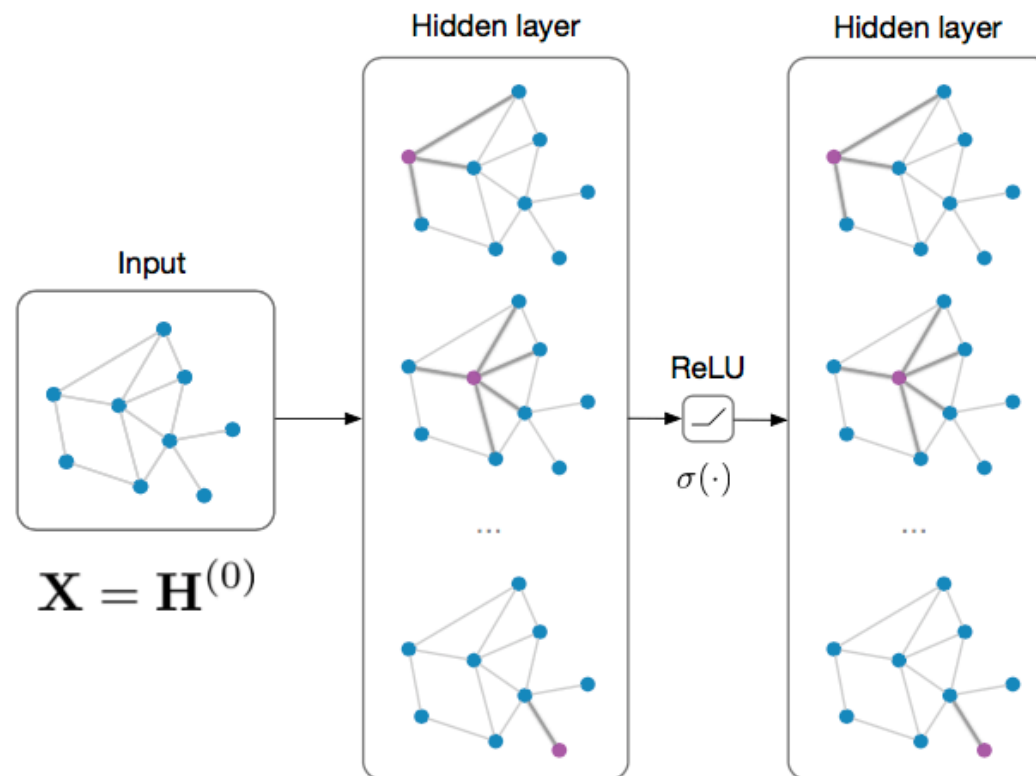
# Graph convolutional networks (GCN)

- Forward pass:  $\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L)f) \right)$



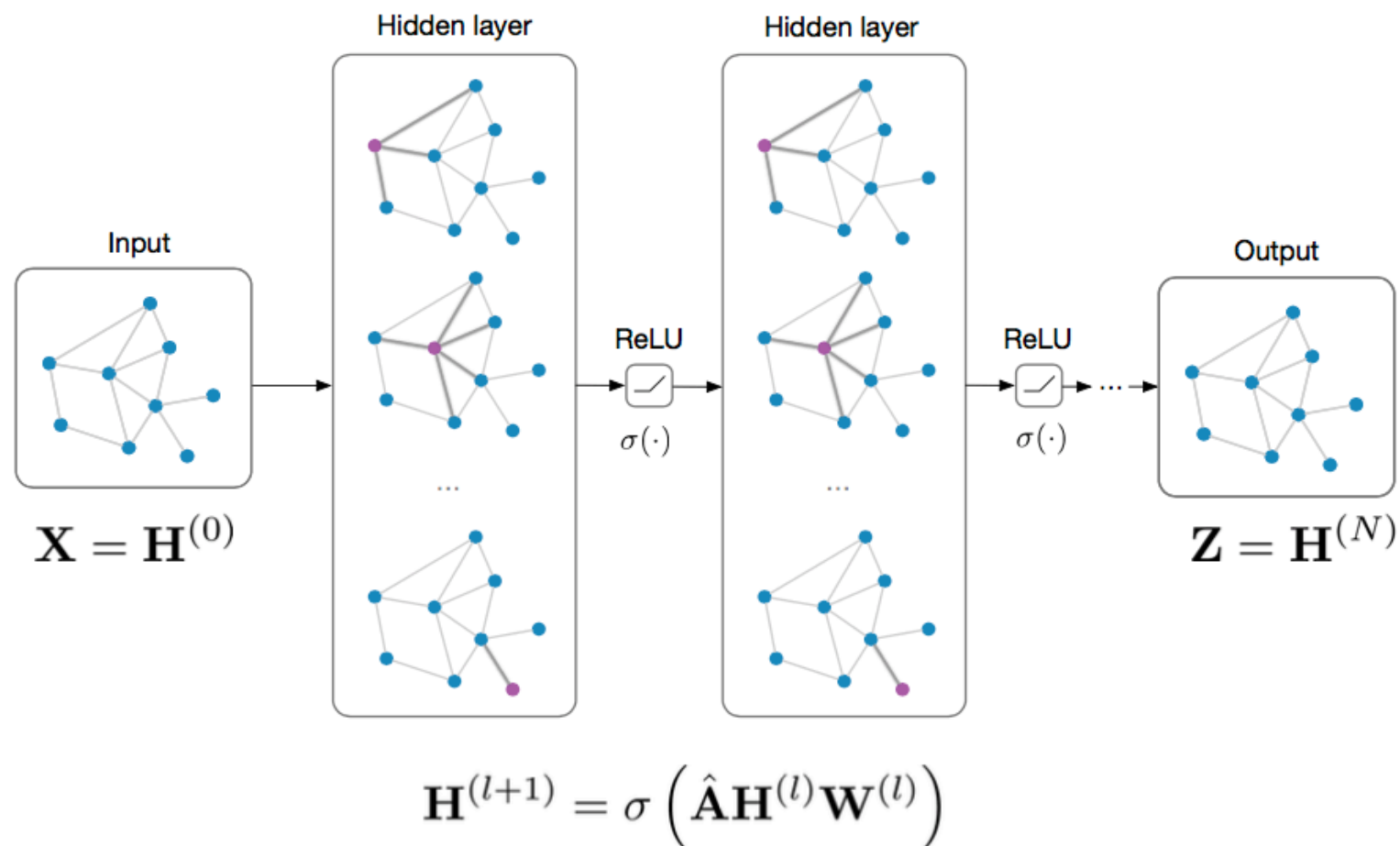
# Graph convolutional networks (GCN)

- Forward pass:  $\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L)f) \right)$

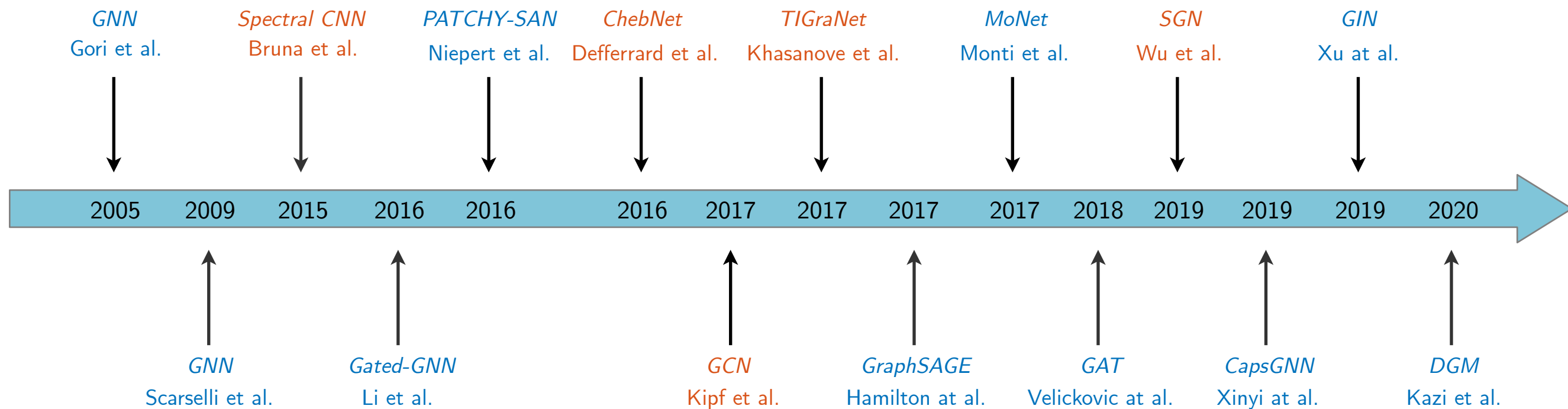


# Graph convolutional networks (GCN)

- Forward pass:  $\hat{g}_{\theta^{(k+1)}}(L) \left( \text{ReLU}(\hat{g}_{\theta^{(k)}}(L)f) \right)$



# Graph neural networks (GNNs)



**spatial-based methods (message-passing, attention)**

**spectral-based methods (spectral filtering)**

# Outline

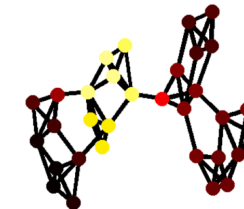
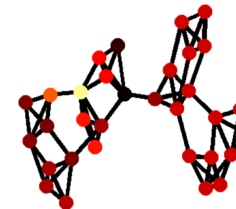
- Graph signal processing (GSP): Basic concepts
- Graph spectral filtering: Basic tool of GSP
- Graph neural networks (GNNs)
- Applications

# Application I: Community detection

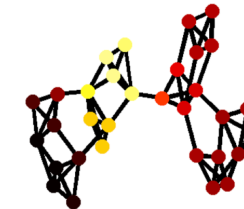
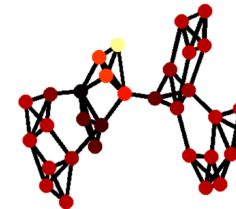
spectral graph wavelets  
at different scales:

$$D_s(a, b) = 1 - \frac{\psi_{s,a}^\top \psi_{s,b}}{\|\psi_{s,a}\|_2 \|\psi_{s,b}\|_2}.$$

NODE  
A:



NODE  
B:  
CORR.  
COEF.:



-0.50

0.97

**small scale**

**large scale**

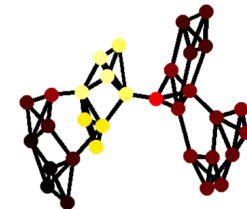


# Application I: Community detection

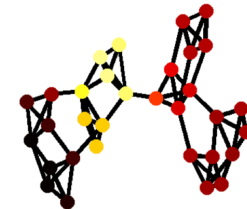
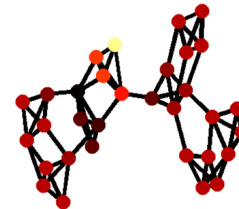
spectral graph wavelets  
at different scales:

$$D_s(a, b) = 1 - \frac{\psi_{s,a}^\top \psi_{s,b}}{\|\psi_{s,a}\|_2 \|\psi_{s,b}\|_2}.$$

NODE  
A:



NODE  
B:  
CORR.  
COEF.:



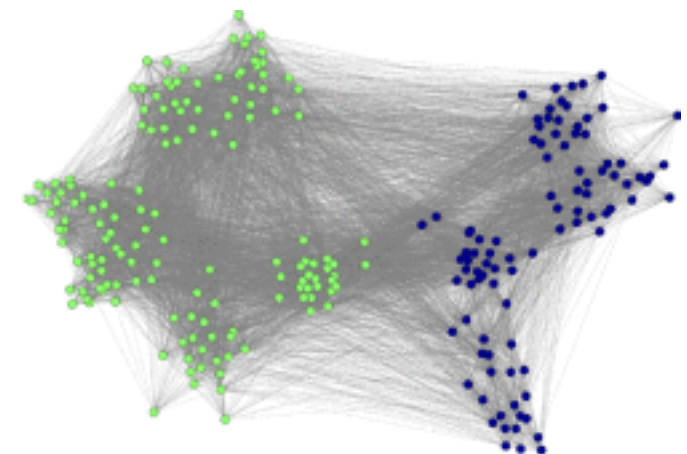
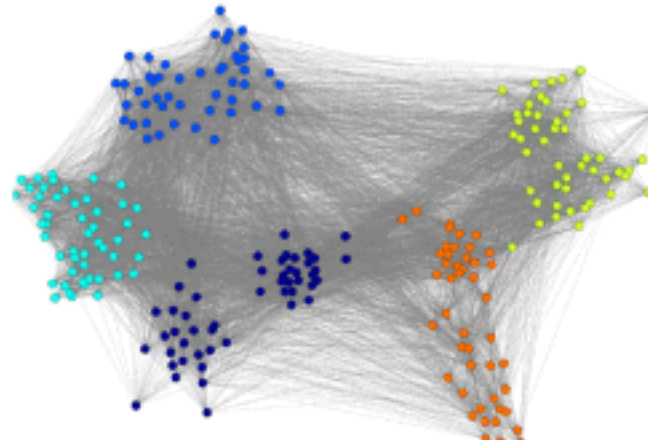
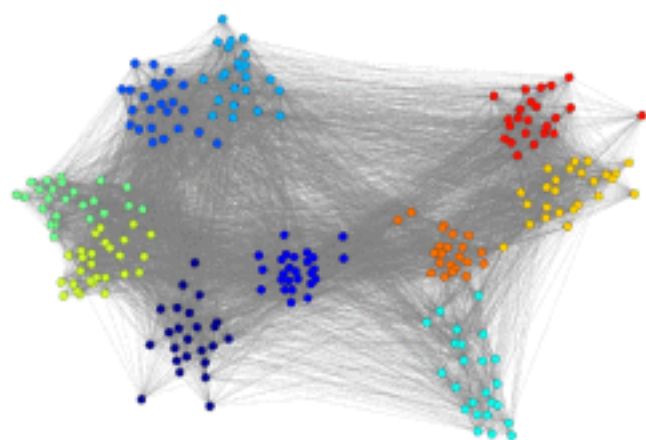
-0.50

0.97

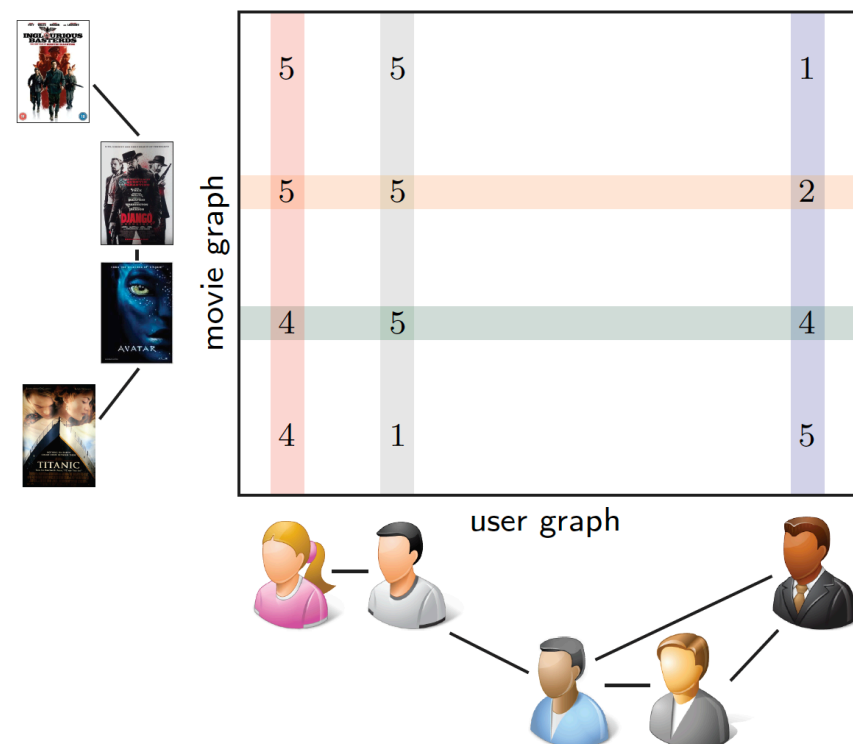
**small scale**

**large scale**

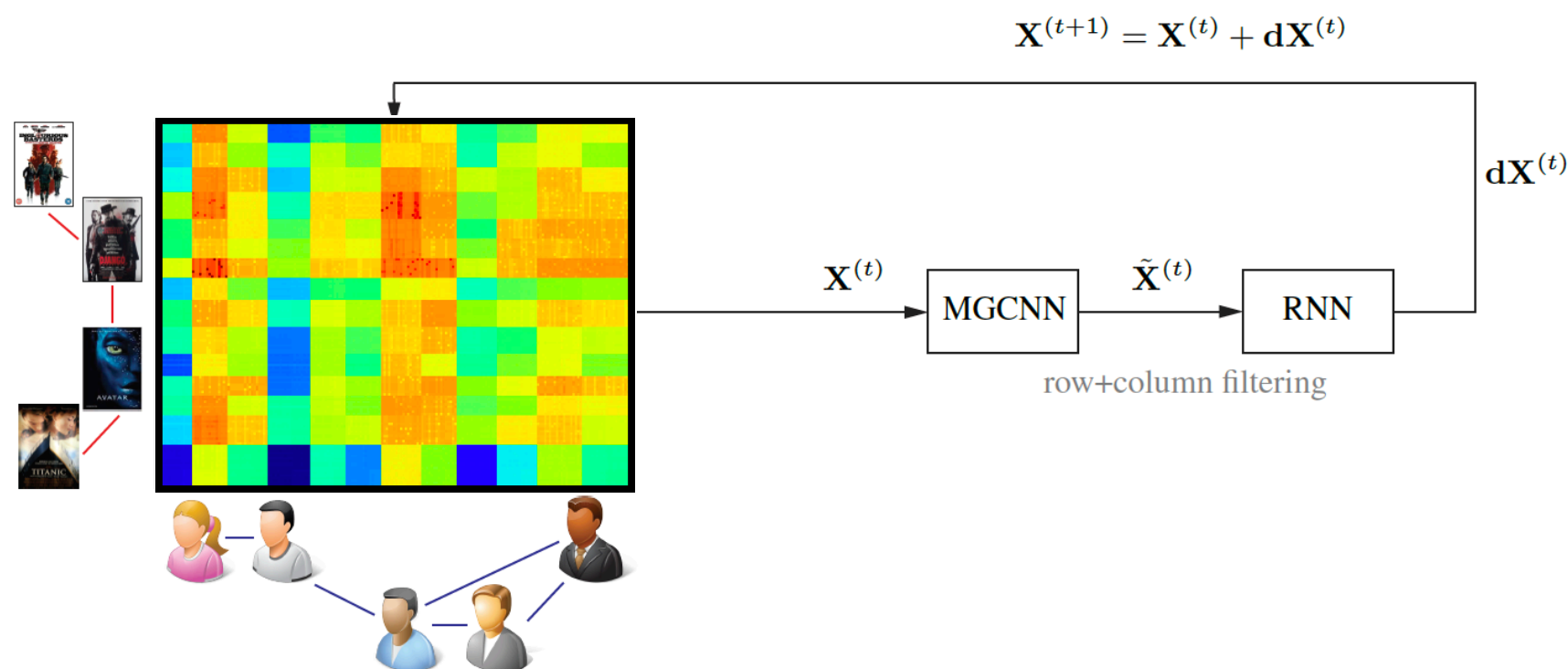
multi-scale community  
detection:



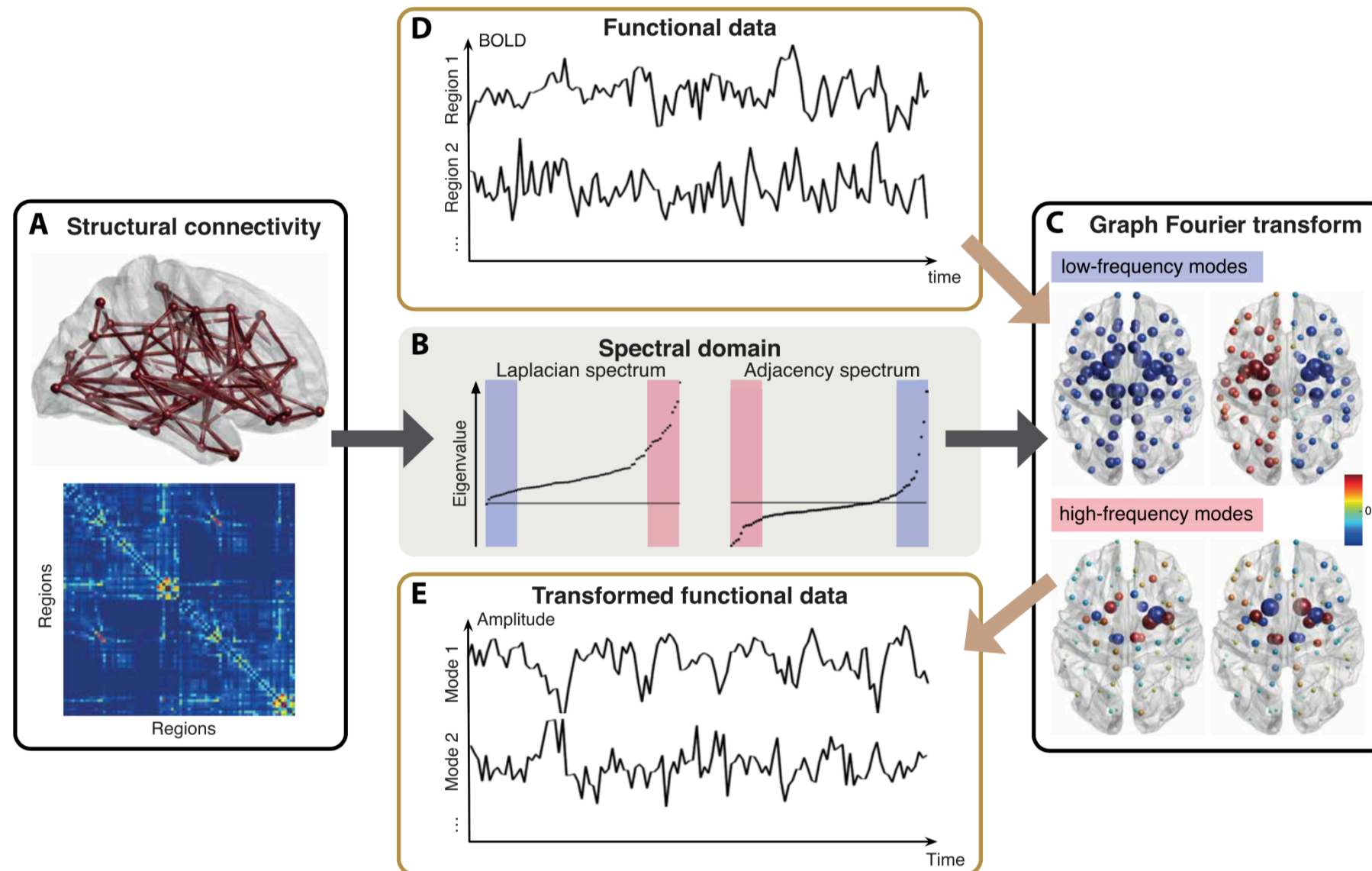
# Application II: Recommender systems



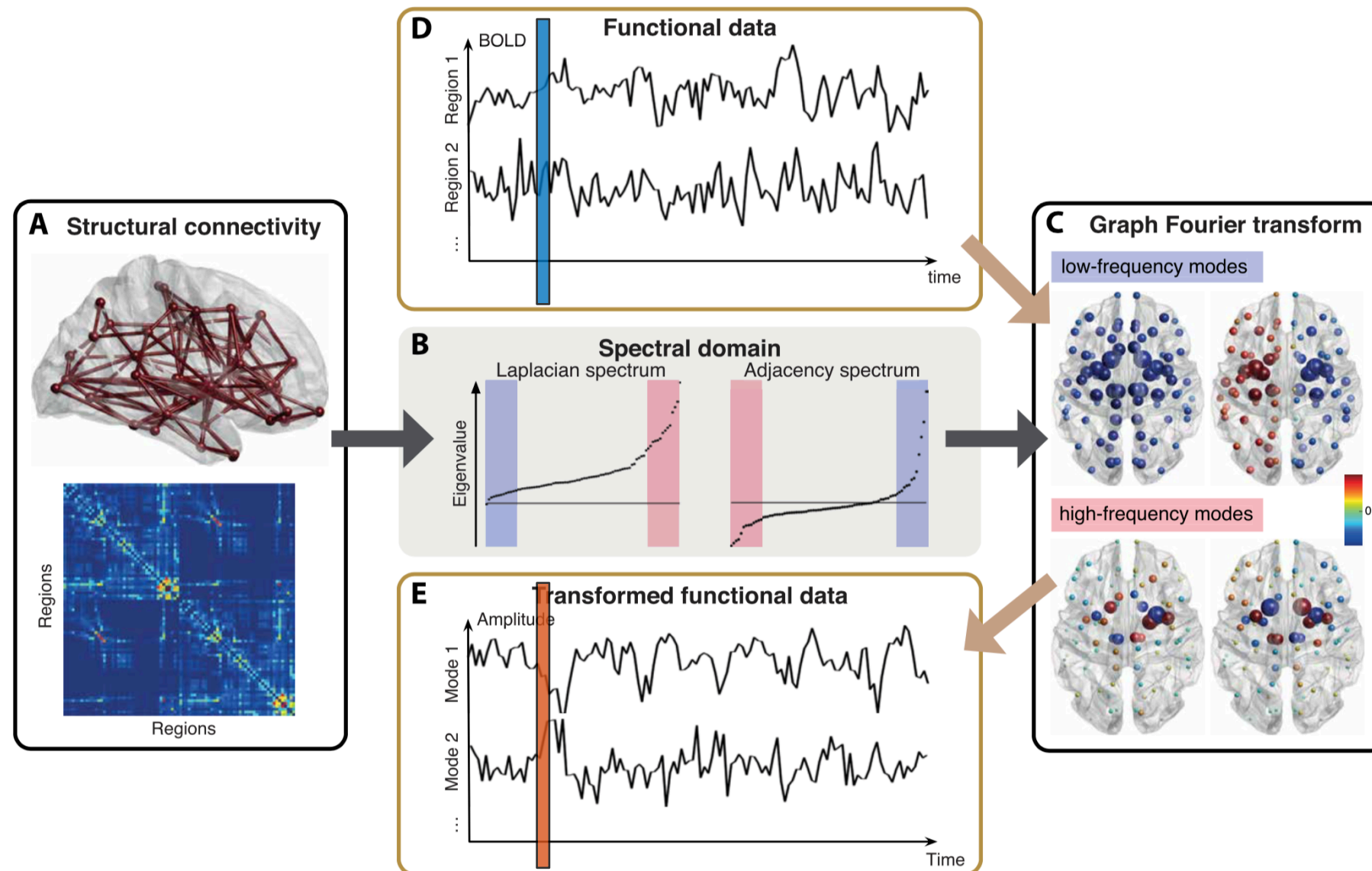
# Application II: Recommender systems



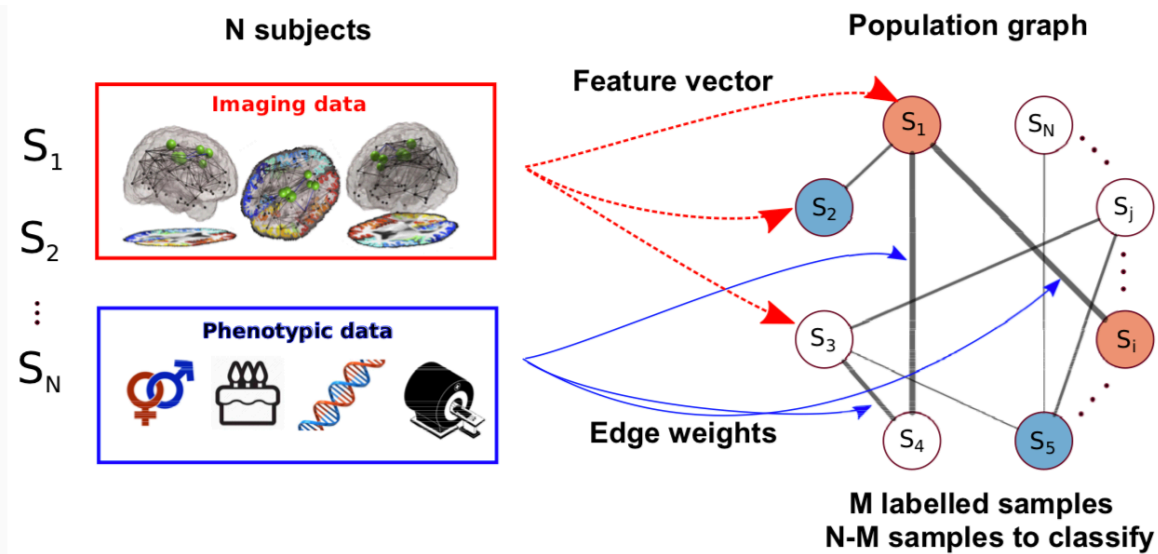
# Application III: Functional brain imaging



# Application III: Functional brain imaging



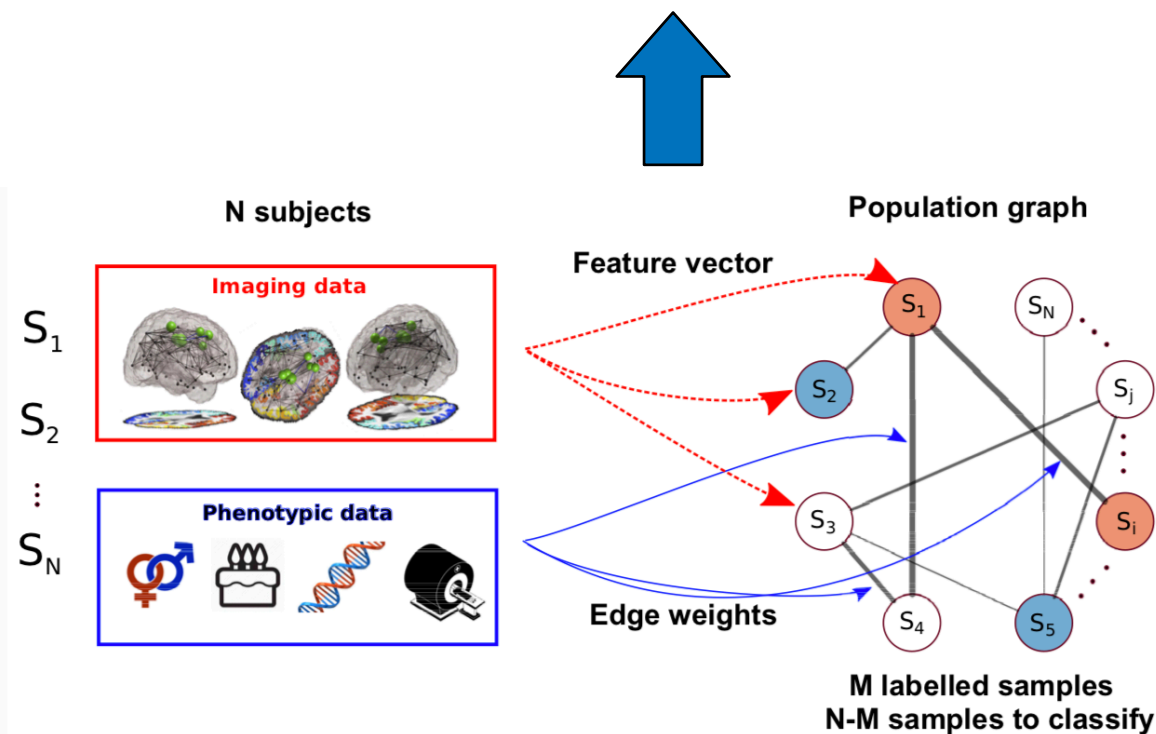
# Application IV: Disease classification





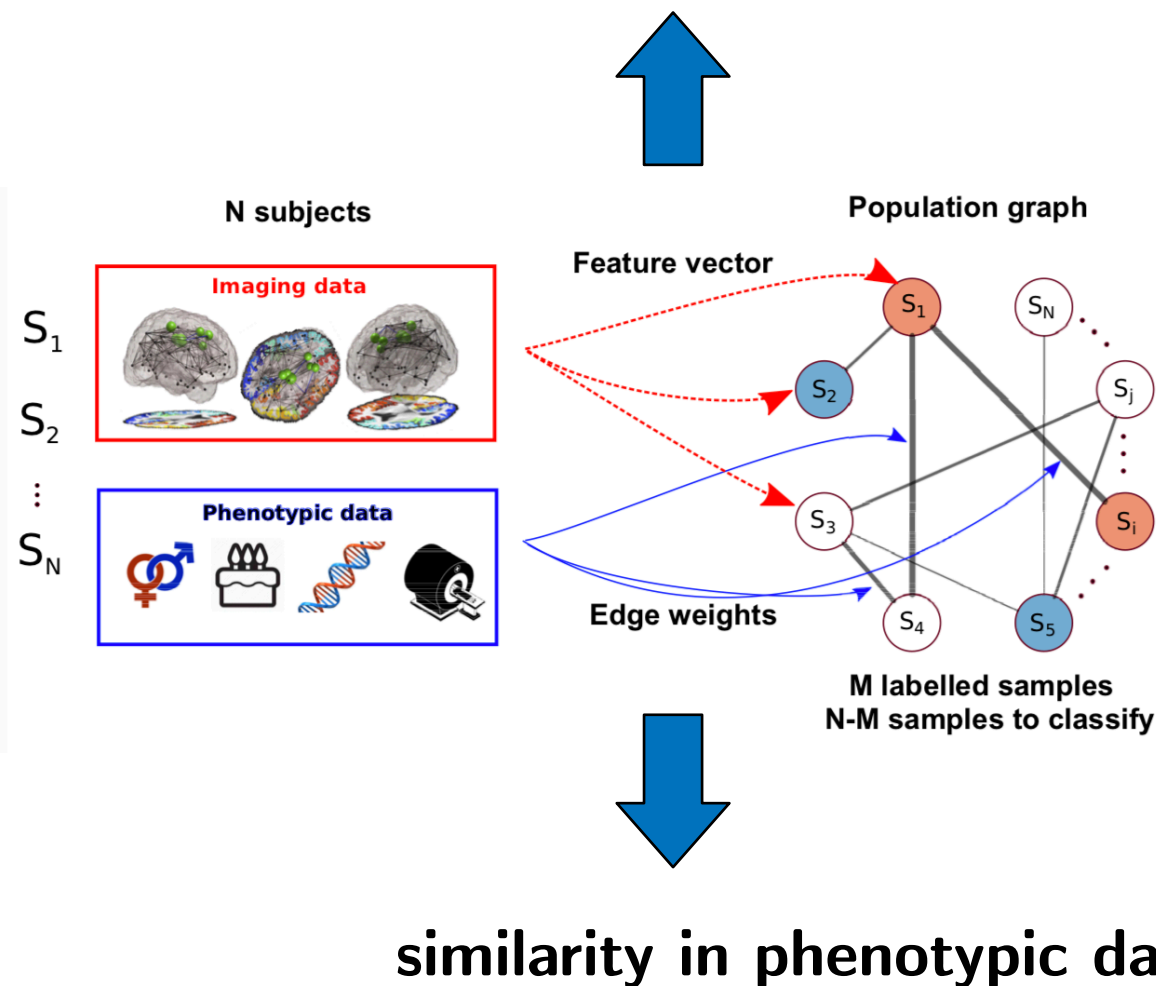
# Application IV: Disease classification

features extracted from brain analysis



# Application IV: Disease classification

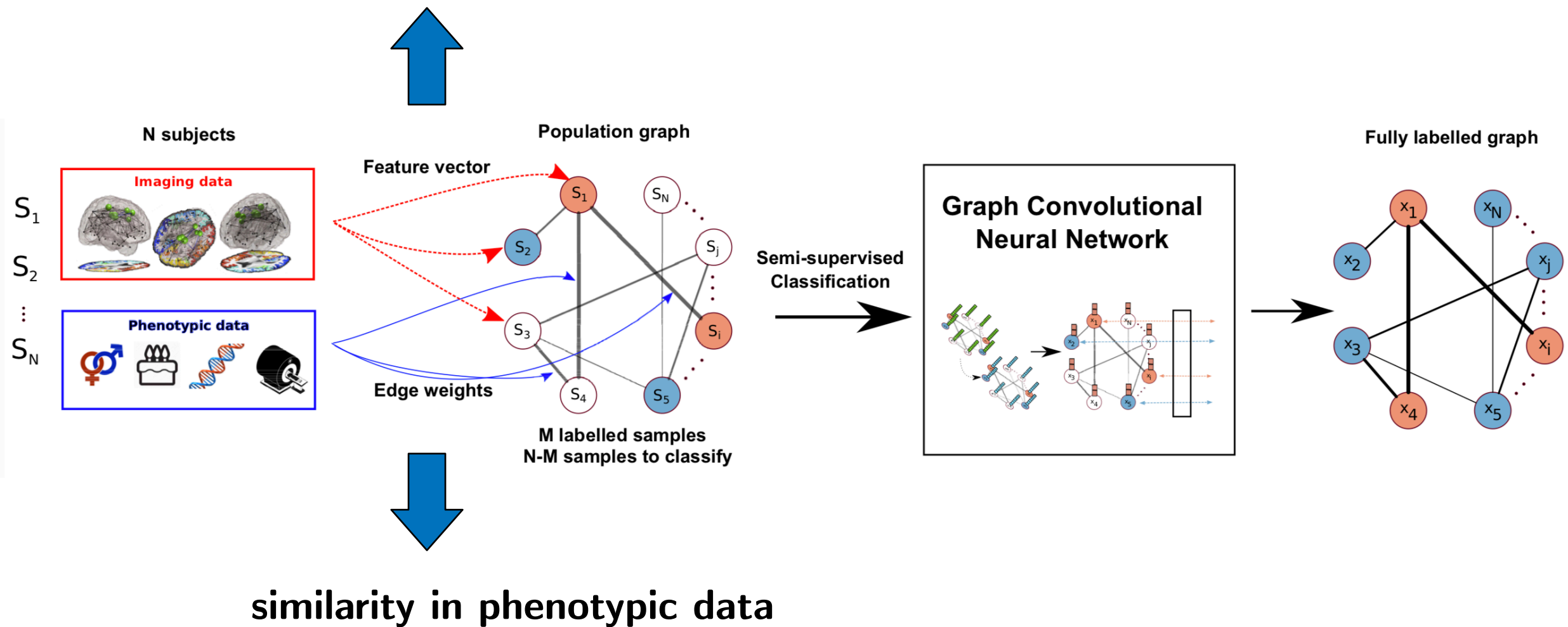
features extracted from brain analysis



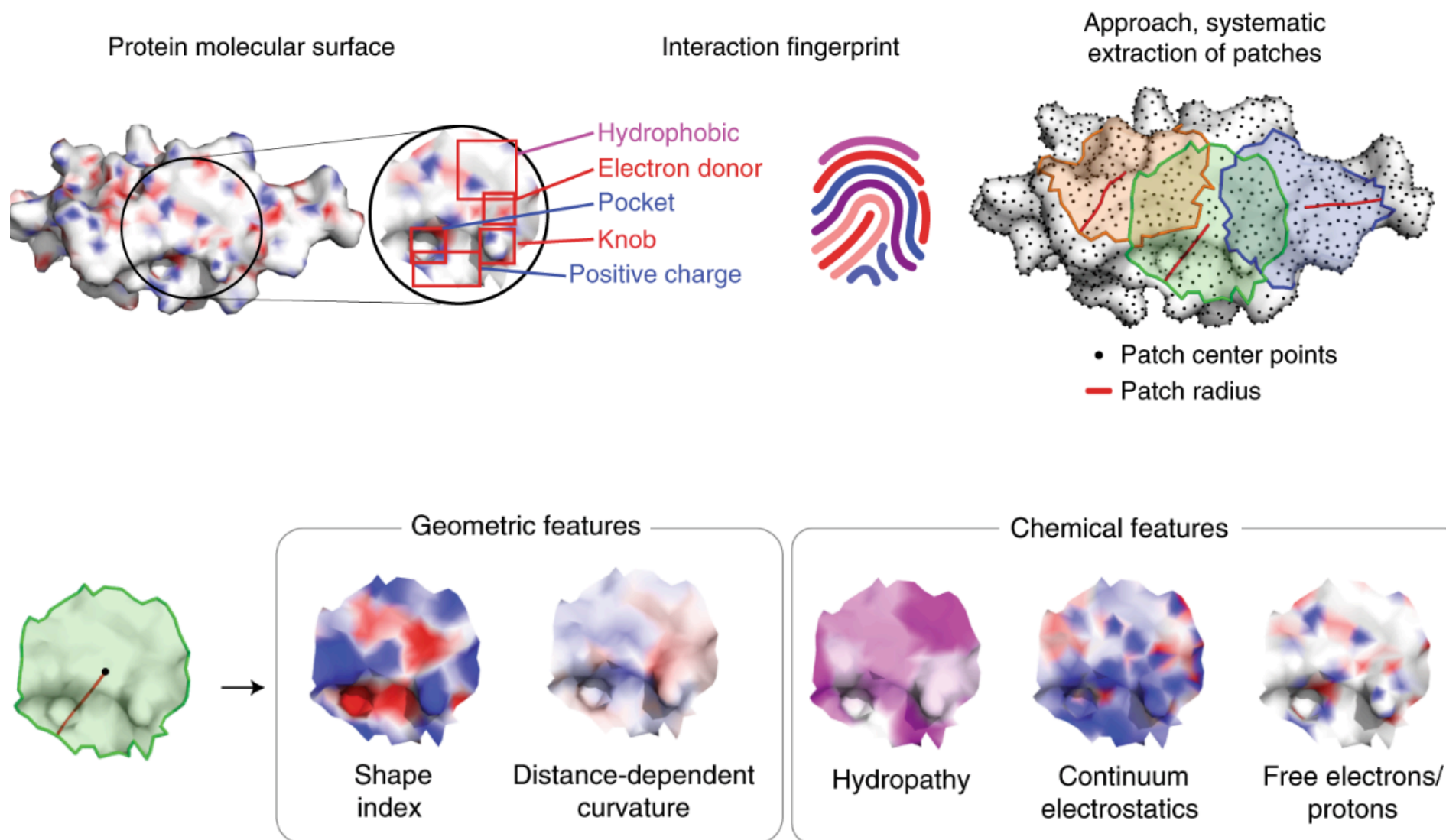


# Application IV: Disease classification

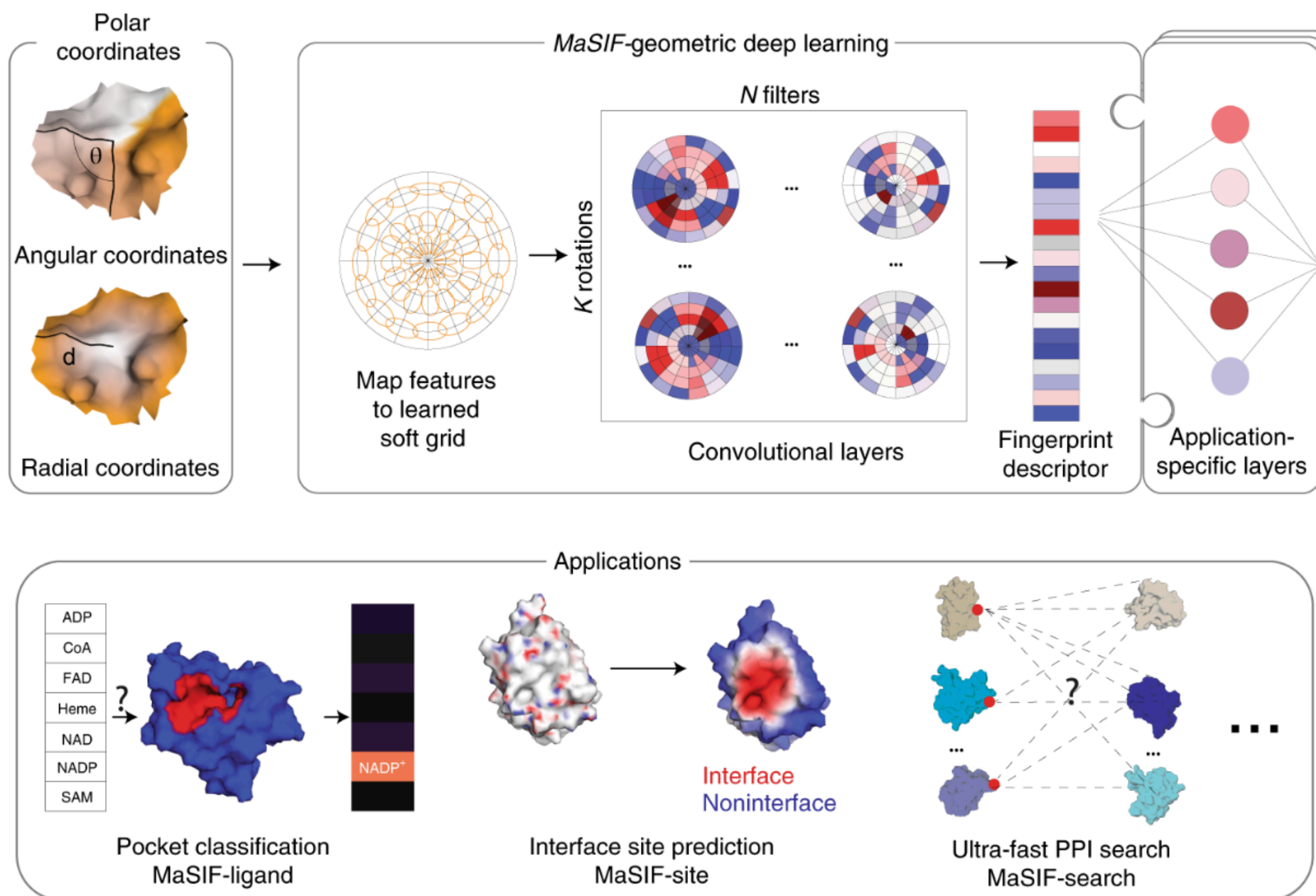
features extracted from brain analysis



# Application V: Protein-protein interaction



# Application V: Protein-protein interaction



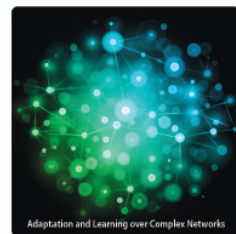
# What's next?

- Mathematical models for graph-structured data
  - global and local smoothness & regularity
  - underlying physical processes
- Robustness & generalisation analysis
  - how robust is the model to topological change
  - how can the trained model be generalised to unseen graph
- Probabilistic interpretation
  - connection to Bayesian inference
  - Gaussian processes on graphs
- Learning graphs from data
  - dynamic graph construction
  - graph generative models
- Fast implementation

# Papers & Resources

[David I Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst]

## The Emerging Field of Signal Processing on Graphs



[Extending high-dimensional data analysis to networks and other irregular domains]

In applications such as social, energy, transportation, sensor, and neuronal networks, high-dimensional data naturally reside on the vertices of weighted graphs. The emerging field of signal processing on graphs merges algebraic and spectral graph theoretic concepts with computational harmonic analysis to process such signals on graphs. In this tutorial overview, we outline the main challenges of the area, discuss different ways to define graph spectral domains, which are the analogs to the classical frequency domain, and highlight the importance of incorporating the irregular structures of graph data domains when processing signals on graphs. We then review methods to generalize fundamental operations such as filtering, translation, modulation, dilation, and downsampling to the graph setting and survey the localized, multiscale transforms that have

been proposed to efficiently extract information from high-dimensional data on graphs. We conclude with a brief discussion of open issues and possible extensions.

### INTRODUCTION

Graphs are generic data representation forms that are useful for describing the geometric structures of data domains in numerous applications, including social, energy, transportation, sensor, and neuronal networks. The weight associated with each edge in the graph often represents the similarity between the two vertices it connects. The connectivities and edge weights are either dictated by the physics of the problem at hand or inferred from the data. For instance, the edge weight may be inversely proportional to the physical distance between nodes in the network. The data on these graphs can be visualized as a finite collection of samples, with one sample at each vertex in the graph. Collectively, we refer to these

Digital Object Identifier 10.1109/SPM.2012.2235282  
Date of publication: 5 April 2013

1053-5888/13/\$31.0002013IEEE

IEEE SIGNAL PROCESSING MAGAZINE | (X) | MAY 2013

Michael M. Bronstein, Joan Bruno, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst

Many scientific fields study data with an underlying structure that is non-Euclidean. Some examples include social networks in computational social sciences, sensor networks in communications, functional networks in brain imaging, regulatory networks in genetics, and meshed surfaces in computer graphics. In many applications, such geometric data are large and complex (in the case of social networks, on the scale of billions) and are natural targets for machine-learning techniques. In particular, we would like to use deep neural networks, which have recently proven to be powerful tools for a broad range of problems from computer vision, natural-language processing, and audio analysis. However, these tools have been most successful on data with an underlying Euclidean or grid-like structure and in cases where the invariances of these structures are built into networks used to model them.

*Geometric deep learning* is an umbrella term for emerging techniques attempting to generalize (structured) deep neural models to non-Euclidean domains, such as graphs and manifolds. The purpose of this article is to overview different examples of geometric deep-learning problems and present available solutions, key difficulties, applications, and future research directions in this nascent field.

### Overview of deep learning

Deep learning refers to learning complicated concepts by building them from simpler ones in a hierarchical or multilayer manner. Artificial neural networks are popular realizations of such deep multilayer hierarchies. In the past few years, the growing computational power of modern graphics processing unit (GPU)-based computers and the availability of large training data sets have allowed successfully training neural networks with many layers and degrees of freedom (DoF) [1]. This has led to qualitative breakthroughs on a wide variety of tasks, from speech recognition [2], [3] and machine translation [4] to image analysis and computer vision [5]–[11] (see [12]

## Geometric Deep Learning

Going beyond Euclidean data

Digital Object Identifier 10.1109/SPM.2017.2681418  
Date of publication: 11 July 2017

18

IEEE SIGNAL PROCESSING MAGAZINE | July 2017 |

1053-5888/17/0702017IEEE

Xiaowen Dong, Dorina Thanou, Laura Toni, Michael Bronstein, and Pascal Frossard

## Graph Signal Processing for Machine Learning

A review and new perspectives



The effective representation, processing, analysis, and visualization of large-scale structured data, especially those related to complex domains, such as networks and graphs, are one of the key questions in modern machine learning. Graph signal processing (GSP), a vibrant branch of signal processing models and algorithms that aims at handling data supported on graphs, opens new paths of research to address this challenge. In this article, we review a few important contributions made by GSP concepts and tools, such as graph filters and transforms, to the development of novel machine learning algorithms. In particular, our discussion focuses on the following three aspects exploring data structure and relational priors, improving data and computational efficiency, and enhancing model interpretability. Furthermore, we provide new perspectives on the future development of GSP techniques that may serve as a bridge between applied mathematics and signal processing on one side and machine learning and network science on the other. Cross-fertilization across these different disciplines may help unlock the numerous challenges of complex data analysis in the modern age.

### Introduction

We live in a connected society. Data collected from large-scale interactive systems, such as biological, social, and financial networks, become largely available. In parallel, the past few decades have seen a significant amount of interest in the machine learning community for network data processing and analysis. Networks have an intrinsic structure that conveys very specific properties to data, e.g., interdependencies between data entities in the form of pairwise relationships. These properties are traditionally captured by mathematical representations such as graphs.

In this context, new trends and challenges have been developing fast. Let us consider, for example, a network of protein-protein interactions and the expression level of individual genes at every point in time. Some typical tasks in network biology related to this type of data are 1) discovery of key genes (via protein grouping) affected by the infection and 2) prediction of how the host organism reacts (in terms of gene expression)

Digital Object Identifier 10.1109/SPM.2020.3014701  
Date of current version: 28 October 2020

1053-5888/20/0202020IEEE

IEEE SIGNAL PROCESSING MAGAZINE | November 2020 |

117

- <http://www.robots.ox.ac.uk/~xdong/resource.html>
- <https://towardsdatascience.com/graph-deep-learning/home>
- <https://graphml.substack.com>