# AIMS CDT - Signal Processing

## Michaelmas Term 2025

Xiaowen Dong

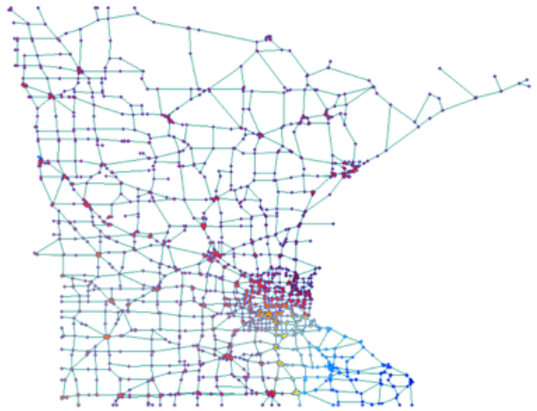
Department of Engineering Science

# Deep Learning on Graphs

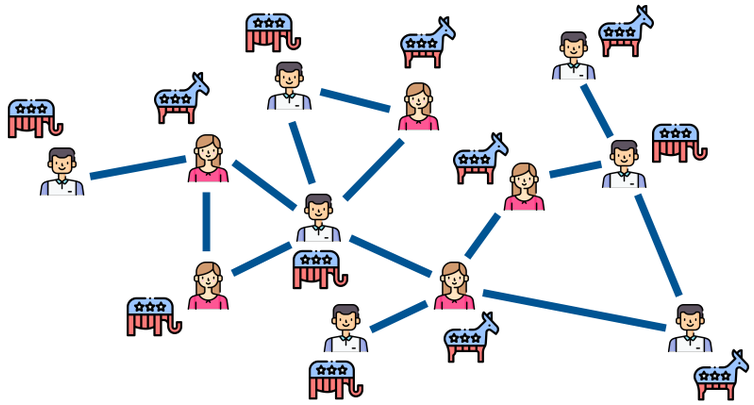
# Lecture 3

- Machine learning on graphs: Overview
- Convolutional neural networks on graphs
- Message passing neural networks
- Recent developments & Applications

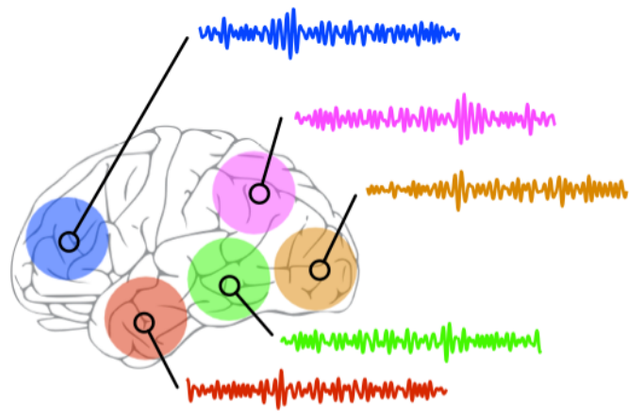
# Graph-structured data are pervasive



**congestion in road junctions**

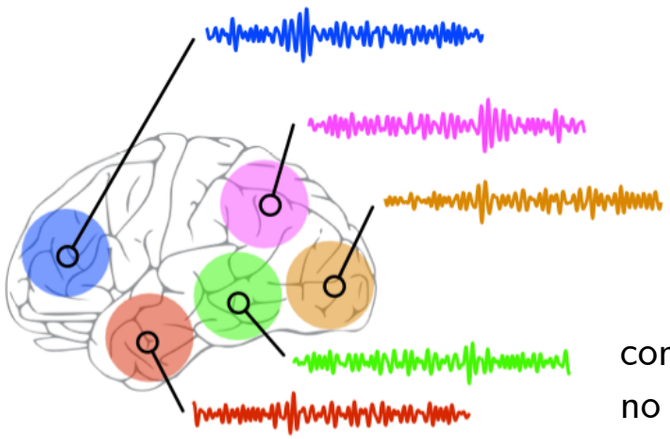


**preferences of individuals**



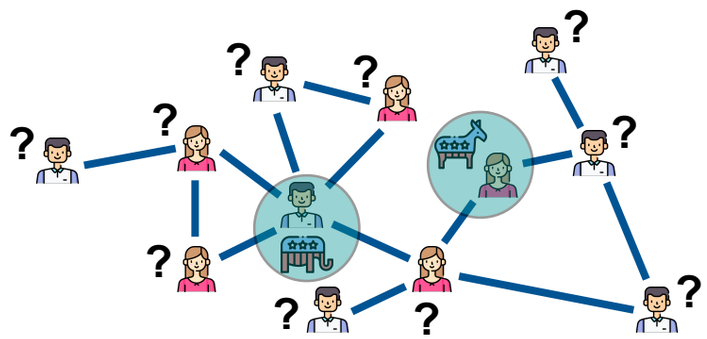
**activities in brain regions**

# Learning with graph-structured data

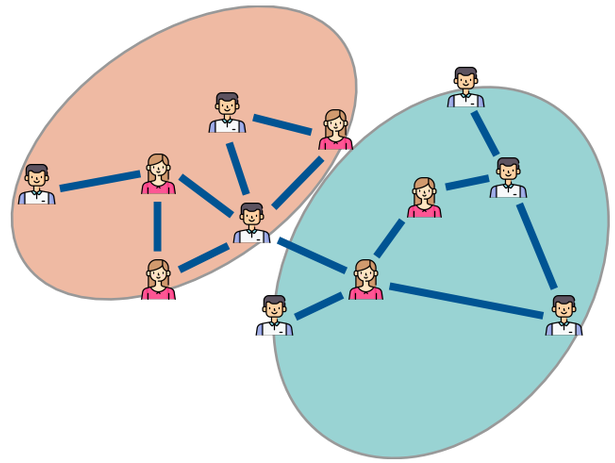


condition?
no condition?

**graph-level classification (supervised)**



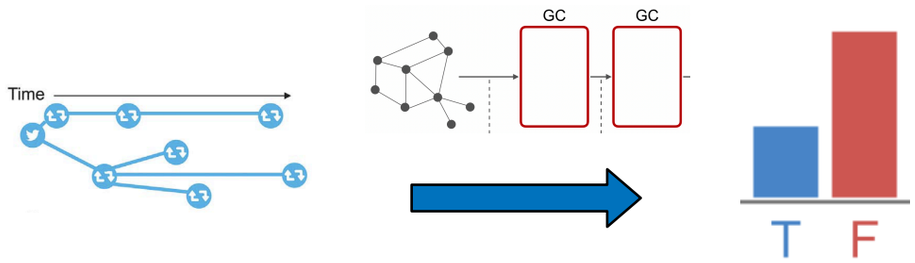
**node-level classification (semi-supervised)**



**graph clustering (unsupervised)**

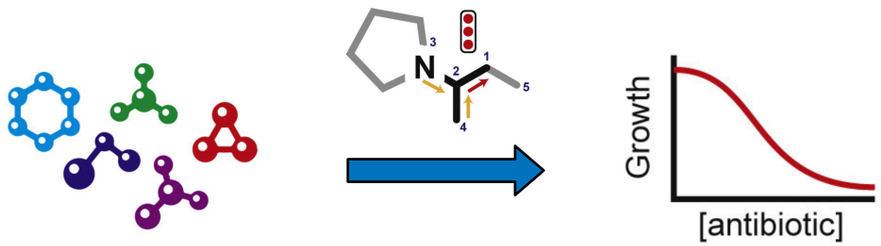
# Learning with graph-structured data

**fake news detection**



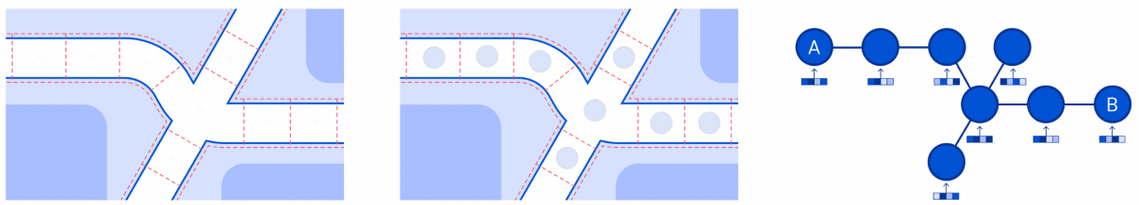


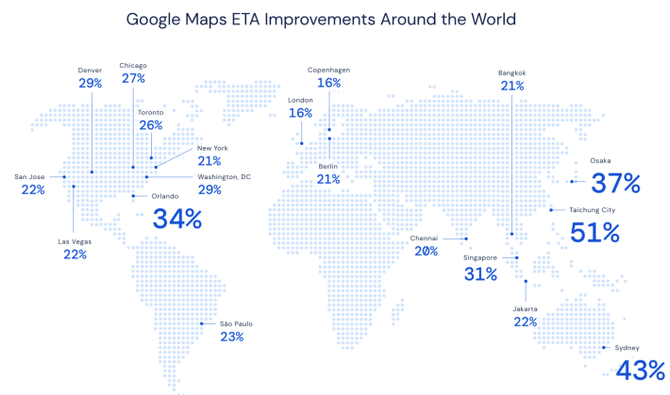
**drug discovery**





**traffic prediction**



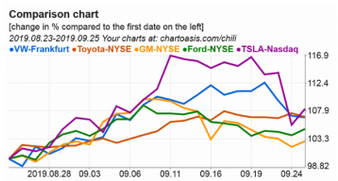



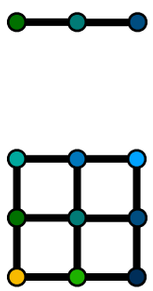

Monti et al., "Fake news detection on social media using geometric deep learning," ICLR Workshop, 2019.
Stokes et al., "A deep learning approach to antibiotic discovery," Cell, 2020.
Derrow-Pinion et al., "ETA prediction with graph neural networks in Google Maps," CIKM, 2021.

# Classical ML vs Graph ML

**Classical ML**

regular domain
(real line, 2D grid)



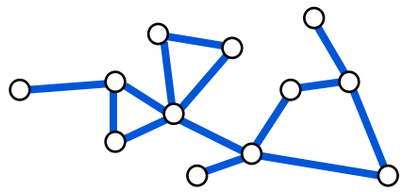
$f(X)$

time series forecasting

image classification

**Graph ML**

irregular domain
(graphs)



$\mathcal{G}$

$X$

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

node classification

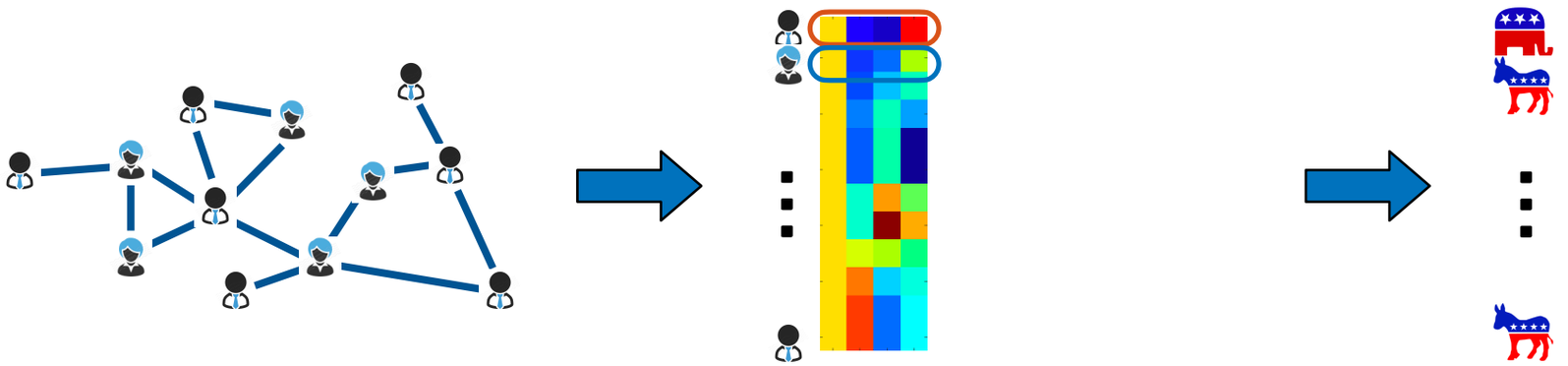
link prediction

graph classification

graph clustering

# How to incorporate graphs into learning?

- Traditional machine learning on graphs

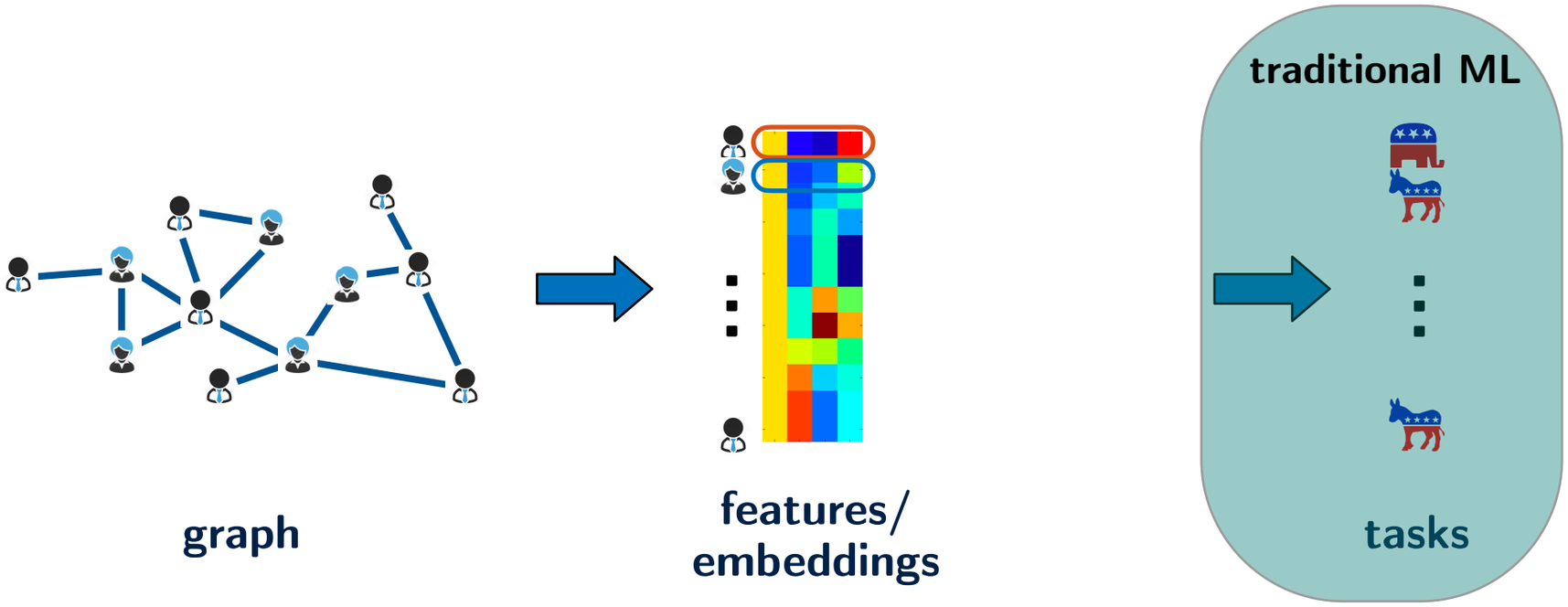


**graph** **features/ embeddings** **tasks**

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

# How to incorporate graphs into learning?

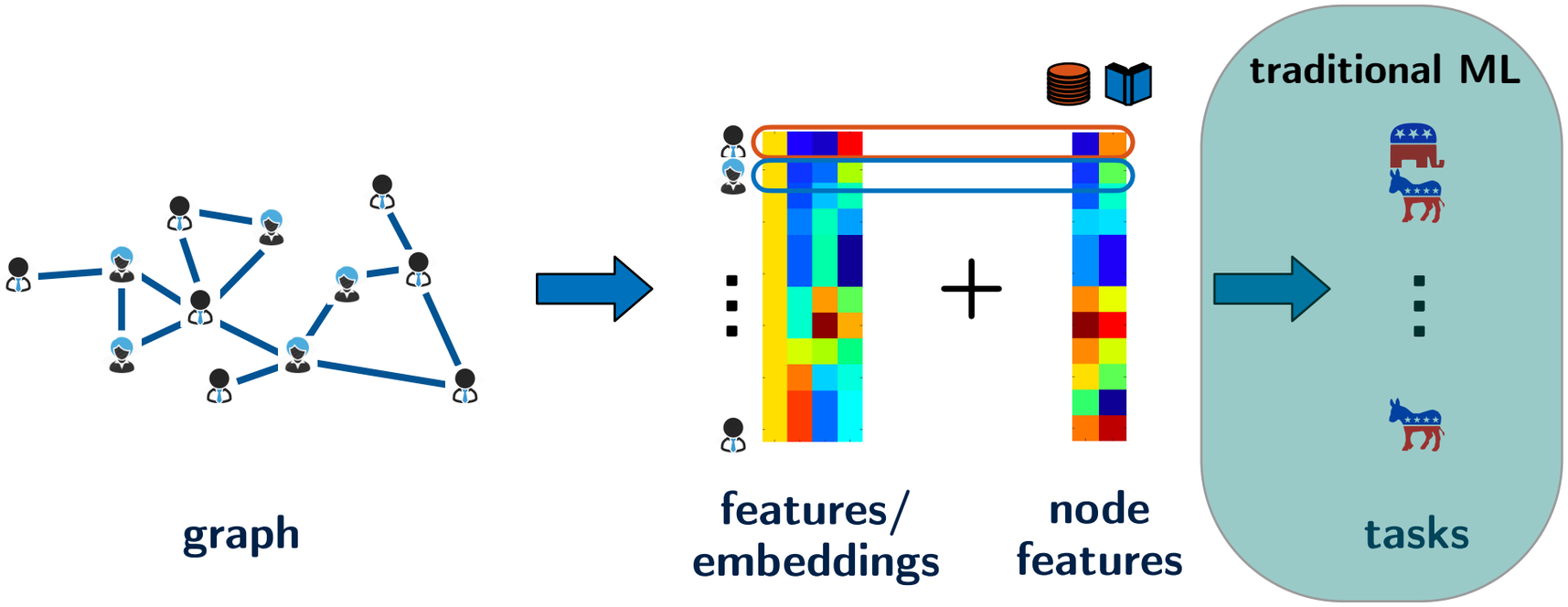
- Traditional machine learning on graphs




- 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?

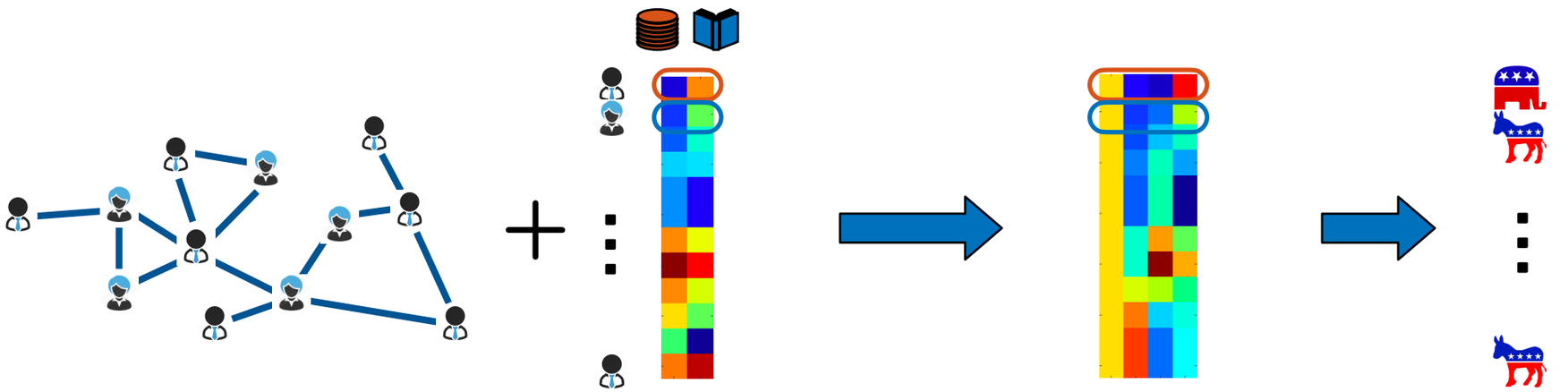
- Traditional machine learning on graphs




- 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

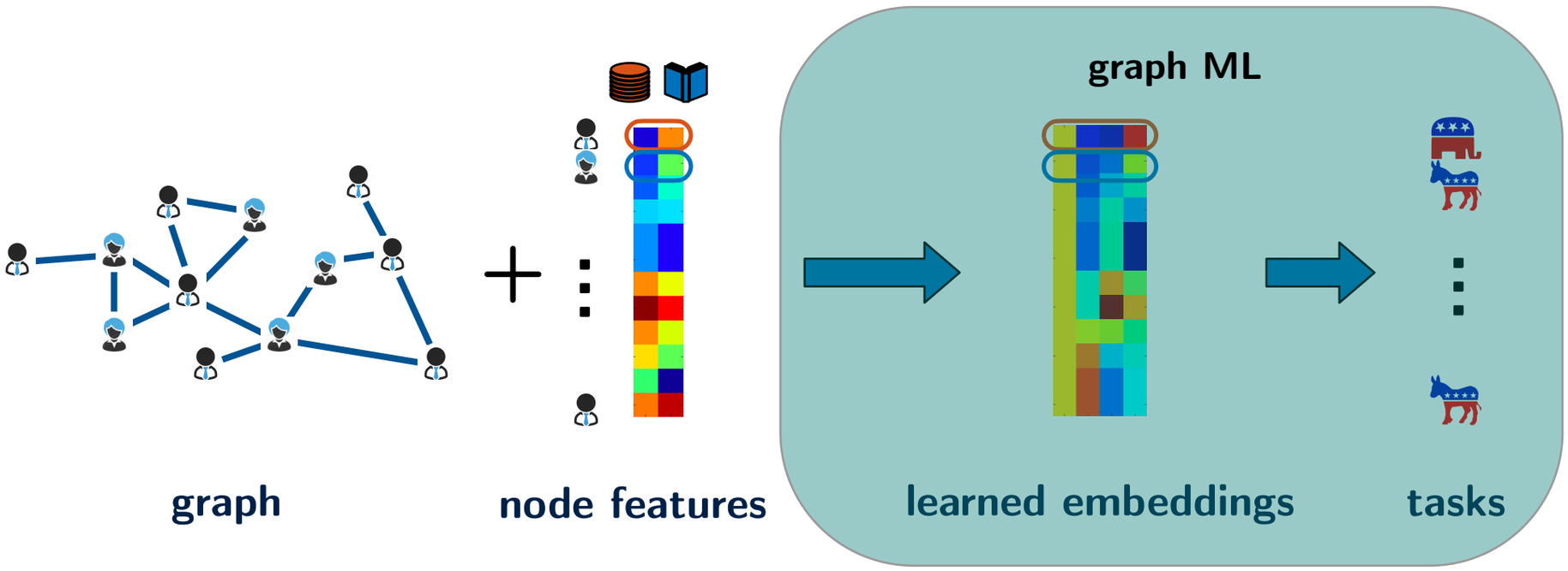


**graph** **node features** **learned embeddings** **tasks**

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

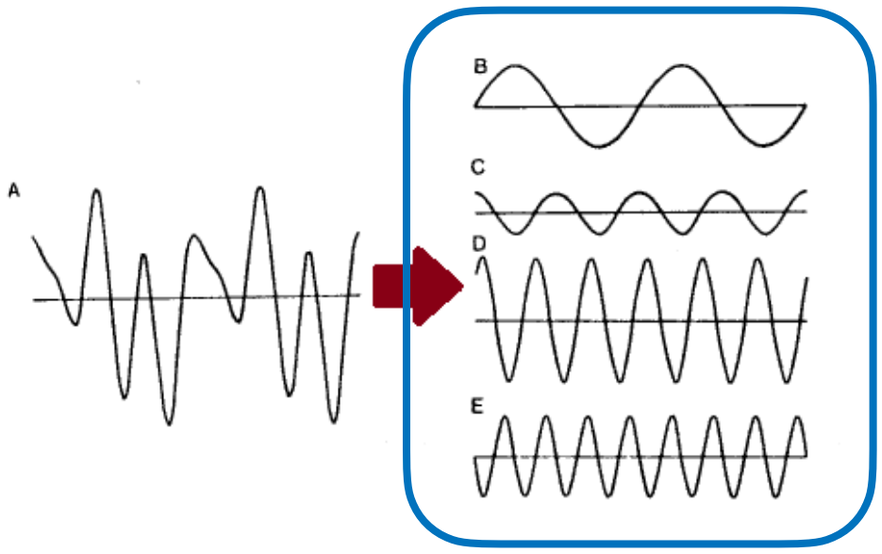
# How to incorporate graphs into learning?

- 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

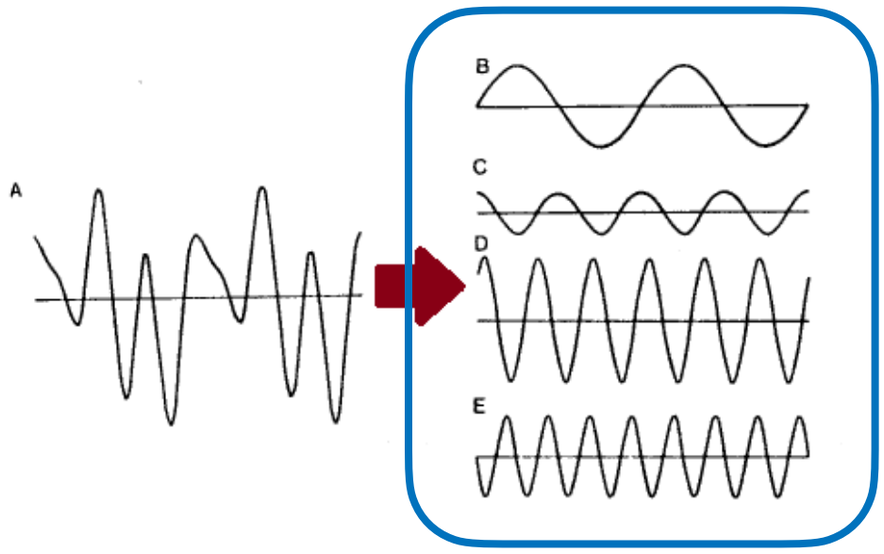
# Graph signal processing




classical signal processing

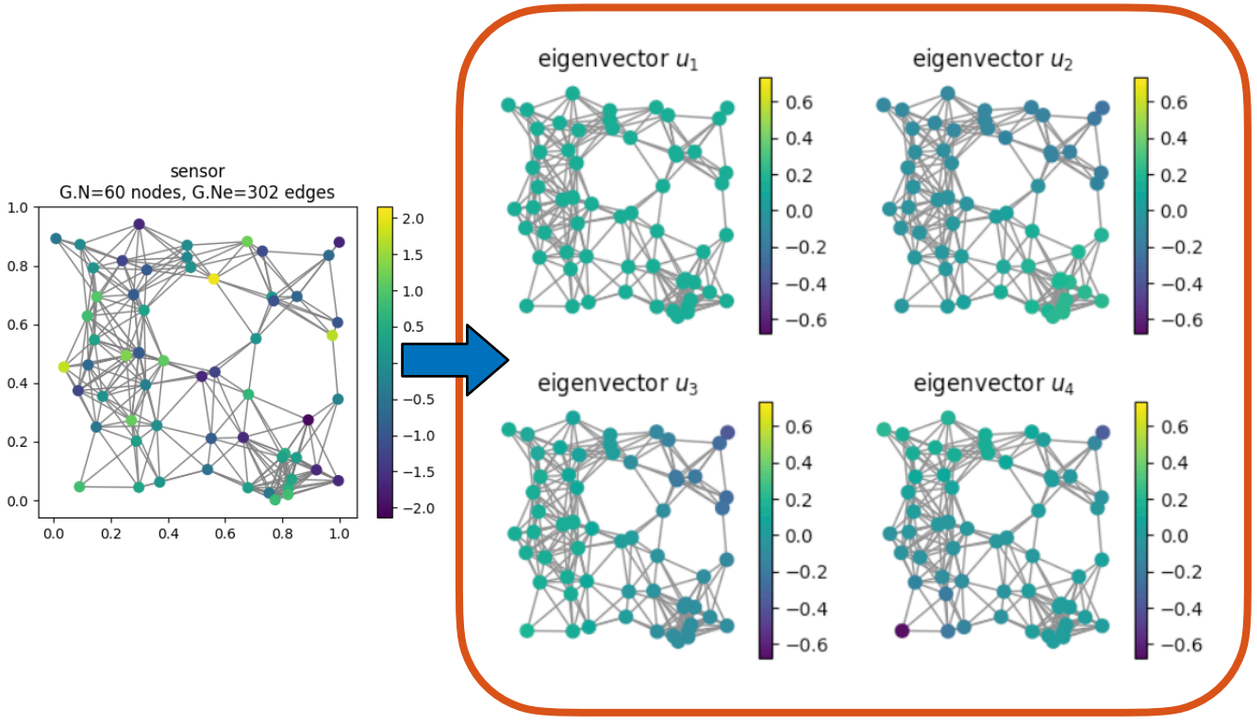
- complex exponentials provide "building blocks" of 1D signal (different oscillations or frequencies)
- leads to **Fourier transform**
- enables convolution and filtering

# Graph signal processing




classical signal processing

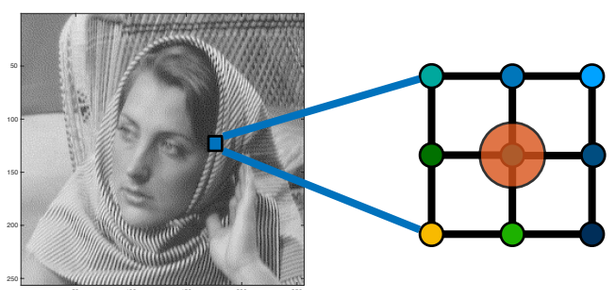
- complex exponentials provide "building blocks" of 1D signal (different oscillations or frequencies)
- leads to **Fourier transform**
- enables convolution and filtering

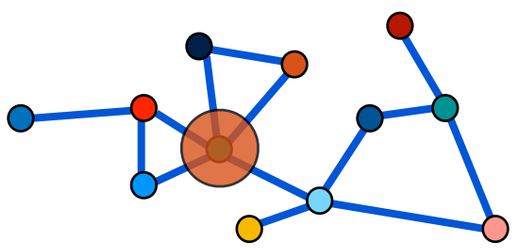



graph signal processing

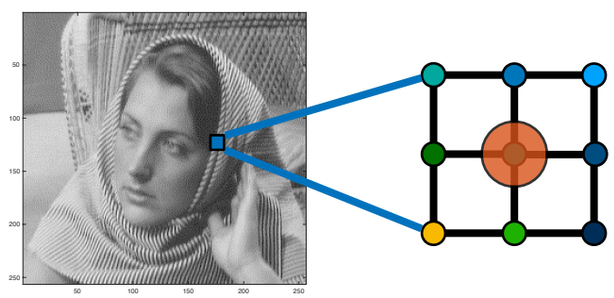
- Laplacian eigenvectors provide "building blocks" of graph signal (different oscillation or frequencies)
- leads to **graph Fourier transform**
- enables **convolution** and **filtering** on graphs

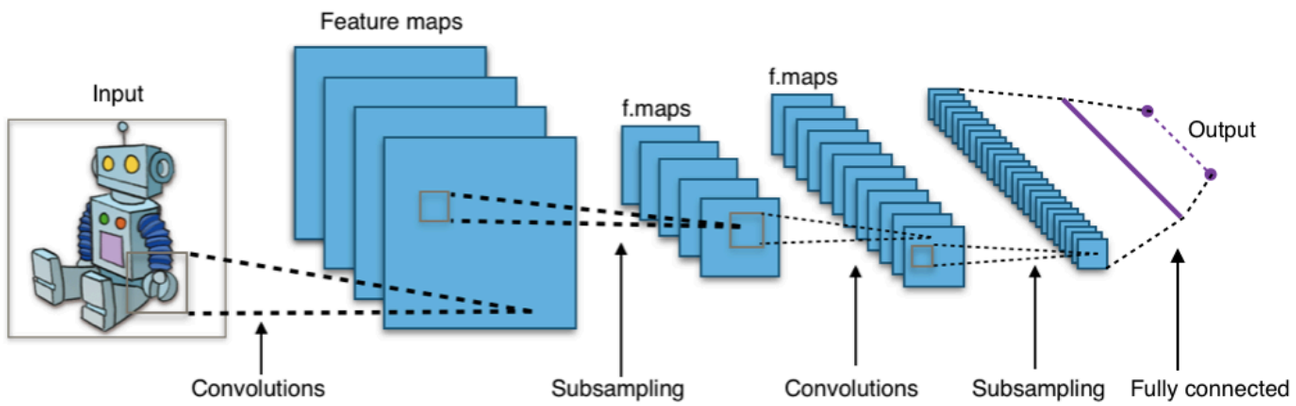
# Convolutional neural networks on graphs

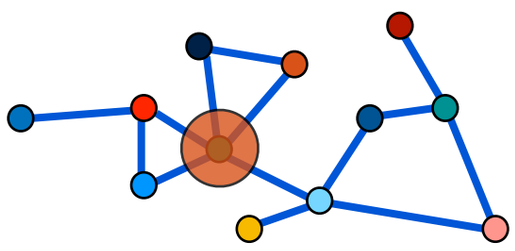


# Convolutional neural networks on graphs

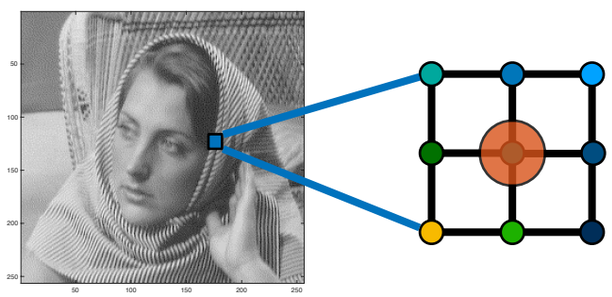


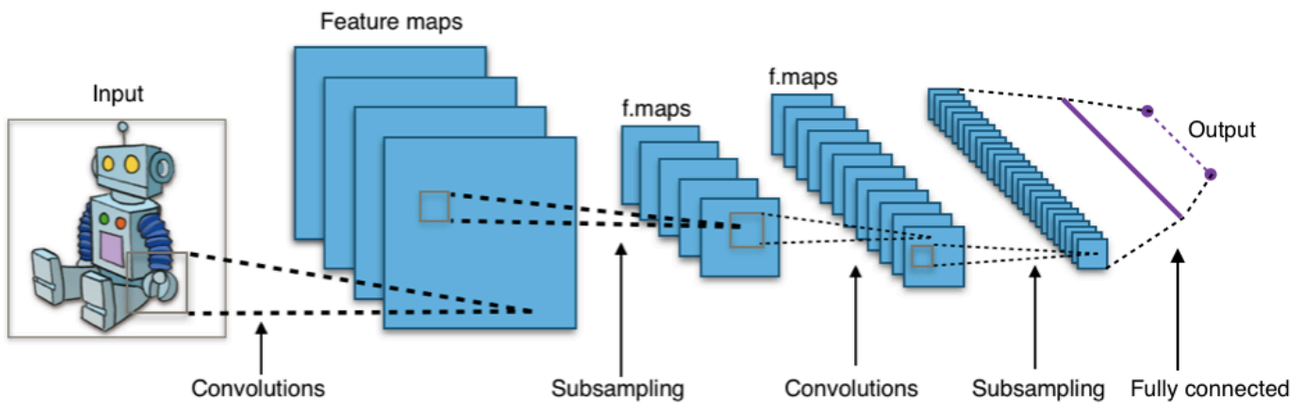


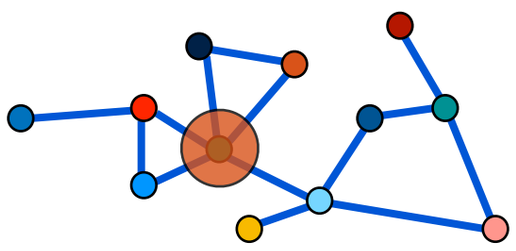



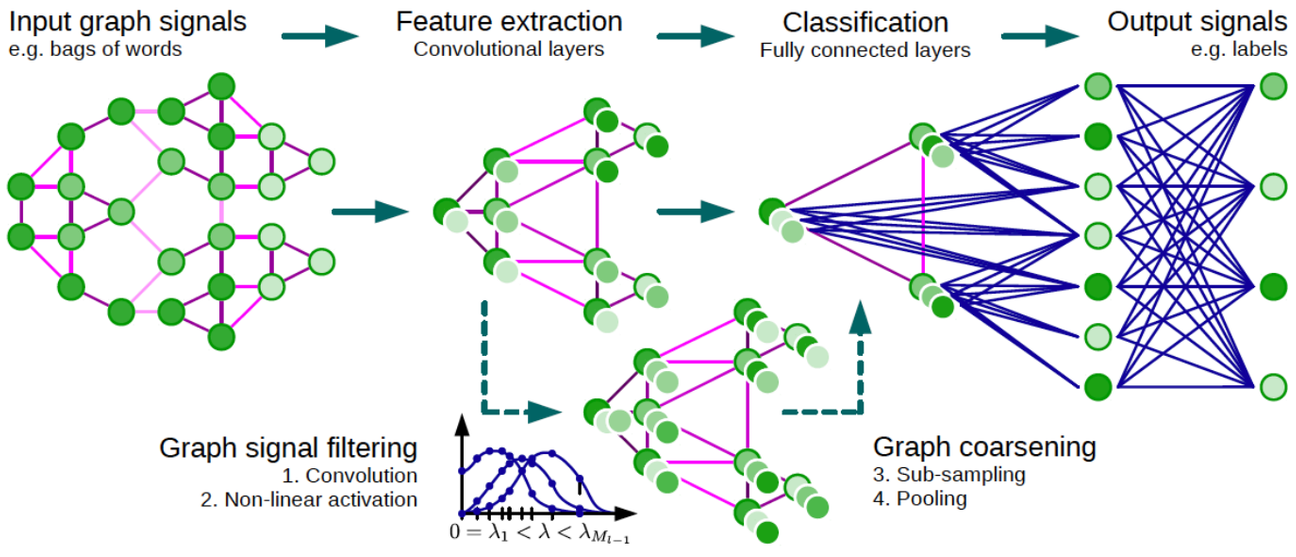
?

# Convolutional neural networks on graphs





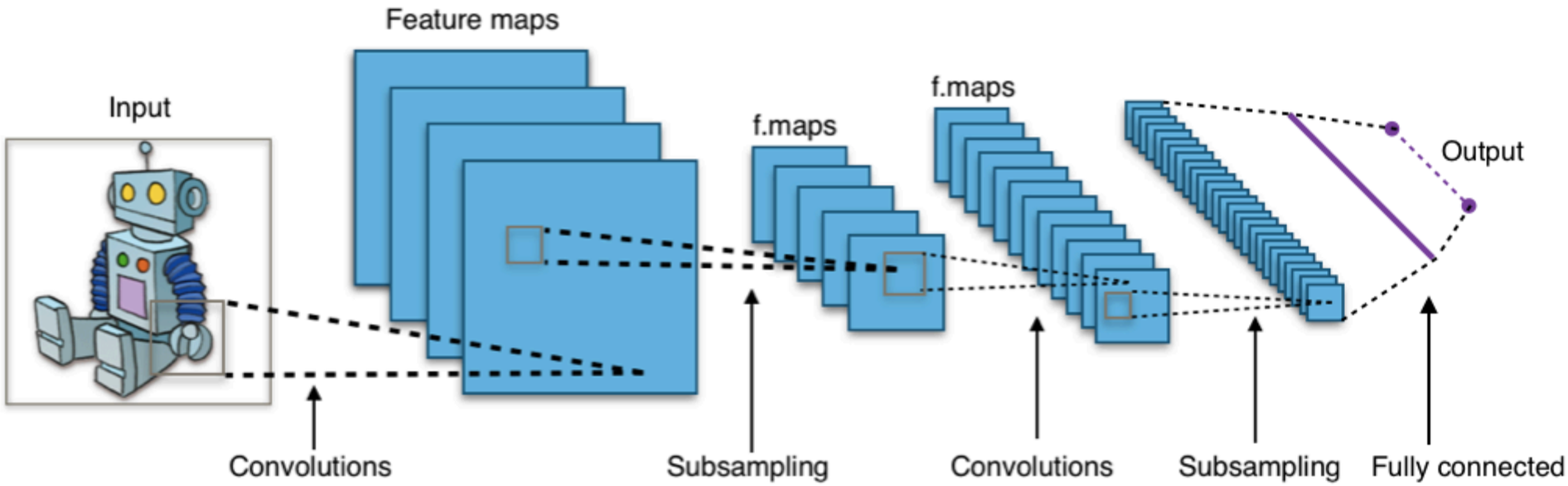





# Lecture 3

- Machine learning on graphs: Overview
- Convolutional neural networks on graphs
- Message passing neural networks
- Recent developments & Applications

# CNNs exploit structure within data

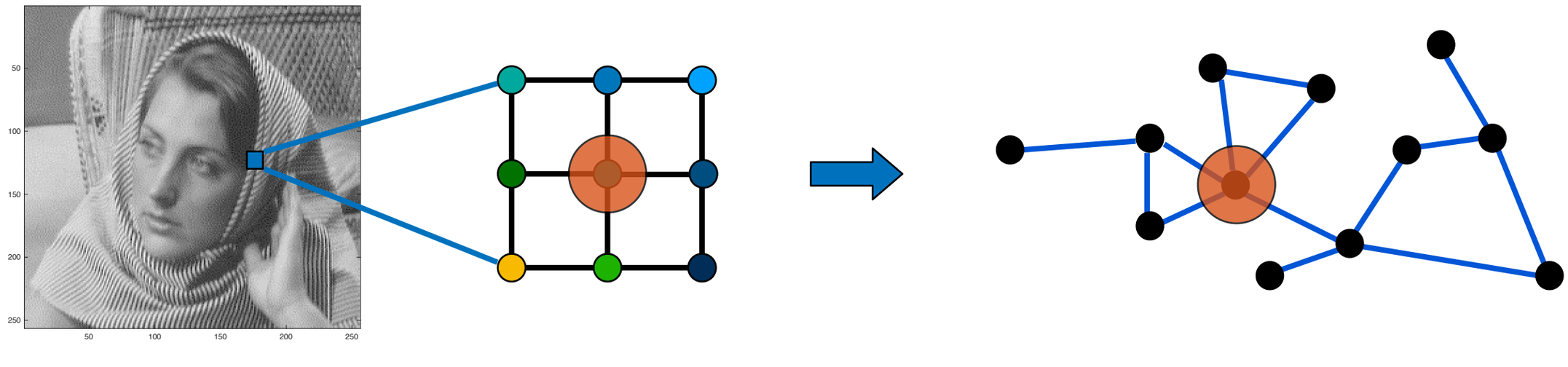



checklist

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

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

# CNNs on graphs?




checklist

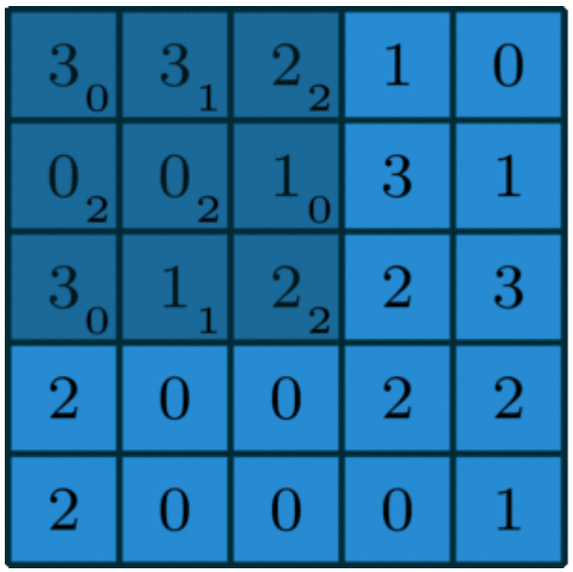
- **convolution:** how to define convolution? what about invariance?
- **localisation:** what is the notion of locality?
- **multi-scale:** how to down-sample on graphs?
- **efficiency:** how to keep the computational complexity low?

# Convolution on graphs

**classical convolution**

time domain

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





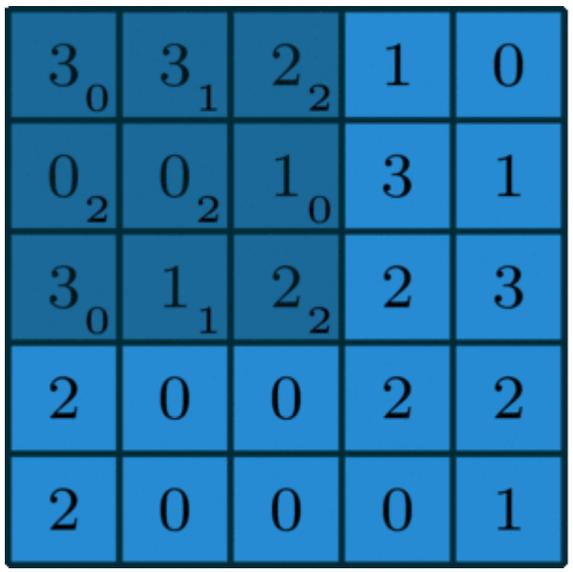
https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

# Convolution on graphs

**classical convolution**

time domain

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





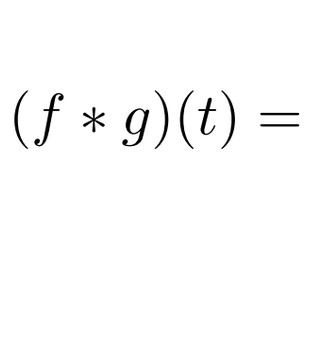
https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

# Convolution on graphs

**classical convolution**

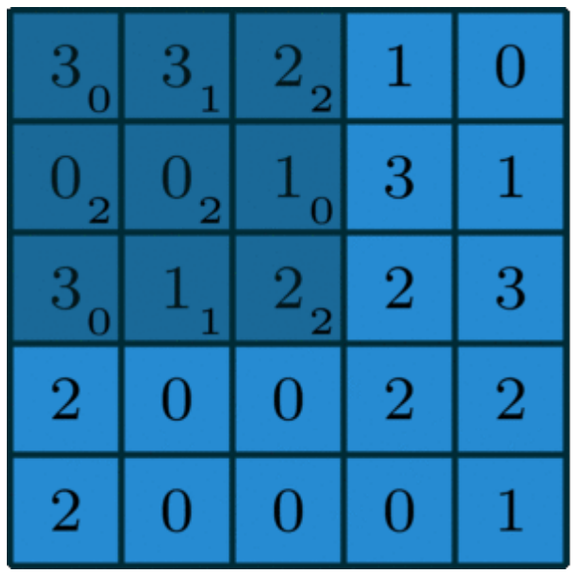
time domain

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



frequency domain

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





https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

# 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

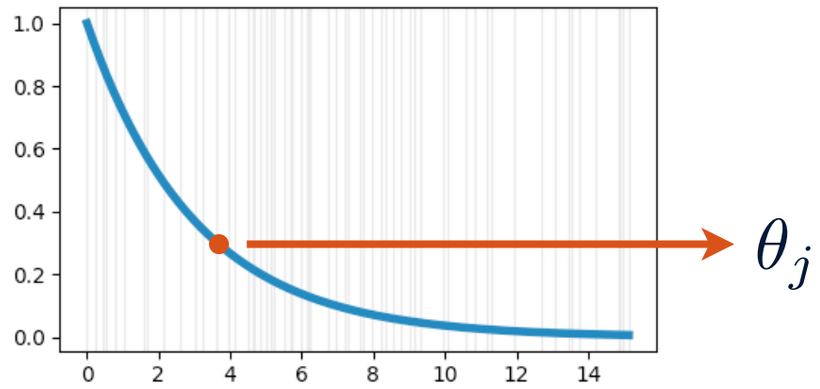
$$\widehat{(f * g)}(\lambda) = \big((\chi^T f) \circ \hat{g}\big)(\lambda)$$

# A non-parametric filter

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

learning a non-parametric filter:

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

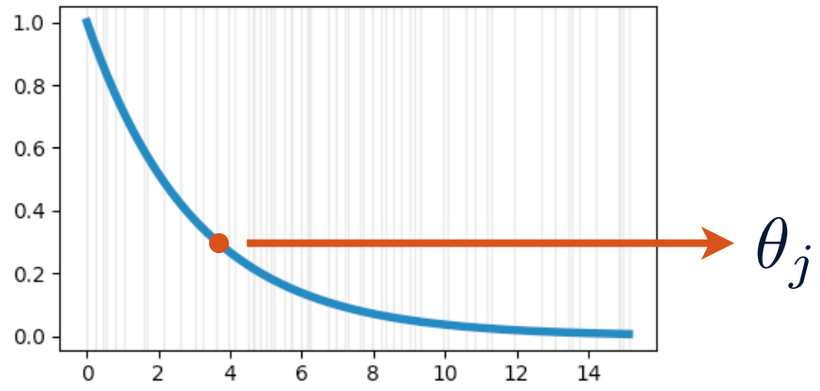


# A non-parametric filter

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

learning a non-parametric filter:

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



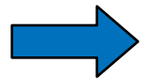

- convolution expressed in the graph spectral domain
- no localisation in the spatial (node) domain
- computationally expensive

# A parametric filter

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

parametric filter as polynomial of Laplacian

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



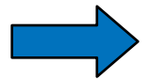
$$\hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$$

# A parametric filter

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

parametric filter as polynomial of Laplacian

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

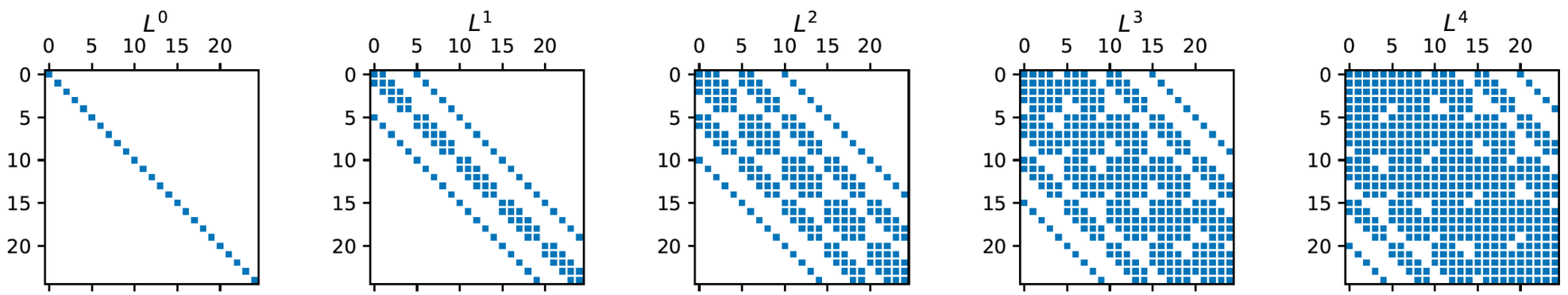


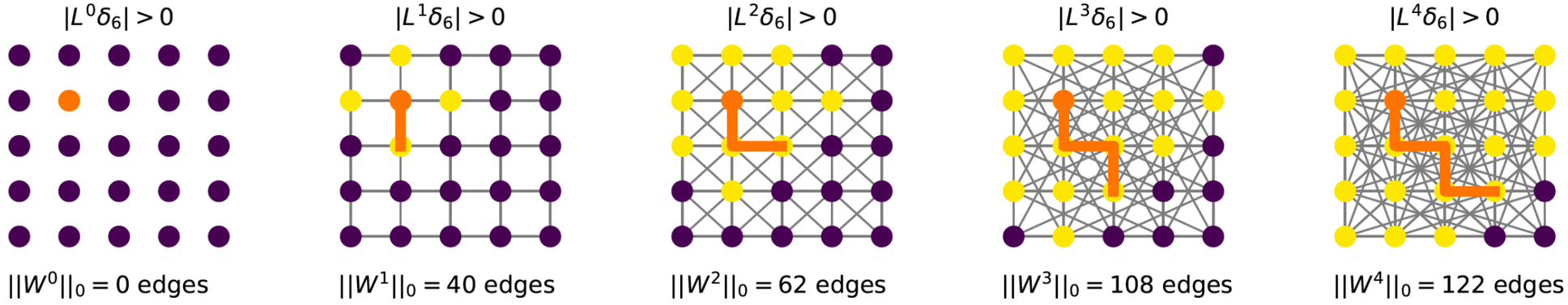
$$\hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$$

what do powers of graph Laplacian capture?

# Powers of graph Laplacian

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




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

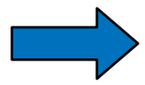
(slide by M. Deferrard)

# A parametric filter

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

parametric filter as polynomial of Laplacian

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



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

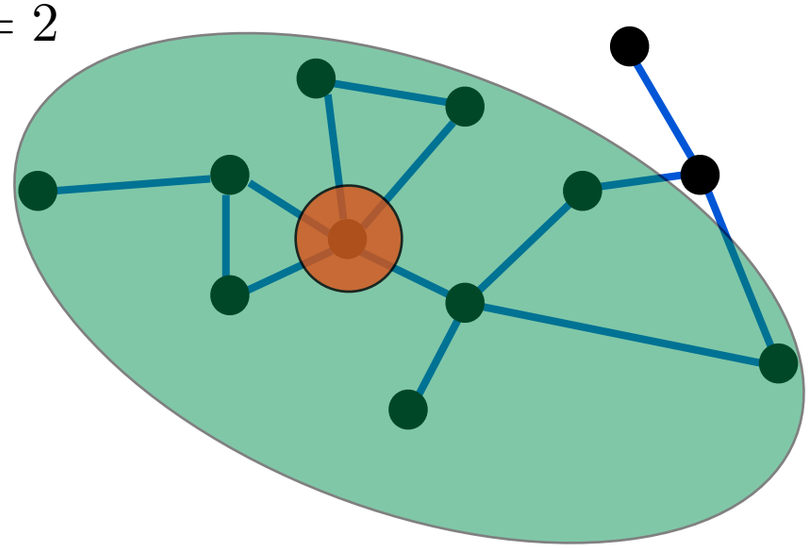
# A parametric filter

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

parametric filter as polynomial of Laplacian

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

$K = 2$



- localisation within $K$-hop neighbourhood

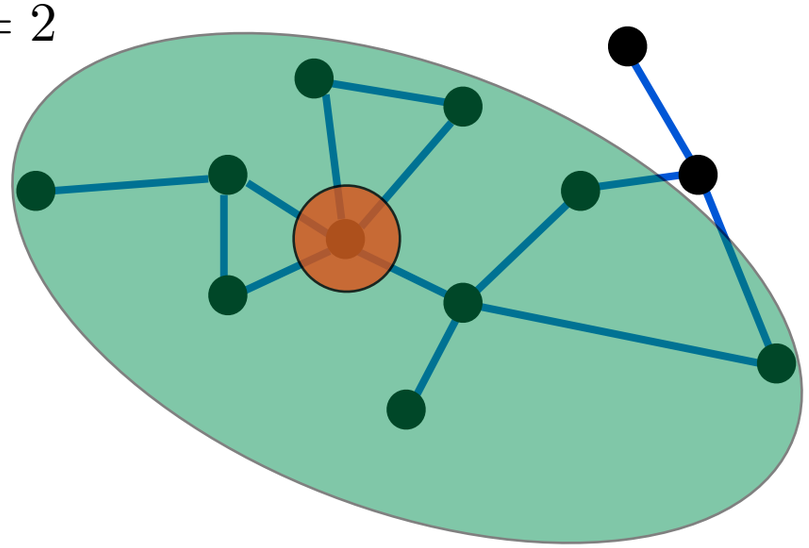
# A parametric filter

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

parametric filter as polynomial of Laplacian

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

$K = 2$



- 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**

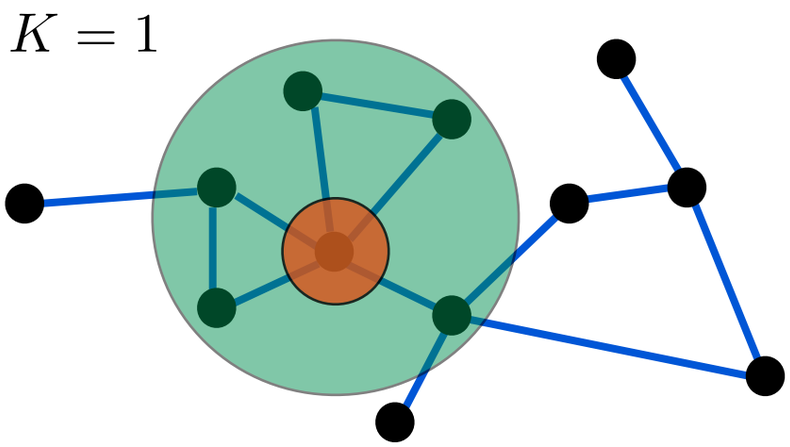
$$\begin{aligned} L_{\text{norm}} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \\ &= D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} \\ &= I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = I - W_{\text{norm}} \end{aligned}$$

$K = 1$

normalised Laplacian

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

(localisation within 1-hop neighbourhood)



# 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**

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

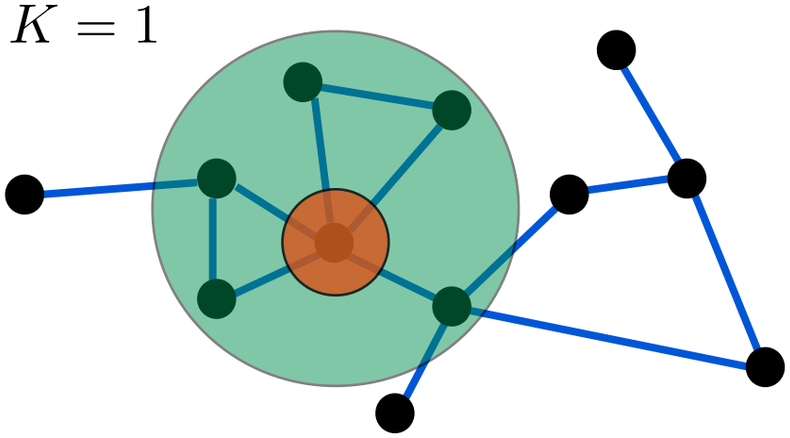
$K = 1$

normalised Laplacian

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

(localisation within 1-hop neighbourhood)

$K = 1$



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

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

# 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**

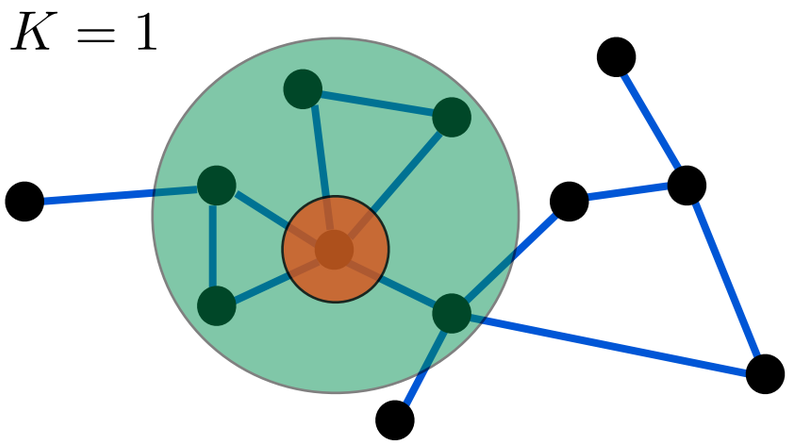
$$L_{\text{norm}} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$
$$= D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}}$$
$$= I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = I - W_{\text{norm}}$$

$K = 1$

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

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

**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

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

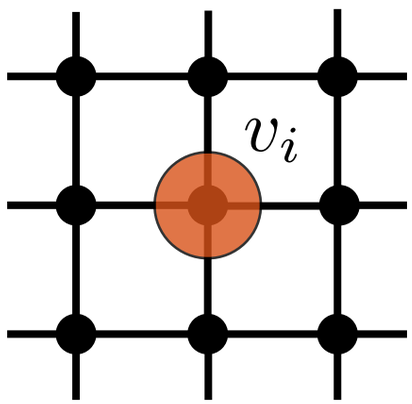
# A simplified parametric filter

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

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

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

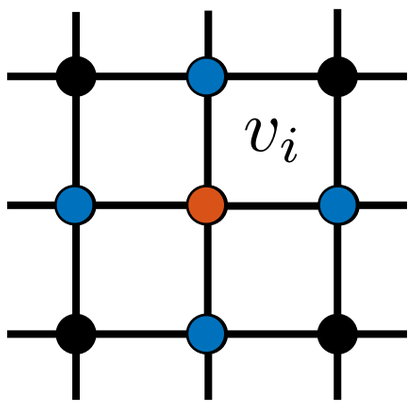
simplified parametric filter

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

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

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

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

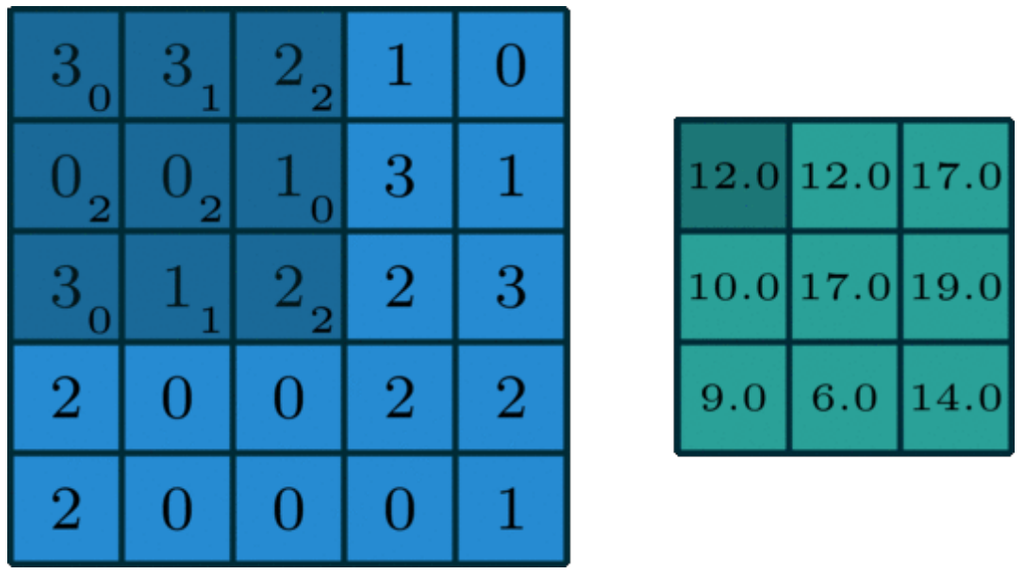
simplified parametric filter

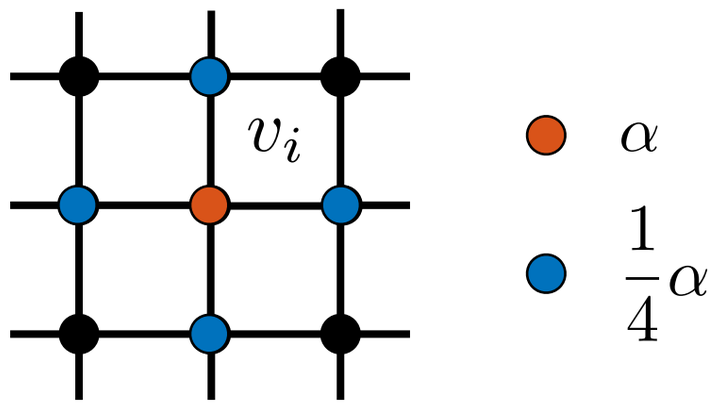
$$\hat{g}_\alpha(L) = \alpha(I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}})$$

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

unitary edge weights

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



# 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 in classical signal processing relies on the shift operator

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

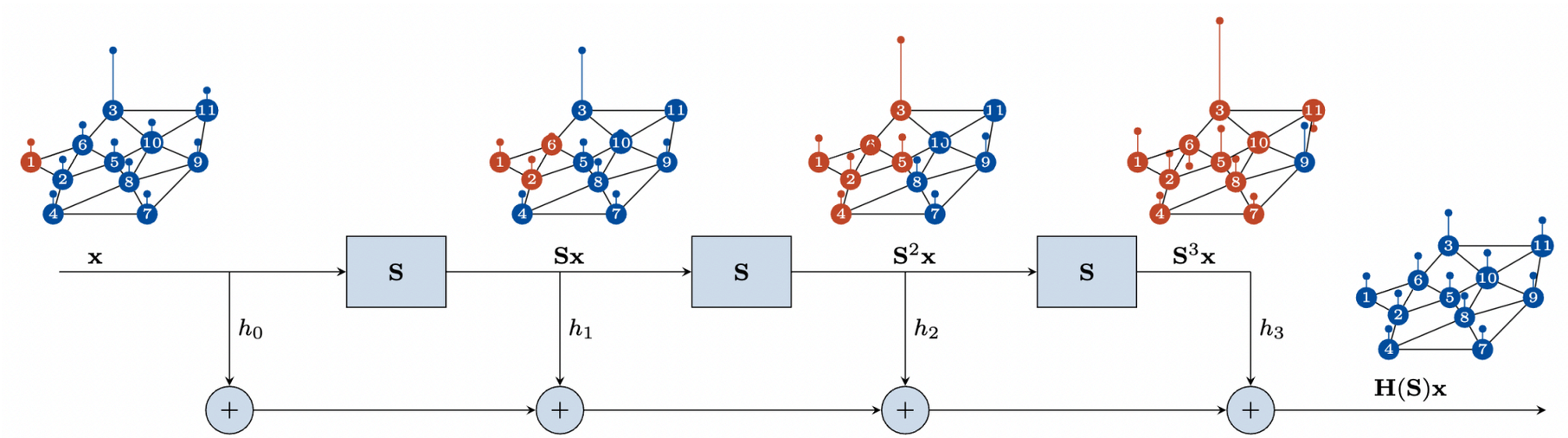
# Convolution on graphs - Remarks

- Convolution in classical signal processing relies on the shift operator

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

- Idea: convolution via a graph shift operator (e.g., adjacency matrix)

$$Sf \quad \Rightarrow \quad g(S)f = \sum_{k=0}^{K} \theta_k S^k f$$




Isufi et al., "Graph filters for signal processing and machine learning on graphs," IEEE TSP, 2024.

# Convolution on graphs - Remarks

- Convolution can also be interpreted as a weighted summation

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

# Convolution on graphs - Remarks

- Convolution can also be interpreted as a weighted summation

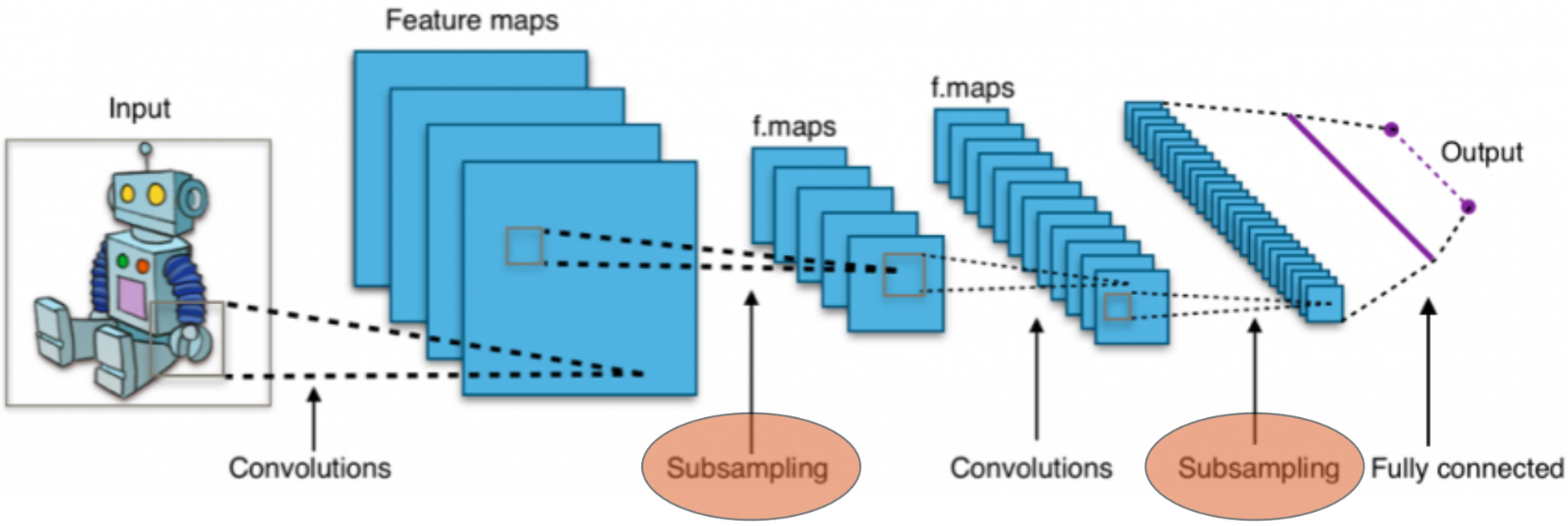
$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

- Idea: convolution via a spatial weighted summation in graph domain

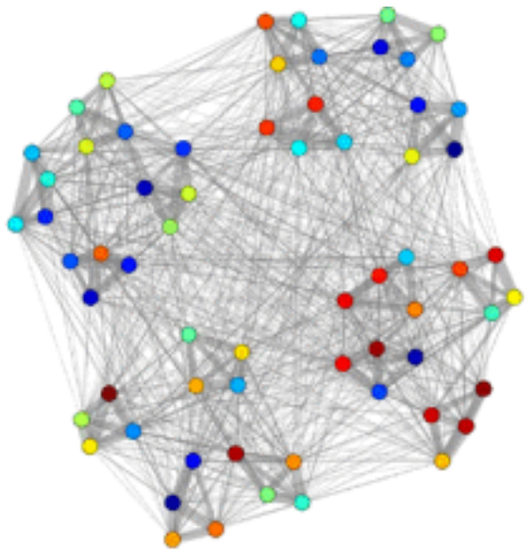
$$(f * g)(v) = \sum_{v' \in \mathcal{V}} f(v') g(v, v')$$

**relative importance of each node to v**
**(can be designed to achieve locality)**

# Pooling on graphs

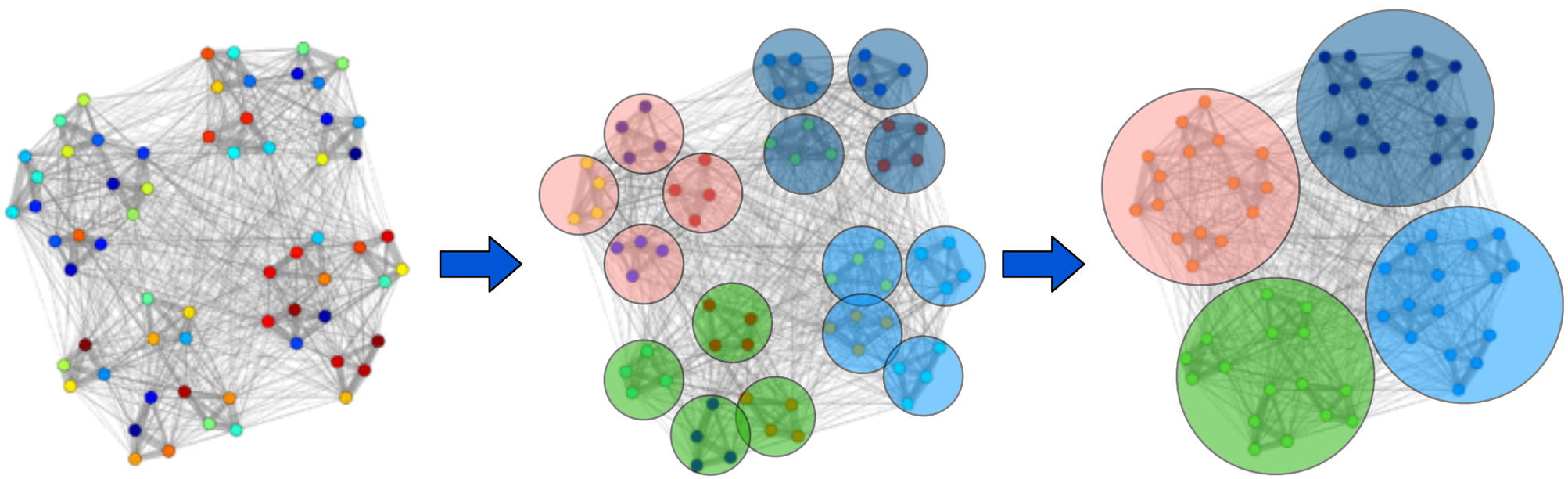


# Pooling on graphs



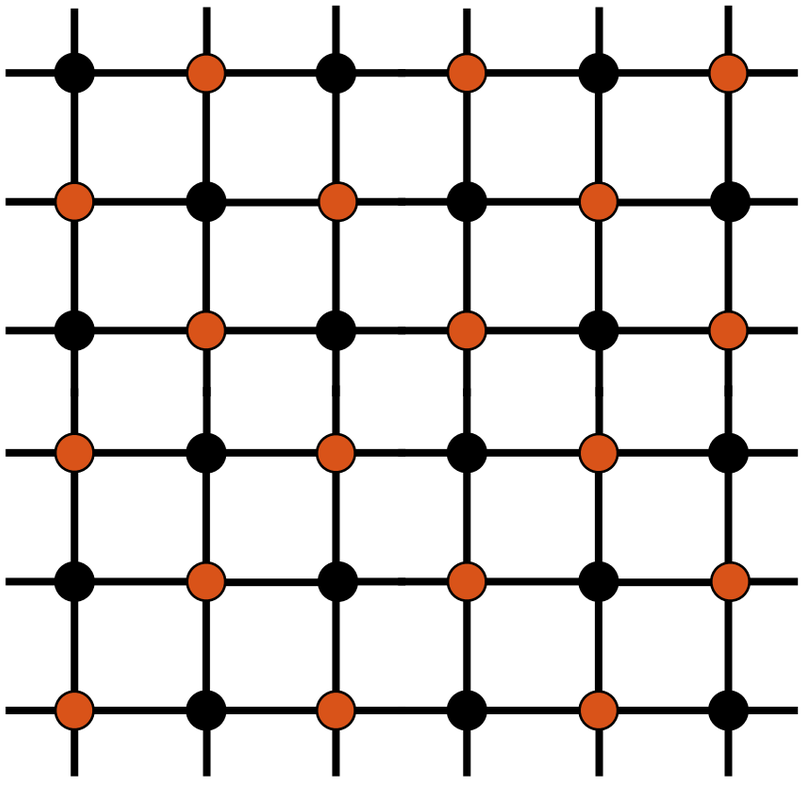
- pooling = downsampling on graphs, but how?

# Pooling on graphs



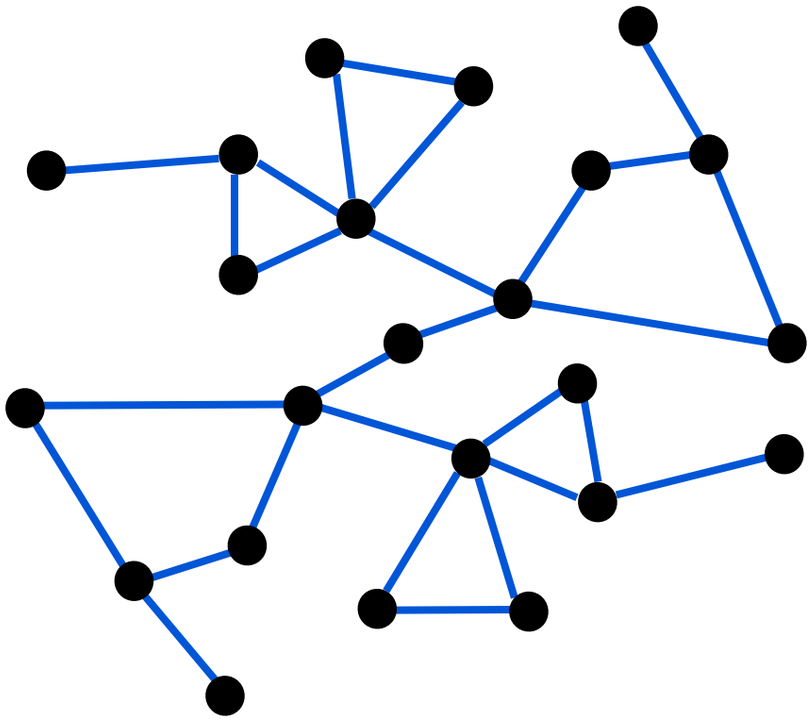
- pooling = downsampling on graphs, but how?
- natural idea: graph coarsening

# Pooling on graphs



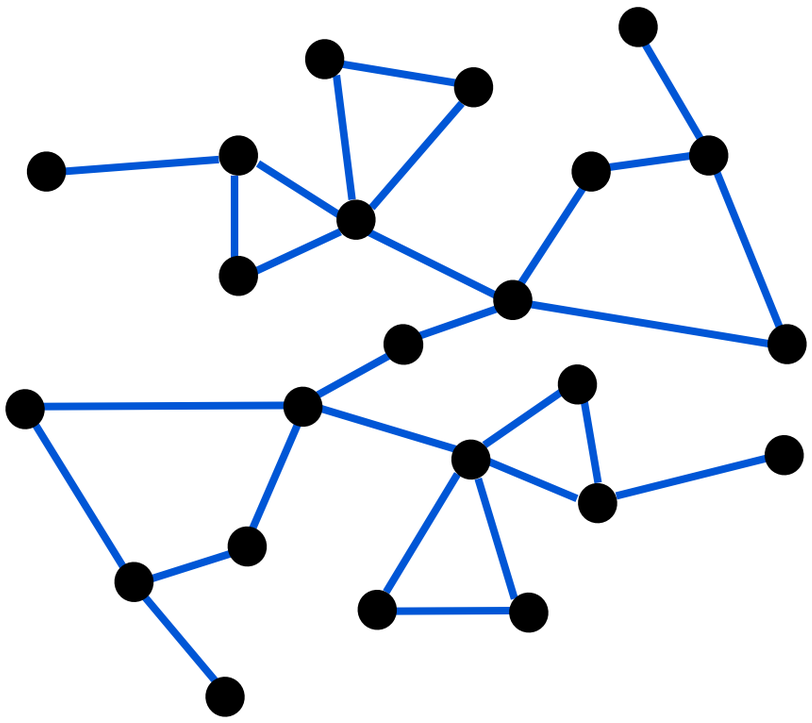
- coarsening is straightforward on regular grids

# Pooling on graphs



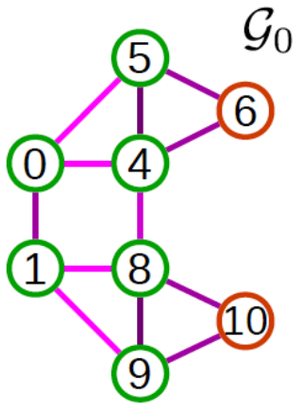
- coarsening is straightforward on regular grids
- not so much on irregular graphs

# Pooling on graphs



- coarsening is straightforward on regular grids
- not so much on irregular graphs
- can be achieved via node clustering
  - multi-level partitioning
  - roughly fixed downsampling factor (e.g., 2)
  - need for efficiency

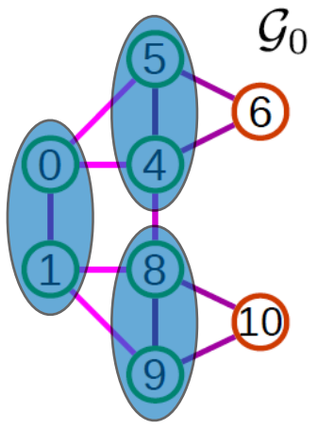
# Pooling on graphs




- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

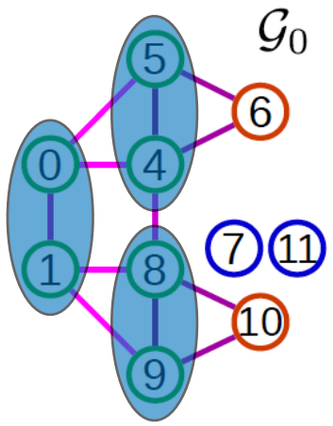
# Pooling on graphs




- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

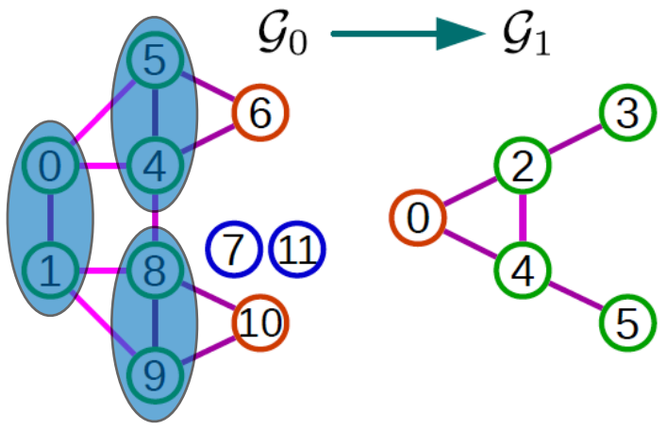
# Pooling on graphs




- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

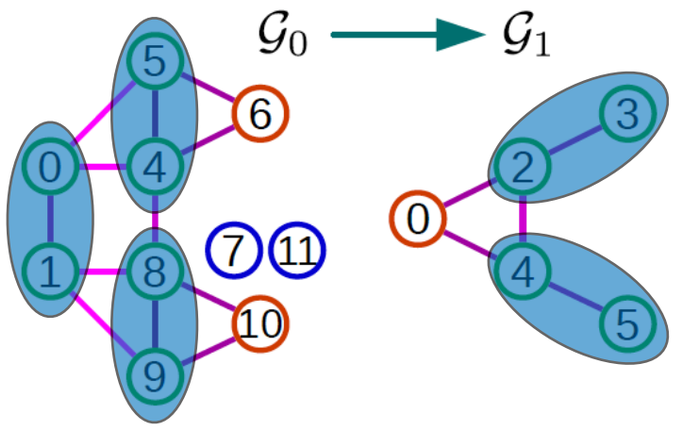
# Pooling on graphs




- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

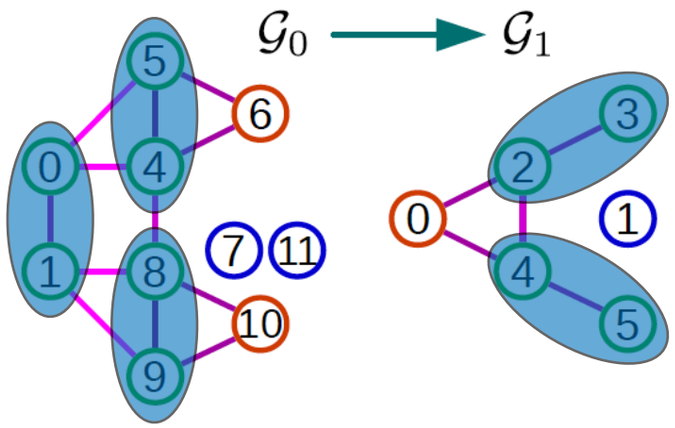
# Pooling on graphs




- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

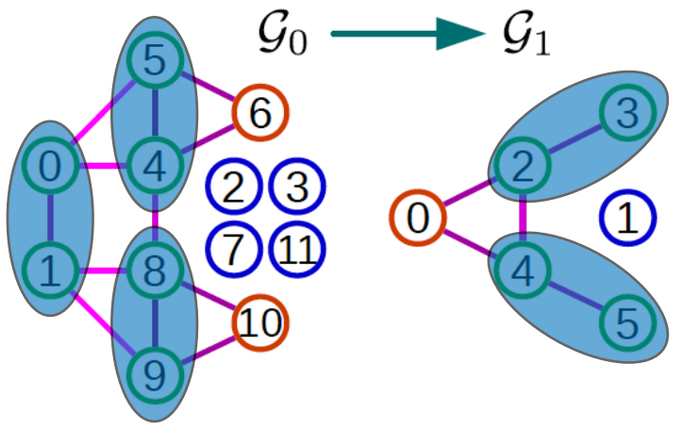
# Pooling on graphs




- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

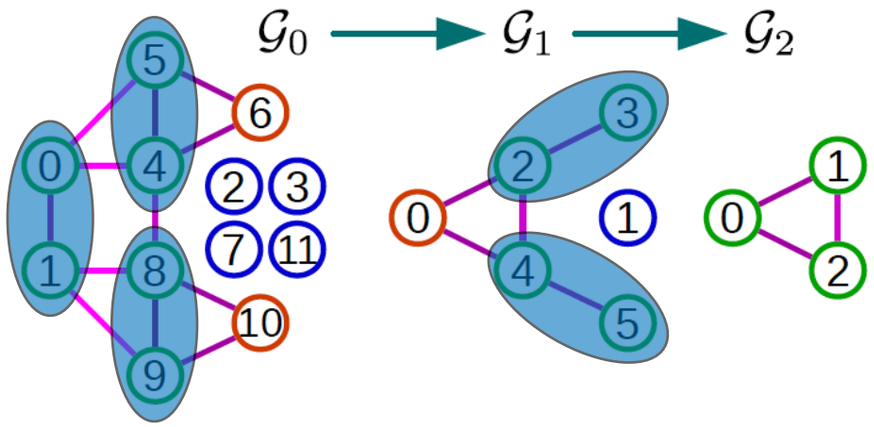
# Pooling on graphs




- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

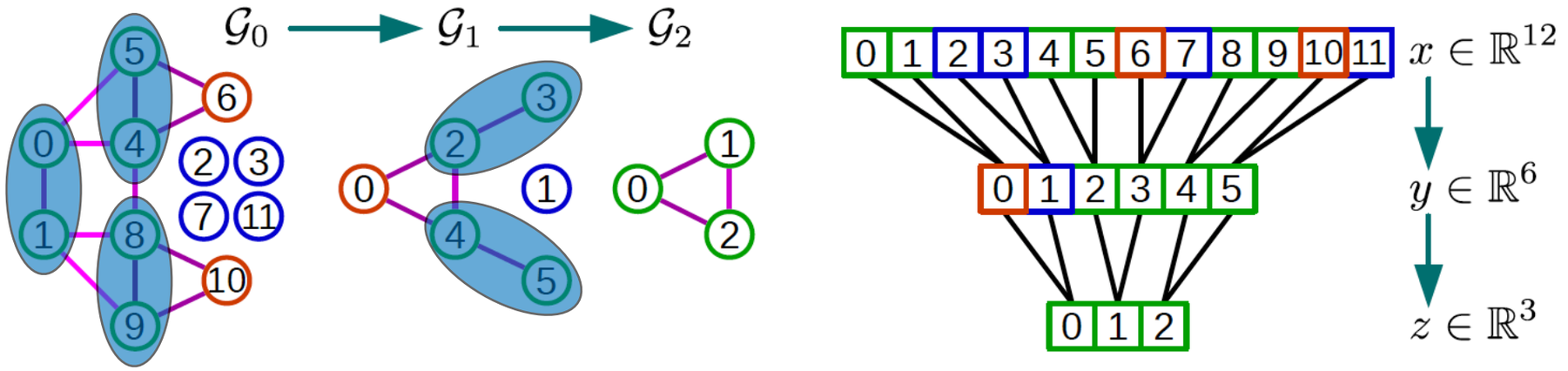
# Pooling on graphs




- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

# Pooling on graphs

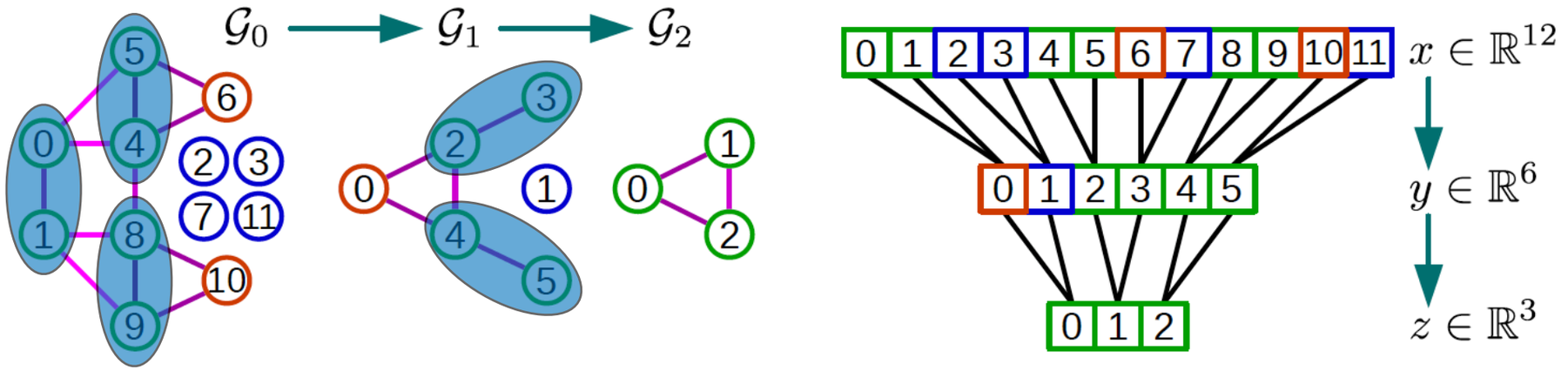



[Defferrard et al. 2016]

- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

# Pooling on graphs

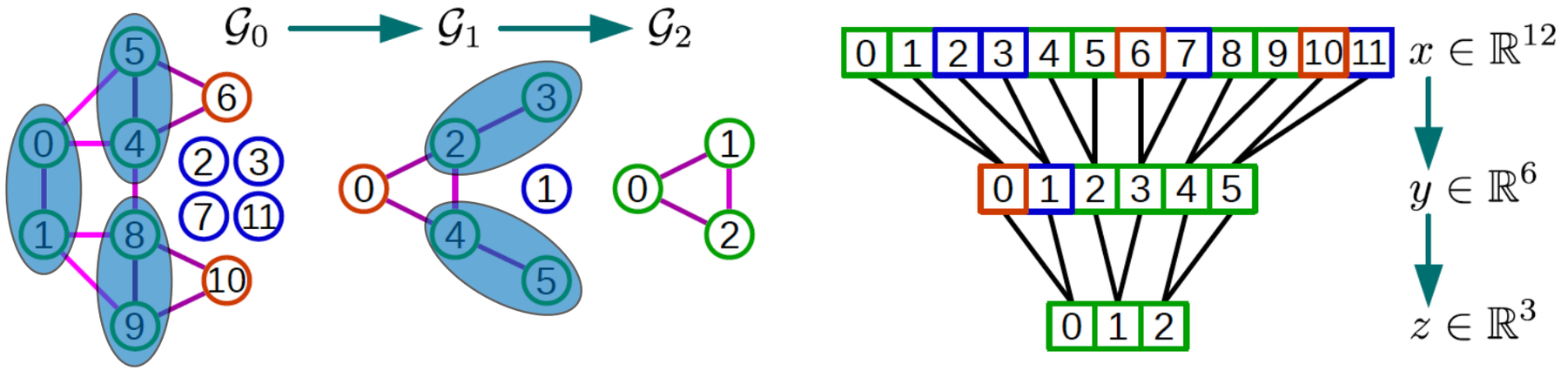



[Defferrard et al. 2016]

- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node
  - 1D grid pooling: [ max(0,1) max(4,5,6) max(8,9,10) ]

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

# Pooling on graphs



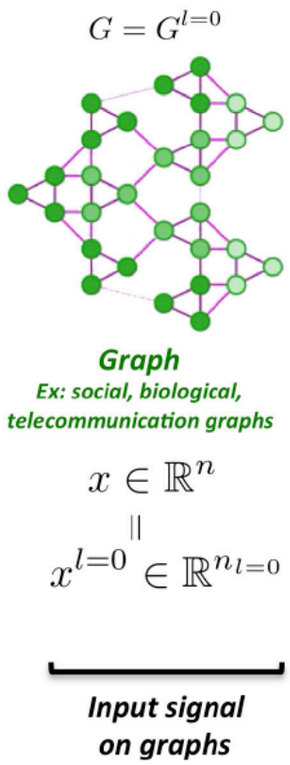

[Defferrard et al. 2016]

- pooling based on Graclus algorithm
  - local greedy way of merging vertices: maximising $w_{ij}(1/d_i + 1/d_j)$
  - adding artificial vertices to ensure two children for each node
  - 1D grid pooling: [ max(0,1) max(4,5,6) max(8,9,10) ]
  - only based on graph (and no signal) information

Dhillon et al., "Weighted graph cuts without eigenvectors: A multilevel approach," IEEE TPAMI, 2007.

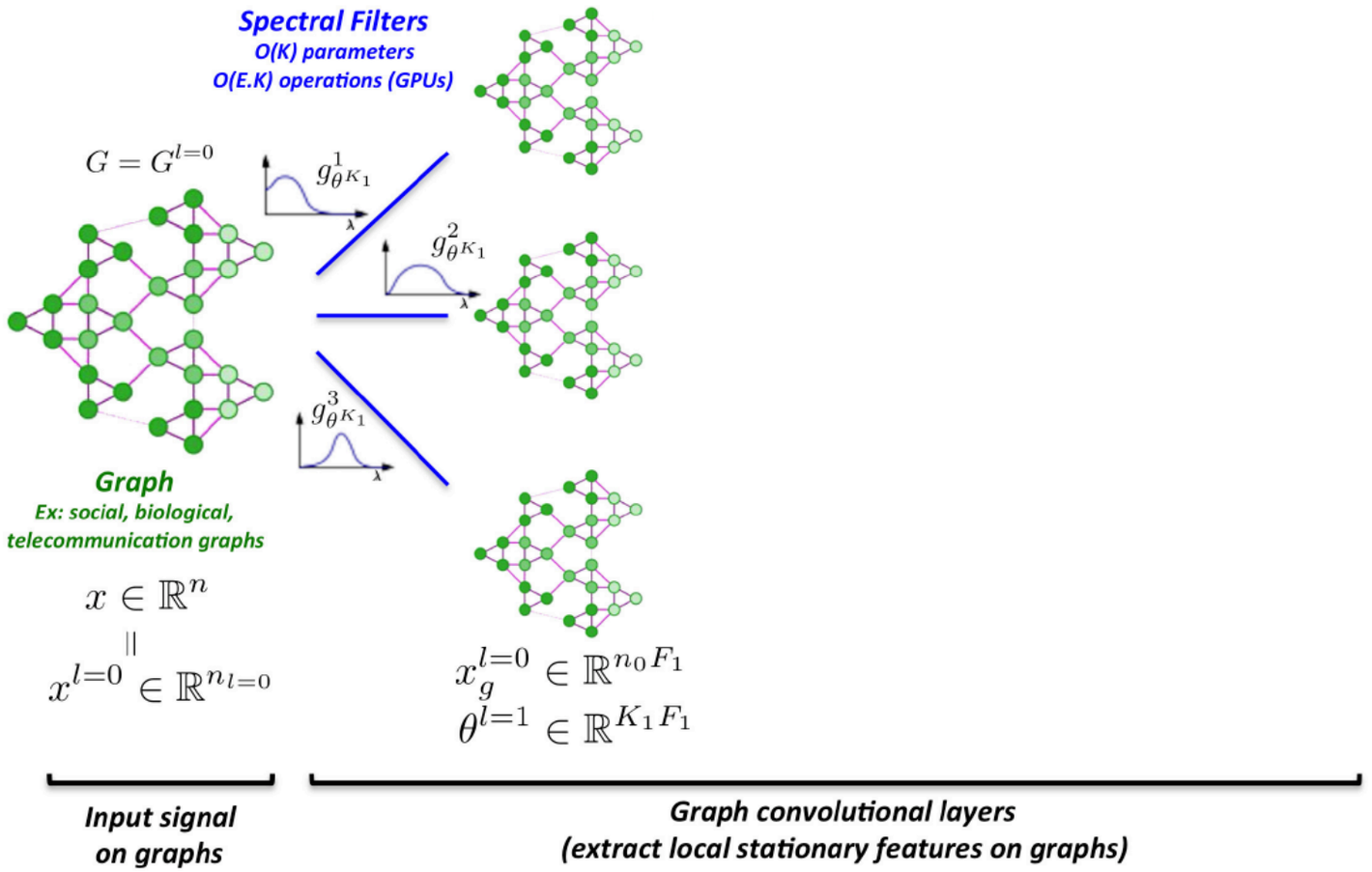
# CNNs on graphs: Graph classification

- ChebNet architecture



# CNNs on graphs: Graph classification

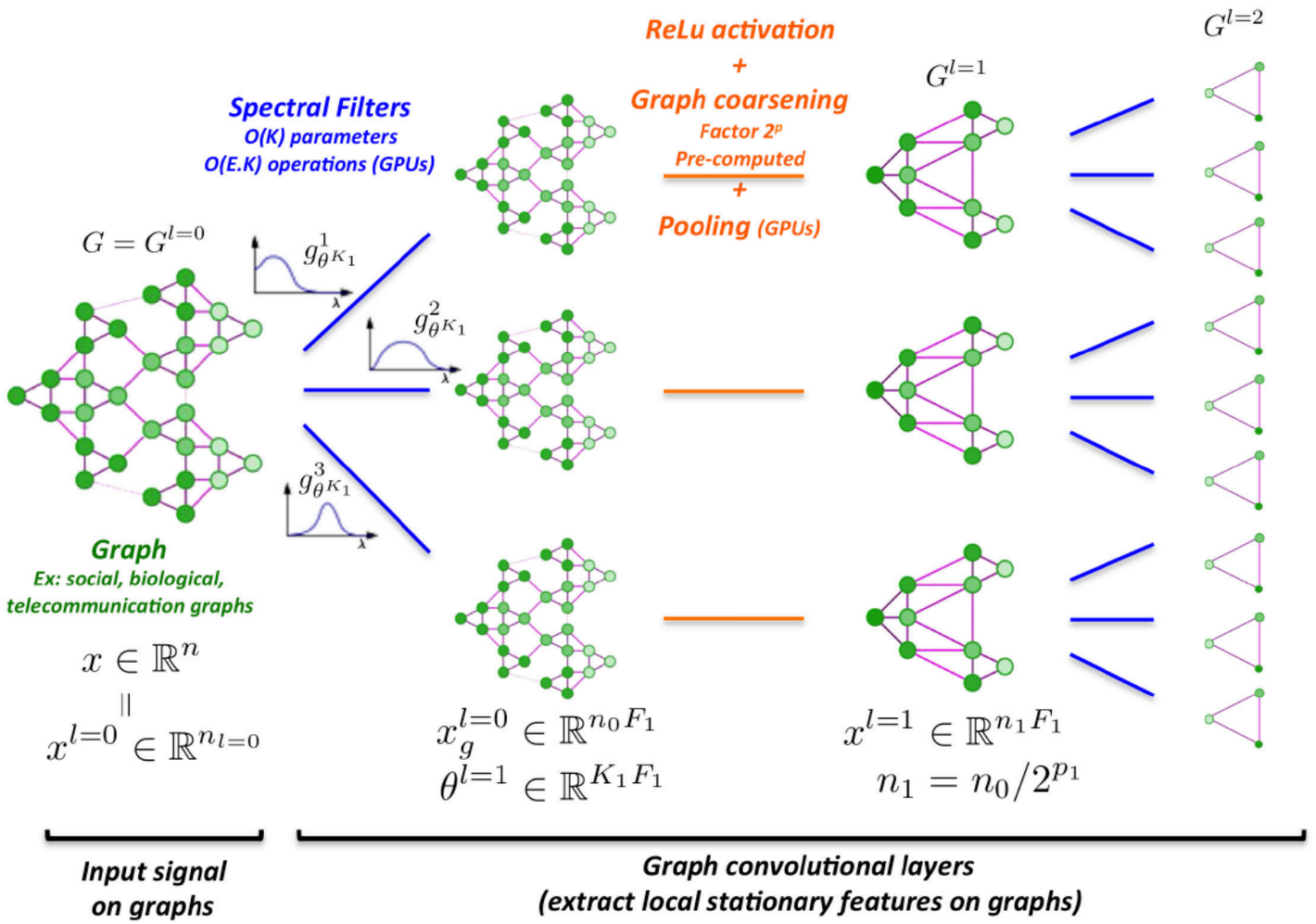
- ChebNet architecture




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

# CNNs on graphs: Graph classification

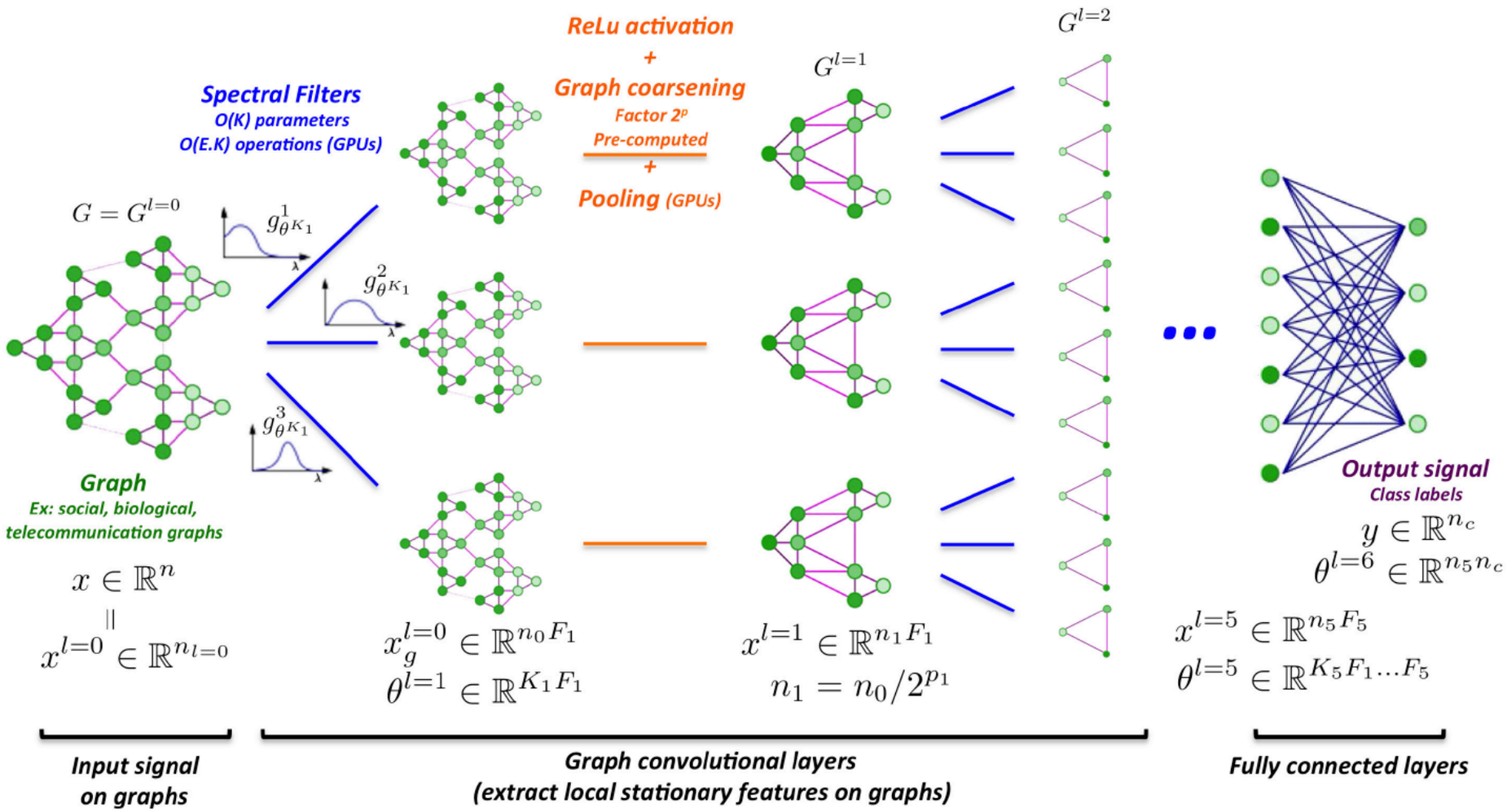
- ChebNet architecture




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

# CNNs on graphs: Graph classification

- ChebNet architecture

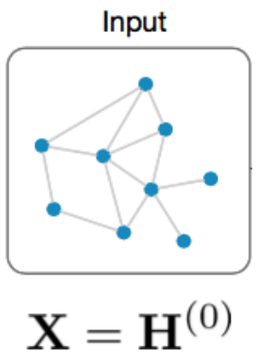



Defferrard et al., "Convolutional neural networks on graphs with fast localized spectral filtering," NIPS, 2016.

# CNNs on graphs: Node classification

- GCN architecture

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

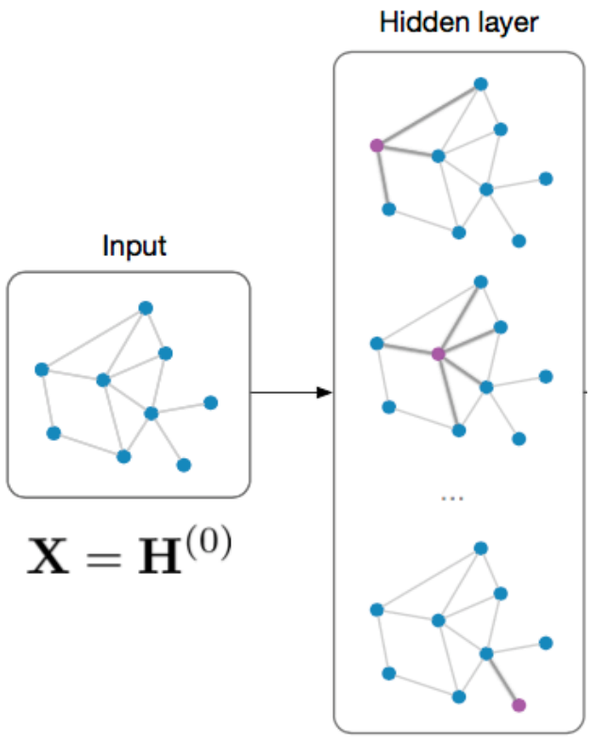



$$\mathbf{X} = \mathbf{H}^{(0)}$$

# CNNs on graphs: Node classification

- GCN architecture

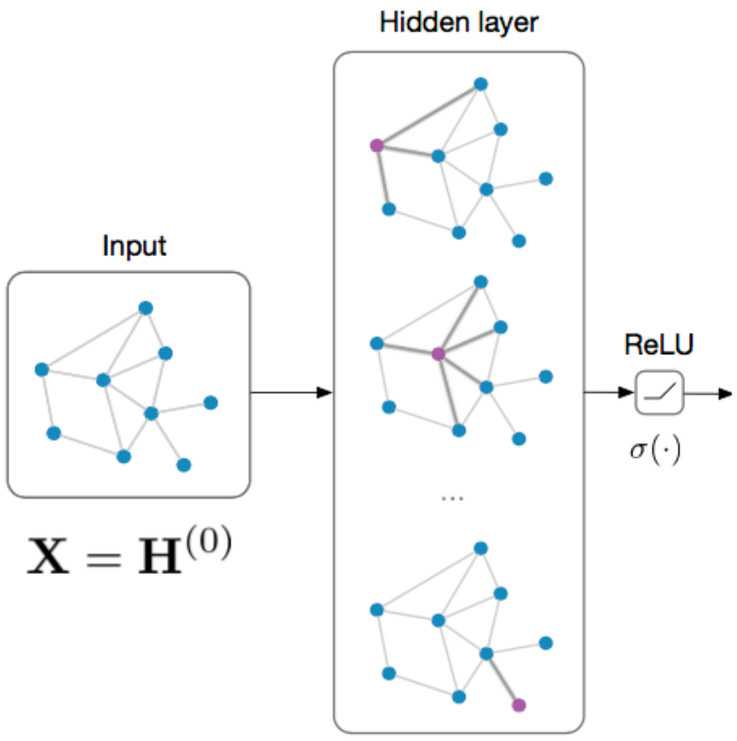
$$\hat{g}_{\theta^{(k+1)}}(L)\Big(\mathrm{ReLU}(\hat{g}_{\theta^{(k)}}(L)f)\Big)$$



# CNNs on graphs: Node classification

- GCN architecture

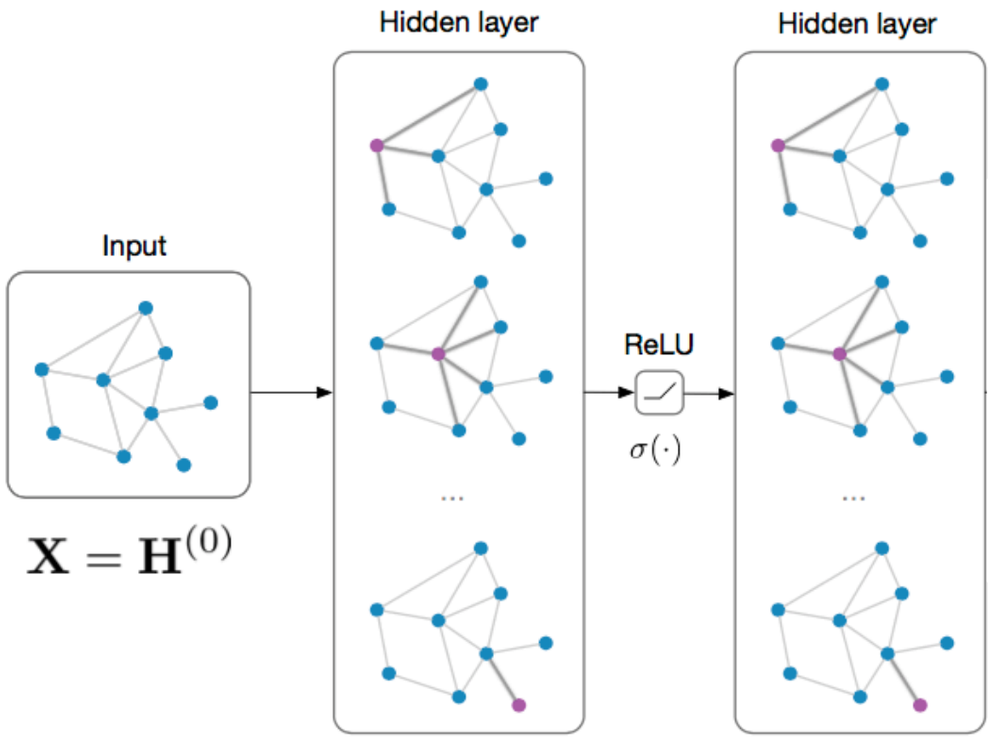
$$\hat{g}_{\theta^{(k+1)}}(L)\left(\mathrm{ReLU}(\hat{g}_{\theta^{(k)}}(L)f)\right)$$



# CNNs on graphs: Node classification

- GCN architecture

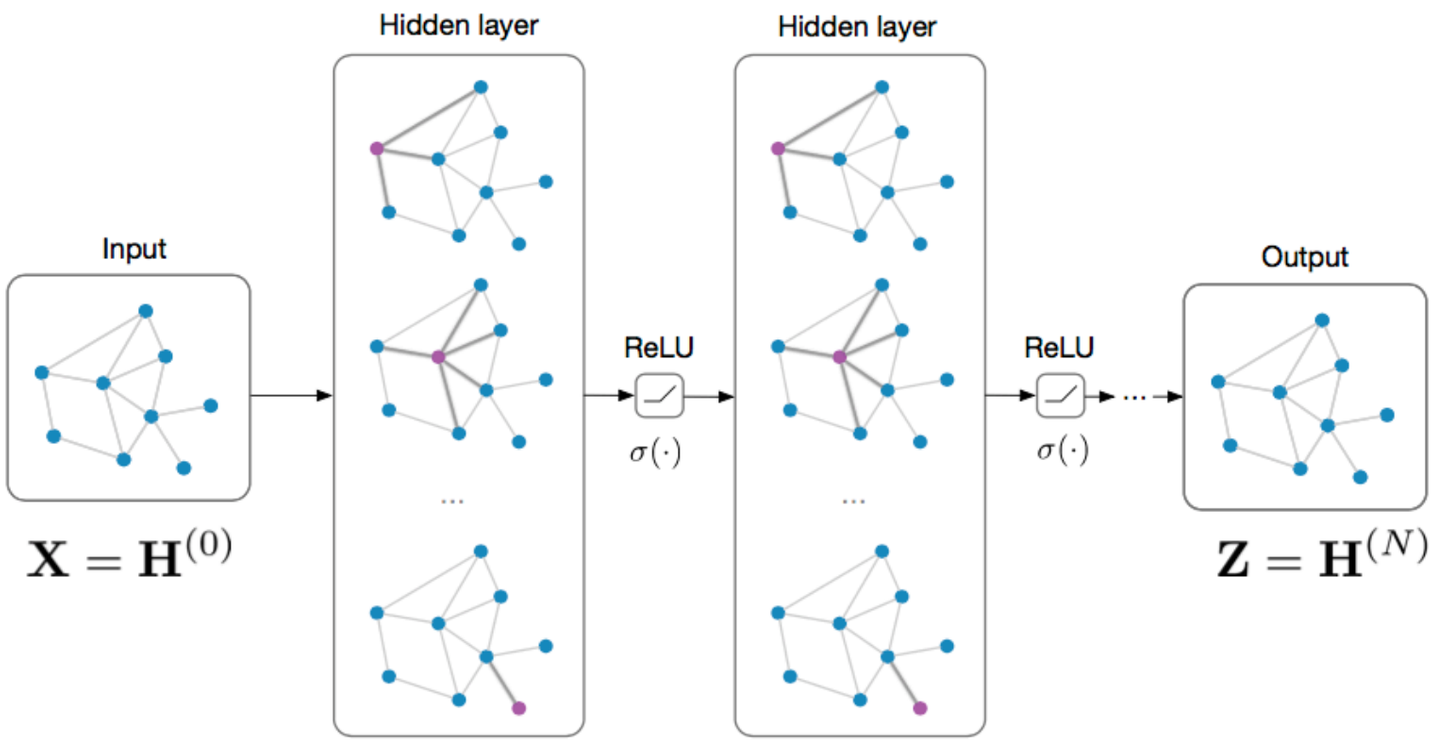
$$\hat{g}_{\theta^{(k+1)}}(L)\Big(\mathrm{ReLU}(\hat{g}_{\theta^{(k)}}(L)f)\Big)$$



# CNNs on graphs: Node classification

- GCN architecture

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



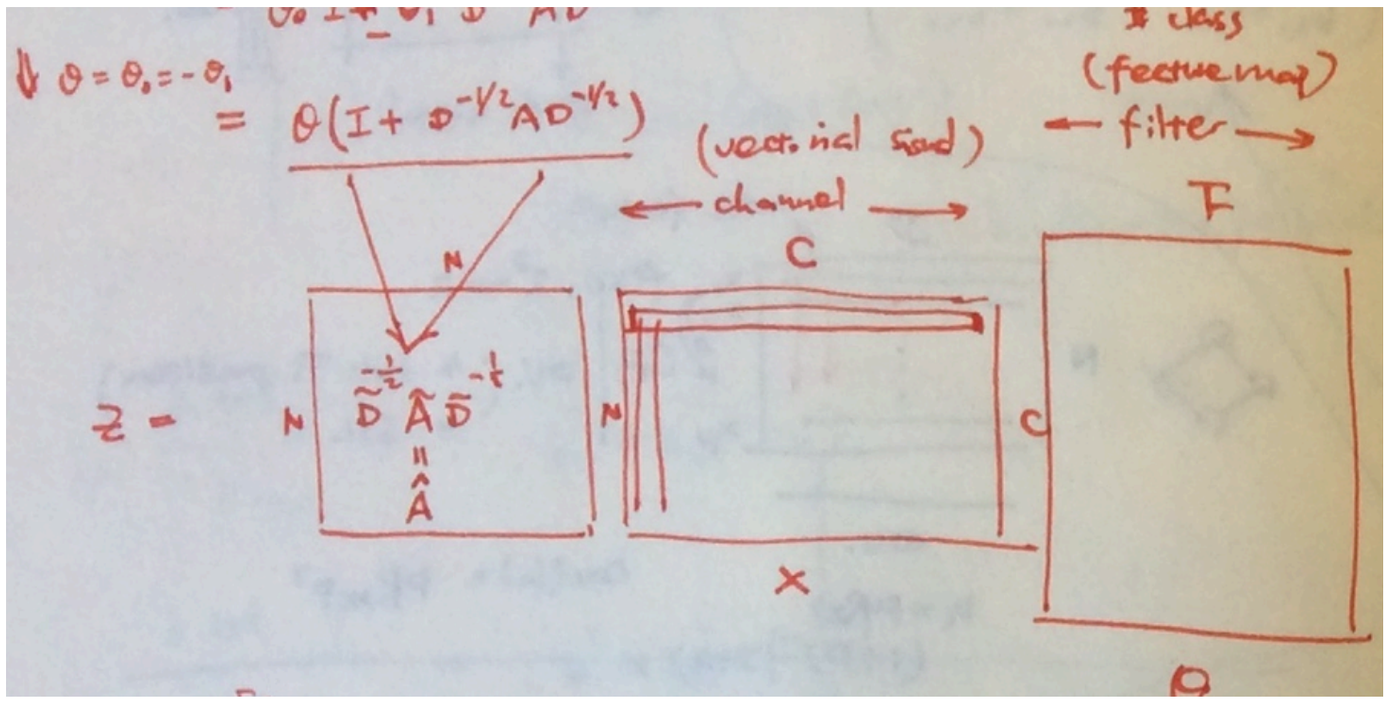

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

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

# CNNs on graphs: Node classification

- GCN architecture

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




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

# Implementing CNNs on graphs

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

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

label prediction (final layer node representation)

label groundtruth

set of labelled (training) nodes

  - 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

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

**label prediction (final layer node representation)**

**label groundtruth**

**set of labelled (training) nodes**

  - 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?

# Implementing CNNs on graphs

- Edge-level task
  - cross-entropy loss function for link prediction

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

**label prediction (produced by final layer representations of pairs of nodes via a similarity function)**

**label groundtruth**

**set of labelled (training) edges**

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

# Implementing CNNs on graphs

- Graph-level task
  - cross-entropy loss function for graph classification

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

**label prediction (produced by final layer representations of all nodes via e.g. global pooling)**

**label groundtruth**

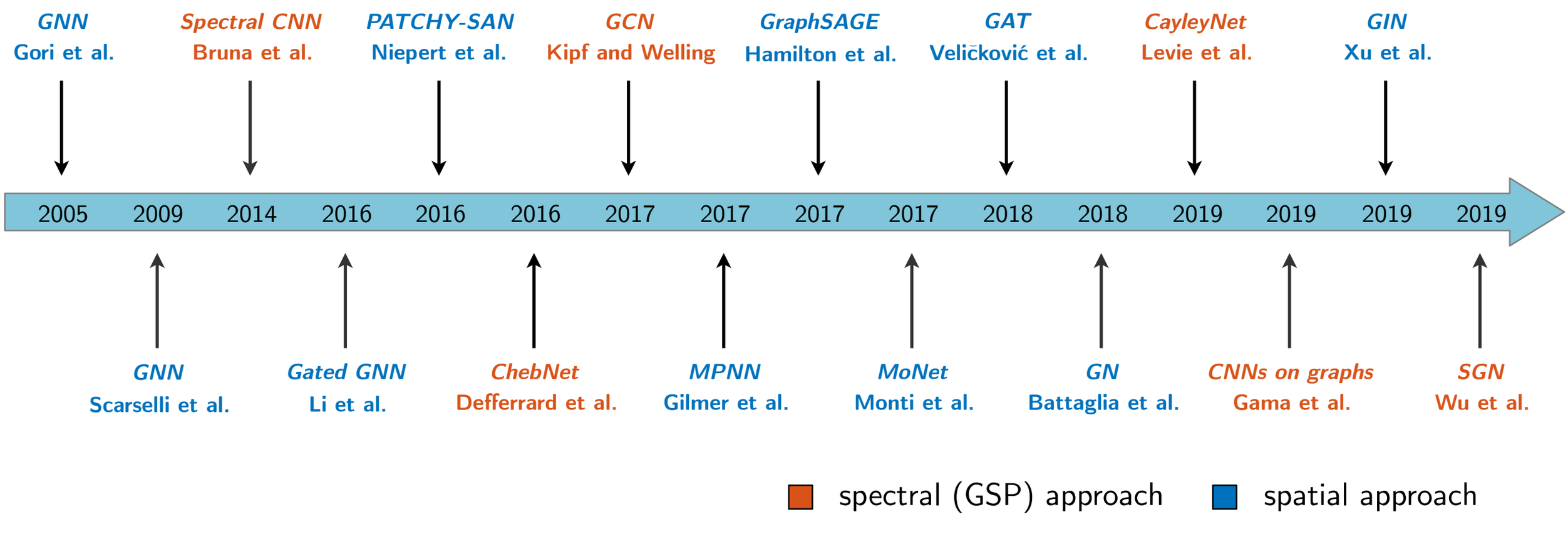
**set of labelled (training) graphs**

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

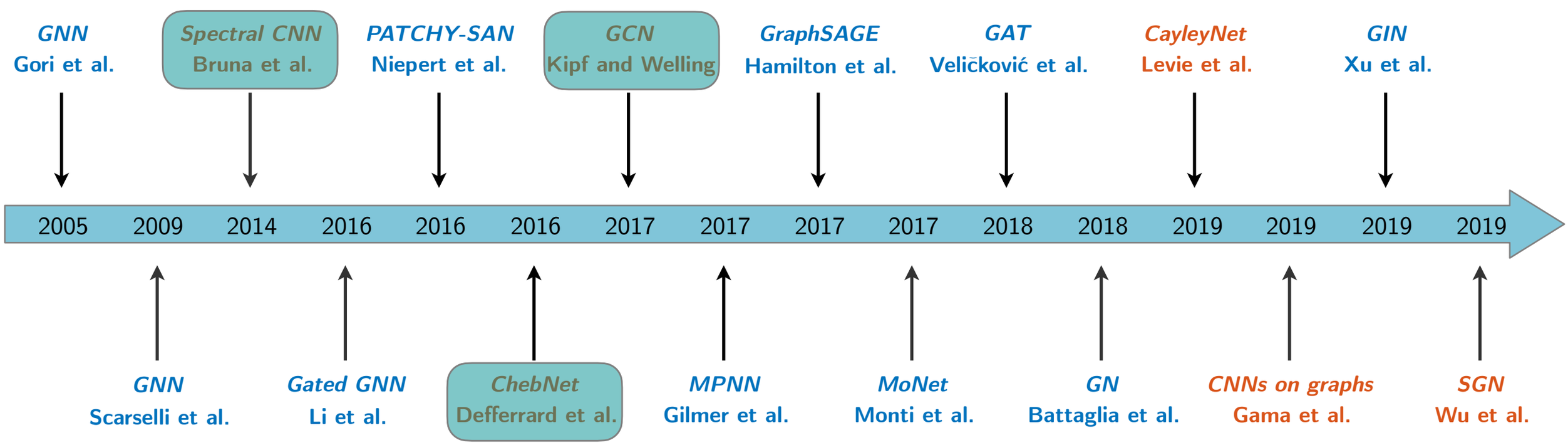
# Lecture 3

- Machine learning on graphs: Overview
- Convolutional neural networks on graphs
- Message passing neural networks
- Recent developments & Applications

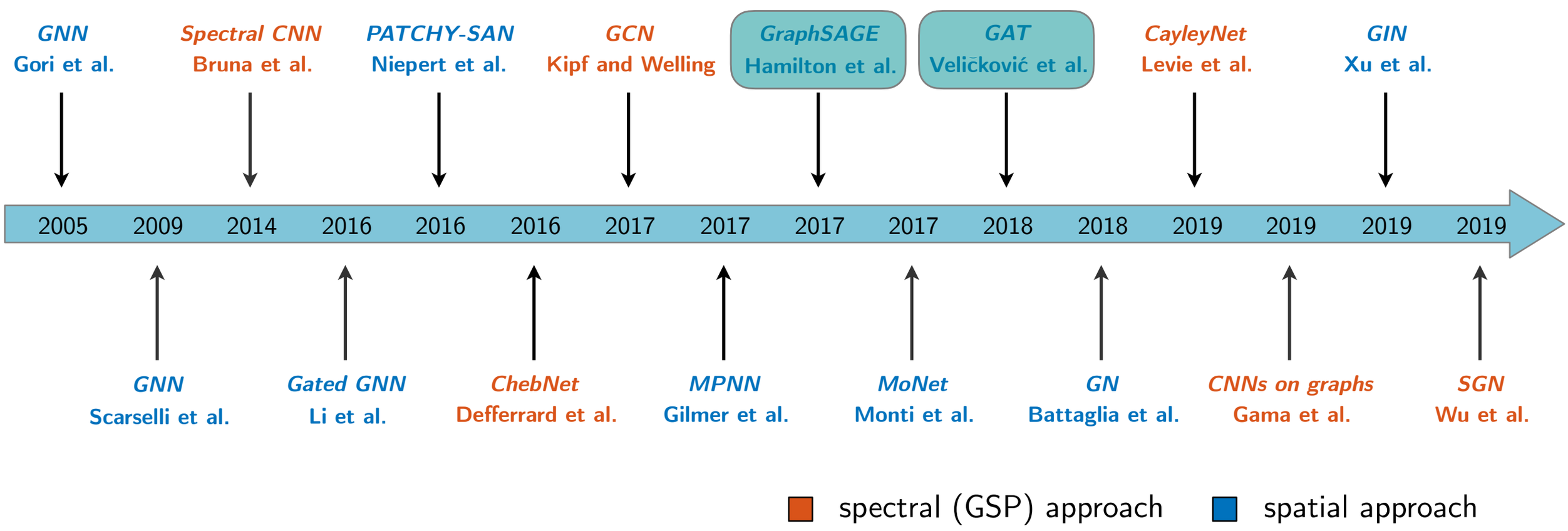
# GNNs - A historical timeline



# GNNs - A historical timeline

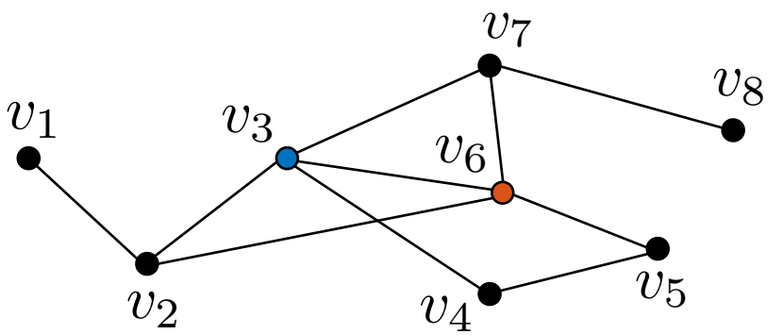


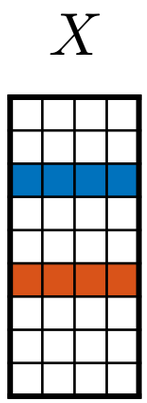
# GNNs - A historical timeline

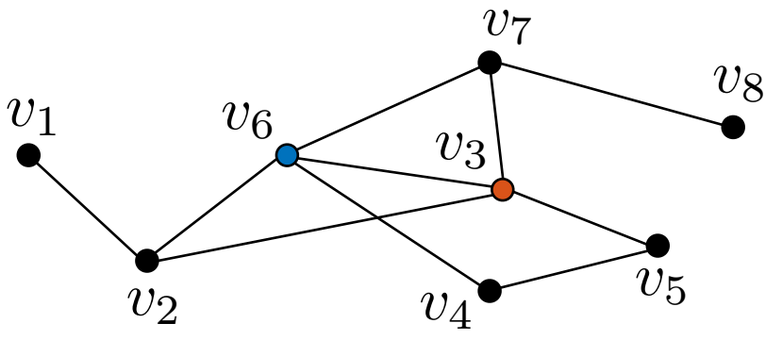


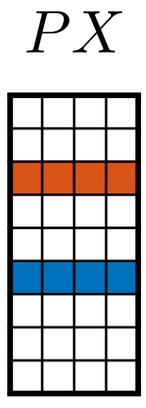
# A look into invariance

- Permutation invariance: function invariant w.r.t. permutation (node re-ordering)



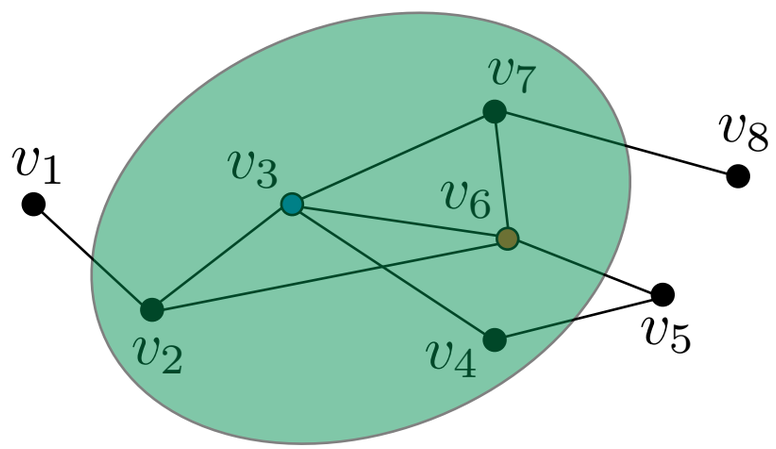



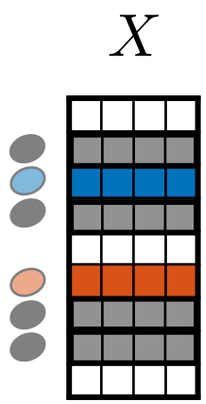



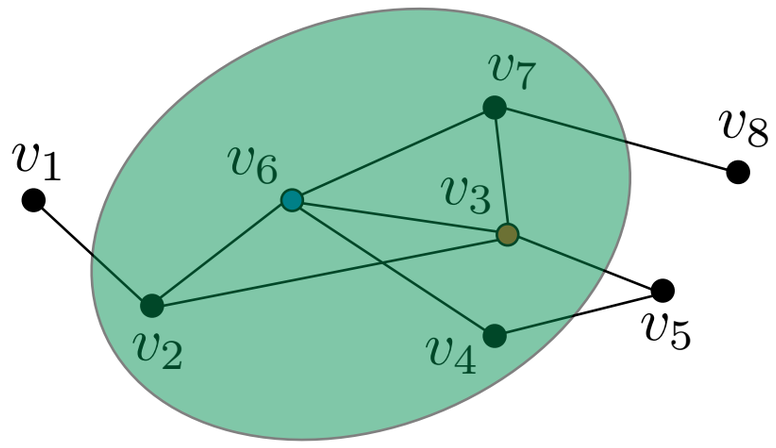



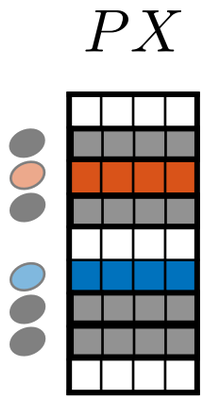
# A look into invariance

- Permutation invariance: function invariant w.r.t. permutation (node re-ordering)



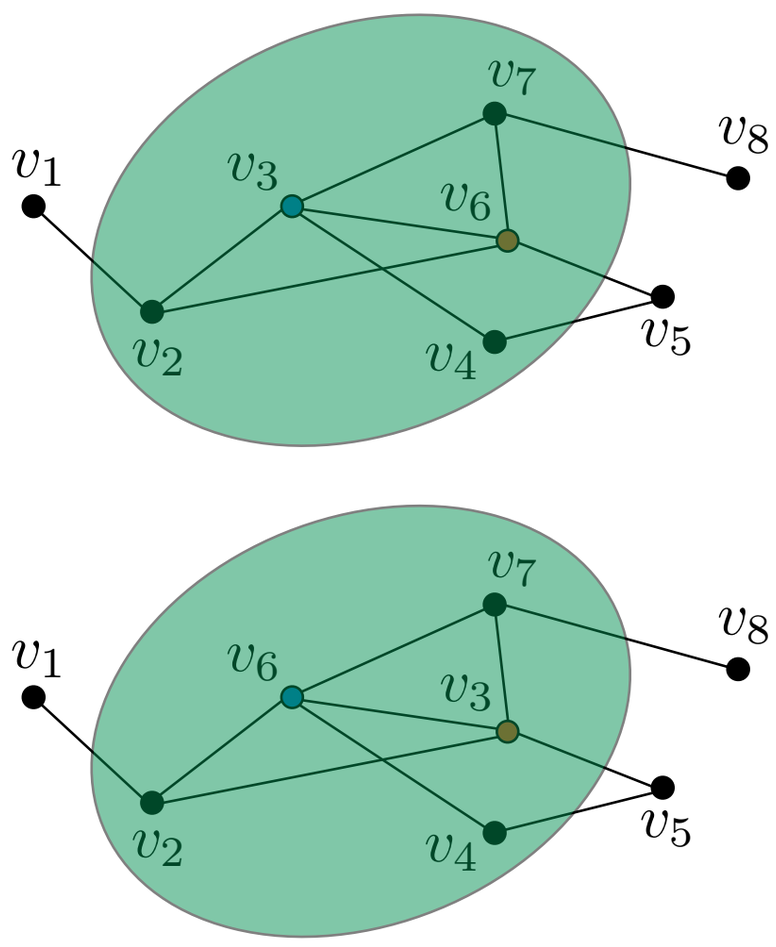



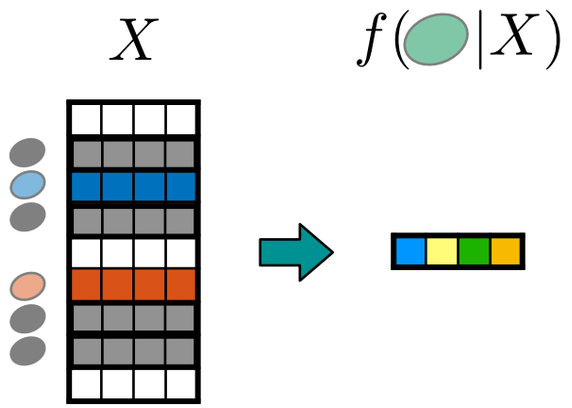



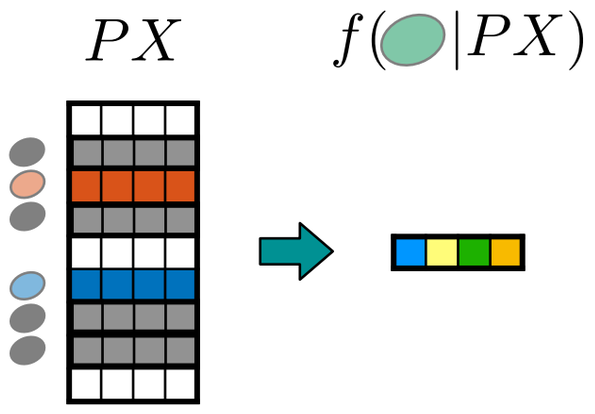



# A look into invariance

- Permutation invariance: function invariant w.r.t. permutation (node re-ordering)

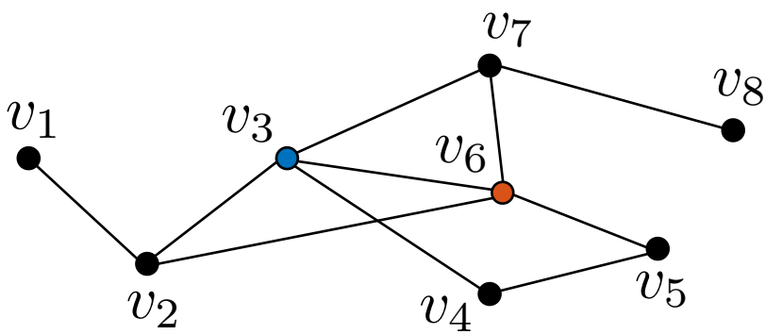


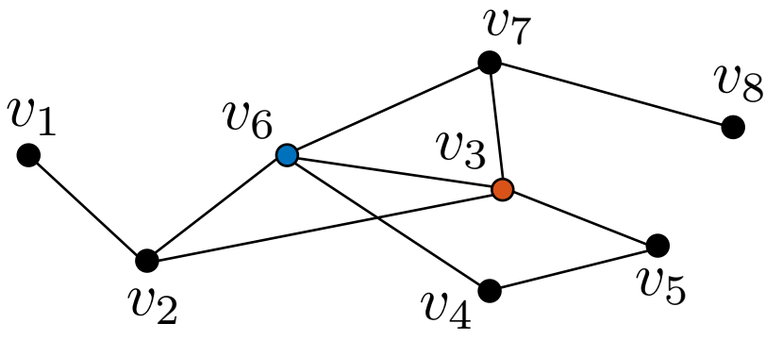



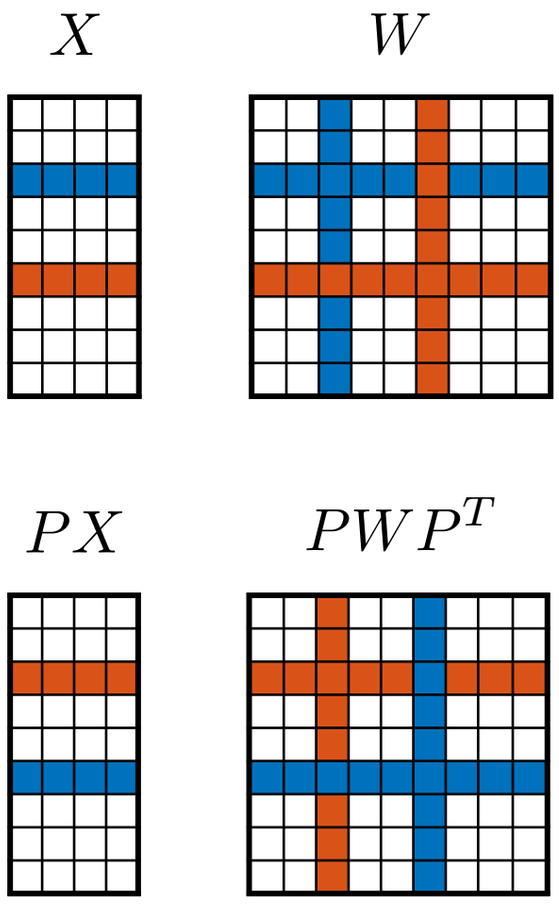



# A look into invariance

- Permutation equivariance: function equivariant w.r.t. permutation (permutation of input $\Rightarrow$ same permutation of output)

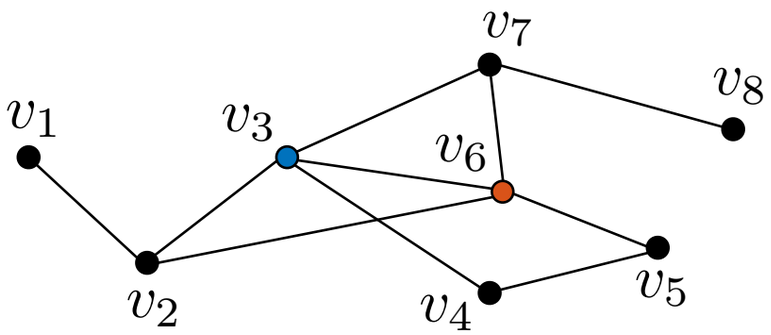


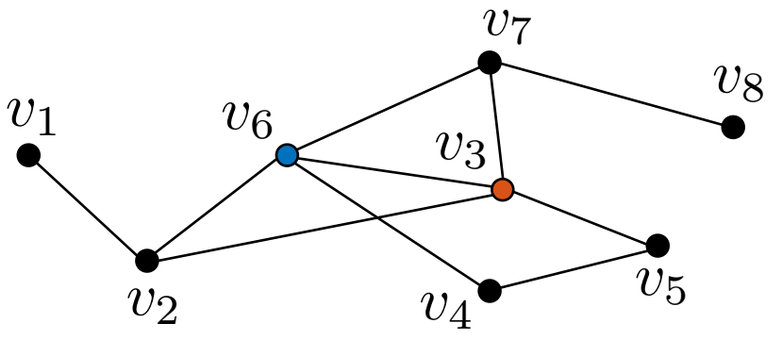



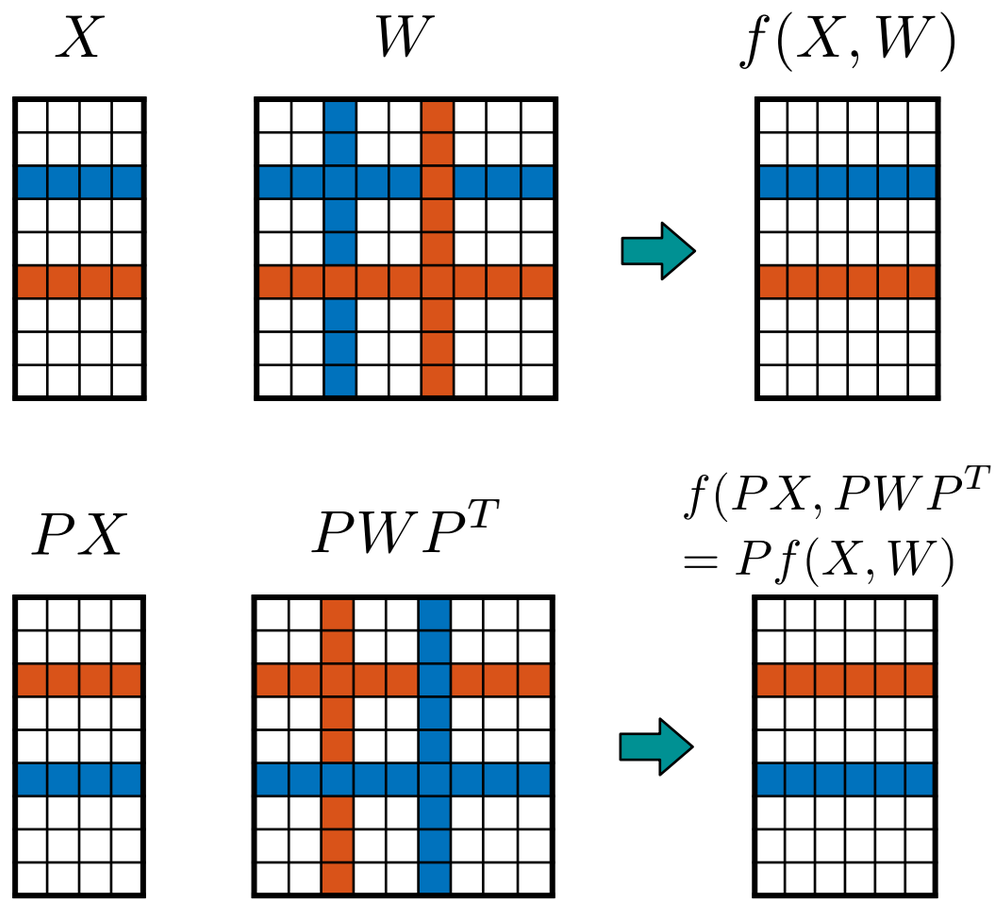



# A look into invariance

- Permutation equivariance: function equivariant w.r.t. permutation (permutation of input $\Rightarrow$ same permutation of output)

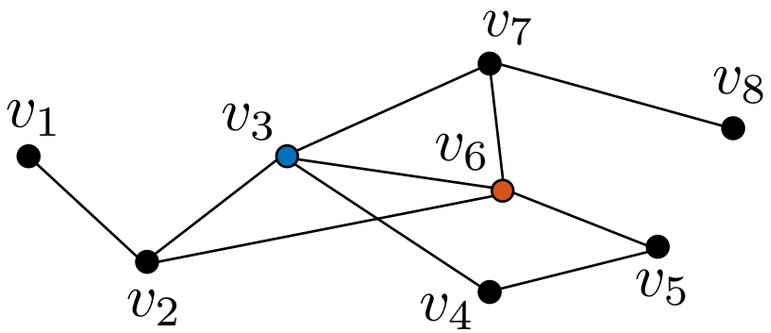


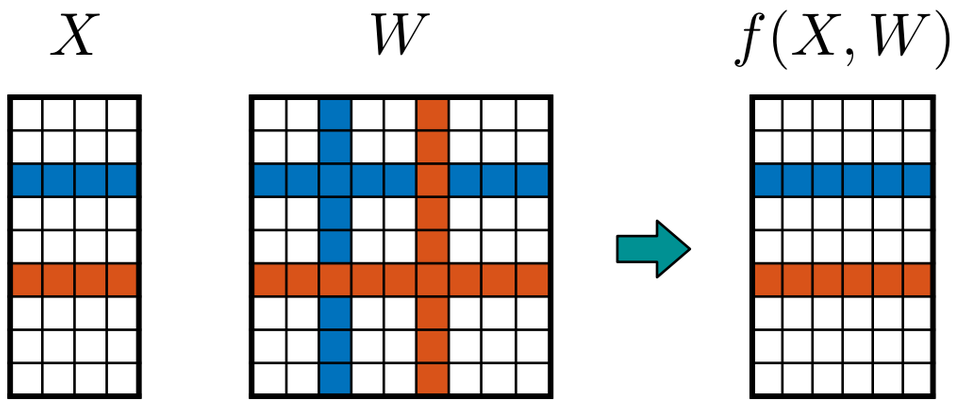



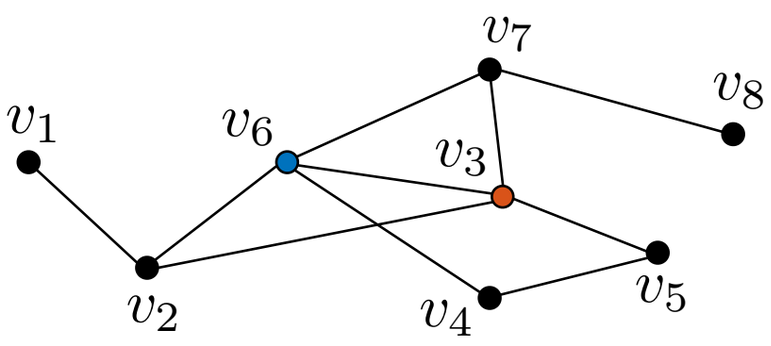



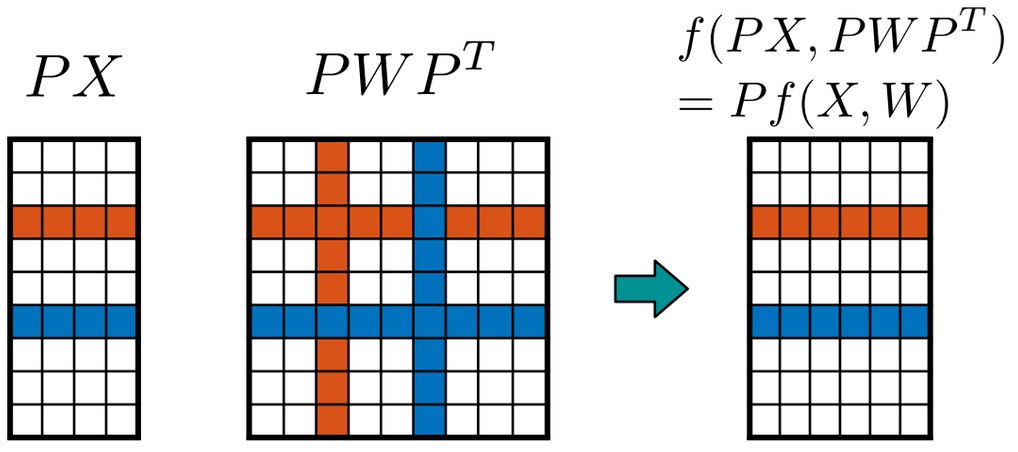
# A look into invariance

- Permutation equivariance: function equivariant w.r.t. permutation (permutation of input $\Rightarrow$ same permutation of output)




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

# GNNs - A spatial approach

- Recall 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 ⇒ $(x * g)(v) = \sum_{v' \in \mathcal{V}} x(v') g(v, v')$

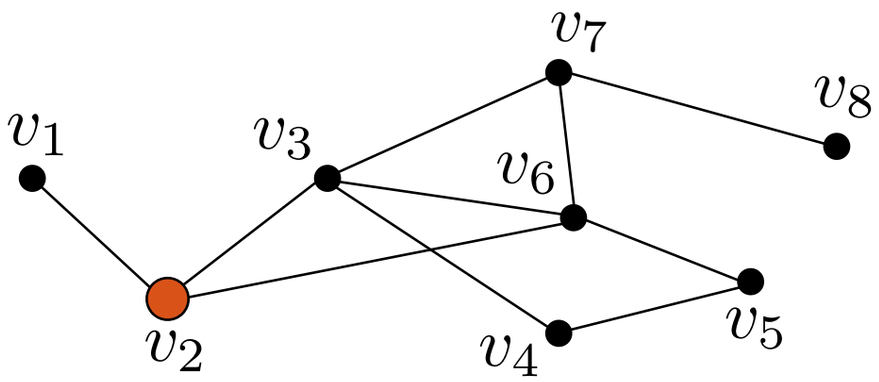
# GNNs - A spatial approach

- Recall 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




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

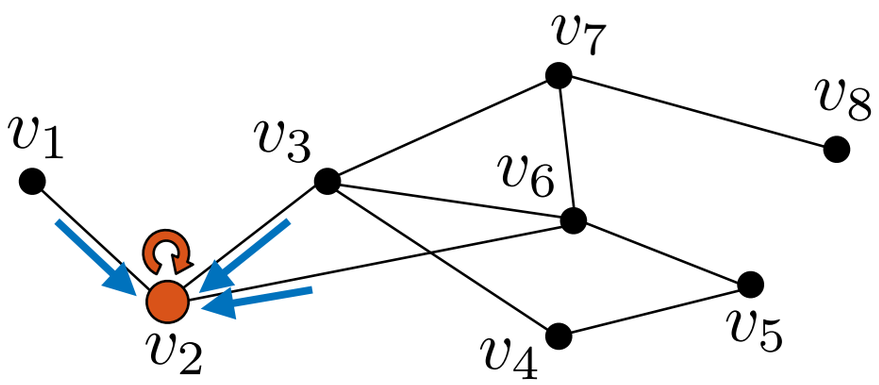
# GNNs - A spatial approach

- Recall 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

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

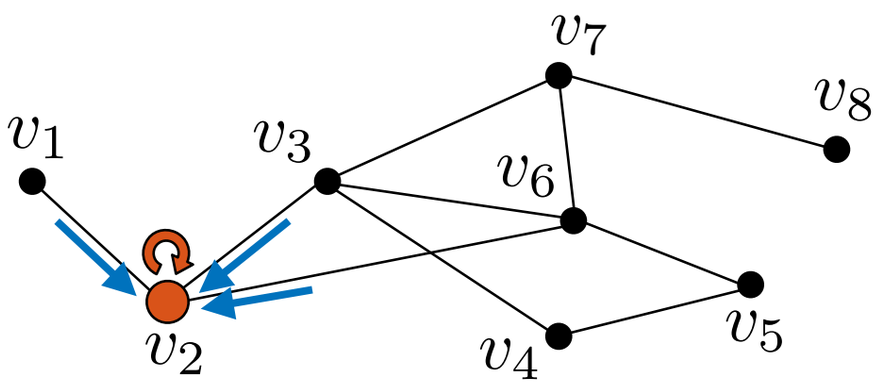
# GNNs - A spatial approach

- Recall 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

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

permutation invariant

local neighbourhood

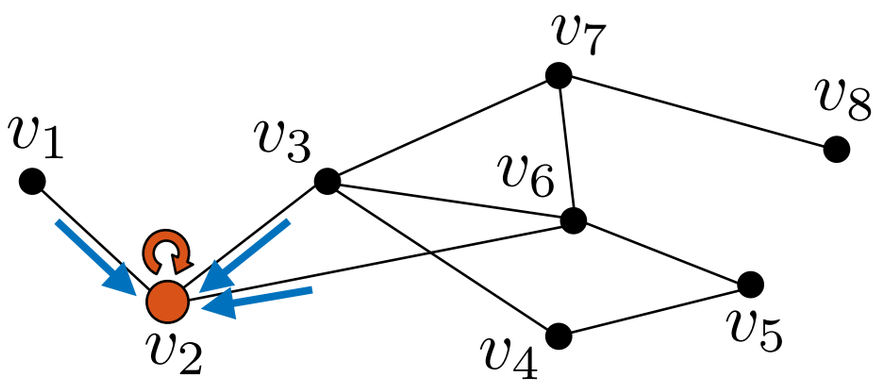
# GNNs - A spatial approach

- Recall 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

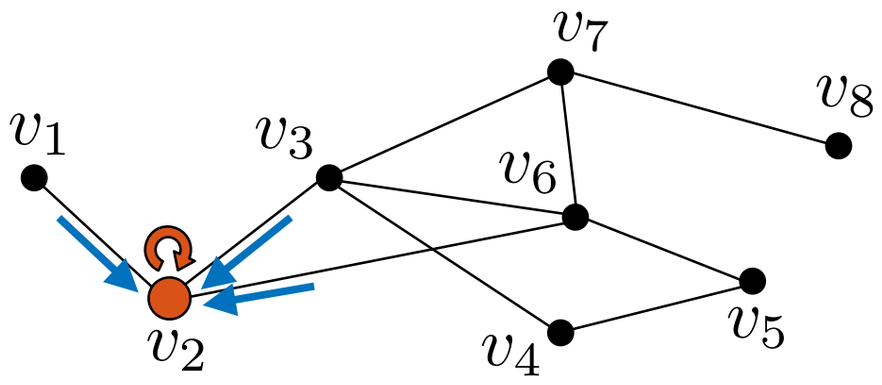



self information neighbour information

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

learnable parameters permutation invariant local neighbourhood

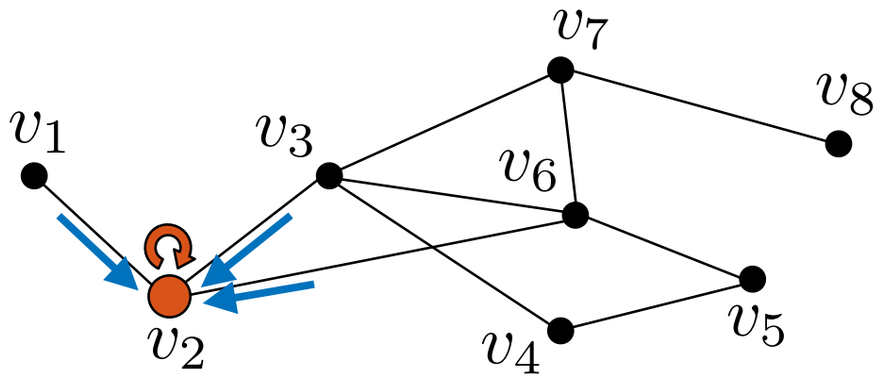
# Message passing neural networks (MPNNs)




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

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

# Message passing neural networks (MPNNs)

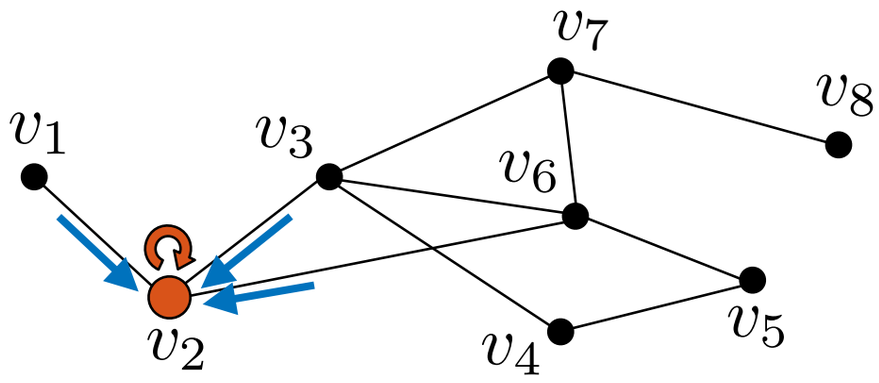



**message function**

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

- nodes exchange **messages** with local neighbours

# Message passing neural networks (MPNNs)



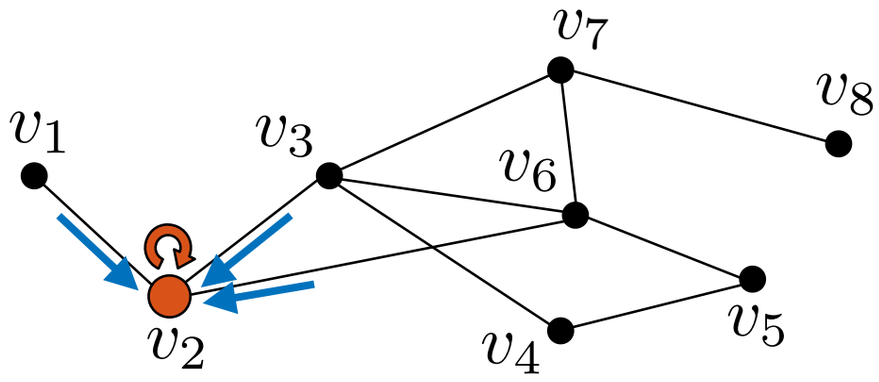

update function message function

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

aggregator function

- nodes exchange **messages** with local neighbours
- each node **updates** its representation by **aggregating** messages from neighbours

# Message passing neural networks (MPNNs)




update function message function

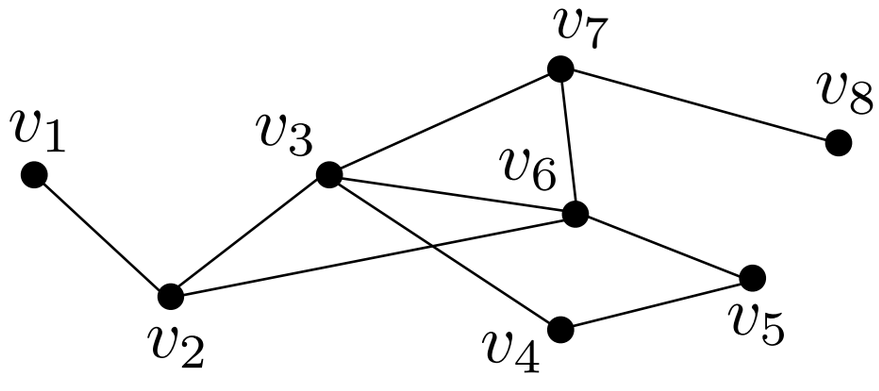
$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

aggregator function

- nodes exchange **messages** with local neighbours
- each node **updates** its representation by **aggregating** messages from neighbours
- functions are differentiable and parameters are learned by minimising loss of downstream task
- key difference between architectures: how nodes aggregate information from neighbours and across layers

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

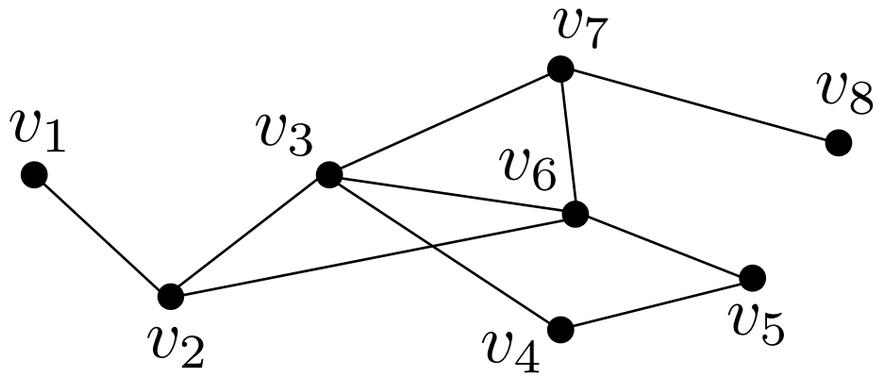
# MPNNs - A simple example




weighted sum  sum  $h_j^l$

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

# MPNNs - A simple example

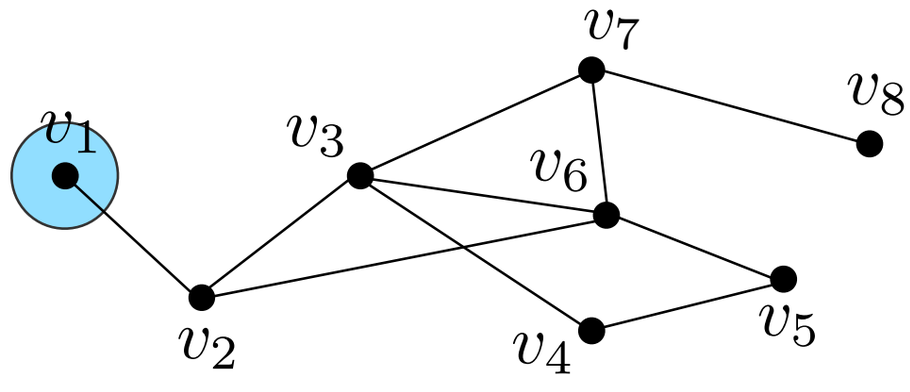



weighted sum sum $h_j^l$

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

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

# MPNNs - A simple example




**one message passing layer**

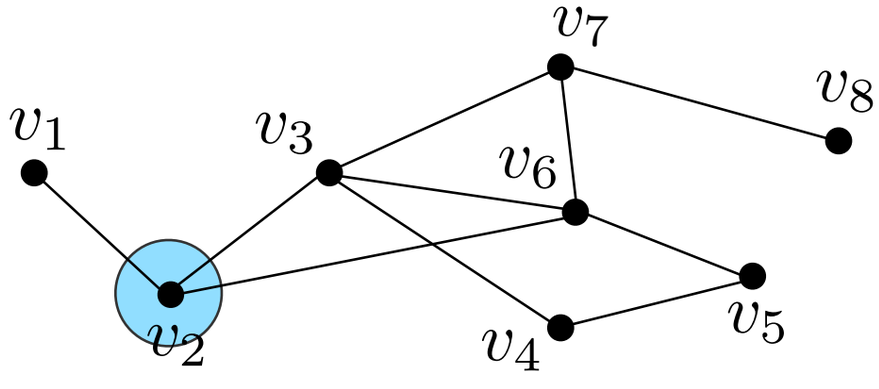
$$h_1^{l+1} = \theta_0^l h_1^l + \theta_1^l h_2^l$$

weighted sum  sum  $h_j^l$

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

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

# MPNNs - A simple example




**one message passing layer**

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

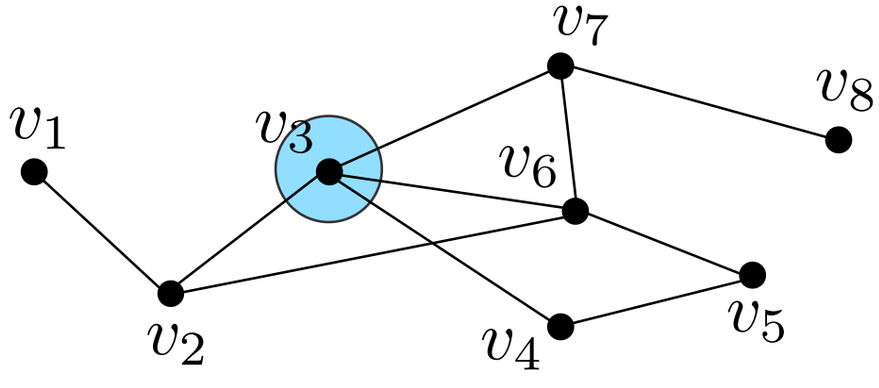
$$h_2^{l+1} = \theta_0^l h_2^l + \theta_1^l (h_1^l + h_3^l + h_6^l)$$

**weighted sum** **sum** $h_j^l$

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

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

# MPNNs - A simple example




**one message passing layer**

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

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

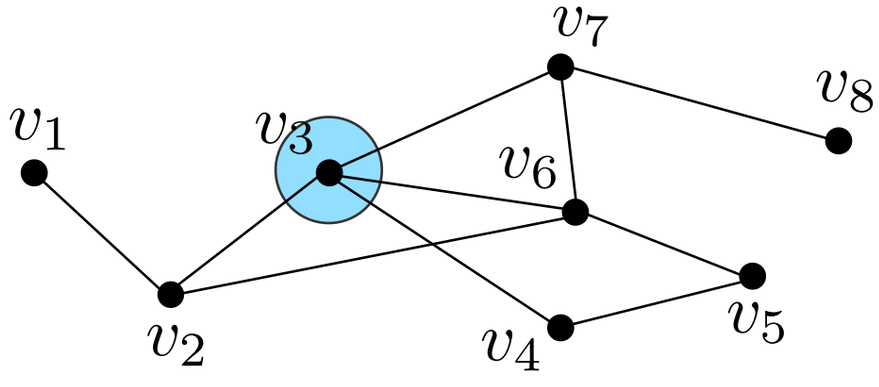
$$h_3^{l+1} = \theta_0^l h_3^l + \theta_1^l (h_2^l + h_4^l + h_6^l + h_7^l)$$

**weighted sum** **sum** $h_j^l$

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

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

# 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**

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

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

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

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

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

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

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

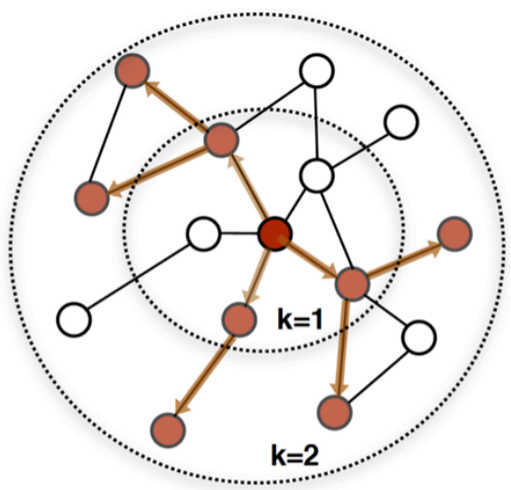
$$h_8^{l+1} = \theta_0^l h_8^l + \theta_1^l h_7^l$$

**final output after L layers**

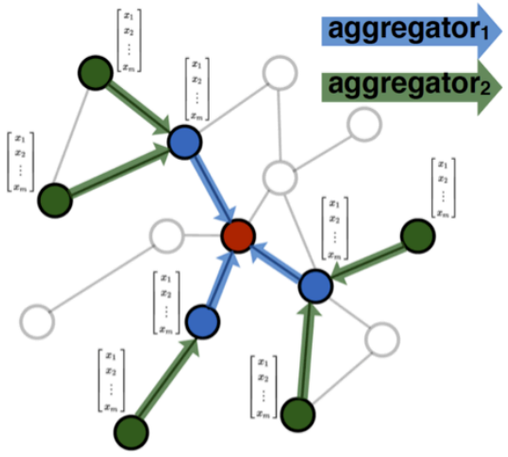
$$Z = \{h_1^L, h_2^L, h_3^L, h_4^L, h_5^L, h_6^L, h_7^L, h_8^L\}^T$$

# GraphSAGE

- Idea: **sample** neighbours from each hop of neighbourhood (i.e., building a "computational" graph) for improved scalability and robustness




1. Sample neighborhood




2. Aggregate feature information from neighbors

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

$$h_i^{l+1} = \sigma\left(W^l \cdot \text{concat}(h_i^l, \bigoplus_{j \in \mathcal{N}_i} h_j^l)\right)$$

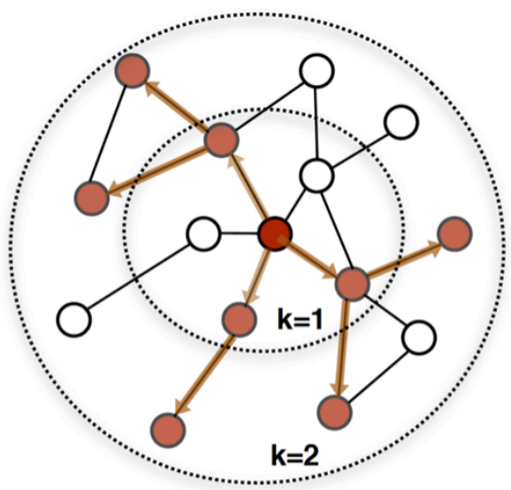
**learnable parameters**

**sampled neighbours**

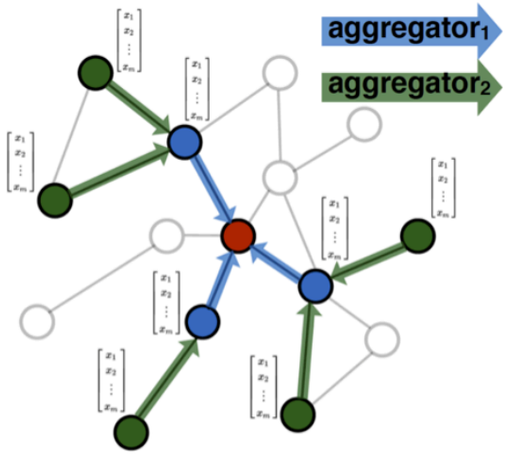
Hamilton et al., "Inductive representation learning on large graphs," NIPS, 2017.

# GraphSAGE

- Idea: **sample** neighbours from each hop of neighbourhood (i.e., building a "computational" graph) for improved scalability and robustness
- **Concatenation** of self and neighbour embeddings acts as implicit "skip connections" to prevent loss of self information




1. Sample neighborhood




2. Aggregate feature information from neighbors

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

concatenation

$$h_i^{l+1} = \sigma\left(W^l \cdot \text{concat}(h_i^l, \bigoplus_{j \in \mathcal{N}_i} h_j^l)\right)$$

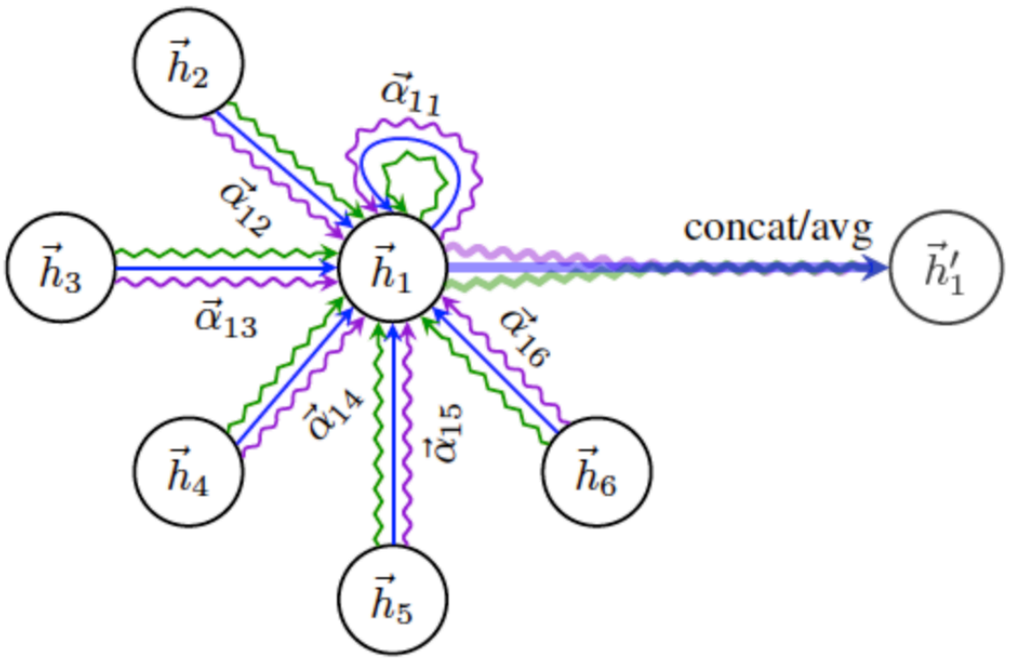
**learnable parameters**

**sampled neighbours**

Hamilton et al., "Inductive representation learning on large graphs," NIPS, 2017.

# Graph attention network (GAT)

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




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

**relative importance**

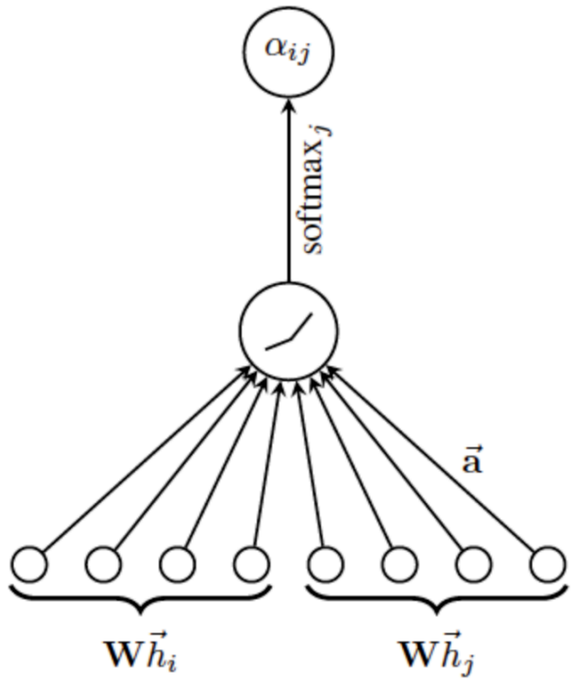
$$h_i^{l+1} = \sigma\left(\alpha_{ii} W^l h_i^l + \sum_{j \in \mathcal{N}_i} \alpha_{ij} W^l h_j^l\right)$$

**learnable parameters**

# Graph attention network (GAT)

- Idea: learn **relative importance** of neighbours in aggregation
- An **attention** function weighs 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

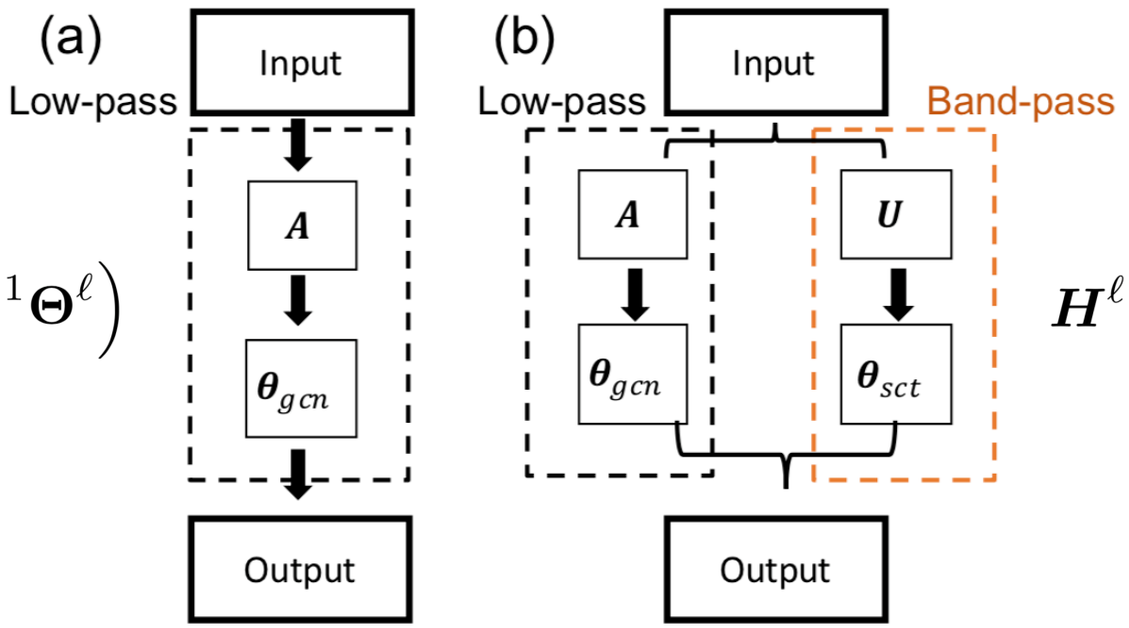
# Lecture 3

- Machine learning on graphs: Overview
- Convolutional neural networks on graphs
- Message passing neural networks
- Recent developments & Applications

# Over-smoothing

- GCN implements low-pass filtering (which can lead to "over-smoothing")
- Idea: combine low-pass and band-/high-pass filtering

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



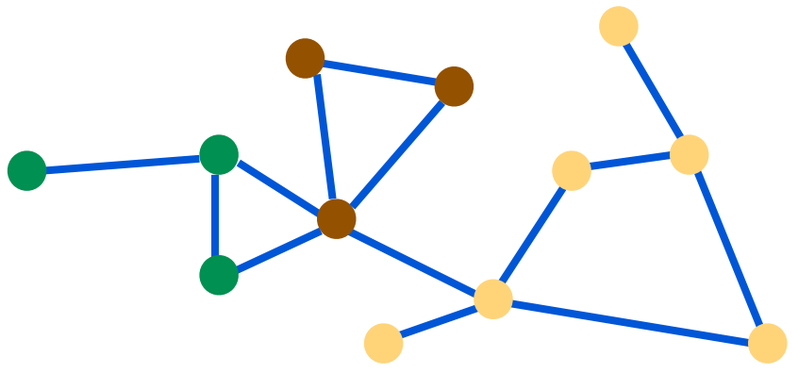

$$\boldsymbol{H}^{\ell} = \sigma\left(\boldsymbol{U}_p\boldsymbol{H}^{\ell-1}\boldsymbol{\Theta}^{\ell}\right)$$

**GCN** **Scattering GCN**

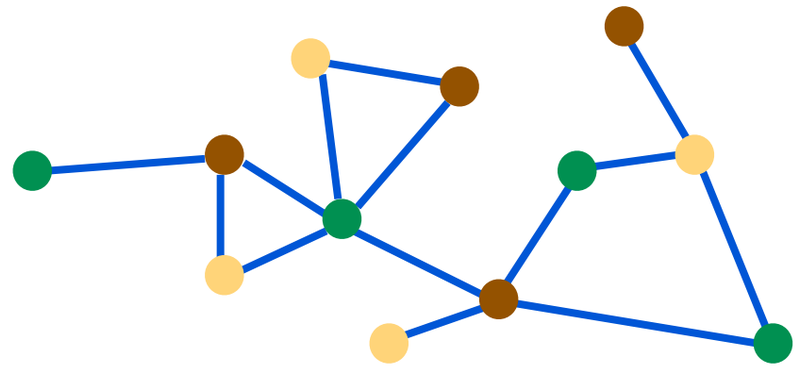
Min et al., "Scattering GCN: Overcoming oversmoothness in graph convolutional networks," NeurIPS, 2020.

# Label homophily

- Heterophily of node labels poses a challenge



**high homophily**

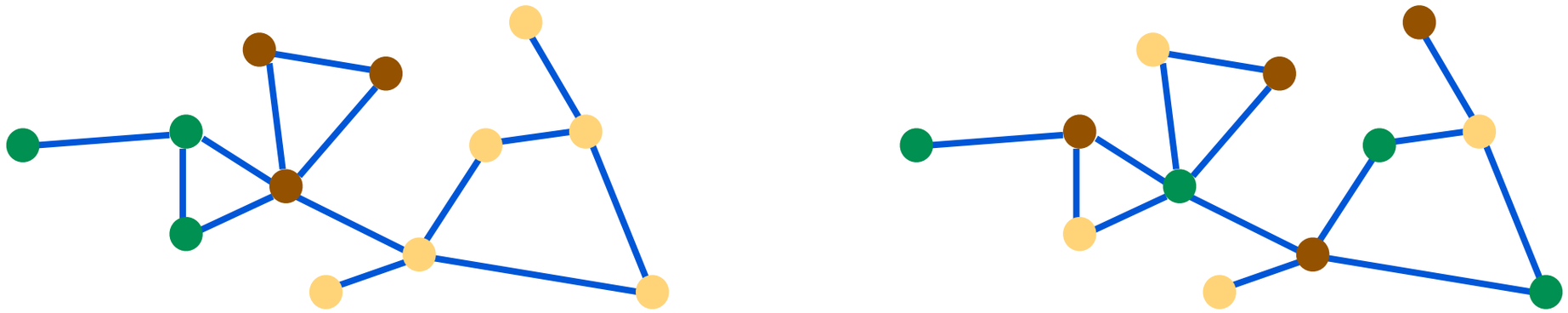


**low homophily**

Zhu et al., "Beyond homophily in graph neural networks: Current limitations and effective designs," NeurIPS, 2020.

# Label homophily

- Heterophily of node labels poses a challenge



high homophily

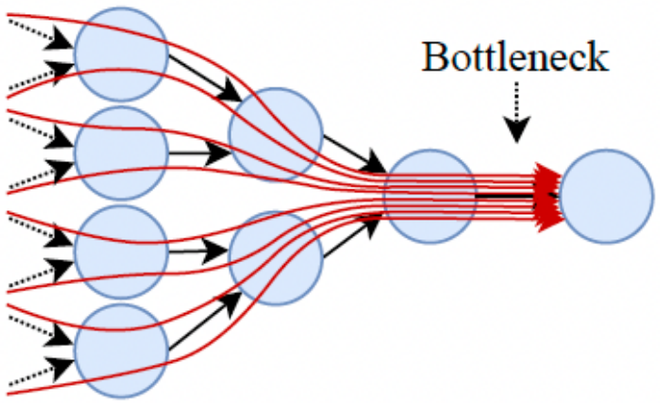
low homophily

- Idea:
  - ego- and neighbour-embedding separation: $\mathbf{r}_v^{(k)} = \texttt{COMBINE}\left(\mathbf{r}_v^{(k-1)}, \texttt{AGGR}(\{\mathbf{r}_u^{(k-1)} : u \in \bar{N}(v)\})\right)$
  - higher-order neighbourhoods: $\mathbf{r}_v^{(k)} = \texttt{COMBINE}\left(\mathbf{r}_v^{(k-1)}, \texttt{AGGR}(\{\mathbf{r}_u^{(k-1)} : u \in N_1(v)\}), \texttt{AGGR}(\{\mathbf{r}_u^{(k-1)} : u \in N_2(v)\}), \ldots\right)$
  - combination of intermediate representations: $\mathbf{r}_v^{(\text{final})} = \texttt{COMBINE}\left(\mathbf{r}_v^{(1)}, \mathbf{r}_v^{(2)}, \ldots, \mathbf{r}_v^{(K)}\right)$

Zhu et al., "Beyond homophily in graph neural networks: Current limitations and effective designs," NeurIPS, 2020.

# Over-squashing

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



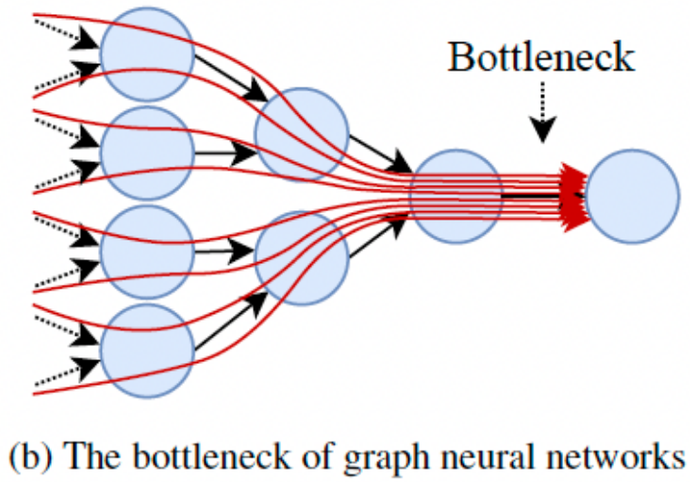

(b) The bottleneck of graph neural networks

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.

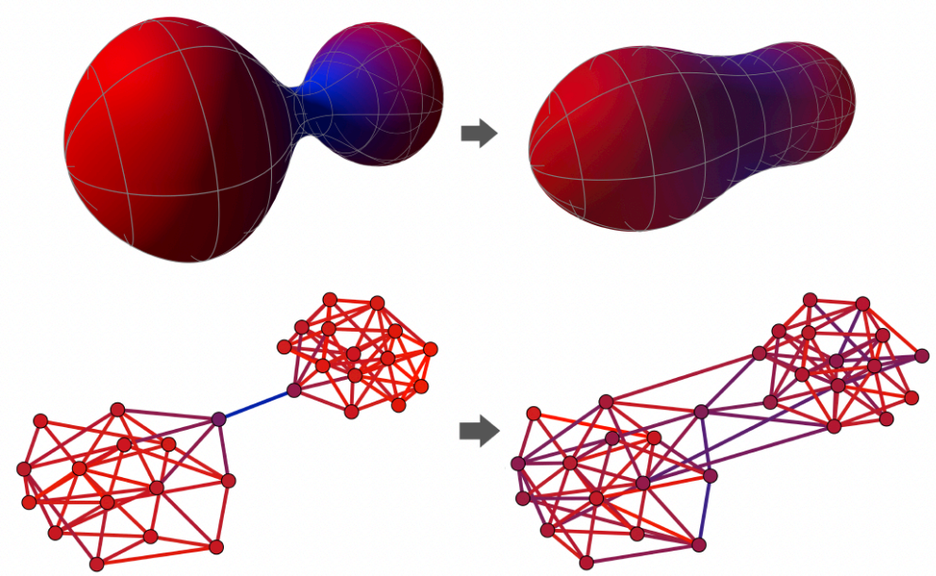
# Over-squashing

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




(b) The bottleneck of graph neural networks

over-squashing caused by
**bottlenecks**

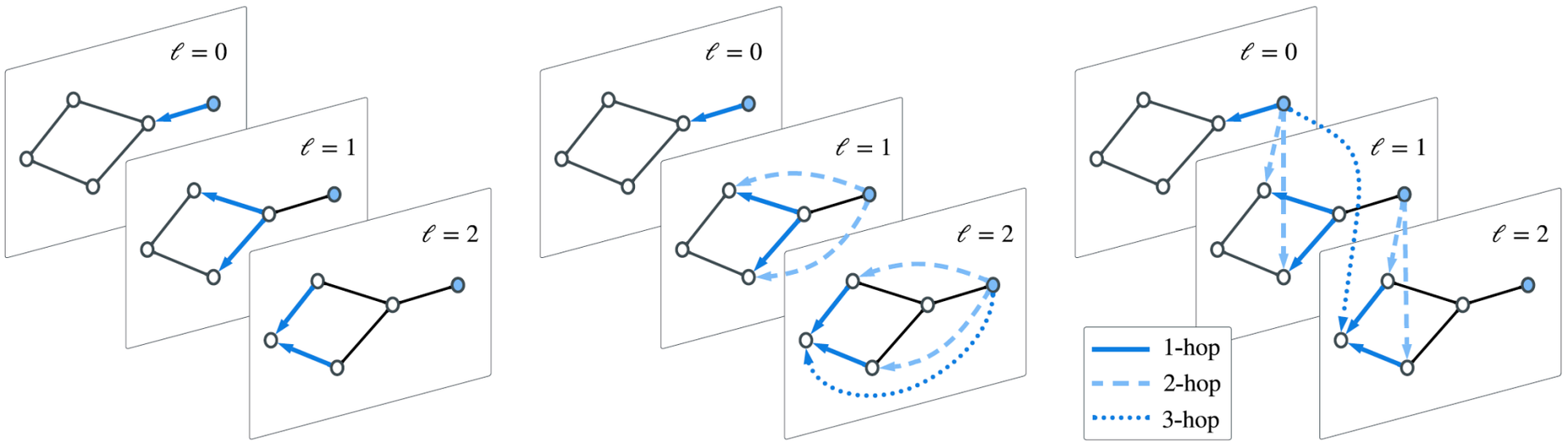


bottlenecks are linked to
**negatively curved** edges

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.

# Adaptive message passing

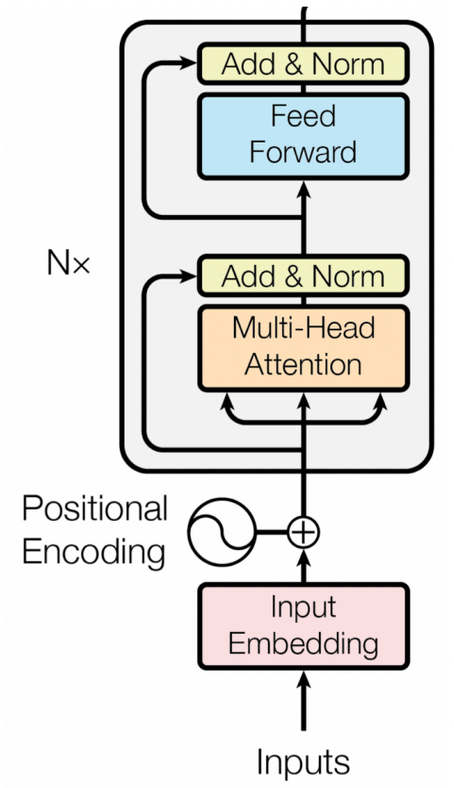
- Idea: modifying the "computational" graph for adaptive message passing



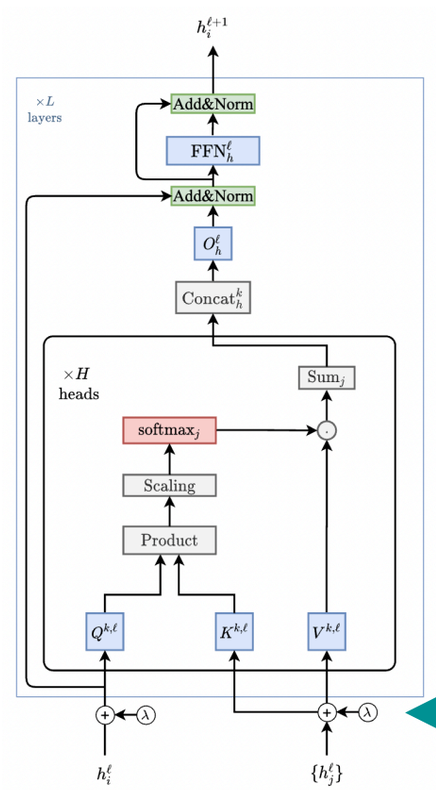

(a) Classical MPNN (b) DRew (c) $\nu$DRew

# Graph transformers

- Idea: build transformers on graphs by generalising GAT to global attention




**transformers**




**graph transformers**

**positional encodings retaining graph info**

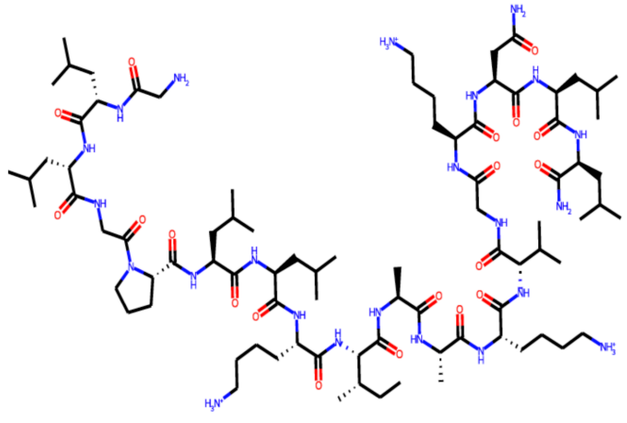
# Long-range interactions

- Which tasks require long-range interactions and how to measure them?

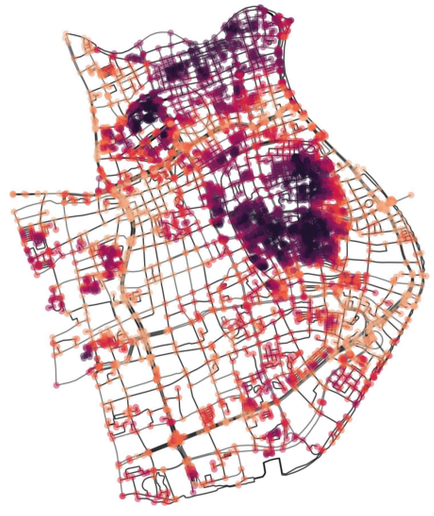
Dwivedi et al., "Long range graph benchmark," NeurIPS, 2022.
Liang et al., "Towards quantifying long-range interactions in graph machine learning," NeurIPS Workshop, 2025.
Bamberger et al., "On measuring long-range interactions in graph neural networks," ICML, 2025.

# Long-range interactions

- Which tasks require long-range interactions and how to measure them?



**graph classification**

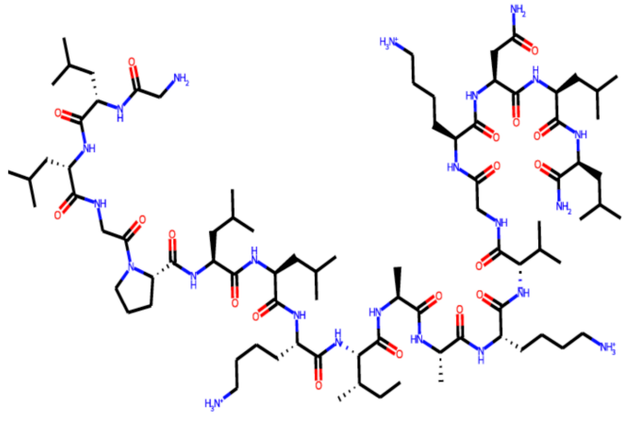


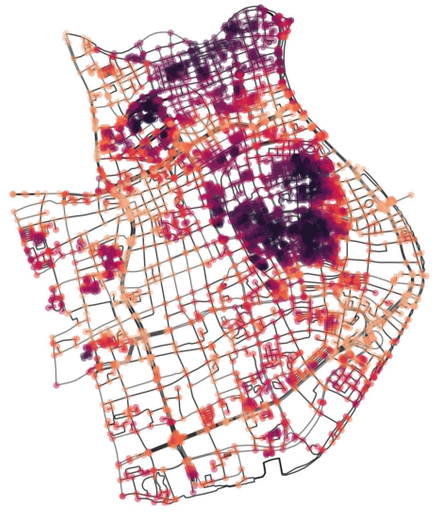
**node classification**

Dwivedi et al., "Long range graph benchmark," NeurIPS, 2022.
Liang et al., "Towards quantifying long-range interactions in graph machine learning," NeurIPS Workshop, 2025.
Bamberger et al., "On measuring long-range interactions in graph neural networks," ICML, 2025.

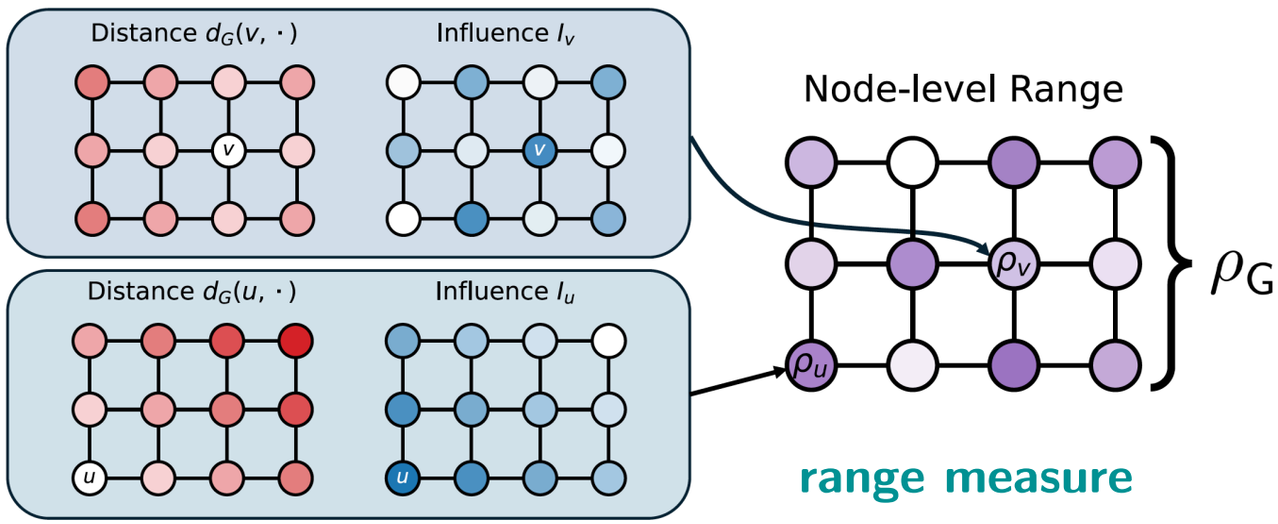
# Long-range interactions

- Which tasks require long-range interactions and how to measure them?





**graph classification** **node classification**

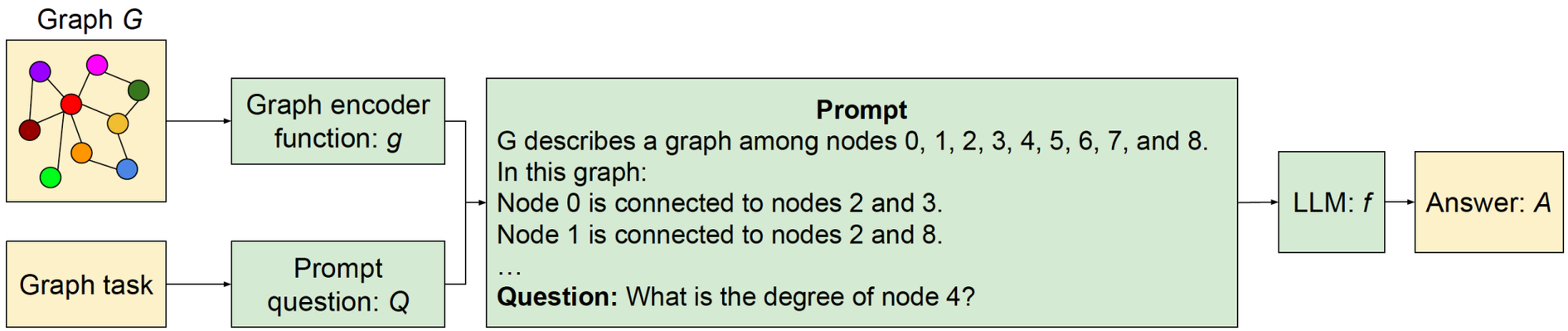



**range measure**


Dwivedi et al., "Long range graph benchmark," NeurIPS, 2022.
Liang et al., "Towards quantifying long-range interactions in graph machine learning," NeurIPS Workshop, 2025.
Bamberger et al., "On measuring long-range interactions in graph neural networks," ICML, 2025.

# LLMs for graph learning

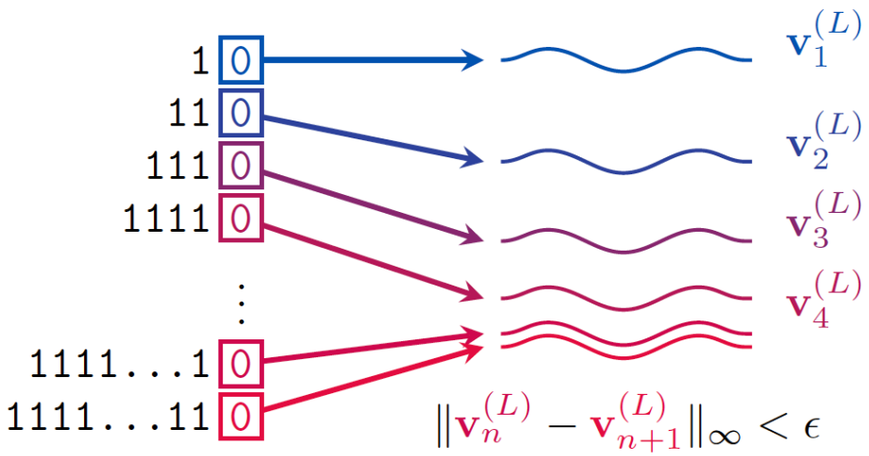
- Can LLMs understand graph-structured data?



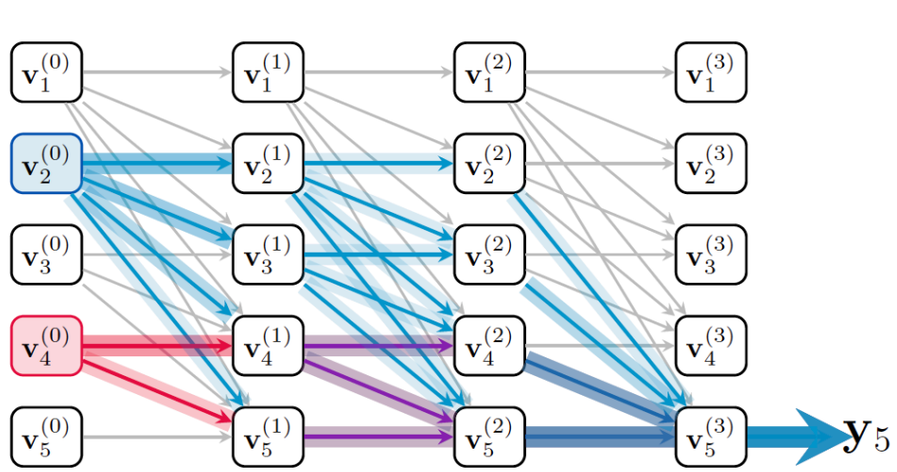

Fatemi et al., "Talk like a graph: Encoding graphs for large language models," ICLR, 2024.

# Graph learning for LLMs

- Can insights from graph learning help understand LLMs?

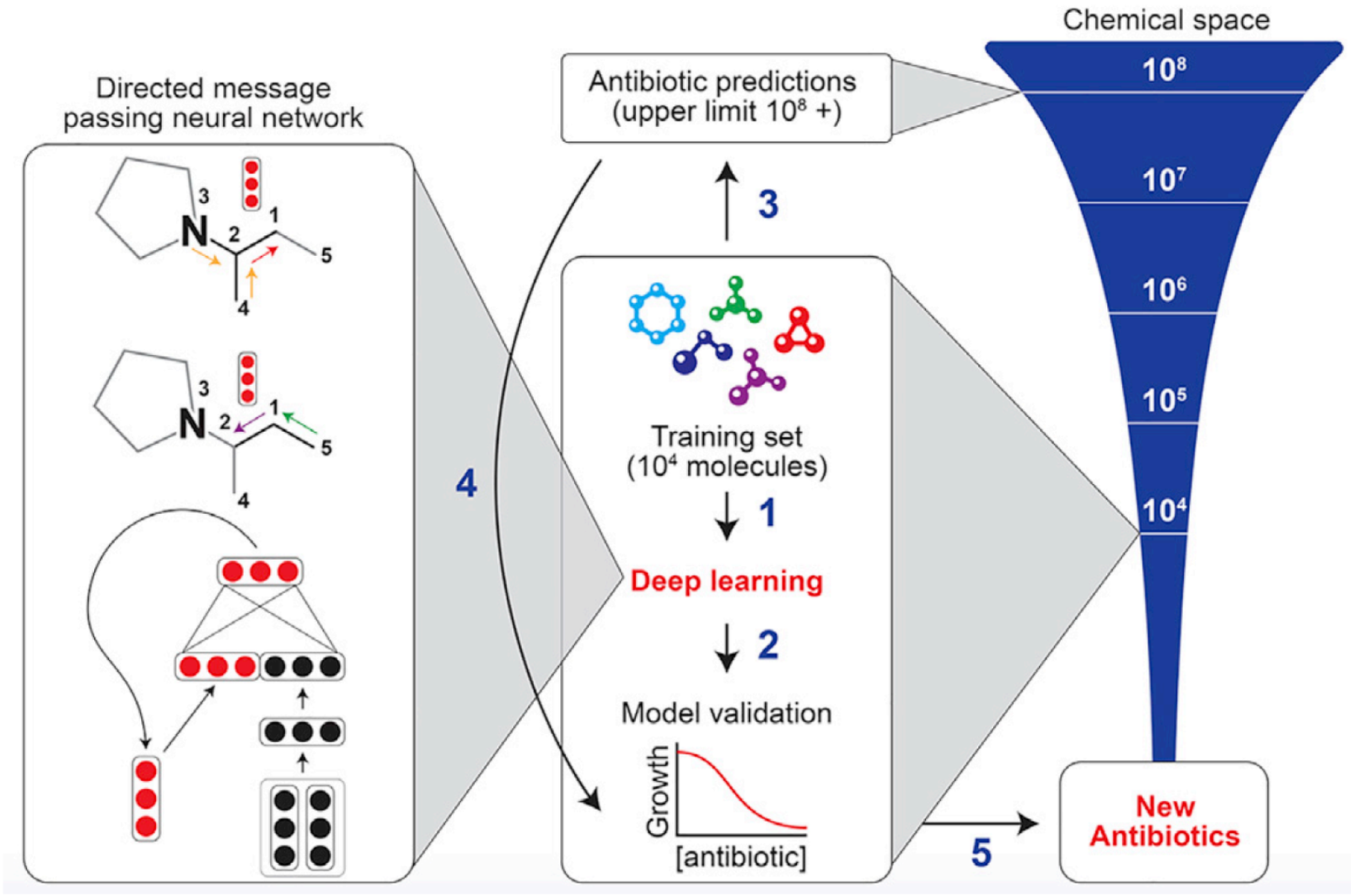



(a) Representational Collapse



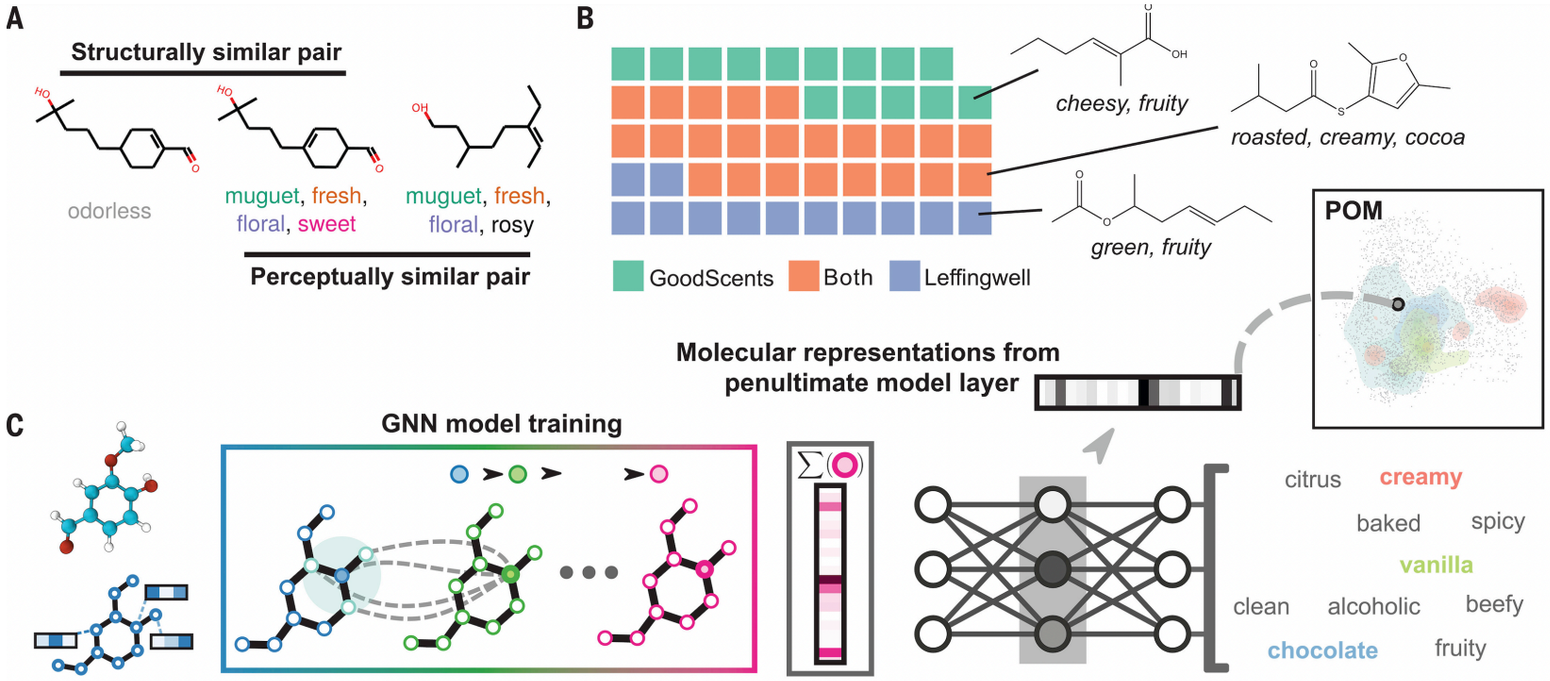

(b) Over-squashing

# Application I: Drug discovery



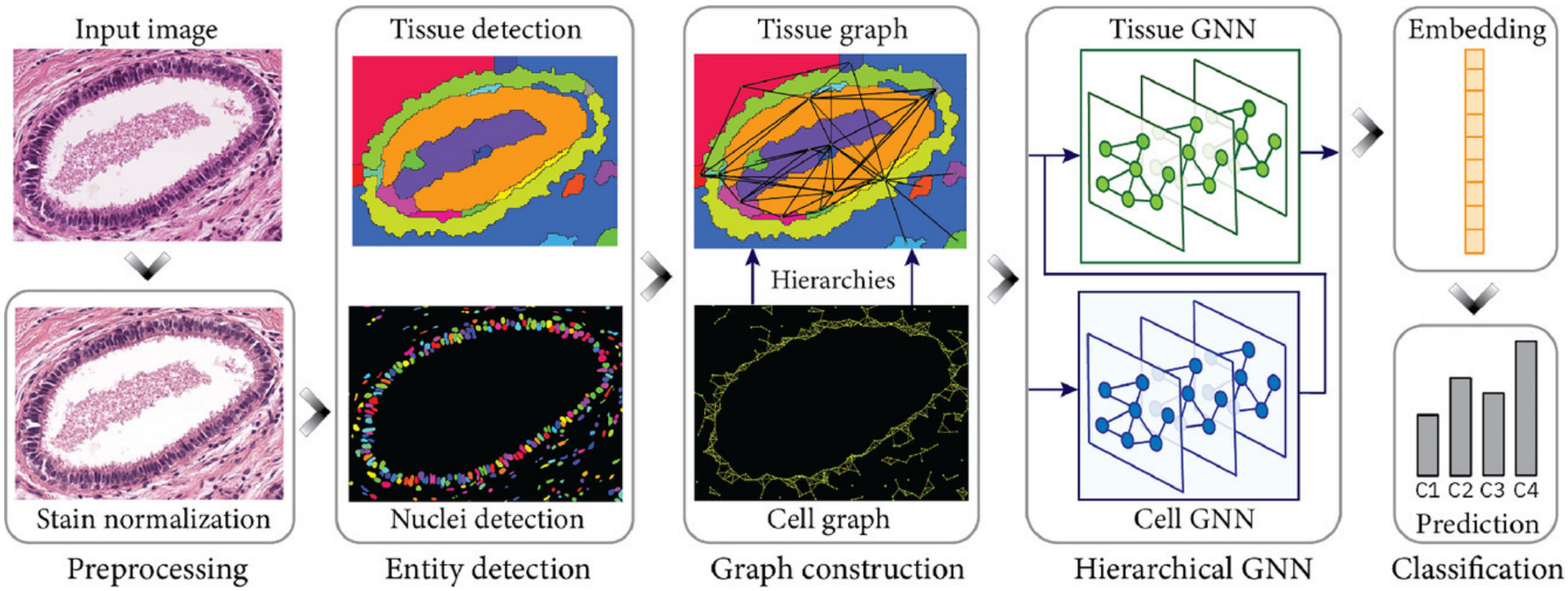

Stokes et al., "A deep learning approach to antibiotic discovery," Cell, 2020.

# Application II: Odour perception




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

# Application III: Medical imaging

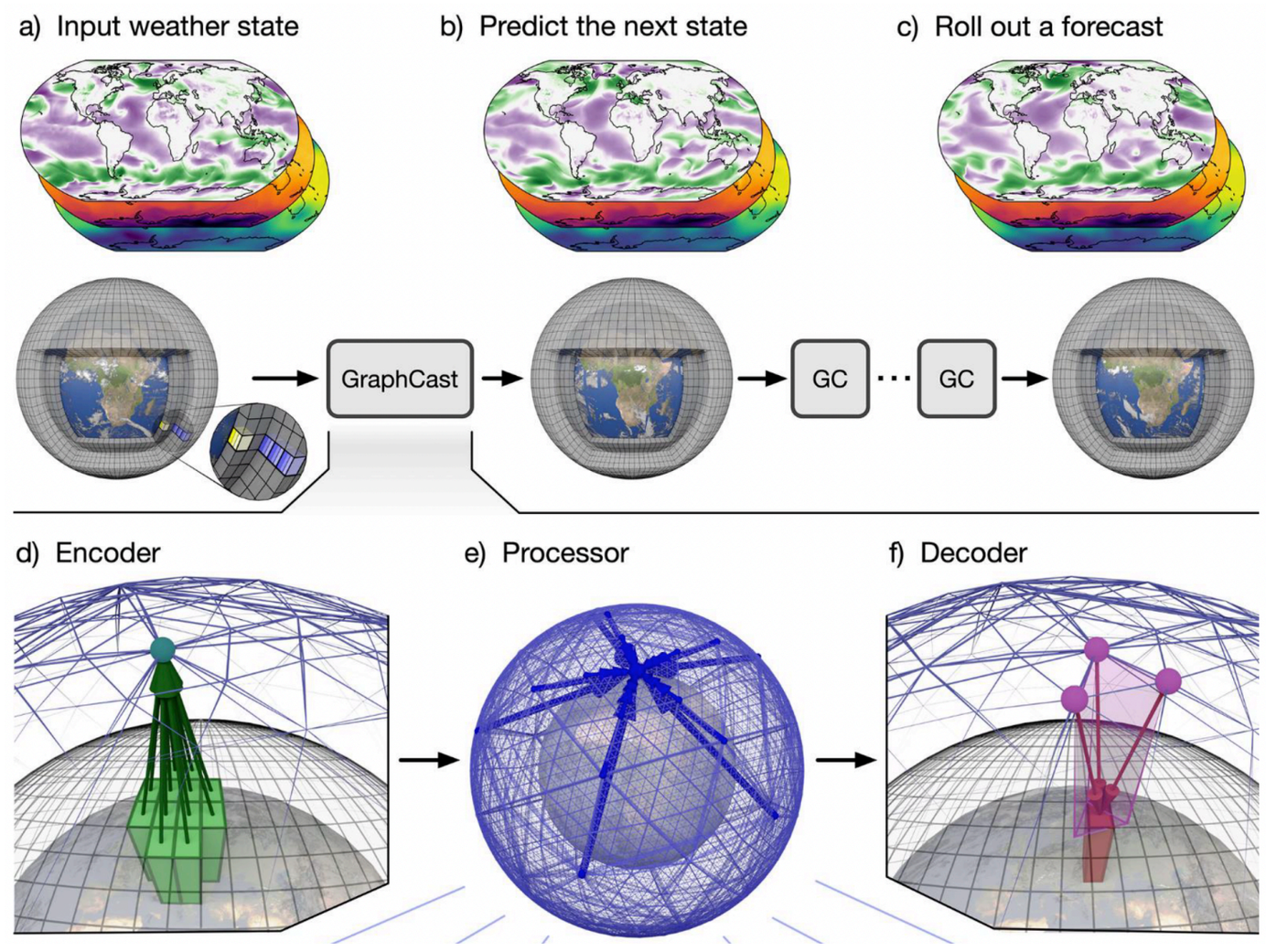



Pati et al., "Hierarchical graph representations in digital pathology," Medical Image Analysis, 2022.

# Application IV: Traffic prediction

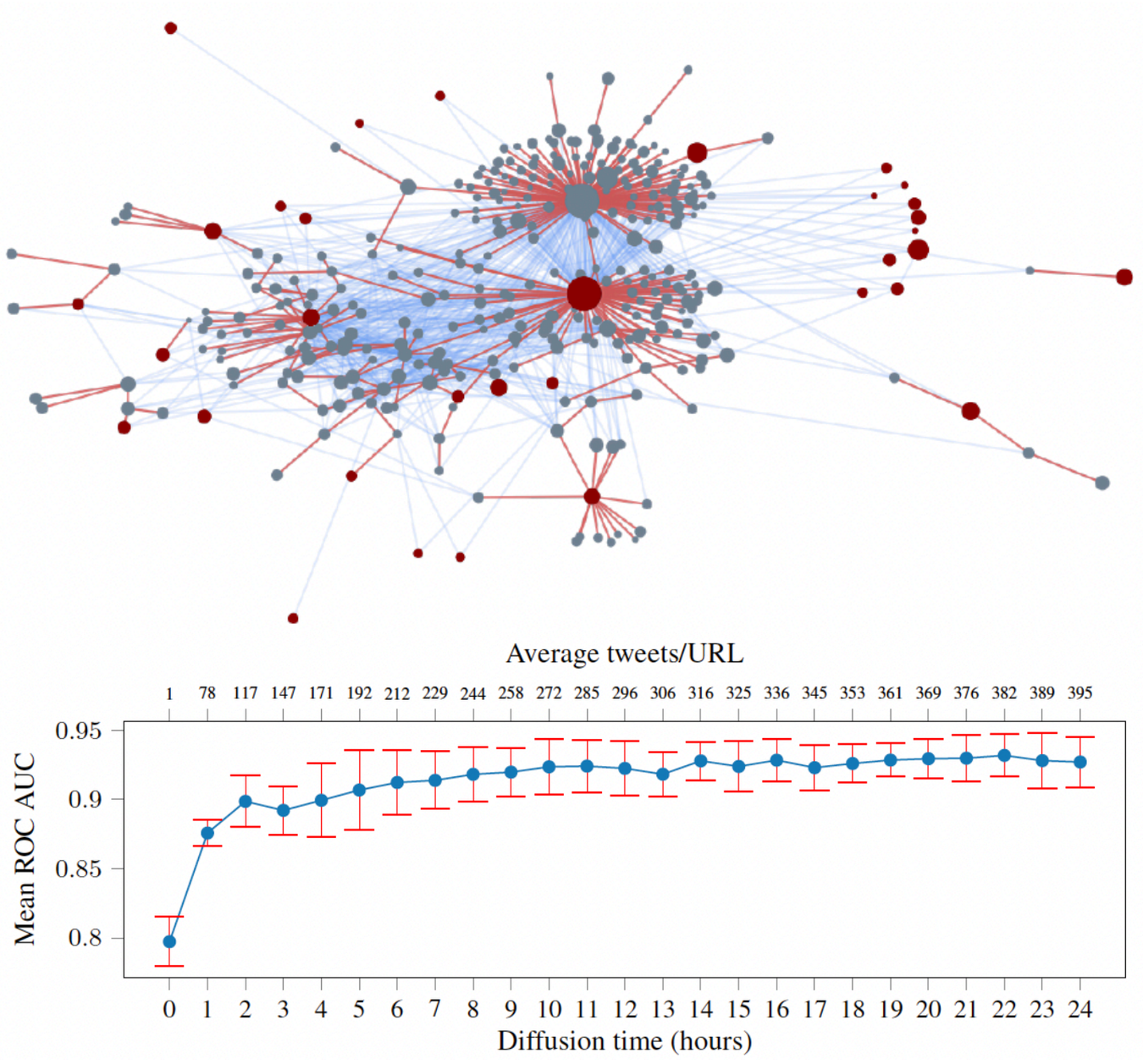
# Application IV: Traffic prediction

# Application V: Weather forecasting



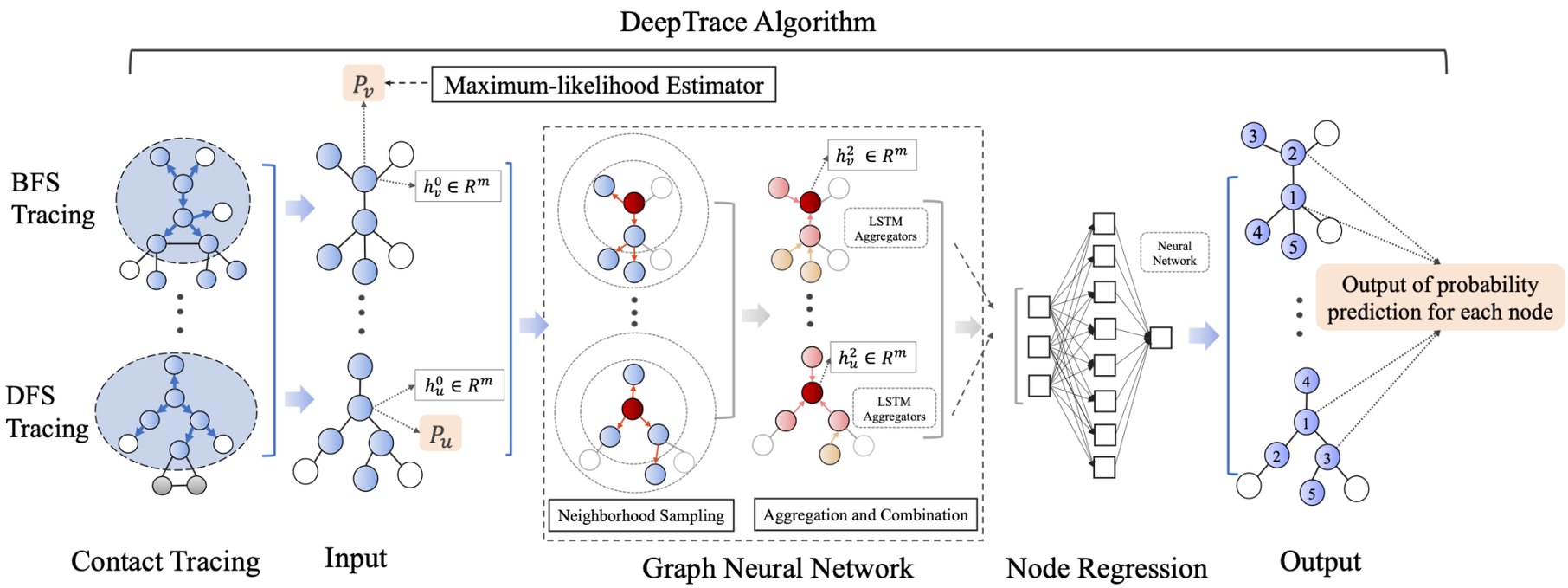

Lam et al., "Learning skillful medium-range global weather forecasting," Science, 2023.

# Application VI: Fake news detection



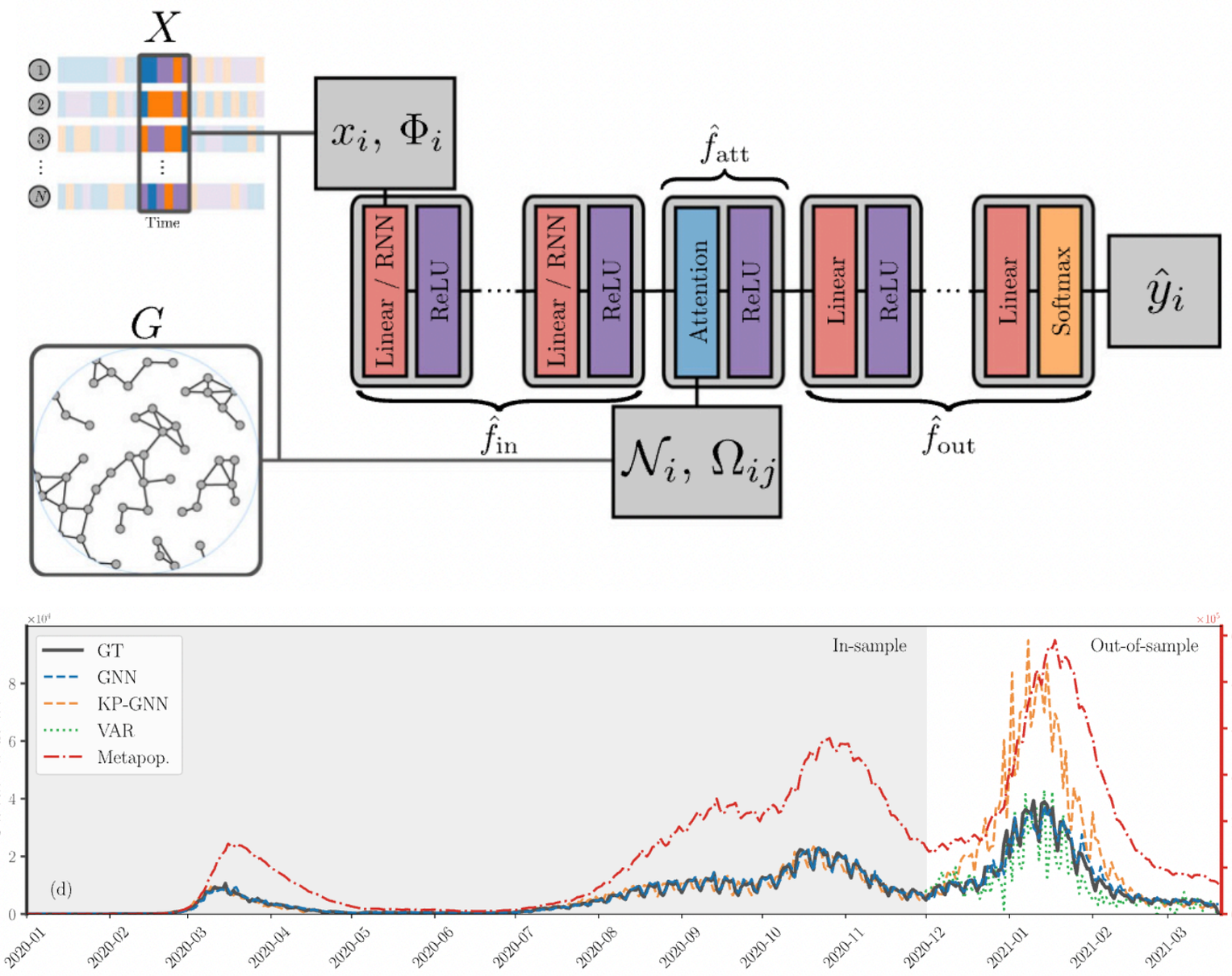

Monti et al., "Fake news detection on social media using geometric deep learning," ICLR Workshop, 2019.

# Application VII: Contact tracing



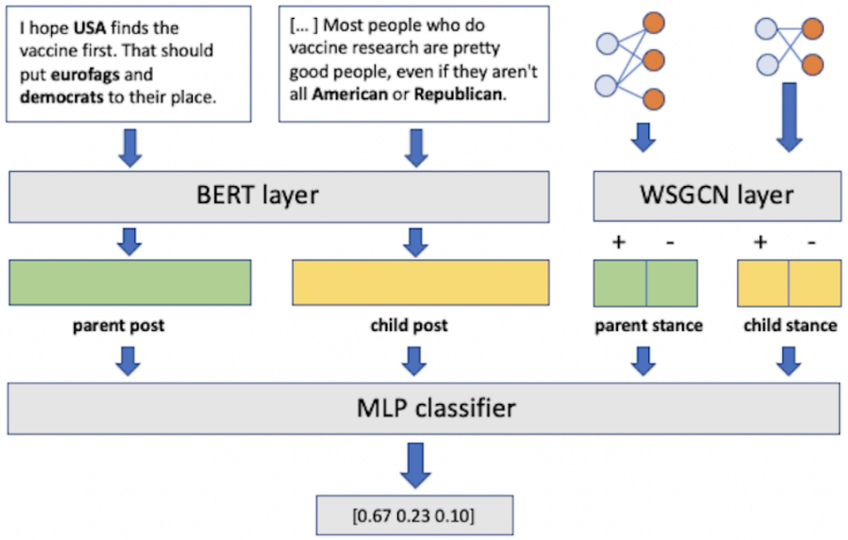

Tan et al., "DeepTrace: Learning to optimize contact tracing in epidemic networks with graph neural networks," IEEE TSIPN, 2025.

# Application VIII: Contagion dynamics

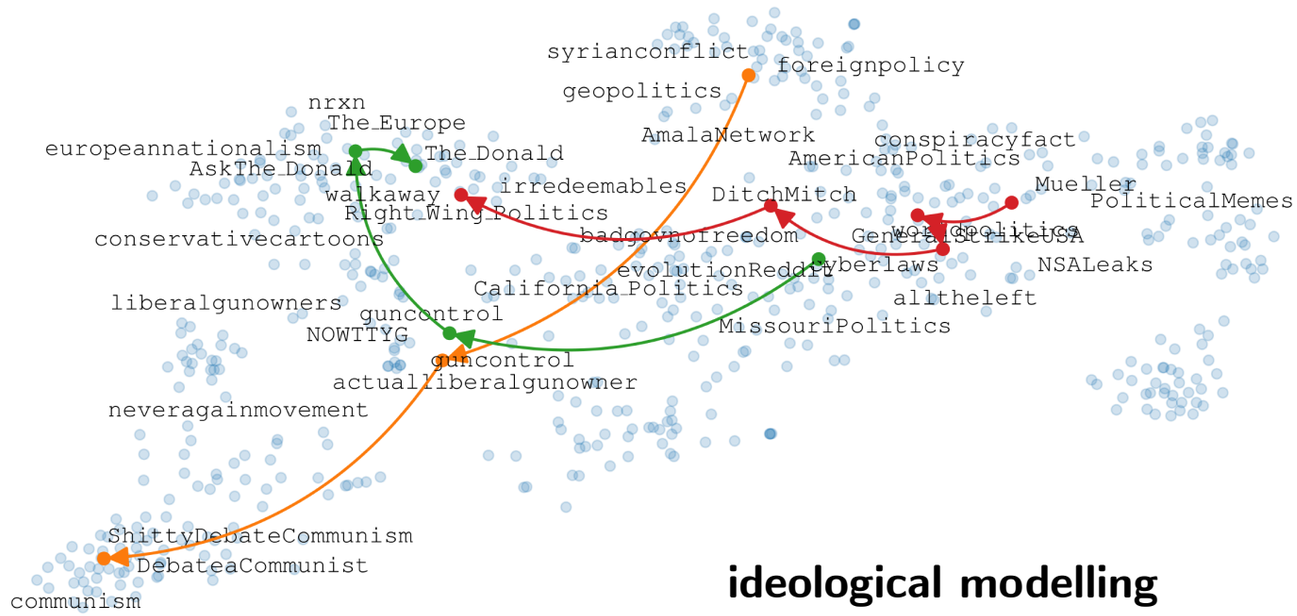



Murphy et al., "Deep learning of contagion dynamics on complex networks," Nature Communications, 2021.

# Application IX: Language modelling



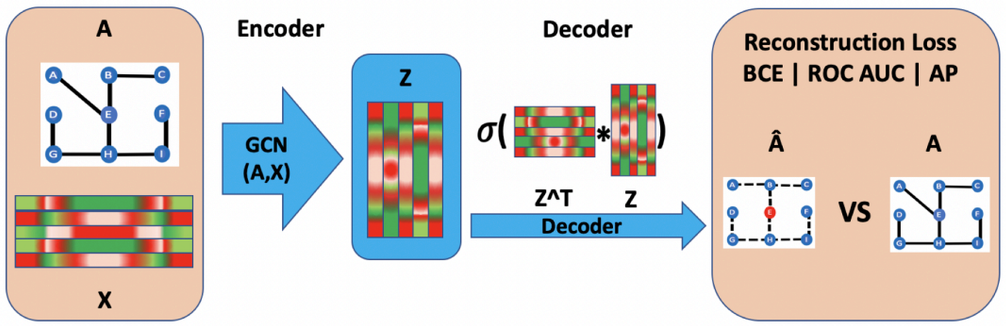

**disagreement prediction**



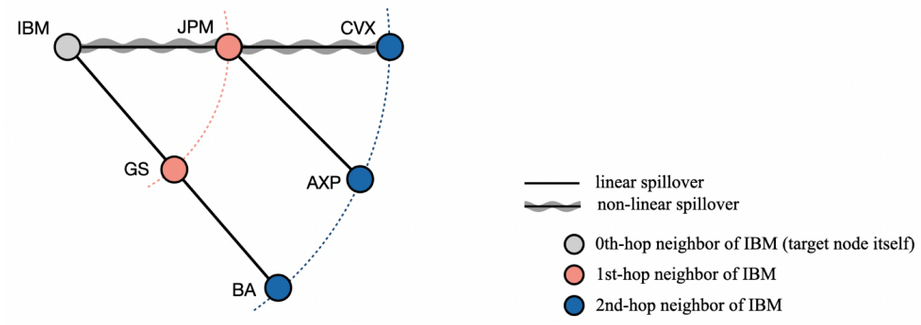

**ideological modelling**

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 X: Stock market analysis




**market instability**



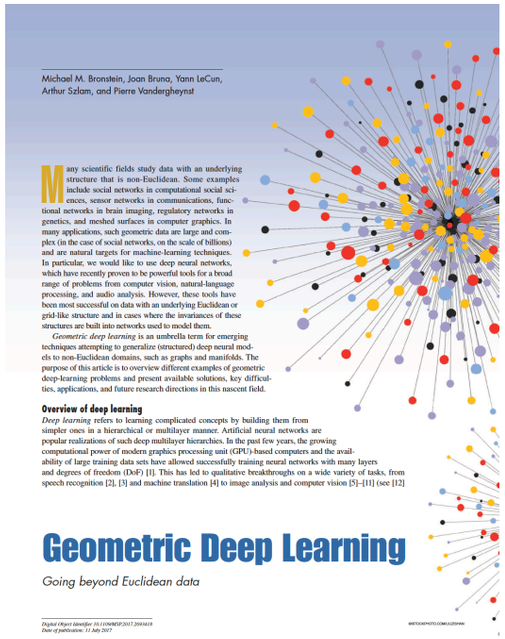

**volatility forecasting**


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.

# Deep learning on graphs - Summary

- Fast-growing field that extends data analysis to non-Euclidean domain
- Highly interdisciplinary: machine learning, signal processing, harmonic analysis, network science, differential geometry, applied statistics
- Promising directions
  - beyond convolutional models or MPNNs
  - expressive power of graph ML models
  - robustness & generalisation & scalability
  - interpretability & causal inference
  - construction & refinement of initial graphs
  - optimisation & implementation issues
  - foundation models for graph-structured data

# References



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

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

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

### Overview of deep learning

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

## Geometric Deep Learning

*Going beyond Euclidean data*

*Digital Object Identifier 10.1109/MSP.2017.2693418*
*Date of publication: 11 July 2017*



### Representation Learning on Graphs: Methods and Applications

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


**Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**





### 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 citationships, 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

Manuscript received January 2, 2019; revised August 7, 2019 and December 3, 2019; accepted March 2, 2020. Date of publication March 24, 2020; date of current version January 5, 2021. This work was supported in part by the Australian Government through the Australian Research Council (ARC) under Grant LP160100630 and Grant LP181010654 and in part by the NSF under Grant III-1526499, Grant III-1763325, Grant III-1909323, and Grant CNS-1930941. *(Corresponding author: Shirui Pan.)*

Zonghan Wu, Fengwen Chen, Guodong Long, and Chengqi Zhang are with the Centre for Artificial Intelligence, Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, NSW 2007, Australia (e-mail: zonghan.wu-3@student.uts.edu.au; fengwen.chen@student.uts.edu.au; guodong.long@uts.edu.au; chengqi.zhang@uts.edu.au).

Shirui Pan is with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia (e-mail: shirui.pan@monash.edu).

Philip S. Yu is with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053 USA (e-mail: psyu@cs.uic.edu).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.2978386





### Machine Learning on Graphs: A Model and Comprehensive Taxonomy

**Ines Chami*** CHAMI@STANFORD.EDU
*Stanford University*
*Stanford, CA, 94305, USA*

**Sami Abu-El-Haija** SAMI@HAIJA.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** CHRISMRE@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.