

Recent advances in learning with graphs

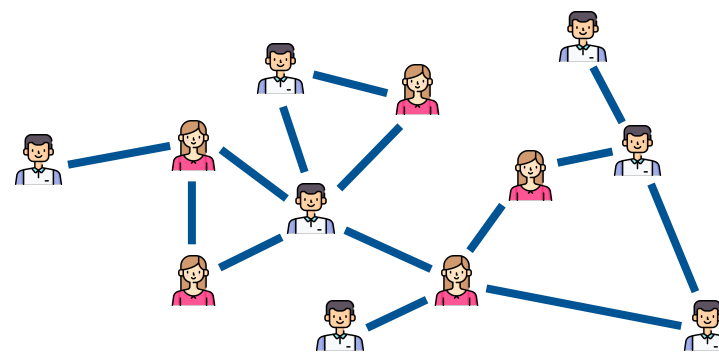
Xiaowen Dong

Department of Engineering Science
University of Oxford

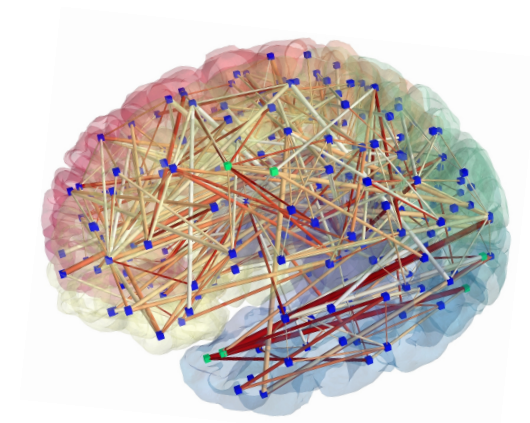
Networks are pervasive



traffic network



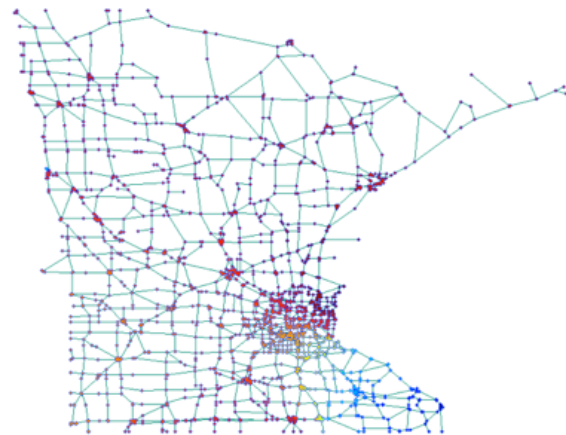
social network



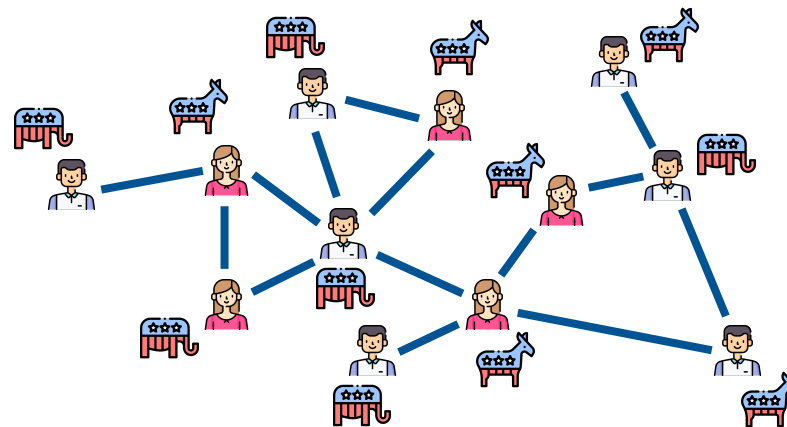
brain network

networks are mathematically represented by **graphs**

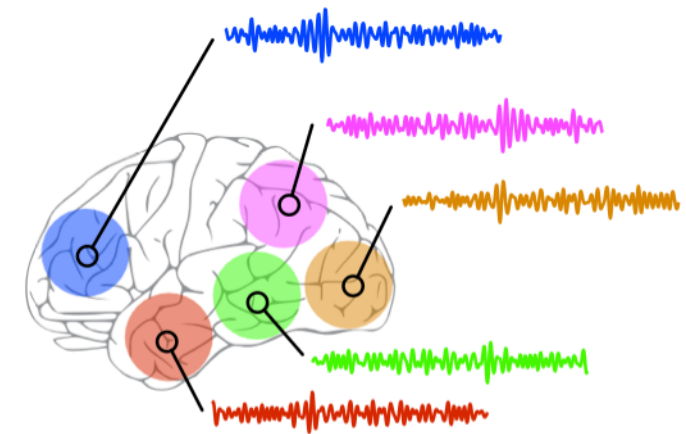
Data collected in networks are pervasive



congestion in road junctions



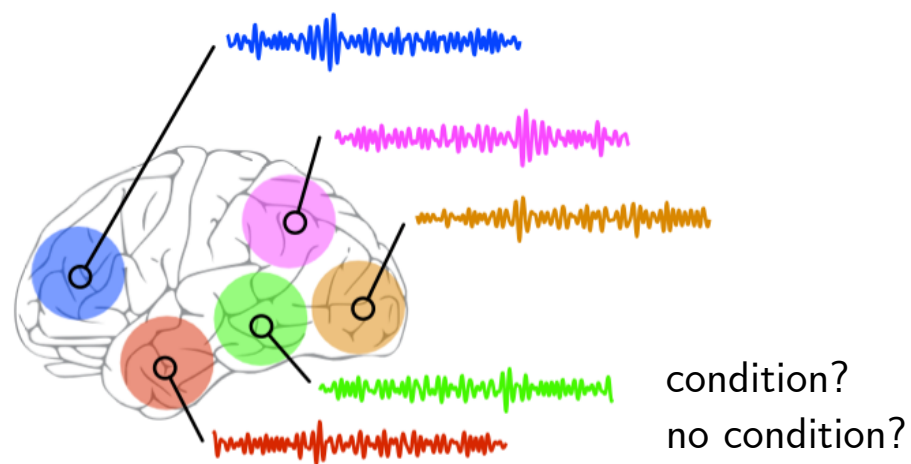
preferences of individuals



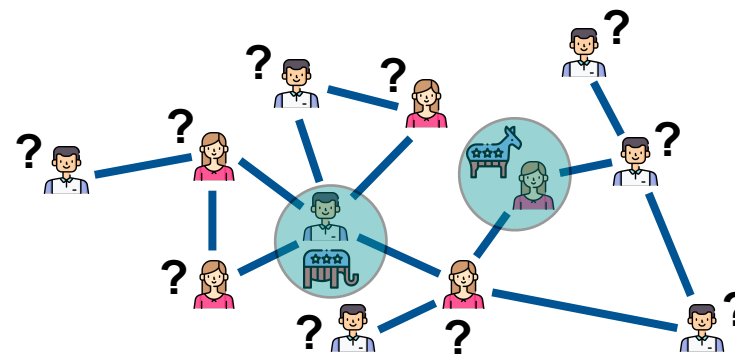
activities in brain regions

from **graphs** to **graph-structured data**

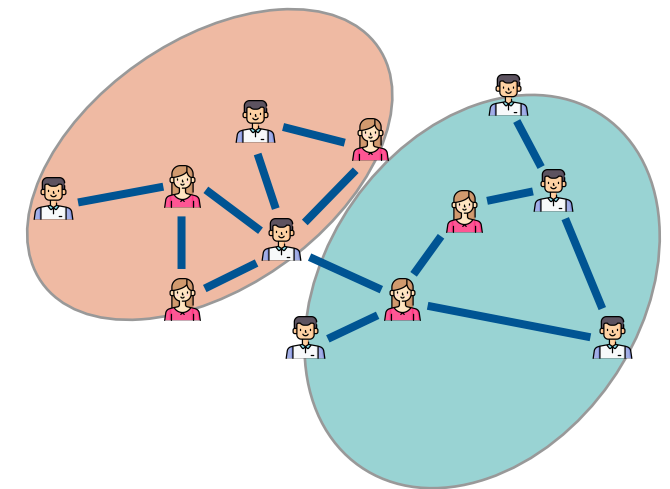
Learning with graph-structured data



**graph-level classification
(supervised)**



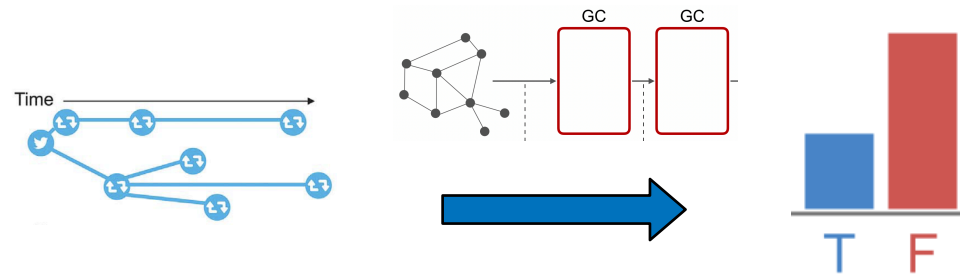
**node-level classification
(semi-supervised)**



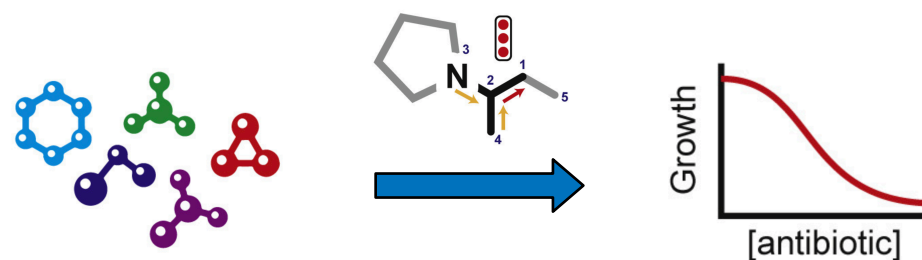
**graph clustering
(unsupervised)**

Exciting possibilities enabled by graph ML

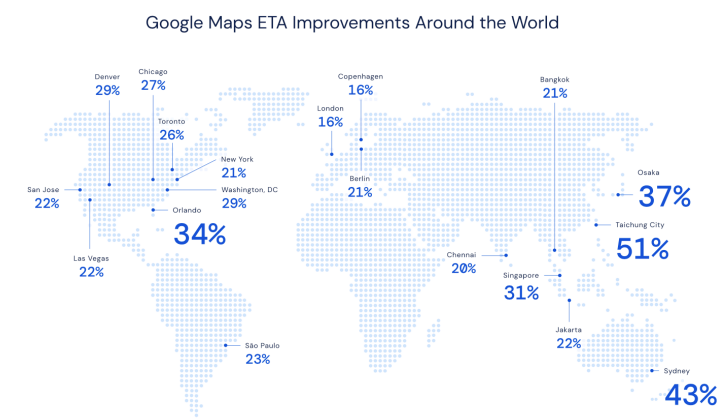
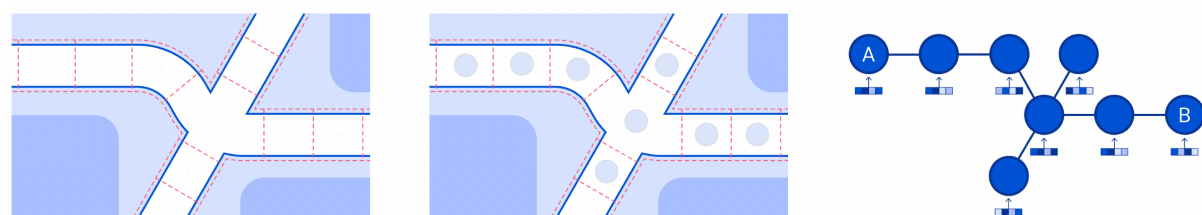
fake news detection



drug discovery



traffic prediction



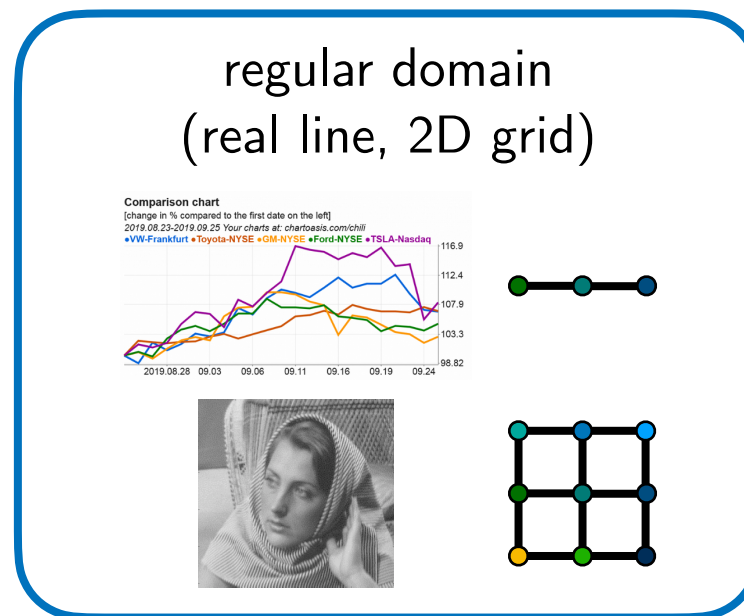
Monti et al., "Fake news detection on social media using geometric deep learning," ICLR Workshop, 2019.

Stokes et al., "A deep learning approach to antibiotic discovery," Cell, 2020.

Derrow-Pinion et al., "ETA prediction with graph neural networks in Google Maps," CIKM, 2021.

Classical ML vs Graph ML

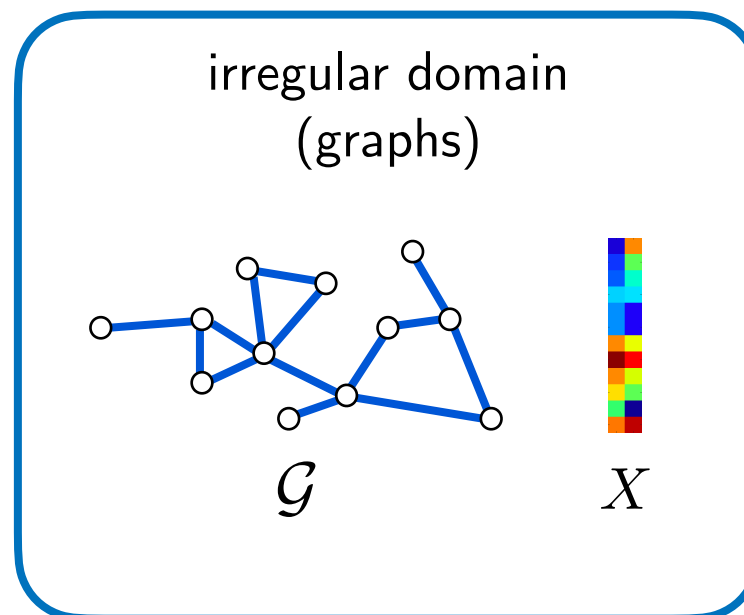
Classical ML



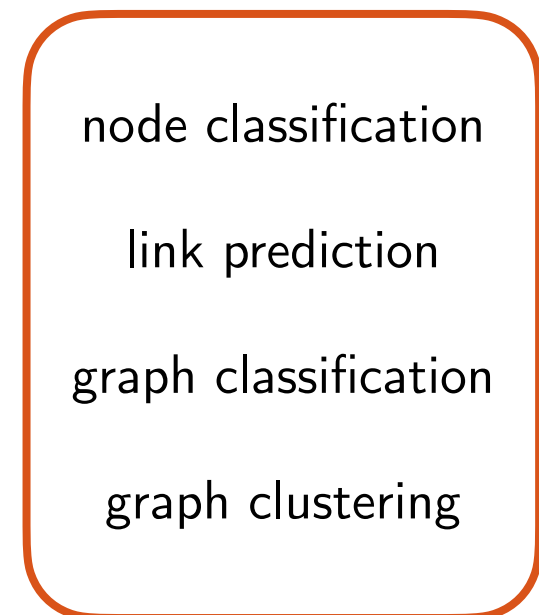
$$f(X)$$



Graph ML

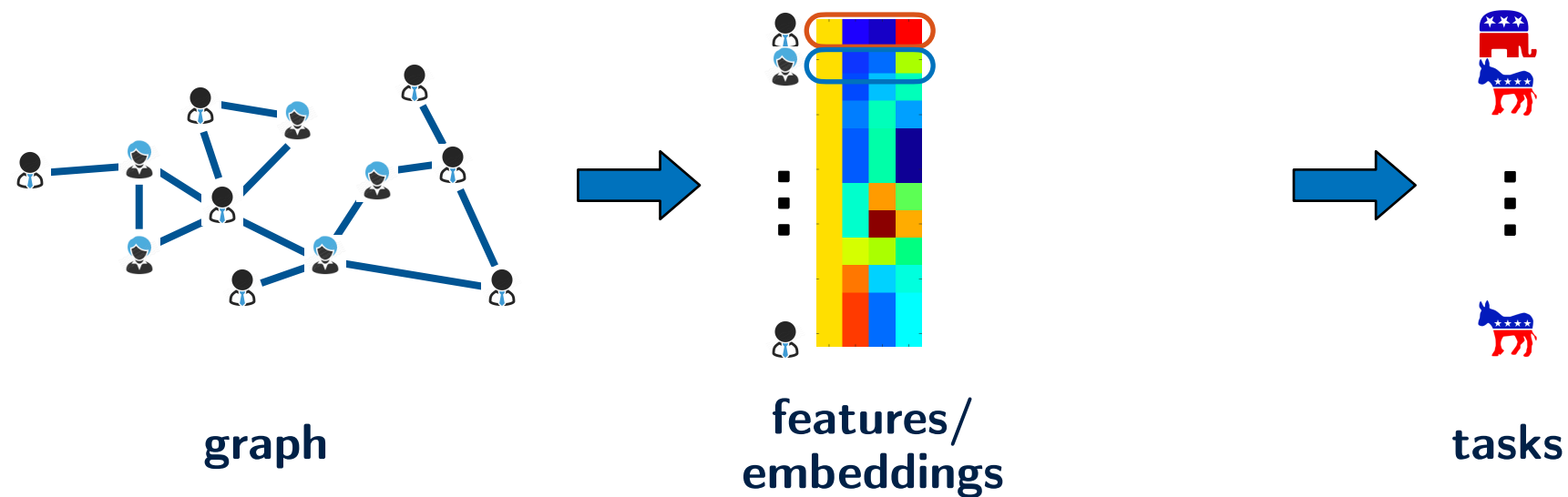


$$f(\mathcal{G}, X)$$



How to incorporate graphs into learning?

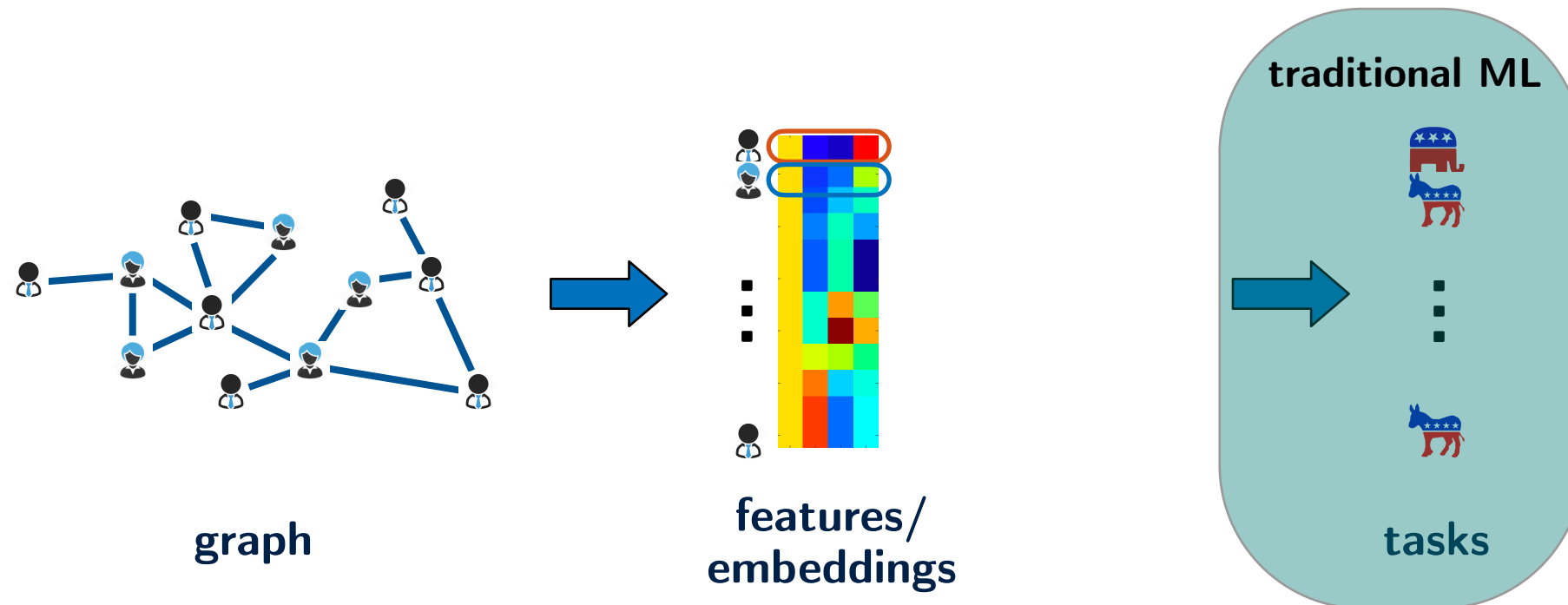
- Traditional machine learning on graphs



- Limitations
 - hand-crafted features or optimised embeddings, often focused on graph structure

How to incorporate graphs into learning?

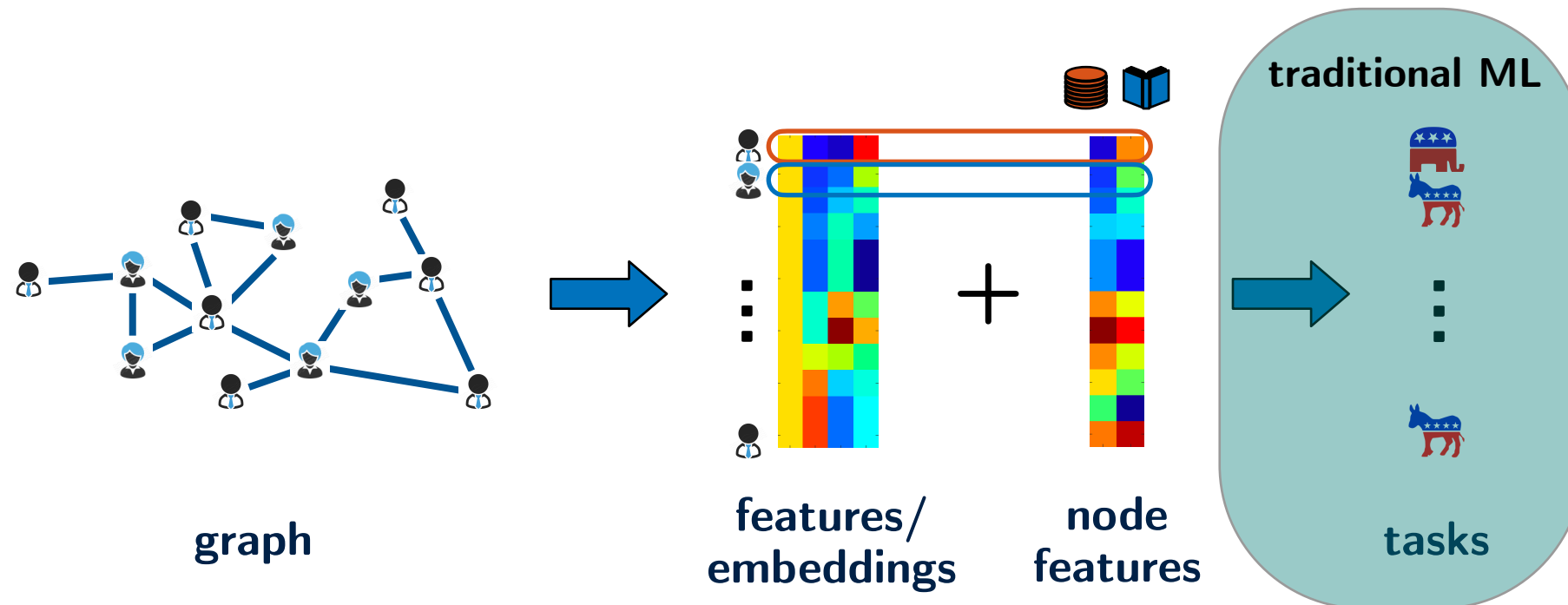
- Traditional machine learning on graphs



- Limitations
 - hand-crafted features or optimised embeddings, often focused on graph structure
 - respect notion of “closeness” in the graph, but do not adapt to downstream tasks

How to incorporate graphs into learning?

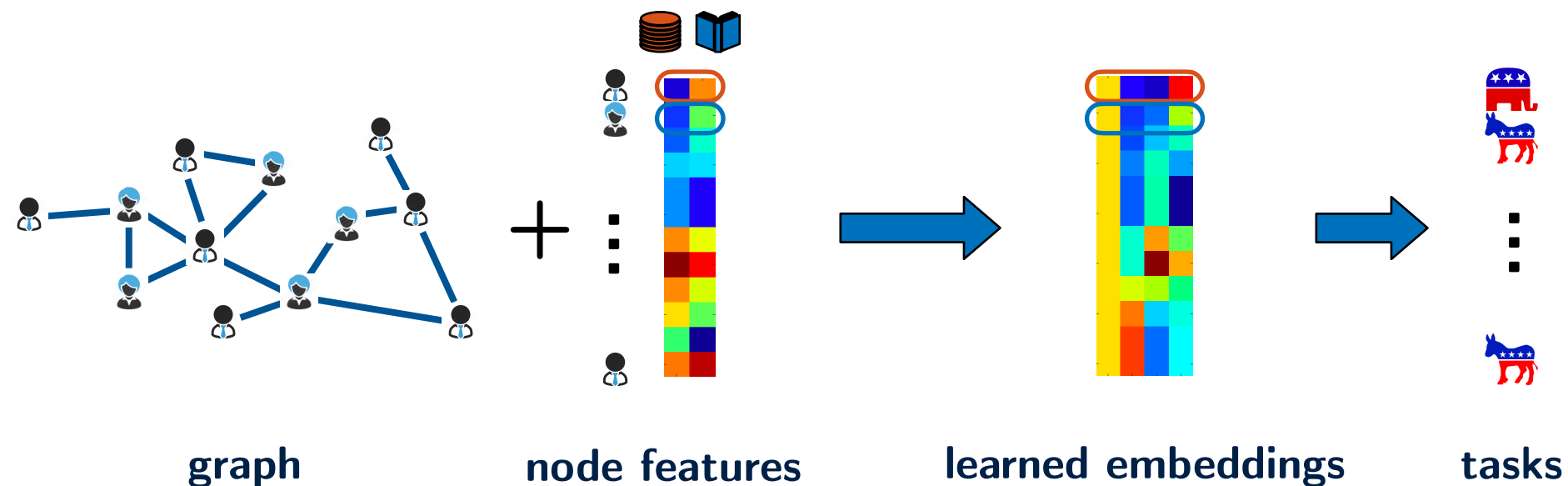
- Traditional machine learning on graphs



- Limitations
 - hand-crafted features or optimised embeddings, often focused on graph structure
 - respect notion of “closeness” in the graph, but do not adapt to downstream tasks
 - can incorporate additional node features, but in a mechanical way

How to incorporate graphs into learning?

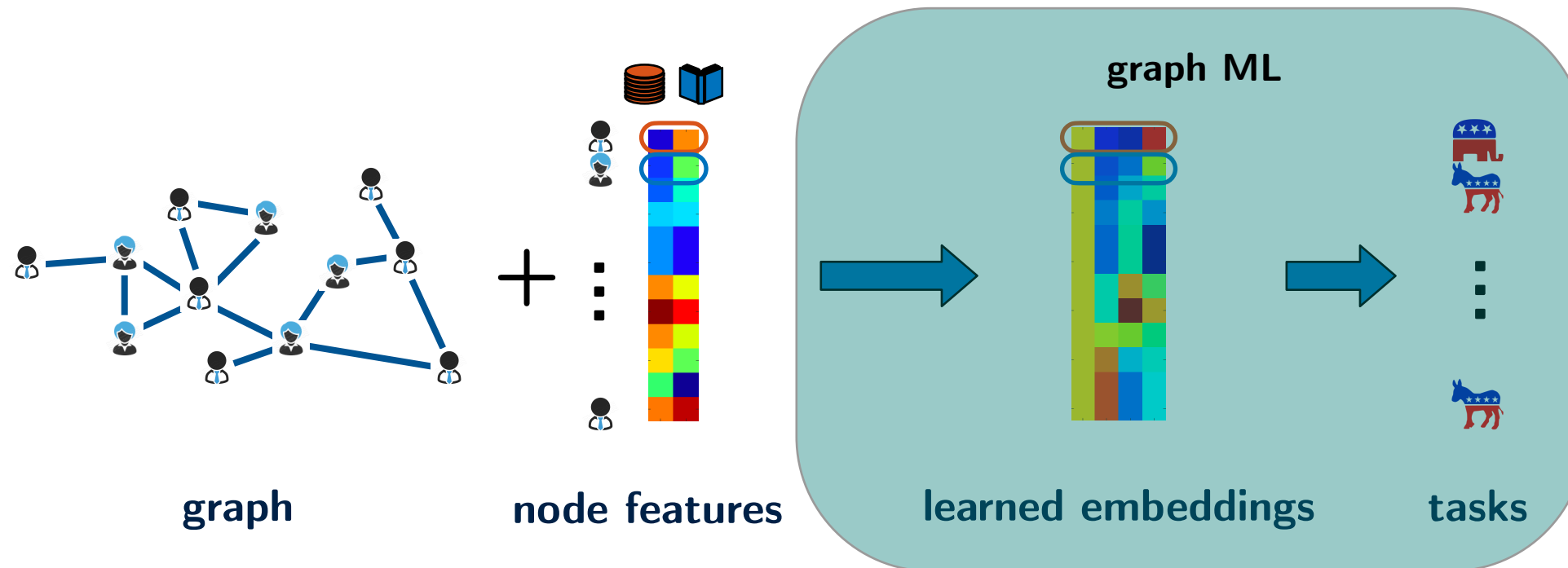
- Graph machine learning



- Advantages
 - naturally combine graph structure and node features in analysis and learning
 - new tools: graph signal processing, graph neural networks

How to incorporate graphs into learning?

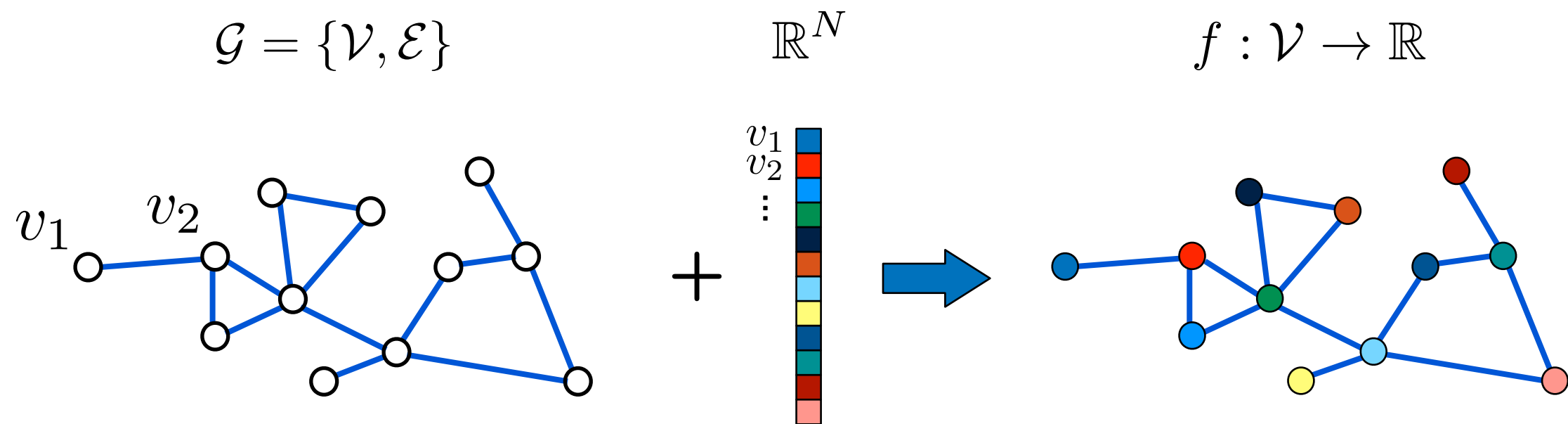
- Graph machine learning



- Advantages
 - naturally combine graph structure and node features in analysis and learning
 - new tools: graph signal processing, graph neural networks
 - embeddings can adapt to downstream tasks and be trained in end-to-end fashion
 - offers more flexibility and enables “deeper” architectures and embeddings

Graph signal processing

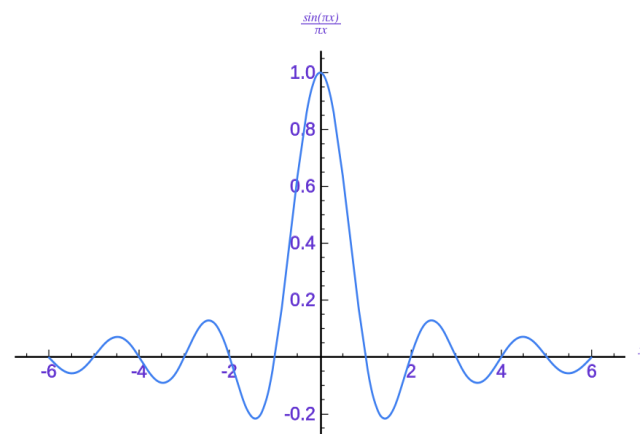
- Graph-structured data can be represented by graph signals



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

Graph signal processing

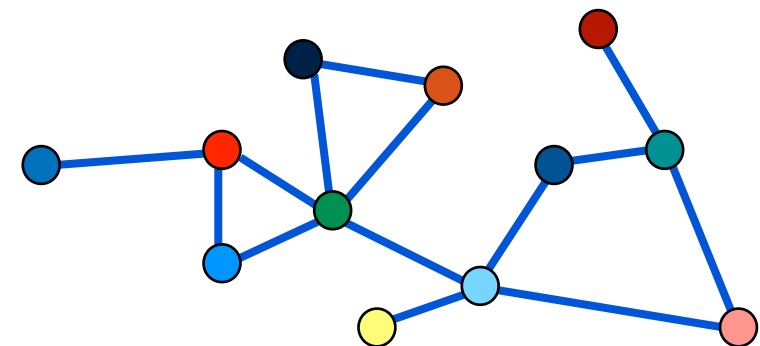
1D signal



2D signal

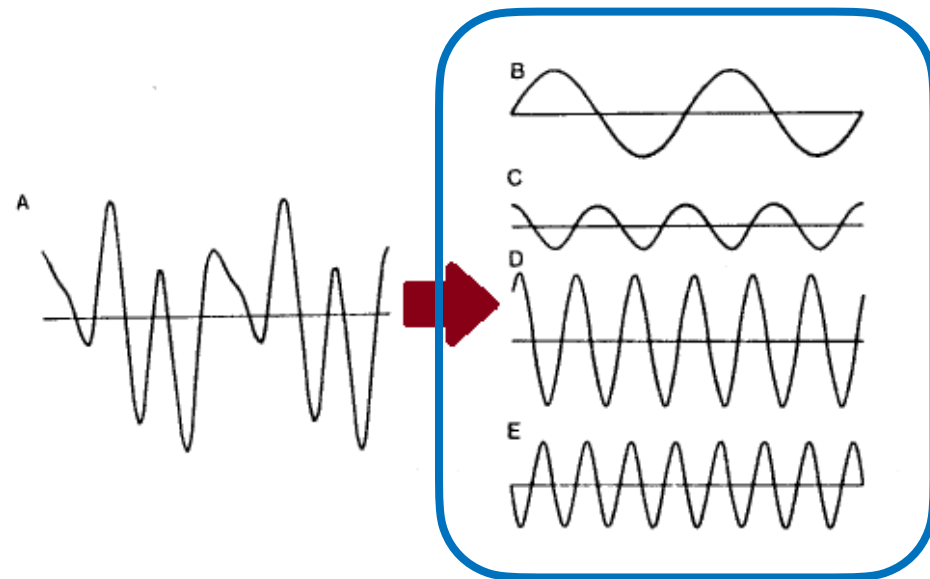


graph signal



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

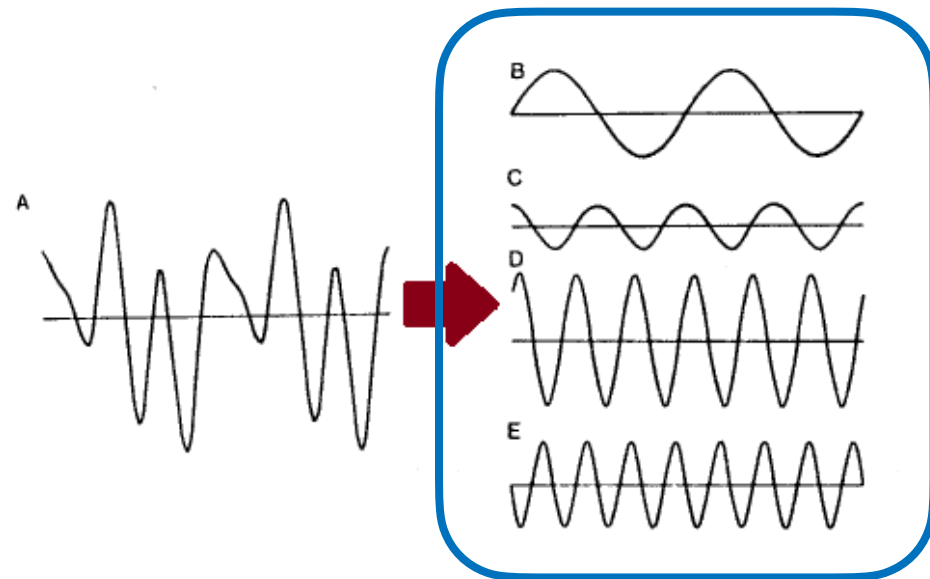
Graph signal processing



classical signal processing

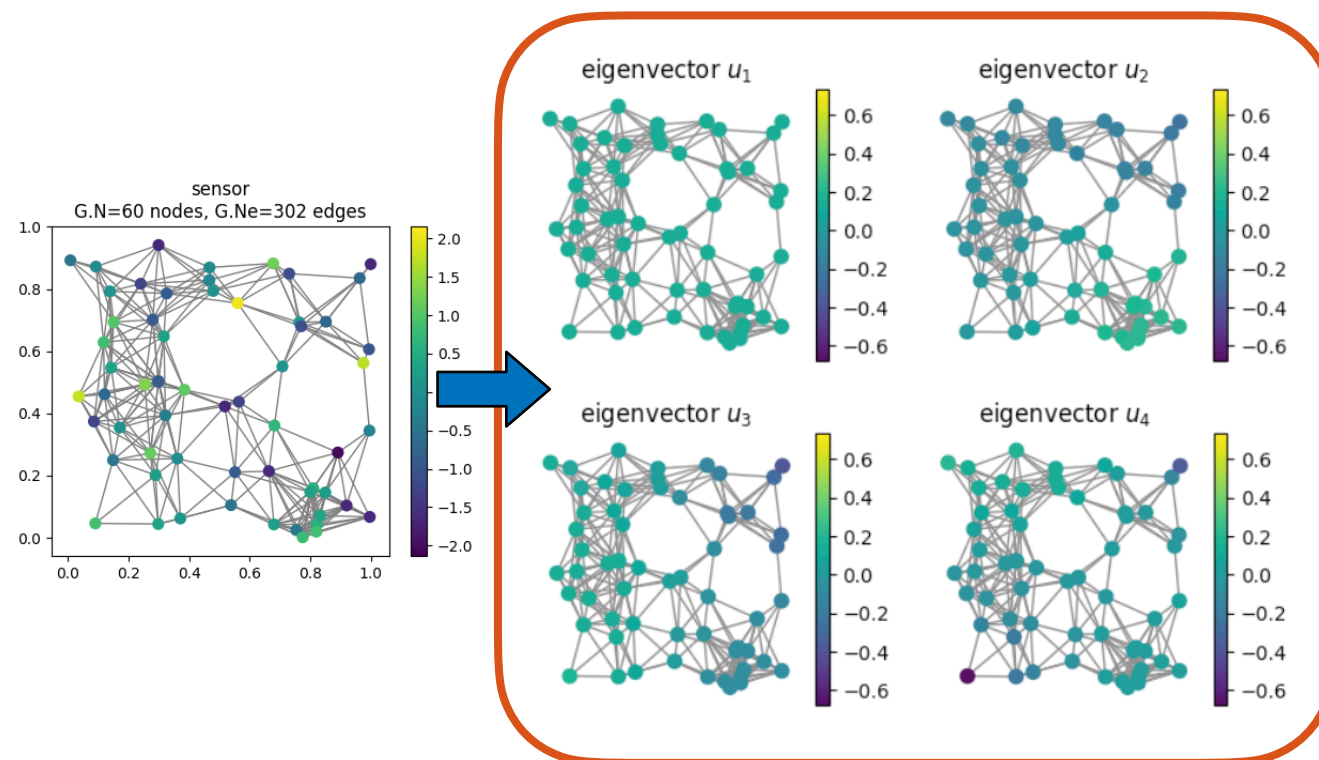
- complex exponentials provide “building blocks” of 1D signal (different oscillations or frequencies)
- leads to **Fourier transform**
- enables convolution and filtering

Graph signal processing



classical signal processing

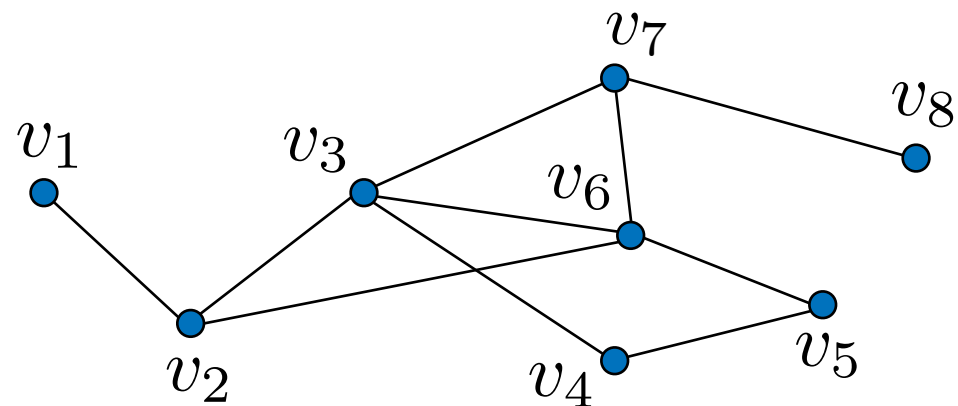
- complex exponentials provide “building blocks” of 1D signal (different oscillations or frequencies)
- leads to **Fourier transform**
- enables convolution and filtering



graph signal processing

- Laplacian eigenvectors provide “building blocks” of graph signal (different oscillation or frequencies)
- leads to **graph Fourier transform**
- enables **convolution** and **filtering** on graphs

Graphs and graph Laplacian



weighted and undirected graph:

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

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

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

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

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

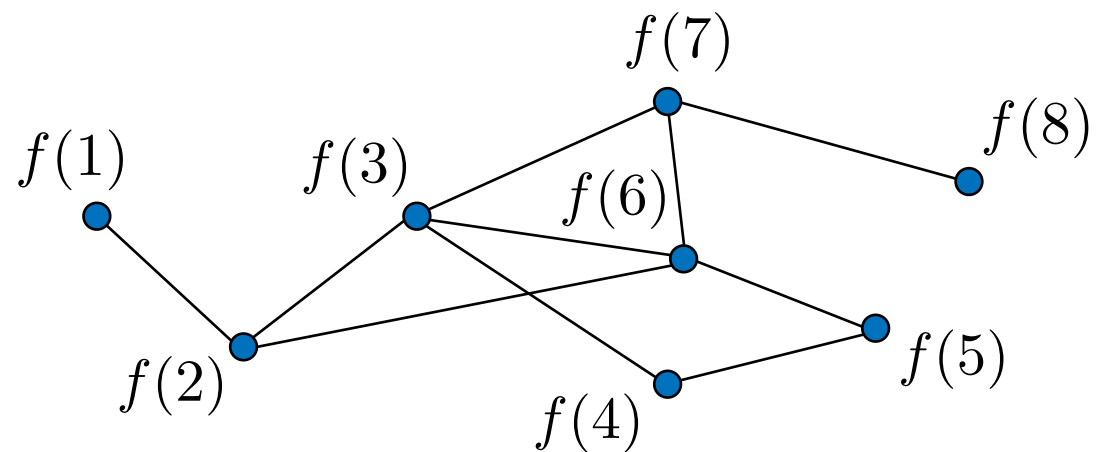
D

W

L

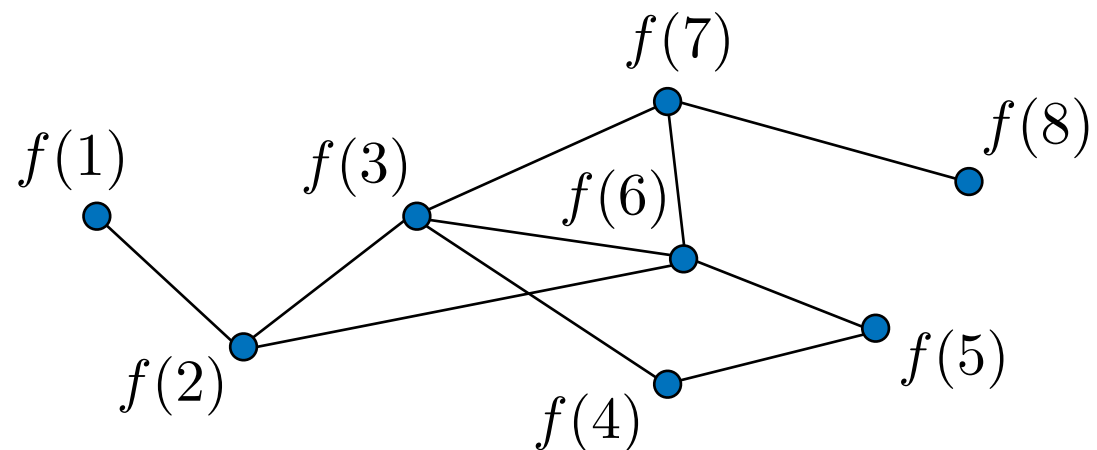
- symmetric
- off-diagonal entries non-positive
- rows sum up to zero

Graph Laplacian



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

Graph Laplacian

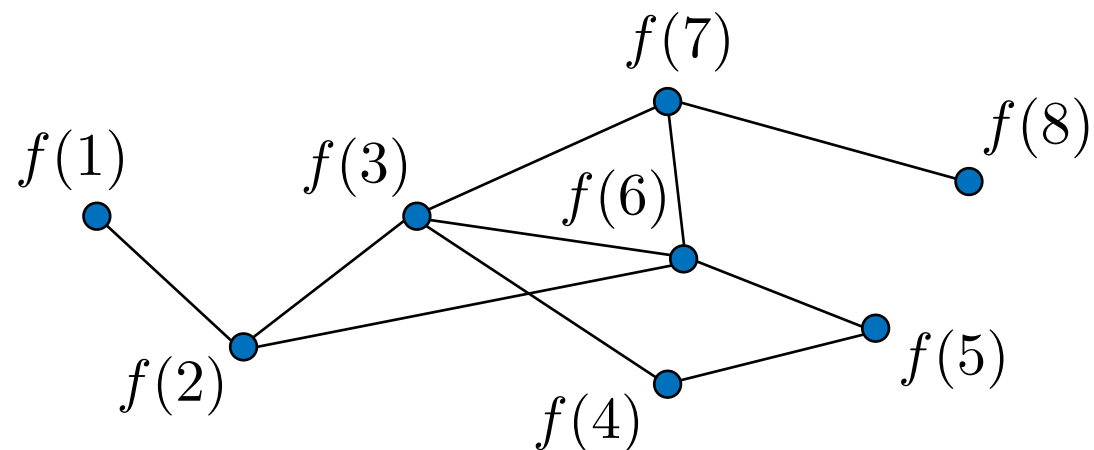


graph signal $f : \mathcal{V} \rightarrow \mathbb{R}$

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

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

Graph Laplacian



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

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

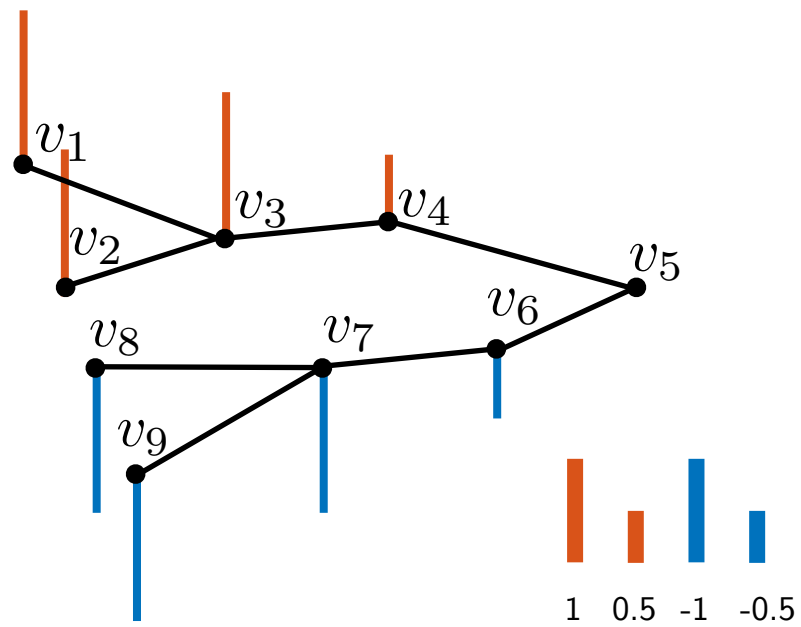
$$\begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{pmatrix}^T \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 &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

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

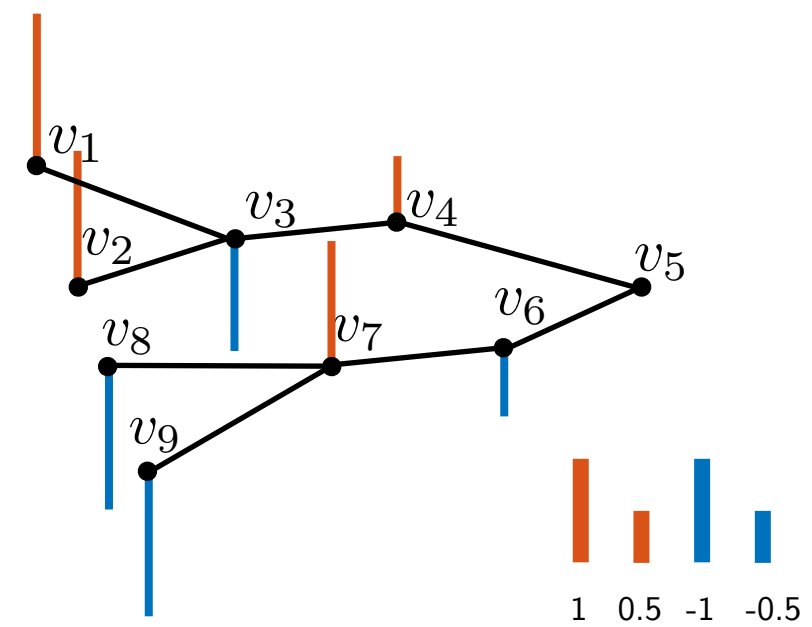
$$f^T Lf = \frac{1}{2} \sum_{i,j=1}^N W_{ij} (f(i) - f(j))^2$$

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

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

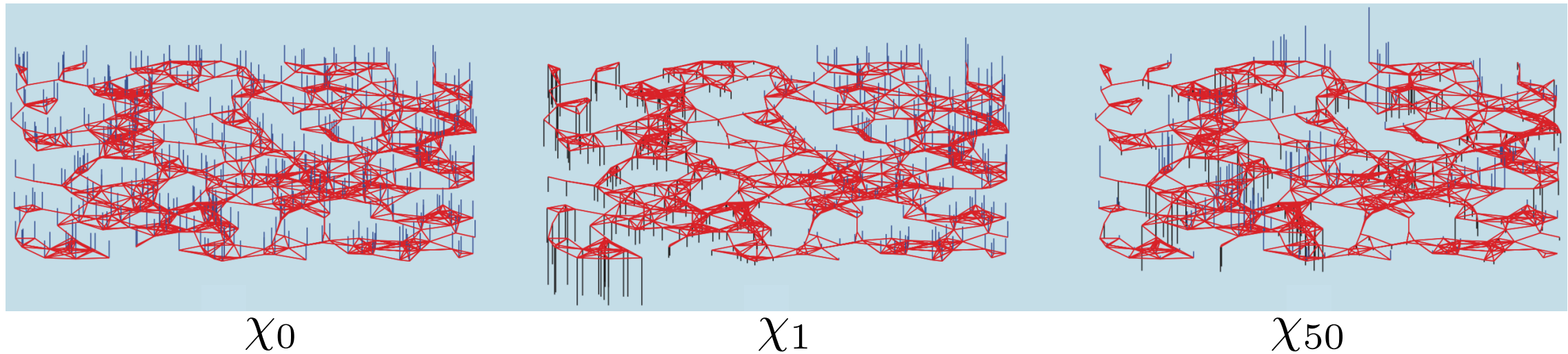
Graph Laplacian

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

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

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

Graph Fourier transform

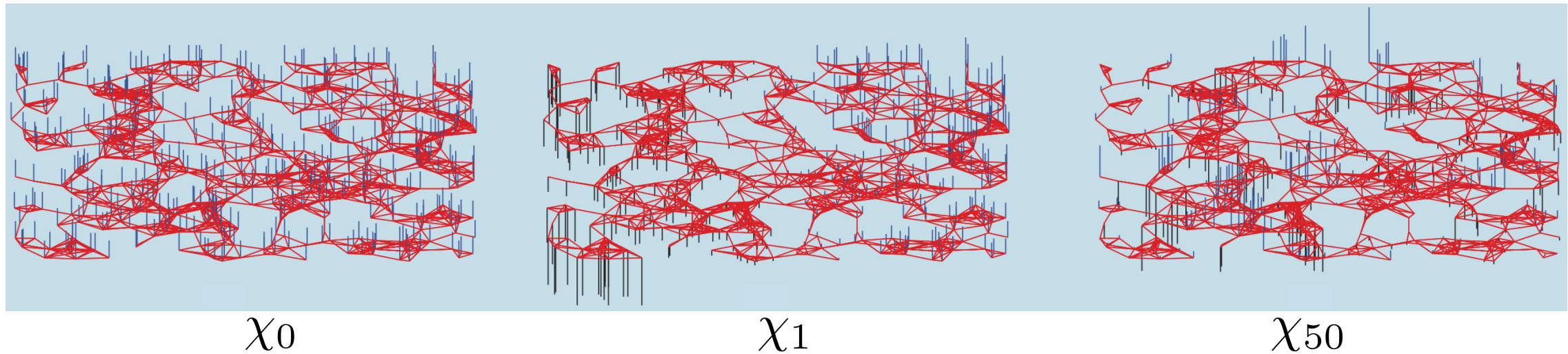


χ_0

χ_1

χ_{50}

Graph Fourier transform



low frequency

high frequency

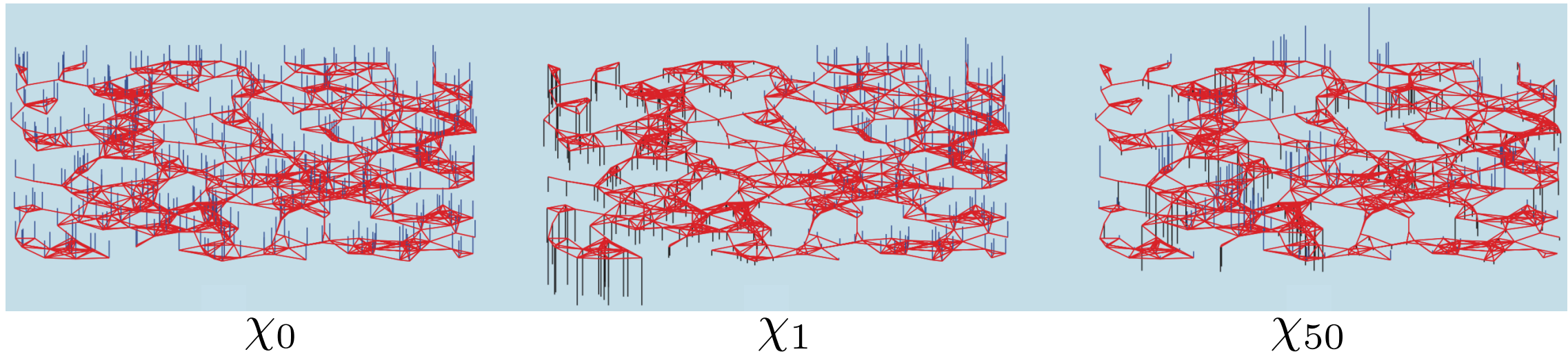
$$L = \chi \Lambda \chi^T$$

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

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

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

Graph Fourier transform



low frequency

high frequency

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

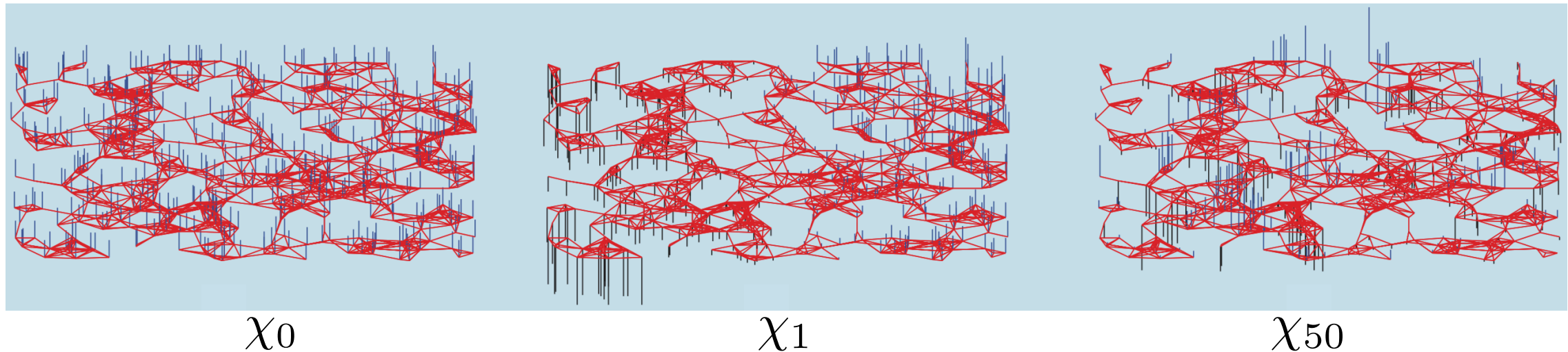
$$\chi_0^T L \chi_0 = \lambda_0 = 0$$

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

graph Fourier transform:

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

Graph Fourier transform



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

low frequency

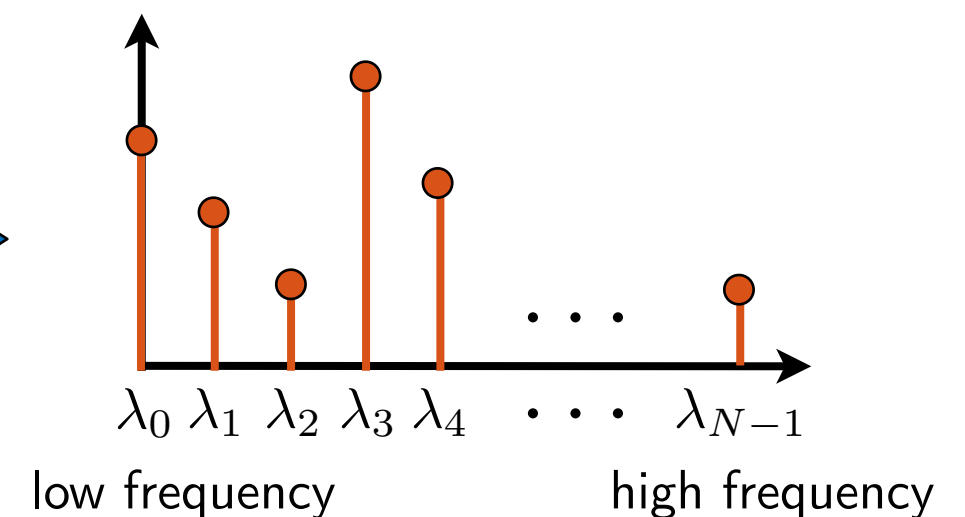
$$\chi_0^T L \chi_0 = \lambda_0 = 0$$

high frequency

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

graph Fourier transform:

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



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: χ_ℓ



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

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: χ_ℓ



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

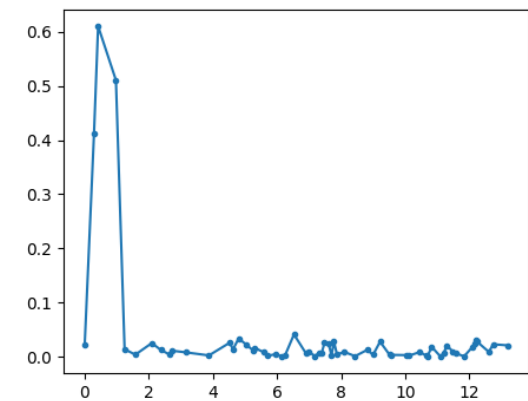
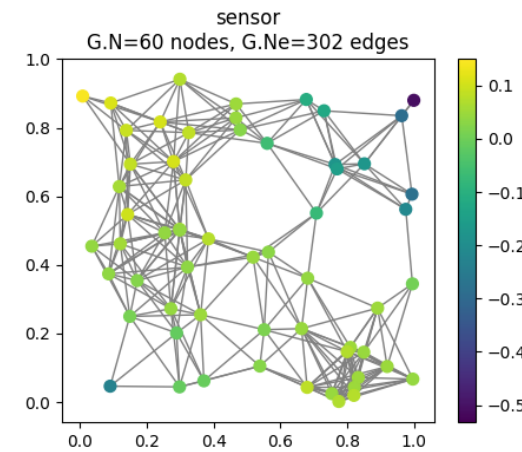
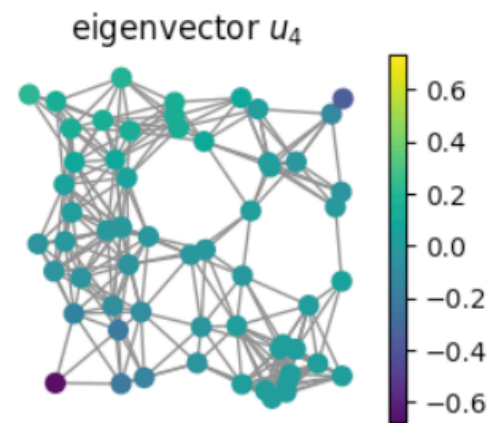
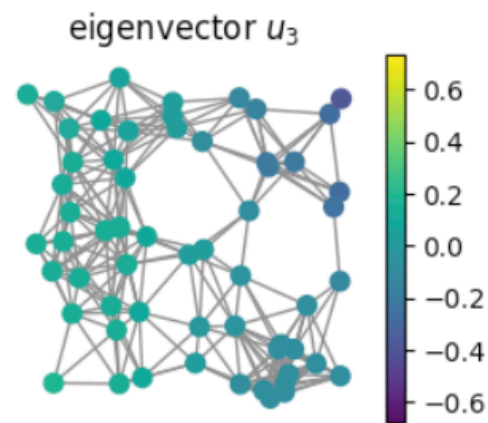
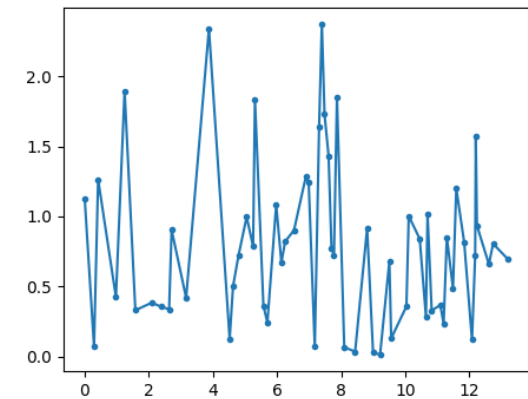
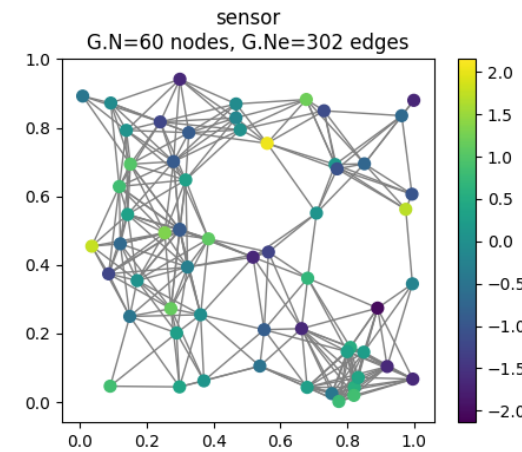
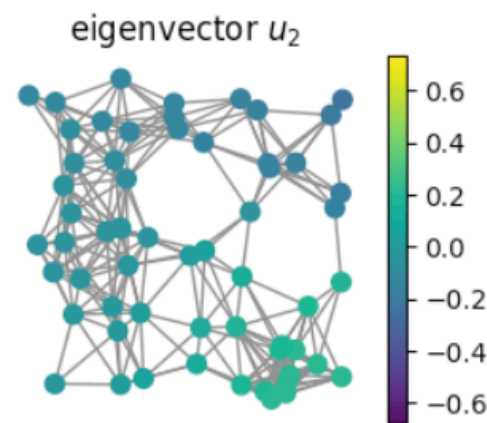
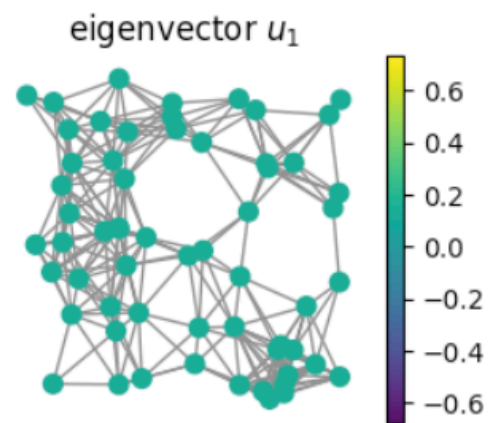
Graph FT: $\hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i)$

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

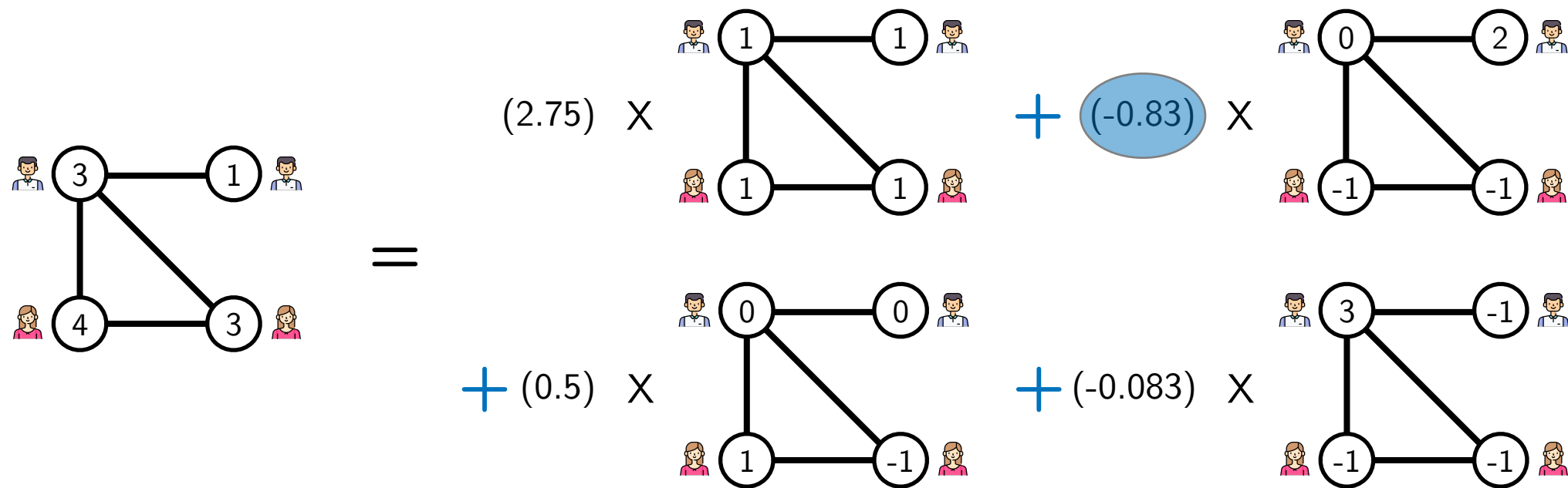
Example on synthetic signals

GFT:

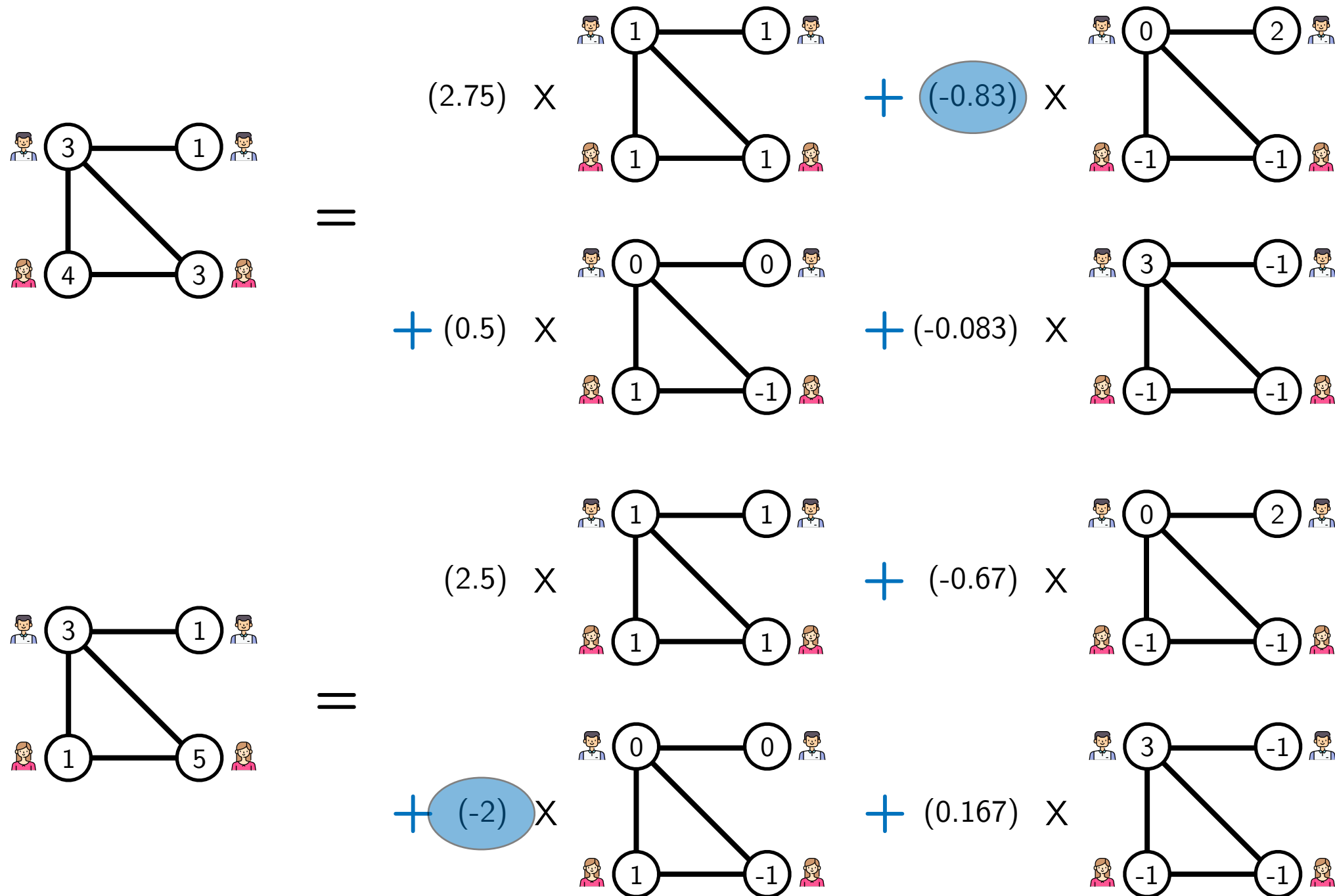
$$\hat{f}(\ell) = \langle \chi_\ell, f \rangle : \begin{bmatrix} | & & | \\ \chi_0 & \cdots & \chi_{N-1} \\ | & & | \end{bmatrix}^T \begin{bmatrix} | \\ f \\ | \end{bmatrix}$$



Example on movie ratings



Example on movie ratings

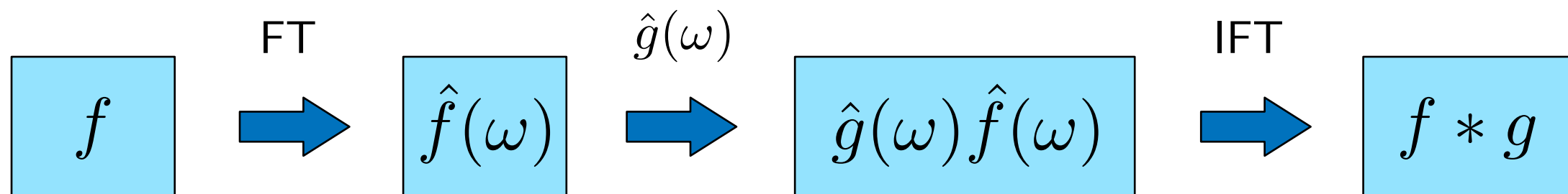


Classical frequency filtering

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

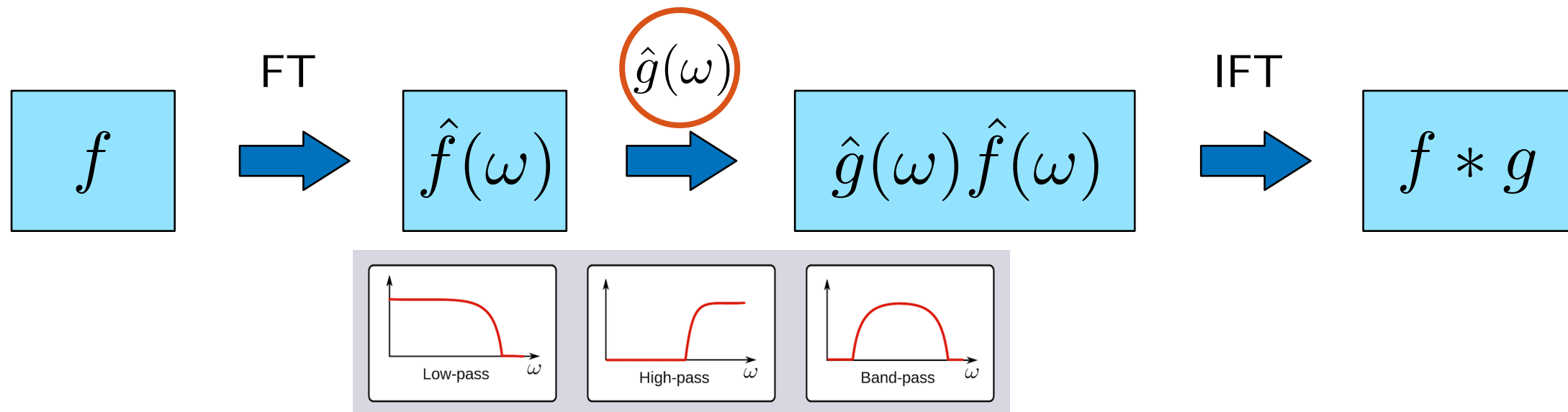
Classical frequency filtering

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



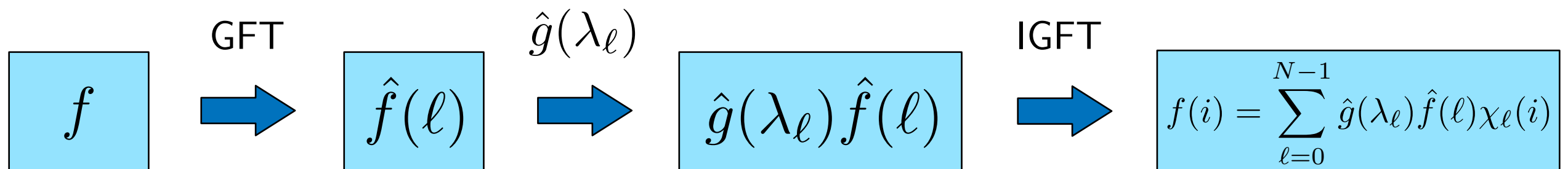
Classical frequency filtering

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



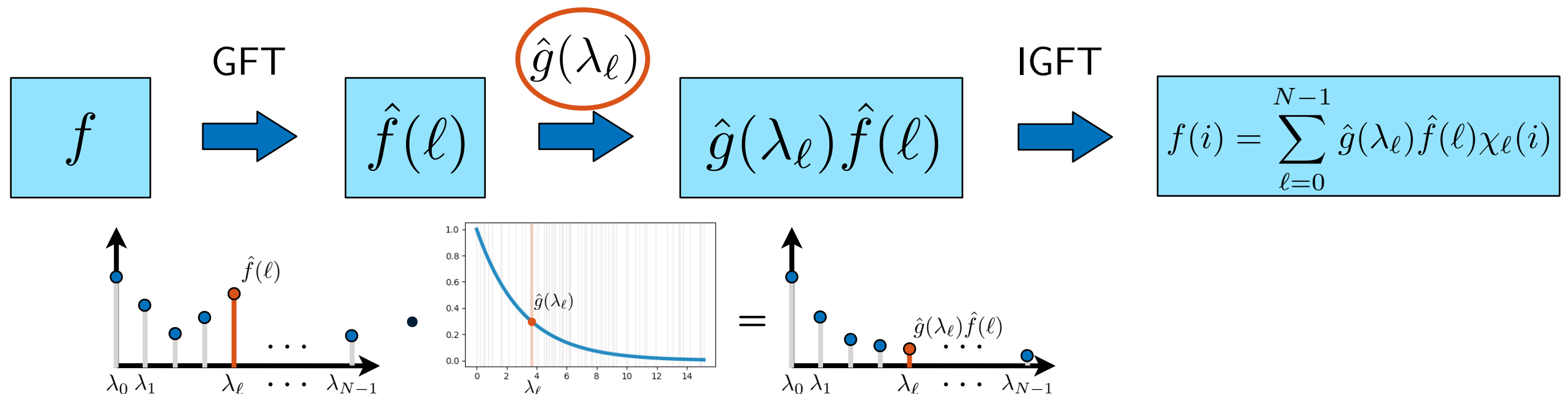
Graph spectral filtering

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



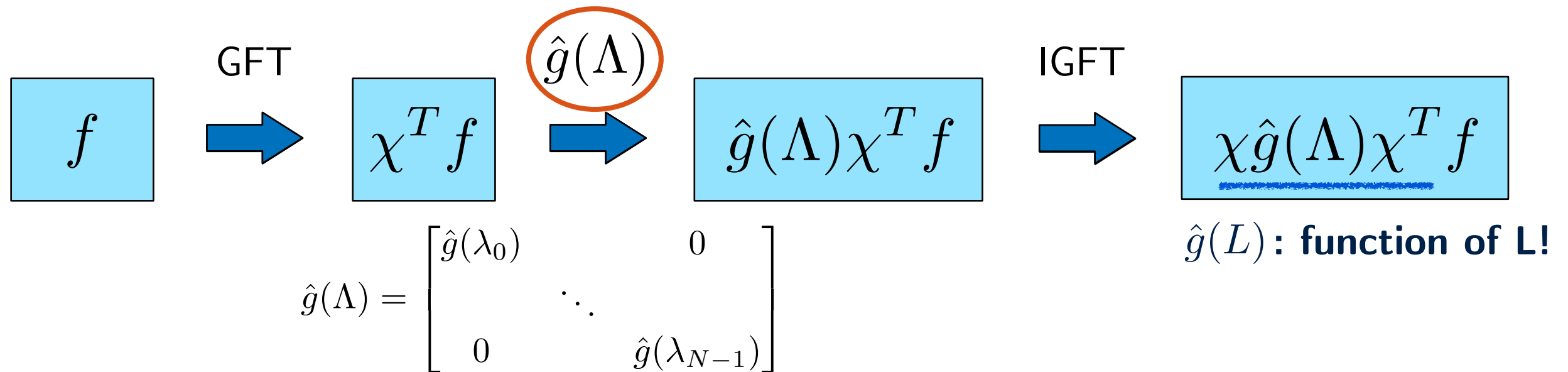
Graph spectral filtering

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



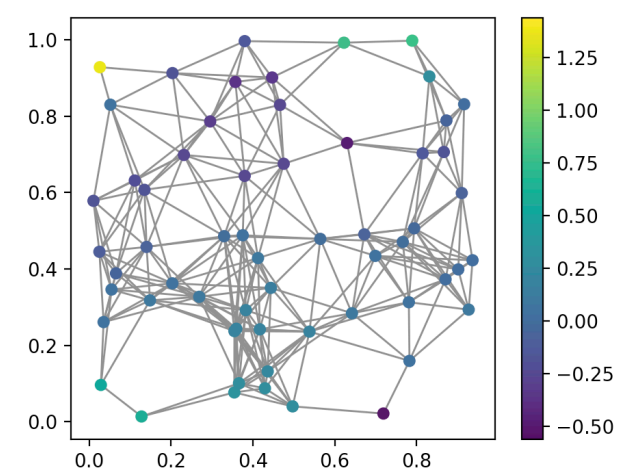
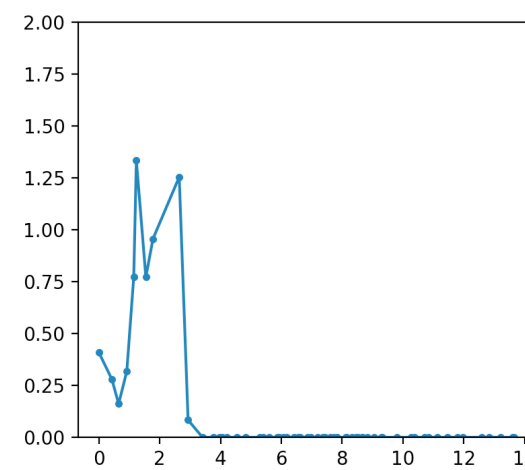
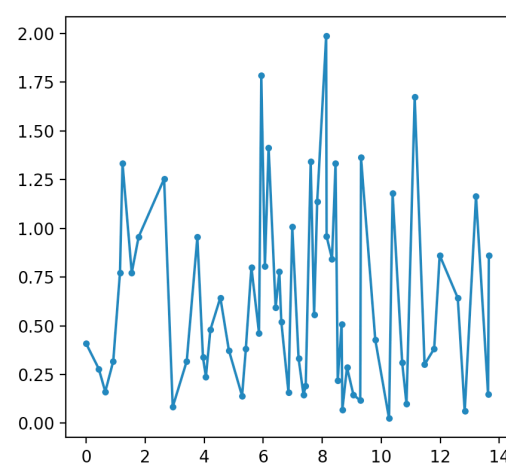
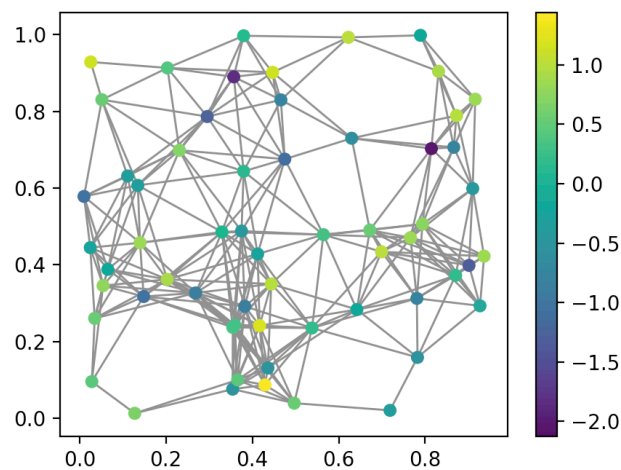
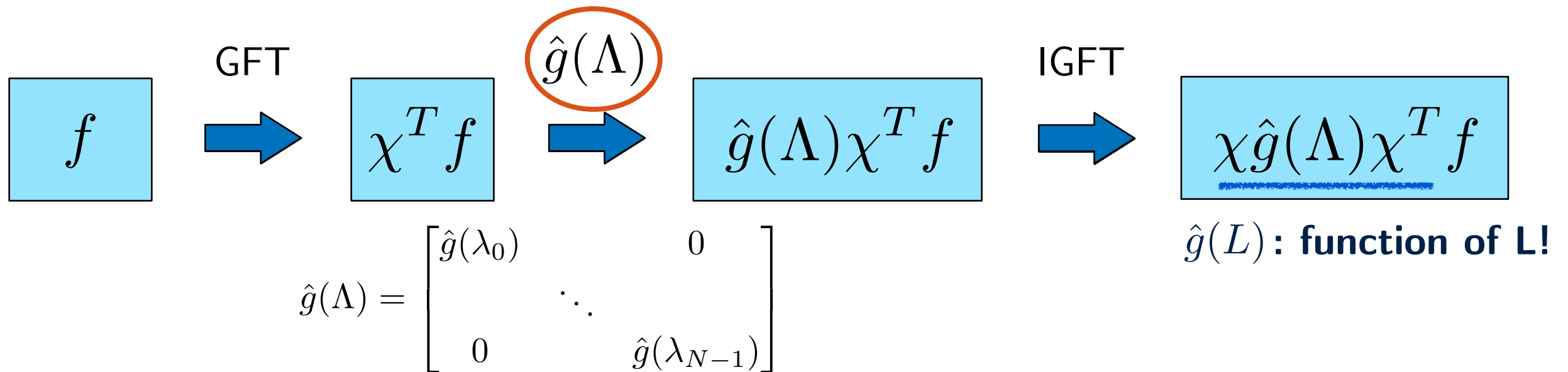
Graph spectral filtering

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

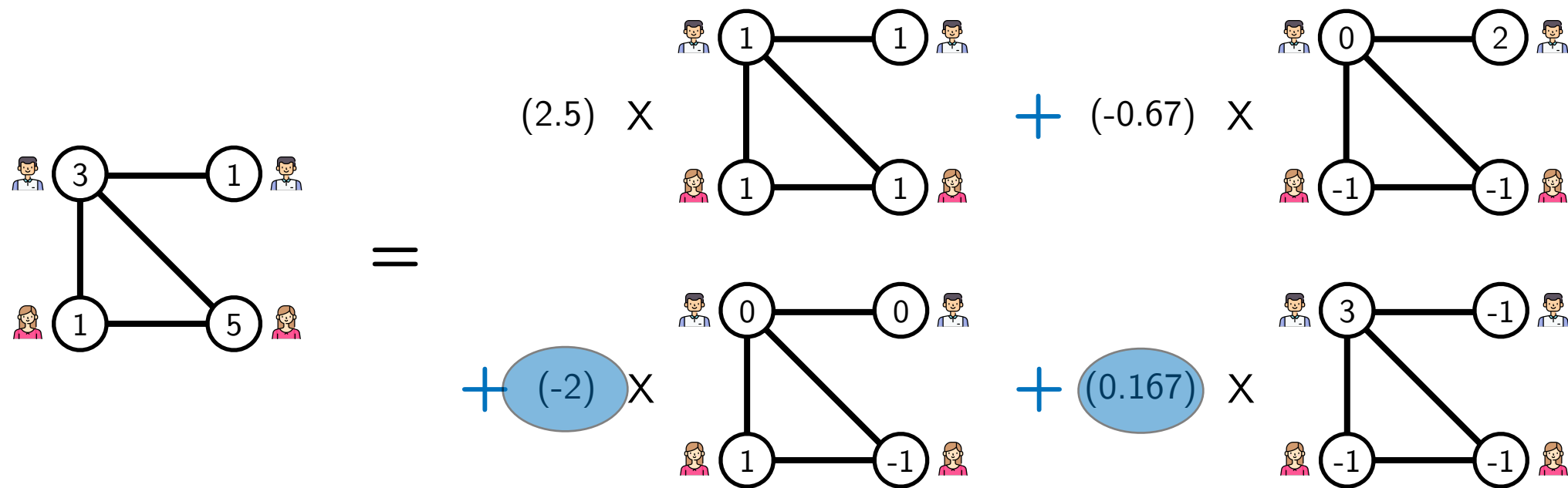


Graph spectral filtering

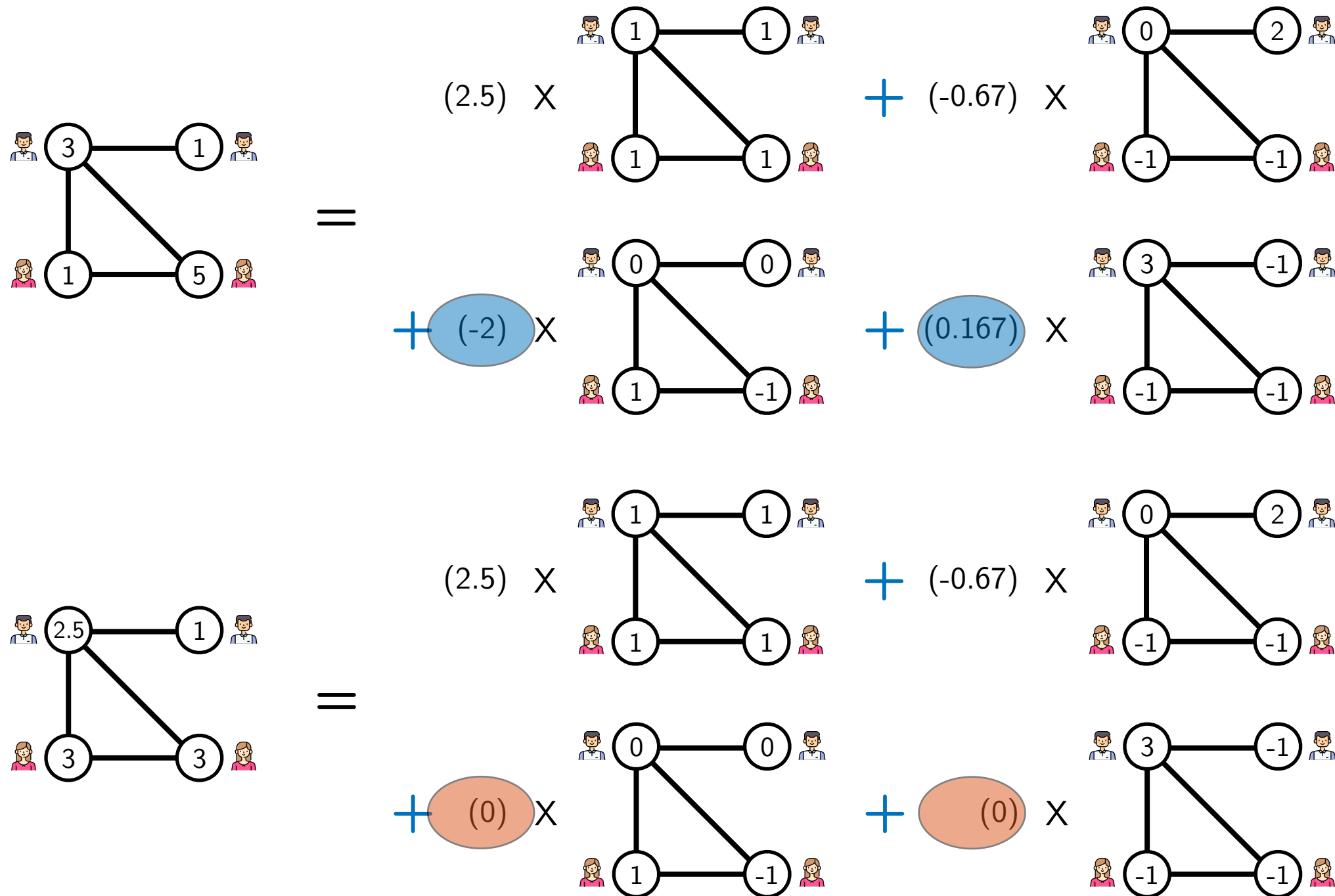
$$\text{GFT: } \hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^N \chi_\ell^*(i) f(i) \quad f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$$



Example on movie ratings

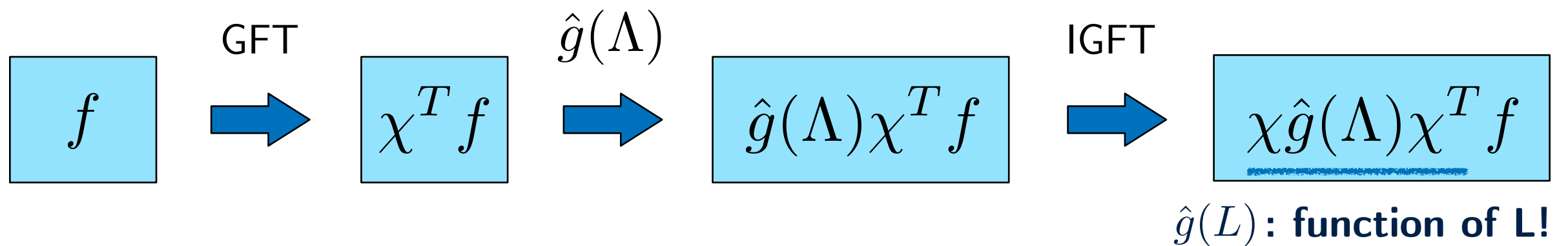


Example on movie ratings



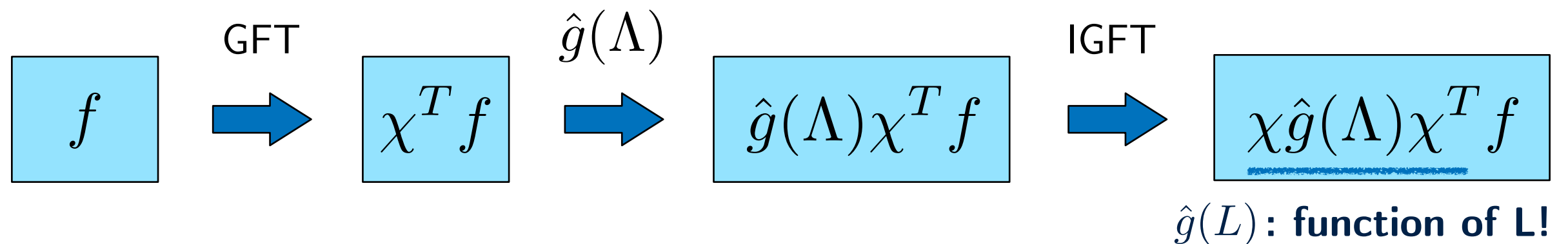
Graph spectral filtering

- Filters can be designed as functions of graph Laplacian



Graph spectral filtering

- Filters can be designed as functions of graph Laplacian



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

Convolution on graphs

classical convolution

time domain

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

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Convolution on graphs

classical convolution

time domain

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

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Convolution on graphs

classical convolution

time domain

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



frequency domain

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

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

spatial (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

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

spatial (node) domain

$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \boxed{\hat{g}(L) f} \quad \text{convolution} \\ = \text{filtering}$$



graph spectral domain

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

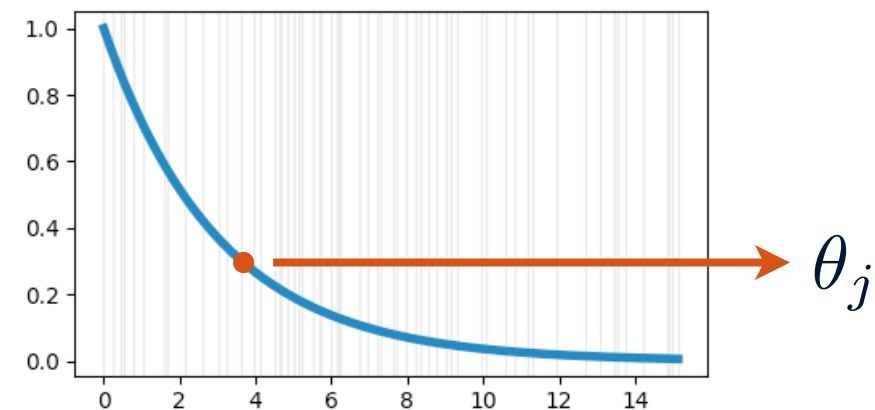
A non-parametric filter

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



learning a non-parametric filter:

$$\hat{g}_\theta(\Lambda) = \text{diag}(\theta), \quad \theta \in \mathbb{R}^N$$



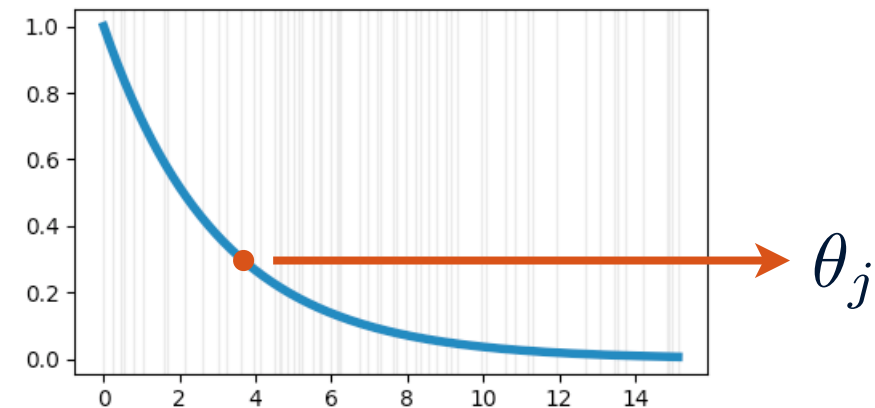
A non-parametric filter

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



learning a non-parametric filter:

$$\hat{g}_\theta(\Lambda) = \text{diag}(\theta), \quad \theta \in \mathbb{R}^N$$



- convolution expressed in the graph spectral domain
- no localisation in the spatial (node) domain
- computationally expensive

A parametric filter

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



parametric filter as polynomial of Laplacian

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



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

A parametric filter

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



parametric filter as polynomial of Laplacian

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

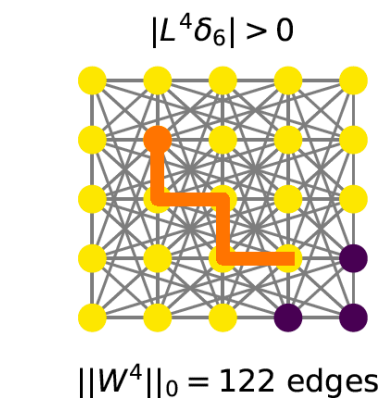
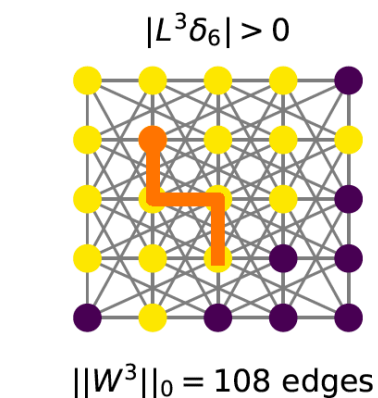
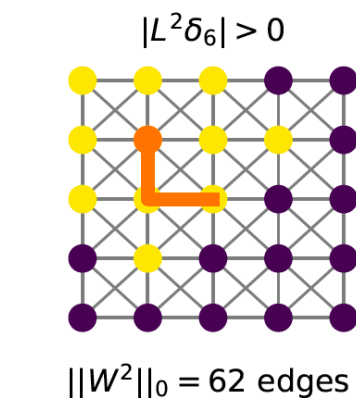
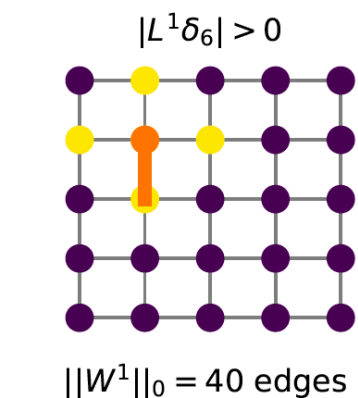
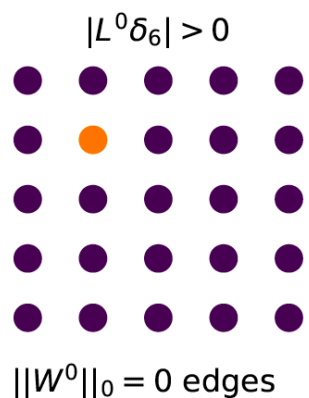
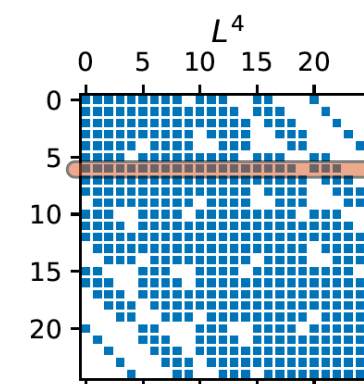
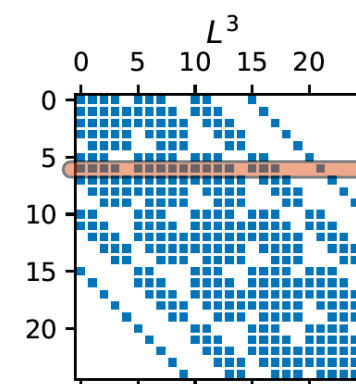
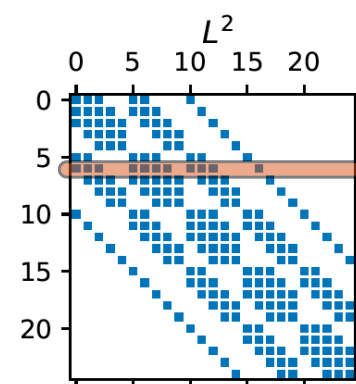
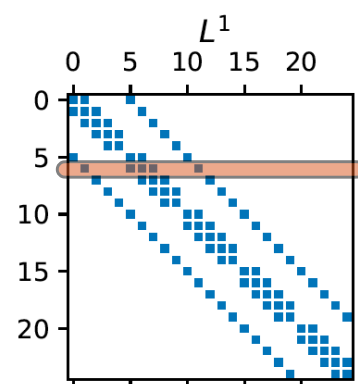
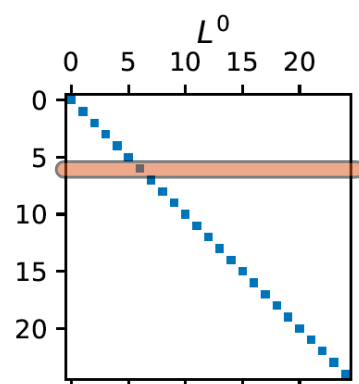


$$\hat{g}_\theta(L) = \sum_{j=0}^K \theta_j L^j$$

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 $(L^K)_{ij} = 0$

(source: M. Defferrard)

A parametric filter

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



parametric filter as polynomial of Laplacian

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



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

A parametric filter

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



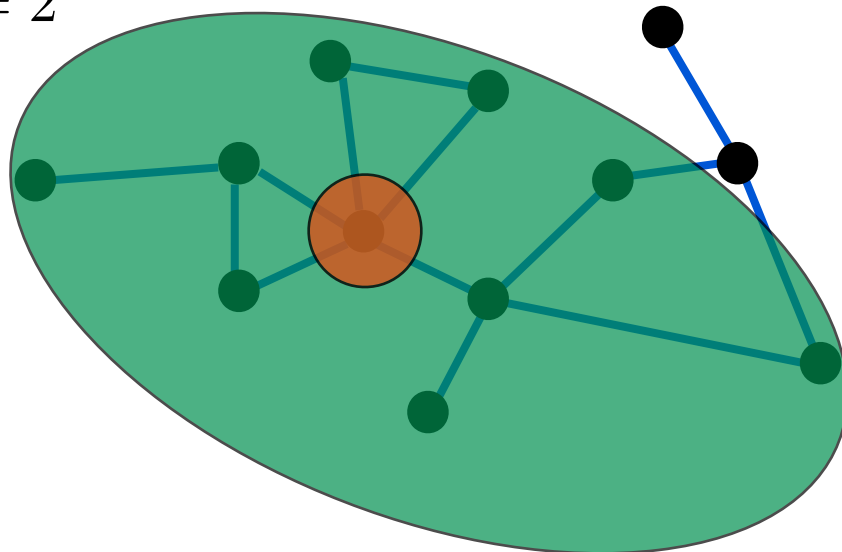
parametric filter as polynomial of Laplacian

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



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

$K = 2$



- localisation within K -hop neighbourhood

A parametric filter

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



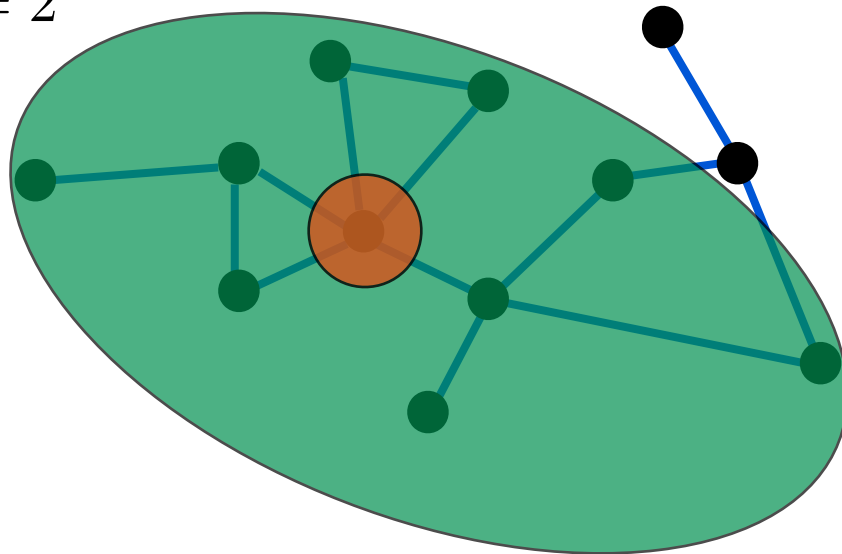
parametric filter as polynomial of Laplacian

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



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

$K = 2$



- localisation within K -hop neighbourhood
- Chebyshev approximation enables efficient computation via recursive multiplication with scaled Laplacian

$$\tilde{L} = \frac{2}{\lambda_{N-1}} L - I$$

A simplified parametric filter

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



simplified parametric filter

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

normalised Laplacian

$$\begin{aligned} L_{\text{norm}} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \\ &= D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} \\ &= I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = I - W_{\text{norm}} \end{aligned}$$

$K = 1$

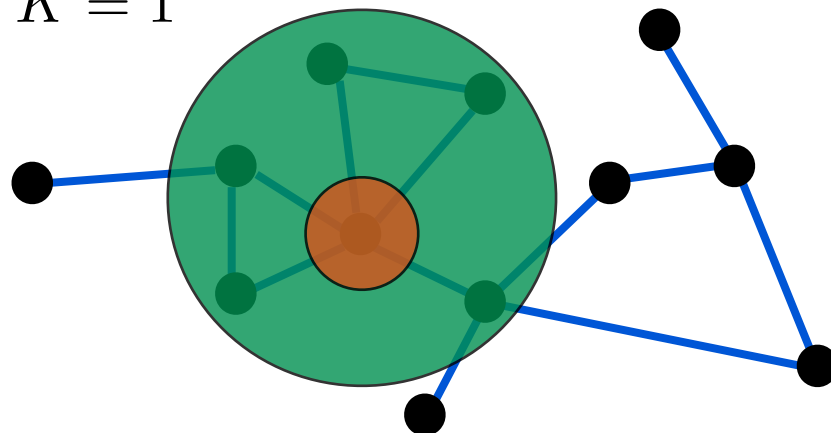
normalised Laplacian



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

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

$K = 1$



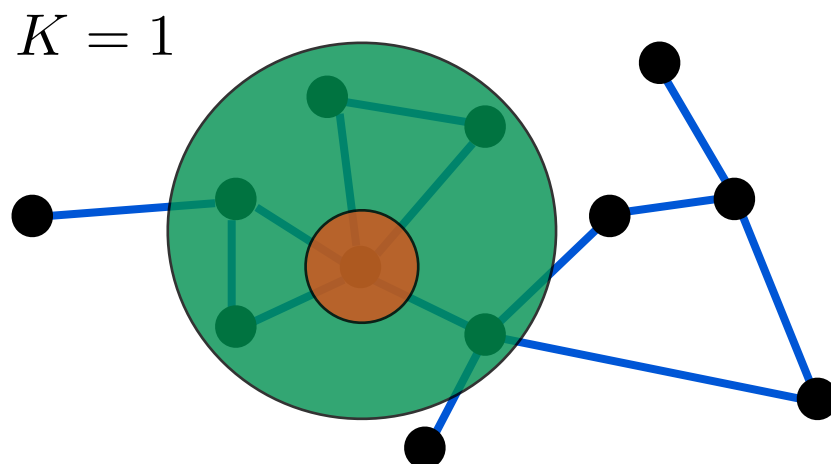
A simplified parametric filter

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



simplified parametric filter

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



normalised Laplacian

$$\begin{aligned} L_{\text{norm}} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \\ &= D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} \\ &= I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = I - W_{\text{norm}} \end{aligned}$$

$K = 1$

normalised Laplacian



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

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

$$\alpha = \theta_0 = -\theta_1$$



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

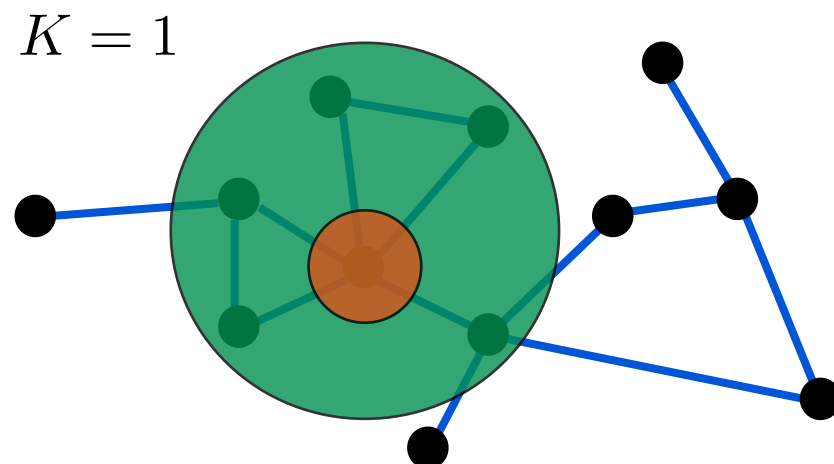
A simplified parametric filter

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



simplified parametric filter

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



$K = 1$
normalised Laplacian



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

(localisation within 1-hop neighbourhood)

$$\alpha = \theta_0 = -\theta_1$$



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

renormalisation



$$\Rightarrow \alpha (\tilde{D}^{-\frac{1}{2}} \tilde{W} \tilde{D}^{-\frac{1}{2}})$$

normalised Laplacian

$$\begin{aligned} L_{\text{norm}} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \\ &= D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} \\ &= I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = I - W_{\text{norm}} \end{aligned}$$

renormalisation

$$\tilde{W} = W + I \quad \tilde{D} = D + I$$

A simplified parametric filter

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



simplified parametric filter

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

A simplified parametric filter

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

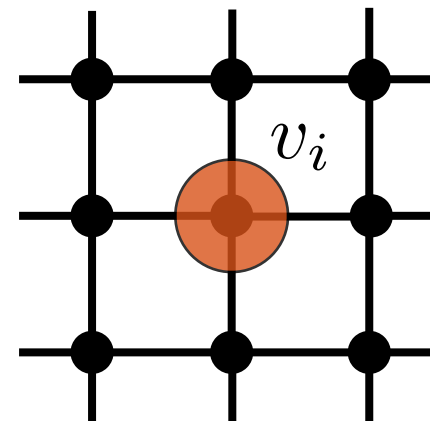


simplified parametric filter

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



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



A simplified parametric filter

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



simplified parametric filter

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

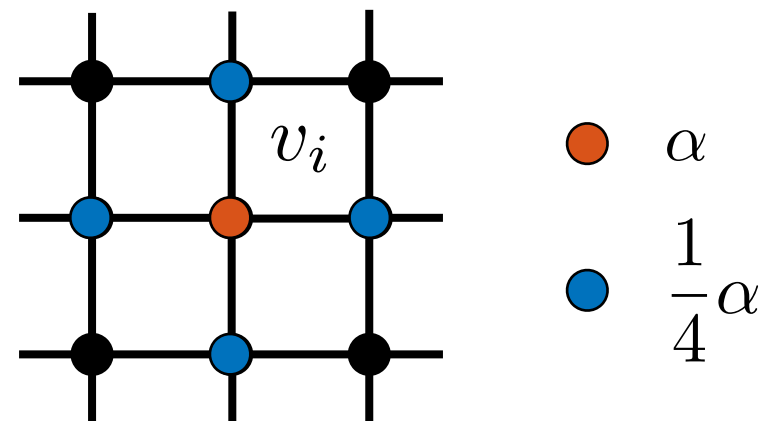


$$y_i = \alpha f_i + \alpha \frac{1}{\sqrt{d_i}} \sum_{j:(i,j) \in \mathcal{E}} w_{ij} \frac{1}{\sqrt{d_j}} f_j$$



unitary edge weights

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



A simplified parametric filter

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



simplified parametric filter

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



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

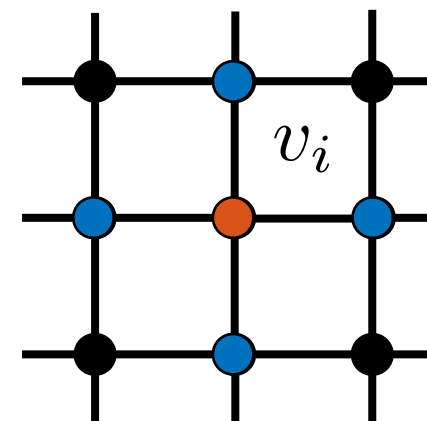


unitary edge weights

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

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



● α
● $\frac{1}{4}\alpha$

Convolution on graphs - Remarks

- 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 (node)** domain
 - **simplified filter:** $y = \hat{g}_\theta(L) f = \alpha(\tilde{D}^{-\frac{1}{2}} \tilde{W} \tilde{D}^{-\frac{1}{2}}) f$
 - **interpretation:** at each layer nodes exchange information in 1-hop neighbourhood
 - **more generally:** receptive field size determined by degree of polynomial

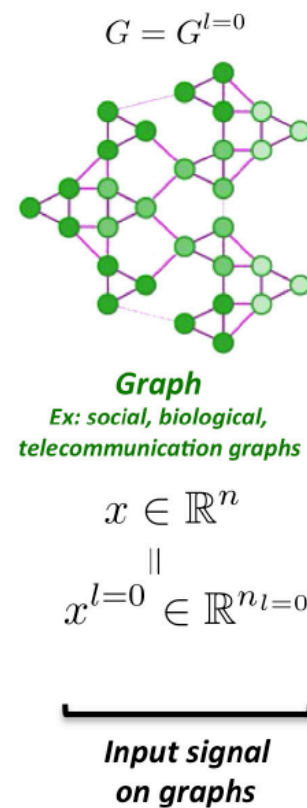
Convolution on graphs - Remarks

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

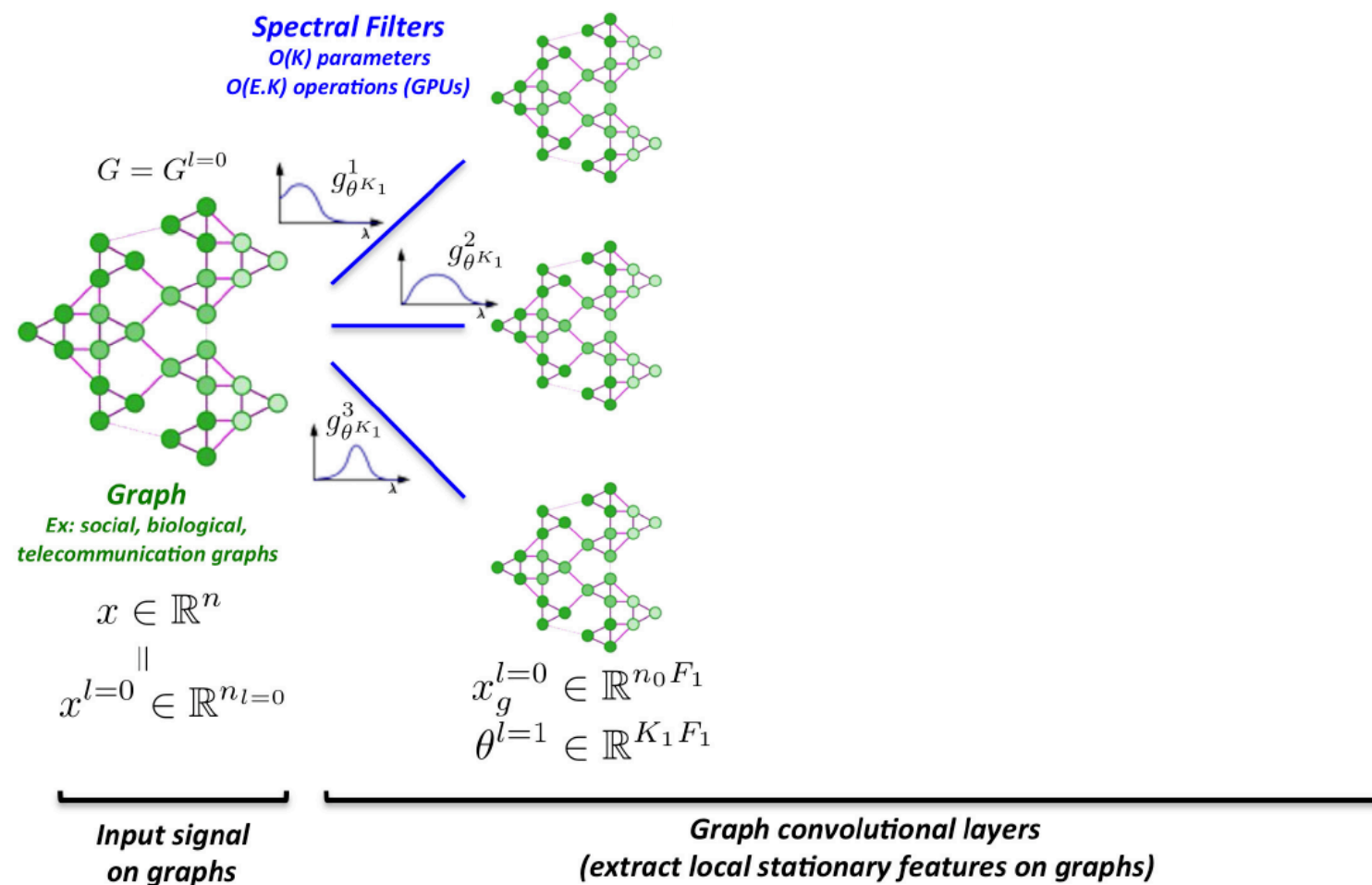
$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

- ..but can be implemented in the **spatial (node)** domain
 - **simplified filter:** $y = \hat{g}_\theta(L) f = \alpha(\tilde{D}^{-\frac{1}{2}} \tilde{W} \tilde{D}^{-\frac{1}{2}}) f$
 - **interpretation:** at each layer nodes exchange information in 1-hop neighbourhood
 - **more generally:** receptive field size determined by degree of polynomial
- Other possibilities exist (e.g., a direct spatial approach)

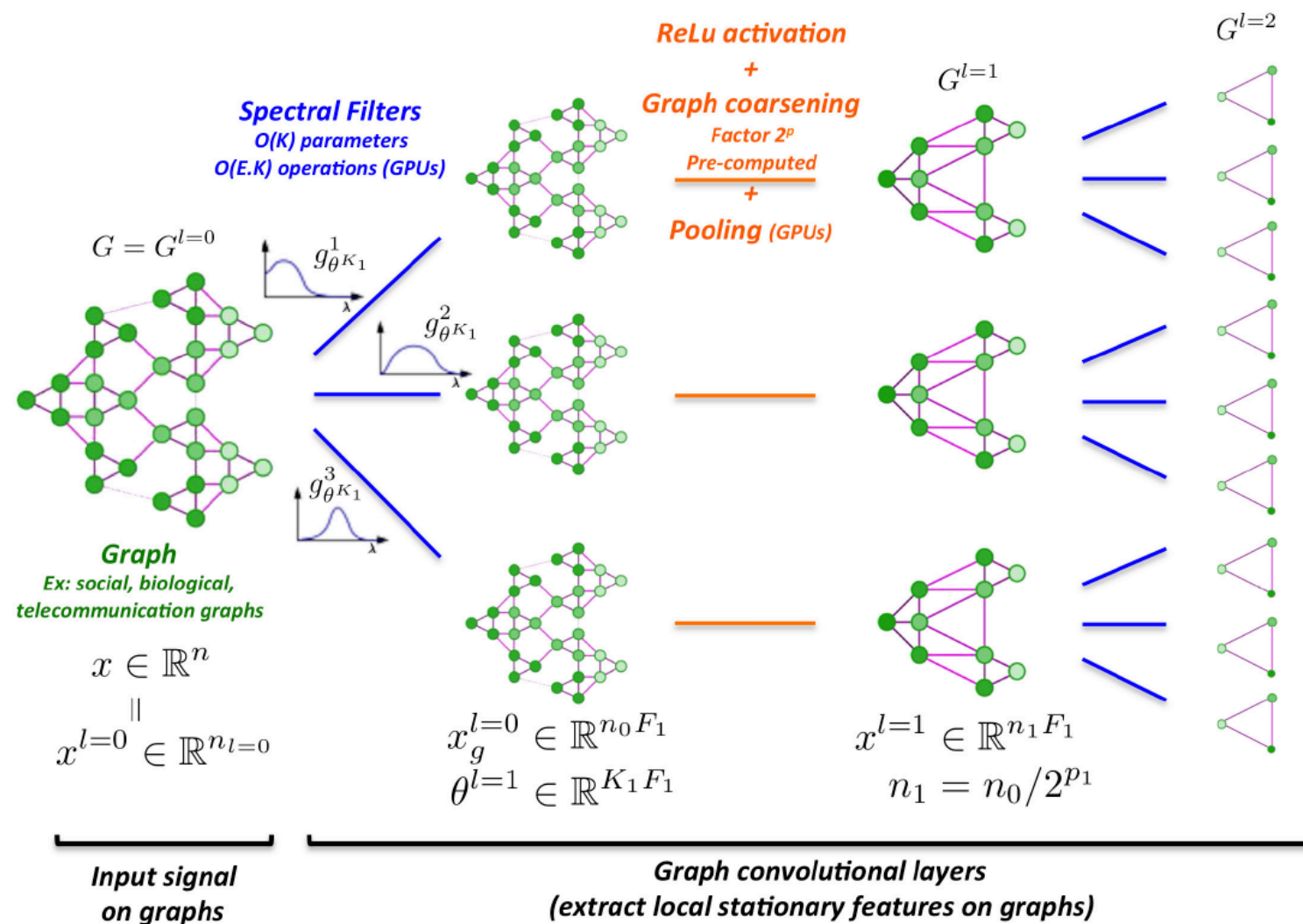
CNNs on graphs: ChebNet



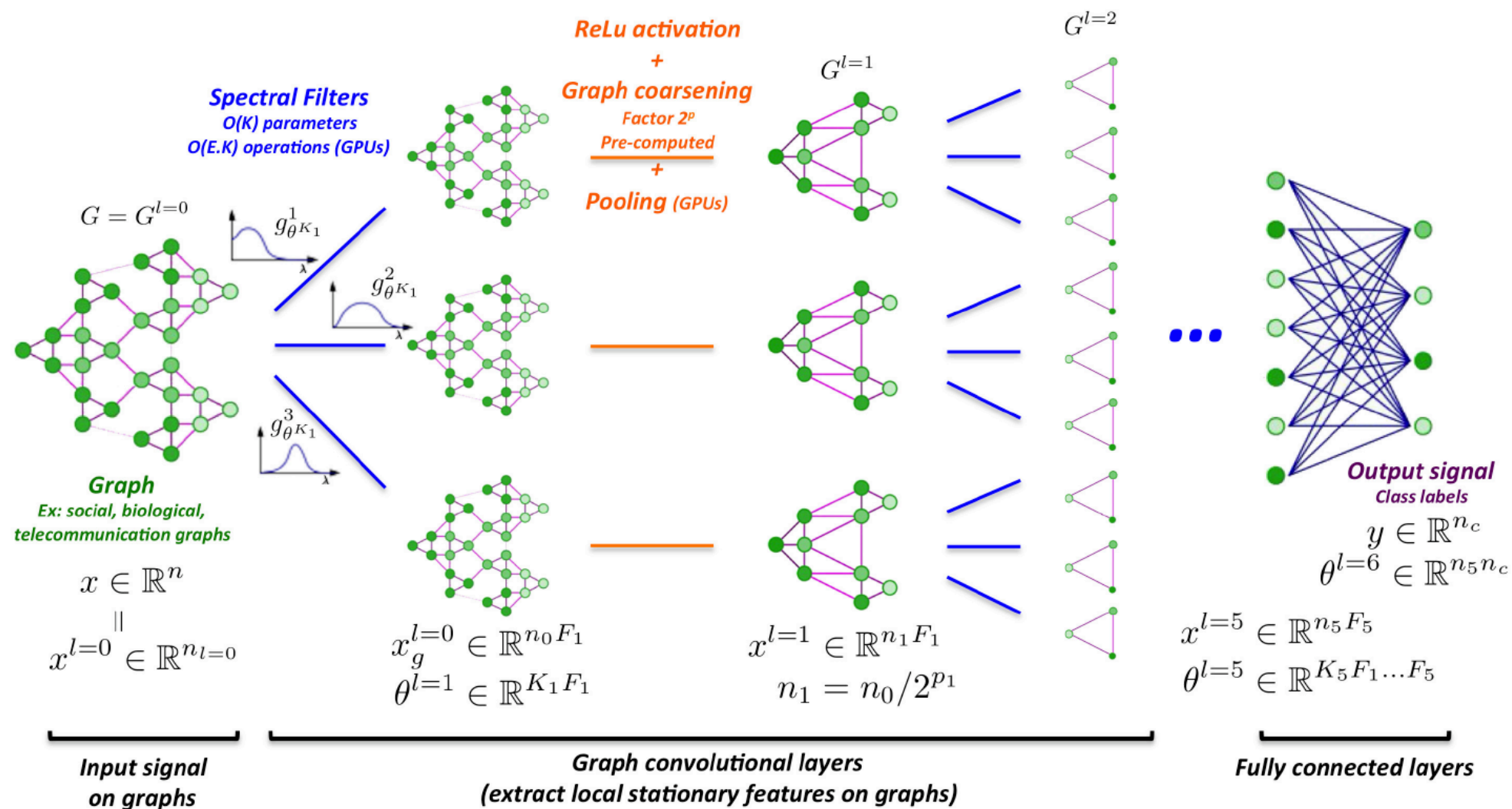
CNNs on graphs: ChebNet



CNNs on graphs: ChebNet

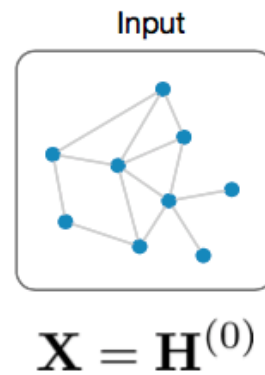


CNNs on graphs: ChebNet



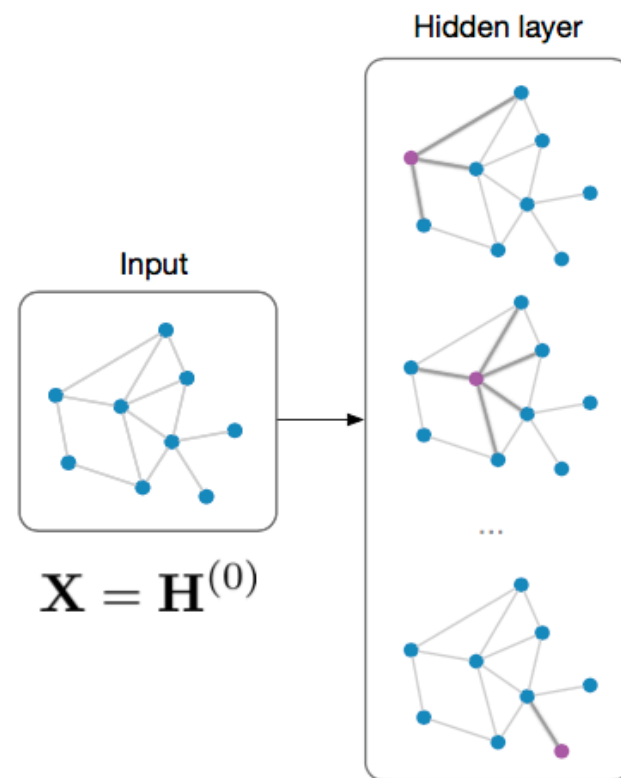
CNNs on graphs: GCN

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



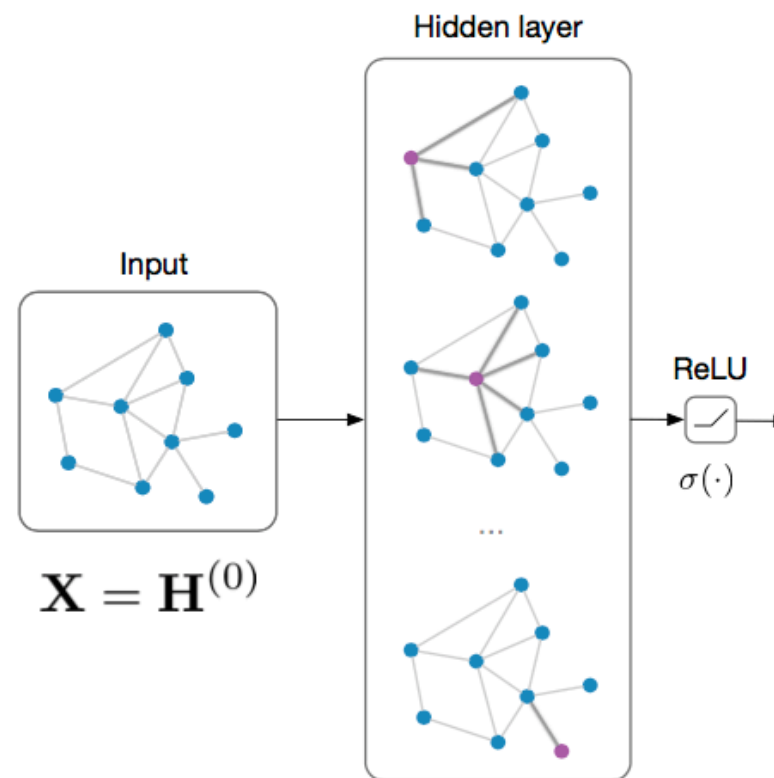
CNNs on graphs: GCN

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



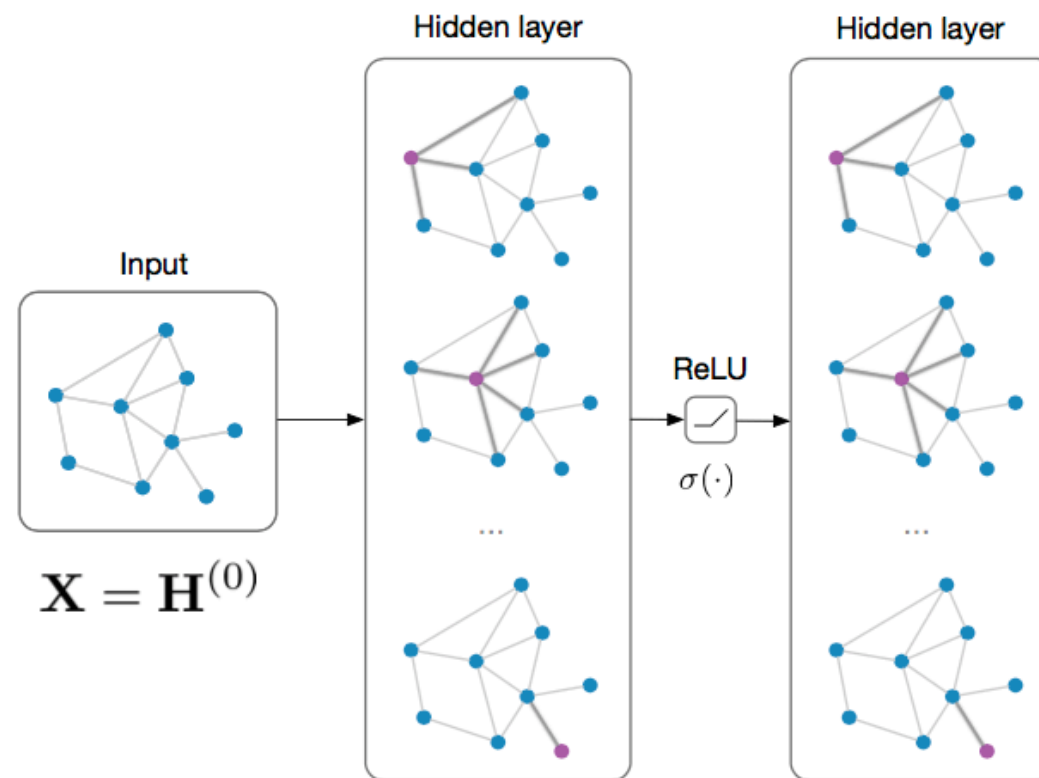
CNNs on graphs: GCN

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



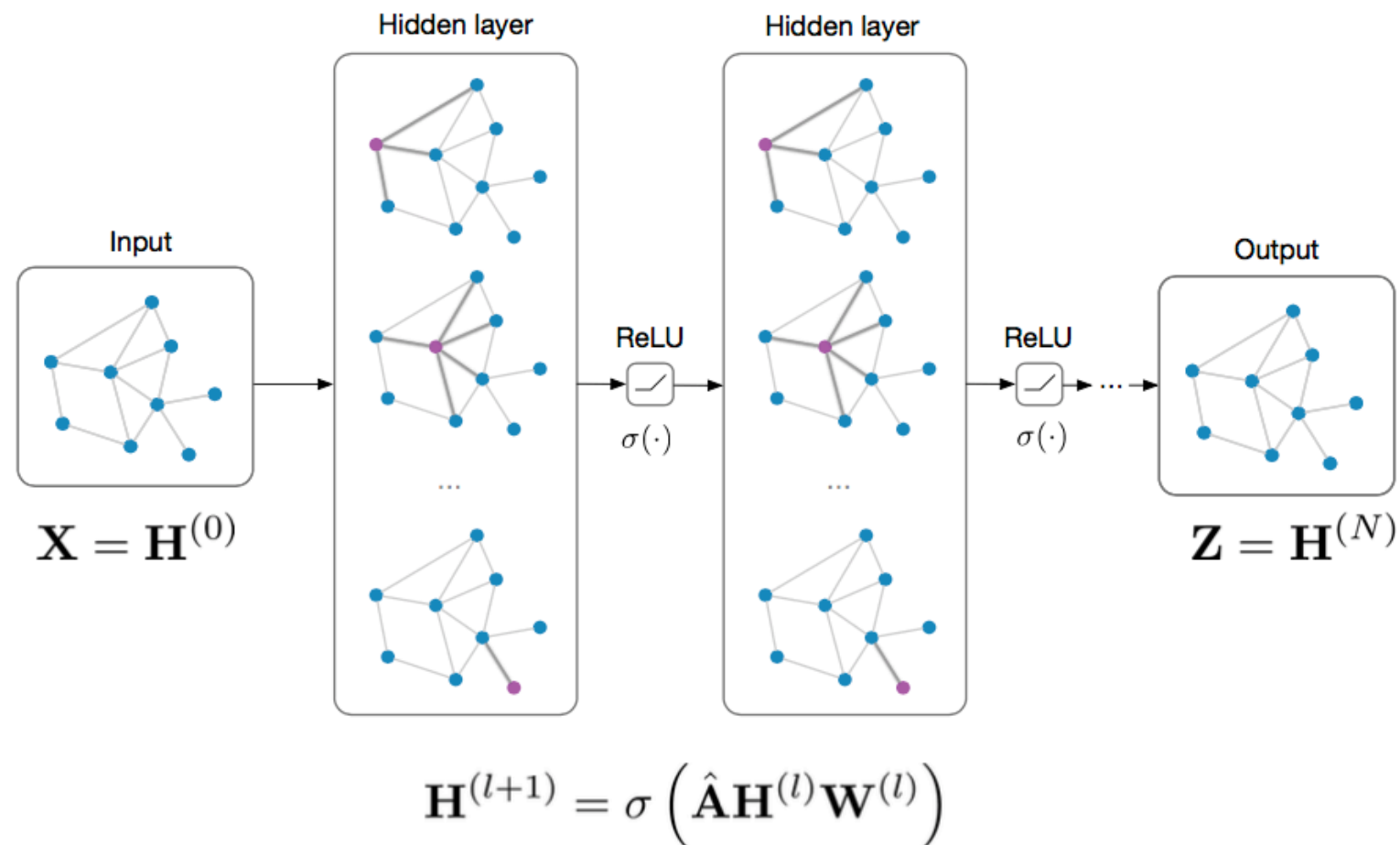
CNNs on graphs: GCN

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



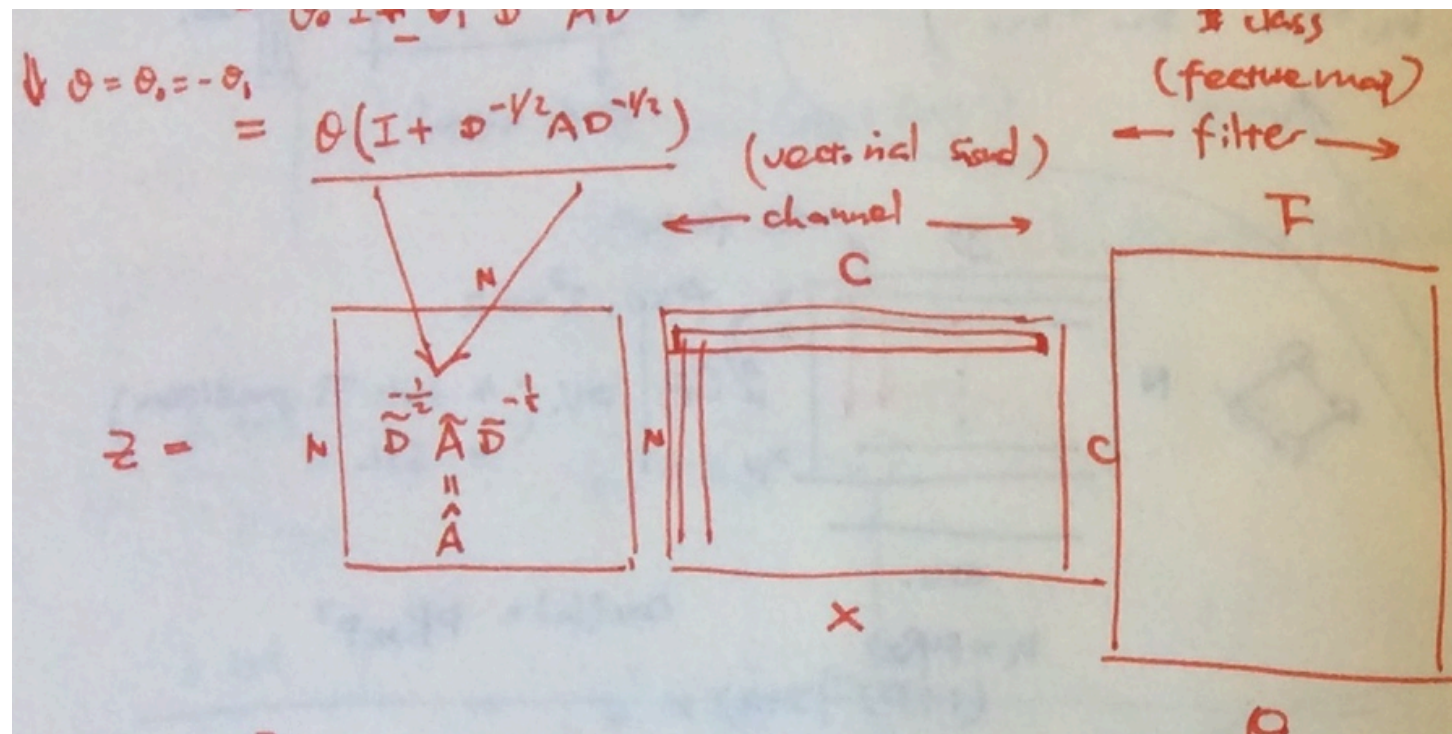
CNNs on graphs: GCN

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



CNNs on graphs: GCN

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



$$\mathbf{H}^{(l+1)} = \sigma \left(\hat{\mathbf{A}} \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right)$$

Implementing CNNs on graphs

- Node-level task
 - cross-entropy loss function for (semi-supervised) node classification

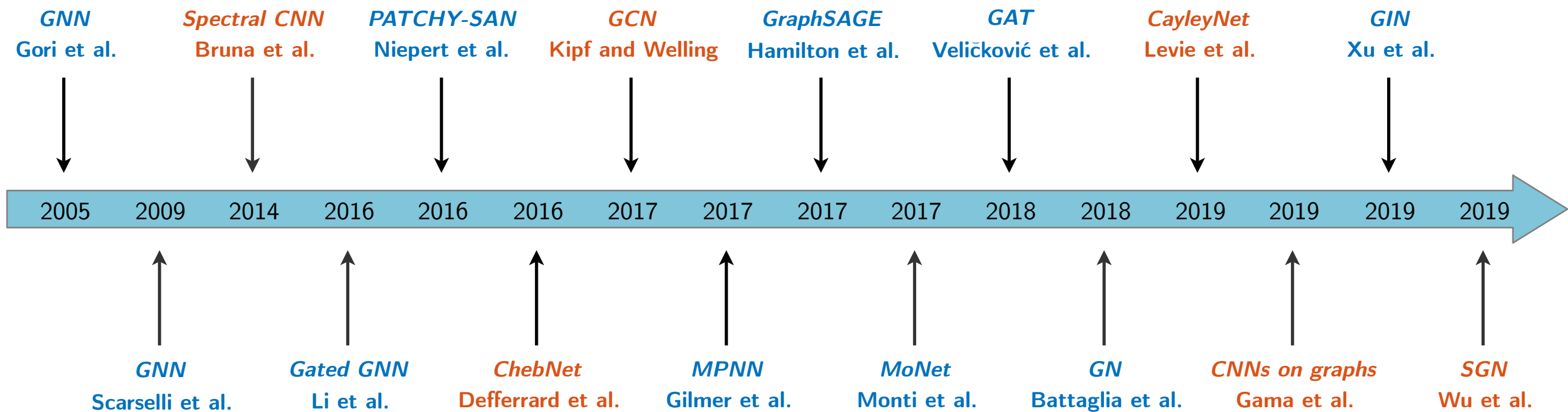
$$\mathcal{L} = - \sum_{l \in \mathcal{Y}_L} \sum_{f=1}^F Y_{lf} \ln Z_{lf}$$

Diagram illustrating the cross-entropy loss function for node classification:

- $l \in \mathcal{Y}_L$: set of labelled (training) nodes (indicated by a teal arrow)
- Y_{lf} : label groundtruth (indicated by a blue arrow)
- Z_{lf} : label prediction (final layer node representation) (indicated by an orange arrow)

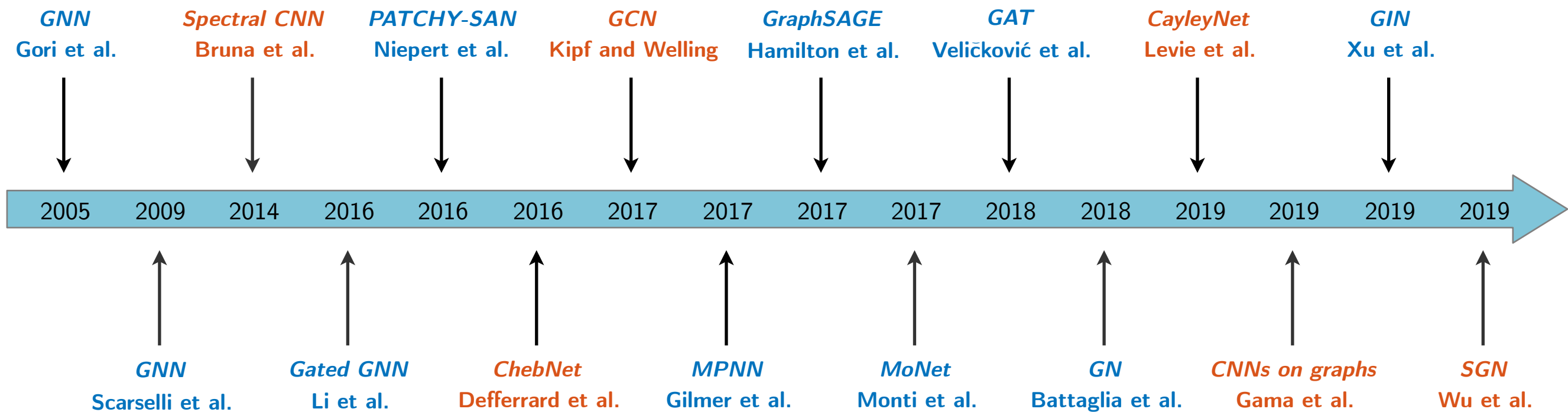
- training by minimising loss function and making predictions on testing nodes
- Factors influencing model behaviour
 - what label distribution favours GCN in this task?
 - what about perturbation of input graph topology?

(More generally) Graph neural networks



■ spectral (GSP) perspective ■ spatial perspective

(More generally) Graph neural networks



■ spectral (GSP) perspective ■ spatial perspective

more recently: **graph transformers** and **LLM-powered models**

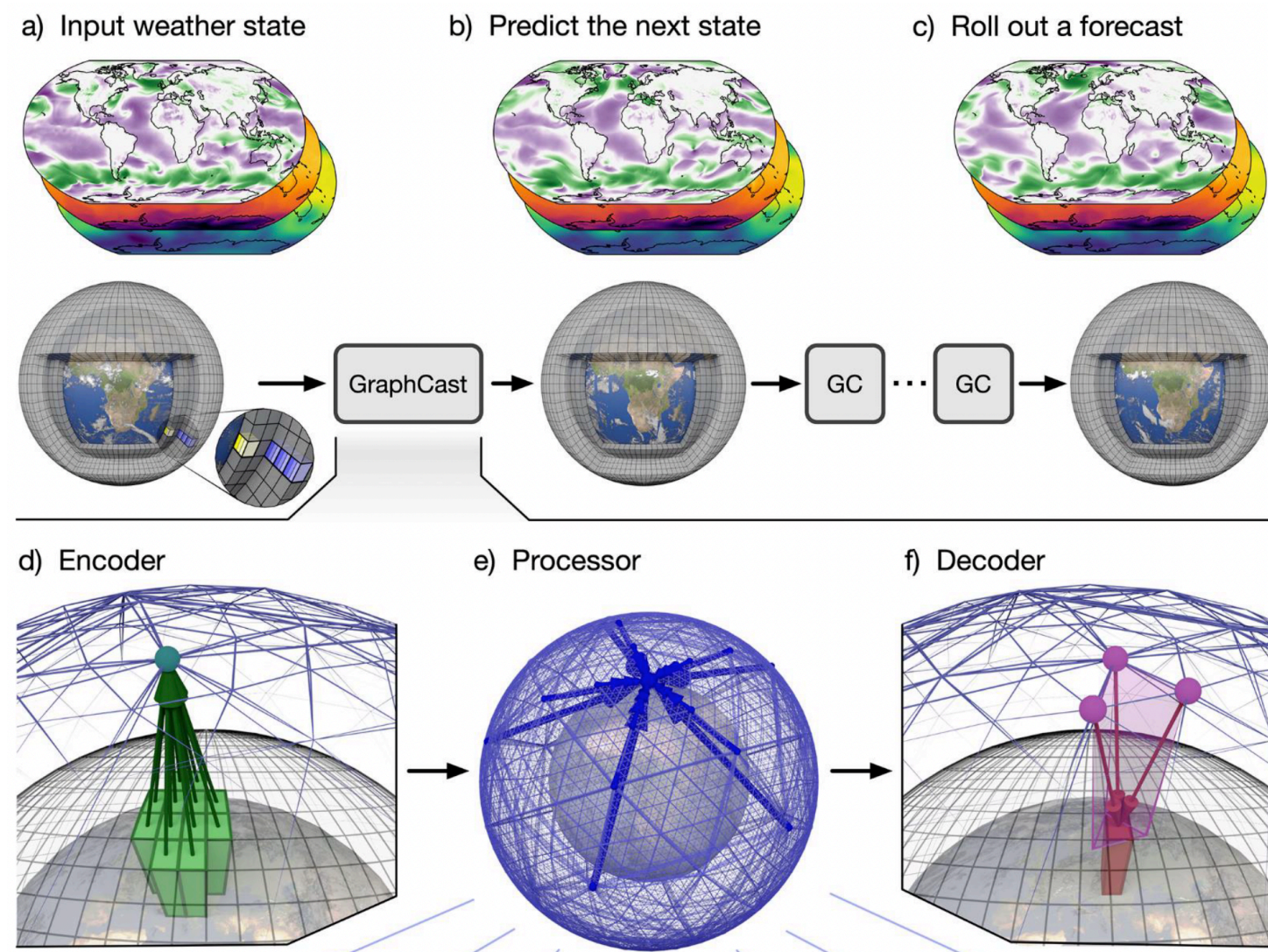
Application I: Traffic prediction



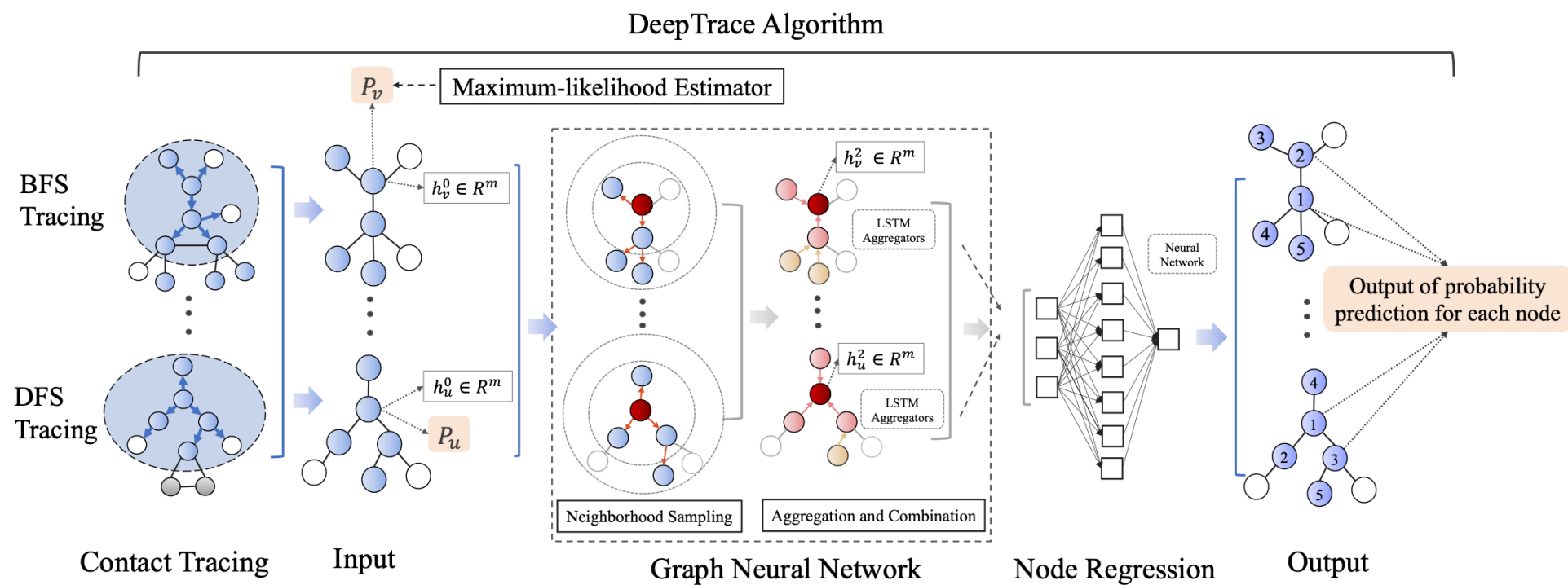
Application I: Traffic prediction



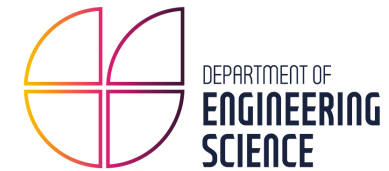
Application II: Weather forecasting



Application III: Contact tracing



Graph machine learning - Summary

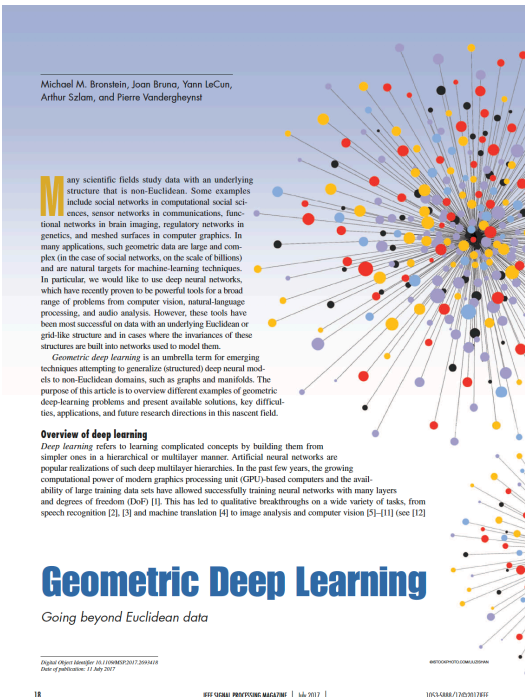


- Fast-growing field that extends data analysis to non-Euclidean domain
- Highly interdisciplinary: machine learning, signal processing, harmonic analysis, applies statistics, differential geometry
- Promising directions
 - going beyond convolutional models (e.g., graph transformers)
 - expressive power of graph ML models
 - robustness & generalisation & scalability
 - interpretability & causal inference
 - construction/refinement of initial graphs
 - applications (particularly in urban science)

References



DEPARTMENT OF
**ENGINEERING
SCIENCE**



Representation Learning on Graphs: Methods and Applications

William L. Hamilton Rex Ying Jure Leskovec
wleif@stanford.edu rexying@stanford.edu jure@cs.stanford.edu

Department of Computer Science
Stanford University
Stanford, CA, 94305

Abstract

Machine learning on graphs is an important and ubiquitous task with applications ranging from drug design to friendship recommendation in social networks. The primary challenge in this domain is finding a way to represent, or encode, graph structure so that it can be easily exploited by machine learning models. Traditionally, machine learning approaches relied on user-defined heuristics to extract features encoding structural information about a graph (e.g., degree statistics or kernel functions). However, recent years have seen a surge in approaches that automatically learn to encode graph structure into low-dimensional embeddings, using techniques based on deep learning and nonlinear dimensionality reduction. Here we provide a conceptual review of key advancements in this area of representation learning on graphs, including matrix factorization-based methods, random-walk based algorithms, and graph convolutional networks. We review methods to embed individual nodes as well as approaches to embed entire (sub)graphs. In doing so, we develop a unified framework to describe these recent approaches, and we highlight a number of important applications and directions for future work.

1 Introduction

Graphs are a ubiquitous data structure, employed extensively within computer science and related fields. Social networks, molecular graph structures, biological protein-protein networks, recommender systems—all of these domains and many more can be readily modeled as graphs, which capture interactions (*i.e.*, edges) between individual units (*i.e.*, nodes). As a consequence of their ubiquity, graphs are the backbone of countless systems, allowing relational knowledge about interacting entities to be efficiently stored and accessed [2].

However, graphs are not only useful as structured knowledge repositories: they also play a key role in modern machine learning. Machine learning applications seek to make predictions, or discover new patterns, using graph-structured data as feature information. For example, one might wish to classify the role of a protein in a biological interaction graph [28], predict the role of a person in a collaboration network, recommend new friends to a user in a social network [3], or predict new therapeutic applications of existing drug molecules, whose structure can be represented as a graph [21].

Copyright 2017 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

4 IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 32, NO. 1, JANUARY 2021

A Comprehensive Survey on Graph Neural Networks

Zonghan Wu[✉], Shirui Pan[✉], Member, IEEE, Fengwen Chen, Guodong Long[✉],
Chengqi Zhang[✉], Senior Member, IEEE, and Philip S. Yu, Life Fellow, IEEE

Abstract—Deep learning has revolutionized many machine learning tasks in recent years, ranging from image classification and video processing to speech recognition and natural language understanding. The data in these tasks are typically represented in the Euclidean space. However, there is an increasing number of applications, where data are generated from non-Euclidean domains and are represented as graphs with complex relationships and interdependency between objects. The complexity of graph data has imposed significant challenges on the existing machine learning algorithms. Recently, many studies on extending deep learning approaches for graph data have emerged. In this article, we provide a comprehensive overview of graph neural networks (GNNs) in data mining and machine learning fields. We propose a new taxonomy to divide the state-of-the-art GNNs into four categories, namely, recurrent GNNs, convolutional GNNs, graph autoencoders, and spatial-temporal GNNs. We further discuss the applications of GNNs across various domains and summarize the open-source codes, benchmark data sets, and model evaluation of GNNs. Finally, we propose potential research directions in this rapidly growing field.

Index Terms—Deep learning, graph autoencoder (GAE), graph convolutional networks (GCNs), graph neural networks (GNNs), graph representation learning, network embedding.

I. INTRODUCTION

THE recent success of neural networks has boosted research on pattern recognition and data mining. Many machine learning tasks, such as object detection [1], [2], machine translation [3], [4], and speech recognition [5], which once heavily relied on handcrafted feature engineering to extract informative feature sets, have recently been revolutionized by various end-to-end deep learning paradigms, e.g., convolutional neural networks (CNNs) [6], recurrent neural

networks (RNNs) [7], and autoencoders [8]. The success of deep learning in many domains is partially attributed to the rapidly developing computational resources (e.g., GPU), the availability of big training data, and the effectiveness of deep learning to extract latent representations from the Euclidean data (e.g., images, text, and videos). Taking image data as an example, we can represent an image as a regular grid in the Euclidean space. CNN is able to exploit the shift-invariance, local connectivity, and compositionality of image data [9]. As a result, CNNs can extract local meaningful features that are shared with the entire data sets for various image analyses.

While deep learning effectively captures hidden patterns of Euclidean data, there are an increasing number of applications, where data are represented in the form of graphs. For example, in e-commerce, a graph-based learning system can exploit the interactions between users and products to make highly accurate recommendations. In chemistry, molecules are modeled as graphs, and their bioactivity needs to be identified for drug discovery. In a citation network, articles are linked to each other via citations, and they need to be categorized into different groups. The complexity of graph data has imposed significant challenges on the existing machine learning algorithms. As graphs can be irregular, a graph may have a variable size of unordered nodes, and nodes from a graph may have a different number of neighbors, resulting in some important operations (e.g., convolutions) being easy to compute in the image domain but difficult to apply to the graph domain. Furthermore, a core assumption of existing machine learning algorithms is that instances are independent of each other. This assumption no longer holds for graph data because each instance (node) is related to others by links of various types, such as citations, friendships, and interactions.

Recently, there is increasing interest in extending deep learning approaches for graph data. Motivated by CNNs, RNNs, and autoencoders from deep learning, new generalizations and definitions of important operations have been rapidly developed over the past few years to handle the complexity of graph data. For example, a graph convolution can be generalized from a 2-D convolution. As illustrated in Fig. 1, an image can be considered as a special case of graphs, where pixels are connected by adjacent pixels. Similar to 2-D convolution, one may perform graph convolutions by taking the weighted average of a node's neighborhood information.

There are a limited number of existing reviews on the topic of graph neural networks (GNNs). Using the term geometric deep learning, Bronstein *et al.* [9] give an overview

Journal of Machine Learning Research 23 (2022) 1–64

Submitted 8/20; Revised 10/21; Published 06/22

Machine Learning on Graphs: A Model and Comprehensive Taxonomy

Ines Chami[✉] CHAMI@STANFORD.EDU
Stanford University
Stanford, CA, 94305, USA
Sami Abu-El-Hajja[✉] SAMI@HAJJA.ORG
USC Information Sciences Institute
Marina Del Rey, CA, 90292, USA
Bryan Perozzi[✉] BPEROZZI@ACM.ORG
Google Research
New York, NY, 10011, USA
Christopher Ré[✉] CHRISRE@CS.STANFORD.EDU
Stanford University
Stanford, CA, 94305, USA
Kevin Murphy[✉] KPMURPHY@GOOGLE.COM
Google Research
Mountain View, CA, 94043, USA

Editor: Ruslan Salakhutdinov

Abstract

There has been a surge of recent interest in graph representation learning (GRL). GRL methods have generally fallen into three main categories, based on the availability of labeled data. The first, network embedding, focuses on learning unsupervised representations of relational structure. The second, graph regularized neural networks, leverages graphs to augment neural network losses with a regularization objective for semi-supervised learning. The third, graph neural networks, aims to learn differentiable functions over discrete topologies with arbitrary structure. However, despite the popularity of these areas there has been surprisingly little work on unifying the three paradigms. Here, we aim to bridge the gap between network embedding, graph regularization and graph neural networks. We propose a comprehensive taxonomy of GRL methods, aiming to unify several disparate bodies of work. Specifically, we propose the GRAPHEDM framework, which generalizes popular algorithms for semi-supervised learning (e.g. GraphSage, GCN, GAT), and unsupervised learning (e.g. DeepWalk, node2vec) of graph representations into a single consistent approach. To illustrate the generality of GRAPHEDM, we fit over thirty existing methods into this framework. We believe that this unifying view both provides a solid foundation for understanding the intuition behind these methods, and enables future research in the area.

Keywords: Network Embedding, Graph Neural Networks, Geometric Deep Learning, Manifold Learning, Relational Learning

* Work partially done during an internship at Google Research.

©2022 Ines Chami, Sami Abu-El-Hajja, Bryan Perozzi, Christopher Ré and Kevin Murphy.

License: CC-BY 4.0, see <https://creativecommons.org/licenses/by/4.0/>. Attribution requirements are provided at <http://jmlr.org/papers/v22/20-882.html>.