A Short Introduction to Graph Machine Learning

Xiaowen Dong
Department of Engineering Science
University of Oxford

AI for Health Summer School, EPFL, July 2025

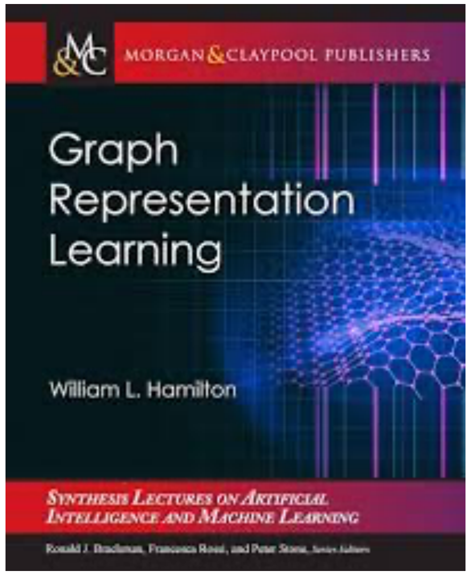
UNIVERSITY OF OXFORD

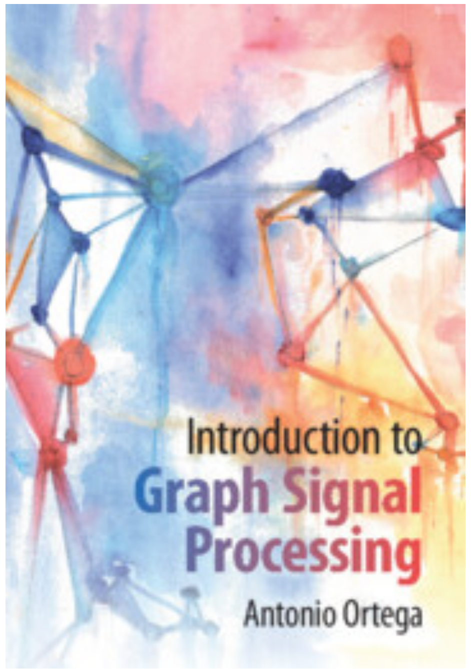
Graph Machine Learning

- Overview of graph machine learning
- Convolutional neural networks on graphs
  - "spectral" approaches enabled by graph signal processing
- State-of-the-art graph neural networks
  - "spatial" approaches enabled by message passing
- Latest developments and applications

Resources

- Textbooks
  - Hamilton, "Graph Representation Learning," Morgan & Claypool Publishers, 2020. Available at https://www.cs.mcgill.ca/~wlh/grl_book/
  - Ortega, "Introduction to Graph Signal Processing," Cambridge University Press, 2022.
- Papers
  - https://web.media.mit.edu/~xdong/resource.html
- Software packages
  - https://pygsp.readthedocs.io/en/stable/
  - https://scikit-network.readthedocs.io/en/latest/
  - https://pytorch-geometric.readthedocs.io/en/latest/



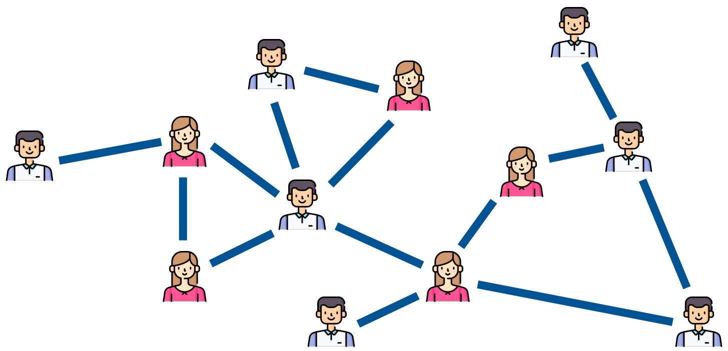


Graph Machine Learning

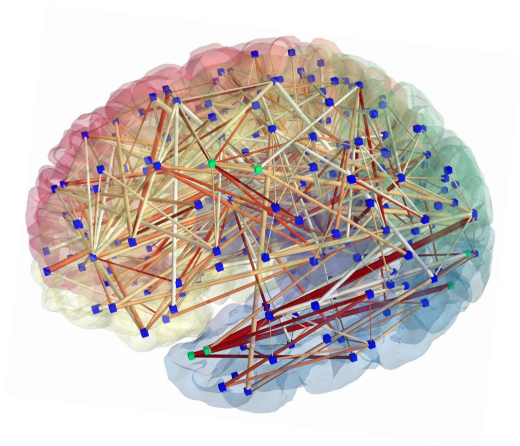
- Overview of graph machine learning
- Convolutional neural networks on graphs
  - "spectral" approaches enabled by graph signal processing
- State-of-the-art graph neural networks
  - "spatial" approaches enabled by message passing
- Latest developments and applications

Networks are pervasive

traffic network



social network

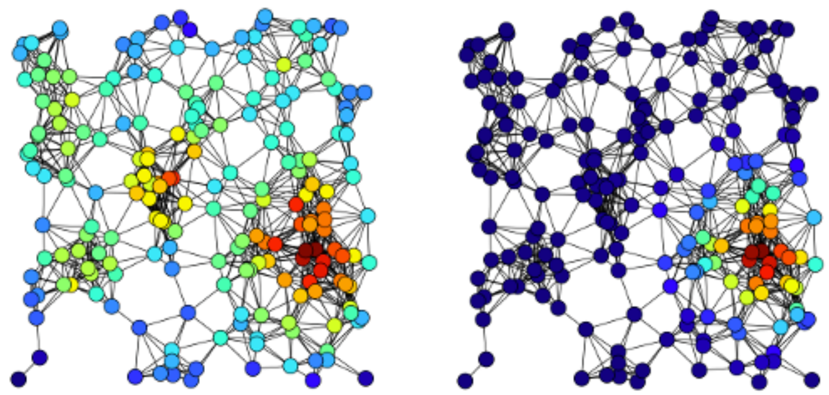


brain network

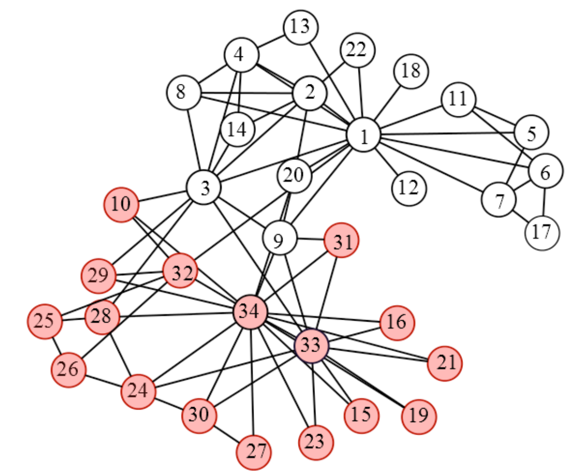
networks are mathematically represented by graphs

The field of network science

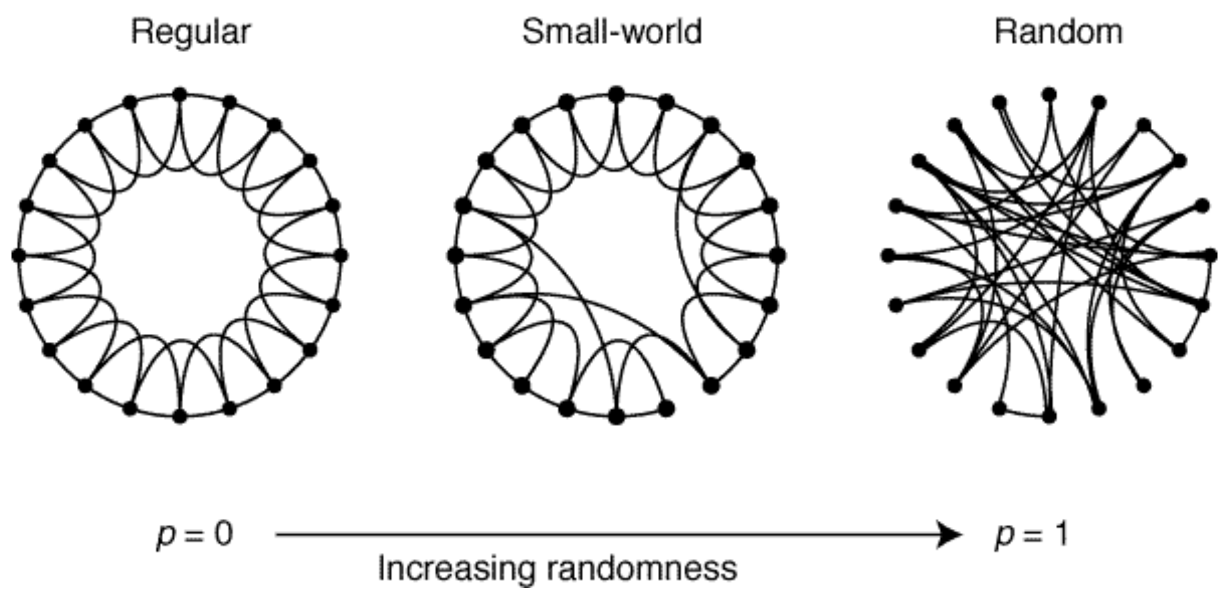
node centrality



graph clustering



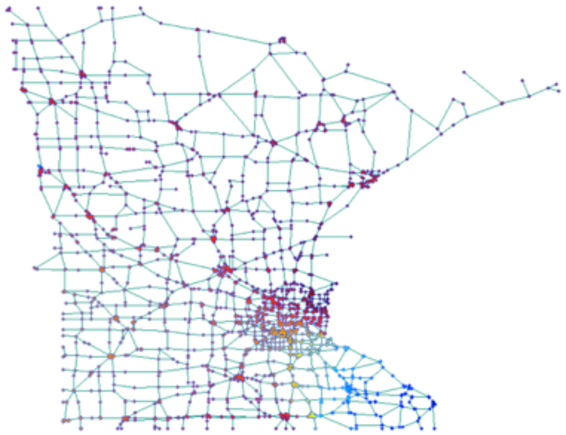
random graph models



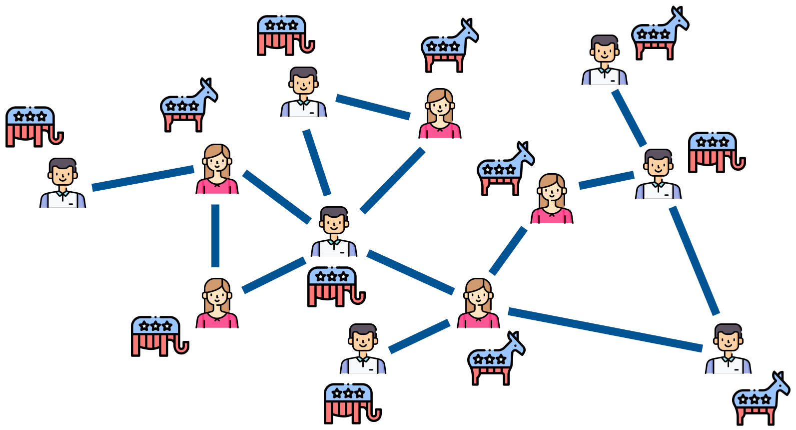
primarily focused on graphs (edge relations) but not node attributes

Watts and Strogatz, "Collective dynamics of 'small-world' networks," Nature, 1998.
Newman, "Networks: An introduction," Oxford University Press, 2010.

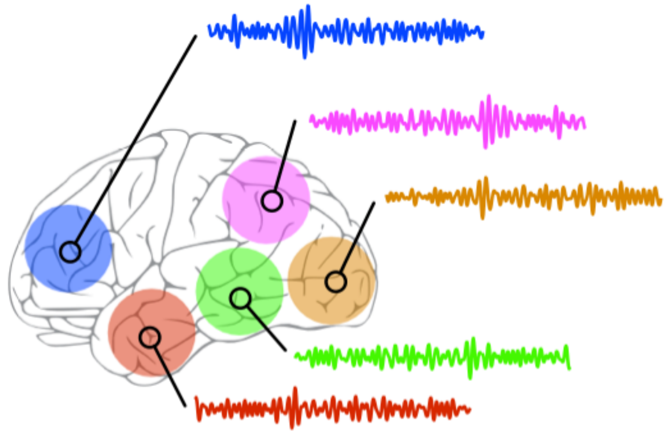
Graph-structured data are pervasive



congestion in road junctions



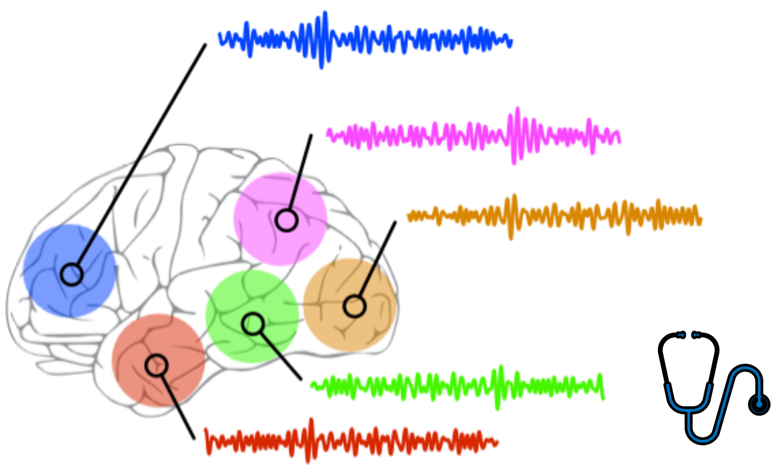
preferences of individuals



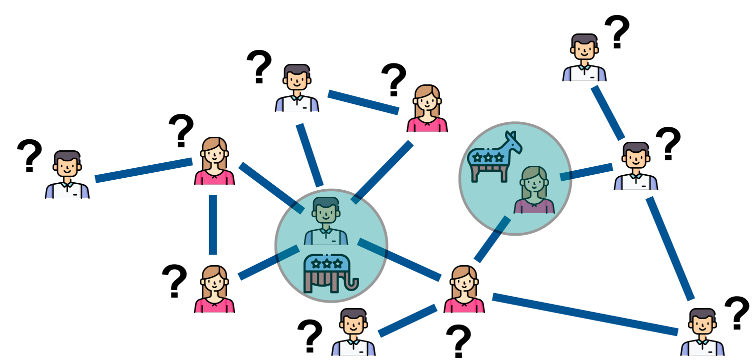
activities in brain regions

from graphs to graph-structured data

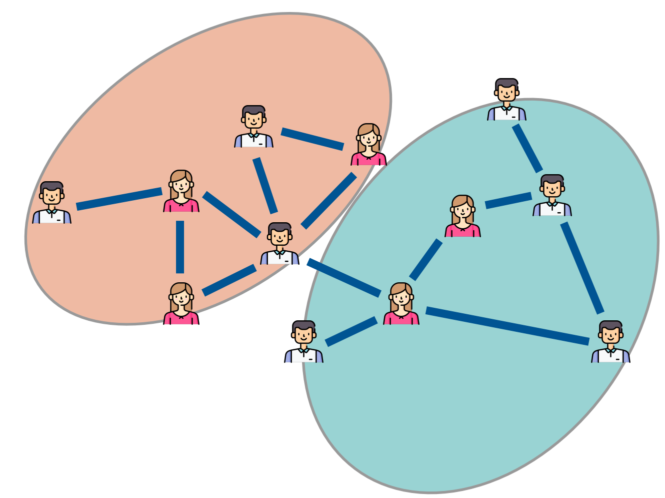
Learning with graph-structured data



graph-level classification (supervised)



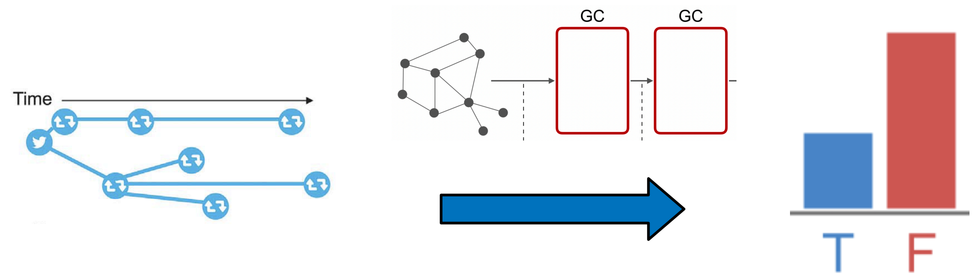
node-level classification (semi-supervised)

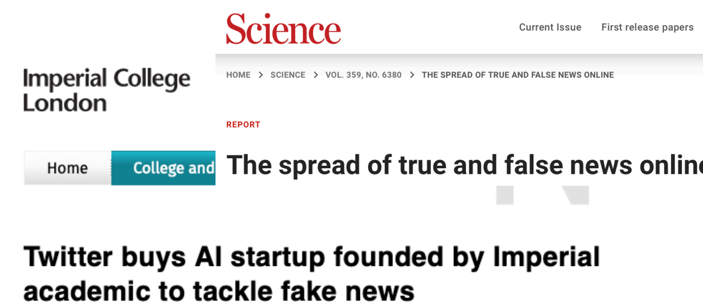


graph clustering (unsupervised)

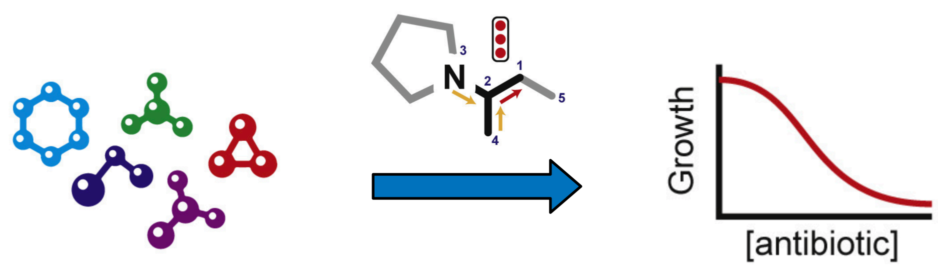
Learning with graph-structured data

fake news detection

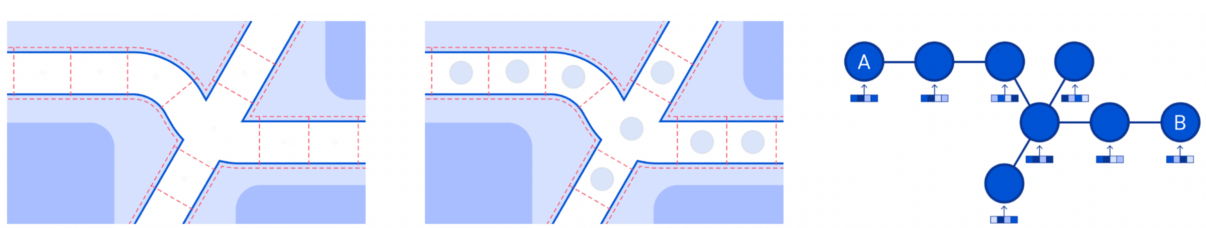


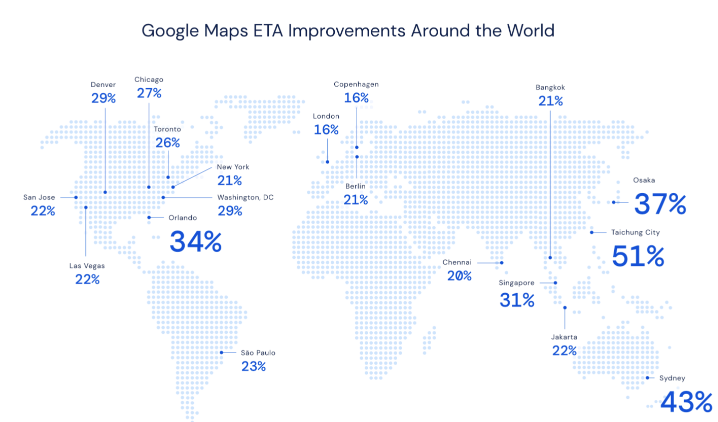


drug discovery



traffic prediction



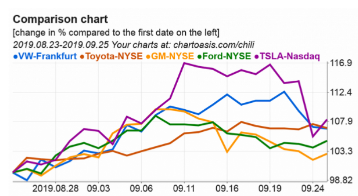


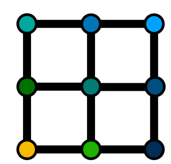
Monti et al., “Fake news detection on social media using geometric deep learning,” ICLR Workshop, 2019.
Stokes et al., “A deep learning approach to antibiotic discovery,” Cell, 2020.
Derrow-Pinion et al., “ETA prediction with graph neural networks in Google Maps,” CIKM, 2021.

Classical ML vs Graph ML

Classical ML

regular domain
(real line, 2D grid)





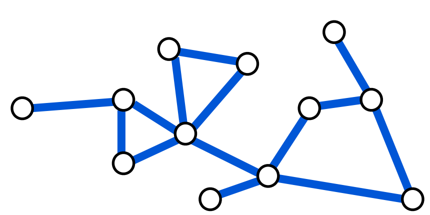
$f(X)$

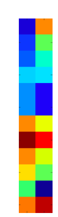
time series
forecasting

image
classification/
segmentation

Graph ML

irregular domain
(graphs)





$\mathcal{G}$ X

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

node classification

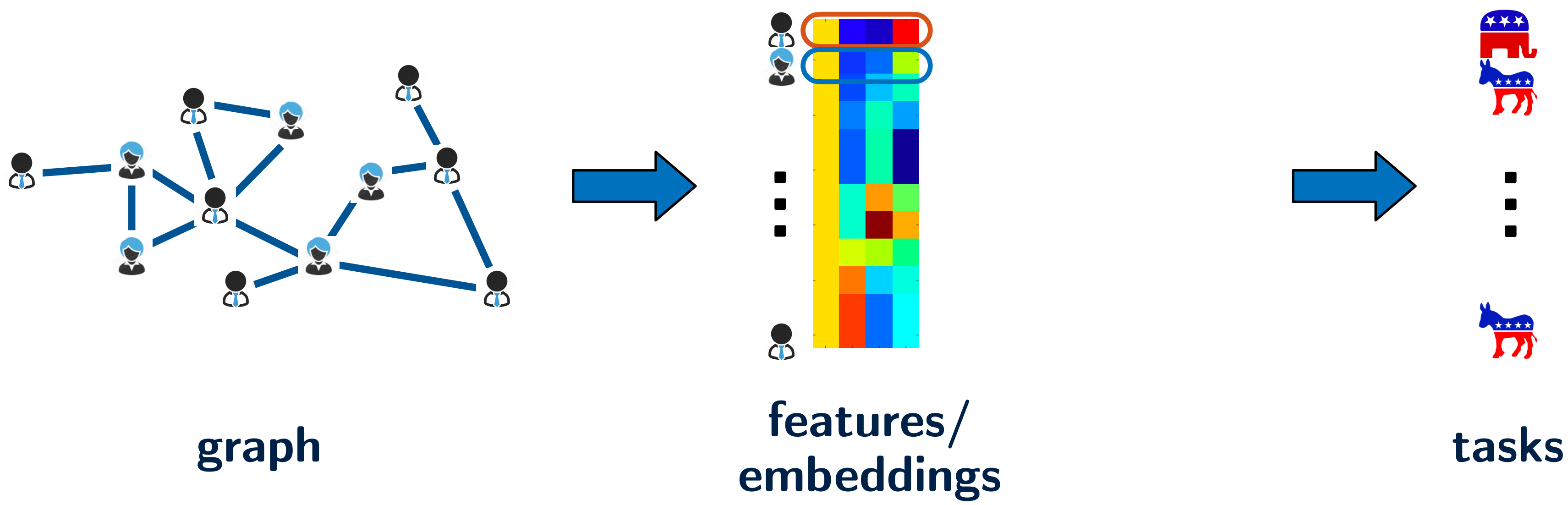
link prediction

graph classification

graph clustering

How to incorporate graphs into learning?

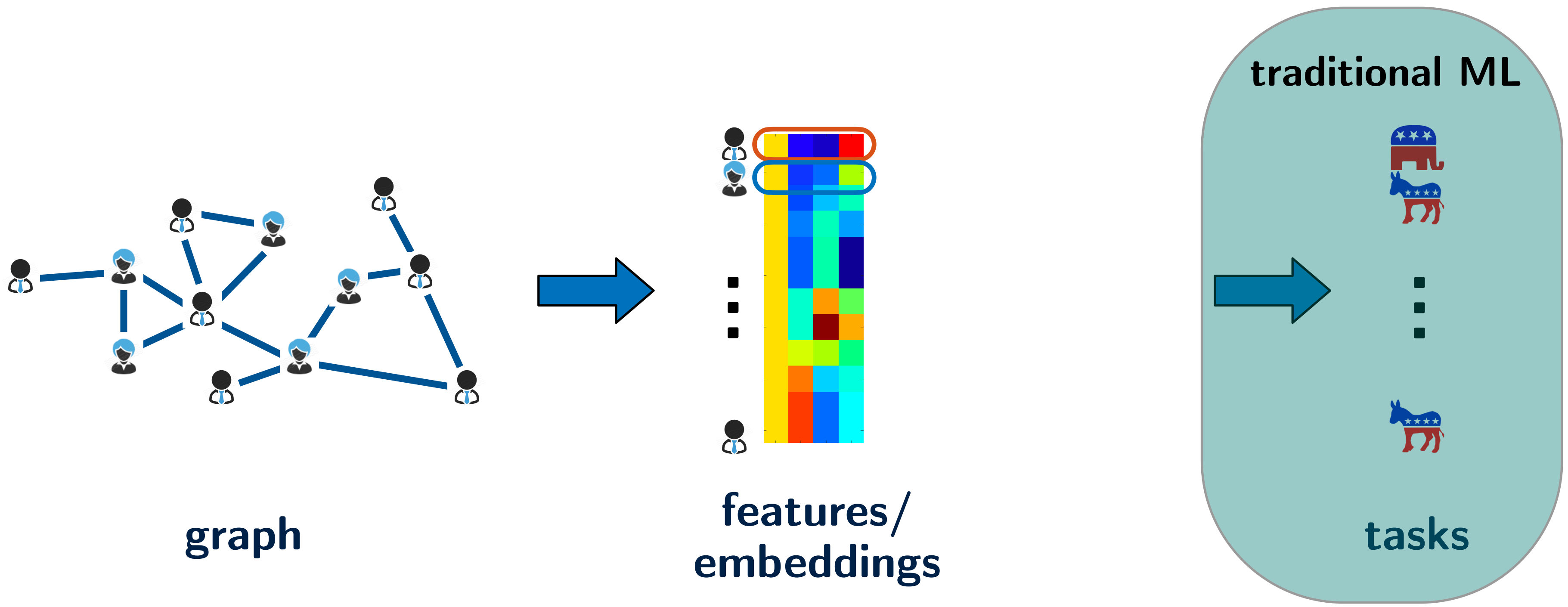
- Traditional machine learning on graphs



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

How to incorporate graphs into learning?

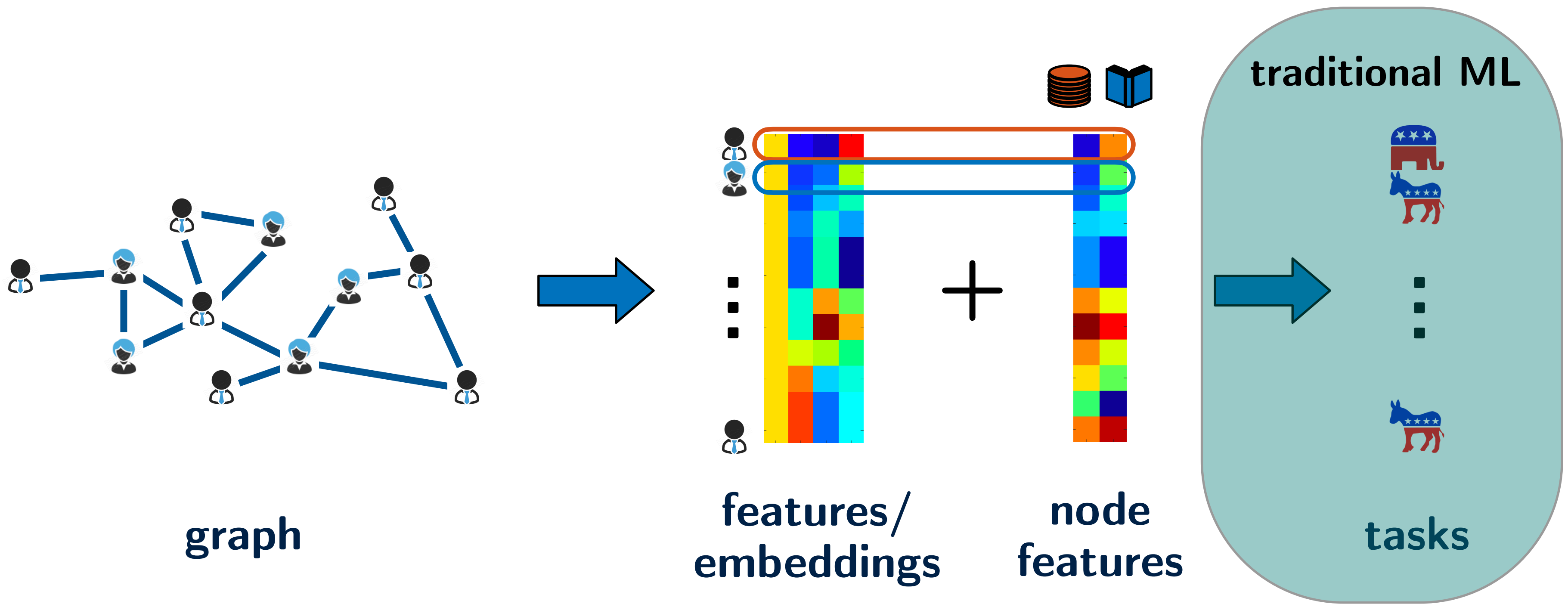
- Traditional machine learning on graphs



- 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?

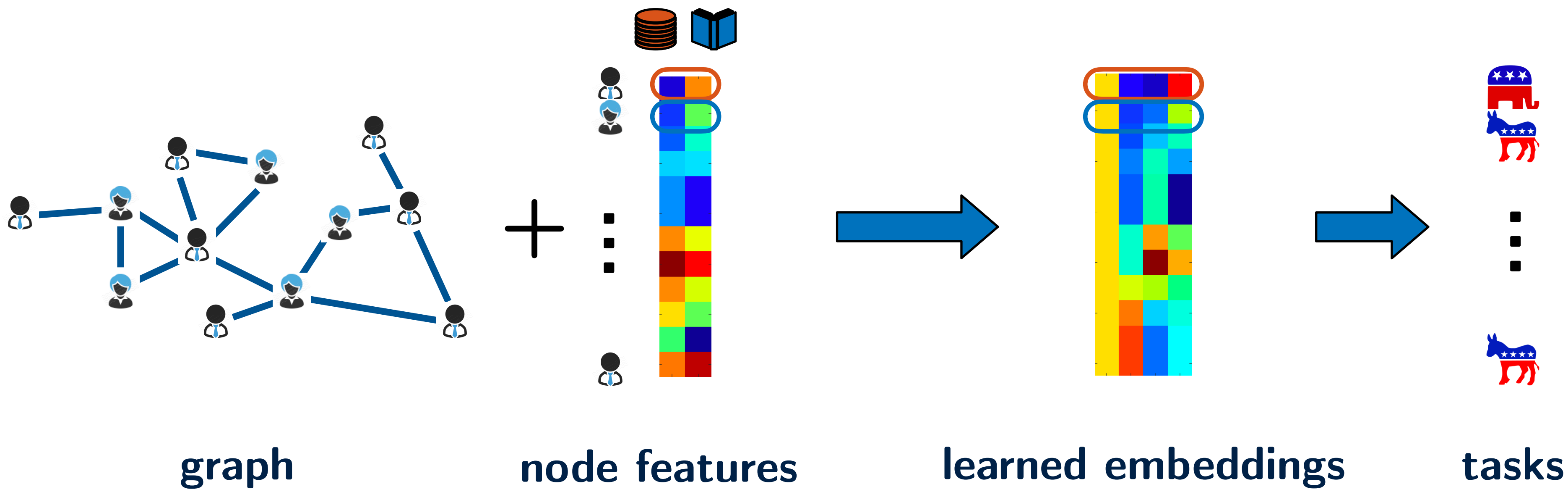
- Traditional machine learning on graphs



- 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?

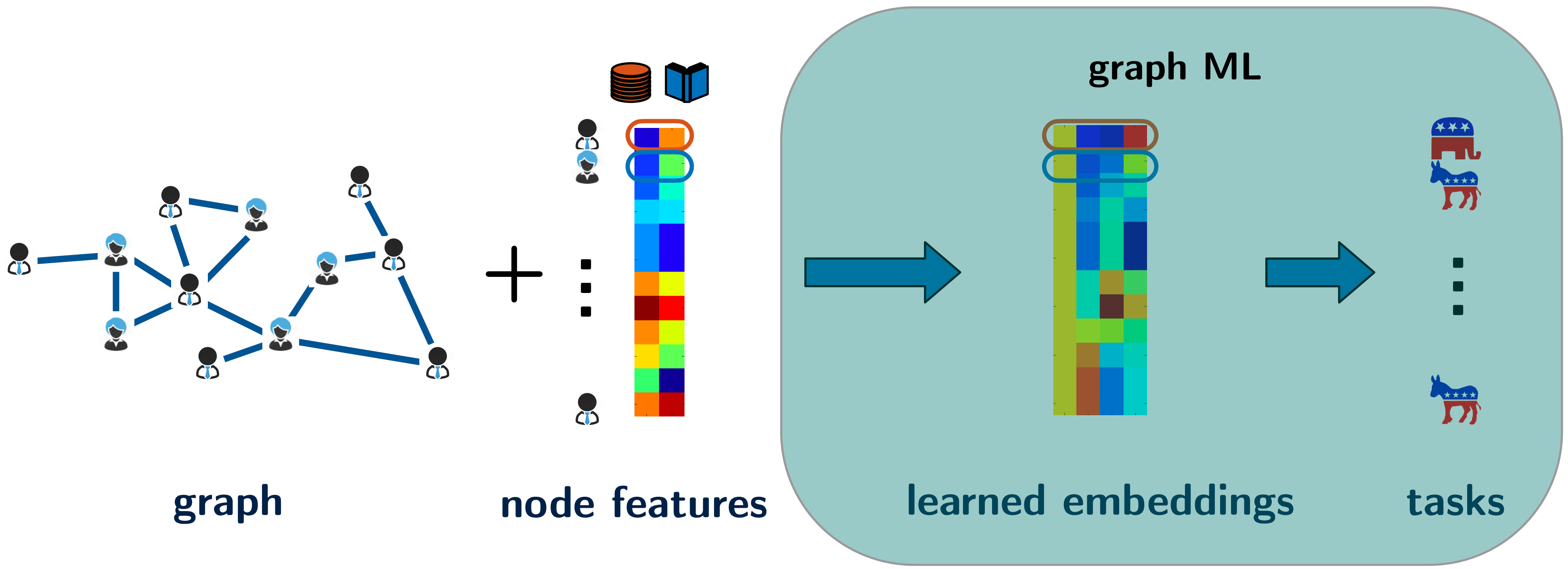
- Graph machine learning



- 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?

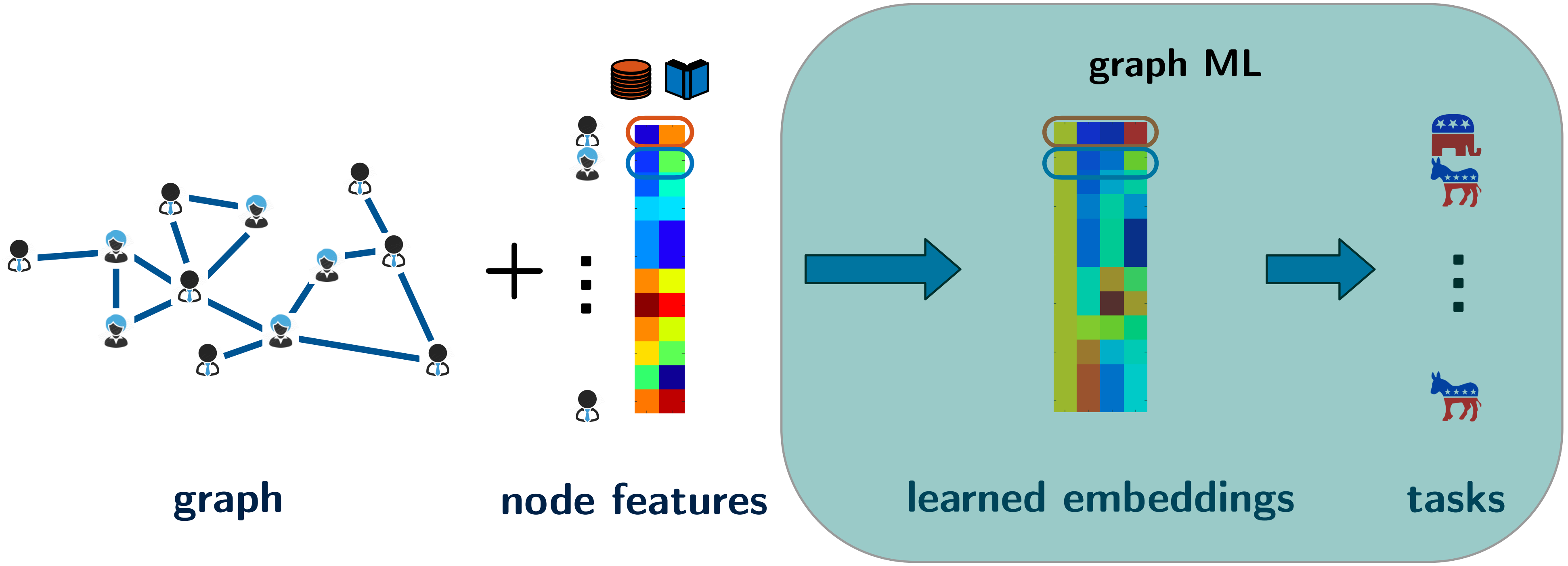
- Graph machine learning



- 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

How to incorporate graphs into learning?

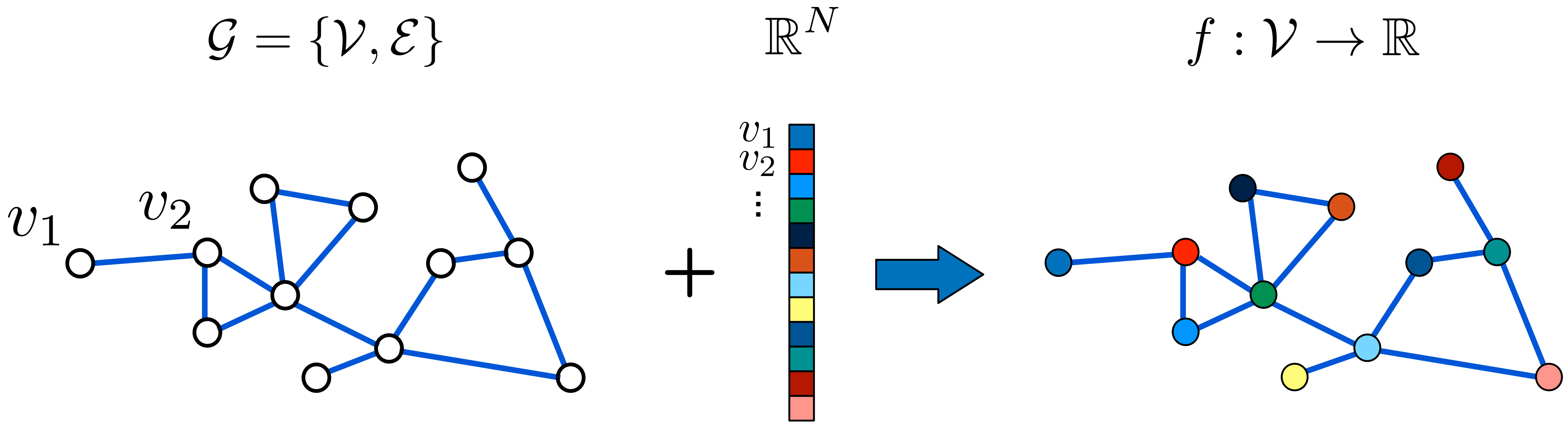
- Graph machine learning



- 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

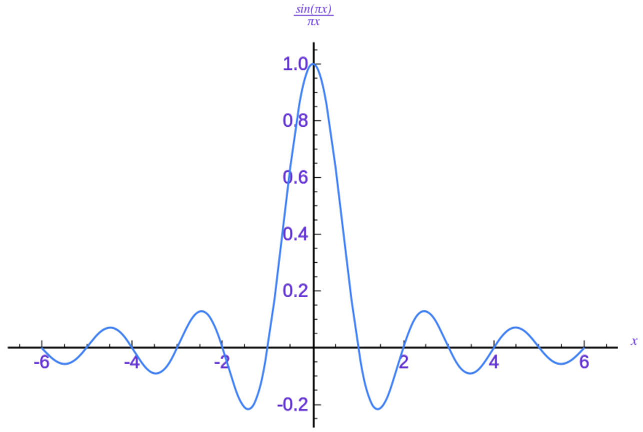
- Graph-structured data can be represented by graph signals



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

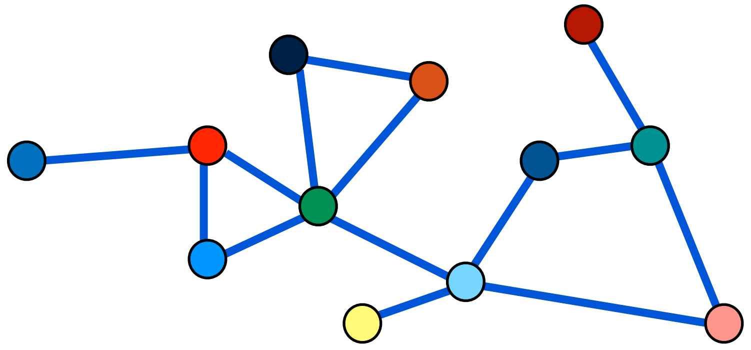
Graph signal processing

1D signal



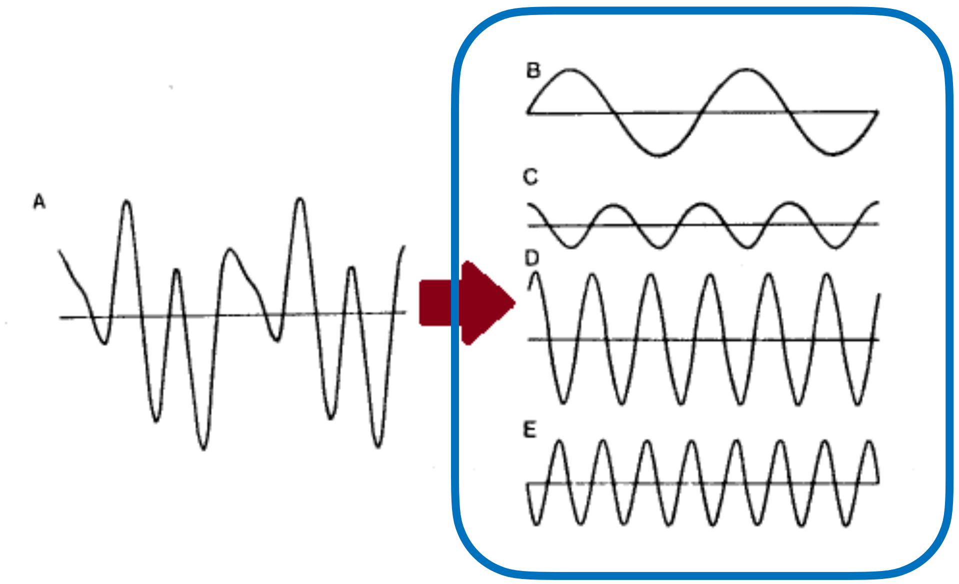
2D signal

graph signal



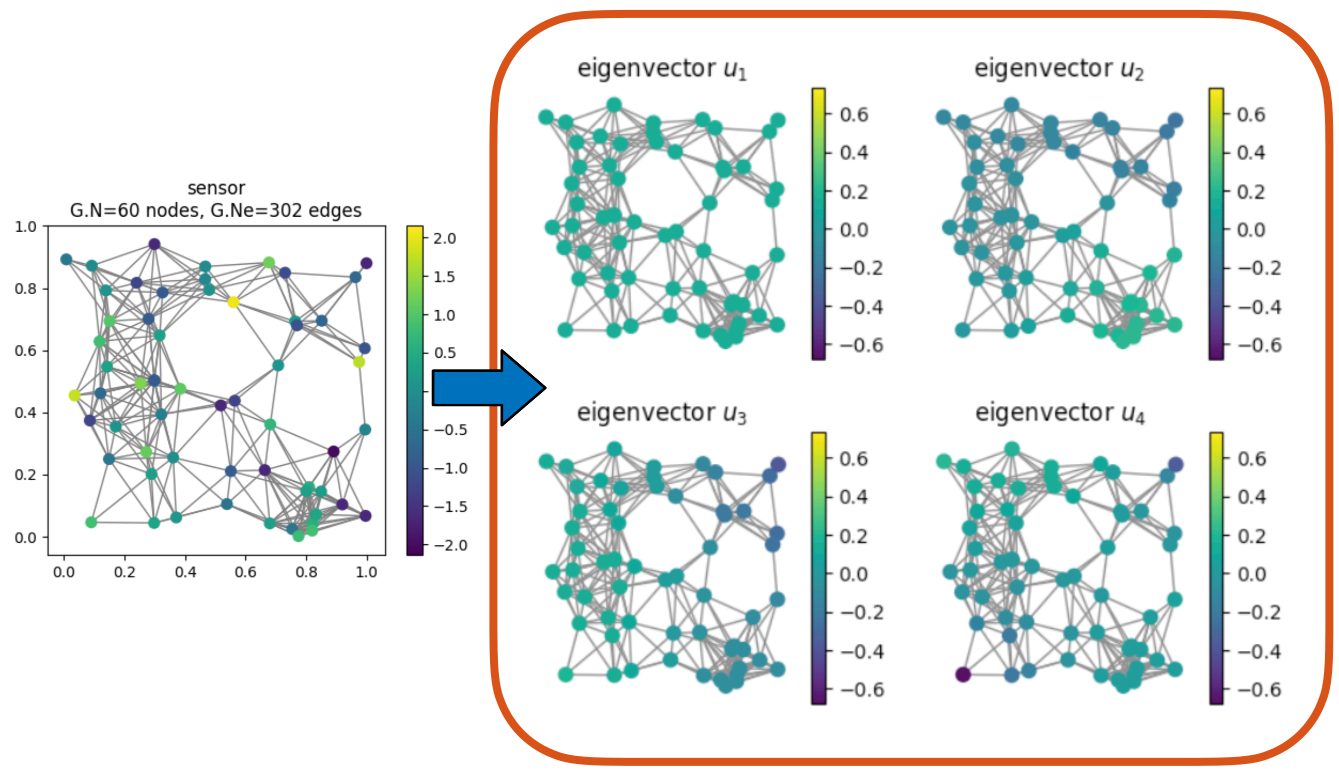
how to generalise classical signal processing tools on irregular domains such as graphs?

Graph signal processing



classical signal processing

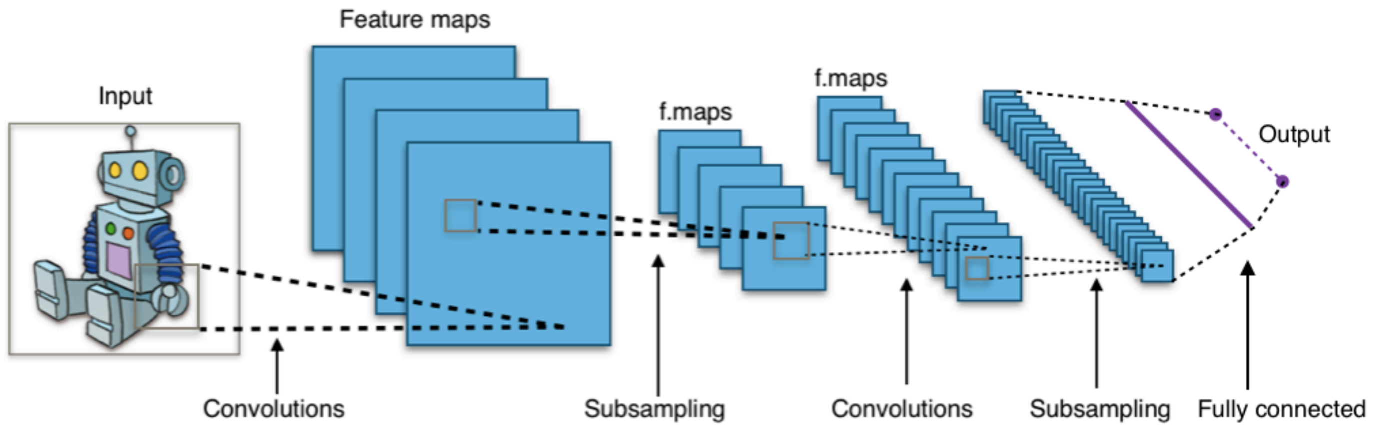
- complex exponentials provide "building blocks" of 1D signal (different oscillations or frequencies)
- leads to **Fourier transform**
- enables **filtering** and **convolution** on regular grids

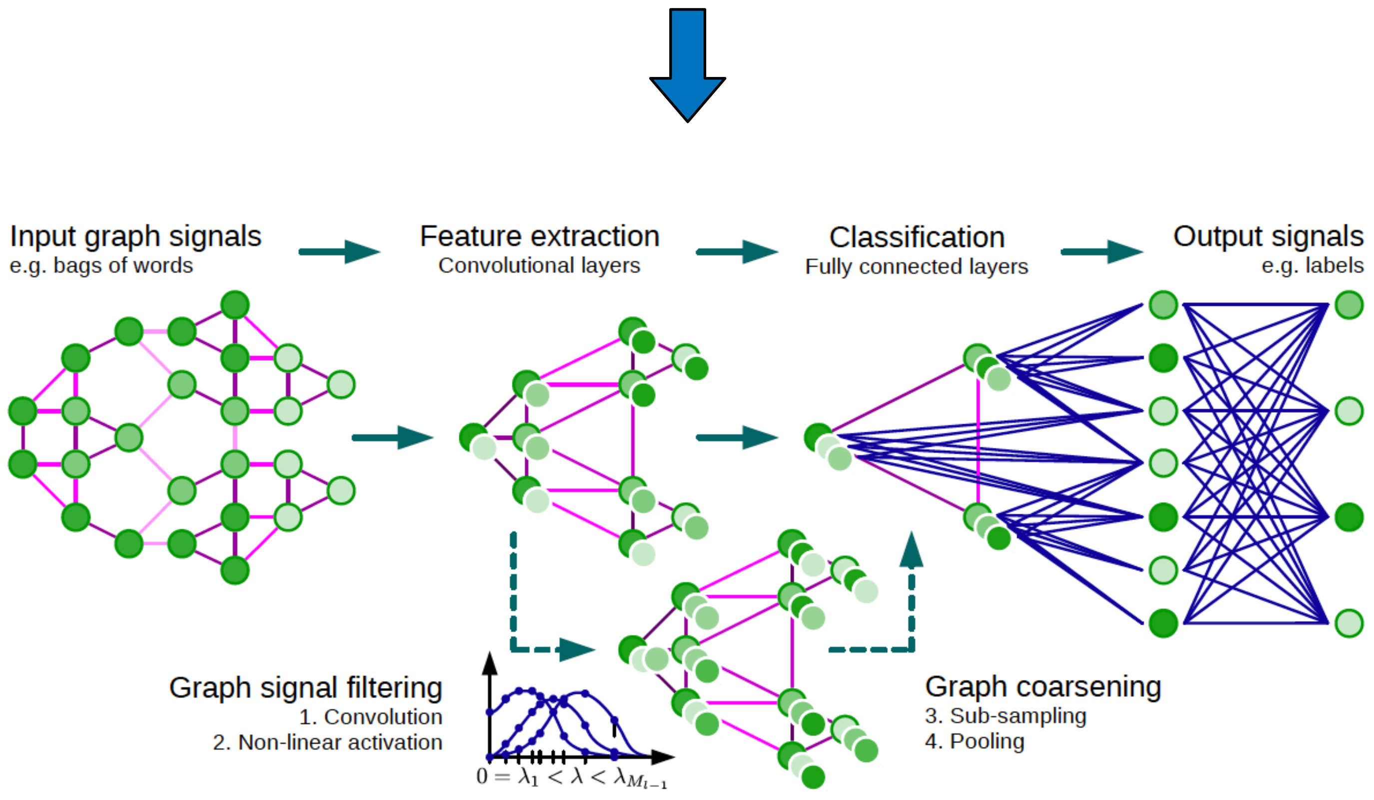


graph signal processing

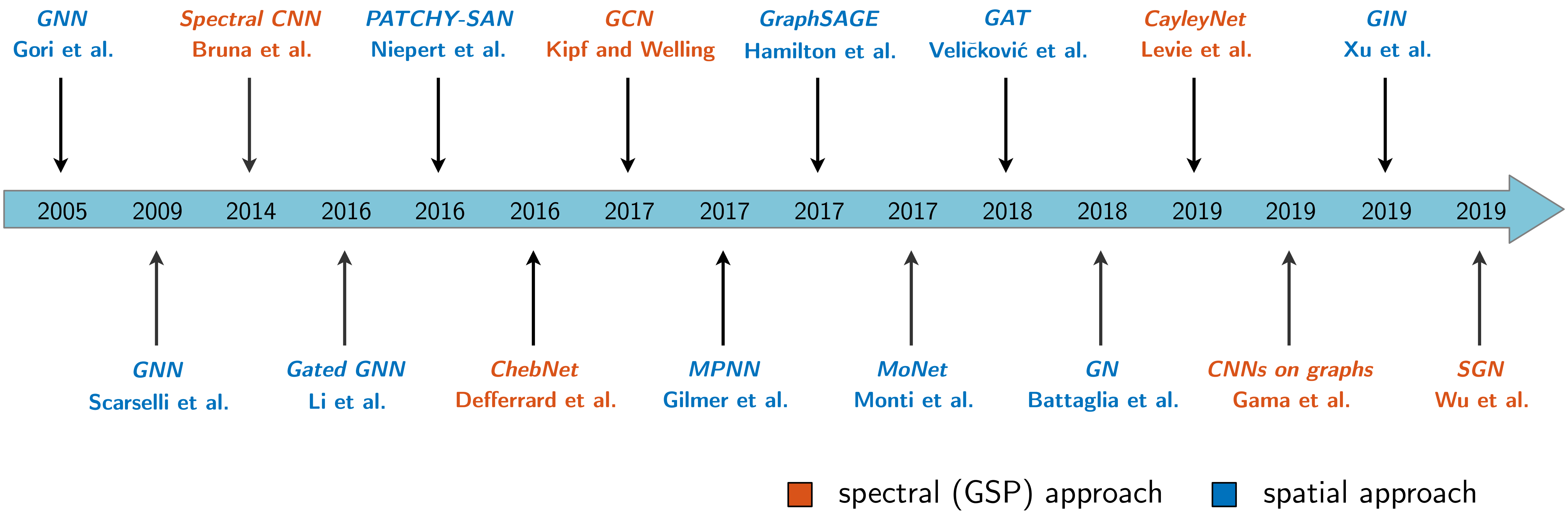
- Laplacian eigenvectors provide "building blocks" of graph signal (different oscillation or frequencies)
- leads to **graph Fourier transform**
- enables **filtering** and **convolution** on graphs

Convolutional neural networks on graphs

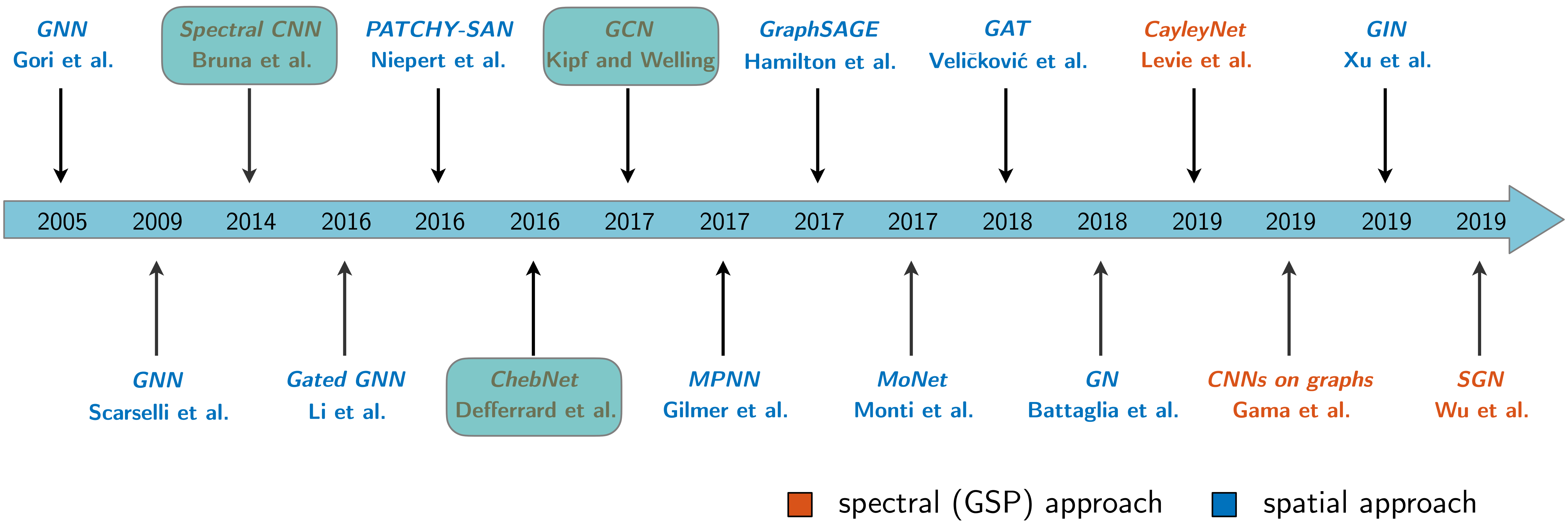




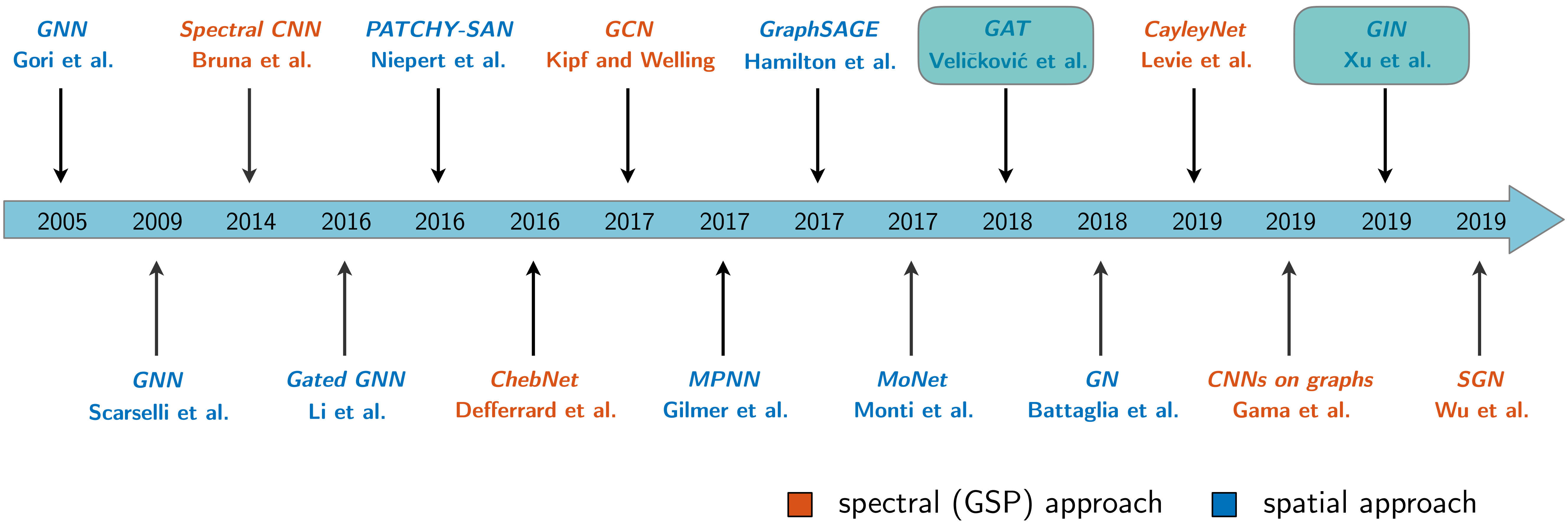
(More generally) Graph neural networks



(More generally) Graph neural networks



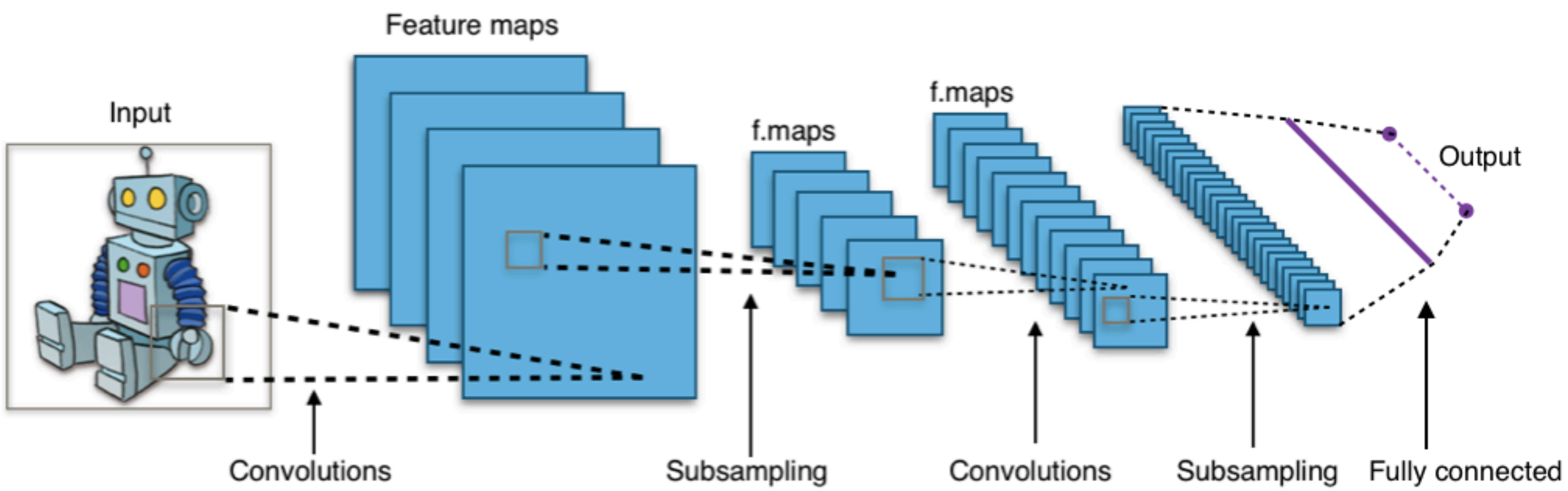
(More generally) Graph neural networks



Graph Machine Learning

- Overview of graph machine learning
- Convolutional neural networks on graphs
  - "spectral" approaches enabled by graph signal processing
- State-of-the-art graph neural networks
  - "spatial" approaches enabled by message passing
- Latest developments and applications

CNNs exploit structure within data

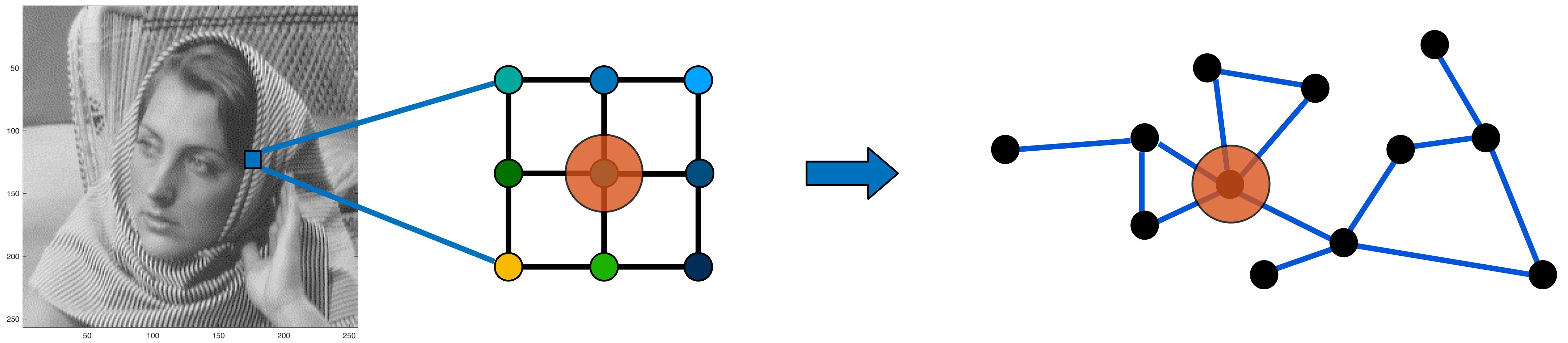


checklist

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

https://en.wikipedia.org/wiki/File:Typical_cnn.png

CNNs on graphs?



checklist

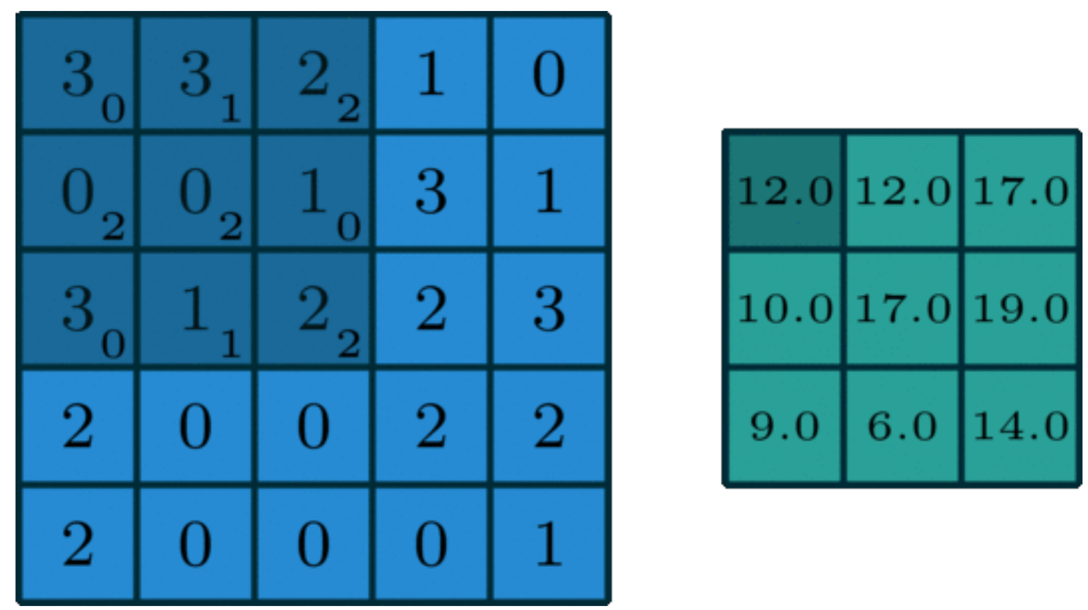
- **convolution:** how to define convolution? what about invariance?
- **localisation:** what is the notion of locality?
- **multi-scale:** how to down-sample on graphs?
- **efficiency:** how to keep computational complexity low?

Convolution on graphs

classical convolution

time domain

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



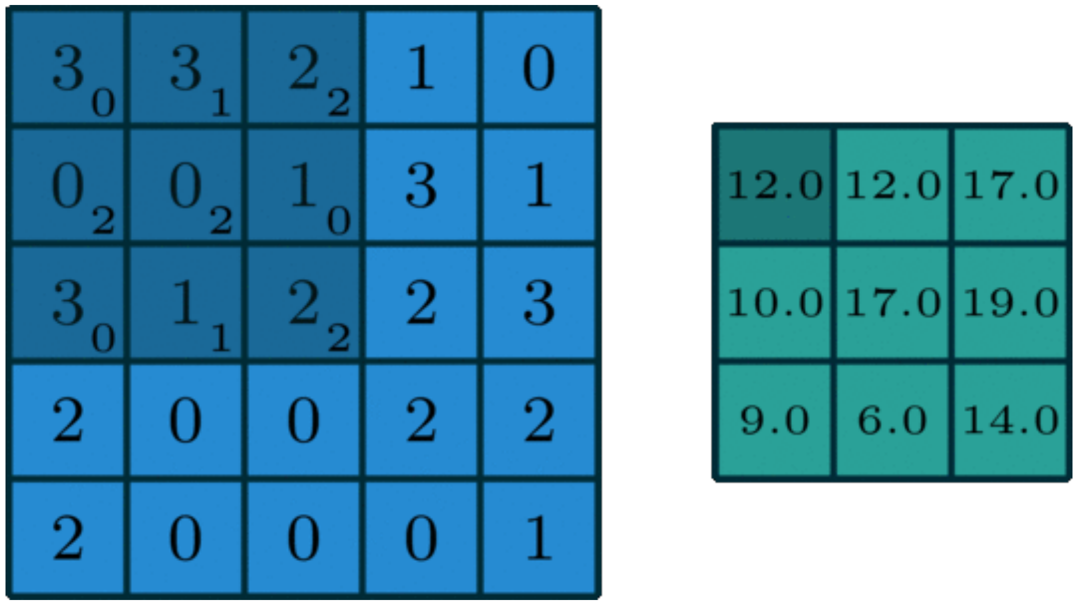
Dumoulin and Visin, "A guide to convolution arithmetic for deep learning," arXiv, 2018.

Convolution on graphs

classical convolution

time domain

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



Dumoulin and Visin, "A guide to convolution arithmetic for deep learning," arXiv, 2018.

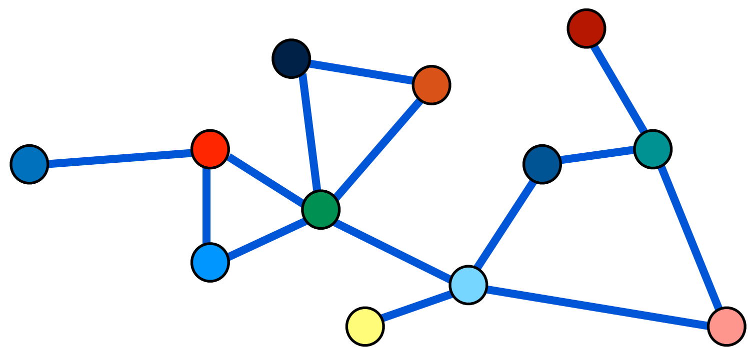
Convolution on graphs

classical convolution

time domain

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

convolution on graphs



Convolution on graphs

classical convolution

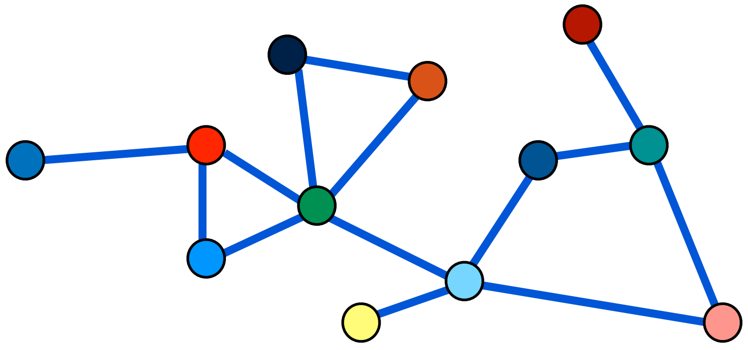
convolution on graphs

time domain

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

frequency domain

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



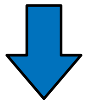
Dumoulin and Visin, "A guide to convolution arithmetic for deep learning," arXiv, 2018.

Convolution on graphs

classical convolution

time domain

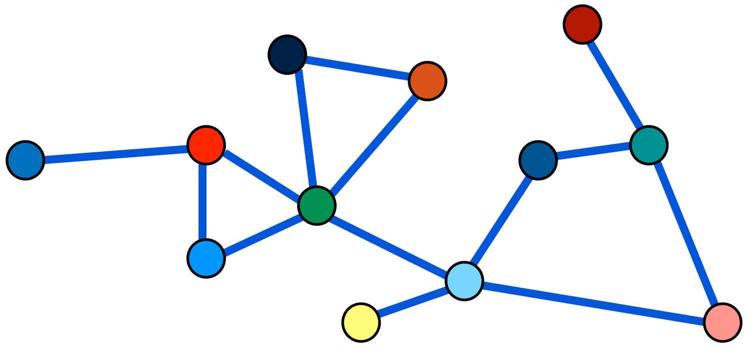
$$(f * g)(t) = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$$

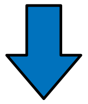


frequency domain

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

convolution on graphs

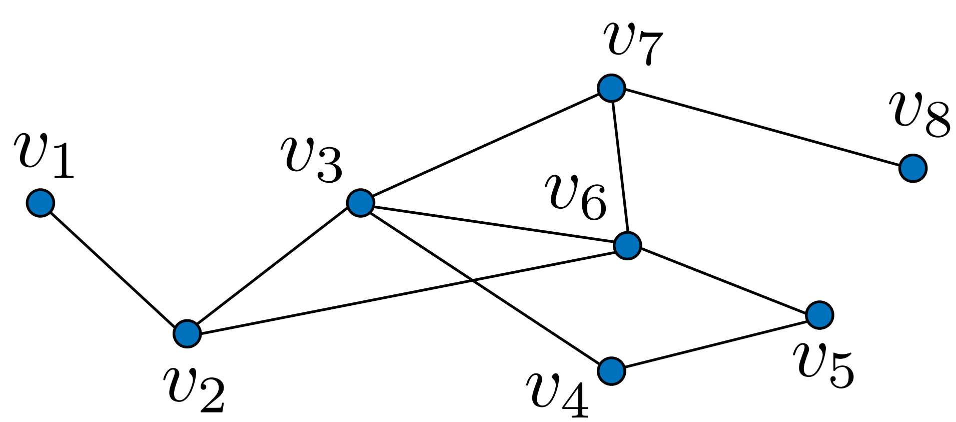




?

Dumoulin and Visin, "A guide to convolution arithmetic for deep learning," arXiv, 2018.

Graph Laplacian



weighted and undirected graph:

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

$$D = \mathrm{diag}(d(v_1), \cdots, d(v_N))$$

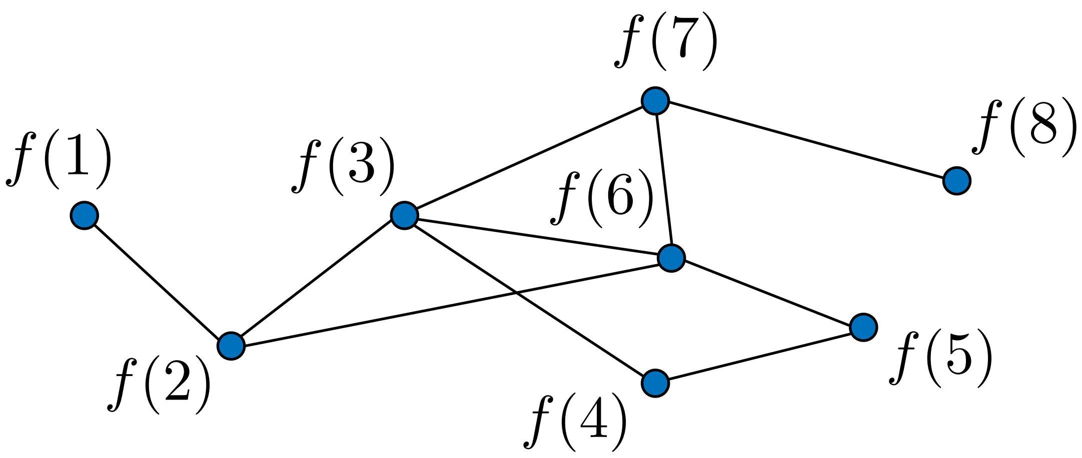
$$L = D - W$$ **equivalent to G!**

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

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{D} - \underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}}_{W} = \underbrace{\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}}_{L}$$

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

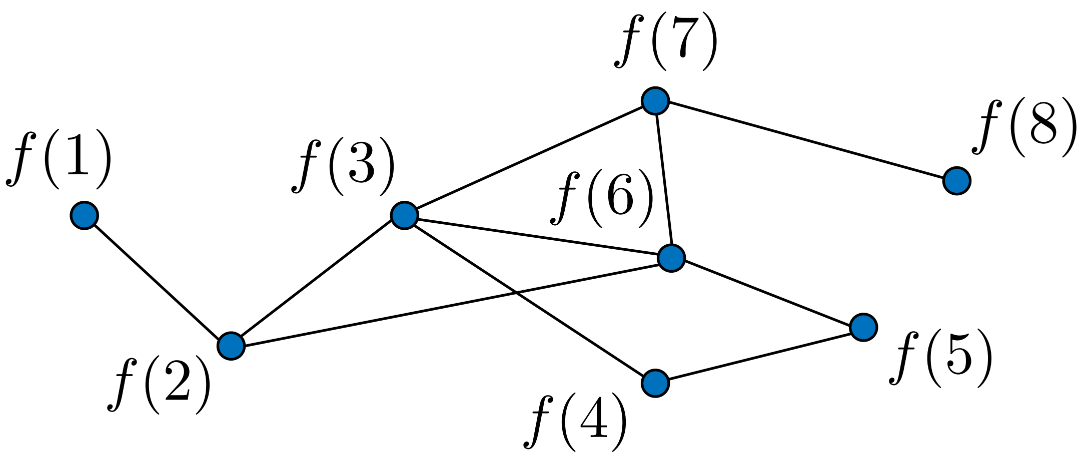
Graph Laplacian



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

Zhou and Schölkopf, "A regularization framework for learning from graph data," ICML Workshop, 2004.

Graph Laplacian



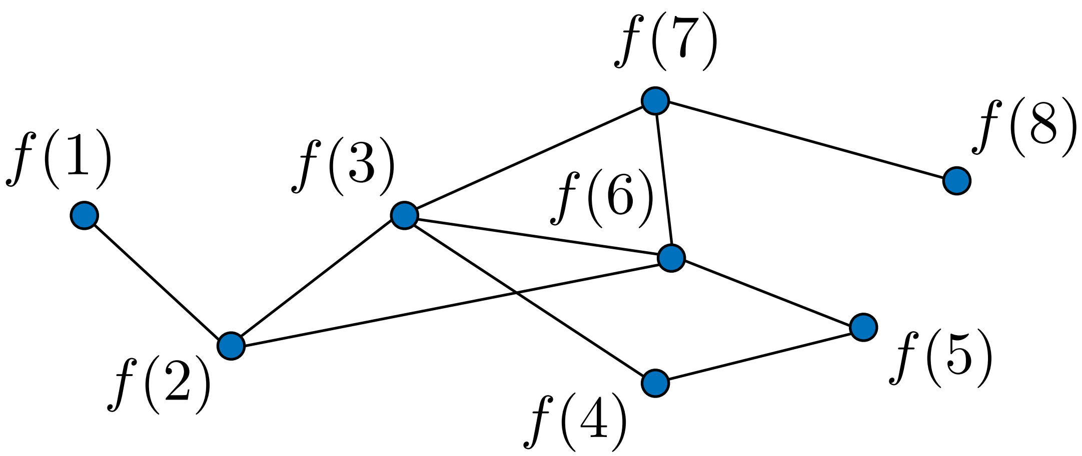
graph signal $f : \mathcal{V} \to \mathbb{R}$

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

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

Zhou and Schölkopf, "A regularization framework for learning from graph data," ICML Workshop, 2004.

Graph Laplacian



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

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

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

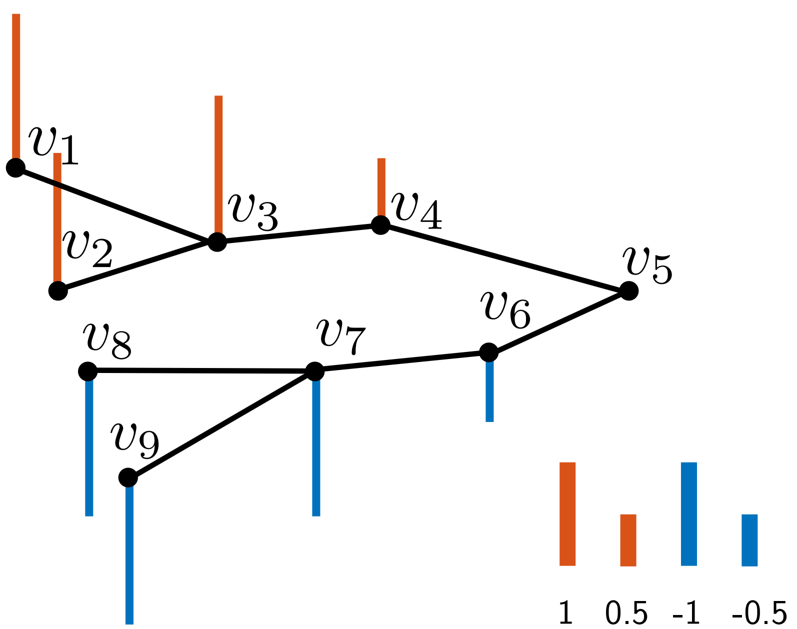
$$\begin{pmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{pmatrix}^T \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 &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

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

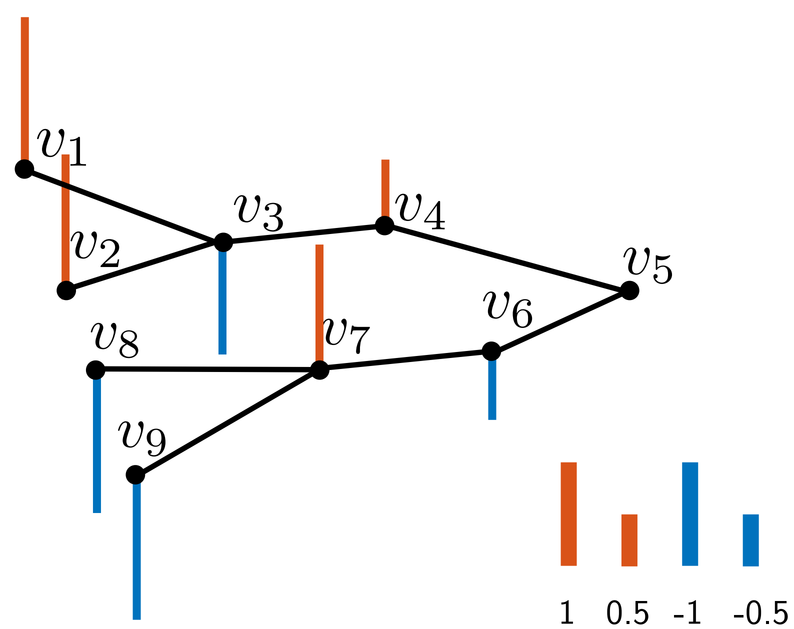
a measure of "smoothness"

Zhou and Schölkopf, "A regularization framework for learning from graph data," ICML Workshop, 2004.

Graph Laplacian



$$f^T L f = 1$$



$$f^T L f = 21$$

Graph Laplacian

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

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

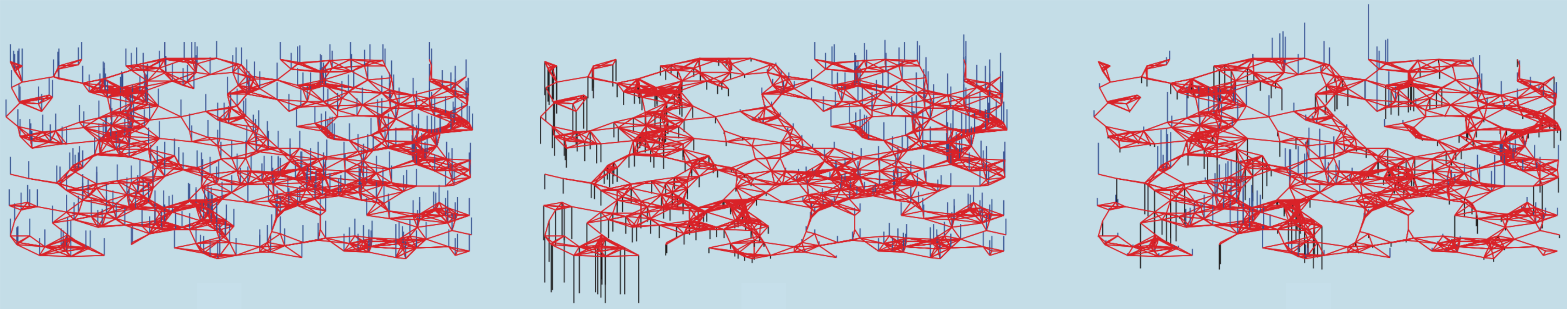
Graph Laplacian

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

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

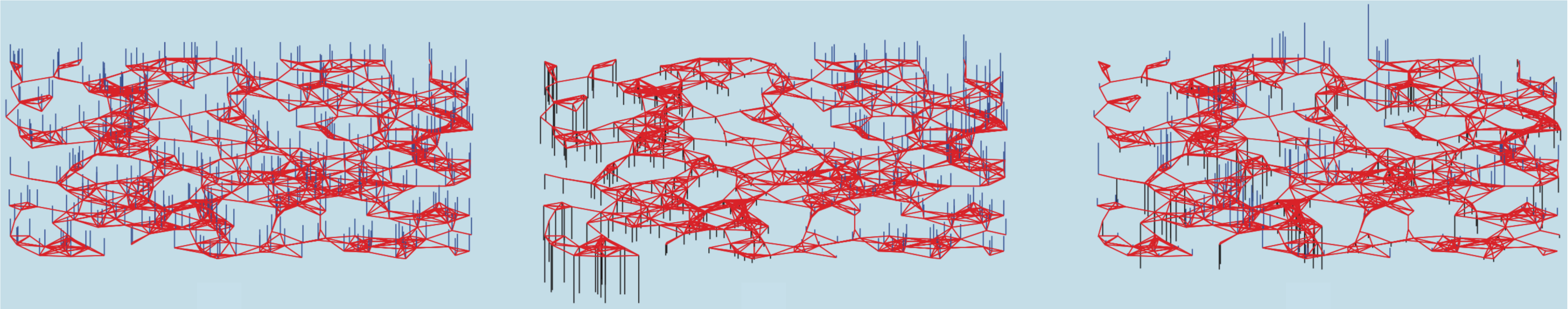
- Eigenvalues are usually sorted increasingly: $0 = \lambda_0 < \lambda_1 \leq \ldots \leq \lambda_{N-1}$

Graph Fourier transform



χ_0 χ_1 χ_{50}

Graph Fourier transform



χ_0 χ_1 χ_{50}

low frequency → high frequency

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

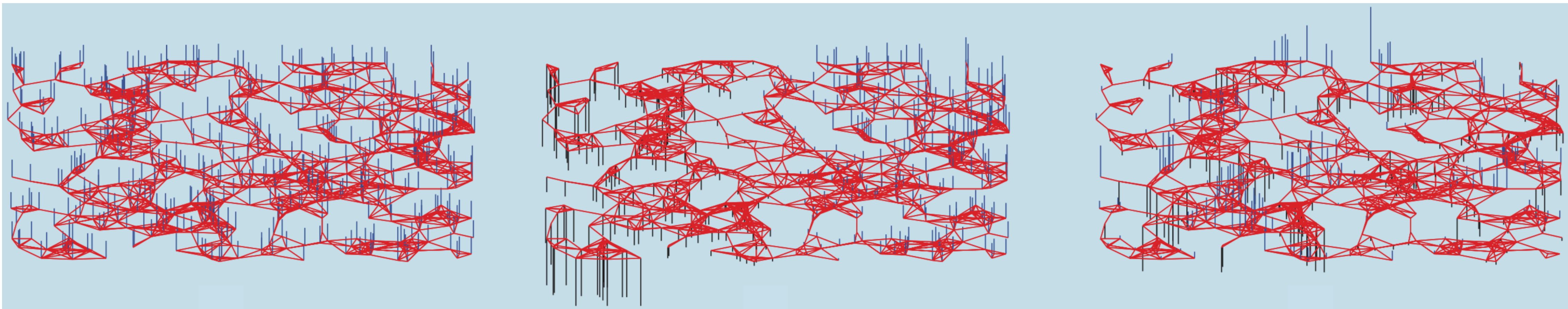
$\chi_0^T L \chi_0 = \lambda_0 = 0$

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

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

Shuman et al., "The emerging field of signal processing on graphs," IEEE SPM, 2013.

Graph Fourier transform



χ_0 χ_1 χ_{50}

low frequency → high frequency

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

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

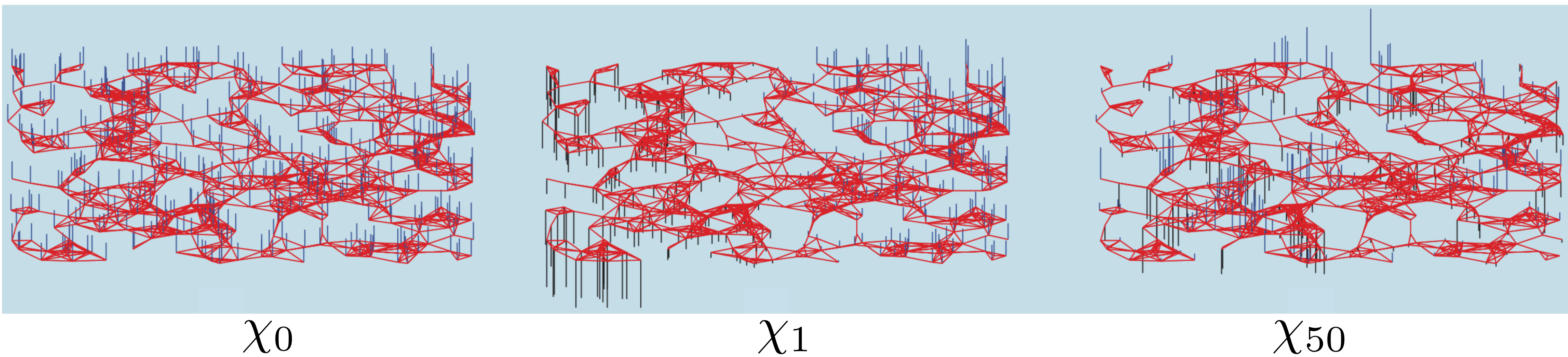
$\chi_{50}^T L \chi_{50} = \lambda_{50}$

graph Fourier transform:

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

Shuman et al., "The emerging field of signal processing on graphs," IEEE SPM, 2013.

Graph Fourier transform



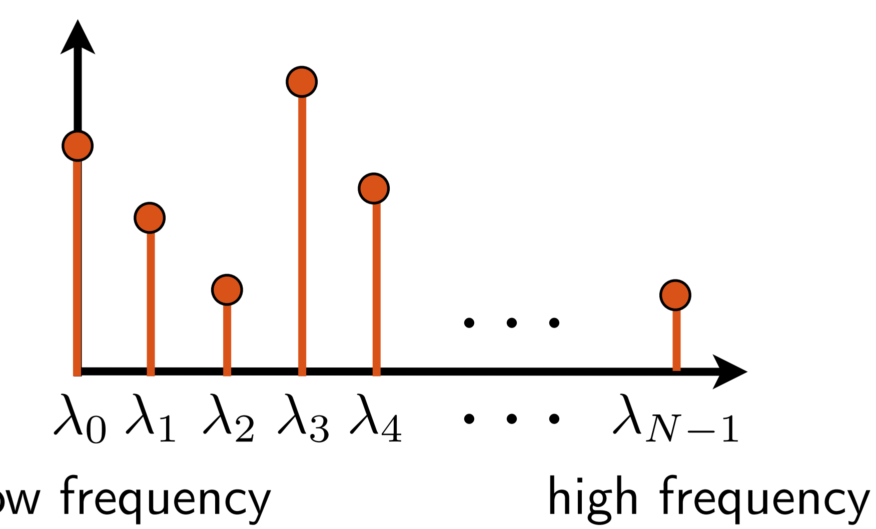
low frequency → high frequency

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

$\chi_0^T L \chi_0 = \lambda_0 = 0$ $\qquad$ $\chi_{50}^T L \chi_{50} = \lambda_{50}$

graph Fourier transform:

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



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: $-\nabla^2$	graph Laplacian: L
⬇	⬇
eigenfunctions: $e^{j\omega x}$	eigenvectors: χ_ℓ
⬇	⬇ $f: V \to \mathbb{R}^N$
Classical FT: $\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx$	Graph FT: $\hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^{N} \chi_\ell^*(i) f(i)$
$f(x) = \frac{1}{2\pi} \int \hat{f}(\omega) e^{j\omega x} d\omega$	$f(i) = \sum_{\ell=0}^{N-1} \hat{f}(\ell) \chi_\ell(i)$

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

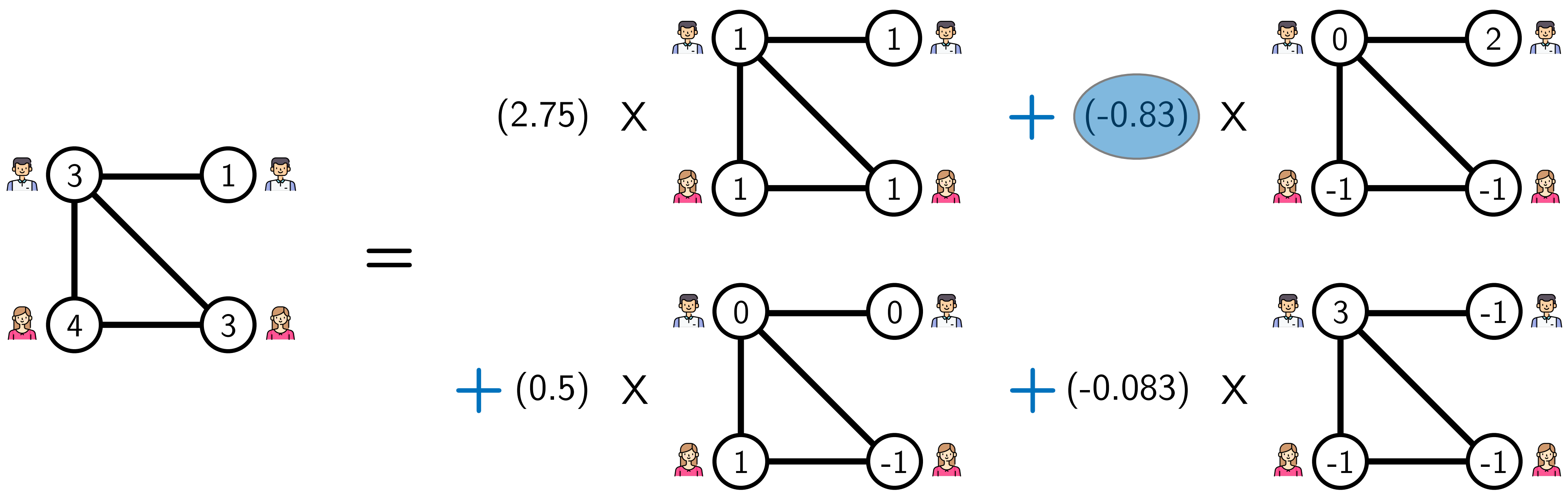
eigenvectors: χ_ℓ

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

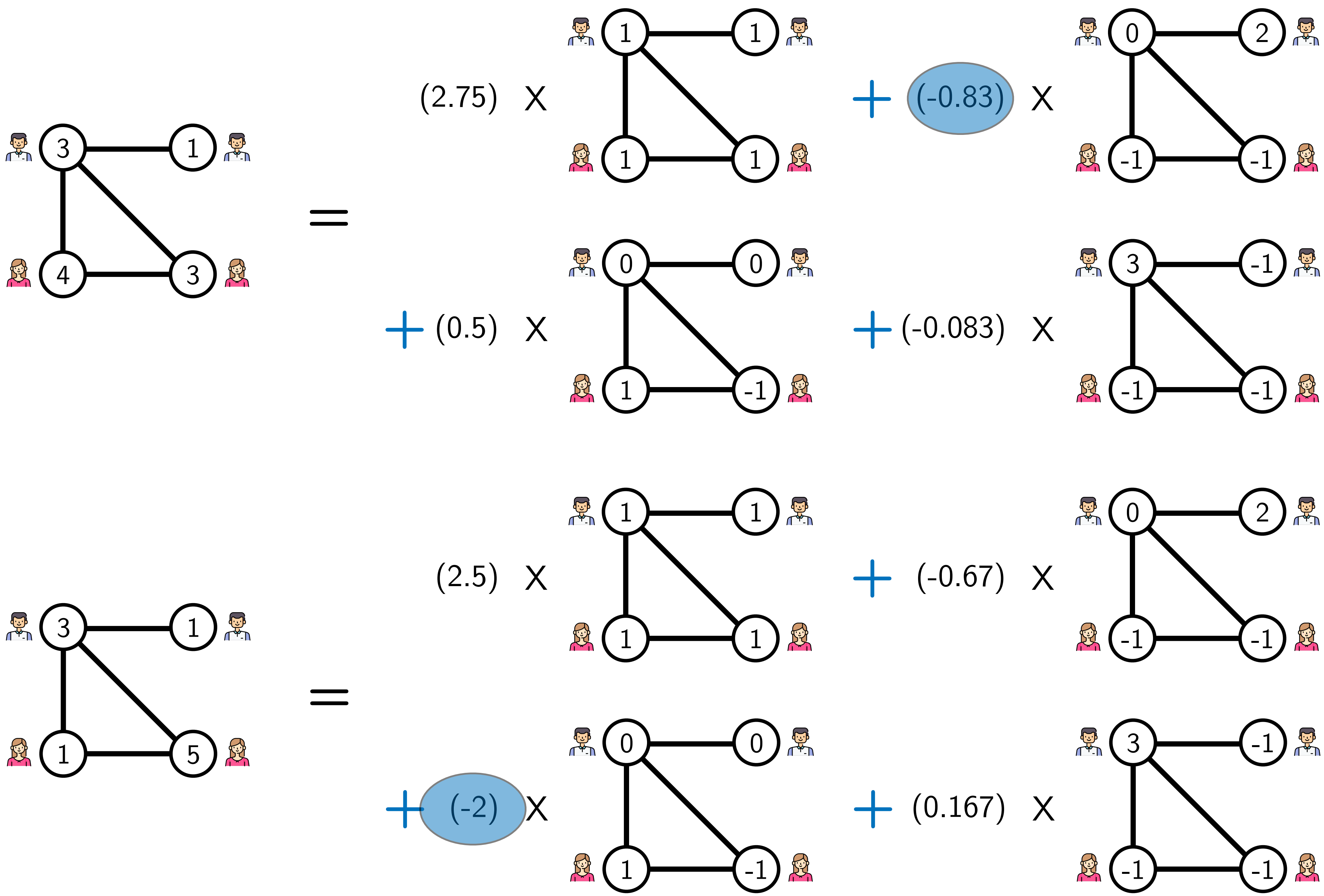
Graph FT: $\hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^{N} \chi_\ell^*(i) f(i)$

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

Example on movie rating



Example on movie rating

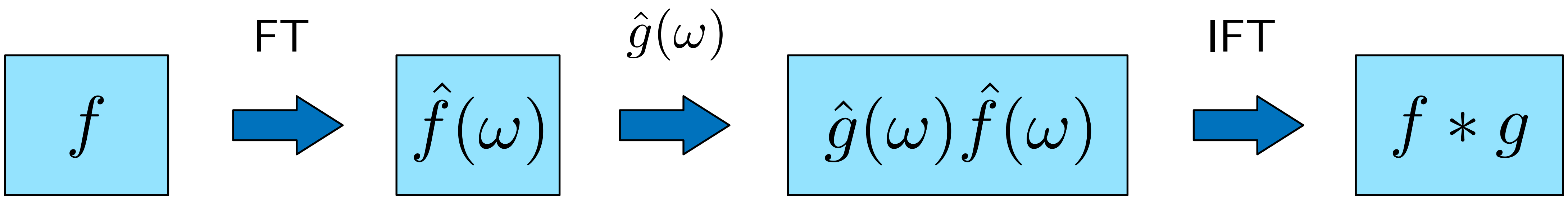


Classical frequency filtering

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

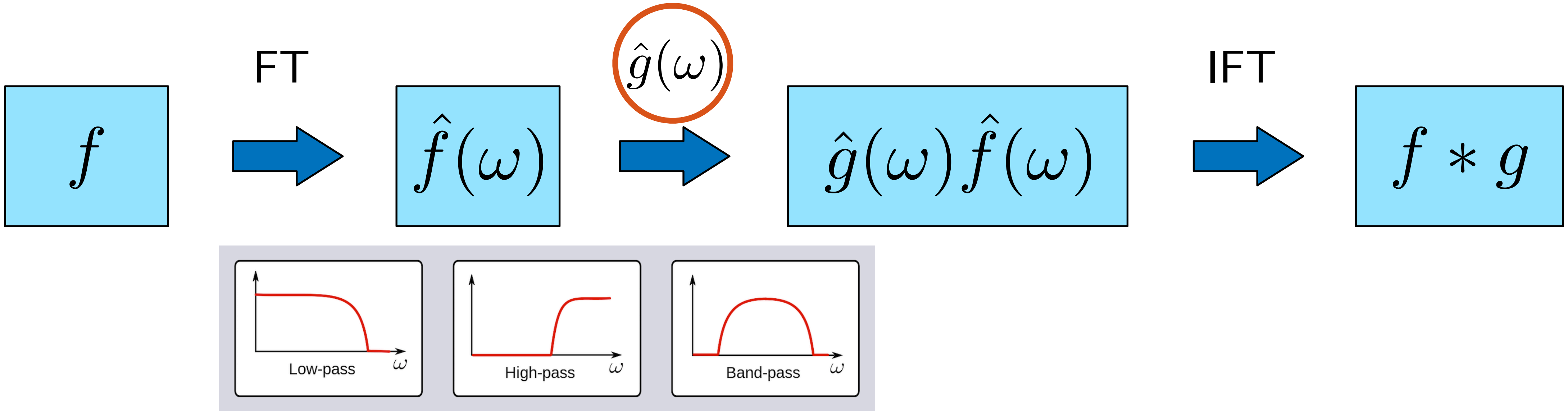
Classical frequency filtering

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



Classical frequency filtering

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



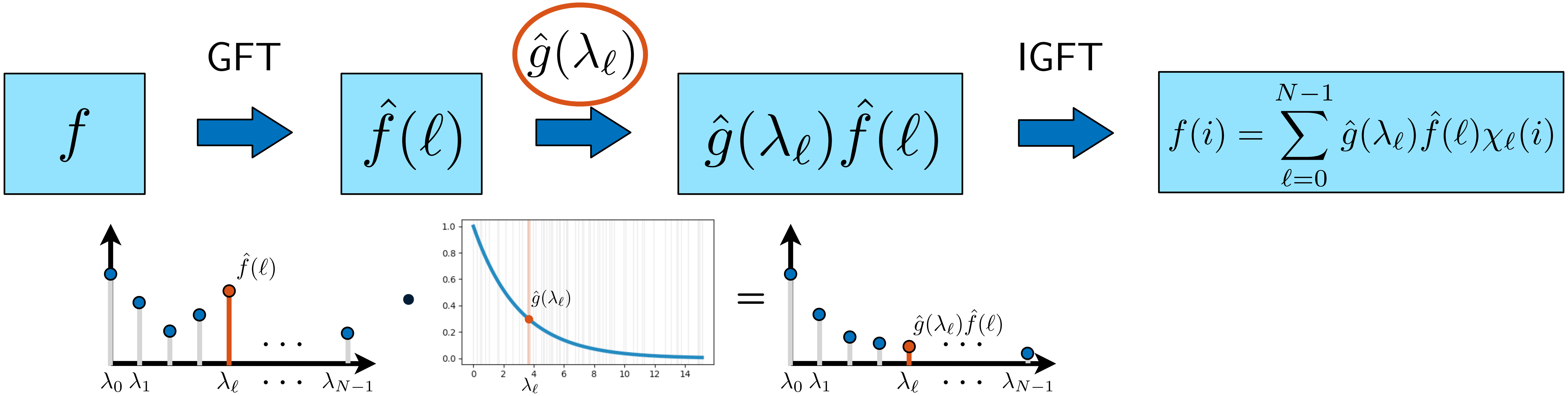
Graph spectral filtering

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

$$f \xrightarrow{\text{GFT}} \hat{f}(\ell) \xrightarrow{\hat{g}(\lambda_\ell)} \hat{g}(\lambda_\ell)\hat{f}(\ell) \xrightarrow{\text{IGFT}} f(i) = \sum_{\ell=0}^{N-1} \hat{g}(\lambda_\ell)\hat{f}(\ell)\chi_\ell(i)$$

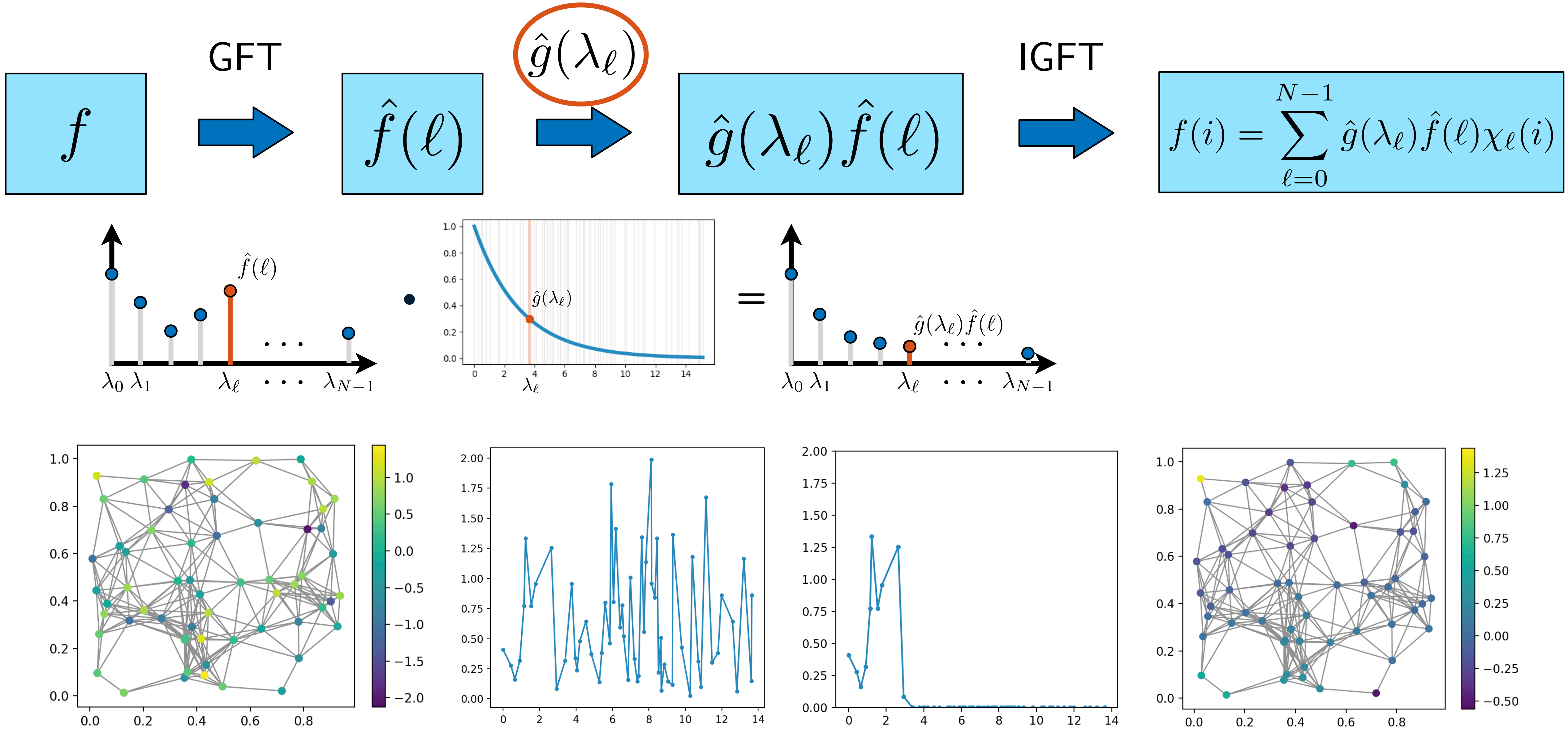
Graph spectral filtering

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



Graph spectral filtering

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



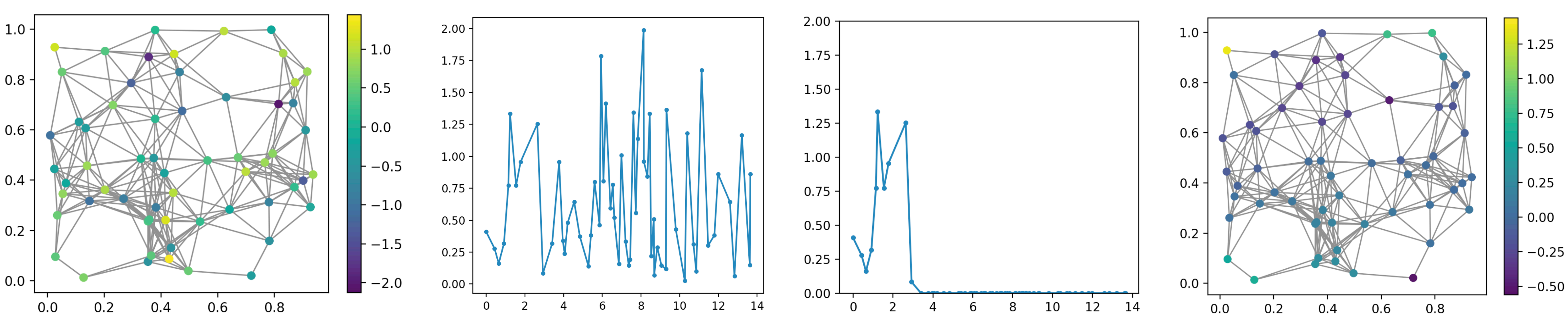
Graph spectral filtering

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

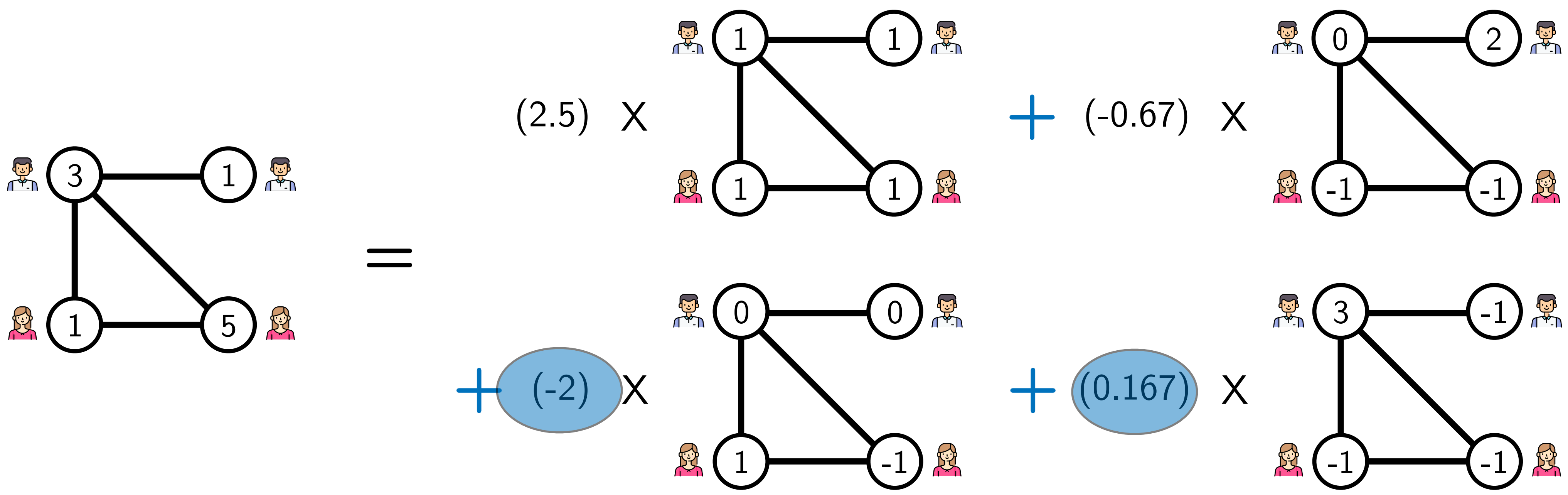
f —GFT→ $\chi^T f$ —$\hat{g}(\Lambda)$→ $\hat{g}(\Lambda)\chi^T f$ —IGFT→ $\chi\hat{g}(\Lambda)\chi^T f$

$$\hat{g}(\Lambda) = \begin{bmatrix} \hat{g}(\lambda_0) & & 0 \\ & \ddots & \\ 0 & & \hat{g}(\lambda_{N-1}) \end{bmatrix}$$

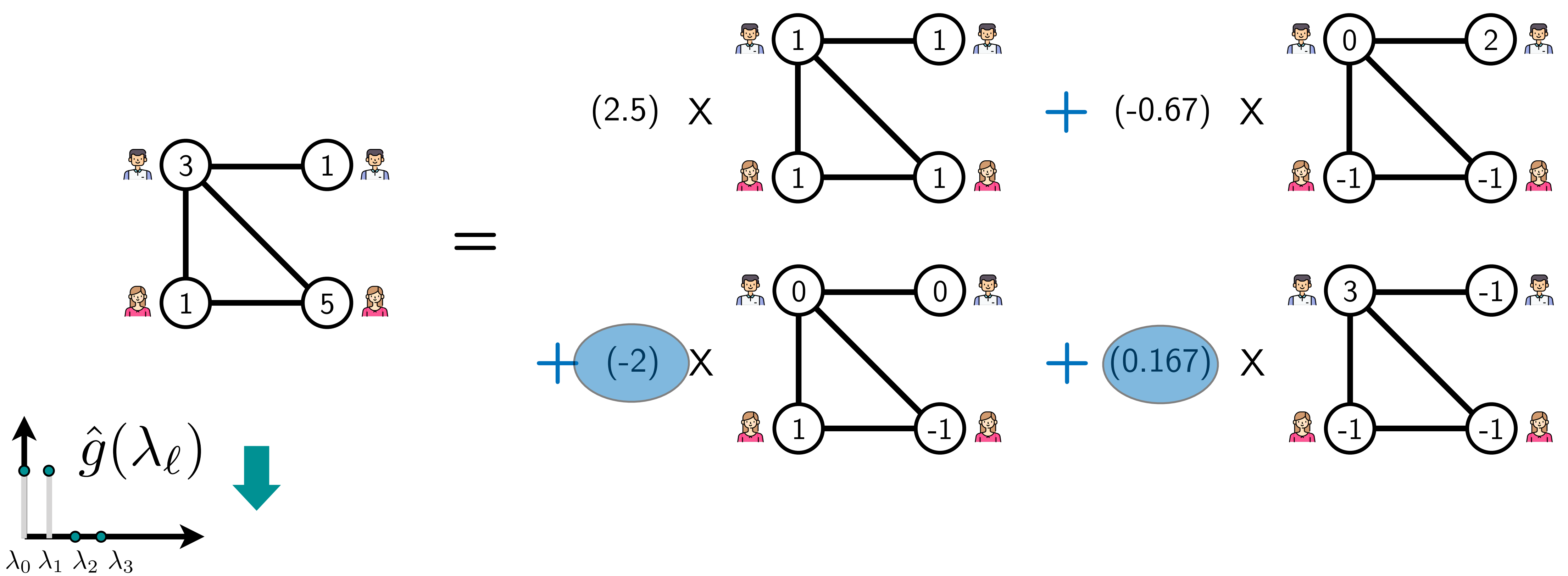
$\hat{g}(L)$**: function of L!**



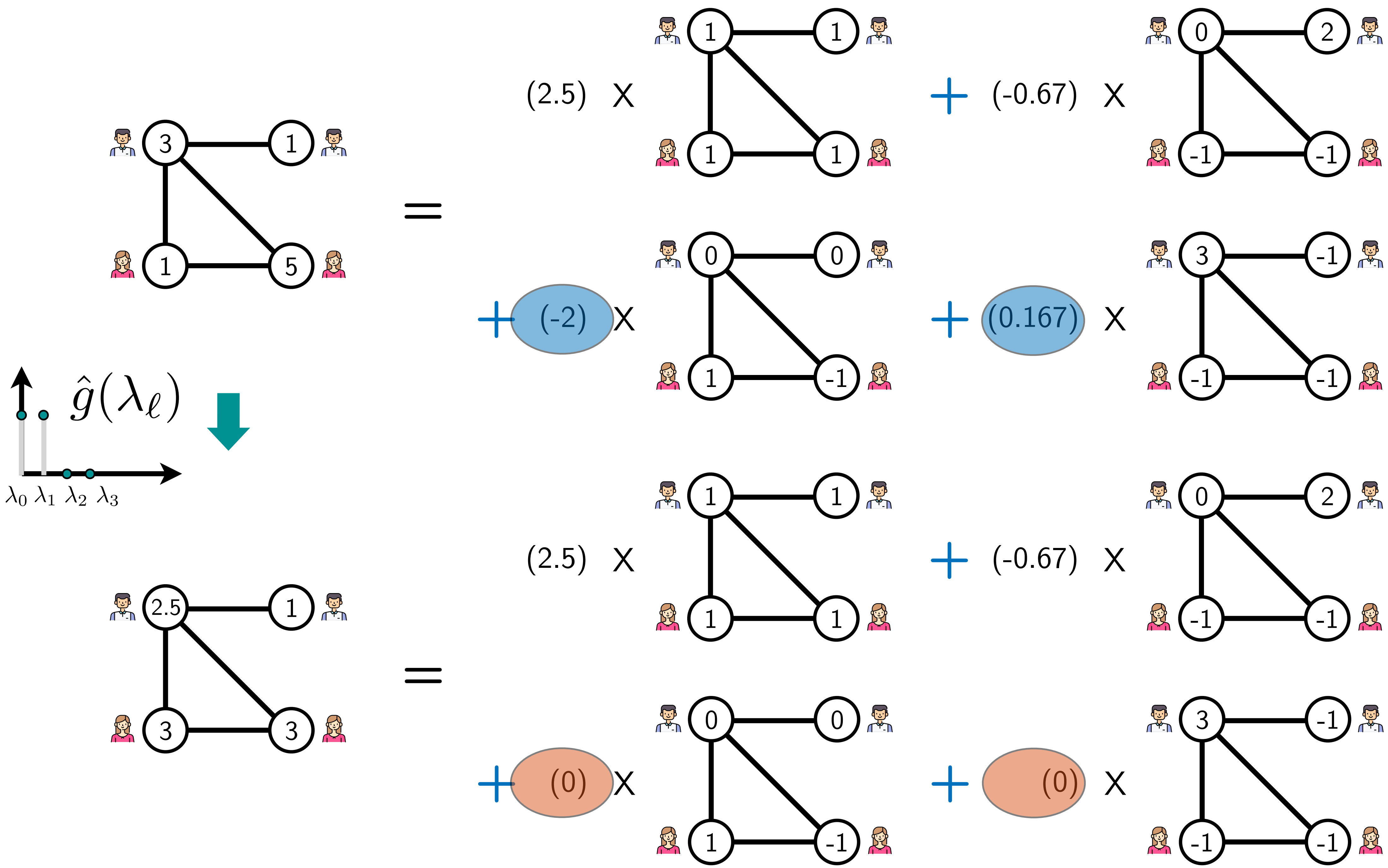
Example on movie rating



Example on movie rating



Example on movie rating



Convolution on graphs

classical convolution

convolution on graphs

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

classical convolution

convolution on graphs

time domain

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

frequency domain

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

graph spectral domain

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

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

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

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

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

convolution = filtering

graph spectral domain

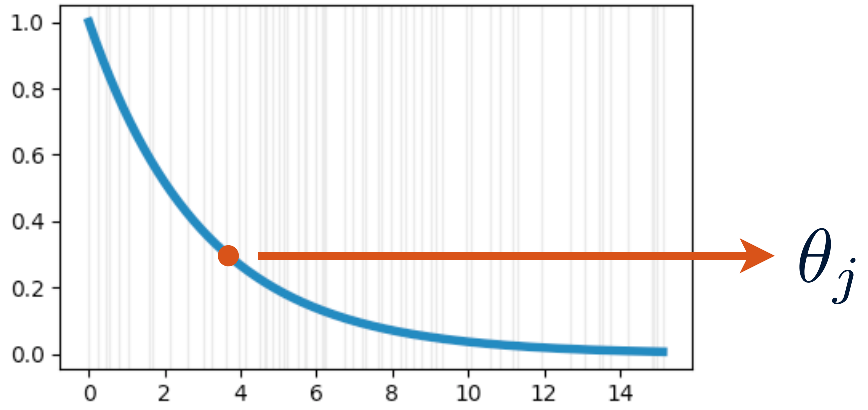
$$\widehat{(f * g)}(\lambda) = \big((\chi^T f) \circ \hat{g}\big)(\lambda)$$

A non-parametric filter

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

learning a non-parametric filter:

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



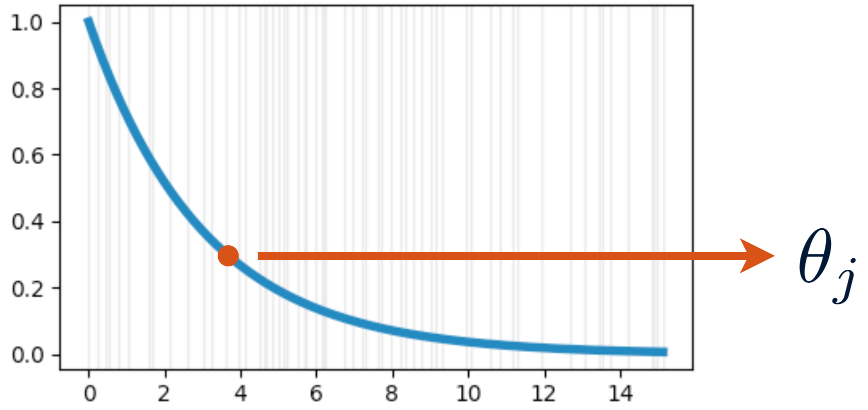
Bruna et al., "Spectral networks and deep locally connected networks on graphs," ICLR, 2014.

A non-parametric filter

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

learning a non-parametric filter:

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



- convolution expressed in the graph spectral domain
- no localisation in the spatial (node) domain
- computationally expensive (e.g., eigendecomposition)

Bruna et al., "Spectral networks and deep locally connected networks on graphs," ICLR, 2014.

A parametric filter

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

⇓

parametric filter as polynomial of Laplacian

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

Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

A parametric filter

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

parametric filter as polynomial of Laplacian

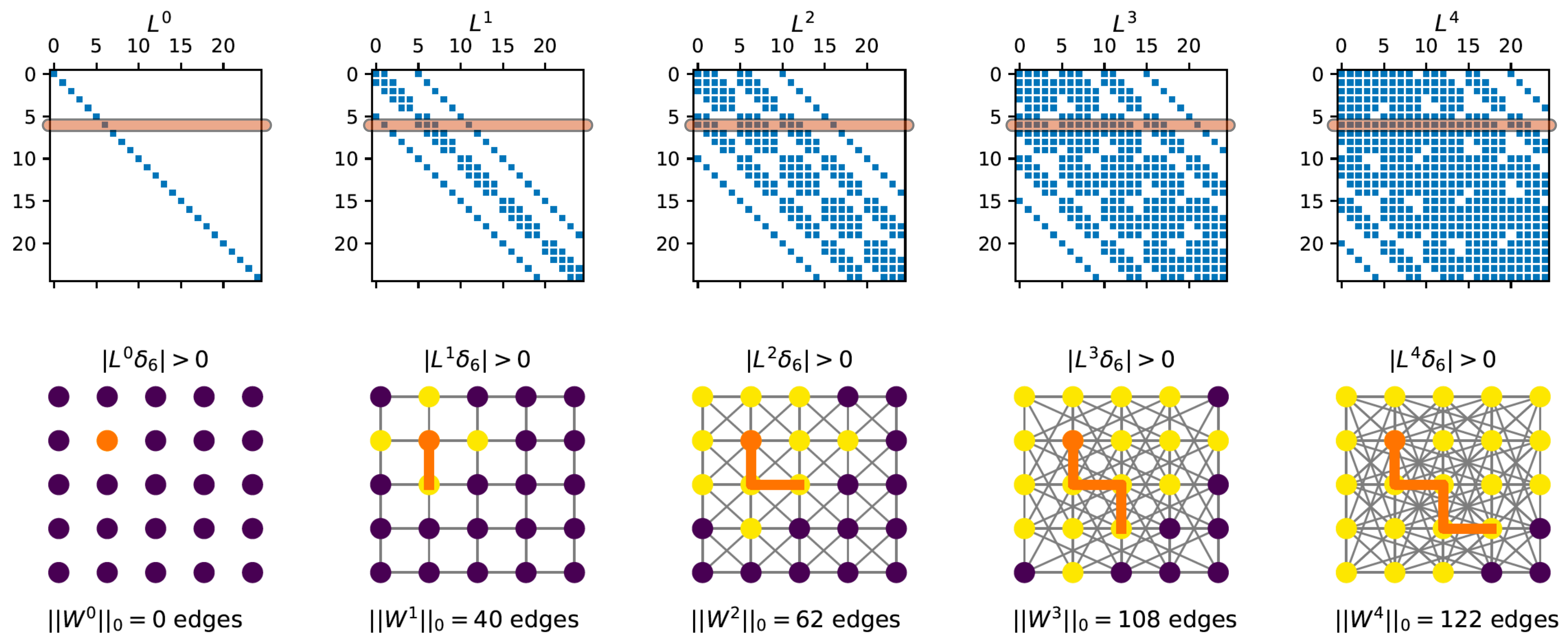
$$\hat{g}_\theta(\lambda) = \sum_{j=0}^{K} \theta_j \lambda^j, \ \theta \in \mathbb{R}^{K+1} \quad \Rightarrow \quad \hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$$

what do powers of graph Laplacian capture?

Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

Powers of graph Laplacian

L^k defines the k-neighborhood



(source: M. Deferrard)

Localization: $d_{\mathcal{G}}(v_i, v_j) > K$ implies $(L^K)_{ij} = 0$

Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

A parametric filter

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

parametric filter as polynomial of Laplacian

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

Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

A parametric filter

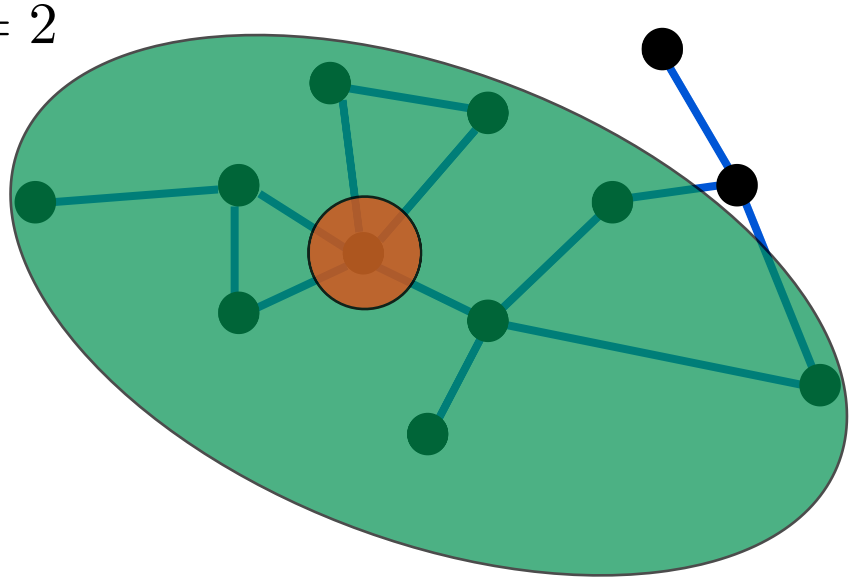
$$f * g = \chi \hat{g}(\Lambda) \chi^T f = \hat{g}(L) f$$

parametric filter as polynomial of Laplacian

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

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

$K = 2$



- localisation within *K*-hop neighbourhood

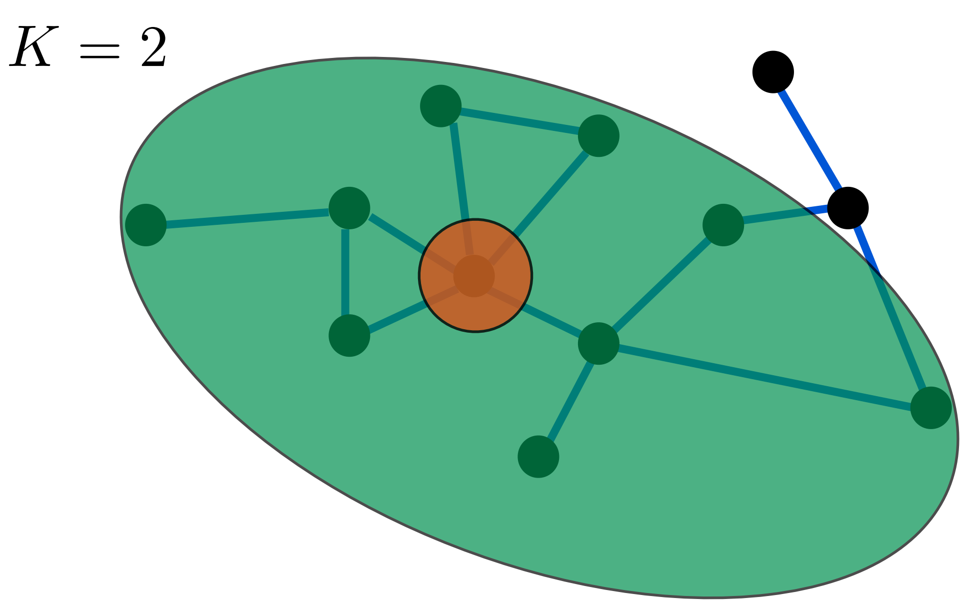
Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

A parametric filter

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

parametric filter as polynomial of Laplacian

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



- localisation within *K*-hop neighbourhood
- Chebyshev approximation for efficient computation via recursive multiplication
- scaled Laplacian for stability in learning

Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

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

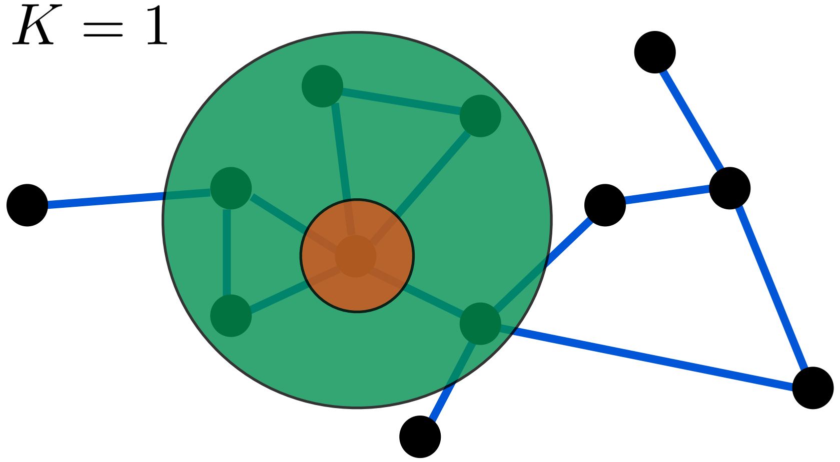
$$= \theta_0 I - \theta_1 (D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$

(localisation within 1-hop neighbourhood)

normalised Laplacian

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

$K = 1$



Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

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

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

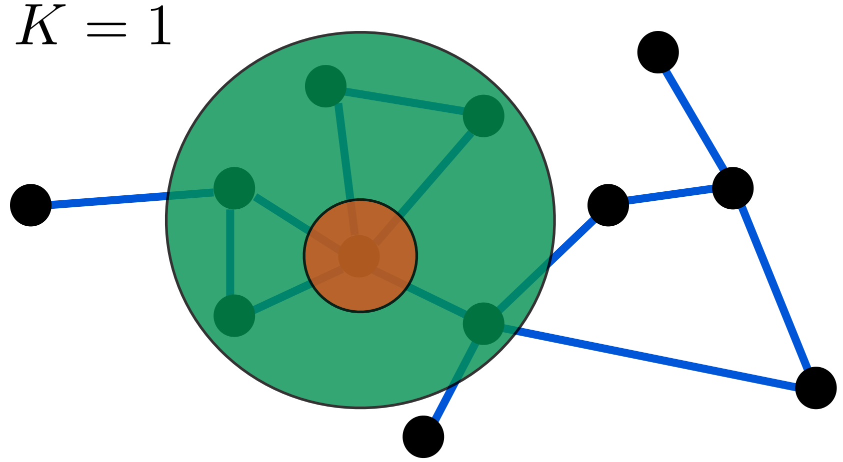
$K = 1$

normalised Laplacian

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

(localisation within 1-hop neighbourhood)

$K = 1$



$\alpha = \theta_0 = -\theta_1$

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

Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

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

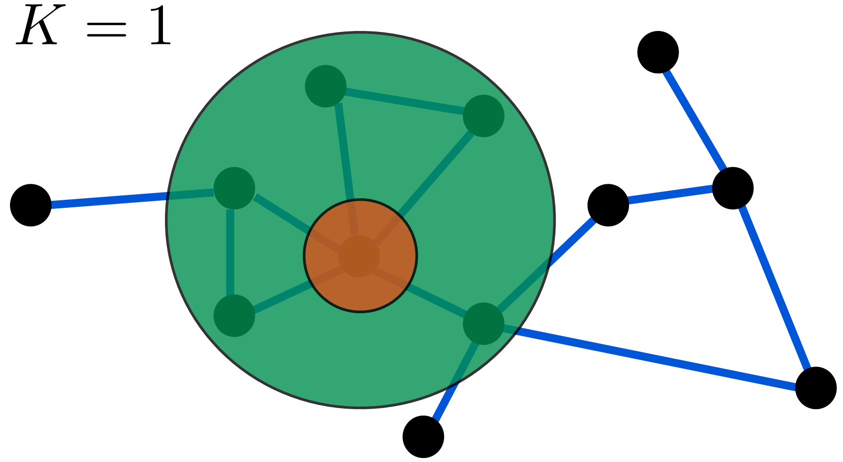
$$\begin{aligned} L_{\text{norm}} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \\ &= D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} \\ &= I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = I - W_{\text{norm}} \end{aligned}$$

$K = 1$
normalised Laplacian

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

(localisation within 1-hop neighbourhood)

$K = 1$



$\alpha = \theta_0 = -\theta_1$

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

renormalisation

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

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

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

Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

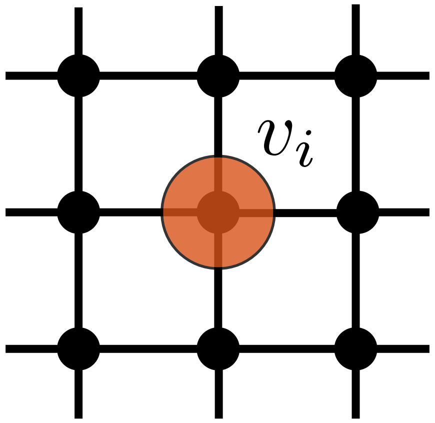
A simplified parametric filter

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

simplified parametric filter

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

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



Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

A simplified parametric filter

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

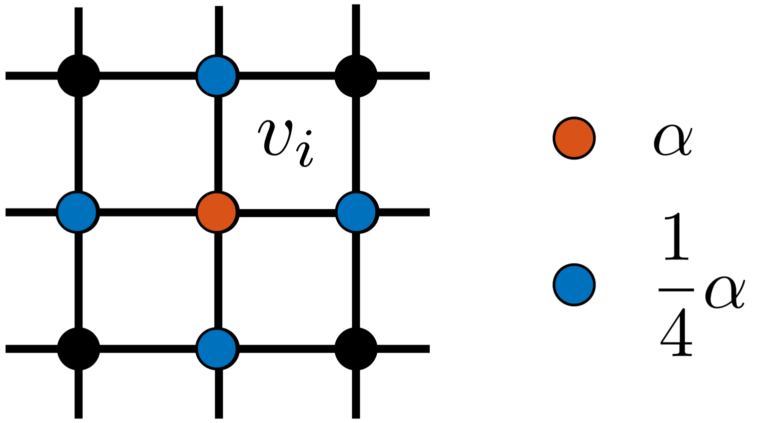
simplified parametric filter

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

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

unitary edge weights

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



Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

A simplified parametric filter

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

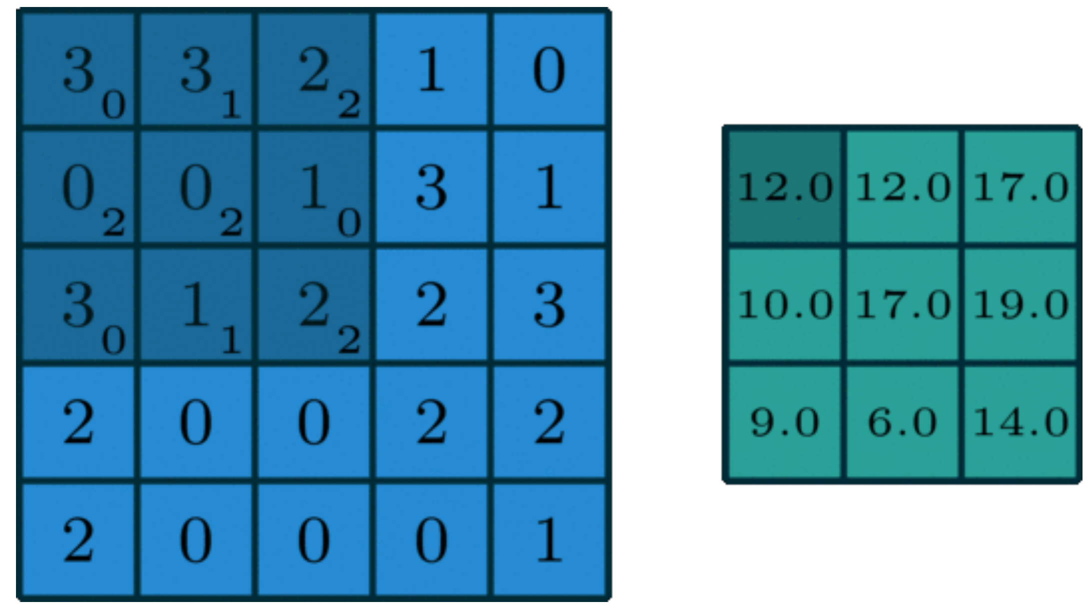
simplified parametric filter

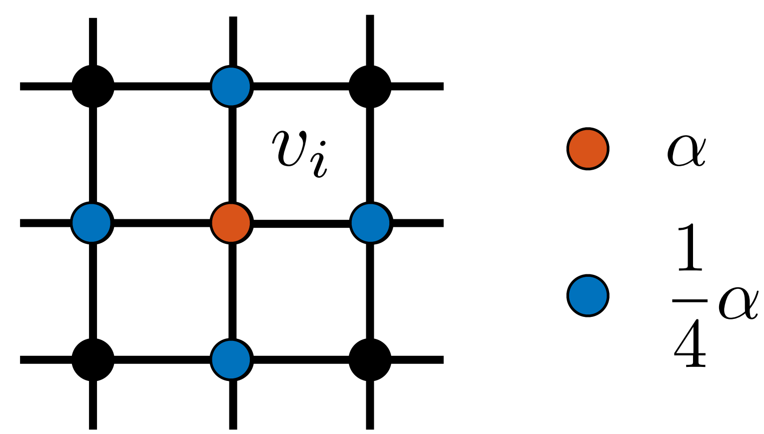
$$\hat{g}_\alpha(L) = \alpha(I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$

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

unitary edge weights

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





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
  - general filter: receptive field size determined by degree of polynomial

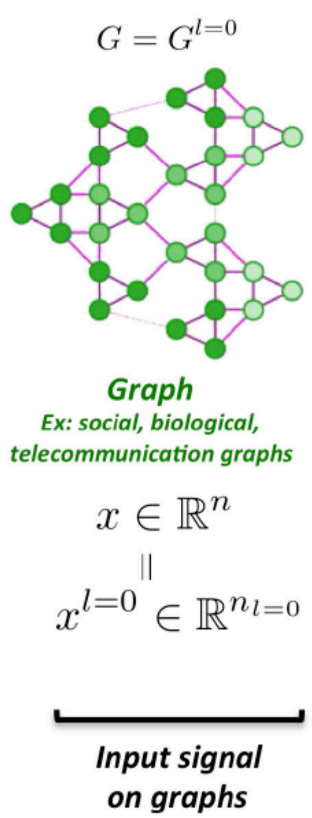
Convolution on graphs - Remarks

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

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

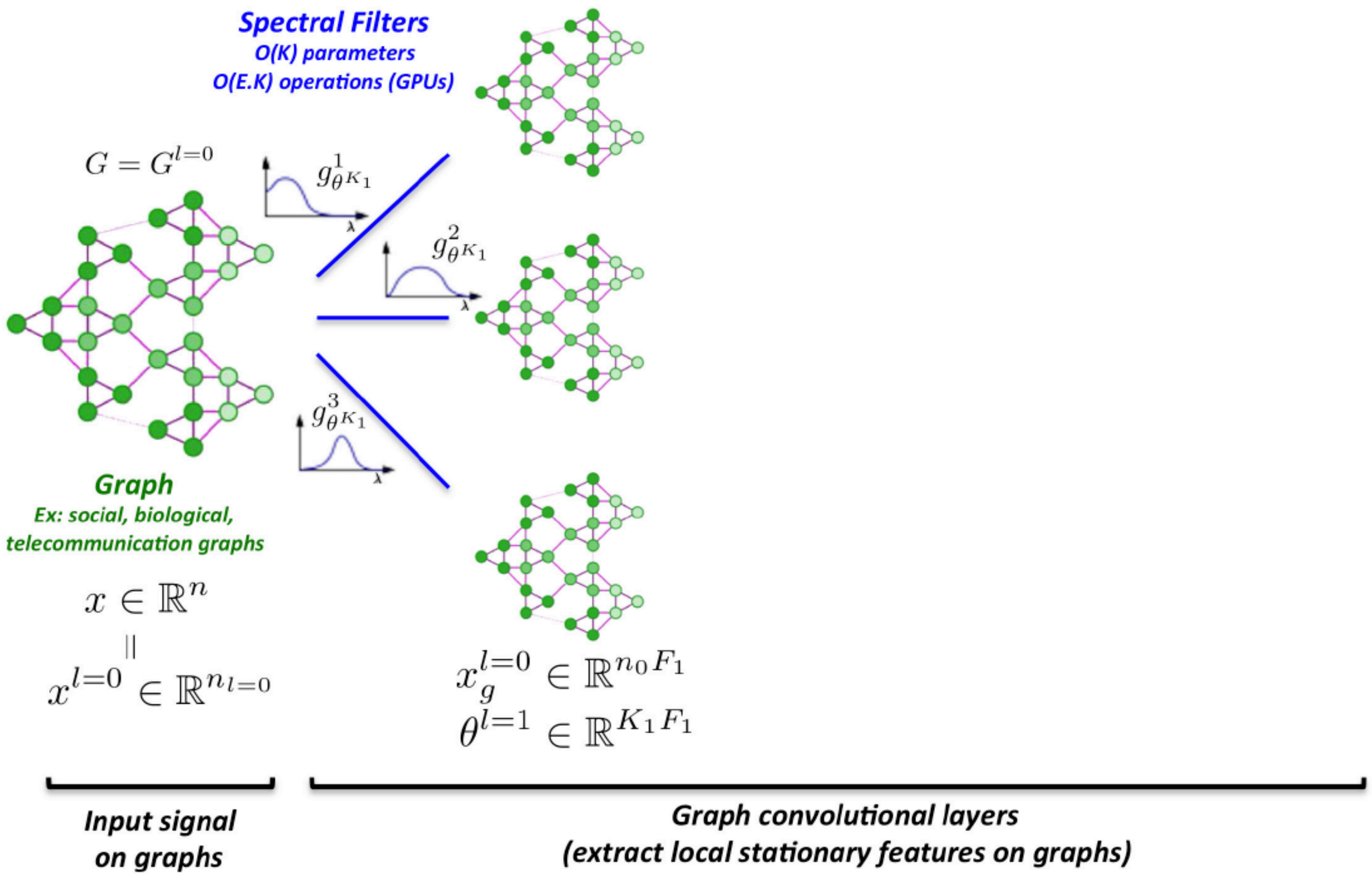
- ..but can be implemented in the **spatial (node)** domain
  - simplified filter: $y = \hat{g}_\theta(L) f = \alpha(\tilde{D}^{-\frac{1}{2}} \tilde{W} \tilde{D}^{-\frac{1}{2}}) f$
  - interpretation: at each layer nodes exchange information in 1-hop neighbourhood
  - general filter: receptive field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

ChebNet



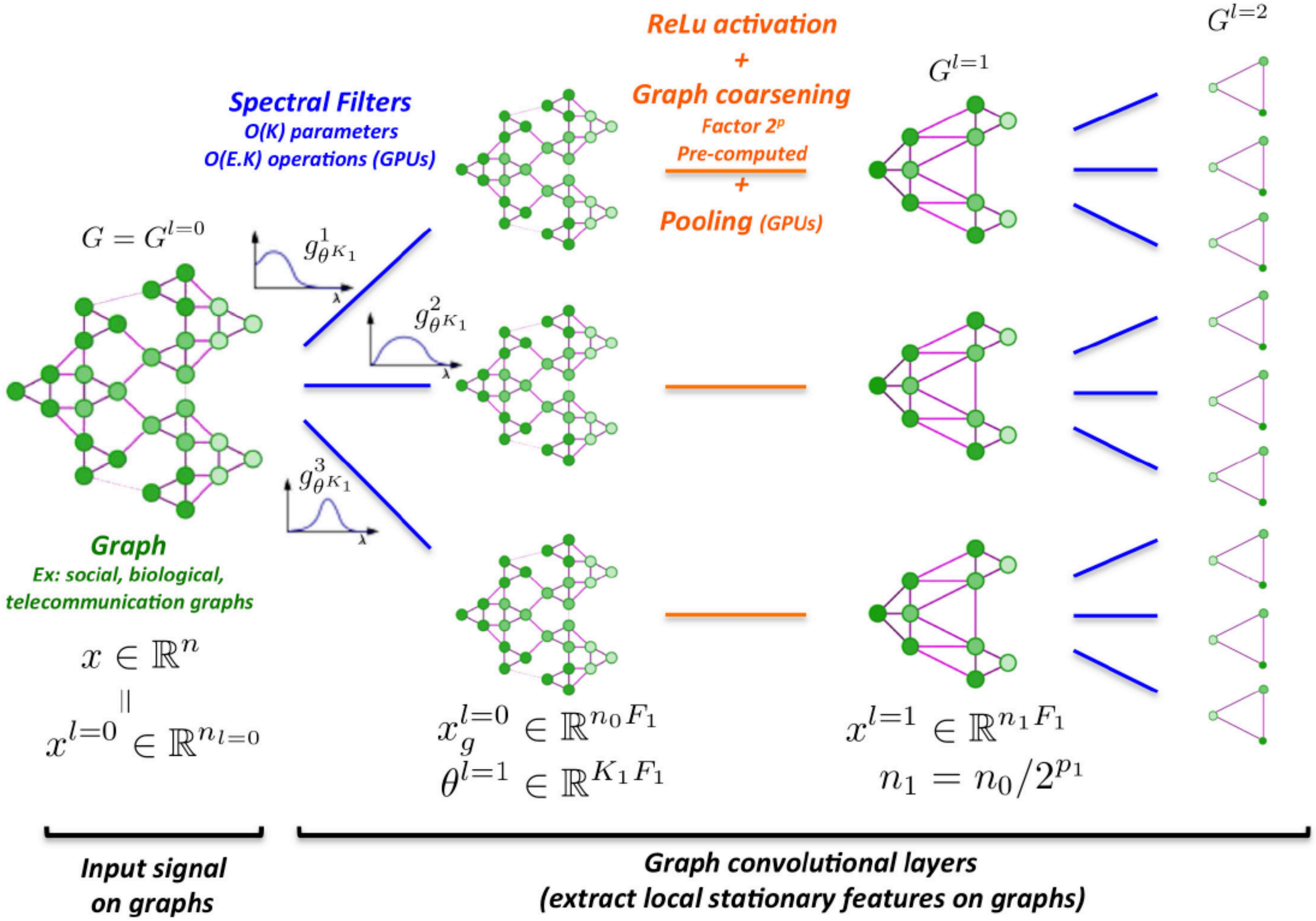
Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

ChebNet



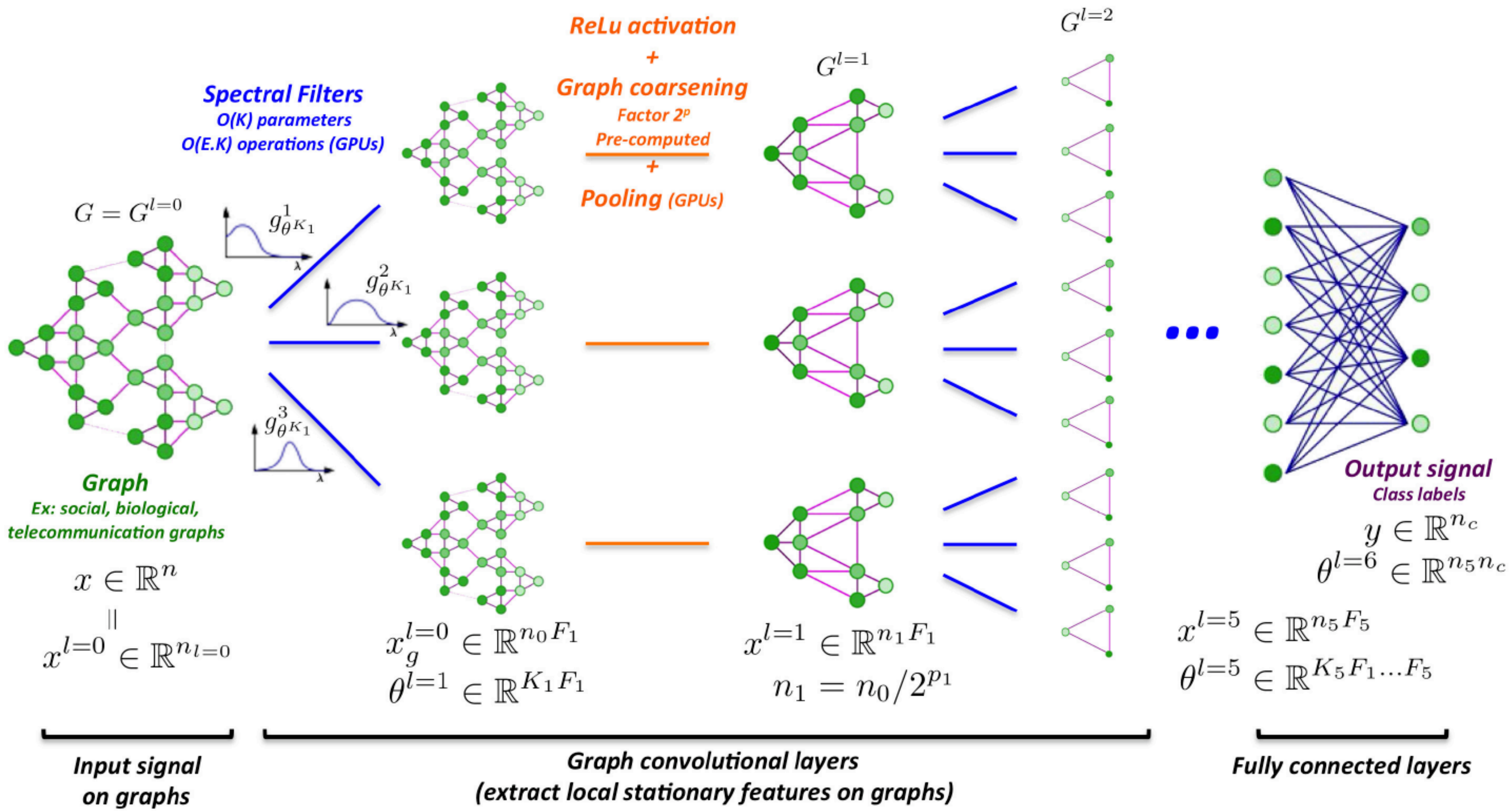
Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

ChebNet



Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

ChebNet



Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

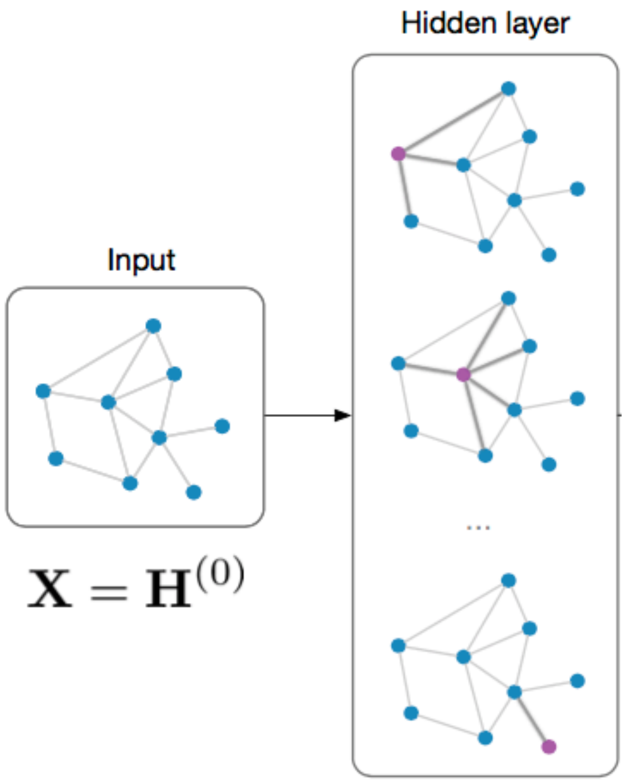
Graph convolutional network

$$\hat{g}_{\theta^{(k+1)}}(L)\Big(\mathrm{ReLU}(\hat{g}_{\theta^{(k)}}(L)f)\Big)$$

$\mathbf{X} = \mathbf{H}^{(0)}$

Graph convolutional network

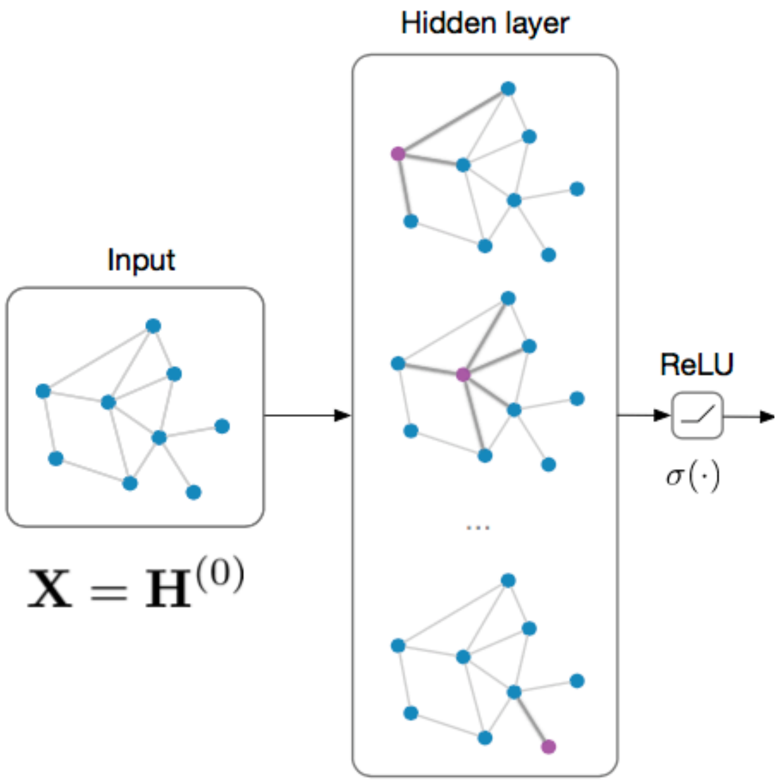
$$\hat{g}_{\theta^{(k+1)}}(L)\left(\mathrm{ReLU}(\hat{g}_{\theta^{(k)}}(L)f)\right)$$



Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

Graph convolutional network

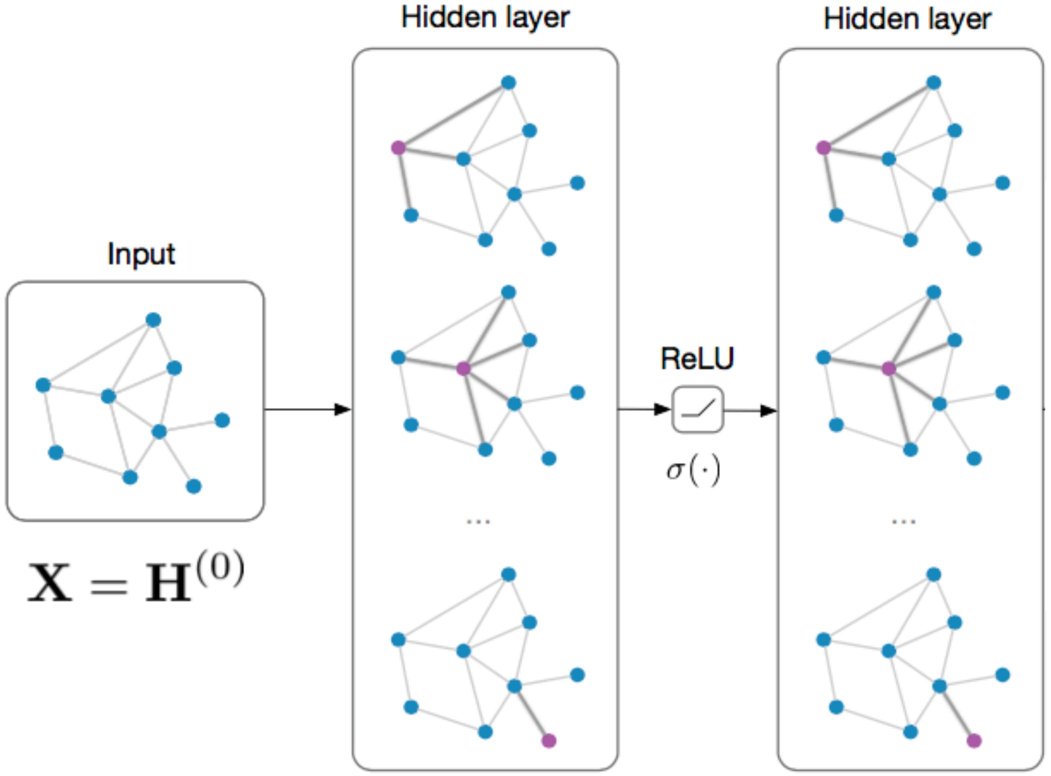
$$\hat{g}_{\theta^{(k+1)}}(L)\left(\mathrm{ReLU}(\hat{g}_{\theta^{(k)}}(L)f)\right)$$



Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

Graph convolutional network

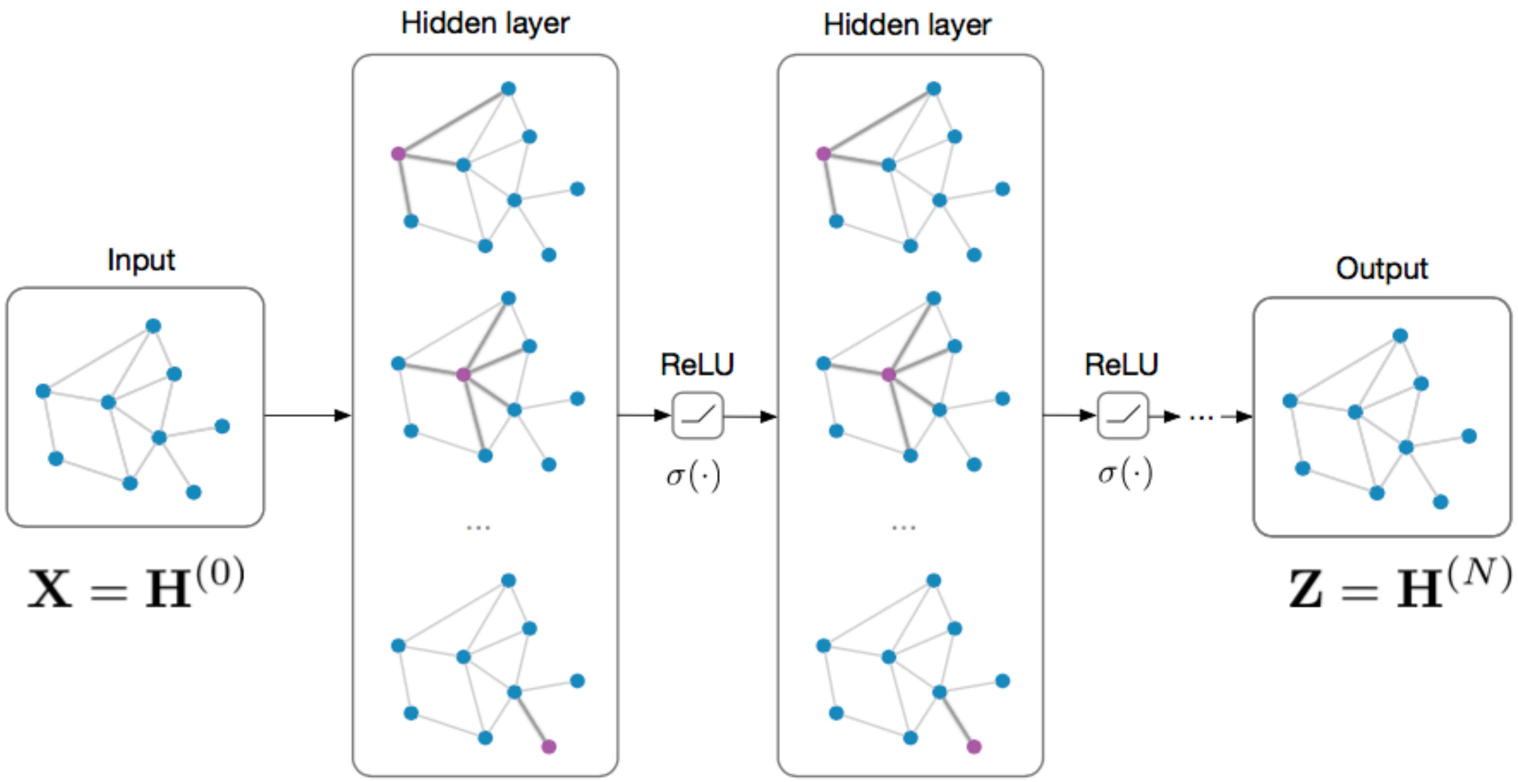
$$\hat{g}_{\theta^{(k+1)}}(L)\Big(\mathrm{ReLU}(\hat{g}_{\theta^{(k)}}(L)f)\Big)$$



Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

Graph convolutional network

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

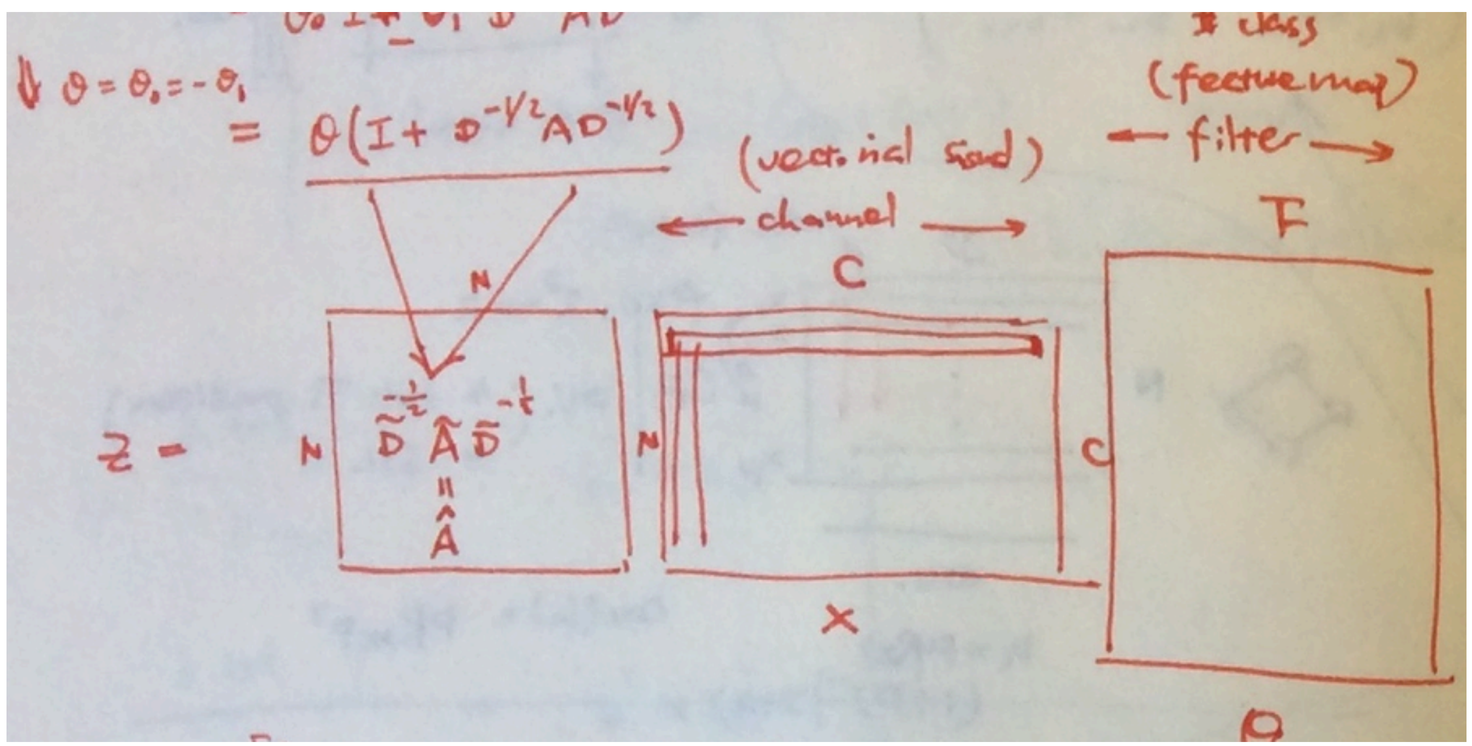


$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Kipf and Welling, "Semi-supervised classification with graph convolutional networks," ICLR, 2017.

Graph convolutional network

$$\hat{g}_{\theta^{(k+1)}}(L)\Big(\mathrm{ReLU}(\hat{g}_{\theta^{(k)}}(L)f)\Big)$$



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

Implementing CNNs on graphs

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

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

set of labelled (training) nodes

label groundtruth

label prediction (final layer node representation)

  - training by minimising loss function and making predictions on testing nodes

Implementing CNNs on graphs

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

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

set of labelled (training) nodes

label groundtruth

label prediction (final layer node representation)

  - training by minimising loss function and making predictions on testing nodes

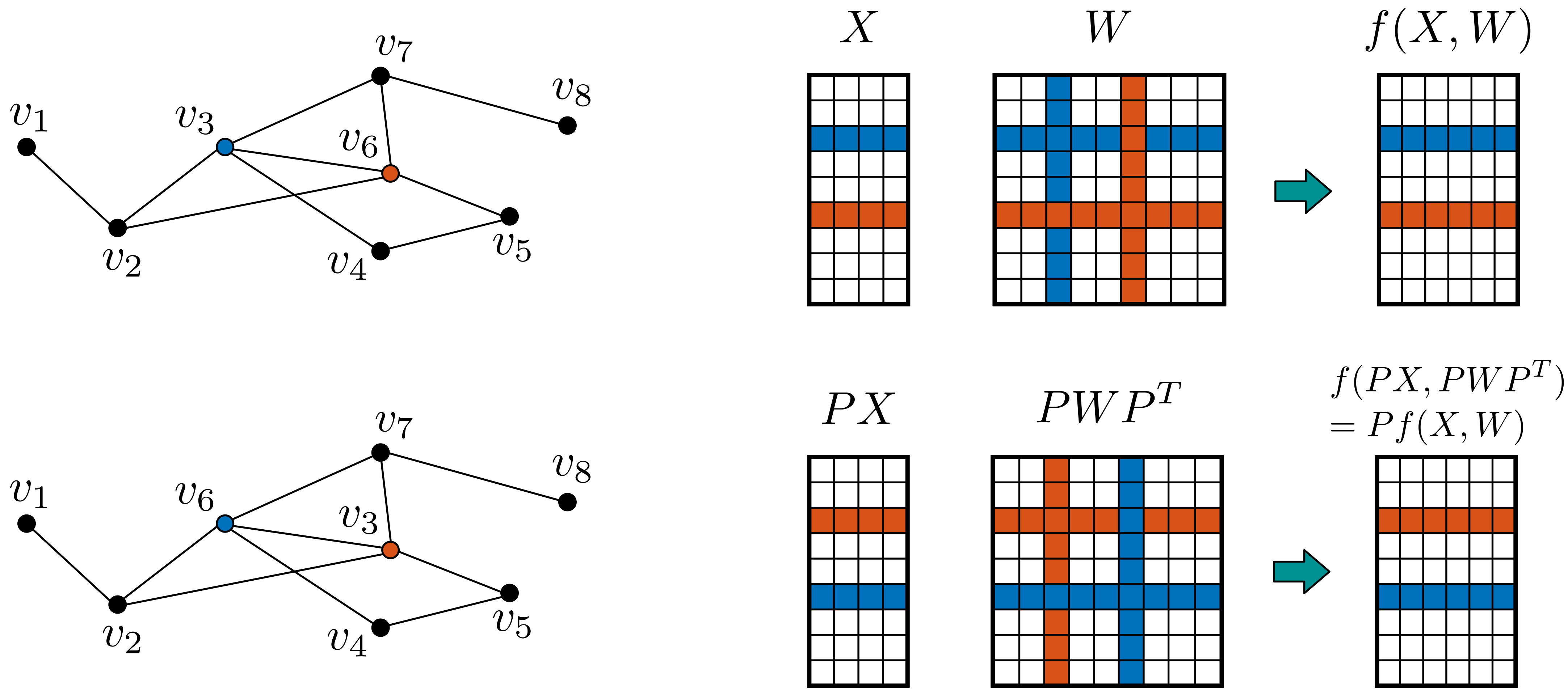
- Factors influencing model behaviour
  - graph connectivity, label distribution, neural architecture, training dynamics

Graph Machine Learning

- Overview of graph machine learning
- Convolutional neural networks on graphs
  - "spectral" approaches enabled by graph signal processing
- State-of-the-art graph neural networks
  - "spatial" approaches enabled by message passing
- Latest developments and applications

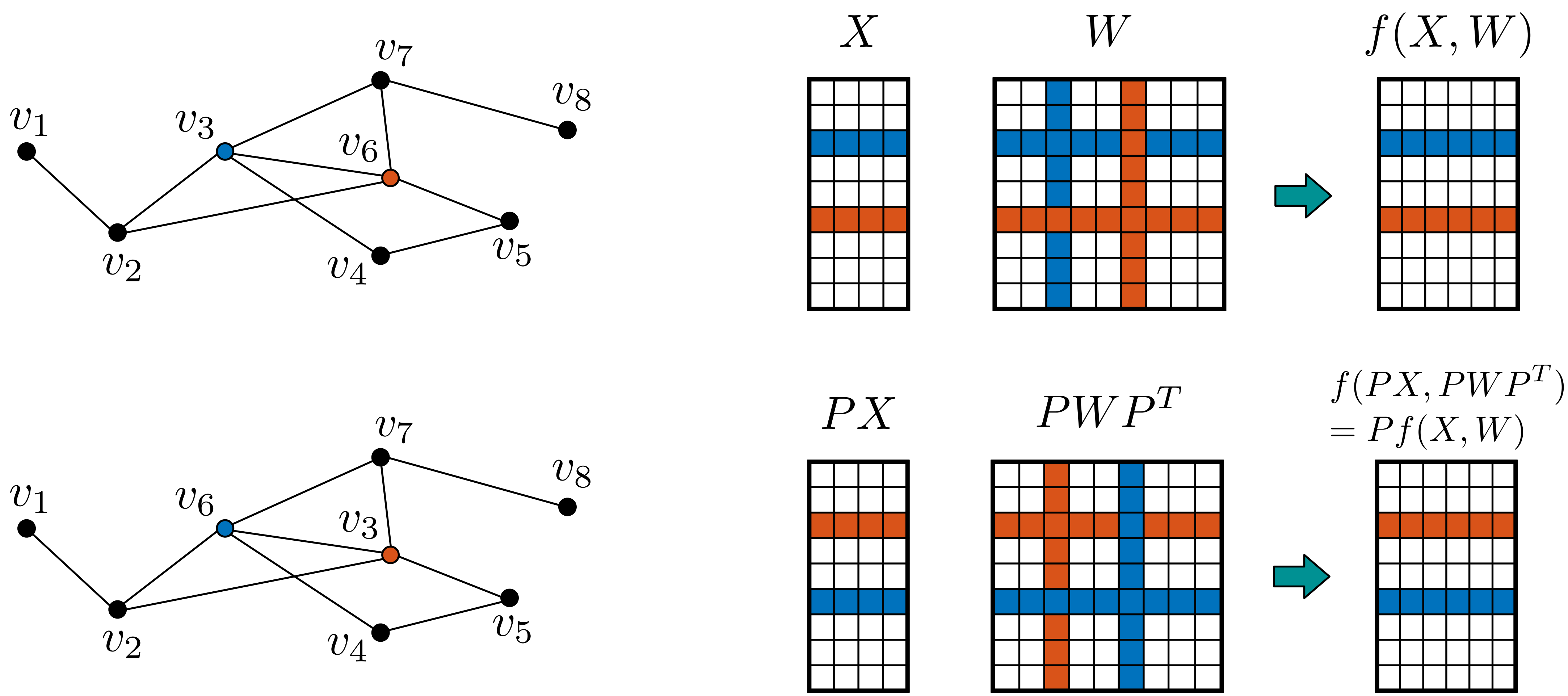
A look into invariance

- Permutation equivariance: function equivariant w.r.t. permutation (permutation of input should lead to same permutation of output)



A look into invariance

- Permutation equivariance: function equivariant w.r.t. permutation (permutation of input should lead to same permutation of output)



- Spectral GNNs: $\hat{g}_\theta(L)$ is permutation equivariant because it acts on **local neighbourhood** and its behaviour is **invariant to permutation**

GNNs - A spatial approach

- Graph convolution can also be defined in spatial (node) domain

via a graph shift operator ➡ $$g(S)x = \sum_{k=0}^{K} \theta_k S^k x$$

via a spatial weighted summation ➡ $$(x * g)(v) = \sum_{v' \in \mathcal{V}} x(v') g(v, v')$$

Isufi et al., "Graph filters for signal processing and machine learning on graphs," IEEE TSP, 2024.
Walker and Glocker, "Graph convolutional Gaussian processes," ICML, 2019.

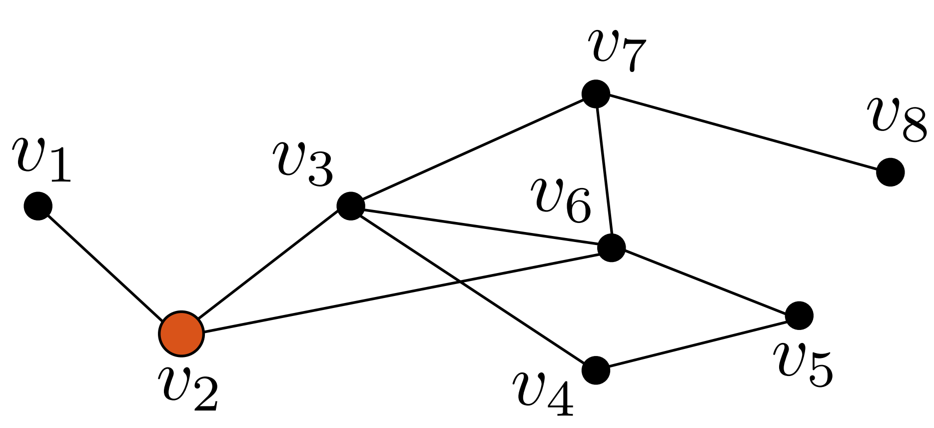
GNNs - A spatial approach

- Graph convolution can also be defined in spatial (node) domain

via a graph shift operator ⇒ $$g(S)x = \sum_{k=0}^{K} \theta_k S^k x$$

via a spatial weighted summation ⇒ $$(x * g)(v) = \sum_{v' \in \mathcal{V}} x(v')g(v, v')$$

- Common idea: nodes exchange information locally, which can be summarised into a basic form



$$h_i^{l+1} = \sigma\Big(W_{\text{self}}^l h_i^l + W_{\text{neigh}}^l \sum_{j \in \mathcal{N}_i} h_j^l\Big)$$

Isufi et al., "Graph filters for signal processing and machine learning on graphs," IEEE TSP, 2024.
Walker and Glocker, "Graph convolutional Gaussian processes," ICML, 2019.

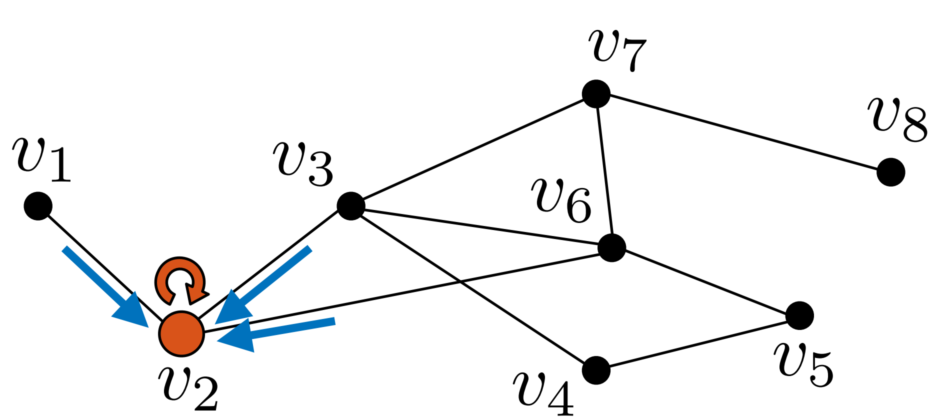
GNNs - A spatial approach

- Graph convolution can also be defined in spatial (node) domain

via a graph shift operator ➡ $g(S)x = \sum_{k=0}^{K} \theta_k S^k x$

via a spatial weighted summation ➡ $(x * g)(v) = \sum_{v' \in \mathcal{V}} x(v') g(v, v')$

- Common idea: nodes exchange information locally, which can be summarised into a basic form



self information — neighbour information

$$h_i^{l+1} = \sigma\left(W_{\text{self}}^l h_i^l + W_{\text{neigh}}^l \sum_{j \in \mathcal{N}_i} h_j^l\right)$$

Isufi et al., "Graph filters for signal processing and machine learning on graphs," IEEE TSP, 2024.
Walker and Glocker, "Graph convolutional Gaussian processes," ICML, 2019.

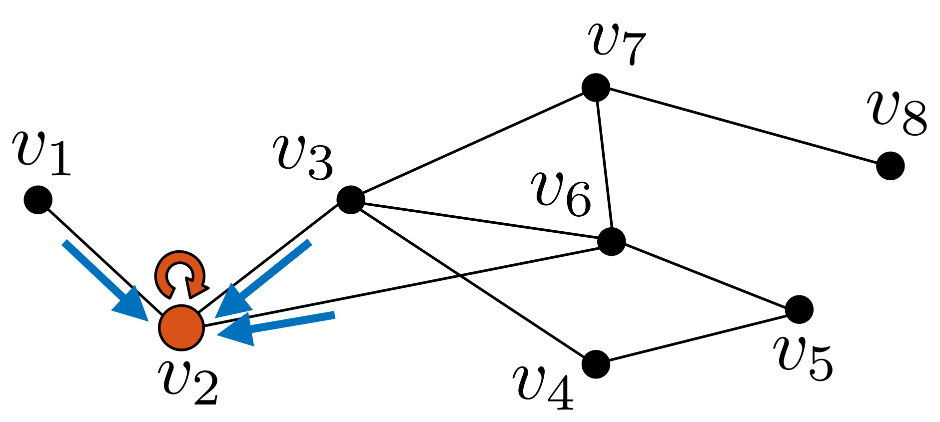
GNNs - A spatial approach

- Graph convolution can also be defined in spatial (node) domain

via a graph shift operator ➡ $g(S)x = \sum_{k=0}^{K} \theta_k S^k x$

via a spatial weighted summation ➡ $(x * g)(v) = \sum_{v' \in \mathcal{V}} x(v') g(v, v')$

- Common idea: nodes exchange information locally, which can be summarised into a basic form



self information　　neighbour information

$$h_i^{l+1} = \sigma\Big(W_{\text{self}}^l h_i^l + W_{\text{neigh}}^l \sum_{j \in \mathcal{N}_i} h_j^l\Big)$$

local neighbourhood
permutation invariant

Isufi et al., "Graph filters for signal processing and machine learning on graphs," IEEE TSP, 2024.
Walker and Glocker, "Graph convolutional Gaussian processes," ICML, 2019.

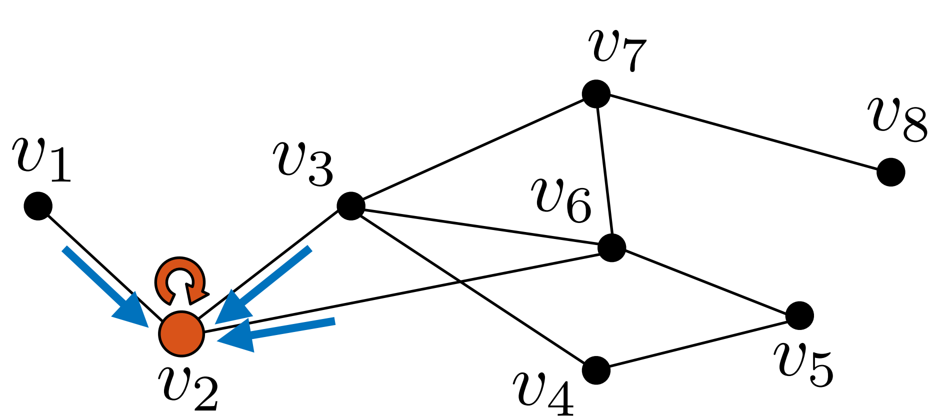
GNNs - A spatial approach

- Graph convolution can also be defined in spatial (node) domain

via a graph shift operator ➡ $$g(S)x = \sum_{k=0}^{K} \theta_k S^k x$$

via a spatial weighted summation ➡ $$(x * g)(v) = \sum_{v' \in \mathcal{V}} x(v')g(v, v')$$

- Common idea: nodes exchange information locally, which can be summarised into a basic form



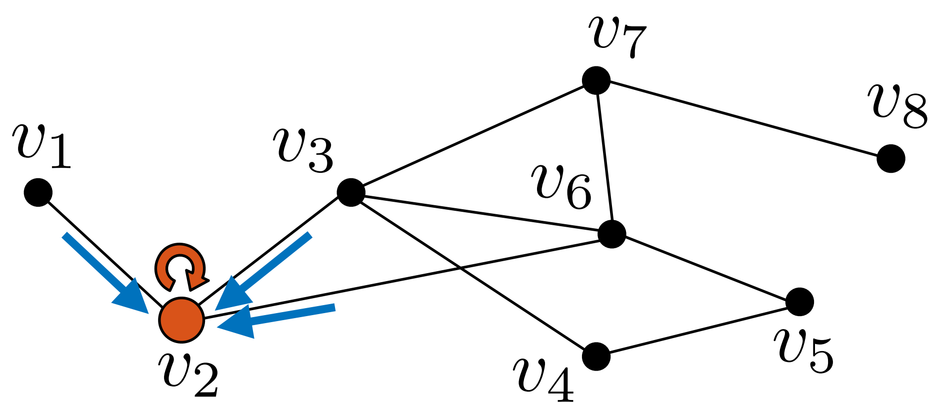
self information — neighbour information

$$h_i^{l+1} = \sigma\left(W_{\text{self}}^l h_i^l + W_{\text{neigh}}^l \sum_{j \in \mathcal{N}_i} h_j^l\right)$$

learnable parameters — local neighbourhood permutation invariant

Isufi et al., "Graph filters for signal processing and machine learning on graphs," IEEE TSP, 2024.
Walker and Glocker, "Graph convolutional Gaussian processes," ICML, 2019.

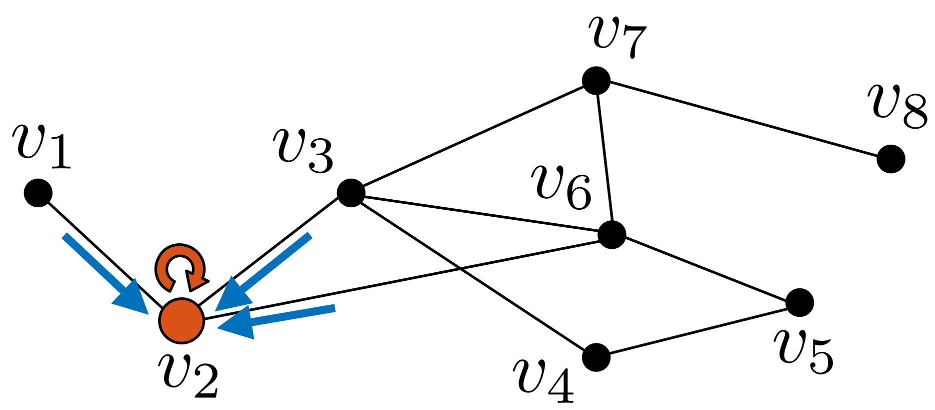
Message passing neural networks



$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

Gilmer et al., "Neural message passing for quantum chemistry," ICML, 2017.

Message passing neural networks



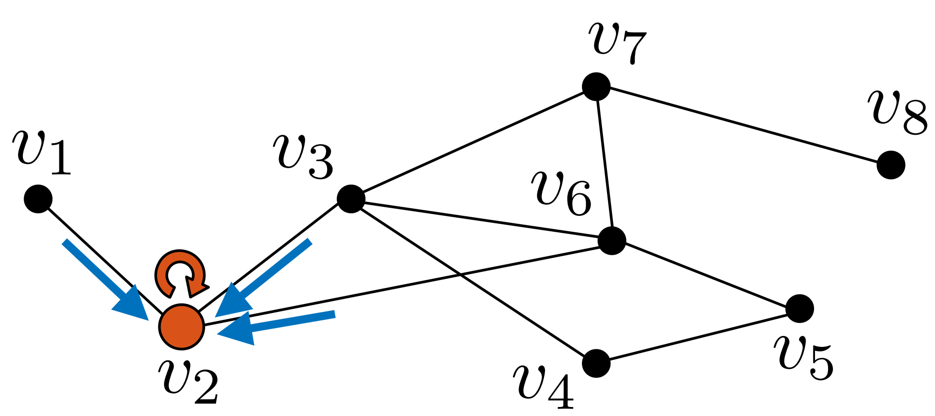
message function

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

- nodes exchange **messages** with local neighbours

Gilmer et al., "Neural message passing for quantum chemistry," ICML, 2017.

Message passing neural networks



update function

message function

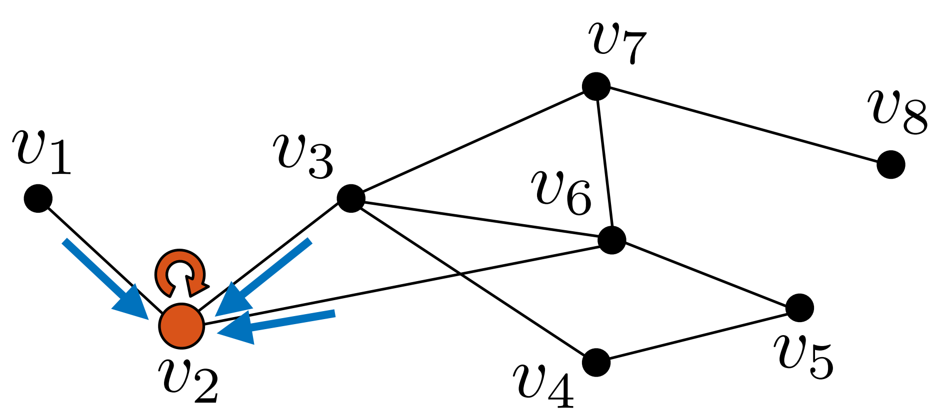
$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

aggregator function

- nodes exchange **messages** with local neighbours
- each node **aggregates** messages from its neighbours before **updating** its representation

Gilmer et al., "Neural message passing for quantum chemistry," ICML, 2017.

Message passing neural networks



update function | message function

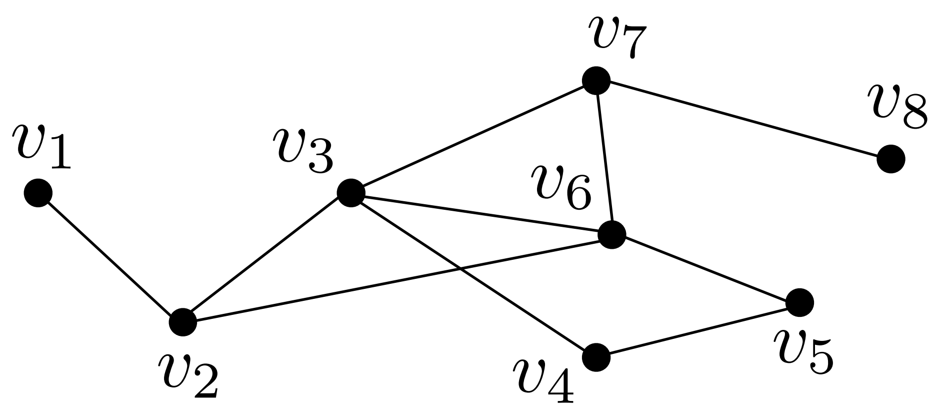
$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

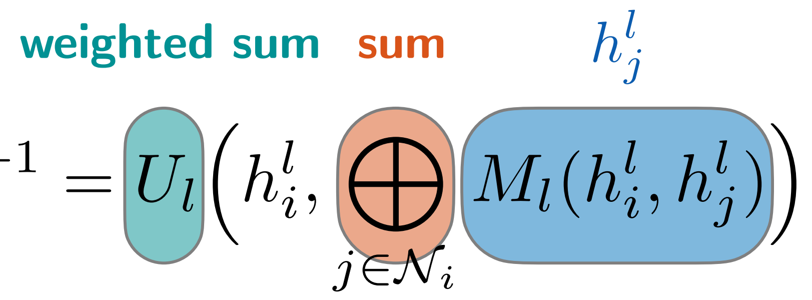
aggregator function

- nodes exchange **messages** with local neighbours
- each node **aggregates** messages from its neighbours before **updating** its representation
- functions are differentiable and parameters are learned by minimising loss of downstream task (e.g., classification)
- key difference between architectures: how nodes aggregate information from neighbours and across layers

Gilmer et al., "Neural message passing for quantum chemistry," ICML, 2017.

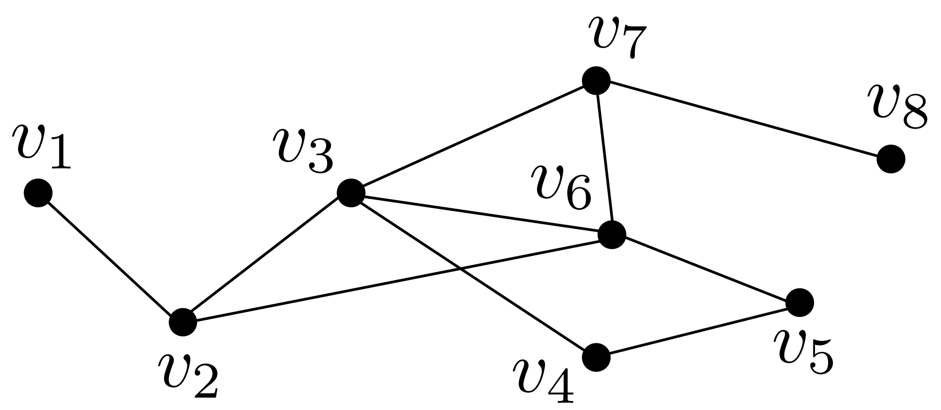
MPNNs - A simple example





$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

MPNNs - A simple example



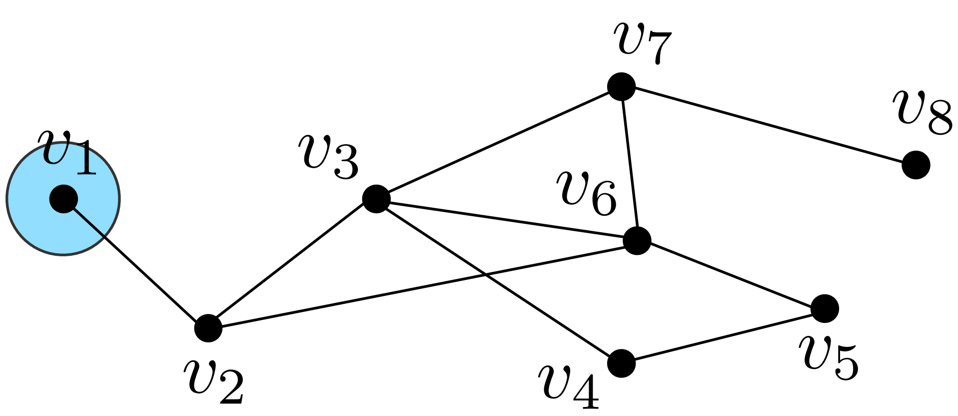
weighted sum **sum** h_j^l

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

↓

$$h_i^{l+1} = \theta_0^l h_i^l + \theta_1^l \sum_{j \in \mathcal{N}_i} h_j^l$$

MPNNs - A simple example



one message passing layer

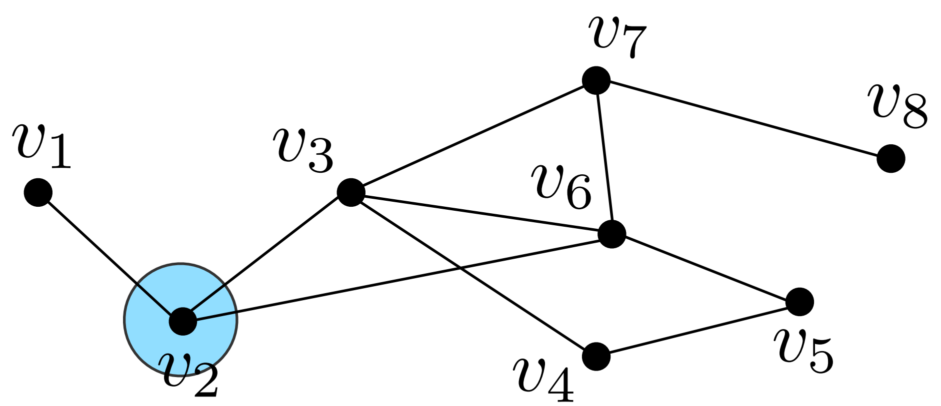
$$h_1^{l+1} = \theta_0^l h_1^l + \theta_1^l h_2^l$$

weighted sum **sum** h_j^l

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

$$h_i^{l+1} = \theta_0^l h_i^l + \theta_1^l \sum_{j \in \mathcal{N}_i} h_j^l$$

MPNNs - A simple example



one message passing layer

$$h_1^{l+1} = \theta_0^l h_1^l + \theta_1^l h_2^l$$

$$h_2^{l+1} = \theta_0^l h_2^l + \theta_1^l (h_1^l + h_3^l + h_6^l)$$

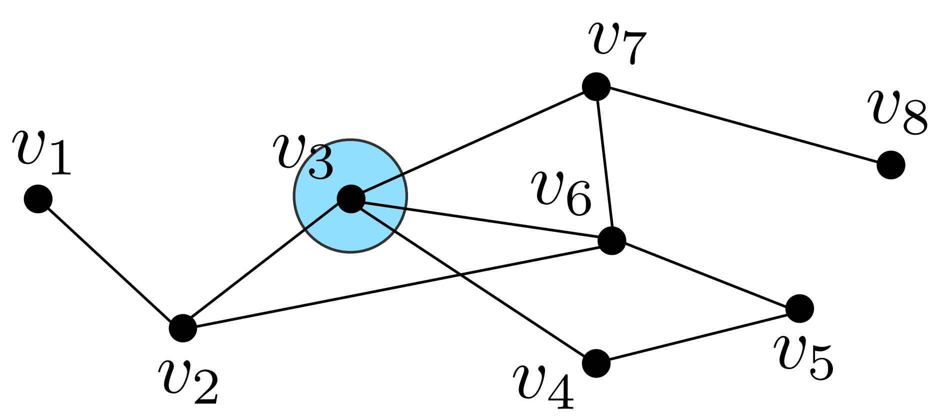
weighted sum **sum** h_j^l

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

⇓

$$h_i^{l+1} = \theta_0^l h_i^l + \theta_1^l \sum_{j \in \mathcal{N}_i} h_j^l$$

MPNNs - A simple example



one message passing layer

$$h_1^{l+1} = \theta_0^l h_1^l + \theta_1^l h_2^l$$

$$h_2^{l+1} = \theta_0^l h_2^l + \theta_1^l (h_1^l + h_3^l + h_6^l)$$

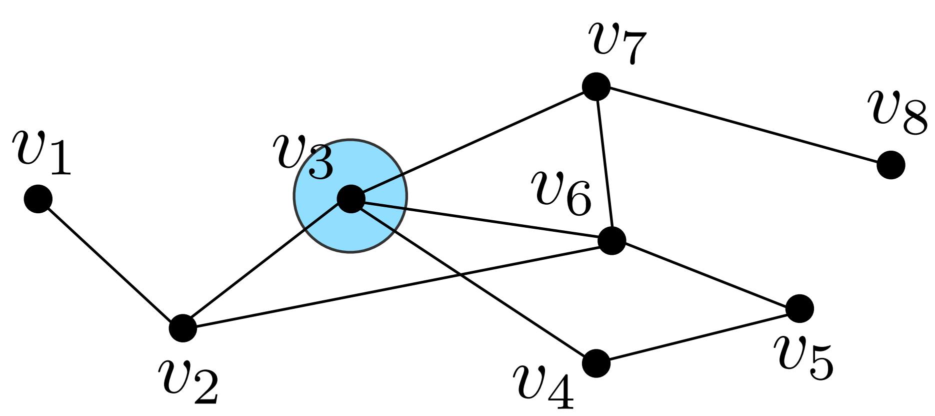
$$h_3^{l+1} = \theta_0^l h_3^l + \theta_1^l (h_2^l + h_4^l + h_6^l + h_7^l)$$

weighted sum **sum** h_j^l

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

$$h_i^{l+1} = \theta_0^l h_i^l + \theta_1^l \sum_{j \in \mathcal{N}_i} h_j^l$$

MPNNs - A simple example



weighted sum sum h_j^l

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

$$h_i^{l+1} = \theta_0^l h_i^l + \theta_1^l \sum_{j \in \mathcal{N}_i} h_j^l$$

one message passing layer

$$h_1^{l+1} = \theta_0^l h_1^l + \theta_1^l h_2^l$$

$$h_2^{l+1} = \theta_0^l h_2^l + \theta_1^l (h_1^l + h_3^l + h_6^l)$$

$$h_3^{l+1} = \theta_0^l h_3^l + \theta_1^l (h_2^l + h_4^l + h_6^l + h_7^l)$$

$$h_4^{l+1} = \theta_0^l h_4^l + \theta_1^l (h_3^l + h_5^l)$$

$$h_5^{l+1} = \theta_0^l h_5^l + \theta_1^l (h_4^l + h_6^l)$$

$$h_6^{l+1} = \theta_0^l h_6^l + \theta_1^l (h_2^l + h_3^l + h_5^l + h_7^l)$$

$$h_7^{l+1} = \theta_0^l h_7^l + \theta_1^l (h_3^l + h_6^l + h_8^l)$$

$$h_8^{l+1} = \theta_0^l h_8^l + \theta_1^l h_7^l$$

final output after L layers

$$Z = \{h_1^L, h_2^L, h_3^L, h_4^L, h_5^L, h_6^L, h_7^L, h_8^L\}^T$$

Graph isomorphism network

- Motivation: introduce a way of defining **expressive power** of MPNNs
  - e.g., for graph classification, what kinds of graphs can they distinguish?

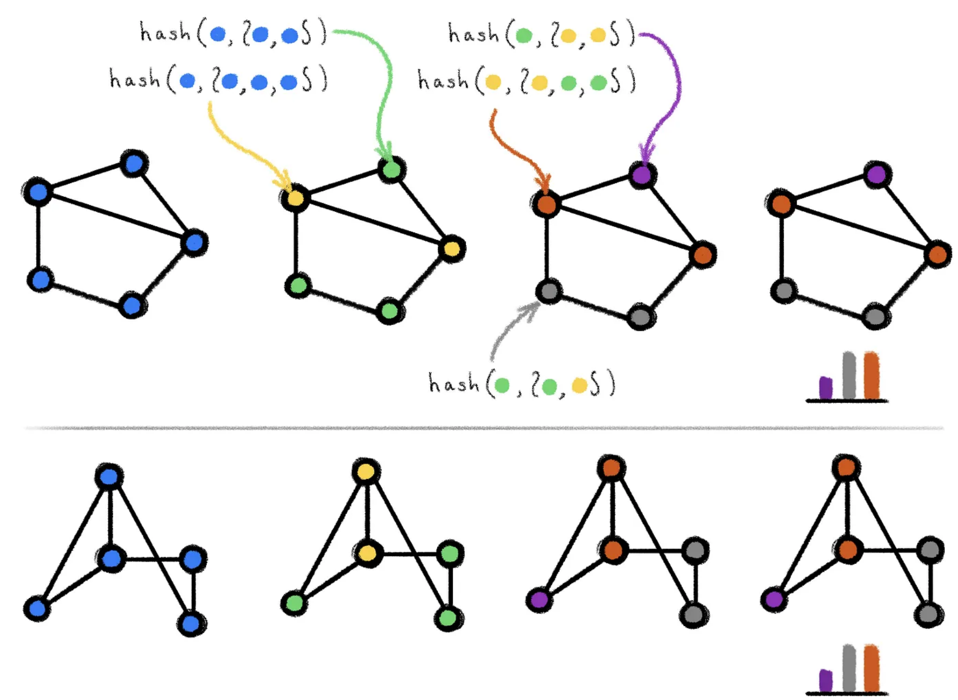
Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph isomorphism network

- Motivation: introduce a way of defining **expressive power** of MPNNs
  - e.g., for graph classification, what kinds of graphs can they distinguish?
- Inspiration from the Weisfeiler-Lehman test for graph isomorphism (whether two graphs are isomorphic)

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as
  $$c_i^{l+1} = \mathrm{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$
- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$



potentially isomorphic graphs (necessary but not sufficient) [source: M. Bronstein]

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as
  $$c_i^{l+1} = \text{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$
- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as
  $$h_i^{l+1} = U_l(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l))$$
- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

[source: D. Thanou]

Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as
  $$c_i^{l+1} = \mathrm{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$
- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as
  $$h_i^{l+1} = U_l(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l))$$
- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

[source: D. Thanou]

Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as
  $$c_i^{l+1} = \mathrm{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$
- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as
  $$h_i^{l+1} = U_l(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l))$$
- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

[source: D. Thanou]

Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as
  $$c_i^{l+1} = \mathrm{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$
- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as
  $$h_i^{l+1} = U_l(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l))$$
- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

[source: D. Thanou]

Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update
- Can we design an MPNN that is as powerful as the WL test?
  - more than graph isomorphism we can use it for graph classification or regression
  - hash func in WL is injective $\Rightarrow$ injective aggregator/update func in MPNNs?

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as

$$c_i^{l+1} = \mathrm{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$

- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as

$$h_i^{l+1} = U_l(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l))$$

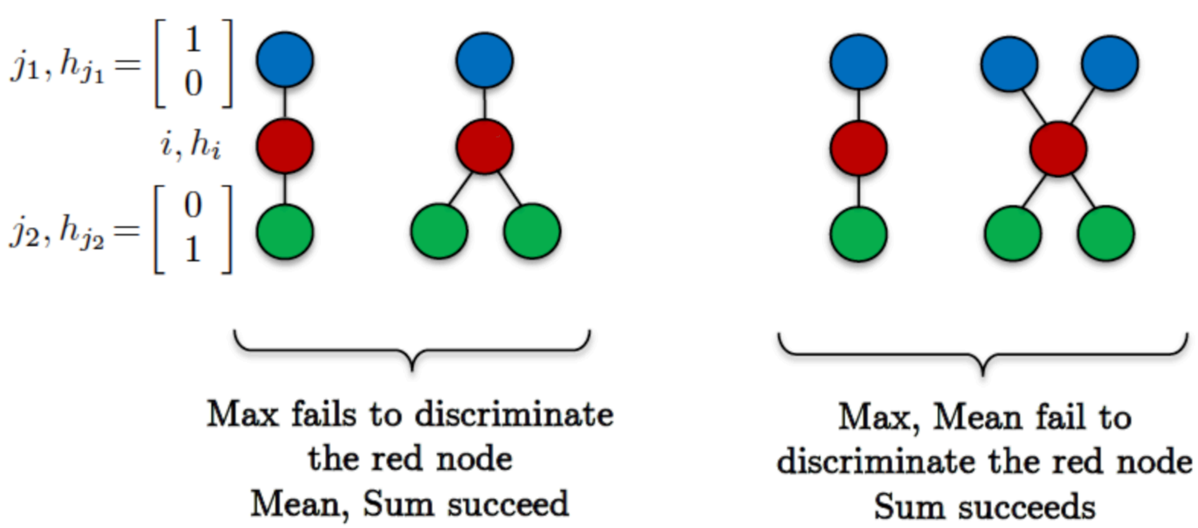
- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

[source: D. Thanou]

Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph isomorphism network

- What aggregator function is injective?
  - expressive power on a multiset: sum > mean > max-pooling



[source: X. Bresson]

Graph isomorphism network

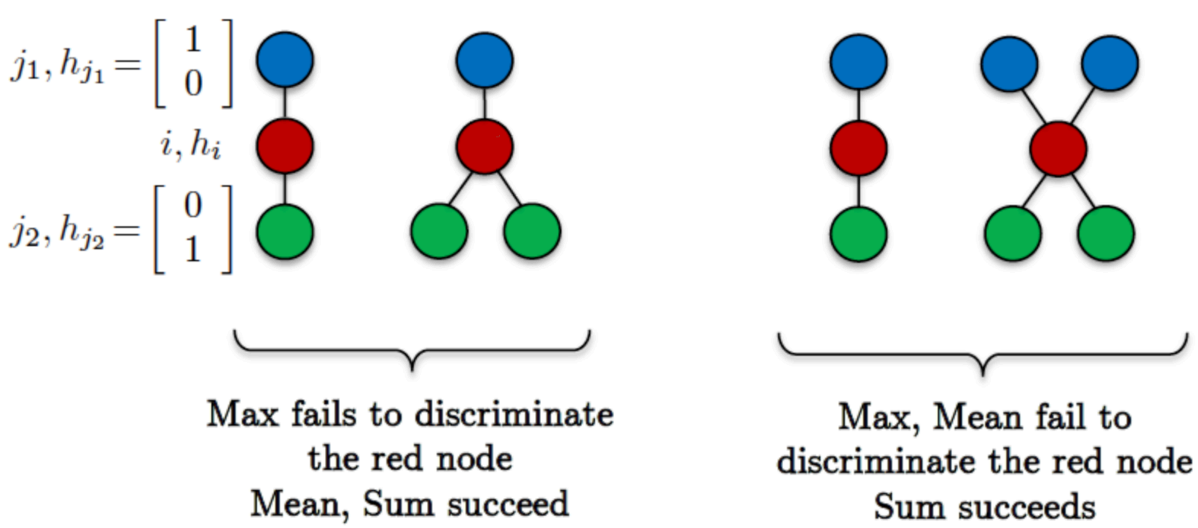
- What aggregator function is injective?
  - expressive power on a multiset: sum > mean > max-pooling
- GIN is provably as powerful as 1-WL test (under certain conditions)
  - aggregator & update: sum + MLP

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

sum aggregator

$$h_i^{l+1} = \mathrm{MLP}^l\Big((1 + \epsilon^l)h_i^l + \sum_{j \in \mathcal{N}_i} h_j^l\Big)$$

learnable parameters



[source: X. Bresson]

Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph isomorphism network

- What aggregator function is injective?
  - expressive power on a multiset: sum > mean > max-pooling
- GIN is provably as powerful as 1-WL test (under certain conditions)
  - aggregator & update: sum + MLP
  - global readout: sum + MLP

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

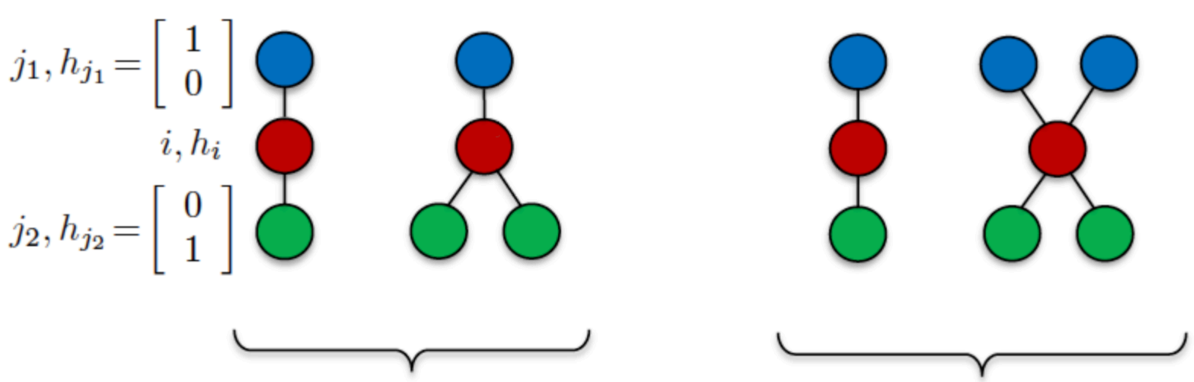
sum aggregator

$$h_i^{l+1} = \mathrm{MLP}^l\Big((1+\epsilon^l)h_i^l + \sum_{j \in \mathcal{N}_i} h_j^l\Big)$$

learnable parameters

$$h_{\mathcal{G}} = \mathrm{MLP}\Big(\sum_{i \in \mathcal{V}} h_i^L\Big)$$

sum aggregator



Max fails to discriminate the red node
Mean, Sum succeed

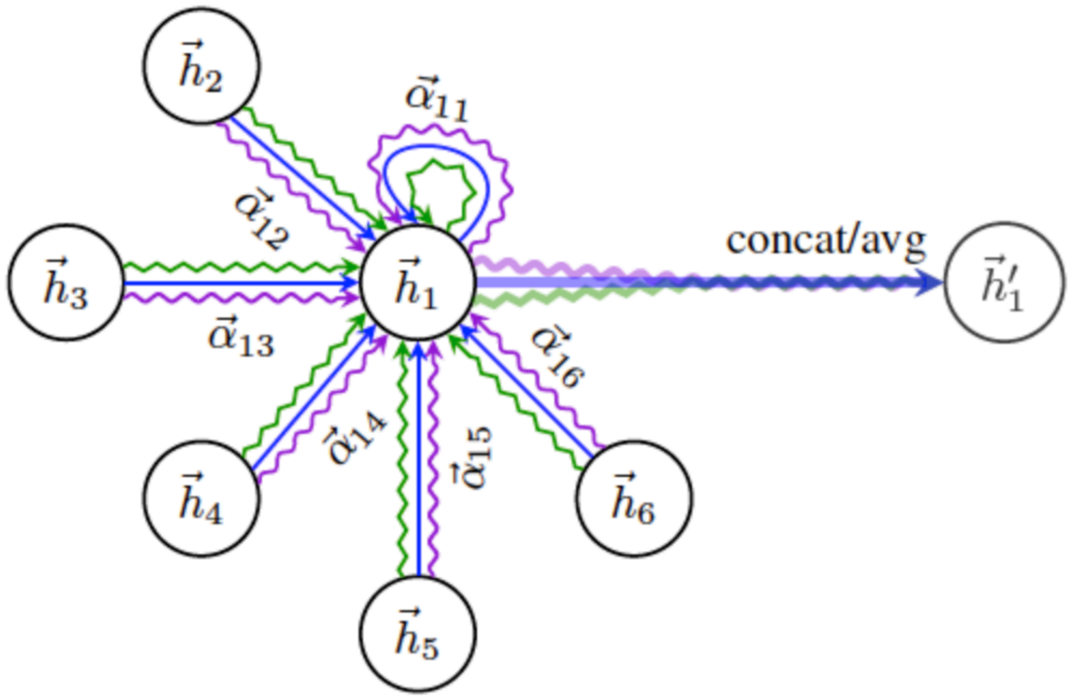
Max, Mean fail to discriminate the red node
Sum succeeds

[source: X. Bresson]

Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph attention network

- Idea: learn **relative importance** of neighbours in aggregation



$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

relative importance

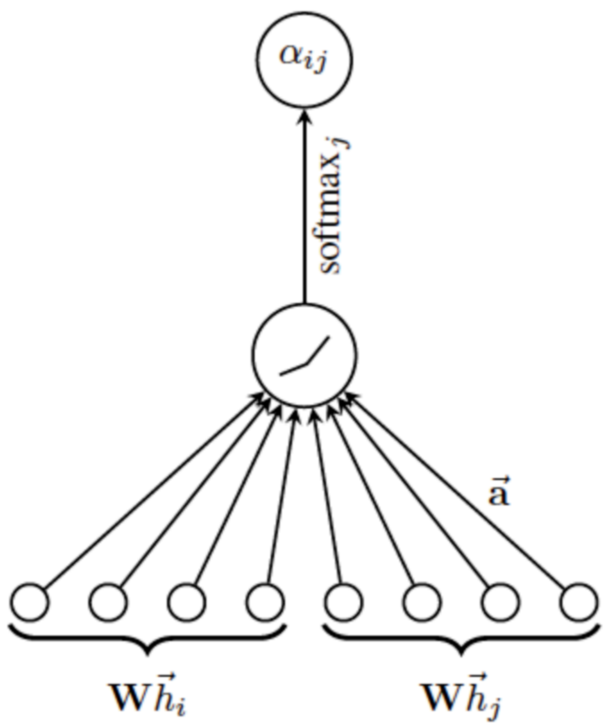
$$h_i^{l+1} = \sigma\Big(\alpha_{ii} W^l h_i^l + \sum_{j \in \mathcal{N}_i} \alpha_{ij} W^l h_j^l\Big)$$

learnable parameters

Veličković et al., "Graph attention networks," ICLR, 2018.

Graph attention network

- Idea: learn **relative importance** of neighbours in aggregation
- An **attention** function compares importance of neighbours and computes attention scores



$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

relative importance

$$h_i^{l+1} = \sigma\Big(\alpha_{ii} W^l h_i^l + \sum_{j \in \mathcal{N}_i} \alpha_{ij} W^l h_j^l\Big)$$

learnable parameters

$$\alpha_{ij} = \alpha(W^l h_i^l, W^l h_j^l)$$

attention function

Veličković et al., "Graph attention networks," ICLR, 2018.

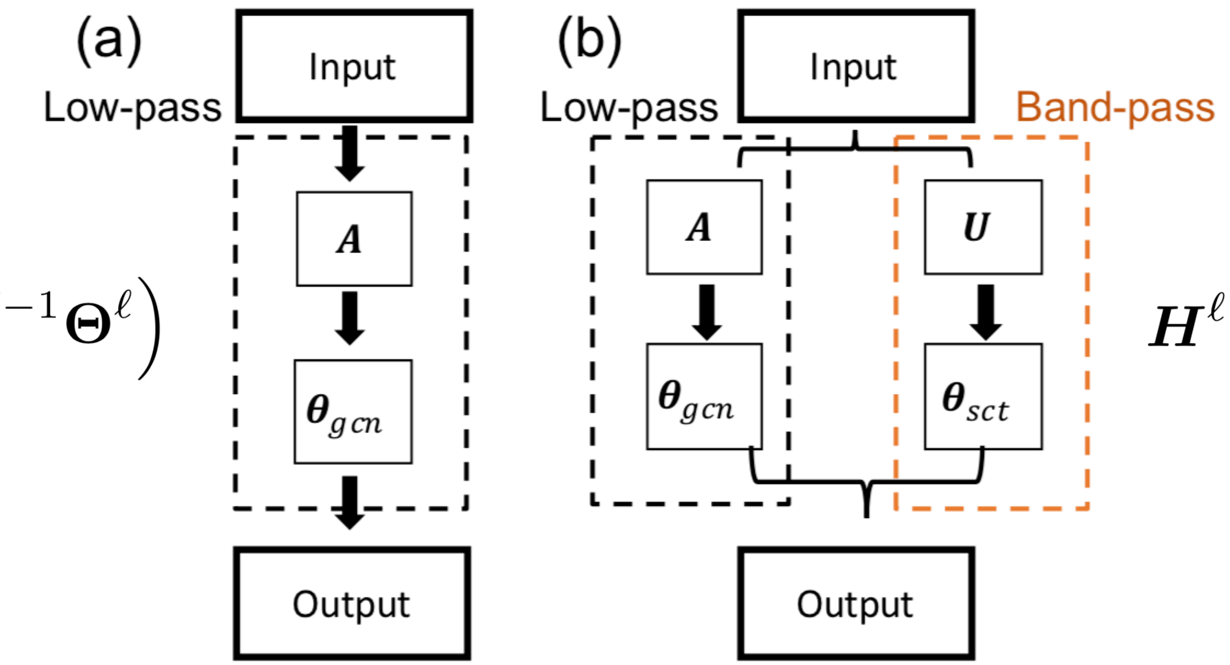
Graph Machine Learning

- Overview of graph machine learning
- Convolutional neural networks on graphs
  - "spectral" approaches enabled by graph signal processing
- State-of-the-art graph neural networks
  - "spatial" approaches enabled by message passing
- Latest developments and applications

Extension I: Beyond low-pass filtering

- GCN does low-pass filtering (may lead to "over-smoothing")

$$\boldsymbol{H}^{\ell} = \sigma\left(\boldsymbol{A}\boldsymbol{H}^{\ell-1}\boldsymbol{\Theta}^{\ell}\right)$$



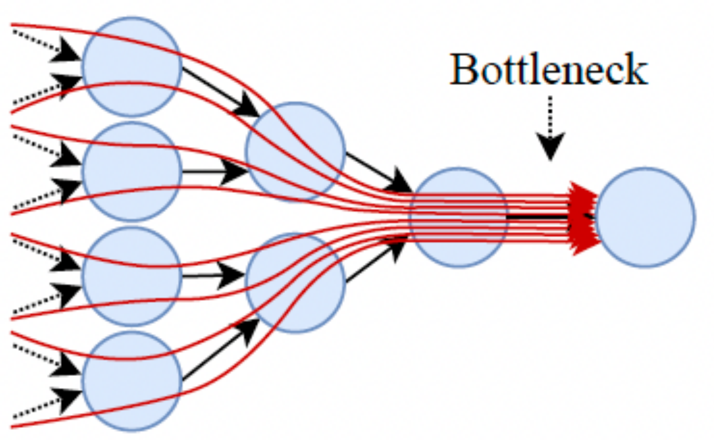
$$\boldsymbol{H}^{\ell} = \sigma\left(\boldsymbol{U}_p\boldsymbol{H}^{\ell-1}\boldsymbol{\Theta}^{\ell}\right)$$

GCN **Scattering GCN**

Min et al., "Scattering GCN: Overcoming oversmoothness in graph convolutional networks," NeurIPS, 2020.
Bo et al., "Beyond low-frequency information in graph convolutional networks," AAAI, 2021.

Extension II: Graph rewiring

- Input graph may not be ideal for message passing (e.g., "over-squashing")



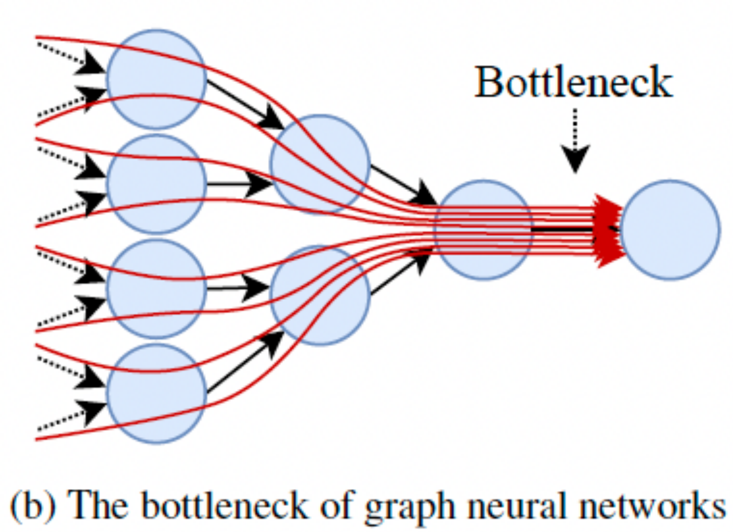
(b) The bottleneck of graph neural networks

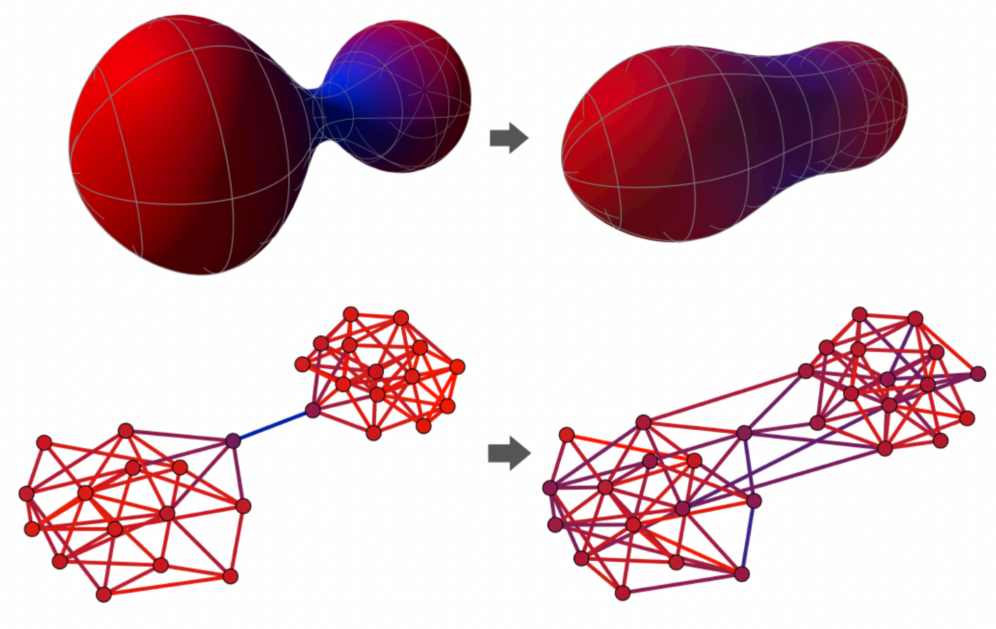
over-squashing caused by **bottlenecks**

Alon and Yahav, "On the bottleneck of graph neural networks and its practical implications," ICLR, 2021.
Topping et al., "Understanding over-squashing and bottlenecks on graphs via curvature," ICLR, 2022.

Extension II: Graph rewiring

- Input graph may not be ideal for message passing (e.g., "over-squashing")
- "Rewiring" as pre-processing step to mitigate over-squashing





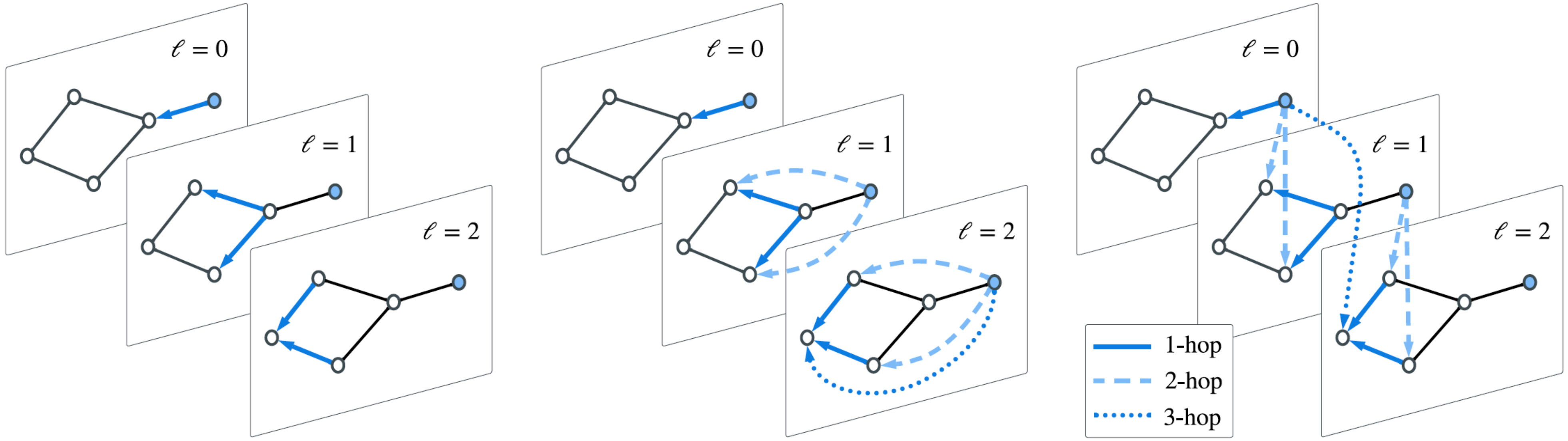
over-squashing caused by **bottlenecks**

bottlenecks are linked to **negatively curved** edges

Alon and Yahav, "On the bottleneck of graph neural networks and its practical implications," ICLR, 2021.
Topping et al., "Understanding over-squashing and bottlenecks on graphs via curvature," ICLR, 2022.

Extension III: Dynamic message passing

- Modifying the "computational" graph for improved message passing

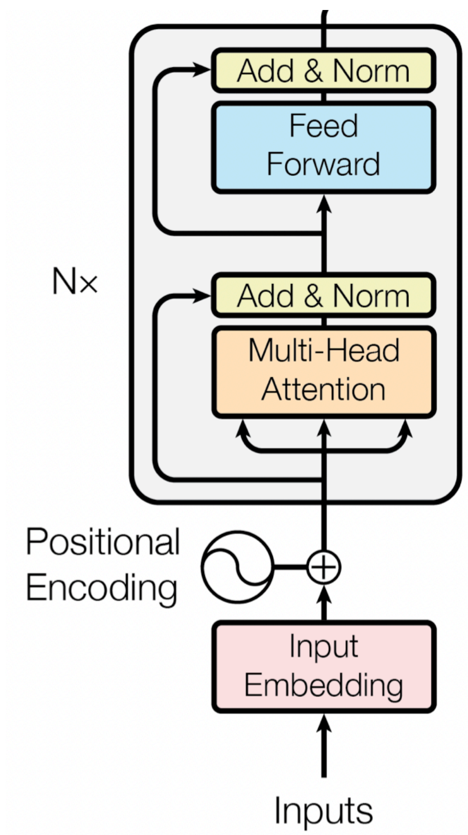


(a) Classical MPNN (b) DRew (c) νDRew

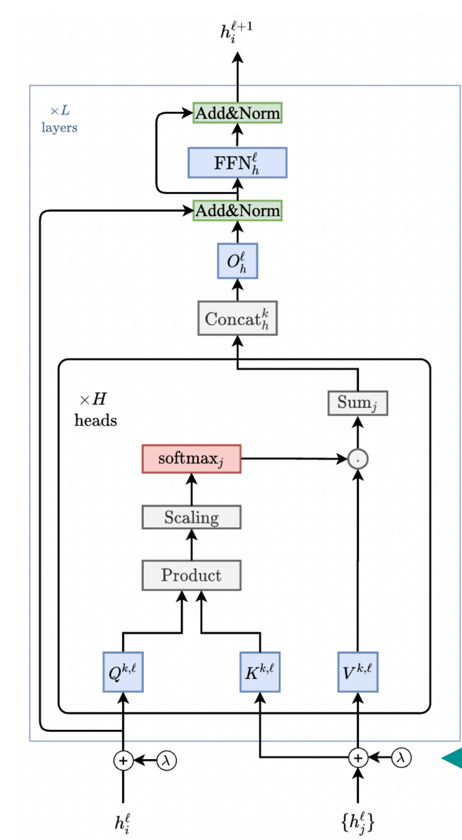
Gutteridge et al., "DRew: Dynamically rewired message passing with delay," ICML, 2023.

Extension IV: Graph transformers

- Generalise GAT to global attention
- Capture long-range dependencies (but computationally expensive)



transformers



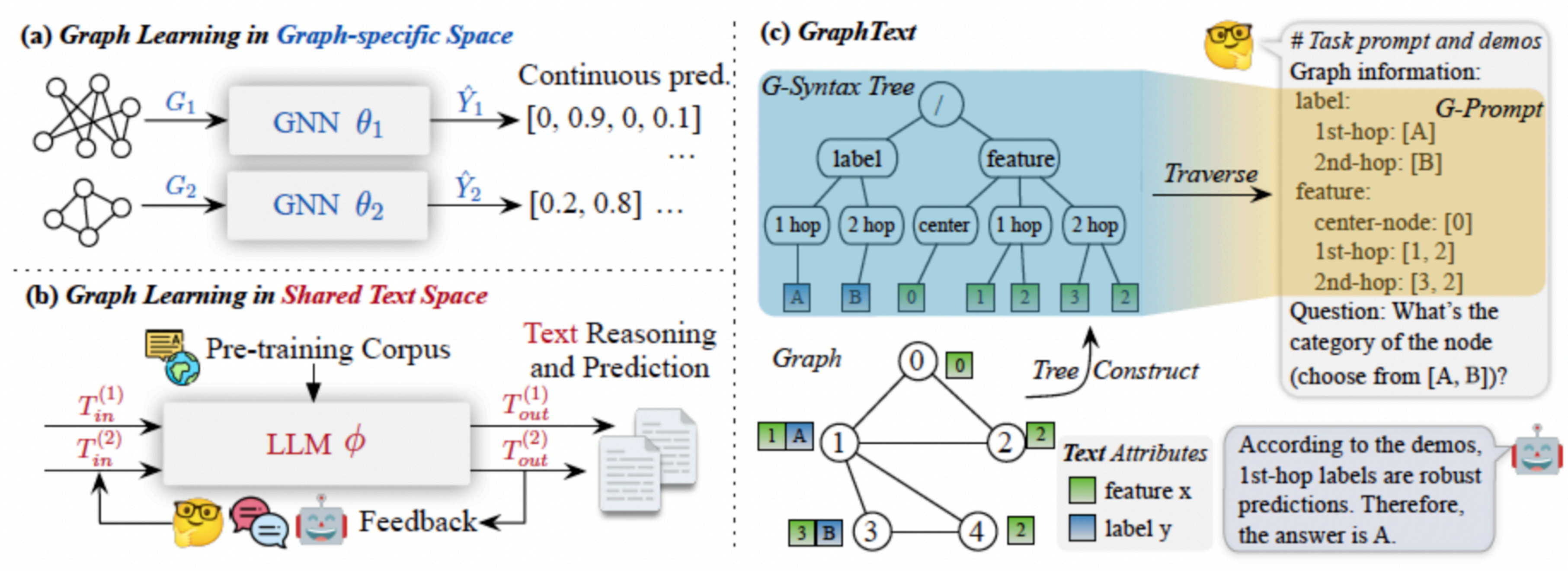
PEs retaining graph info

graph transformers

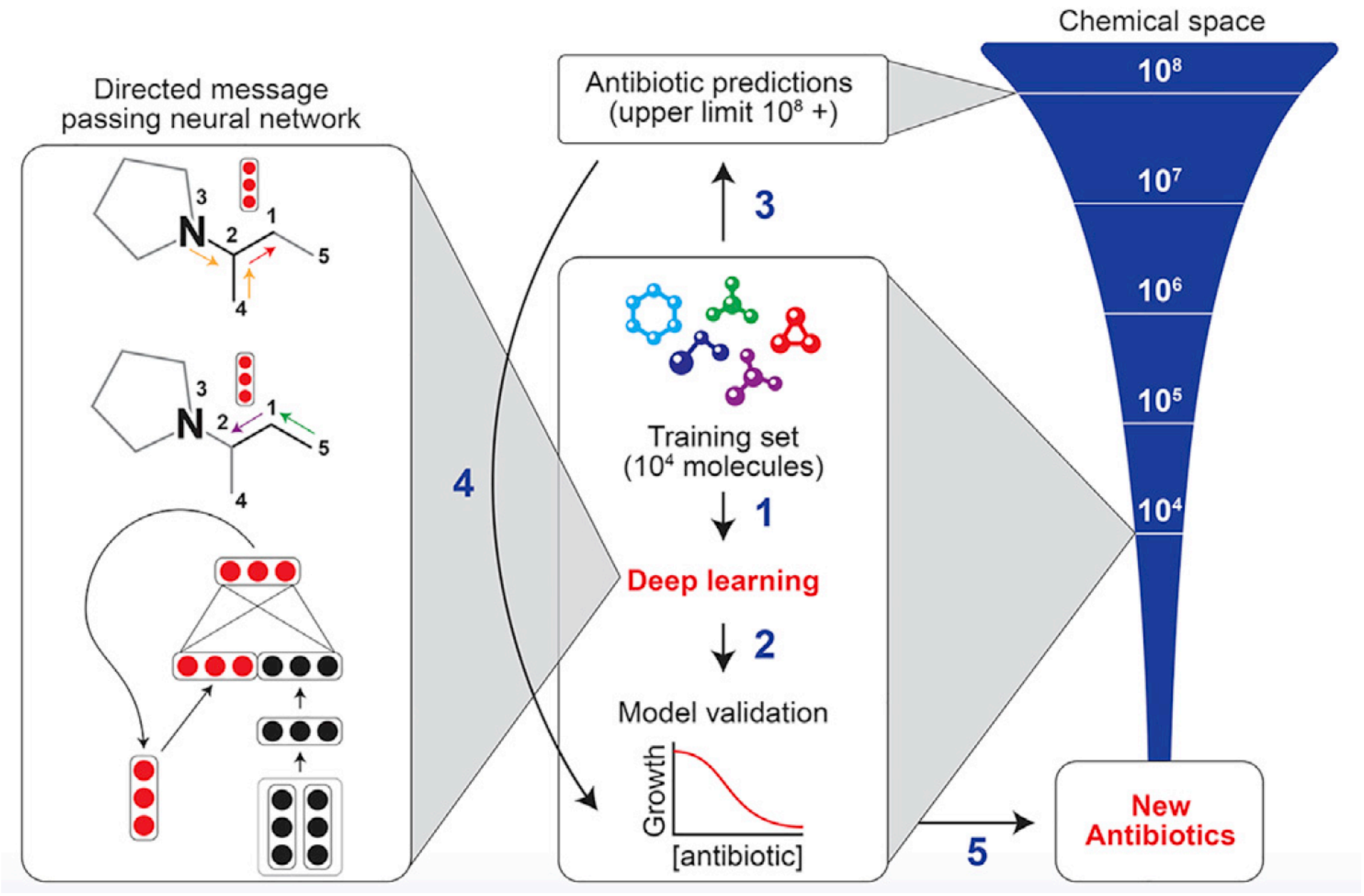
Dwivedi and Bresson, "A generalization of transformer networks to graphs," AAAI Workshop, 2021.

Extension V: LLMs for graph learning

- Can LLMs/foundation models understand graph-structured data?

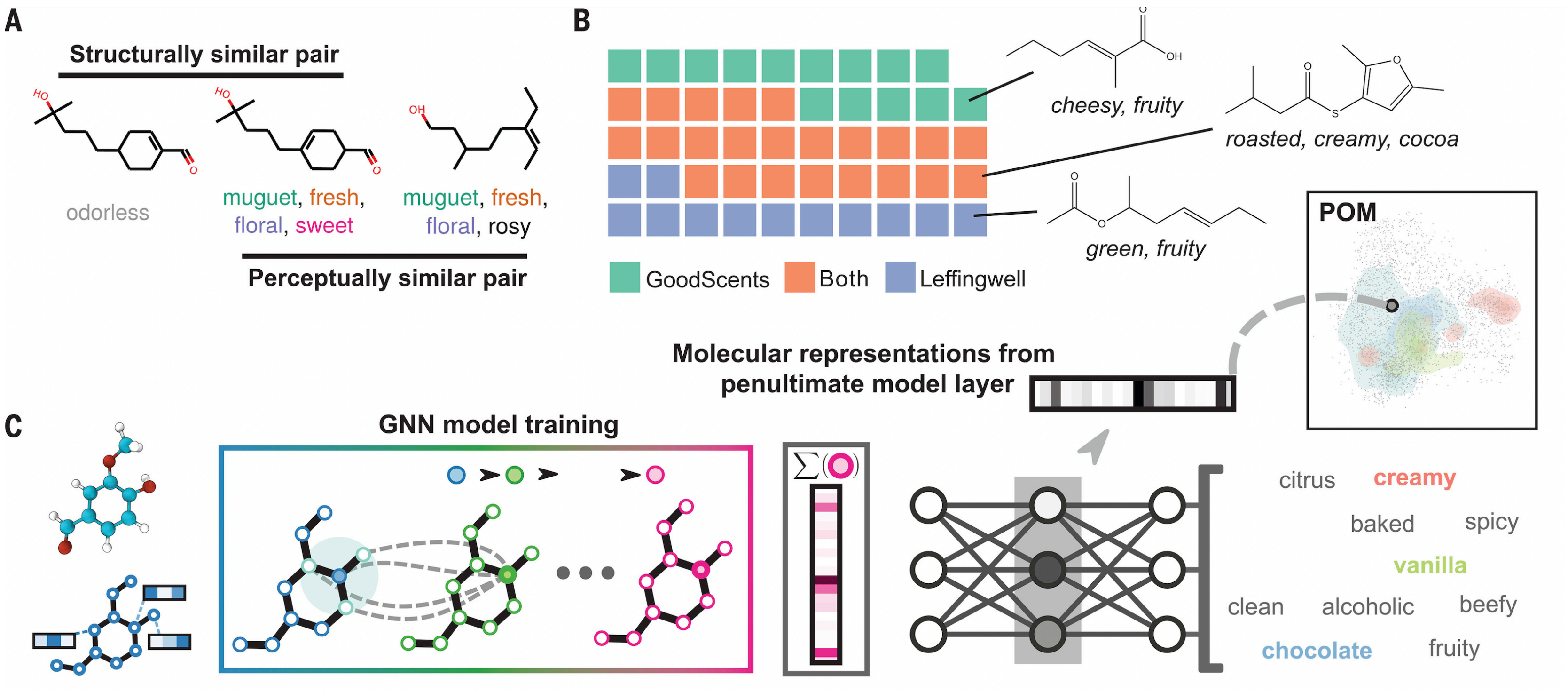


Application I: Drug discovery



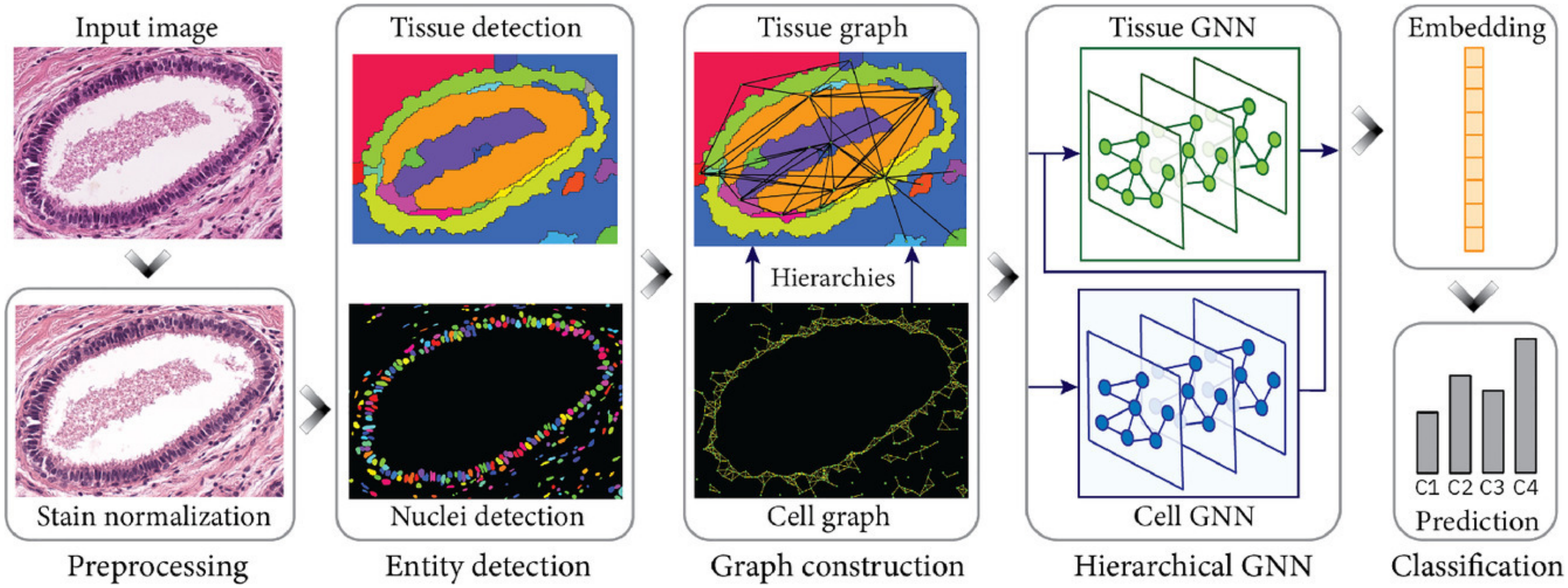
Stokes et al., "A deep learning approach to antibiotic discovery," Cell, 2020.

Application II: Odour perception



Lee et al., "A principal odor map unifies diverse tasks in olfactory perception," Science, 2023.

Application III: Medical imaging



Pati et al., "Hierarchical graph representations in digital pathology," Medical Image Analysis, 2022.

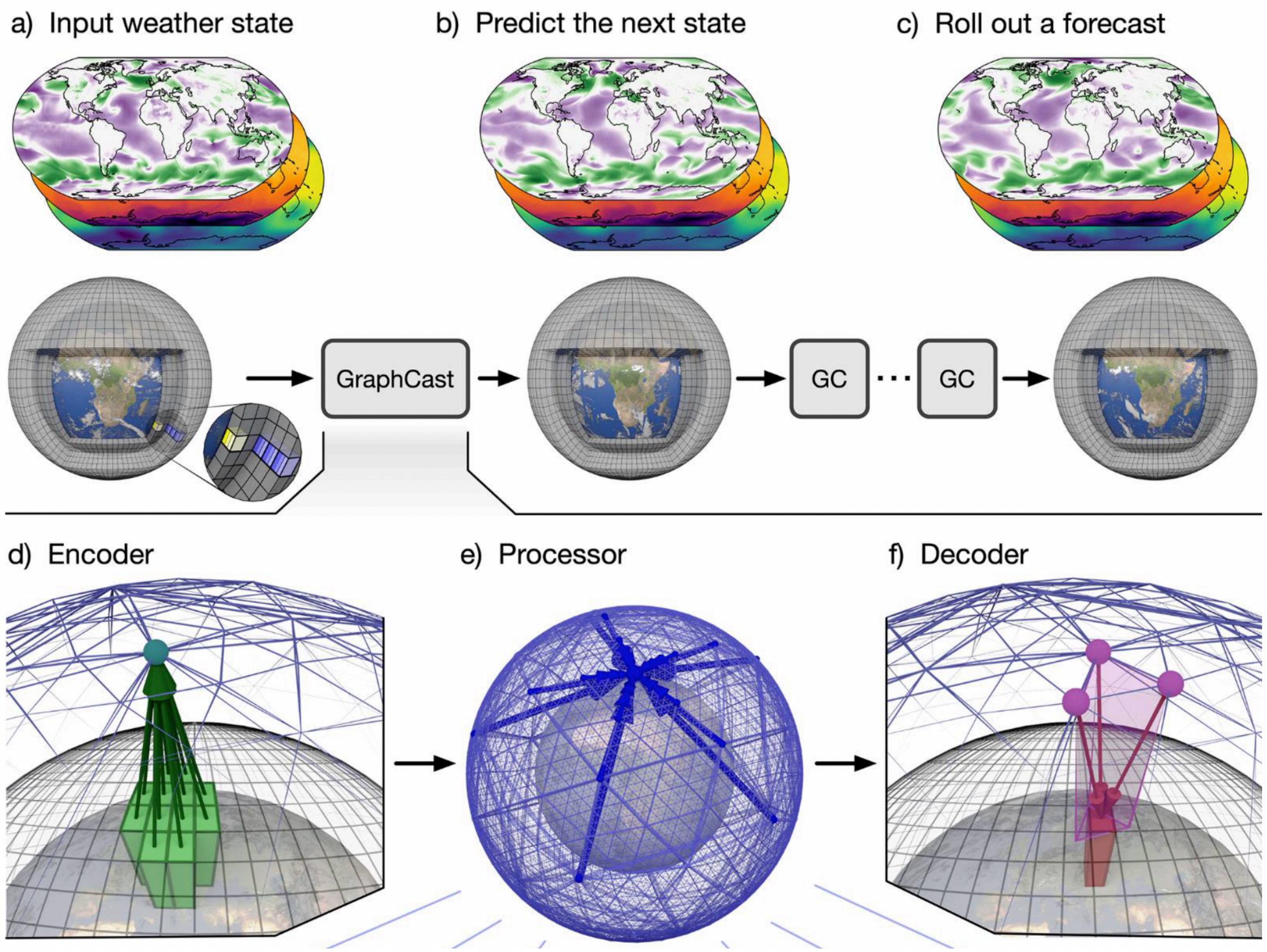
Application IV: Traffic prediction

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

Application IV: Traffic prediction

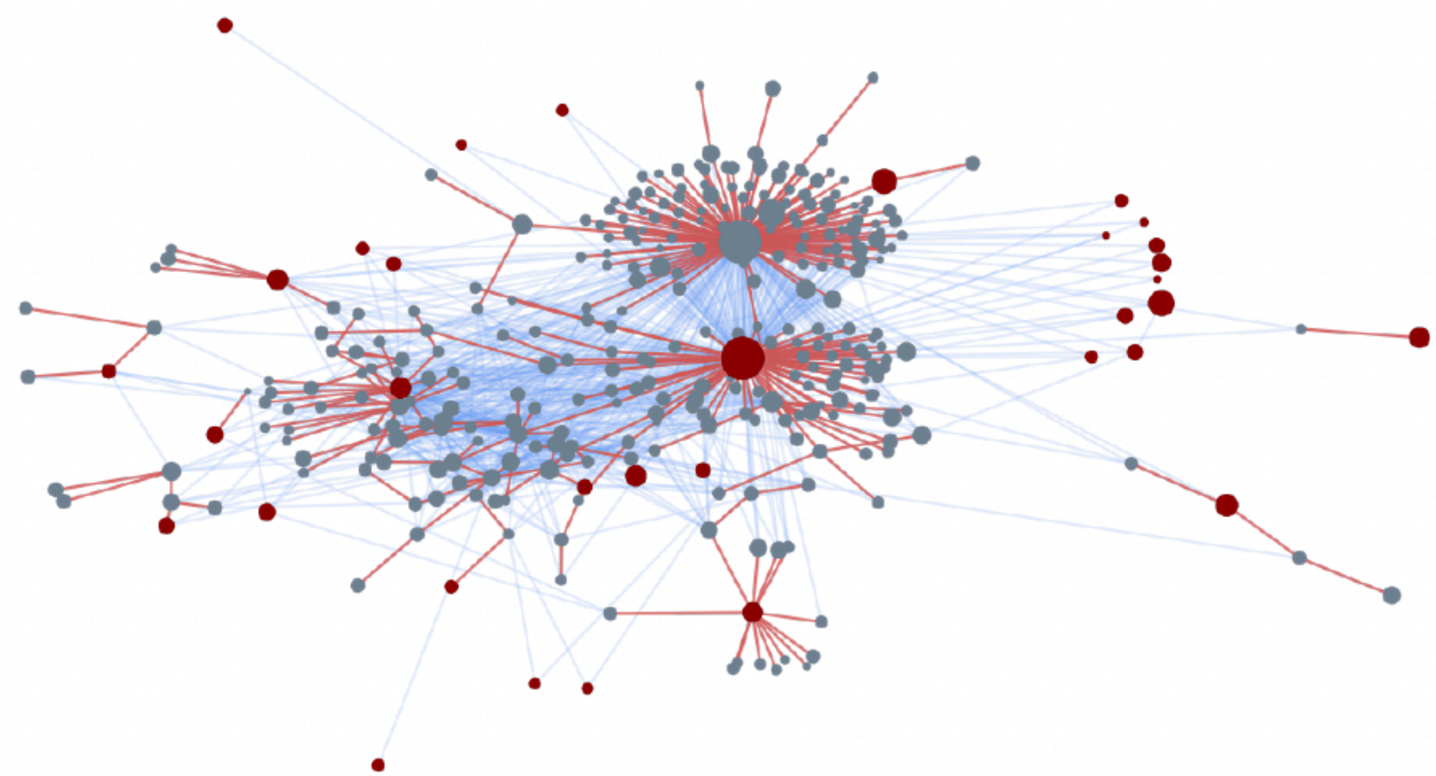
Derrow-Pinion et al., "ETA prediction with graph neural networks in Google Maps," CIKM, 2021.

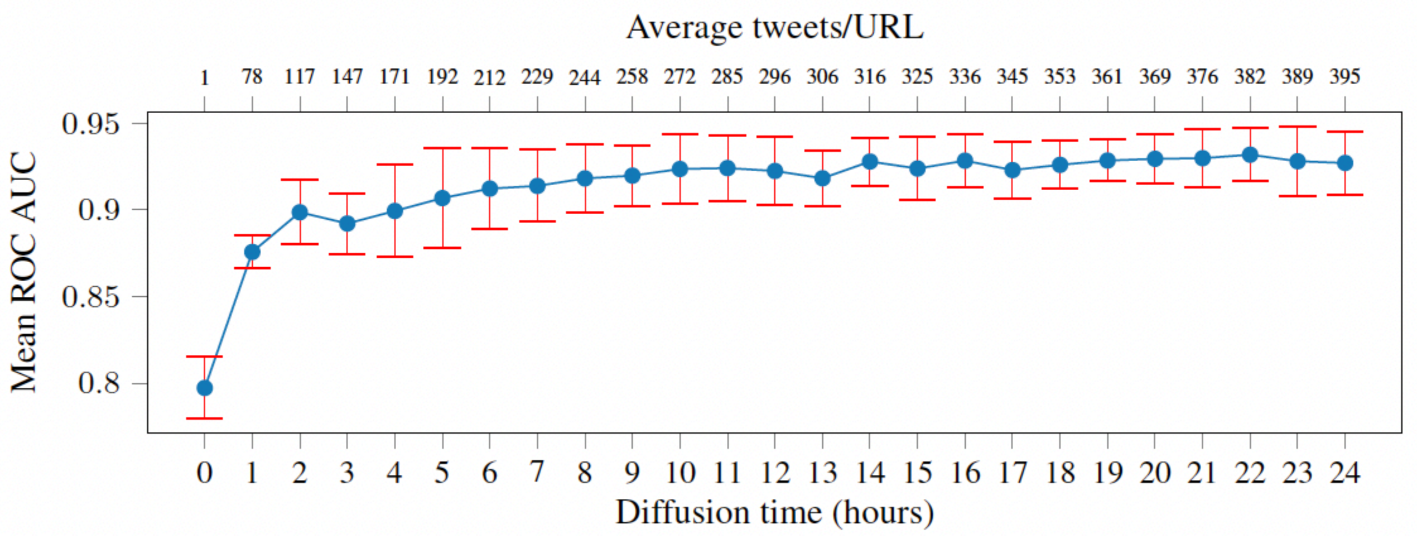
Application V: Weather forecasting



Lam et al., "Learning skillful medium-range global weather forecasting," Science, 2023.

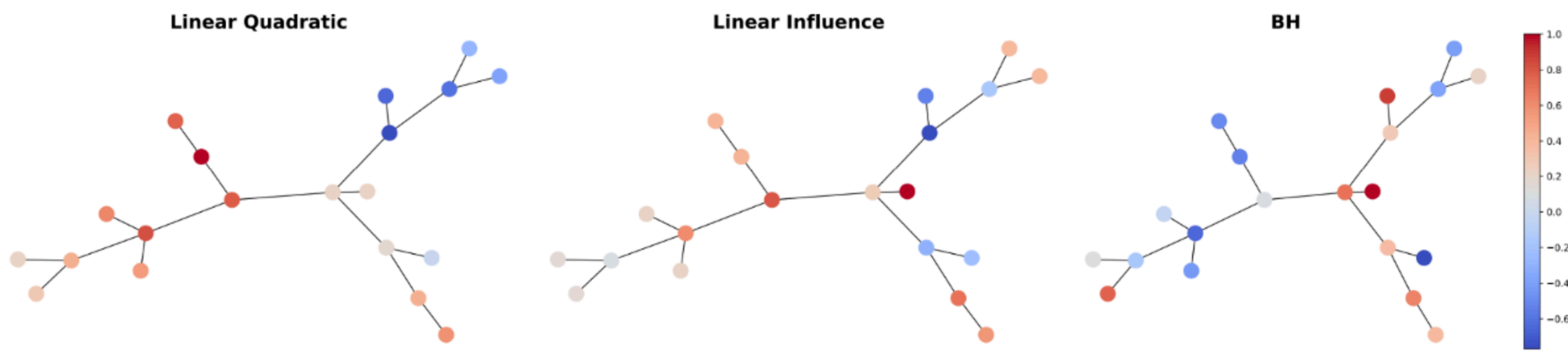
Application VI: Fake news detection



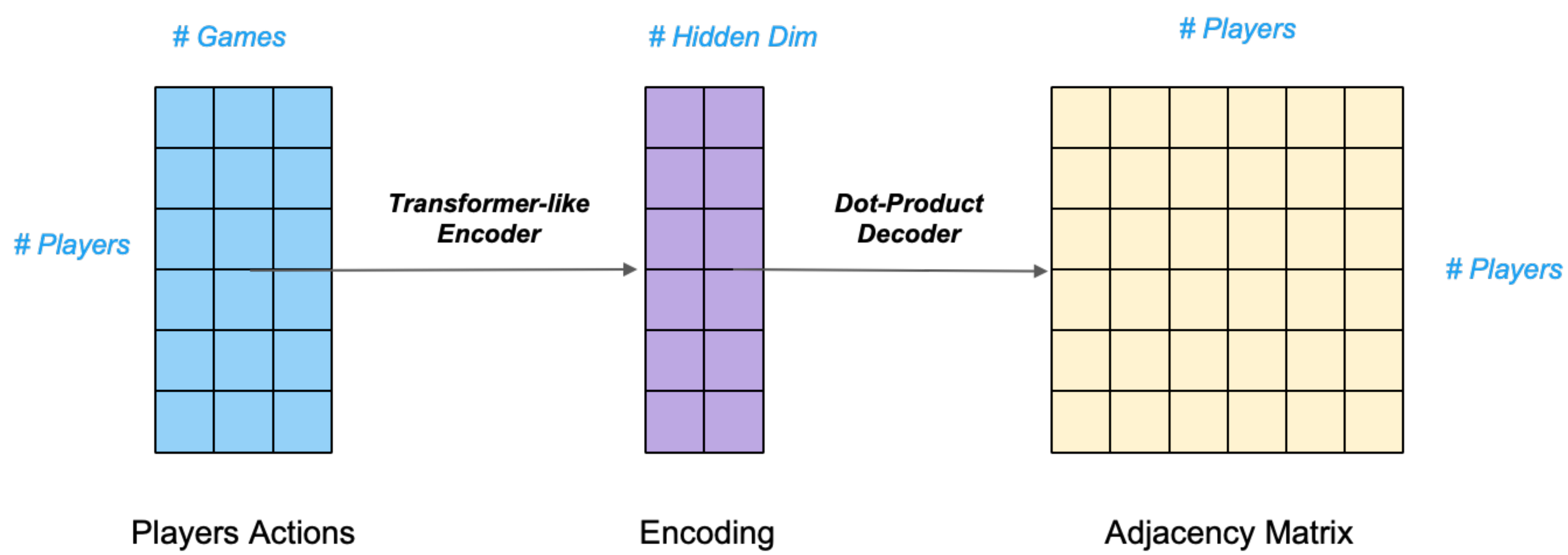


Monti et al., "Fake news detection on social media using geometric deep learning," ICLR Workshop, 2019.

Application VII: Social interactions

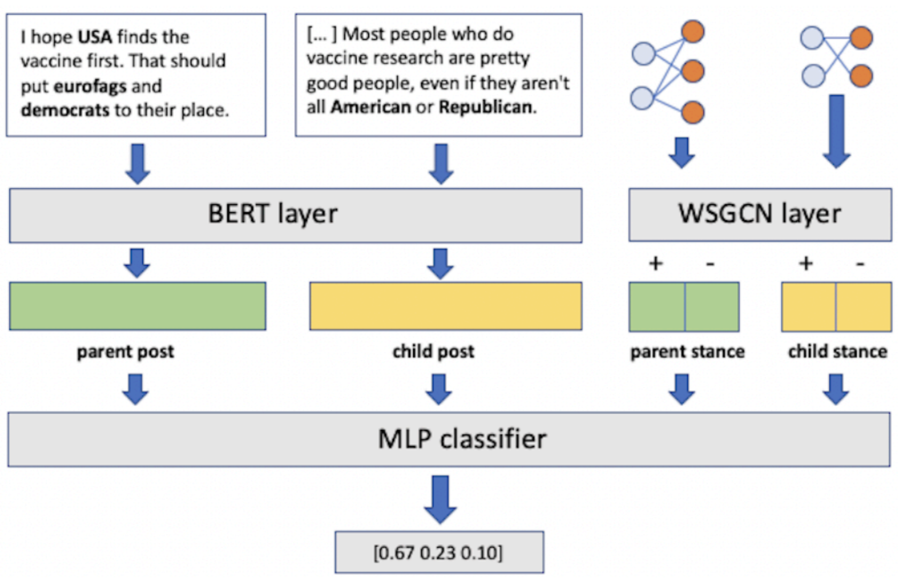


$\mathbf{x}^* = (\mathbf{I} - \beta\mathbf{A})^{-1}\mathbf{b}$ $\quad$ $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$. $\quad$ $\mathbf{x}^* = \mathbf{A}\mathbf{x}^*$

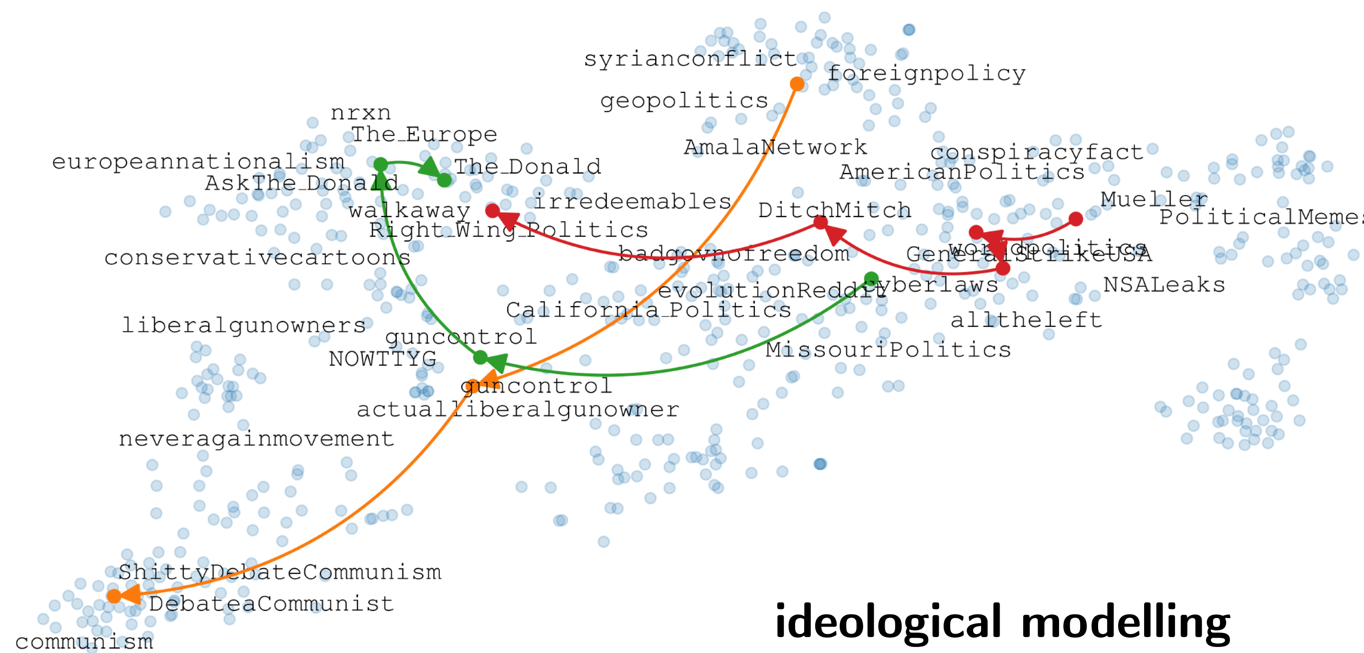


Rossi et al., "Learning to infer structures of network games," ICML, 2022.

Application VIII: Language modelling



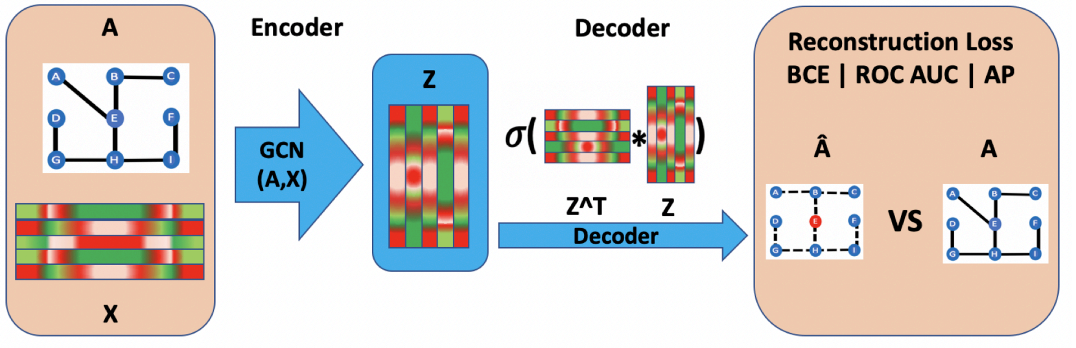
disagreement prediction

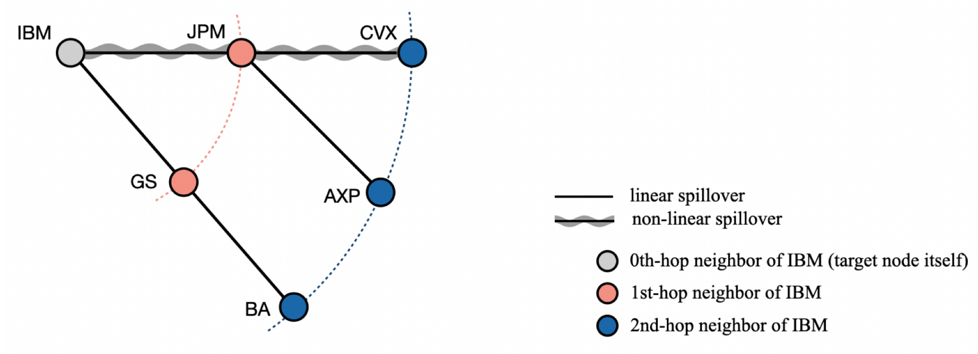


ideological modelling

Lorge et al., "STEntConv: Predicting disagreement between reddit users with stance detection and a signed graph convolutional network," LREC-Coling 2024.
Hofmann et al., "Modeling ideological salience and framing in polarized online groups with graph neural networks and structured sparsity," NAACL, 2022.

Application IV: Stock market analysis





market instability **volatility forecasting**

Gorduza et al., "Understanding stock market instability via graph auto-encoders," arXiv, 2022.
Zhang et al., "Forecasting realized volatility with spillover effects," International Journal of Forecasting, 2024.

Graph machine learning - Closing remarks

- Fast-growing field that extends data analysis to non-Euclidean domain
- Highly interdisciplinary: machine learning, signal processing, harmonic analysis, network science, differential geometry, applies statistics
- Active research directions
  - beyond convolutional models or MPNNs
  - expressive power of graph ML models
  - robustness & generalisation & scalability
  - interpretability & causal inference
  - optimisation and implementation issues
  - applications (in particular healthcare!)