

# Graph Representation Learning: Foundations, Methods, Applications and Systems

Part 1: Wei Jin, Yao Ma, Yiqi Wang, Xiaorui Liu, Jiliang Tang

Part 2: Yukuo Cen, Jie Tang

Part 3: Jiezhong Qiu, Jie Tang

Part 4: Chuan Shi, Yanfang Ye, Jiawei Zhang, Philip S. Yu

Tutorial website: <http://kdd2021graph.github.io/>

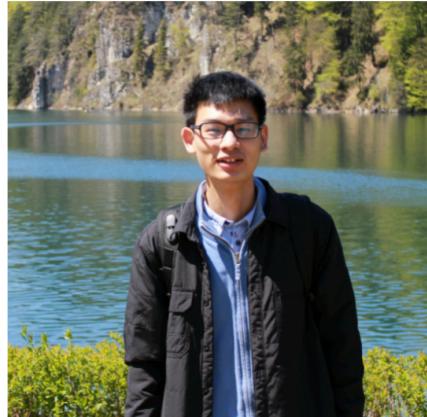
# Tutorial Outline and Tutors (1)

## ➤ Part 1

- Graph theory and Graph Fourier Analysis
- Foundations of Graph Neural Networks



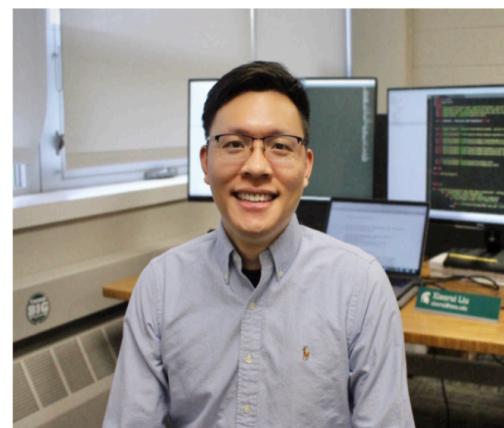
**Wei Jin**  
Michigan State University, USA



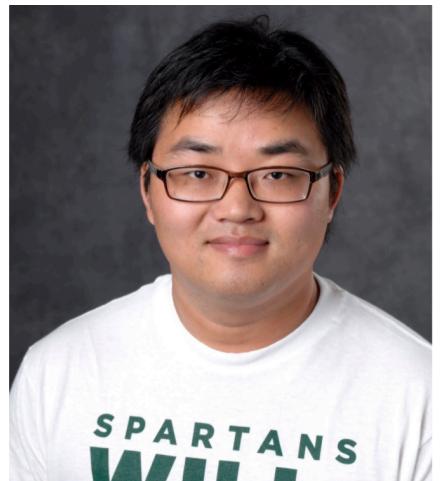
**Yao Ma**  
New Jersey Institute of Technology, USA



**Yiqi Wang**  
Michigan State University, USA



**Xiaorui Liu**  
Michigan State University, USA



**Jiliang Tang**  
Michigan State University, USA

# Tutorial Outline and Tutors (2)

- Part 2
  - Scalable Graph Neural Networks
  - CogDL Toolkit for Graph Neural Networks
- Part 3
  - Network embedding theories and systems



**Yukuo Cen**  
Tsinghua University, China



**Jiezhong Qiu**  
Tsinghua University, China



**Jie Tang**  
Tsinghua University, China

# Tutorial Outline and Tutors (3)

- Part 4
  - Heterogeneous Graph Neural Networks



**Chuan Shi**

Beijing University of Posts and  
Telecommunications, China



**Yanfang Ye**

Case Western Reserve University, USA



**Jiawei Zhang**

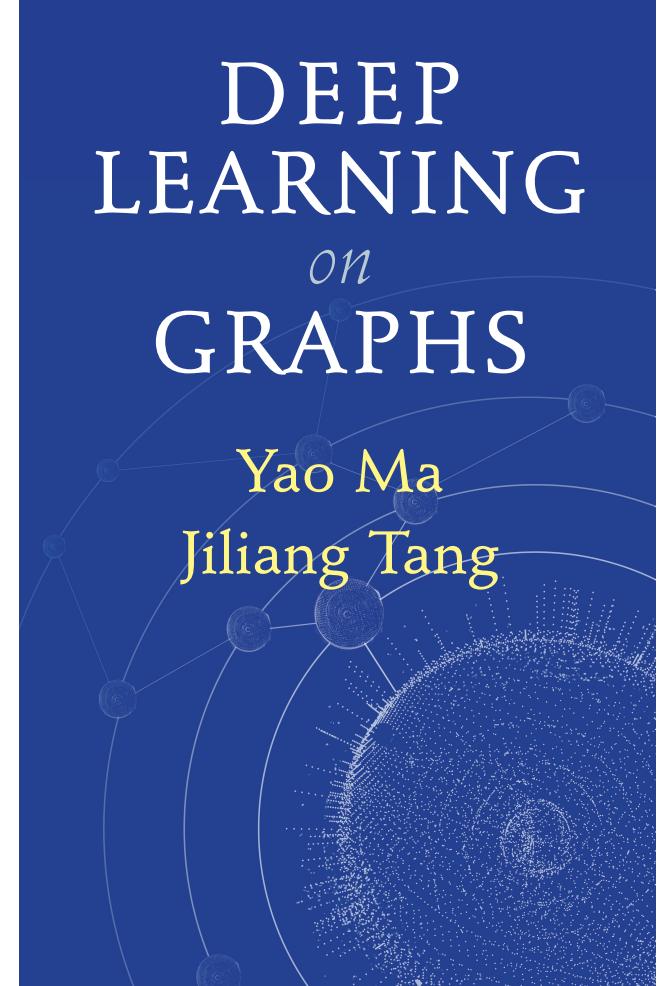
Florida State University, USA



**Philip S. Yu**

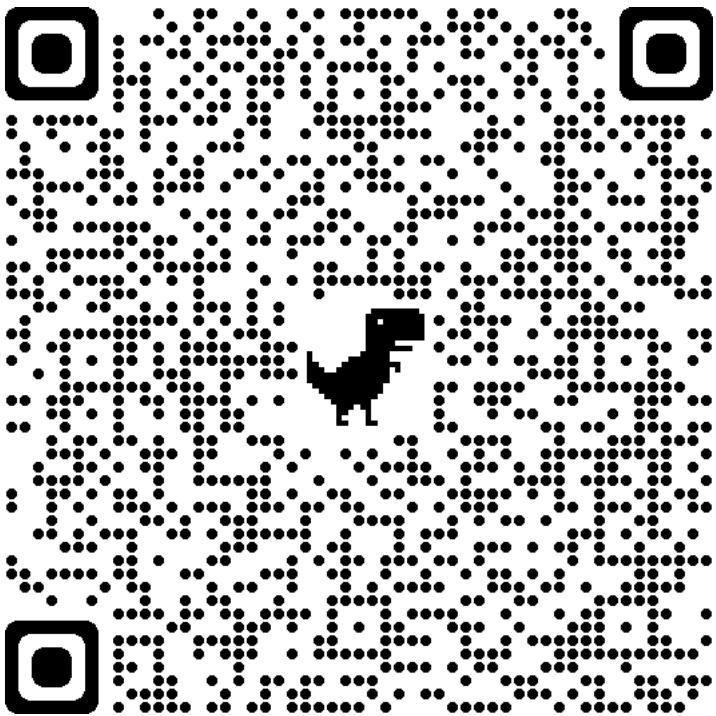
University of Illinois at Chicago, USA

# Book: Deep Learning on Graphs



[https://cse.msu.edu/~mayao4/dlg\\_book/](https://cse.msu.edu/~mayao4/dlg_book/)

# Chinese Version





- Graph theory and Graph Fourier Analysis
- Foundations of Graph Neural Networks

Yao Ma<sup>1</sup>, Xiaorui Liu<sup>2</sup>, Yiqi Wang<sup>2</sup> and Wei Jin<sup>2</sup>

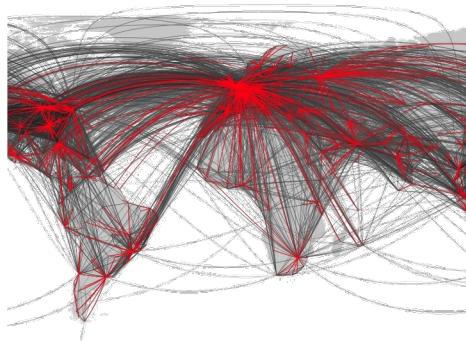
<sup>1</sup>New Jersey Institute of Technology

<sup>2</sup>Michigan State University

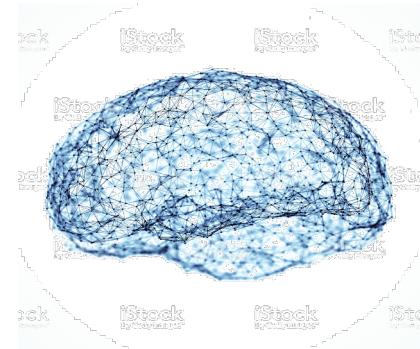
# Data as Graphs - Explicit



Social Graphs



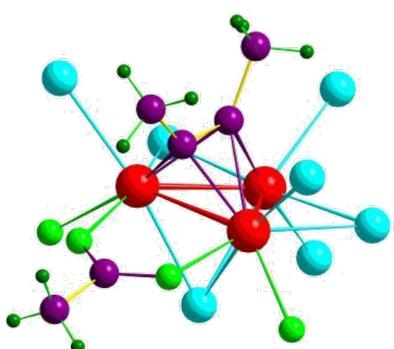
Transportation Graphs



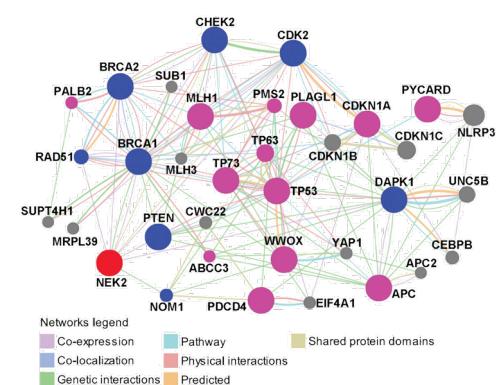
Brain Graphs



Web Graphs

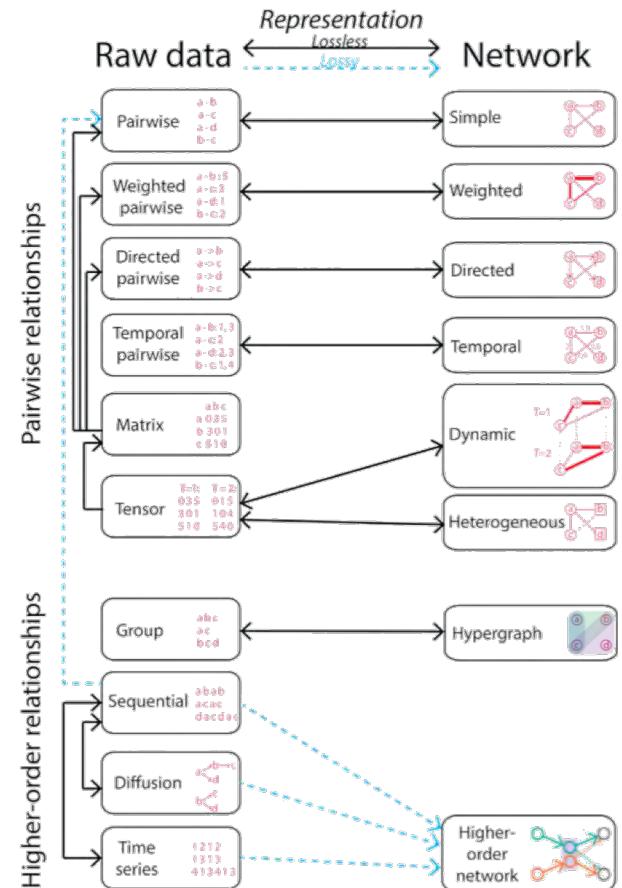
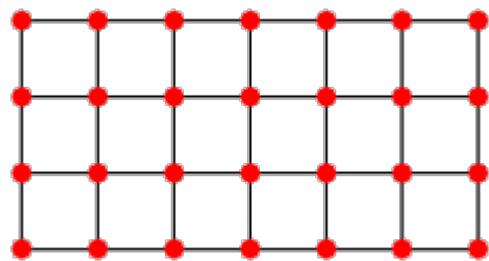


Molecular Graphs



Gene Graphs

# Data as Graphs - Implicit



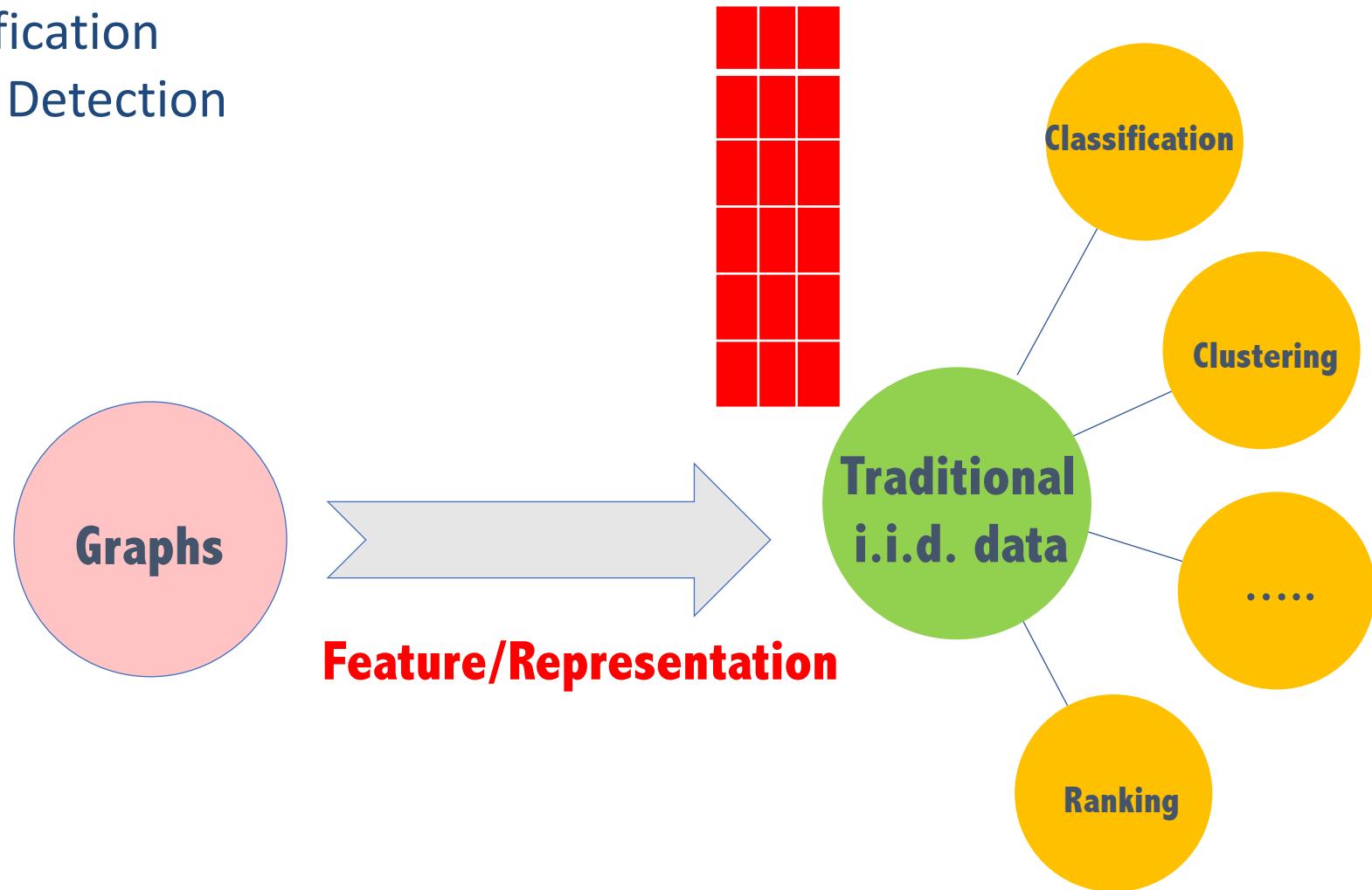
Jian Xu. Representing Big Data as Networks. PhD Dissertation,  
University of Notre Dame

# ML on Graphs

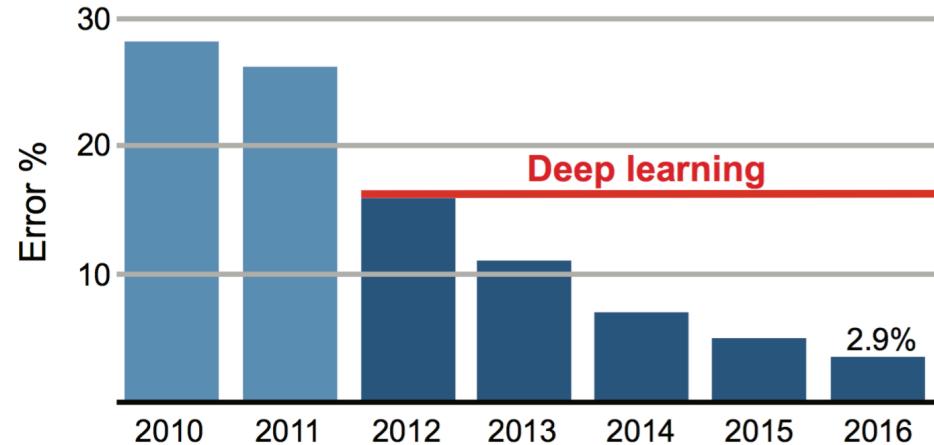
Numerous real-world problems can be summarized as a set of tasks on graphs

- Link prediction
- Node Classification
- Community Detection
- Ranking ....

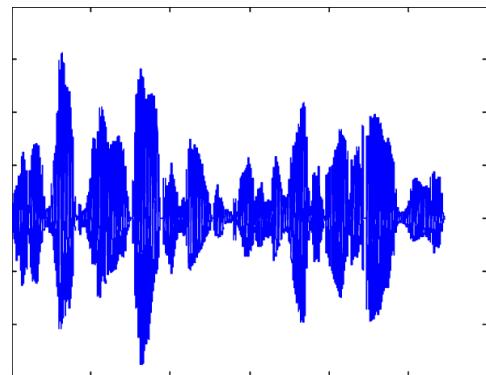
ML solutions



# The Power of Deep Learning



IMAGENET

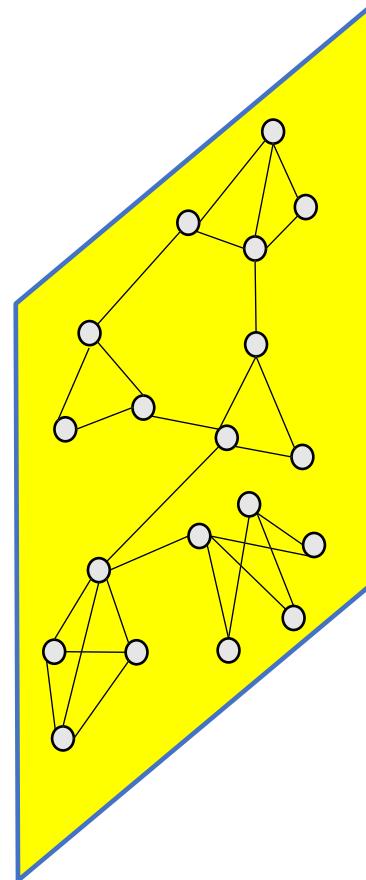
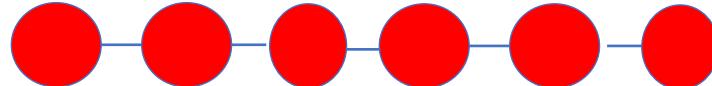
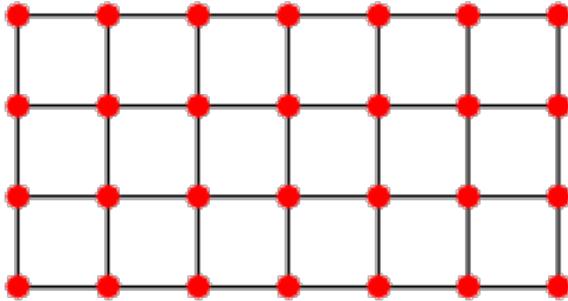


Acoustic model	Recog \ WER	RT03S FSH	Hub5 SWB
Traditional features	1-pass -adapt	27.4	23.6
Deep Learning	1-pass -adapt	18.5	16.1

# Deep Learning Meets Graphs: Challenges

Traditional DL is designed for simple grids or sequences

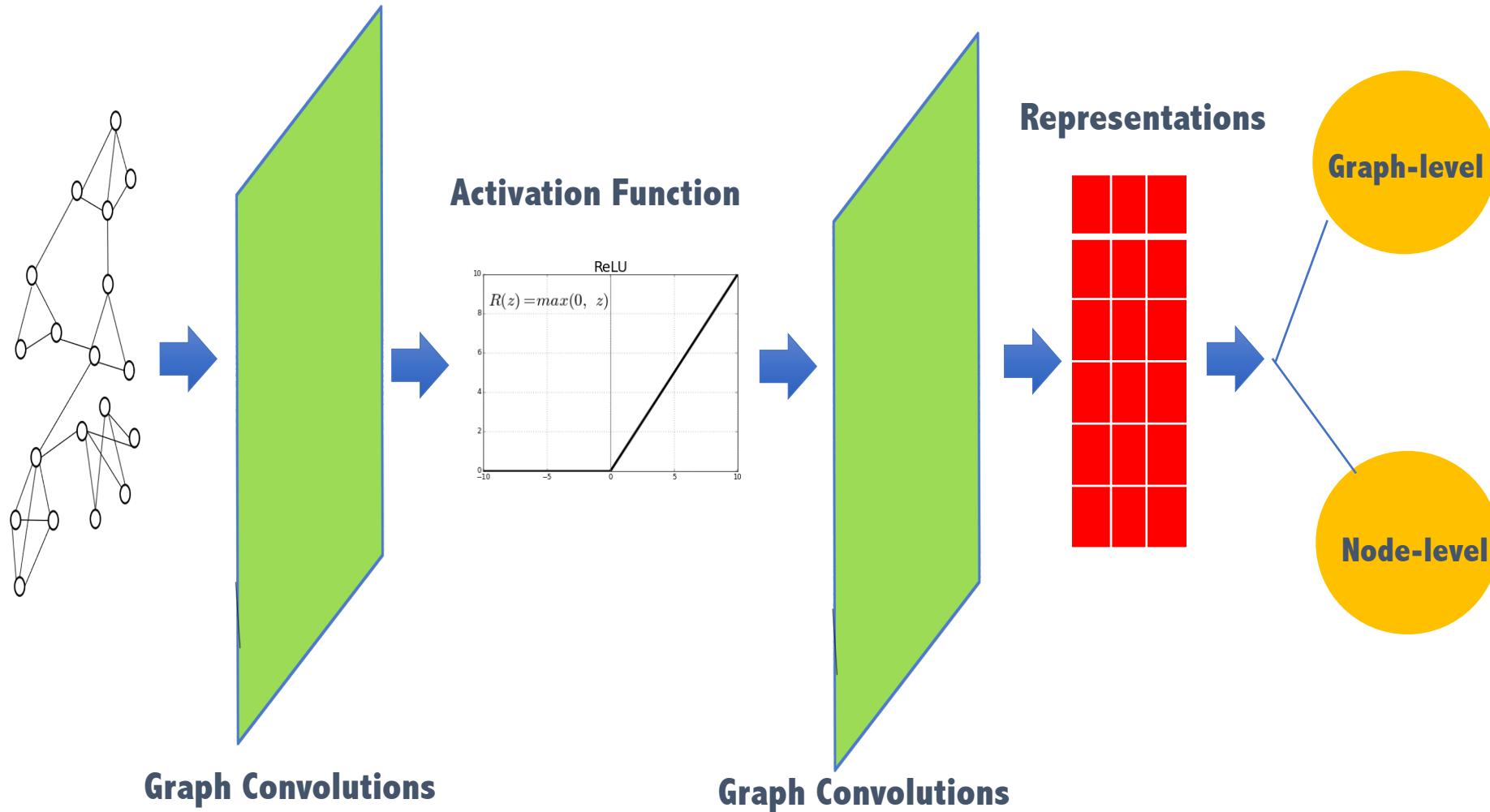
- CNNs for fixed-size images/grids
- RNNs for text/sequences



But nodes on graphs have different connections

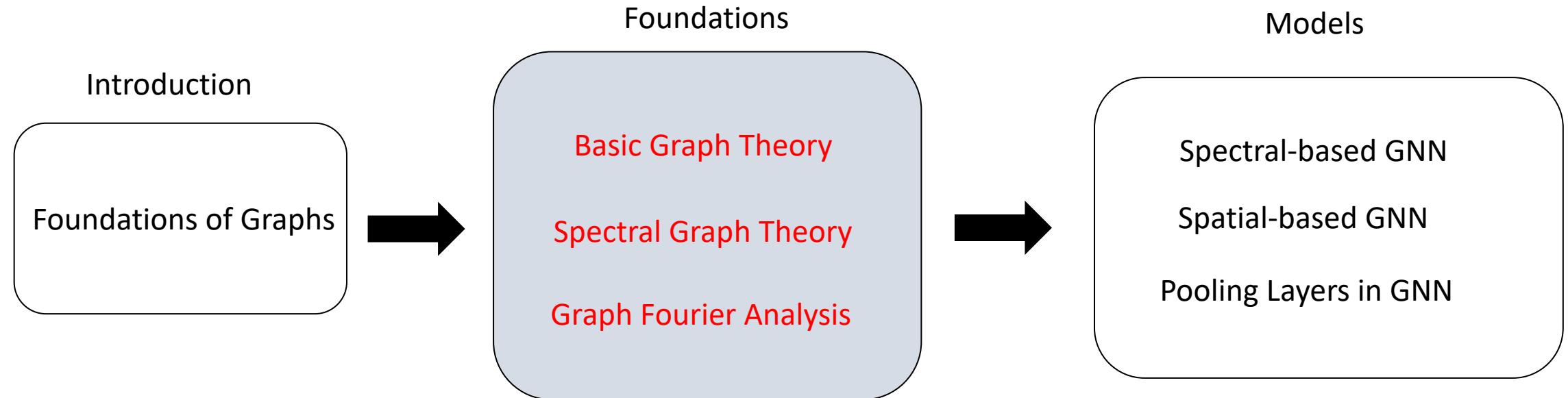
- Arbitrary neighbor size
- Complex topological structure
- No fixed node ordering

# Graph Neural Networks



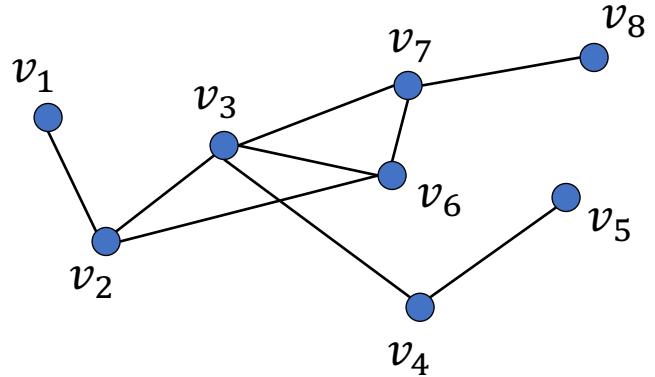


# Overview





# Graphs and Graph Signals



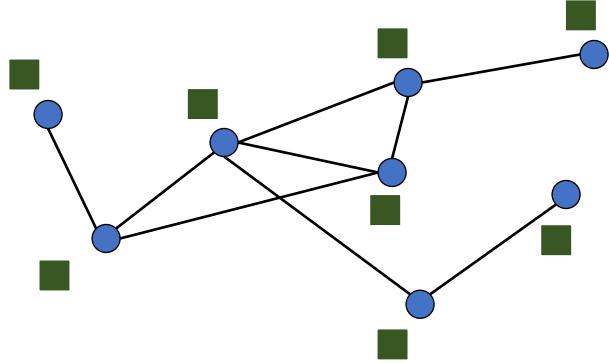
$$\mathcal{V} = \{v_1, \dots, v_N\}$$

$$\mathcal{E} = \{e_1, \dots, e_M\}$$

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



# Graphs and Graph Signals



$$\mathcal{V} = \{v_1, \dots, v_N\}$$

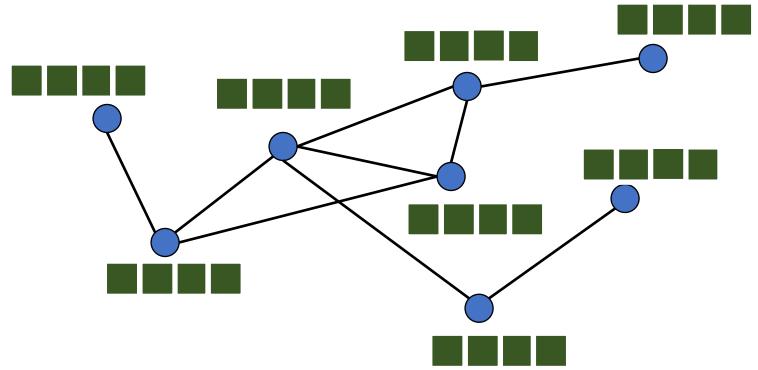
$$\mathcal{E} = \{e_1, \dots, e_M\}$$

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

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

$$\mathcal{V} \rightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$

# Graphs and Graph Signals



$$\mathcal{V} = \{v_1, \dots, v_N\}$$

$$\mathcal{E} = \{e_1, \dots, e_M\}$$

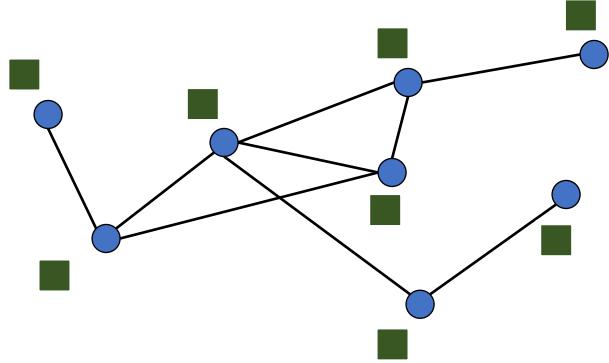
$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

Graph Signal:  $f : \mathcal{V} \rightarrow \mathbb{R}^{N \times d}$

$$\mathcal{V} \rightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$



# Graphs and Graph Signals



$$\mathcal{V} = \{v_1, \dots, v_N\}$$

$$\mathcal{E} = \{e_1, \dots, e_M\}$$

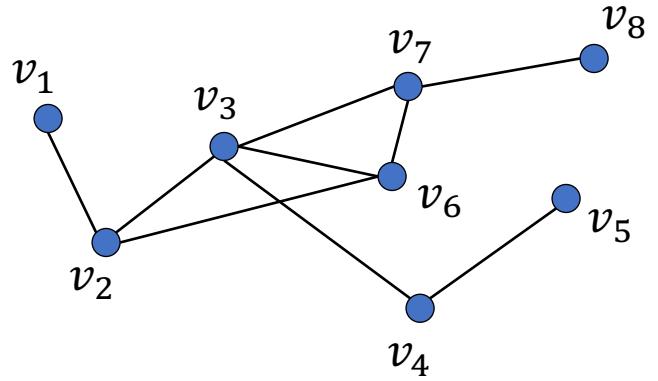
$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

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

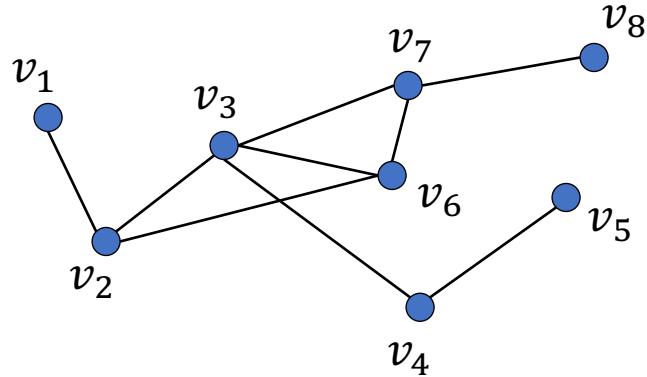
$$\mathcal{V} \rightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$



# Matrix Representations of Graphs



# Matrix Representations of Graphs



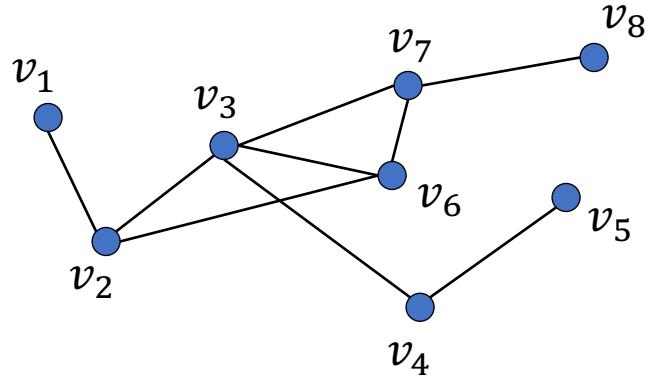
Adjacency Matrix:  $A[i, j] = 1$  if  $v_i$  is adjacent to  $v_j$   
 $A[i, j] = 0$ , otherwise

Adjacency Matrix

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

$A$

# Matrix Representations of Graphs



Adjacency Matrix:  $A[i, j] = 1$  if  $v_i$  is adjacent to  $v_j$   
 $A[i, j] = 0$ , otherwise

Degree Matrix:  $\mathbf{D} = \text{diag}(\text{degree}(v_1), \dots, \text{degree}(v_N))$

Degree Matrix

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

**D**

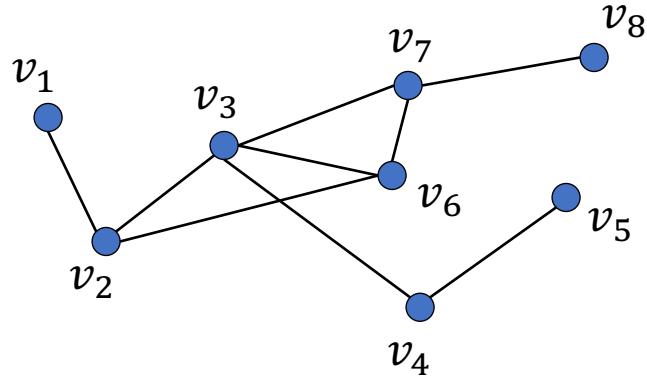
Adjacency Matrix

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

**A**



# Matrix Representations of Graphs



Adjacency Matrix:  $A[i, j] = 1$  if  $v_i$  is adjacent to  $v_j$   
 $A[i, j] = 0$ , otherwise

Degree Matrix:  $\mathbf{D} = \text{diag}(\text{degree}(v_1), \dots, \text{degree}(v_N))$

$$\begin{array}{c}
 \text{Degree Matrix} \\
 \left( \begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right) - \left( \begin{array}{cccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \right) = \left( \begin{array}{cccccccc} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{array} \right)
 \end{array}$$

**D**                    **A**                    **L**



# Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$



# Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$



# Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$

Laplacian quadratic form:

$$\mathbf{f}^T \mathbf{L} \mathbf{f} = \frac{1}{2} \sum_{i,j=1}^N \mathbf{A}[i,j] (\mathbf{f}(i) - \mathbf{f}(j))^2$$



# Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$

Laplacian quadratic form:

$$\mathbf{f}^T \mathbf{L} \mathbf{f} = \frac{1}{2} \sum_{i,j=1}^N \mathbf{A}[i,j] (\mathbf{f}(i) - \mathbf{f}(j))^2$$

---

“Smoothness” or “Frequency” of the signal  $f$



# Laplacian Matrix as an Operator

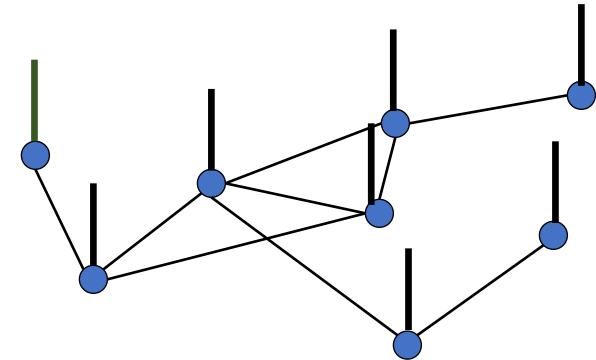
Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$

Laplacian quadratic form:

$$\mathbf{f}^T \mathbf{L} \mathbf{f} = \frac{1}{2} \sum_{i,j=1}^N \mathbf{A}[i,j] (\mathbf{f}(i) - \mathbf{f}(j))^2$$



Low frequency graph signal

---

“Smoothness” or “Frequency” of the signal  $f$



# Laplacian Matrix as an Operator

Laplacian matrix is a difference operator:

$$\mathbf{h} = \mathbf{L}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f}$$

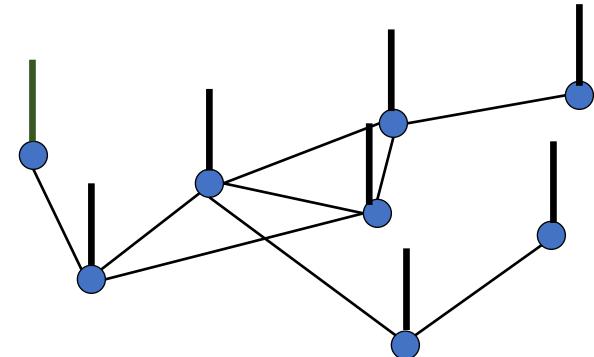
$$\mathbf{h}(i) = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}(i) - \mathbf{f}(j))$$

Laplacian quadratic form:

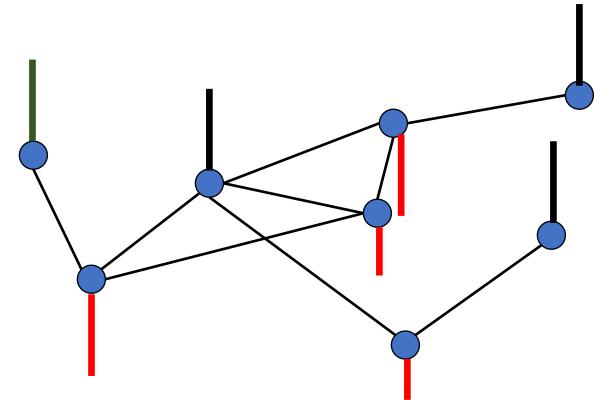
$$\mathbf{f}^T \mathbf{L} \mathbf{f} = \frac{1}{2} \sum_{i,j=1}^N \mathbf{A}[i,j] (\mathbf{f}(i) - \mathbf{f}(j))^2$$

---

“Smoothness” or “Frequency” of the signal  $f$



Low frequency graph signal



High frequency graph signal



# Eigen-decomposition of Laplacian Matrix

Laplacian matrix has a complete set of orthonormal eigenvectors:

$$\mathbf{L} = \begin{bmatrix} | & & | & & | \\ \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \\ | & & | & & | \end{bmatrix} \begin{bmatrix} \lambda_0 & & 0 & & \\ & \ddots & & & \\ 0 & & \lambda_{N-1} & & \end{bmatrix} \begin{bmatrix} | & \mathbf{u}_0 & | & | \\ \vdots & & & \\ | & \mathbf{u}_{N-1} & | & | \end{bmatrix}$$



# Eigen-decomposition of Laplacian Matrix

Laplacian matrix has a complete set of orthonormal eigenvectors:

$$\mathbf{L} = \begin{bmatrix} | & & | & & | \\ \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \\ | & & | & & | \end{bmatrix} \begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix} \begin{bmatrix} | & \mathbf{u}_0 & | & | \\ \vdots & & & \\ | & \mathbf{u}_{N-1} & | & | \end{bmatrix}$$

$\mathbf{U}$                                    $\Lambda$                                    $\mathbf{U}^T$



# Eigen-decomposition of Laplacian Matrix

Laplacian matrix has a complete set of orthonormal eigenvectors:

$$\mathbf{L} = \begin{bmatrix} | & & | & & | \\ \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \\ | & & | & & | \end{bmatrix} \begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix} \begin{bmatrix} | & \mathbf{u}_0 & | & | \\ \vdots & & & \\ \mathbf{u}_{N-1} & & | & | \end{bmatrix}$$

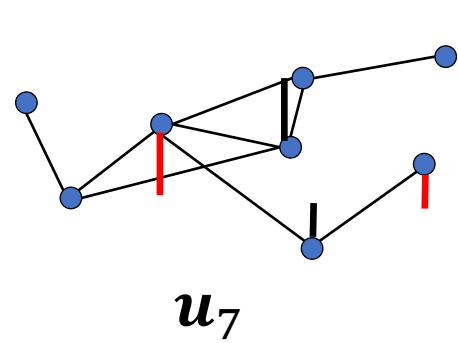
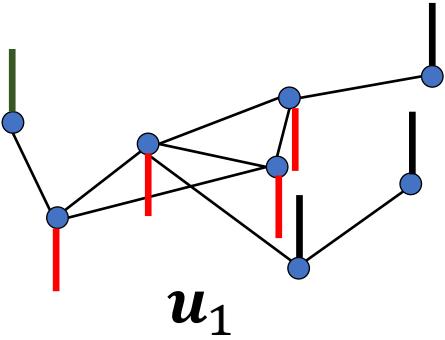
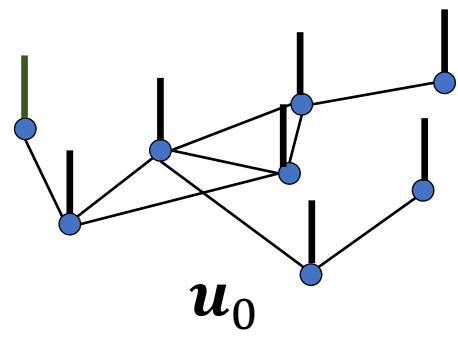
$\mathbf{U}$                                      $\Lambda$                                      $\mathbf{U}^T$

Eigenvalues are sorted non-decreasingly:

$$0 = \lambda_0 < \lambda_1 \leq \cdots \leq \lambda_{N-1}$$



# Eigenvectors as Graph Signals



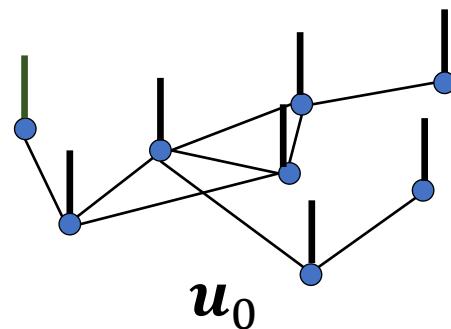


# Eigenvectors as Graph Signals

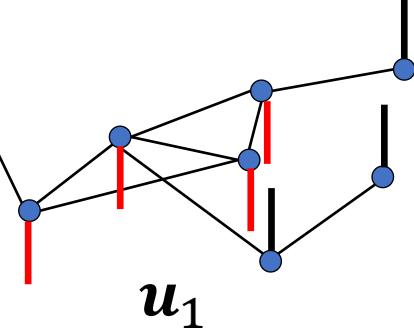
The frequency of an eigenvector of Laplacian matrix is its corresponding eigenvalue:

$$\mathbf{u}_i^T \mathbf{L} \mathbf{u}_i = \mathbf{u}_i^T \lambda_i \mathbf{u}_i = \lambda_i$$

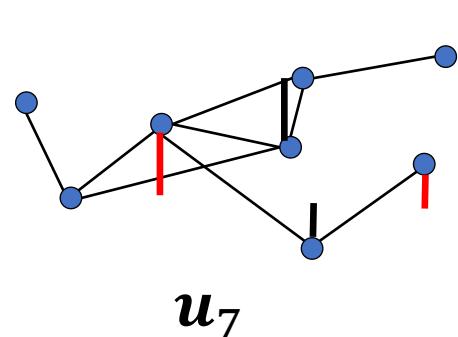
Frequency of the signal  $\mathbf{u}_i$



$$\mathbf{u}_0^T \mathbf{L} \mathbf{u}_0 = \lambda_0 = 0$$



$$\mathbf{u}_1^T \mathbf{L} \mathbf{u}_1 = \lambda_1$$



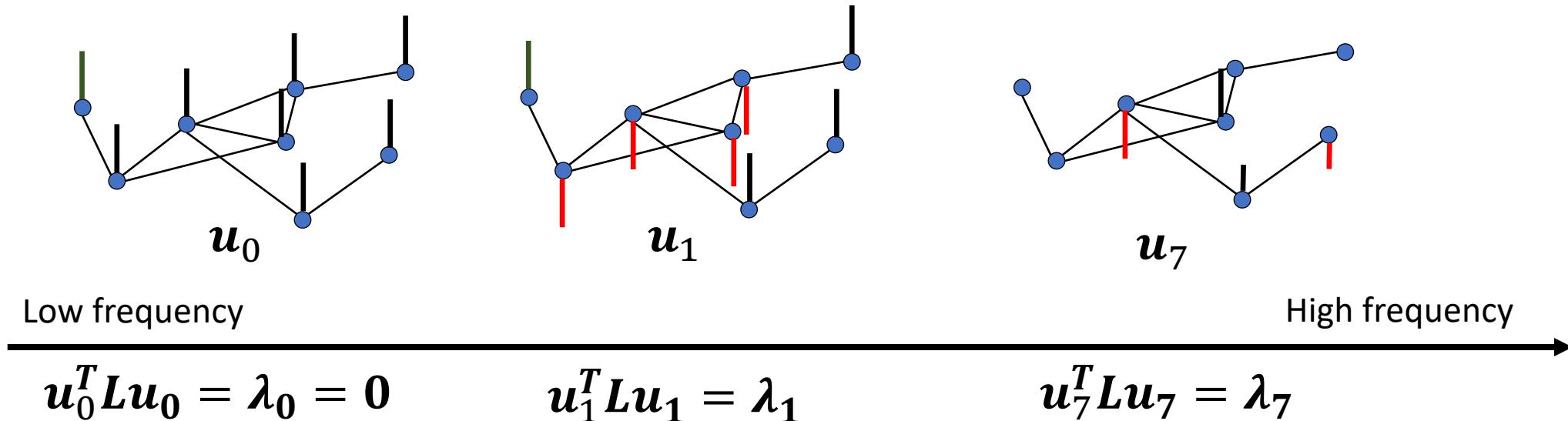
$$\mathbf{u}_7^T \mathbf{L} \mathbf{u}_7 = \lambda_7$$

# Eigenvectors as Graph Signals

The frequency of an eigenvector of Laplacian matrix is its corresponding eigenvalue:

$$\mathbf{u}_i^T \mathbf{L} \mathbf{u}_i = \mathbf{u}_i^T \lambda_i \mathbf{u}_i = \lambda_i$$

Frequency of the signal  $\mathbf{u}_i$





# Graph Fourier Transform

A signal  $f$  can be written as graph Fourier series:

$$f = \sum_{i=0}^{N-1} \hat{f}_i \cdot u_i$$

$u_i$ : graph Fourier mode

$\lambda_i$ : frequency

$\hat{f}_i$ : graph Fourier coefficients



# Graph Fourier Transform

A signal  $f$  can be written as graph Fourier series:

$$f = \sum_{i=0}^{N-1} \hat{f}_i \cdot u_i$$

$u_i$ : graph Fourier mode

$\lambda_i$ : frequency

$\hat{f}_i$ : graph Fourier coefficients

# Graph Fourier Transform (GFT)

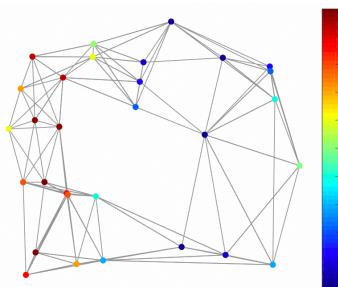
A signal  $f$  can be written as graph Fourier series:

$$f = \sum_{i=0}^{N-1} \hat{f}_i \cdot u_i$$

$u_i$ : graph Fourier mode

$\lambda_i$ : frequency

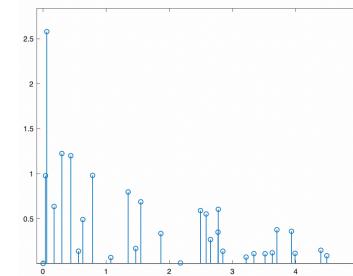
$\hat{f}_i$ : graph Fourier coefficients



Spatial domain:  $f$

$$\hat{f} = U^T f$$

Decompose signal  $f$



Spectral domain:  $\hat{f}$

# Inverse Graph Fourier Transform (IGFT)

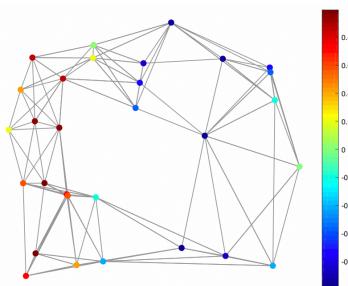
A signal  $f$  can be written as graph Fourier series:

$$f = \sum_{i=0}^{N-1} \hat{f}_i \cdot u_i$$

$u_i$ : graph Fourier mode

$\lambda_i$ : frequency

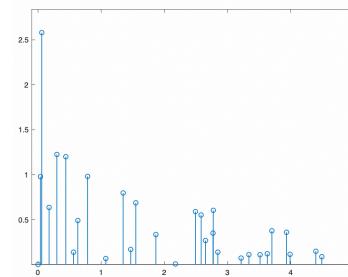
$\hat{f}_i$ : graph Fourier coefficients



Spatial domain:  $f$

$$f = U\hat{f}$$

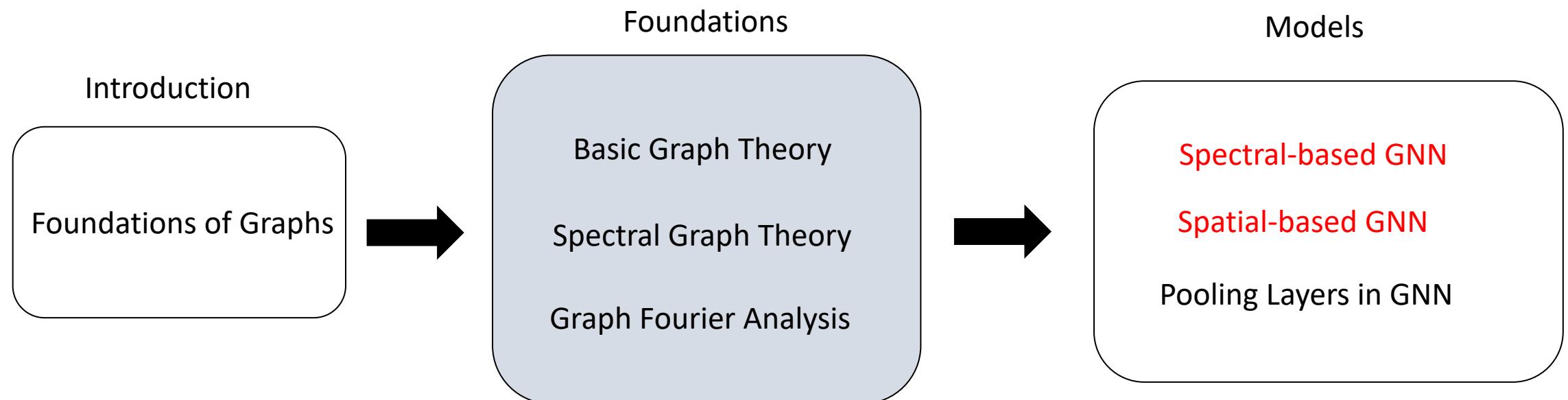
Reconstruct signal  $f$



Spectral domain:  $\hat{f}$



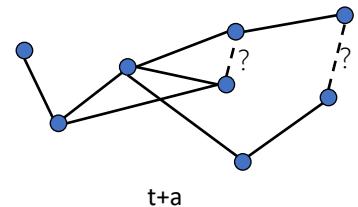
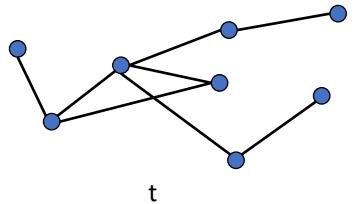
# Overview



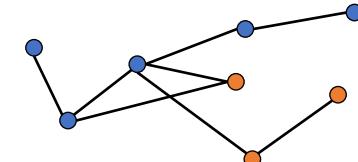
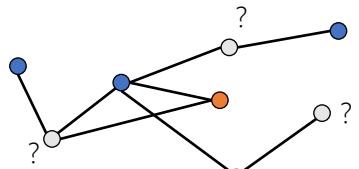
# Tasks on Graph-Structured Data

## Node-level

### Link Prediction

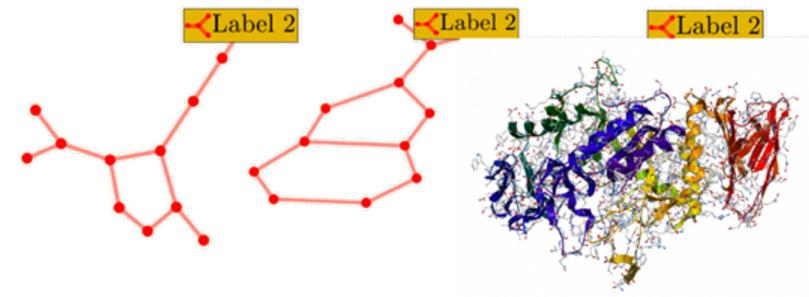
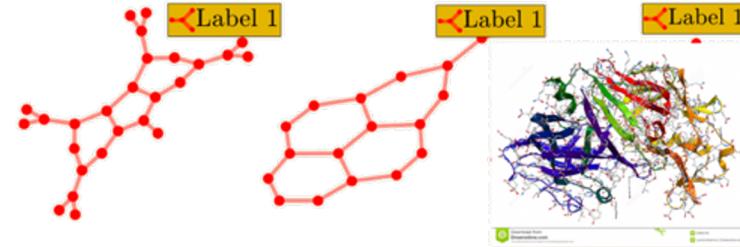


### Node Classification



## Graph-level

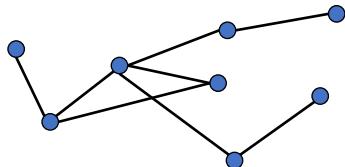
### Graph Classification





# Tasks on Graph-Structured Data

Node-level

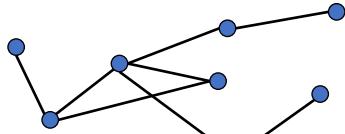


Graph-level

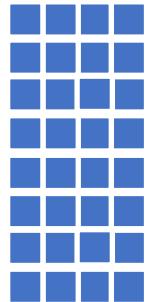


# Tasks on Graph-Structured Data

Node-level



Graph-level

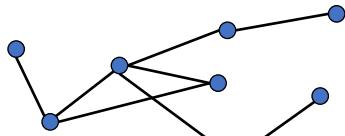


Node Representations

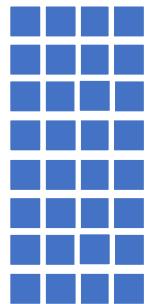


# Tasks on Graph-Structured Data

Node-level



Graph-level



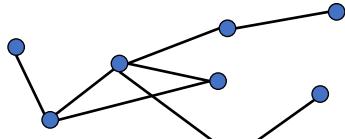
Node Representations

Graph Representation

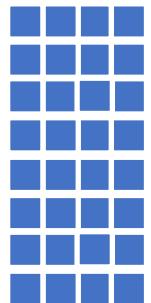


# Tasks on Graph-Structured Data

Node-level



Filtering



Node Representations

Graph-level

Pooling

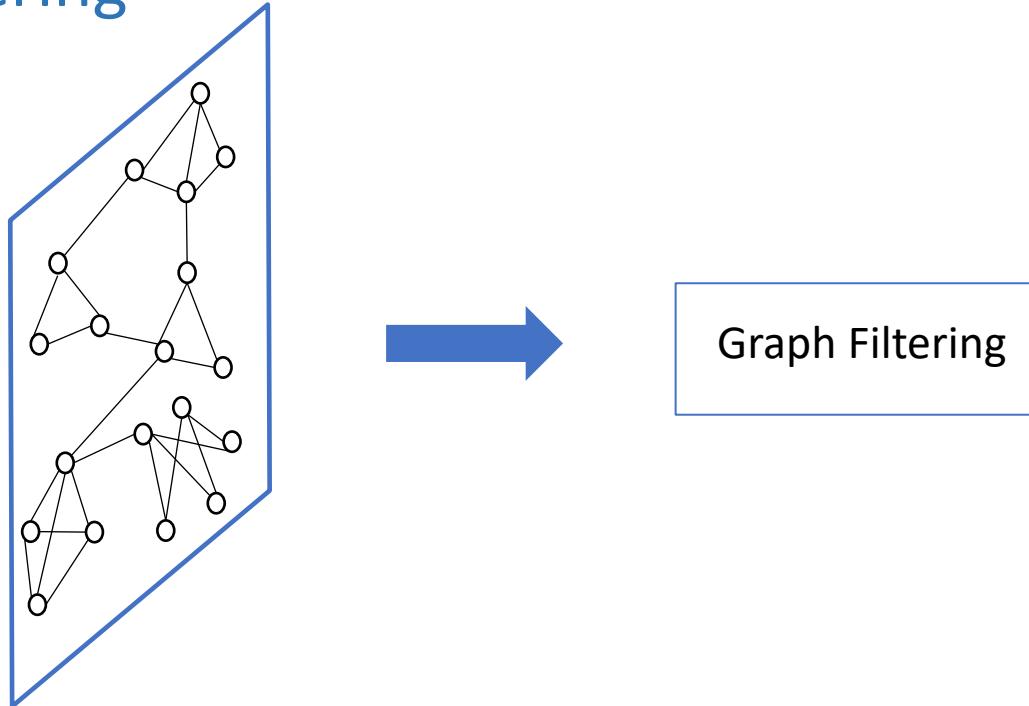


Graph Representations



# Two Main Operations in GNN

## Graph Filtering

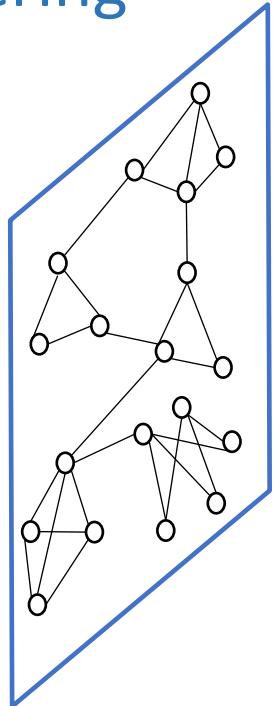


$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

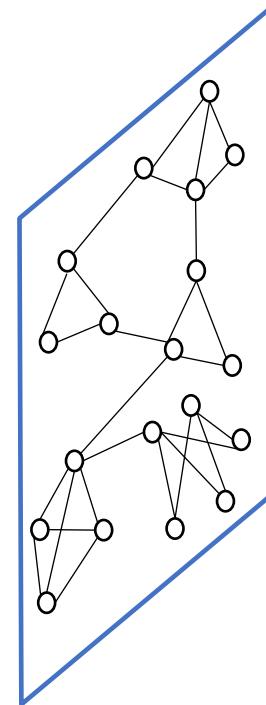


# Two Main Operations in GNN

Graph Filtering



Graph Filtering

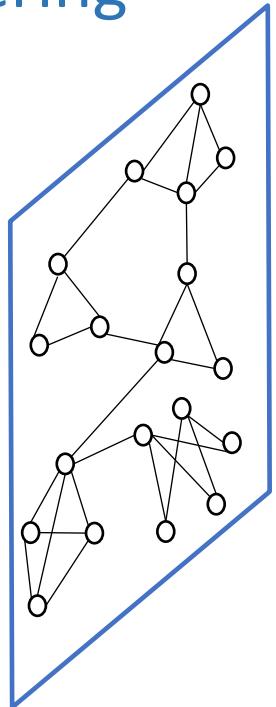


$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

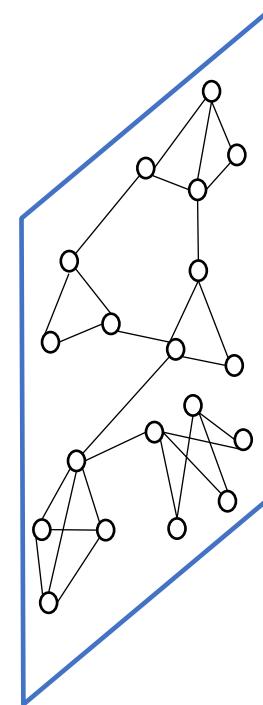
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X}_f \in \mathbb{R}^{n \times d_{new}}$$

# Two Main Operations in GNN

## Graph Filtering



Graph Filtering



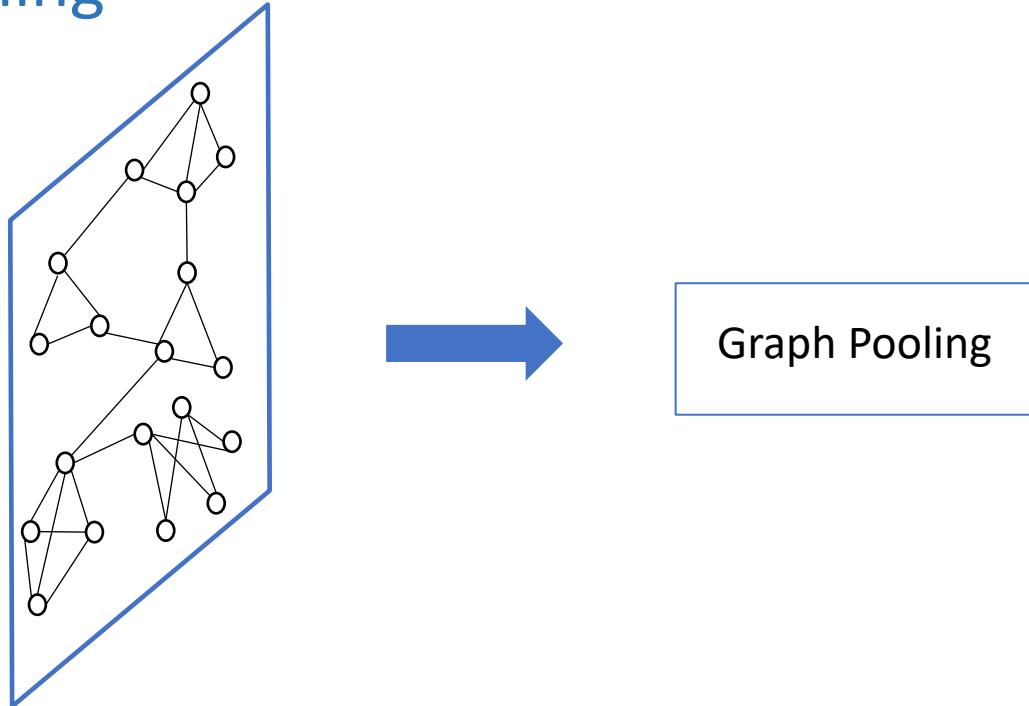
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X}_f \in \mathbb{R}^{n \times d_{new}}$$

Graph filtering refines the node features

# Two Main Operations in GNN

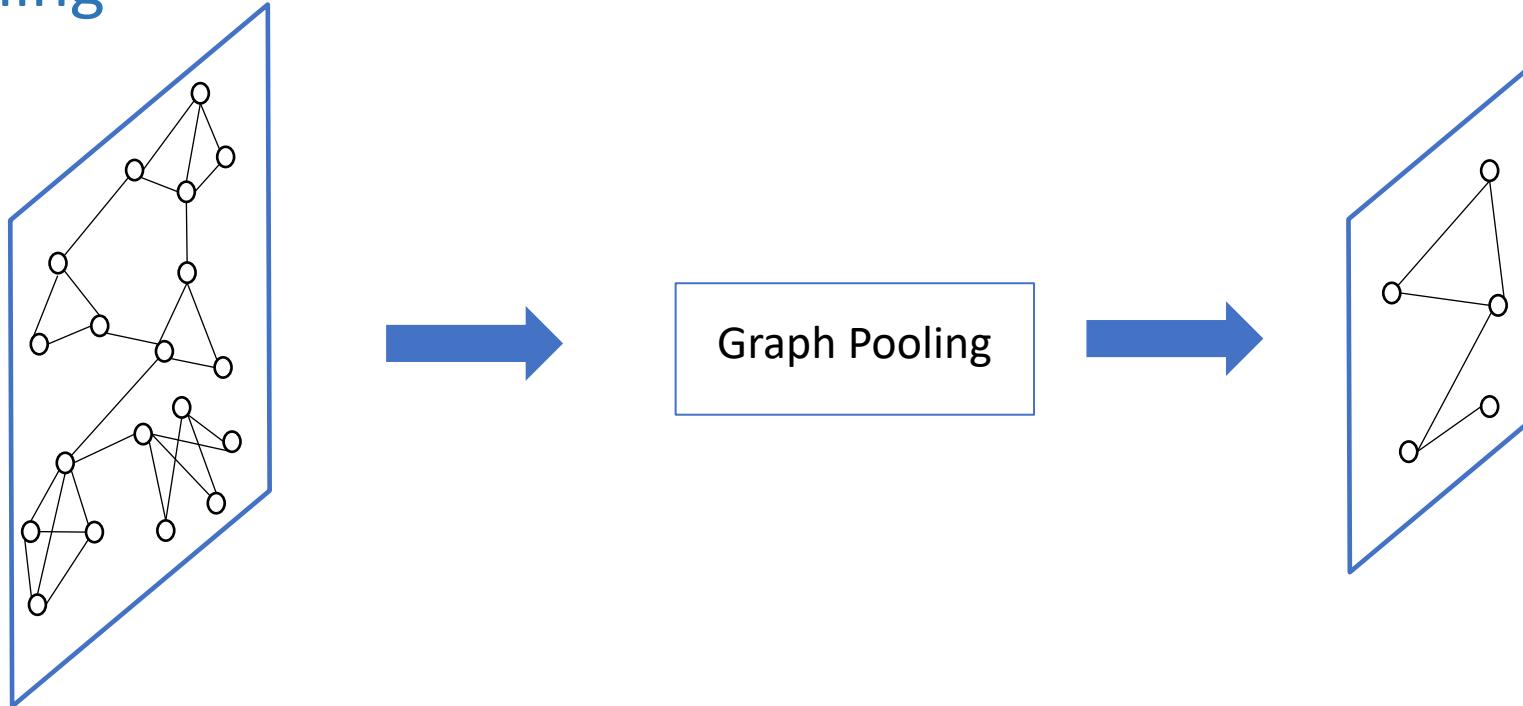
## Graph Pooling



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

# Two Main Operations in GNN

## Graph Pooling

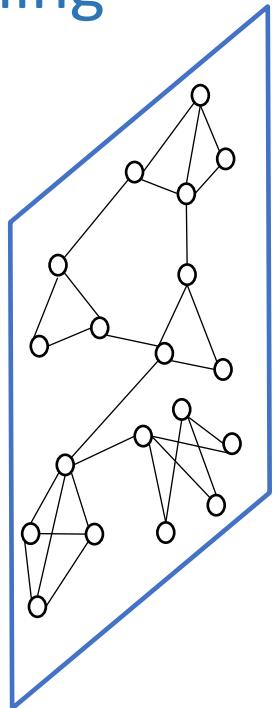


$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

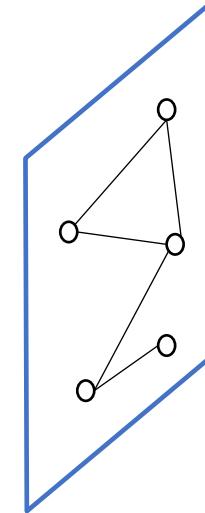
$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{X}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

# Two Main Operations in GNN

## Graph Pooling



Graph Pooling



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

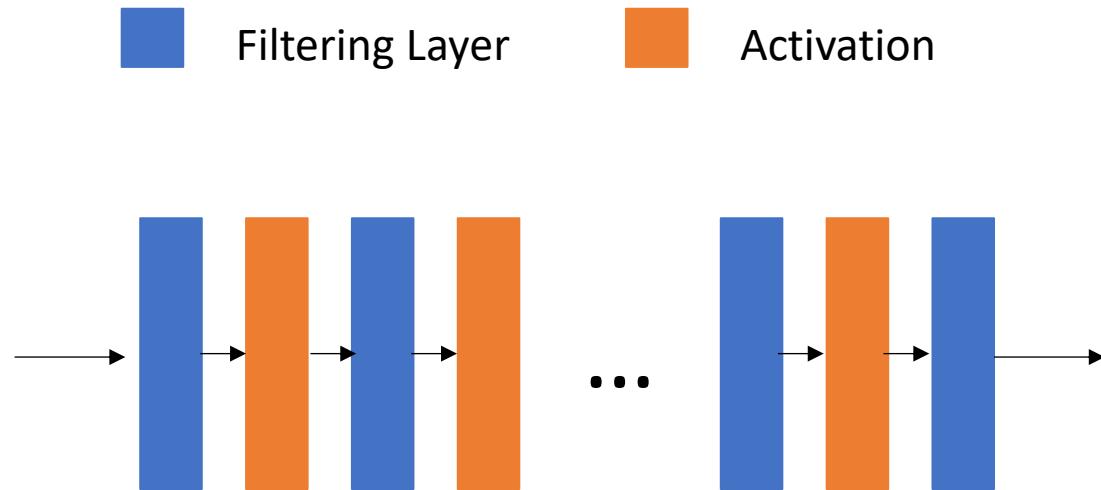
$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{X}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

Graph pooling generates a smaller graph



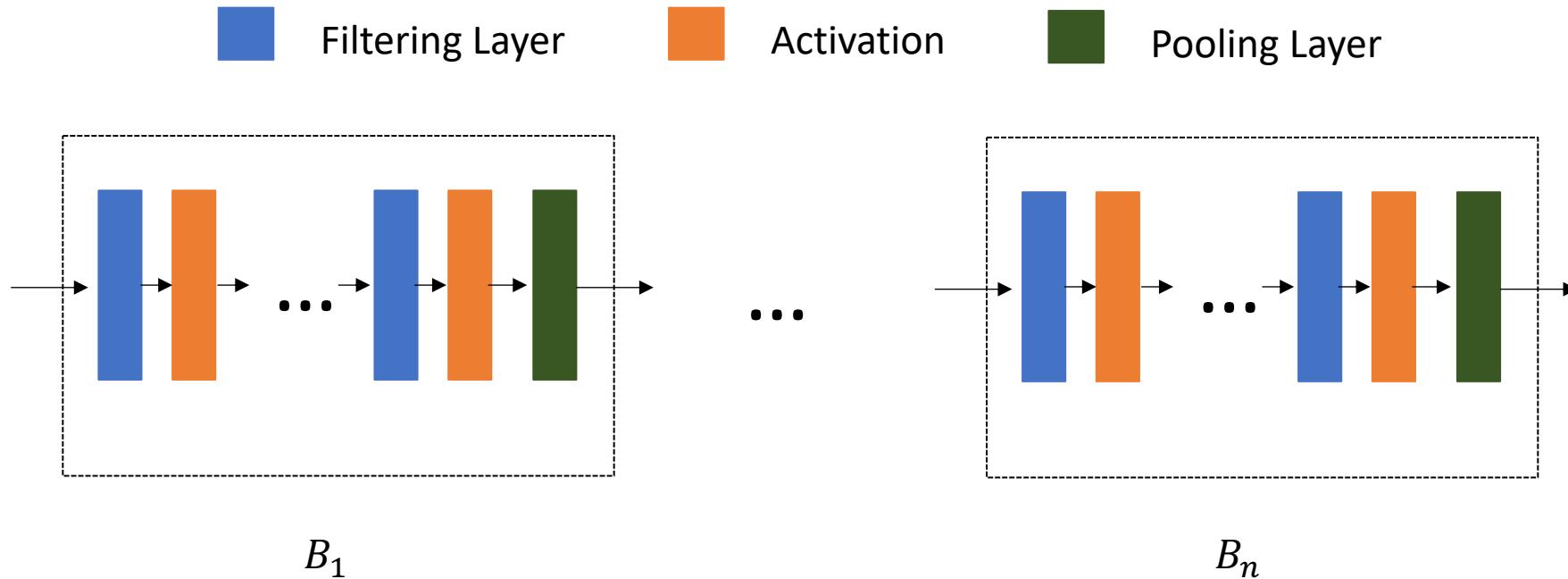
# General GNN Framework

For node-level tasks



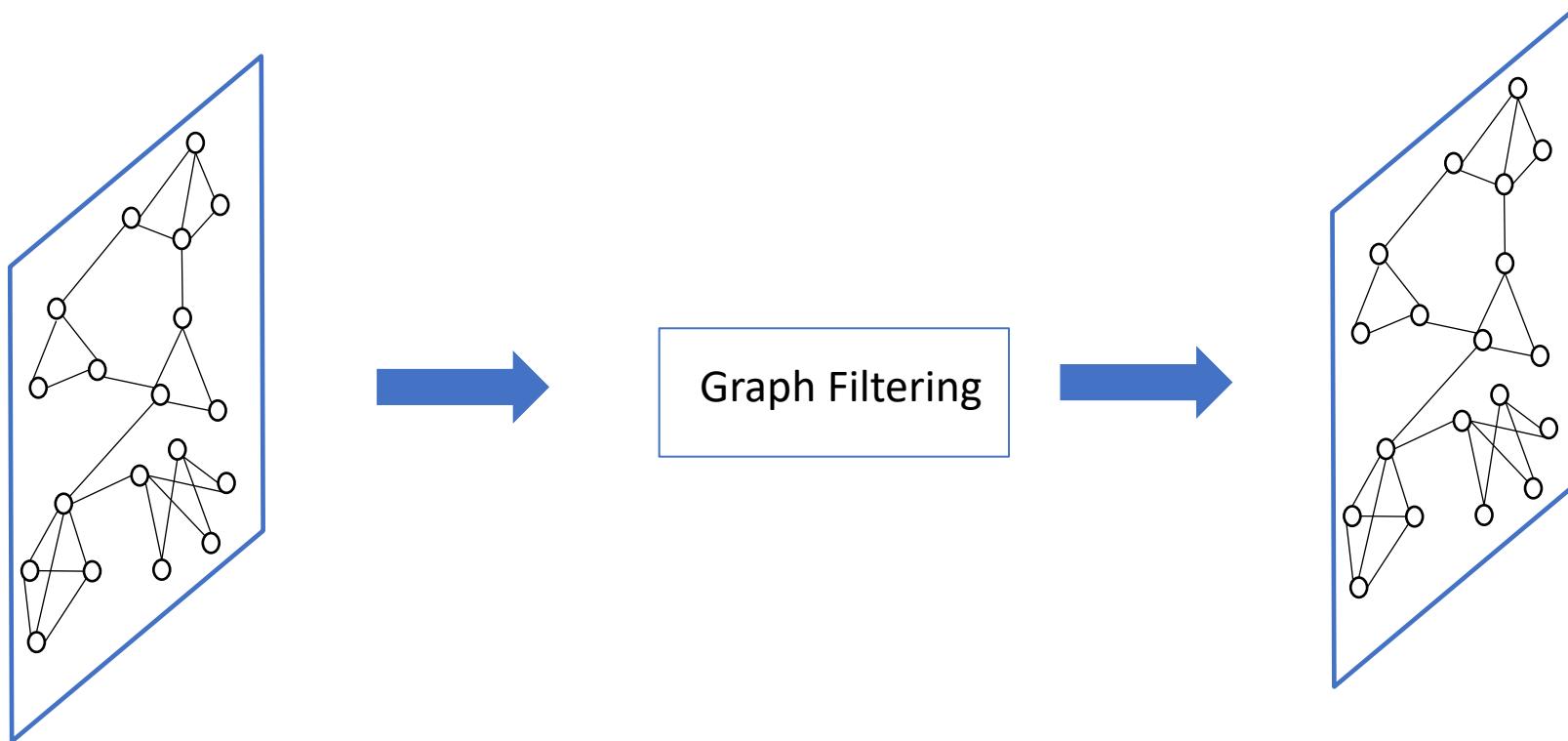
# General GNN Framework

For graph-level tasks





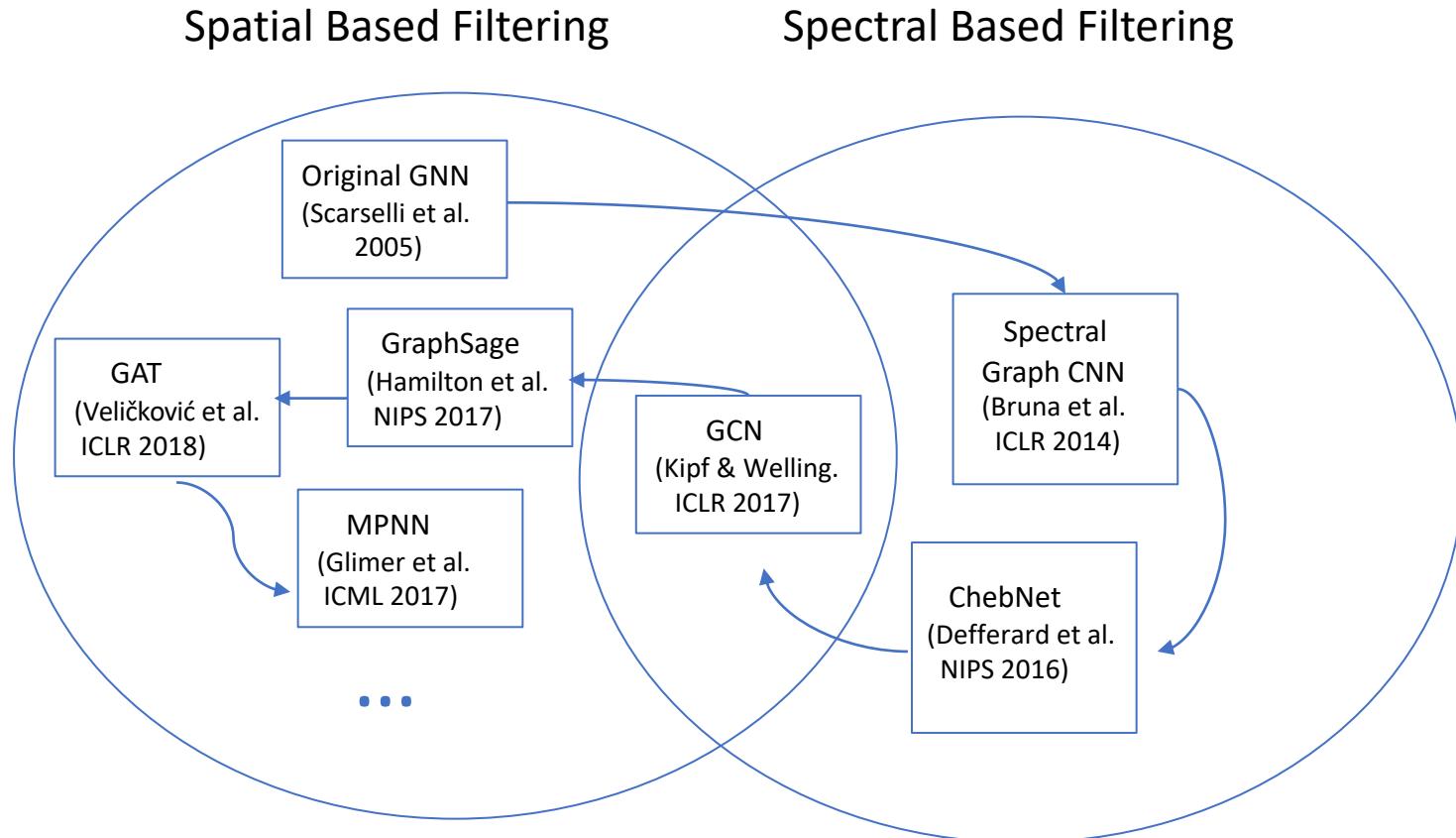
# Graph Filtering Operation



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X}_f \in \mathbb{R}^{n \times d_{new}}$$

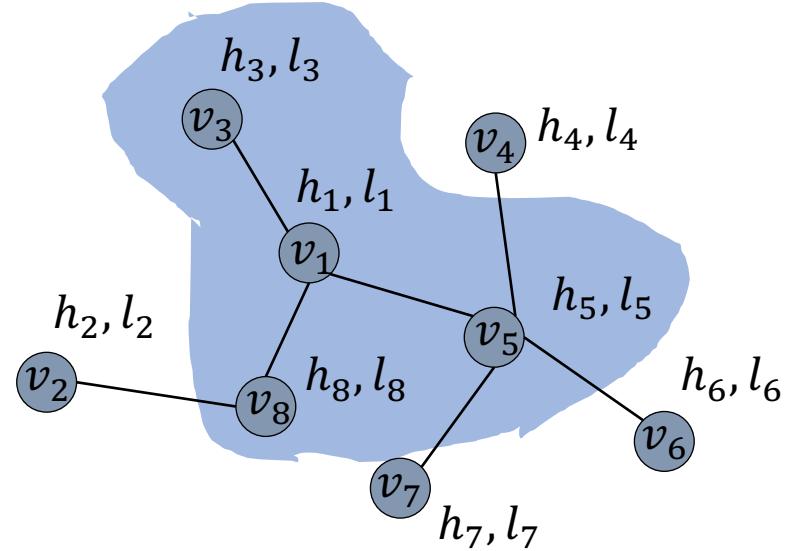
# Two Types of Graph Filtering Operation



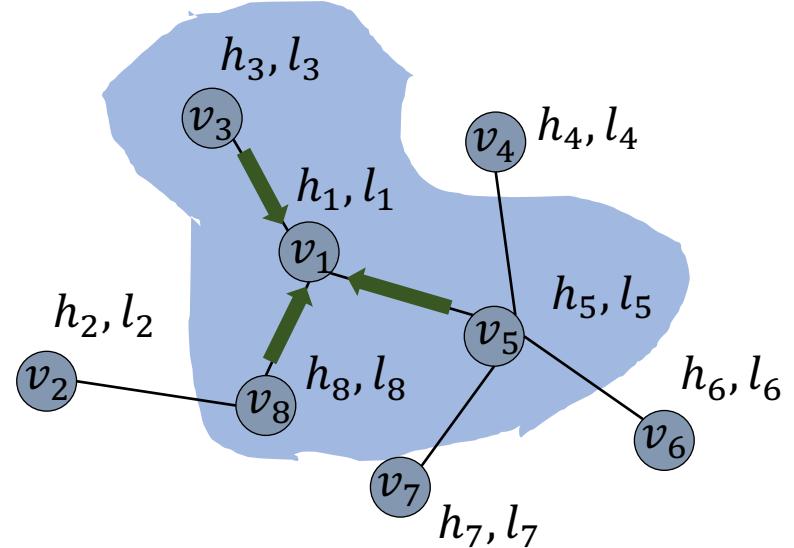
# Graph Filtering in the First GNN Paper

$h_i$ : The hidden features

$l_i$ : The input features



# Graph Filtering in the First GNN Paper



$h_i$ : The hidden features

$l_i$ : The input features

$$h_i^{(k+1)} = \sum_{v_j \in N(v_i)} f(l_i, h_j^{(k)}, l_j), \quad \forall v_i \in V.$$

$N(v_i)$ : Neighbors of the node  $v_i$ .

$f(\cdot)$ : Feedforward neural network.



# Graph Spectral Filtering for Graph Signal

Recall:

$$GFT: \hat{f} = U^T f \quad IGFT: f = U \hat{f}$$

Filter a graph signal  $f$ :

$$f$$

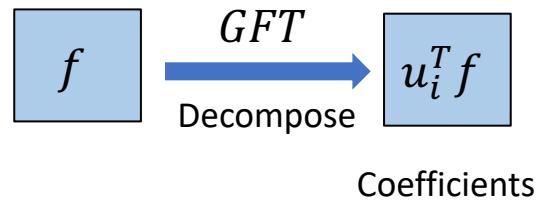


# Graph Spectral Filtering for Graph Signal

Recall:

$$GFT: \hat{f} = U^T f \quad IGFT: f = U \hat{f}$$

Filter a graph signal  $f$ :



Coefficients

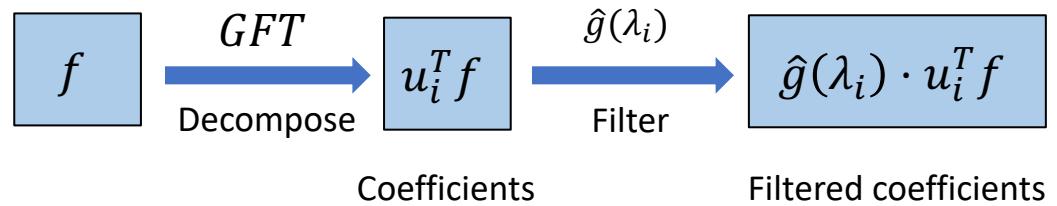


# Graph Spectral Filtering for Graph Signal

Recall:

$$GFT: \hat{f} = U^T f \quad IGFT: f = U \hat{f}$$

Filter a graph signal  $f$ :



Filter  $\hat{g}(\lambda_i)$ : Modulating the frequency

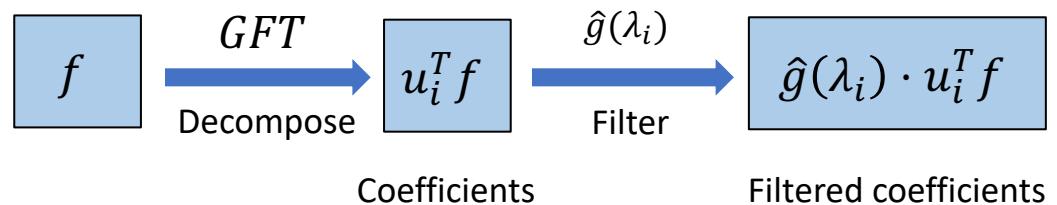


# Graph Spectral Filtering for Graph Signal

Recall:

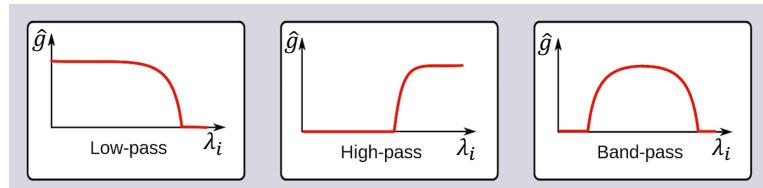
$$GFT: \hat{f} = U^T f \quad IGFT: f = U \hat{f}$$

Filter a graph signal  $f$ :



Example:

Filter  $\hat{g}(\lambda_i)$ : Modulating the frequency





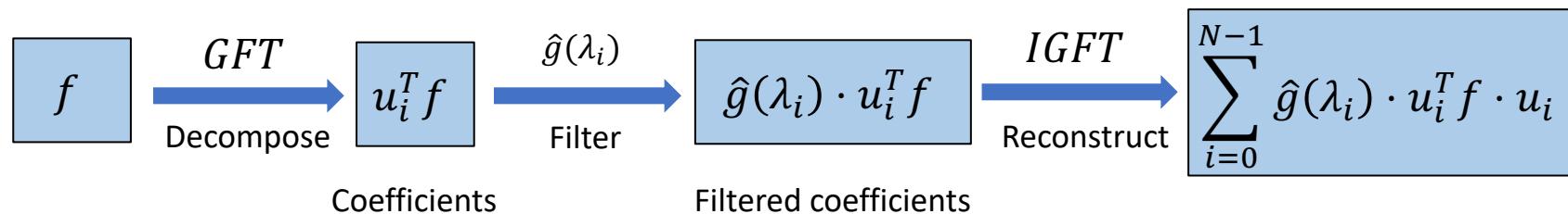
# Graph Spectral Filtering for Graph Signal

Recall:

$$GFT: \hat{f} = U^T f$$

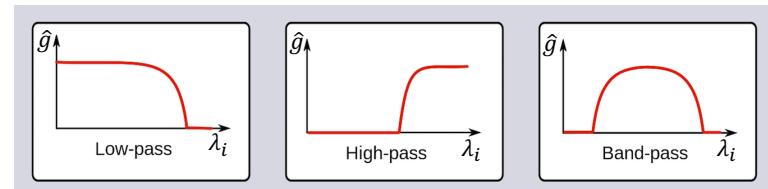
$$IGFT: f = U \hat{f}$$

Filter a graph signal  $f$ :



Example:

Filter  $\hat{g}(\lambda_i)$ : Modulating the frequency



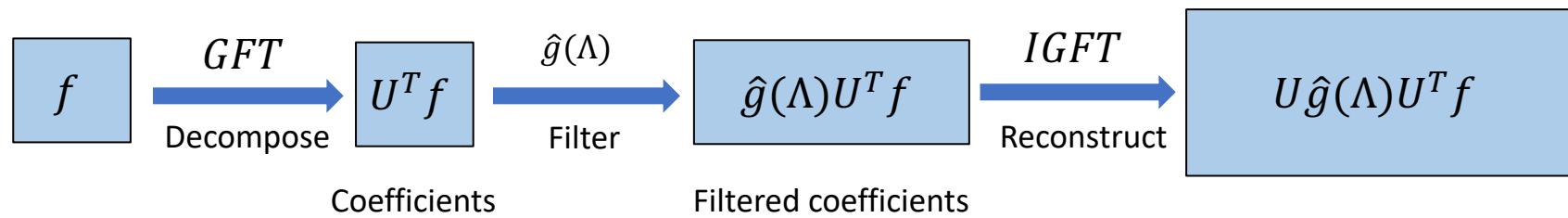


# Graph Spectral Filtering for Graph Signal

Recall:

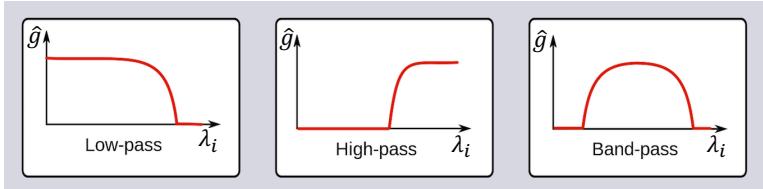
$$GFT: \hat{f} = U^T f \quad IGFT: f = U \hat{f}$$

Filter a graph signal  $f$ :



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

Example:





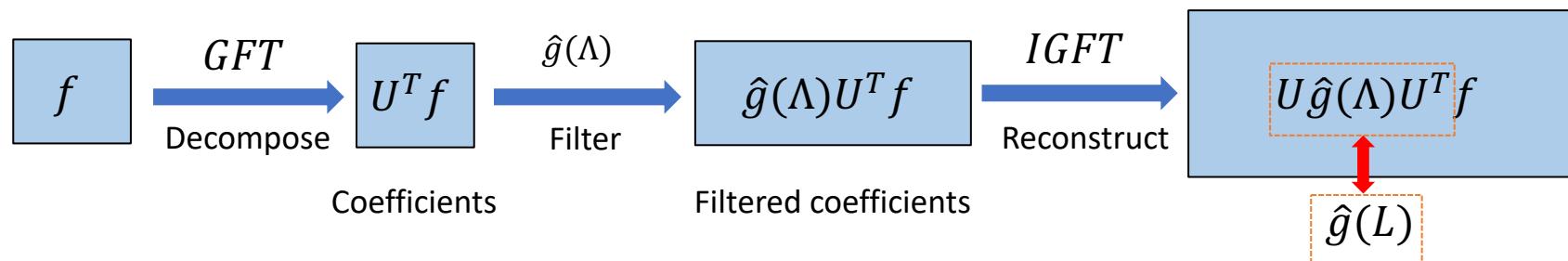
# Graph Spectral Filtering for Graph Signal

Recall:

$$GFT: \hat{f} = U^T f$$

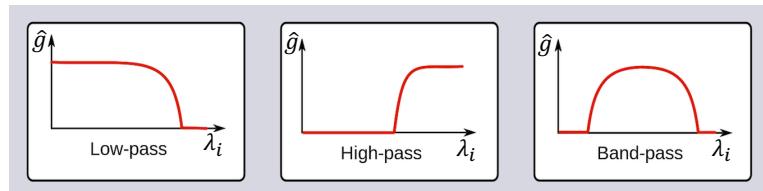
$$IGFT: f = U \hat{f}$$

Filter a graph signal  $f$ :



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

Example:





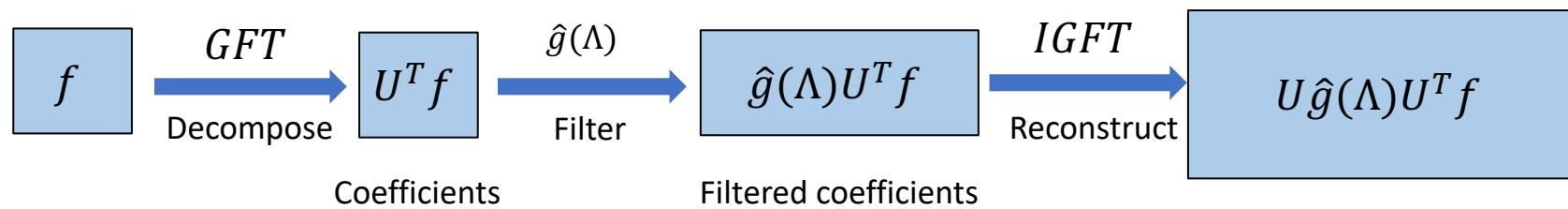
# Graph Spectral Filtering for Graph Signal

Recall:

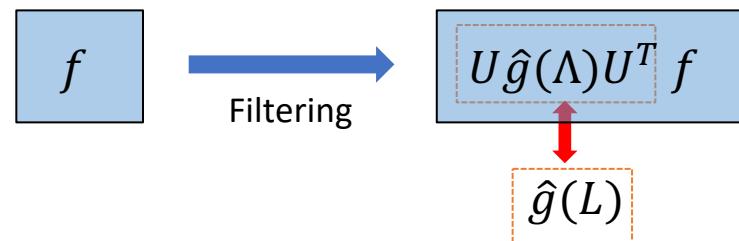
$$GFT: \hat{f} = U^T f$$

$$IGFT: f = U \hat{f}$$

Filter a graph signal  $f$ :



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





# Graph Spectral Filtering for GNN

How to design the filter?



# Graph Spectral Filtering for GNN

How to design the filter?

Data-driven! Learn  $\hat{g}(\Lambda)$  from data!



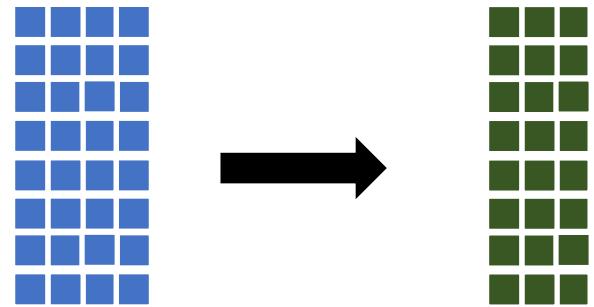
# Graph Spectral Filtering for GNN

How to design the filter?

Data-driven! Learn  $\hat{g}(\Lambda)$  from data!

How to deal with multi-channel signals?

$$\mathbf{F}_{in} \in \mathbb{R}^{N \times d_1} \rightarrow \mathbf{F}_{out} \in \mathbb{R}^{N \times d_2}.$$



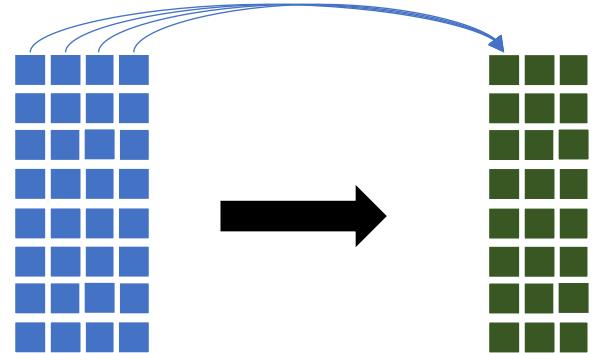
# Graph Spectral Filtering for GNN

How to design the filter?

Data-driven! Learn  $\hat{g}(\Lambda)$  from data!

How to deal with multi-channel signals?

$$\mathbf{F}_{in} \in \mathbb{R}^{N \times d_1} \rightarrow \mathbf{F}_{out} \in \mathbb{R}^{N \times d_2}.$$



Each input channel contributes to each output channel

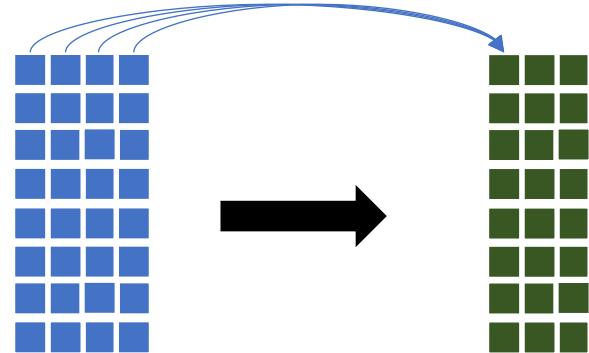
# Graph Spectral Filtering for GNN

How to design the filter?

Data-driven! Learn  $\hat{g}(\Lambda)$  from data!

How to deal with multi-channel signals?

$$\mathbf{F}_{in} \in \mathbb{R}^{N \times d_1} \rightarrow \mathbf{F}_{out} \in \mathbb{R}^{N \times d_2}.$$



Each input channel contributes to each output channel

$$\mathbf{F}_{out}[:, i] = \sum_{j=1}^{d_1} \underline{\hat{g}_{ij}(\mathbf{L}) \mathbf{F}_{in}[:, j]} \quad i = 1, \dots, d_2$$

Filter each input channel

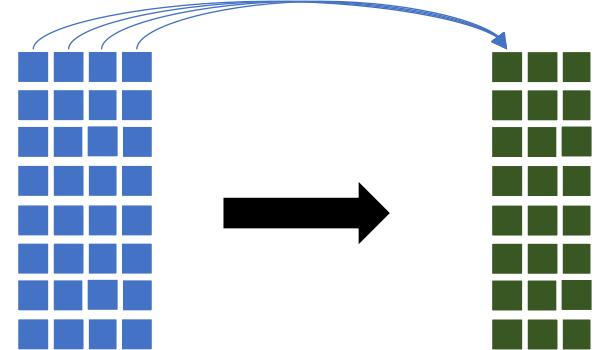
# Graph Spectral Filtering for GNN

How to design the filter?

Data-driven! Learn  $\hat{g}(\Lambda)$  from data!

How to deal with multi-channel signals?

$$\mathbf{F}_{in} \in \mathbb{R}^{N \times d_1} \rightarrow \mathbf{F}_{out} \in \mathbb{R}^{N \times d_2}.$$



Each input channel contributes to each output channel

$$\mathbf{F}_{out}[:, i] = \sum_{j=1}^{d_1} \underline{\hat{g}_{ij}(\mathbf{L}) \mathbf{F}_{in}[:, j]} \quad i = 1, \dots, d_2$$

Filter each input channel

Learn  $d_2 \times d_1$  filters



$\hat{g}(\Lambda)$ : Non-parametric

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



# $\hat{g}(\Lambda)$ : Non-parametric

$$\hat{g}(\Lambda) = \begin{bmatrix} \theta_1 & & & \\ & \theta_2 & & \\ & & \ddots & \\ & & & \theta_N \end{bmatrix}$$



$\hat{g}(\Lambda)$ : Non-parametric

$$\hat{g}(\Lambda) = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dots \\ \theta_N \end{bmatrix}$$

$d_2 \times d_1 \times N$  parameters



$\hat{g}(\Lambda)$ : Non-parametric

$$\hat{g}(\Lambda) = \begin{bmatrix} \theta_1 & & & \\ & \theta_2 & & \\ & & \ddots & \\ & & & \theta_N \end{bmatrix}$$

$d_2 \times d_1 \times N$  parameters

$$U\hat{g}(\Lambda)U^T f$$

Expensive eigen-decomposition



# $\hat{g}(\Lambda)$ : Polynomial Parametrized

$$\hat{g}(\Lambda) = \begin{bmatrix} \sum_{k=0}^K \theta_k \lambda_1^k \\ \sum_{k=0}^K \theta_k \lambda_2^k \\ \dots \\ \sum_{k=0}^K \theta_k \lambda_N^k \end{bmatrix}$$



# $\hat{g}(\Lambda)$ : Polynomial Parametrized

$$\hat{g}(\Lambda) = \begin{bmatrix} \sum_{k=0}^K \theta_k \lambda_1^k \\ \sum_{k=0}^K \theta_k \lambda_2^k \\ \dots \\ \sum_{k=0}^K \theta_k \lambda_N^k \end{bmatrix}$$

$d_2 \times d_1 \times K$  parameters



# $\hat{g}(\Lambda)$ : Polynomial Parametrized

$$\hat{g}(\Lambda) = \begin{bmatrix} \sum_{k=0}^K \theta_k \lambda_1^k \\ \sum_{k=0}^K \theta_k \lambda_2^k \\ \dots \\ \sum_{k=0}^K \theta_k \lambda_N^k \end{bmatrix}$$

$d_2 \times d_1 \times K$  parameters

$$U\hat{g}(\Lambda)U^T f = \sum_{k=0}^K \theta_k L^k f$$



# $\hat{g}(\Lambda)$ : Polynomial Parametrized

$$\hat{g}(\Lambda) = \begin{bmatrix} \sum_{k=0}^K \theta_k \lambda_1^k \\ \sum_{k=0}^K \theta_k \lambda_2^k \\ \dots \\ \sum_{k=0}^K \theta_k \lambda_N^k \end{bmatrix}$$

$d_2 \times d_1 \times K$  parameters

$$U\hat{g}(\Lambda)U^T f = \sum_{k=0}^K \theta_k L^k f$$

No eigen-decomposition needed



# Polynomial Parametrized Filter: a Spatial View

$$U\hat{g}(\Lambda)U^T f(i) = \sum_{j=0}^N \sum_{k=0}^K \theta_k L_{i,j}^k f(j)$$



# Polynomial Parametrized Filter: a Spatial View

$$U\hat{g}(\Lambda)U^T f(i) = \sum_{j=0}^N \sum_{k=0}^K \theta_k L_{i,j}^k f(j)$$

If the node  $v_j$  is more than  $K$ -hops away from node  $v_i$ , then,

$$\sum_{k=0}^K \theta_k L_{i,j}^k = 0$$



# Polynomial Parametrized Filter: a Spatial View

$$U\hat{g}(\Lambda)U^T f(i) = \sum_{j=0}^N \sum_{k=0}^K \theta_k L_{i,j}^k f(j)$$

If the node  $v_j$  is more than  $K$ -hops away from node  $v_i$ , then,

$$\sum_{k=0}^K \theta_k L_{i,j}^k = 0$$

The filter is localized within  $K$ -hops neighbors in the spatial domain



# Chebyshev Polynomials

The polynomials adopted have **non-orthogonal** basis  $1, x, x^2, x^3, \dots$

$$g(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots$$



# Chebyshev Polynomials

The polynomials adopted have **non-orthogonal** basis  $1, x, x^2, x^3, \dots$

$$g(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots$$

Unstable under perturbation of coefficients



# Chebyshev Polynomials

The polynomials adopted have **non-orthogonal** basis  $1, x, x^2, x^3, \dots$

$$g(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots$$

**Unstable under perturbation of coefficients**

Chebyshev polynomials:



# Chebyshev Polynomials

The polynomials adopted have **non-orthogonal** basis  $1, x, x^2, x^3, \dots$

$$g(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots$$

**Unstable under perturbation of coefficients**

Chebyshev polynomials:

Recursive definition:

- $T_0(x) = 1; T_1(x) = x$
- $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$



# Chebyshev Polynomials

The polynomials adopted have **non-orthogonal basis**  $1, x, x^2, x^3, \dots$

$$g(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots$$

**Unstable under perturbation of coefficients**

**Chebyshev polynomials:**

Recursive definition:

- $T_0(x) = 1; T_1(x) = x$
- $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$

The Chebyshev polynomials  $\{T_k\}$  form an **orthogonal basis** for the Hilbert space  $L^2([-1,1], \frac{dy}{\sqrt{1-y^2}})$ .



# Chebyshev Polynomials

The polynomials adopted have **non-orthogonal basis**  $1, x, x^2, x^3, \dots$

$$g(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots$$

**Unstable under perturbation of coefficients**

Chebyshev polynomials:

Recursive definition:

- $T_0(x) = 1; T_1(x) = x$
- $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$

The Chebyshev polynomials  $\{T_k\}$  form an **orthogonal** basis for the Hilbert space  $L^2([-1,1], \frac{dy}{\sqrt{1-y^2}})$ .

$$g(x) = \theta_0 T_0(x) + \theta_1 T_1(x) + \theta_2 T_2(x) + \dots$$



# ChebNet

Parametrize  $\hat{g}(\Lambda)$  with Chebyshev polynomials

$$\hat{g}(\Lambda) = \sum_{k=0}^K \theta_k T_k(\tilde{\Lambda}), \text{ with } \tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - 1$$



# ChebNet

Parametrize  $\hat{g}(\Lambda)$  with Chebyshev polynomials

$$\hat{g}(\Lambda) = \sum_{k=0}^K \theta_k T_k(\tilde{\Lambda}), \text{ with } \tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - 1$$

$d_2 \times d_1 \times K$  parameters



# ChebNet

Parametrize  $\hat{g}(\Lambda)$  with Chebyshev polynomials

$$\hat{g}(\Lambda) = \sum_{k=0}^K \theta_k T_k(\tilde{\Lambda}), \text{ with } \tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - 1$$

$d_2 \times d_1 \times K$  parameters

$$U\hat{g}(\Lambda)U^T f = \sum_{k=0}^K \theta_k T_k(\tilde{L}) f, \text{ with } \tilde{L} = \frac{2L}{\lambda_{max}} - I$$

No eigen-decomposition needed



# ChebNet

Parametrize  $\hat{g}(\Lambda)$  with Chebyshev polynomials

$$\hat{g}(\Lambda) = \sum_{k=0}^K \theta_k T_k(\tilde{\Lambda}), \text{ with } \tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - 1$$

$d_2 \times d_1 \times K$  parameters

$$U\hat{g}(\Lambda)U^T f = \sum_{k=0}^K \theta_k T_k(\tilde{L}) f, \text{ with } \tilde{L} = \frac{2L}{\lambda_{max}} - I$$

No eigen-decomposition needed

Stable under perturbation of coefficients



# GCN: Simplified ChebNet

Use Chebyshev polynomials with  $K = 1$  and assume  $\lambda_{max} = 2$

$$\hat{g}(\Lambda) = \theta_0 + \theta_1(\Lambda - I)$$



# GCN: Simplified ChebNet

Use Chebyshev polynomials with  $K = 1$  and assume  $\lambda_{max} = 2$

$$\hat{g}(\Lambda) = \theta_0 + \theta_1(\Lambda - I)$$

Further constrain  $\theta = \theta_0 = -\theta_1$

$$\hat{g}(\Lambda) = \theta(2I - \Lambda)$$

$$U\hat{g}(\Lambda)U^T f = \theta(2I - L)f = \theta \left( I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \right) f$$



# GCN: Simplified ChebNet

Use Chebyshev polynomials with  $K = 1$  and assume  $\lambda_{max} = 2$

$$\hat{g}(\Lambda) = \theta_0 + \theta_1(\Lambda - I)$$

Further constrain  $\theta = \theta_0 = -\theta_1$

$$\hat{g}(\Lambda) = \theta(2I - \Lambda)$$

$$U\hat{g}(\Lambda)U^T f = \theta(2I - L)f = \theta \left( I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \right) f$$

Apply a renormalization trick

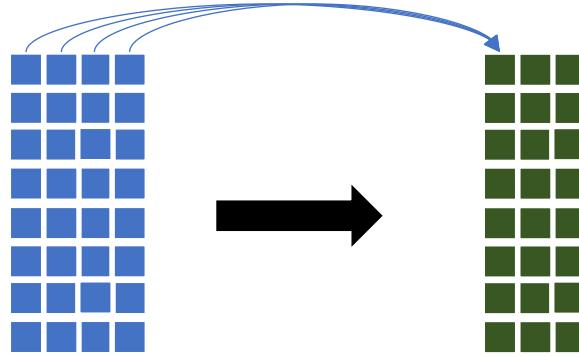
$$U\hat{g}(\Lambda)U^T f = \theta \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right) f, \text{ with } \hat{A} = A + I$$

# GCN for Multi-channel Signal

Recall:

$$\mathbf{F}_{out}[:, i] = \sum_{j=1}^{d_1} \hat{g}_{ij}(\mathbf{L}) \mathbf{F}_{in}[:, j] \quad i = 1, \dots d_2$$

Filter each input channel

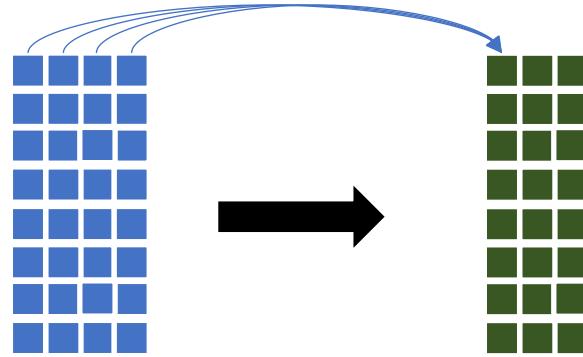


# GCN for Multi-channel Signal

Recall:

$$\mathbf{F}_{out}[:, i] = \sum_{j=1}^{d_1} \hat{g}_{ij}(\mathbf{L}) \mathbf{F}_{in}[:, j] \quad i = 1, \dots d_2$$

Filter each input channel



For GCN:

$$\mathbf{F}_{out}[:, i] = \sum_{j=1}^{d_1} \theta_{ji} (\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}) \mathbf{F}_{in}[:, j] \quad i = 1, \dots d_2$$

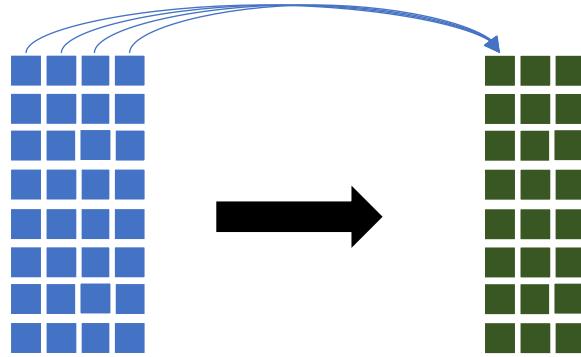
GCN filter

# GCN for Multi-channel Signal

Recall:

$$\mathbf{F}_{out}[:, i] = \sum_{j=1}^{d_1} \hat{g}_{ij}(\mathbf{L}) \mathbf{F}_{in}[:, j] \quad i = 1, \dots, d_2$$

Filter each input channel



For GCN:

$$\mathbf{F}_{out}[:, i] = \sum_{j=1}^{d_1} \theta_{ji} (\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}) \mathbf{F}_{in}[:, j] \quad i = 1, \dots, d_2$$

GCN filter

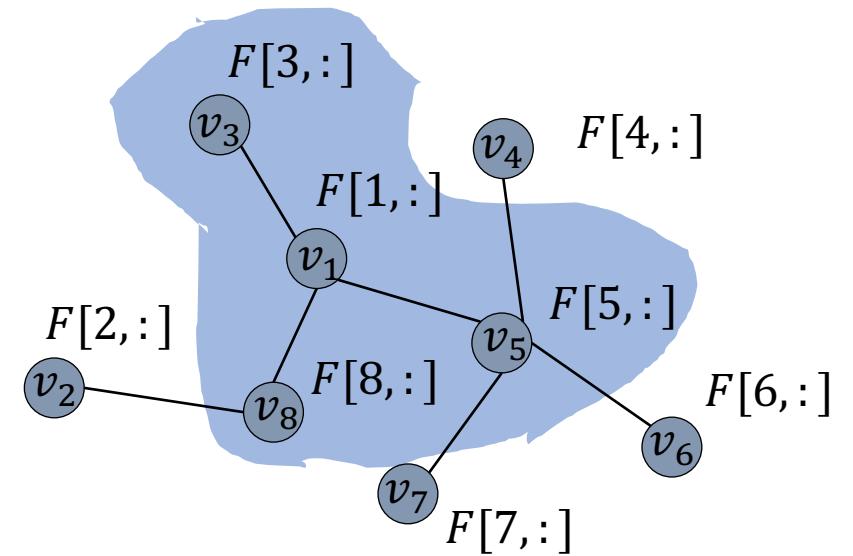
In matrix form:

$$\mathbf{F}_{out} = (\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}) \mathbf{F}_{in} \Theta \text{ with } \Theta \in \mathbb{R}^{d_1 \times d_2} \text{ and } \Theta[j, i] = \theta_{ji}$$



# A Spatial View of GCN Filter

Denote  $C = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

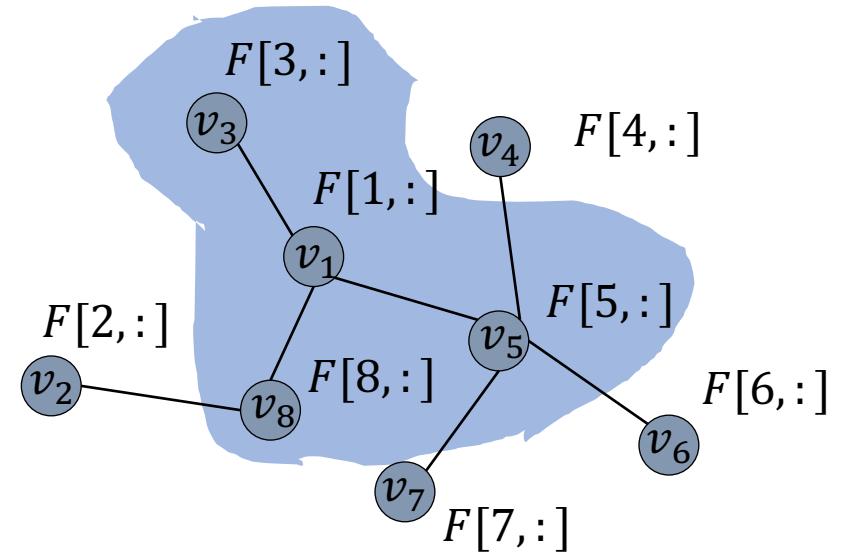




# A Spatial View of GCN Filter

Denote  $C = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

- Then  $F_{out} = CF_{in}\Theta$
- For node  $v_i$ ,  $F_{out}[i, :] = \sum_j C[i, j]F_{in}[j, :]\Theta$





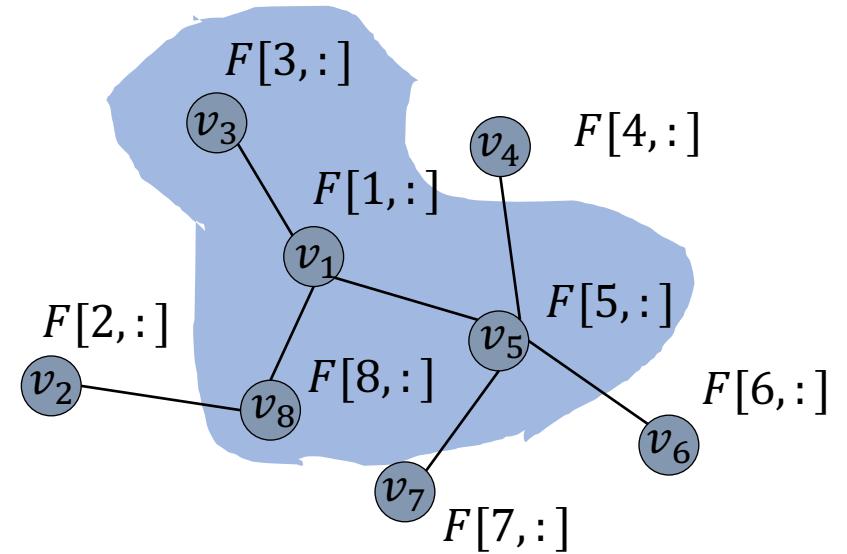
# A Spatial View of GCN Filter

Denote  $C = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

- Then  $F_{out} = CF_{in}\Theta$
- For node  $v_i$ ,  $F_{out}[i, :] = \sum_j C[i, j]F_{in}[j, :]\Theta$

Observe that:

- $C[i, j] = 0$  for  $v_j \notin N(v_i) \cup \{v_i\}$





# A Spatial View of GCN Filter

Denote  $C = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

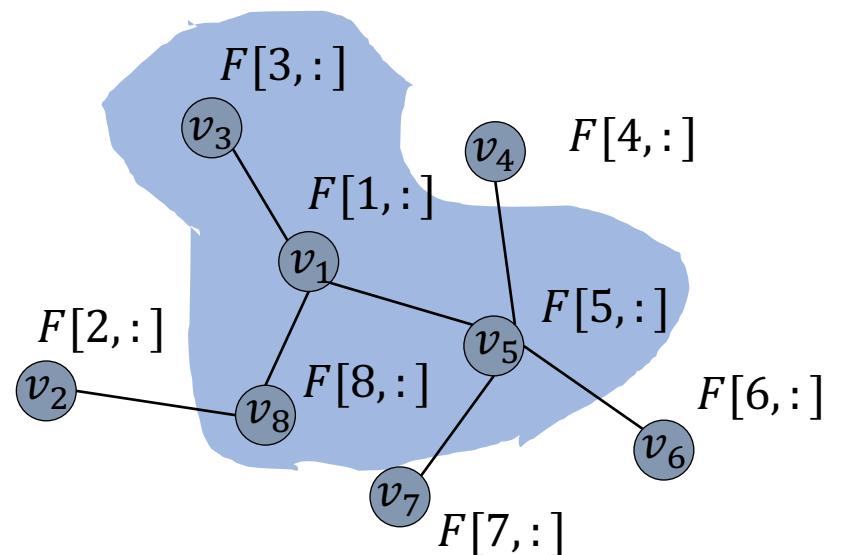
- Then  $F_{out} = CF_{in}\Theta$
- For node  $v_i$ ,  $F_{out}[i, :] = \sum_j C[i, j]F_{in}[j, :]\Theta$

Observe that:

- $C[i, j] = 0$  for  $v_j \notin N(v_i) \cup \{v_i\}$

Hence,

$$\mathbf{F}_{out}[i, :] = \sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} C[i, j] \mathbf{F}_{in}[j, :] \Theta$$





# A Spatial View of GCN Filter

Denote  $C = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

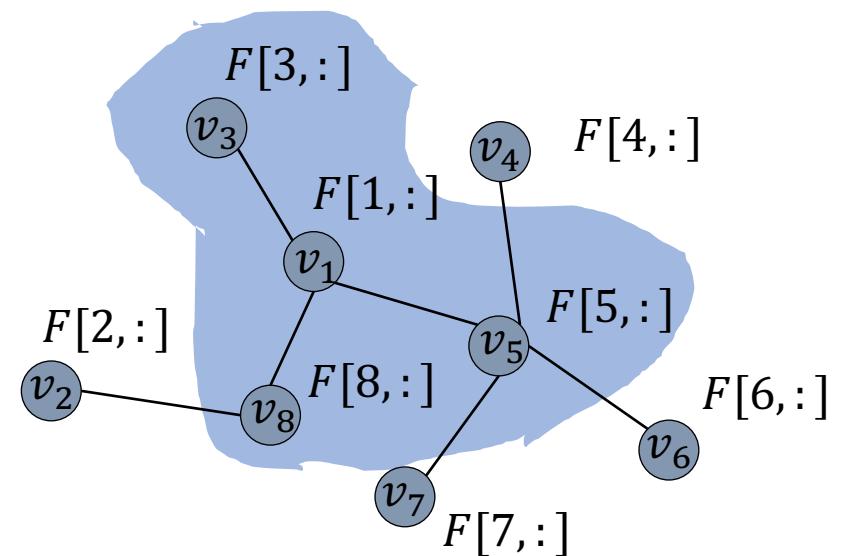
- Then  $F_{out} = CF_{in}\Theta$
- For node  $v_i$ ,  $F_{out}[i, :] = \sum_j C[i, j]F_{in}[j, :]\Theta$

Observe that:

- $C[i, j] = 0$  for  $v_j \notin N(v_i) \cup \{v_i\}$

Hence,

$$\mathbf{F}_{out}[i, :] = \sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} C[i, j] \underline{\mathbf{F}_{in}[j, :]\Theta} \quad \text{Feature transformation}$$



# A Spatial View of GCN Filter

Denote  $C = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

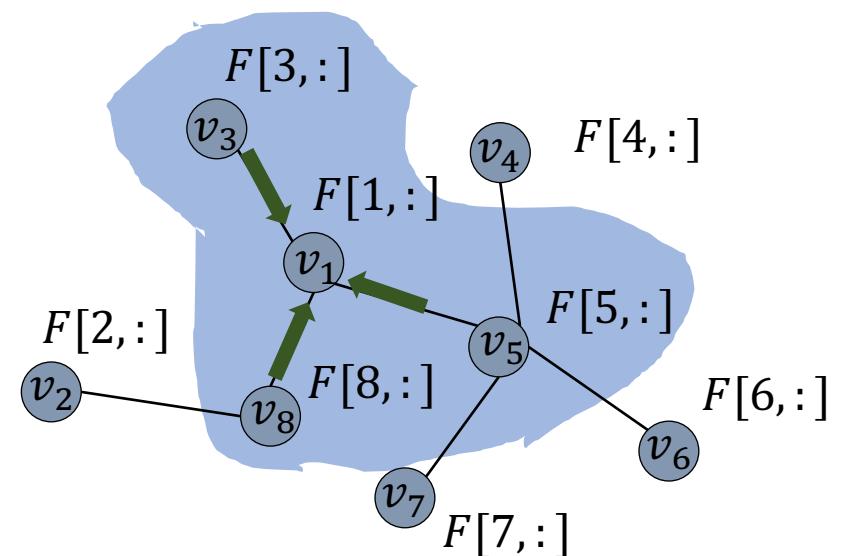
- Then  $F_{out} = CF_{in}\Theta$
- For node  $v_i$ ,  $F_{out}[i, :] = \sum_j C[i, j]F_{in}[j, :] \Theta$

Observe that:

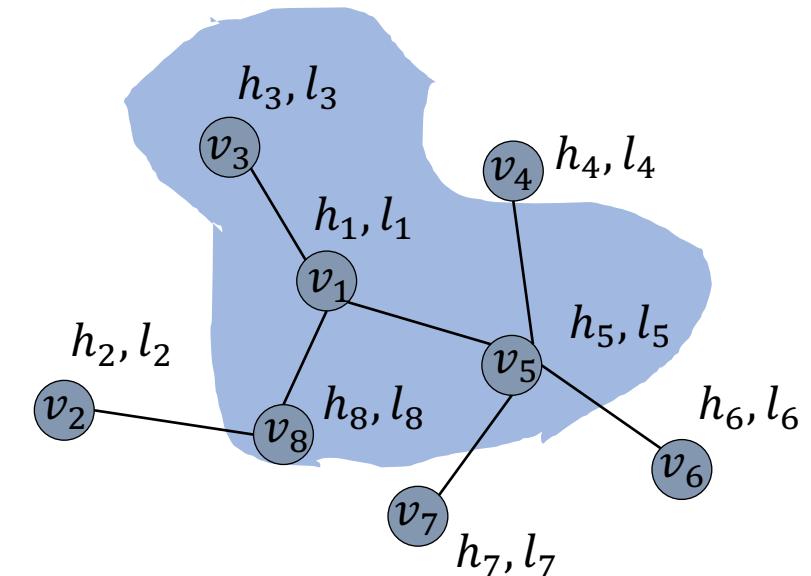
- $C[i, j] = 0$  for  $v_j \notin N(v_i) \cup \{v_i\}$

Hence,

$$F_{out}[i, :] = \underbrace{\sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} C[i, j] F_{in}[j, :] \Theta}_{\text{Aggregation}} \quad \underbrace{\Theta}_{\text{Feature transformation}}$$



# Filter in GCN VS Filter in the First GNN



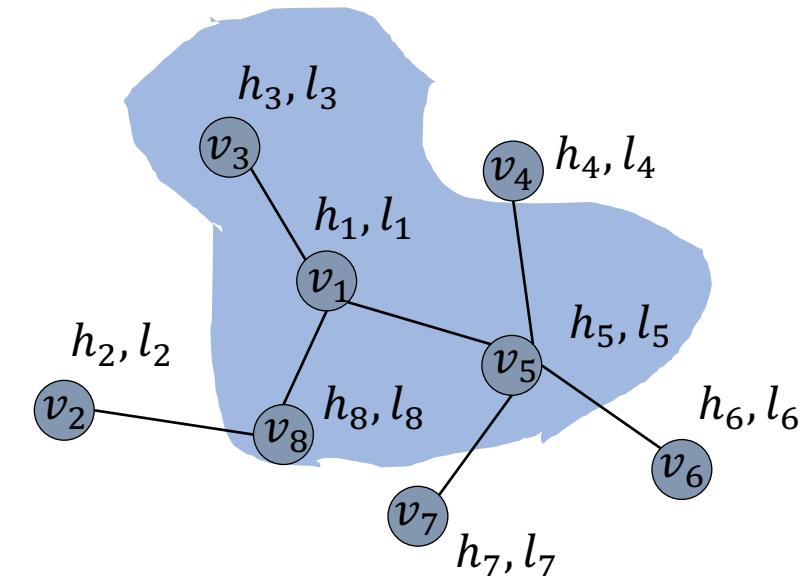
GCN: k-th layer

$$\mathbf{h}_i^{(k+1)} = \sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} C[i, j] \mathbf{h}_j^{(k)} \Theta, \forall v_i \in \mathcal{V}$$

The first GNN: k-th layer

$$\mathbf{h}_i^{(k+1)} = \sum_{v_j \in \mathcal{N}(v_i)} f(l_i, \mathbf{h}_j^{(k)}, l_j), \forall v_i \in \mathcal{V}$$

# Filter in GCN VS Filter in the First GNN



GCN: k-th layer

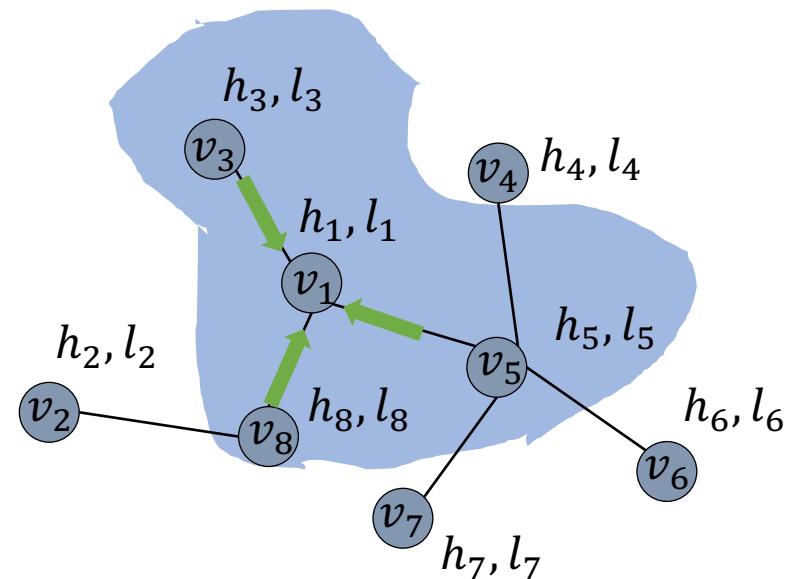
$$\mathbf{h}_i^{(k+1)} = \sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} C[i, j] \underline{\mathbf{h}_j^{(k)} \Theta}, \forall v_i \in \mathcal{V}$$

The first GNN: k-th layer

$$\mathbf{h}_i^{(k+1)} = \sum_{v_j \in \mathcal{N}(v_i)} \underline{f(l_i, \mathbf{h}_j^{(k)}, l_j)}, \forall v_i \in \mathcal{V}$$

# Filter in GCN VS Filter in the First GNN

GCN: k-th layer



$$\mathbf{h}_i^{(k+1)} = \underbrace{\sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} C[i, j] \mathbf{h}_j^{(k)} \Theta}_{\underline{\underline{\Theta}}}, \forall v_i \in \mathcal{V}$$

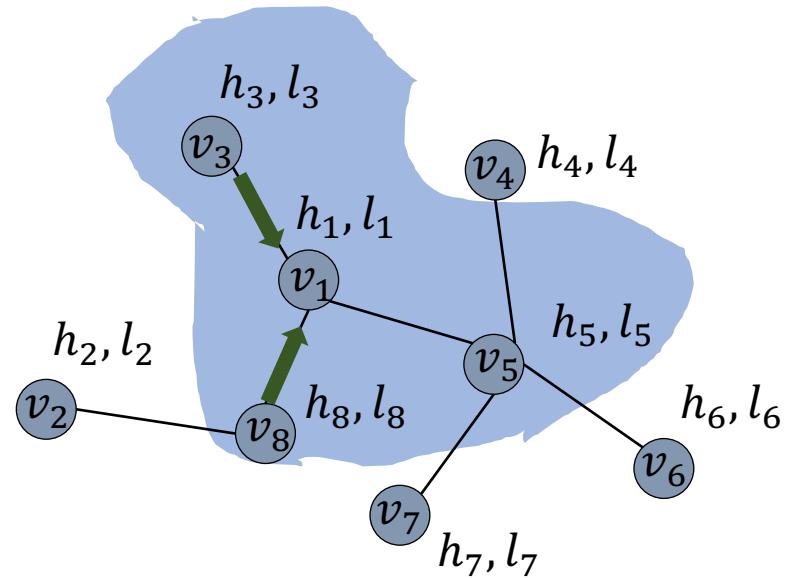
The first GNN: k-th layer

$$\mathbf{h}_i^{(k+1)} = \underbrace{\sum_{v_j \in \mathcal{N}(v_i)} f(l_i, \mathbf{h}_j^{(k)}, l_j)}_{\underline{\underline{f}}}, \forall v_i \in \mathcal{V}$$

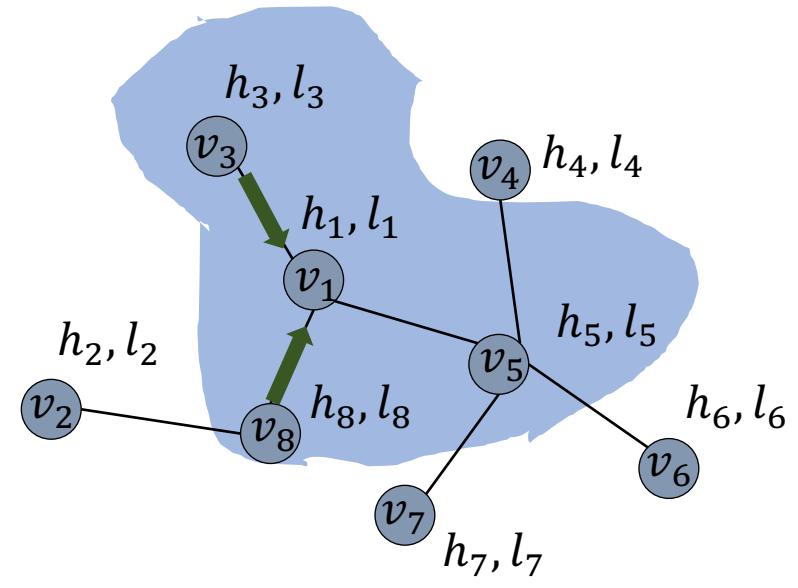
# Filter in GraphSage

## Neighbor Sampling

$$\mathcal{N}(v_i) \rightarrow \mathcal{N}_s(v_i)$$



# Filter in GraphSage



**Neighbor Sampling**

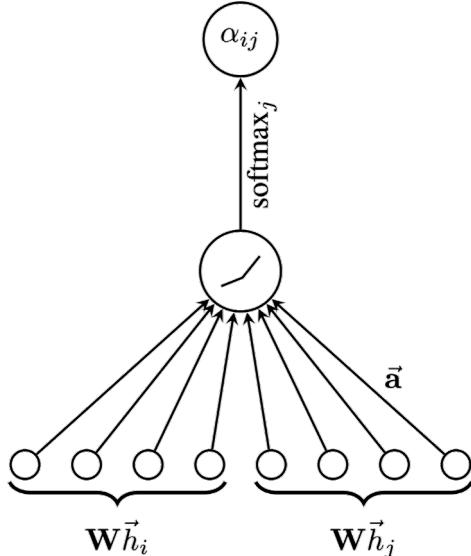
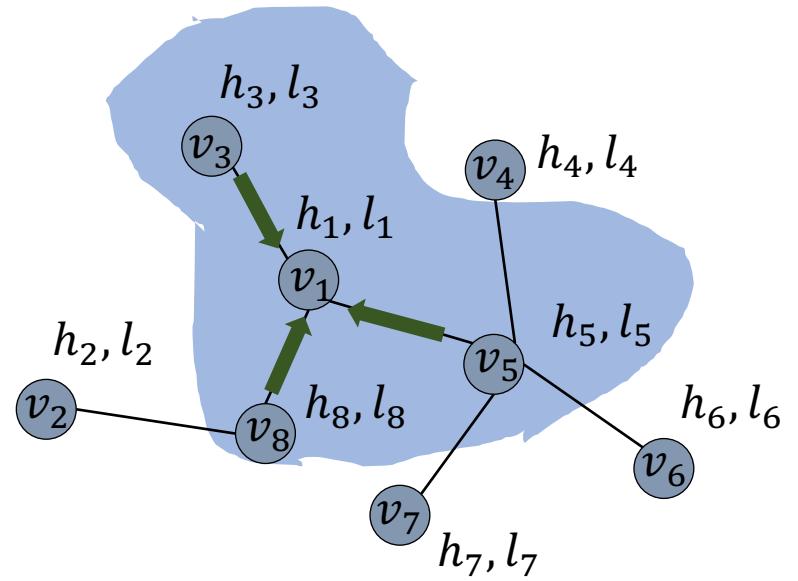
$$\mathcal{N}(v_i) \rightarrow \mathcal{N}_s(v_i)$$

**Aggregation**

$$\mathbf{h}_{\mathcal{N}_s(v_i)}^{(k+1)} = \underline{\text{AGG}}(\{\mathbf{h}_i^{(k)}, v_j \in \mathcal{N}_s(v_i)\})$$

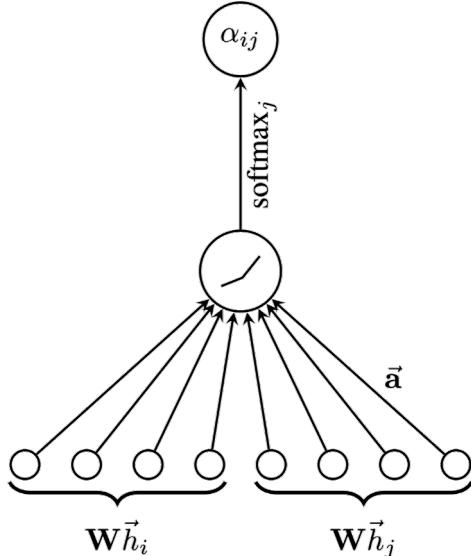
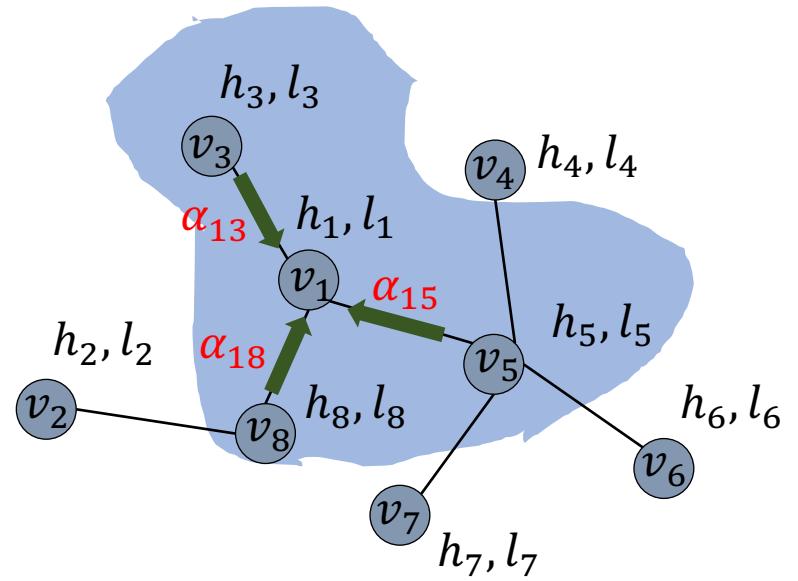
$$\mathbf{h}_i^{(k+1)} = \sigma(\Theta[\mathbf{h}_i^{(k)}, \mathbf{h}_{\mathcal{N}_s(v_i)}^{(k+1)}])$$

# Filter in GAT



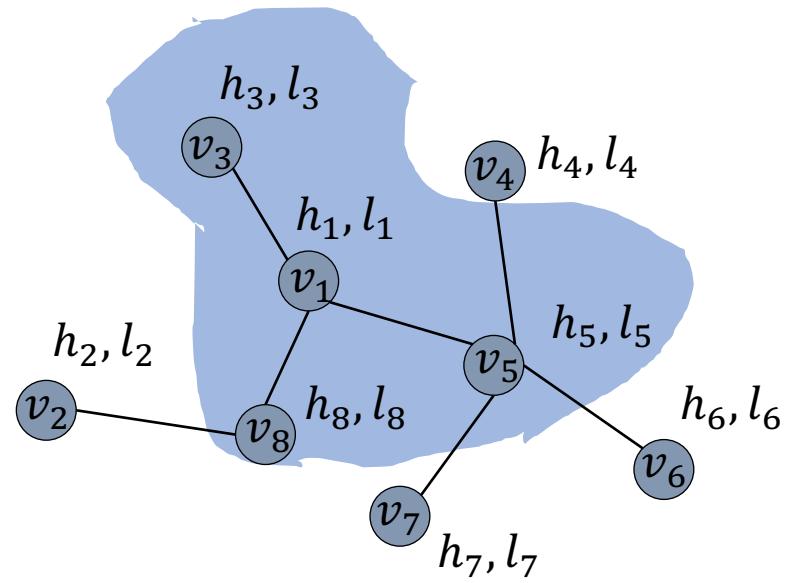
$$\alpha_{ij} = \frac{\exp \left( \text{LeakyReLU} \left( \vec{a}^T [\vec{W}^h_i \| \vec{W}^h_j] \right) \right)}{\sum_{k \in \mathcal{N}_i} \exp \left( \text{LeakyReLU} \left( \vec{a}^T [\vec{W}^h_i \| \vec{W}^h_k] \right) \right)}$$

# Filter in GAT



$$\alpha_{ij} = \frac{\exp \left( \text{LeakyReLU} \left( \vec{a}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j] \right) \right)}{\sum_{k \in \mathcal{N}_i} \exp \left( \text{LeakyReLU} \left( \vec{a}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k] \right) \right)}$$

# Filter in MPNN



## Message Passing

$$m_i^{(k+1)} = \sum_{v_j \in N(v_i)} M_k \left( h_i^{(k)}, h_j^{(k)}, e_{ij} \right)$$

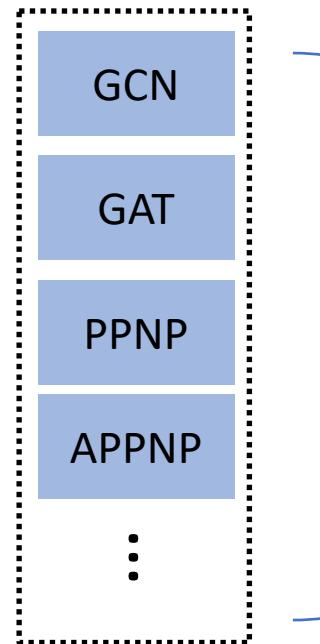
## Feature Updating

$$h_i^{(k+1)} = U_k \left( h_i^{(k)}, m_i^{(k+1)} \right)$$

$M_k()$  and  $U_k()$  are functions to be designed

# UGNN: A Unified Framework

Filtering Operations

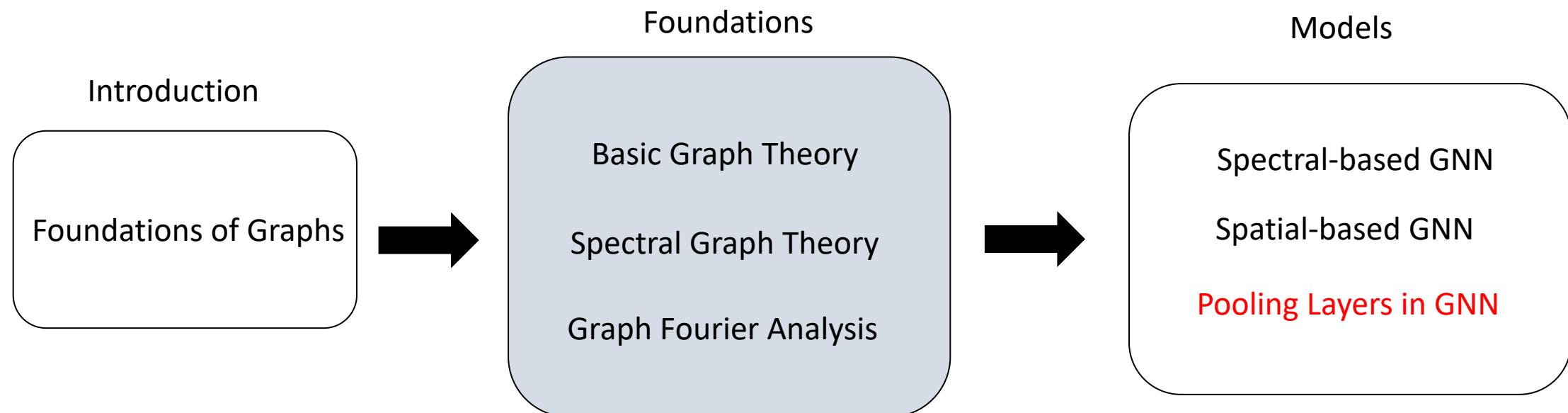


Graph Signal Denoising

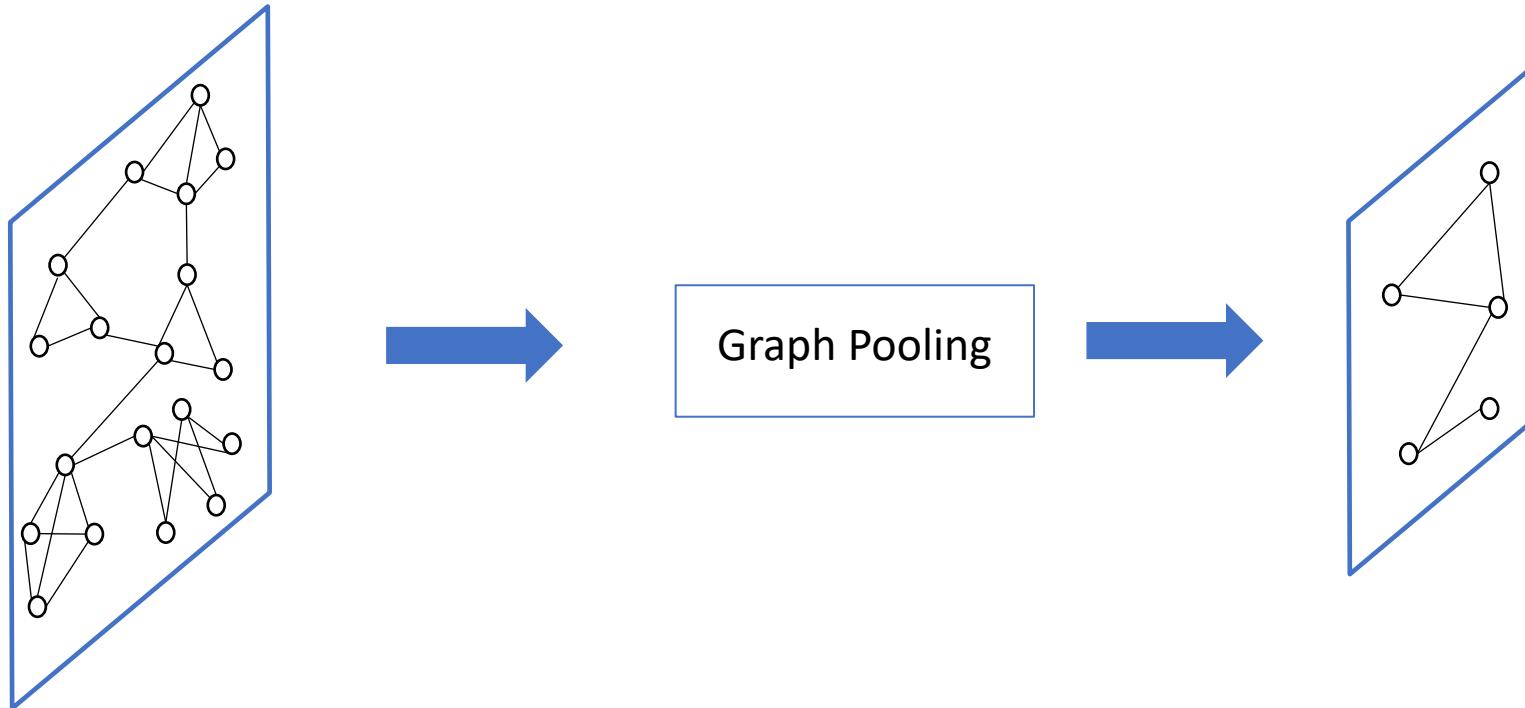
$$\arg \min_{\mathbf{X}_f} \mathcal{L}(\mathbf{X}_f) = \|\mathbf{X}_f - \mathbf{X}'\|_F^2 + c \cdot \frac{1}{2} \sum_{i \in \mathcal{V}} \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \|\mathbf{X}_f[i, :] - \mathbf{X}_f[j, :]\|_2^2$$



# Overview



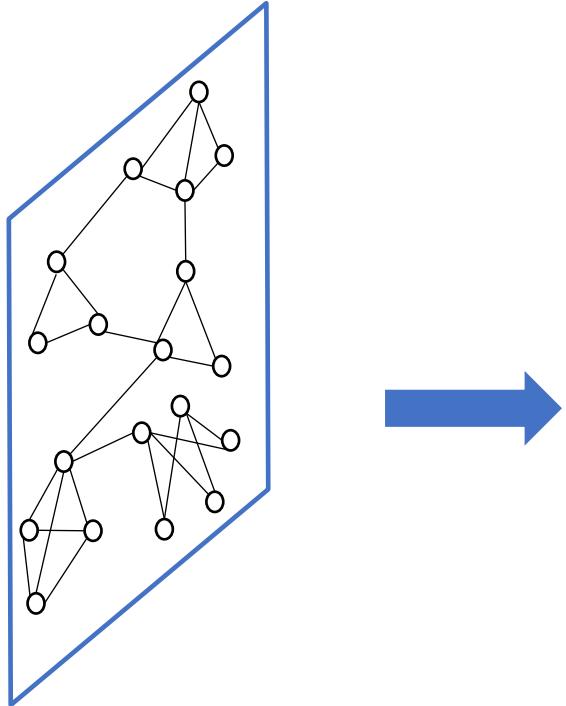
# Graph Pooling Operation



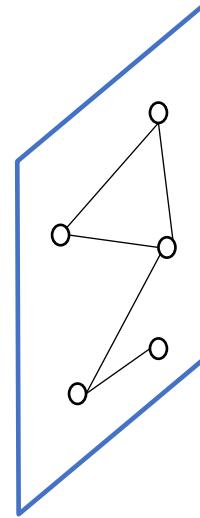
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{X} \in \mathbb{R}^{n \times d}$$

$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{X}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

# How to Perform Graph Pooling?

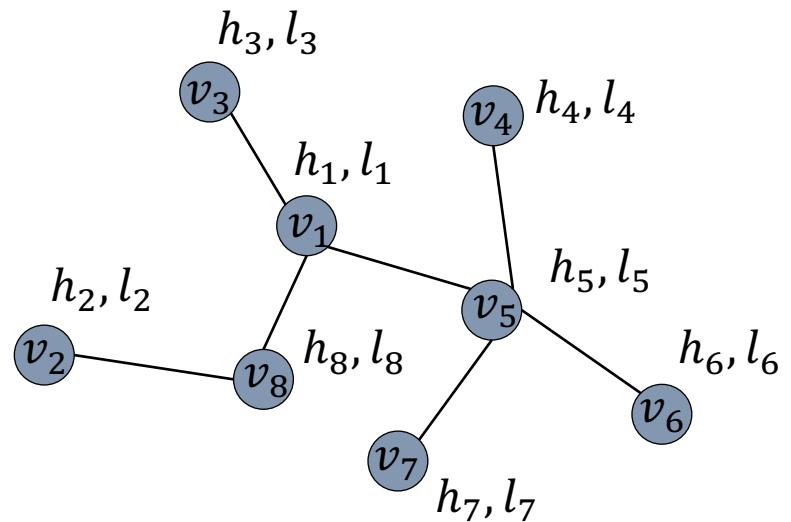


Node Selection  
or  
Node Grouping



# gPool

Downsample by selecting the most importance nodes



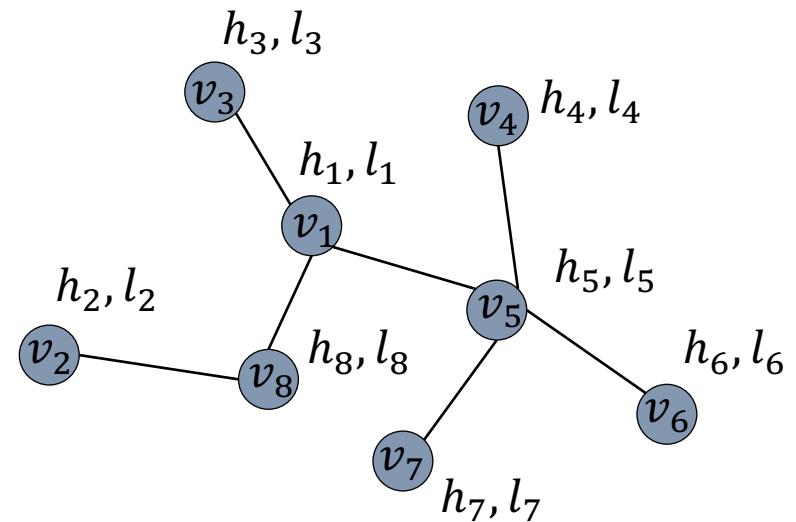
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

# gPool

Downsample by selecting the most importance nodes



Importance Measure

$$v_i \rightarrow y_i \quad y_i = \frac{h_i^T p}{\|p\|}$$

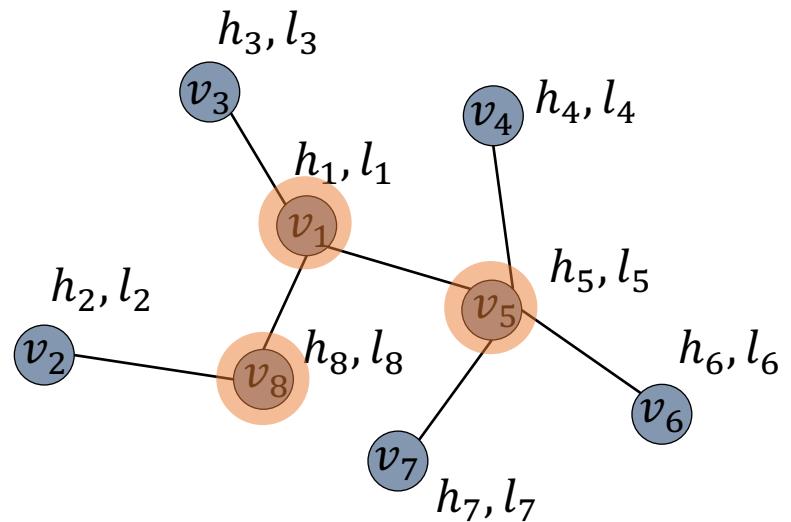
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

# gPool

Downsample by selecting the most importance nodes



Importance Measure

$$v_i \rightarrow y_i \quad y_i = \frac{h_i^T p}{\|p\|}$$

Select top the  $n_p$  nodes

$$idx = rank(\mathbf{y}, n_p)$$

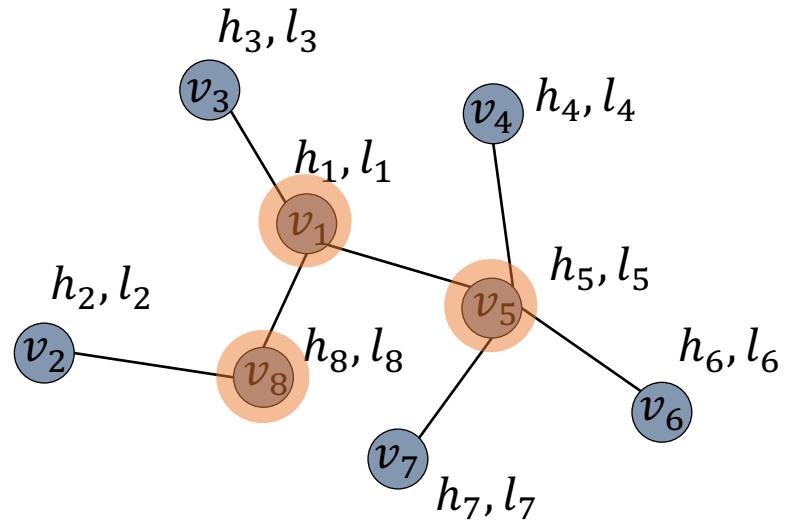
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

# gPool

Downsample by selecting the most importance nodes



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

Importance Measure

$$v_i \rightarrow y_i \quad y_i = \frac{h_i^T p}{\|p\|}$$

Select top the  $n_p$  nodes

$$idx = rank(\mathbf{y}, n_p)$$

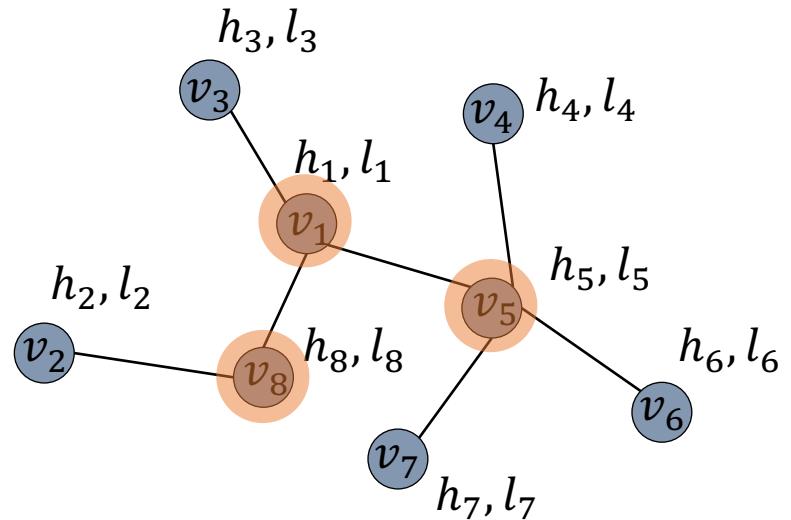
Generate  $A_p$  and intermediate  $H_{inter}$

$$A_p = A[idx, idx]$$

$$H_{inter} = H[idx, :]$$

# gPool

Downsample by selecting the most importance nodes



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

Importance Measure

$$v_i \rightarrow y_i \quad y_i = \frac{h_i^T p}{\|p\|}$$

Select top the  $n_p$  nodes

$$idx = rank(\mathbf{y}, n_p)$$

Generate  $A_p$  and intermediate  $H_{inter}$

$$A_p = A[idx, idx]$$

$$H_{inter} = H[idx, :]$$

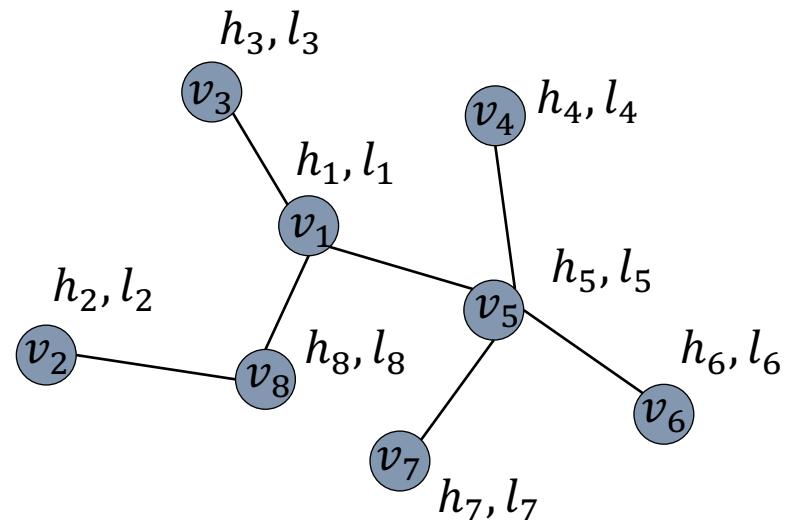
Generate  $H_p$

$$\tilde{y} = sigmoid(y[idx])$$

$$H_p = H_{inter} \odot \tilde{y}$$

# DiffPool

Downsample by clustering the nodes using GNN



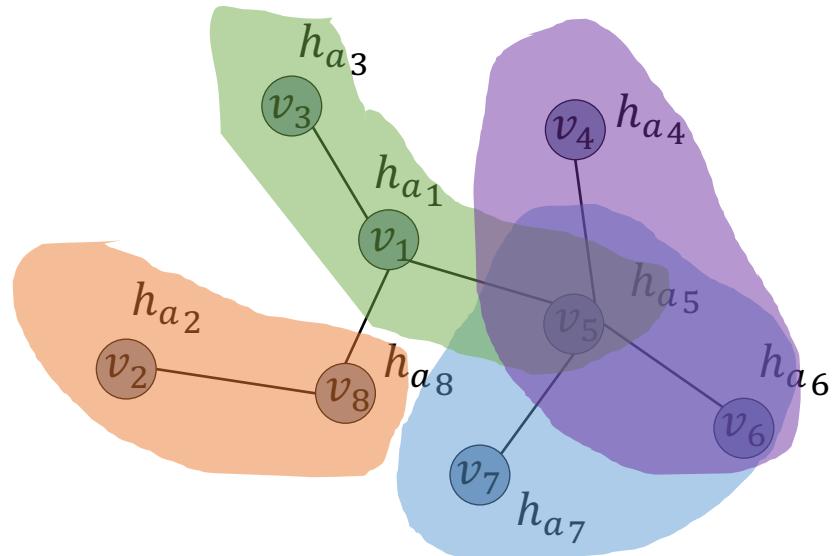
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

# DiffPool

Downsample by clustering the nodes using GNN



Filter1:  
Generate a soft-assign matrix

$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$

$$\downarrow$$

$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

2 filters

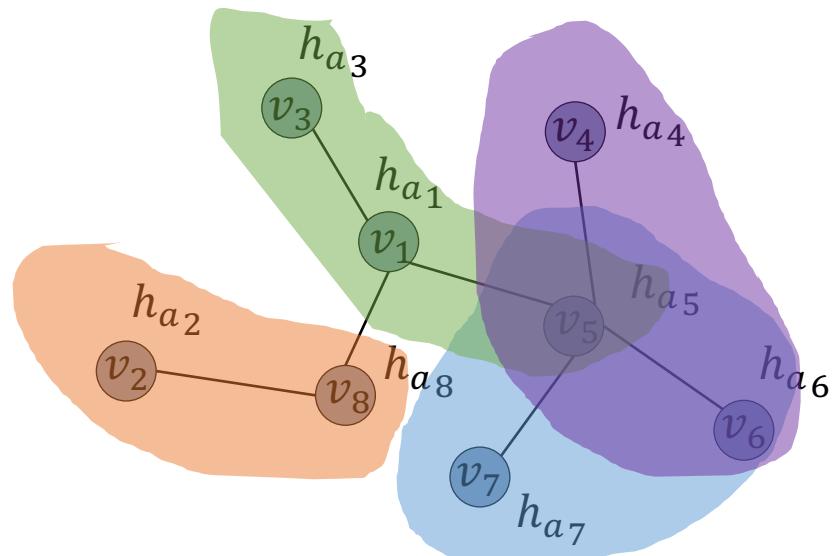
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H}_a \in \mathbb{R}^{n \times n_p}$$

# DiffPool

Downsample by clustering the nodes using GNN



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$

$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

Filter1:

Generate a soft-assign matrix

2 filters

$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H}_a \in \mathbb{R}^{n \times n_p}$$

Filter2:

Generate new features

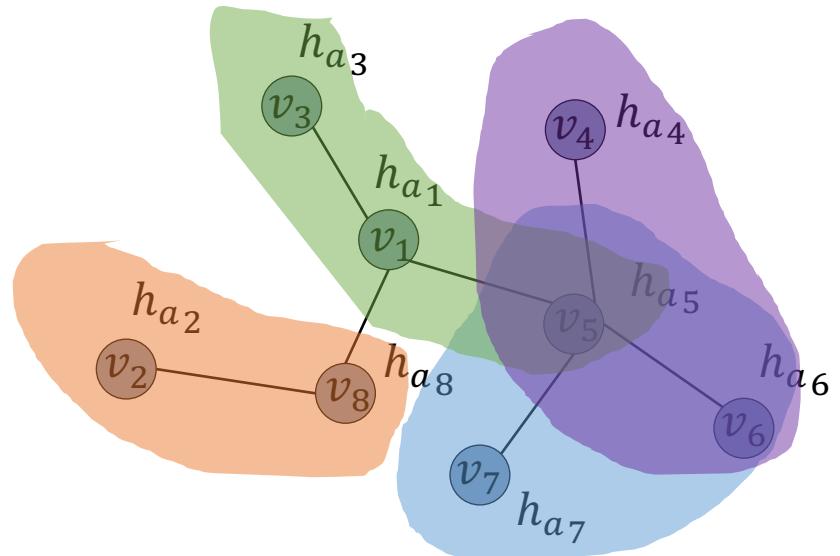
$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H}_f \in \mathbb{R}^{n \times d_{new}}$$

# DiffPool

Downsample by clustering the nodes using GNN



Generated soft-assign matrix

Generated new features

$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$

$$\downarrow$$

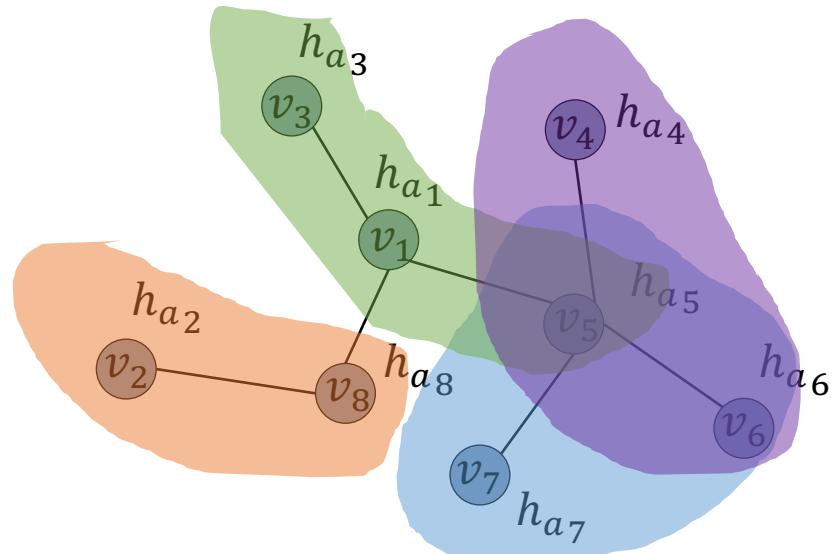
$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

$$\mathbf{H}_a \in \mathbb{R}^{n \times n_p}$$

$$\mathbf{H}_f \in \mathbb{R}^{n \times d_{new}}$$

# DiffPool

Downsample by clustering the nodes using GNN



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$

$$\downarrow$$

$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

Generated soft-assign matrix

$$\mathbf{H}_a \in \mathbb{R}^{n \times n_p}$$

Generated new features

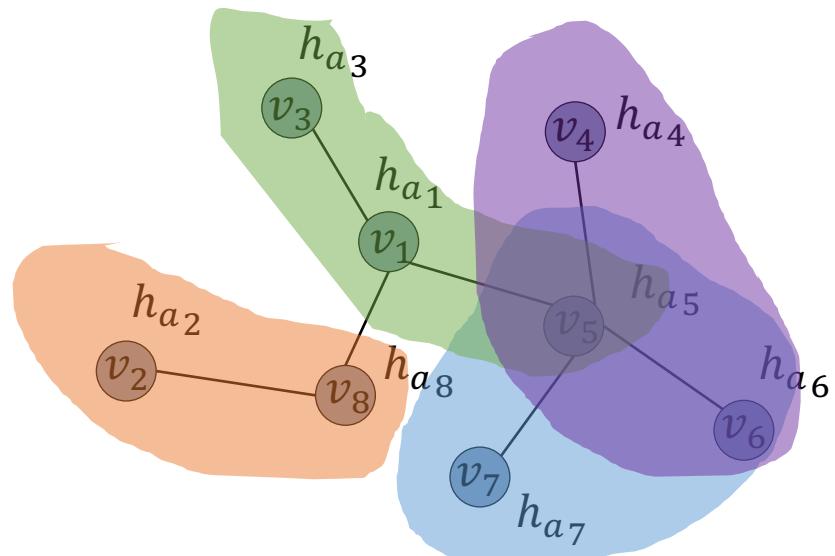
$$\mathbf{H}_f \in \mathbb{R}^{n \times d_{new}}$$

Generate  $A_p$

$$\mathbf{A}_p = \mathbf{H}_a^T \mathbf{A} \mathbf{H}_a$$

# DiffPool

Downsample by clustering the nodes using GNN



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$

$$\downarrow$$

$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

Generated soft-assign matrix

$$\mathbf{H}_a \in \mathbb{R}^{n \times n_p}$$

Generated new features

$$\mathbf{H}_f \in \mathbb{R}^{n \times d_{new}}$$

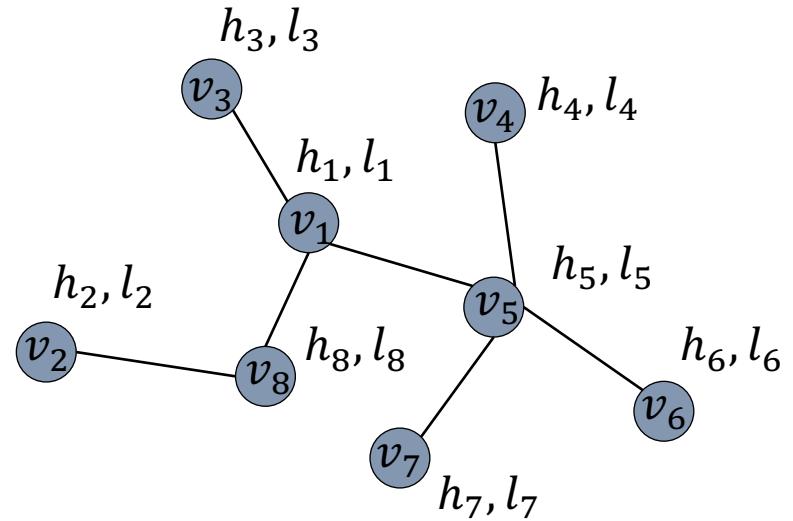
Generate  $A_p$

$$\mathbf{A}_p = \mathbf{H}_a^T \mathbf{A} \mathbf{H}_a$$

Generate  $H_p$

$$\mathbf{H}_p = \mathbf{H}_a^T \mathbf{H}_f$$

# Eigenpooling

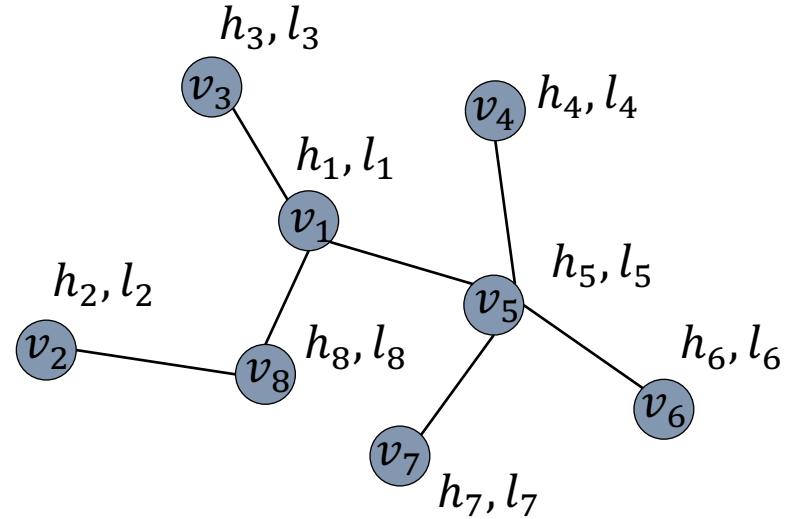


$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$

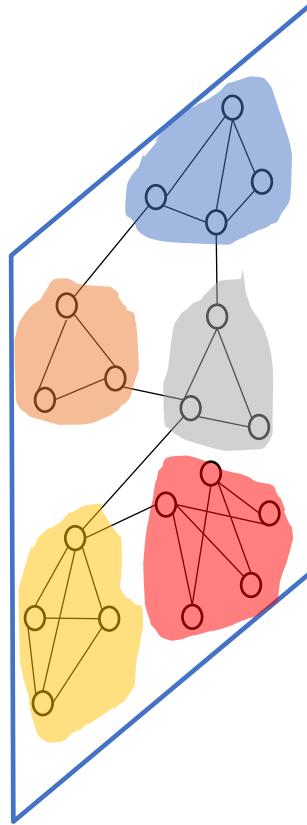
# Eigenpooling



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$

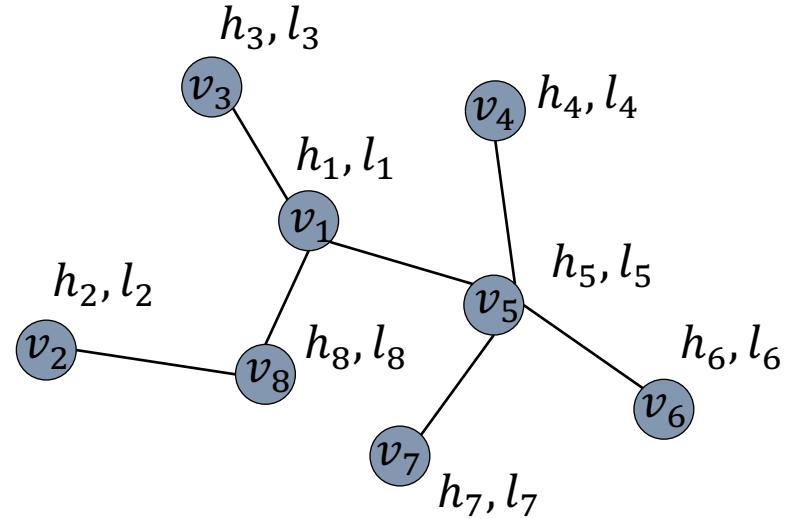


$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$



Learn  $\mathbf{A}_p$  using  
clustering methods

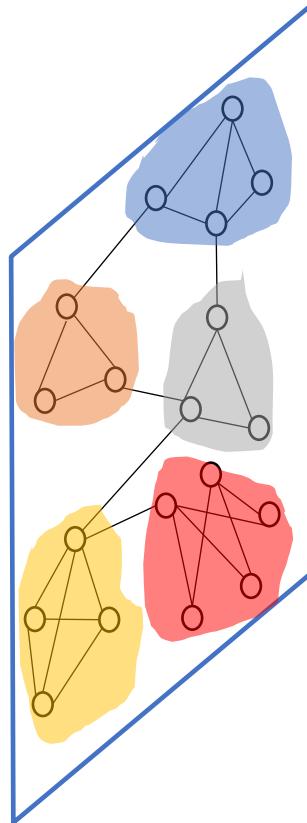
# Eigenpooling



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



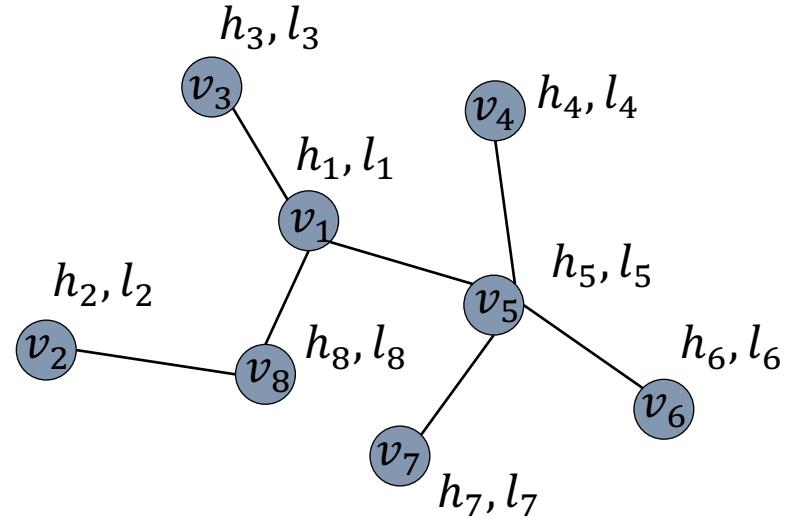
$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$



Learn  $A_p$  using  
clustering methods

Focus on learning  
better  $H_p$

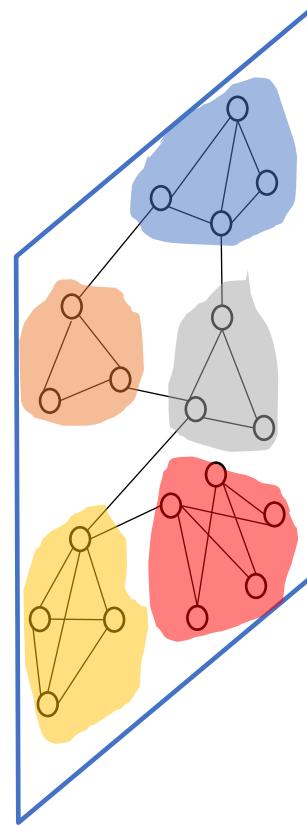
# Eigenpooling



$$\mathbf{A} \in \{0, 1\}^{n \times n}, \mathbf{H} \in \mathbb{R}^{n \times d}$$



$$\mathbf{A}_p \in \{0, 1\}^{n_p \times n_p}, \mathbf{H}_p \in \mathbb{R}^{n_p \times d_{new}}, n_p < n$$



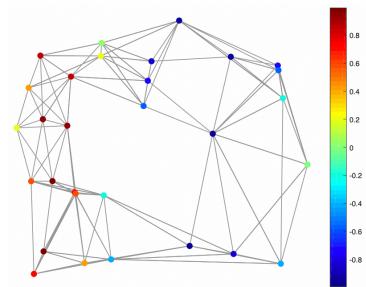
Learn  $A_p$  using  
clustering methods

Focus on learning  
better  $H_p$

Capture both feature  
and graph structure

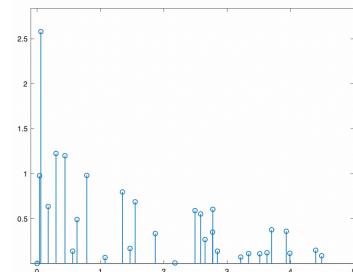
# Going Back to Graph Spectral Theory

Recall:



Spatial domain:  $f$

$$\hat{f} = U^T f \quad \xrightarrow{\hspace{10em}} \text{Decompose signal } f$$



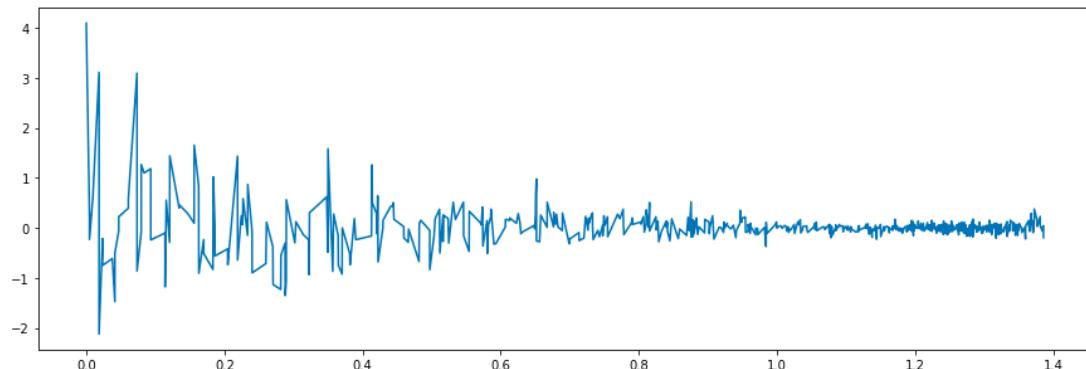
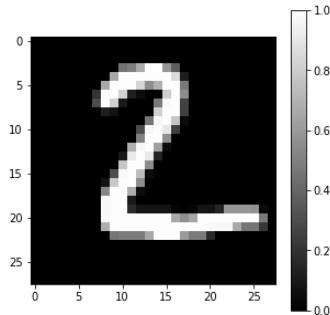
Spectral domain:  $\hat{f}$

$$f = \hat{f}_0 u_0 + \hat{f}_1 u_1 + \dots + \hat{f}_{N-1} u_{N-1}$$



# Going Back to Graph Spectral Theory

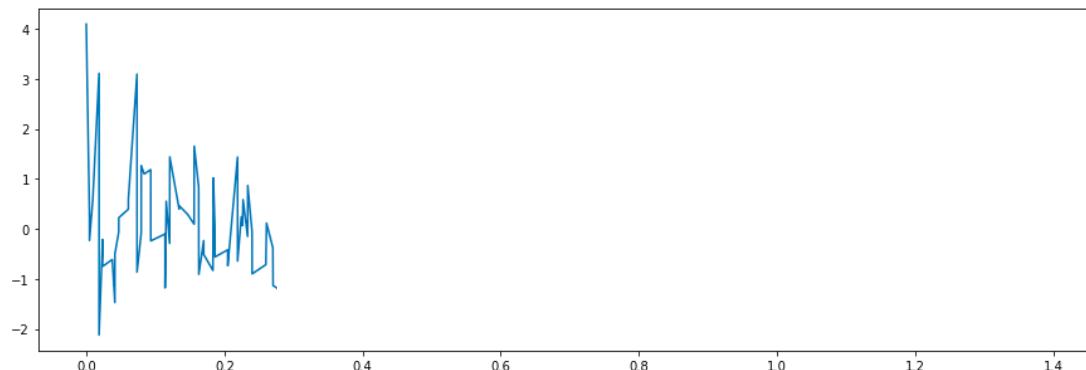
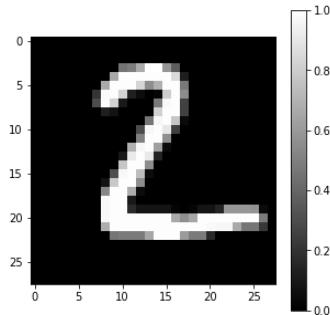
Do we need all the coefficients to reconstruct a “good” signal?





# Going Back to Graph Spectral Theory

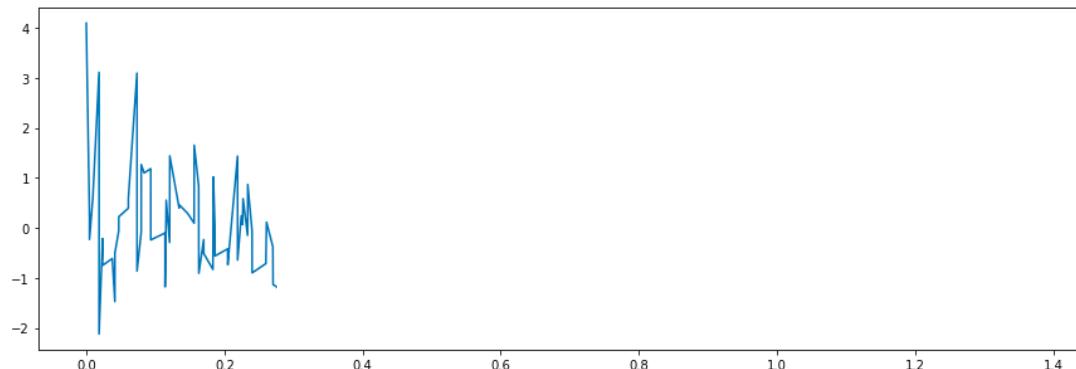
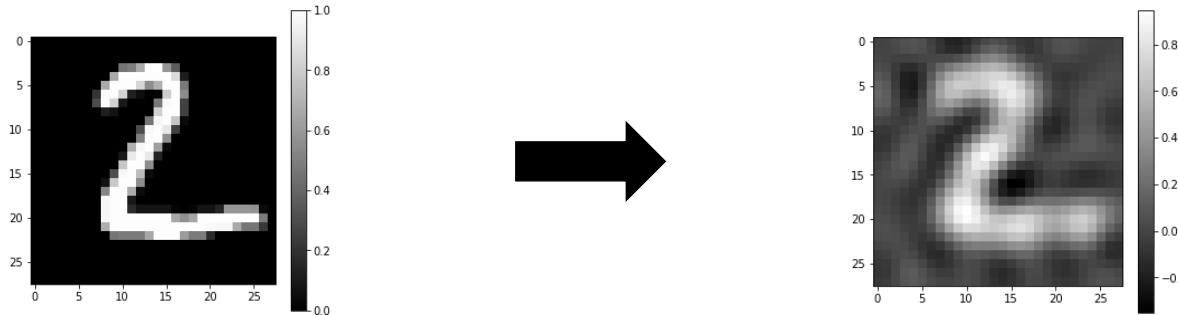
Do we need all the coefficients to reconstruct a “good” signal?





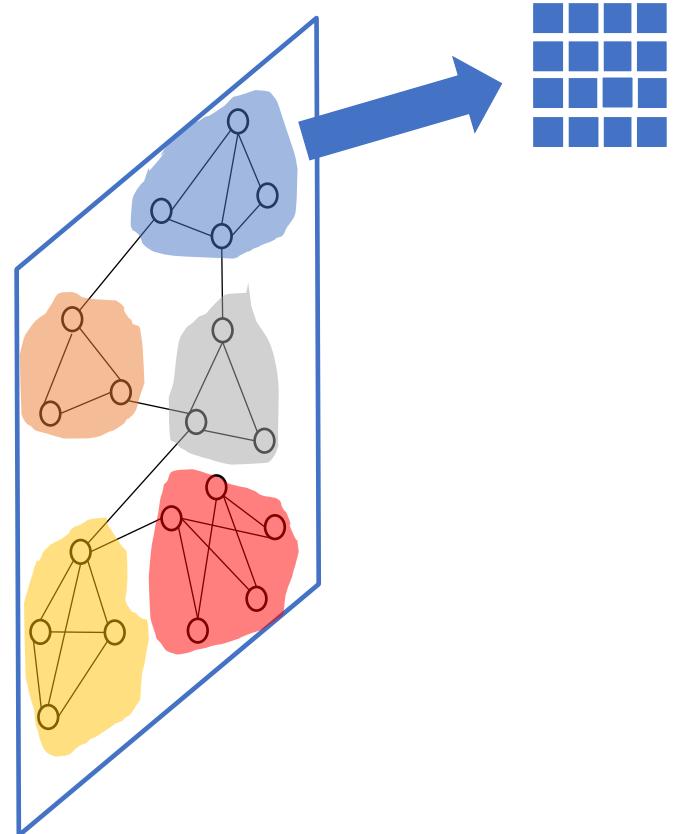
# Going Back to Graph Spectral Theory

Do we need all the coefficients to reconstruct a “good” signal?





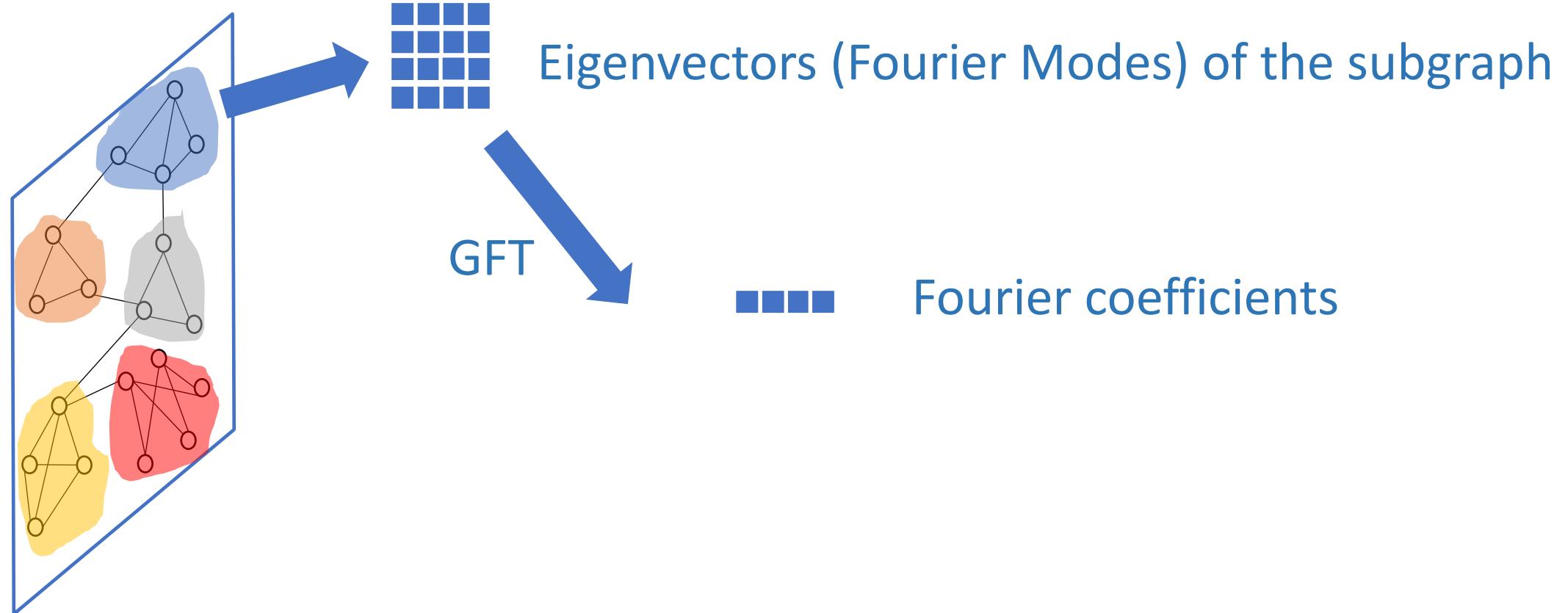
# Eigenpooling: Truncated Fourier Coefficients



Eigenvectors (Fourier Modes) of the subgraph

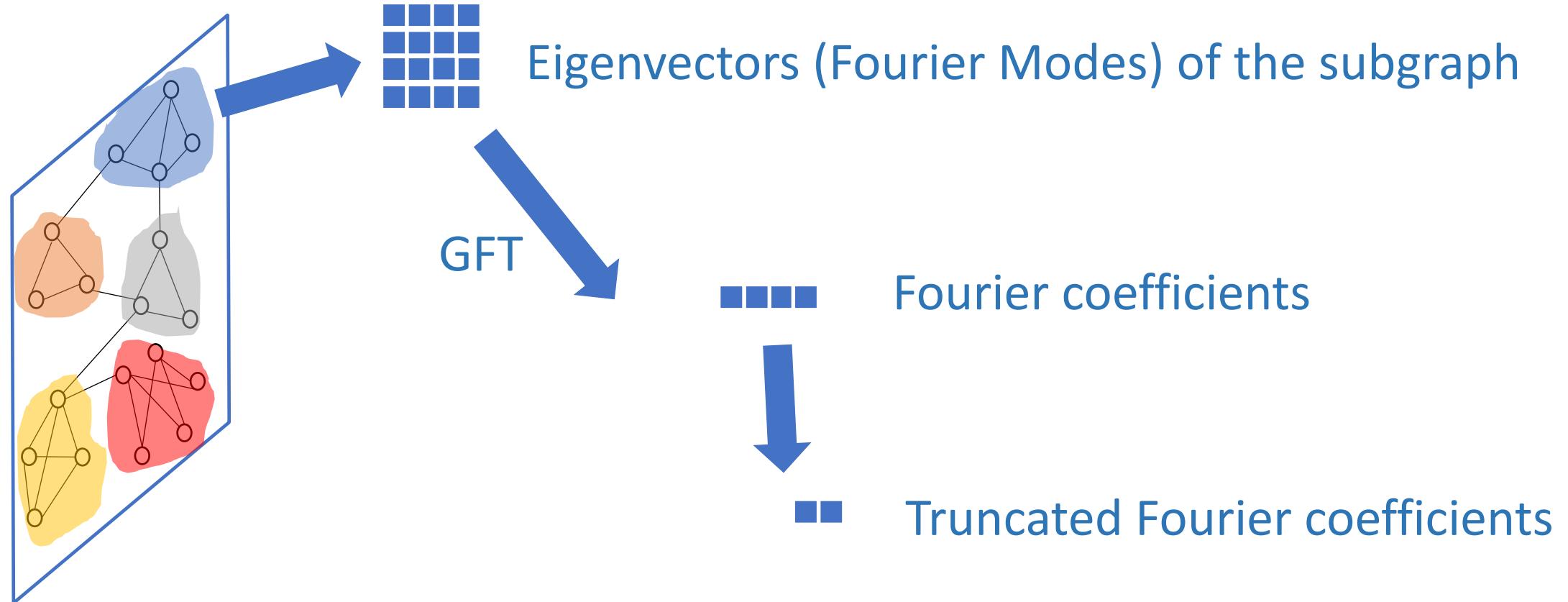


# Eigenpooling: Truncated Fourier Coefficients



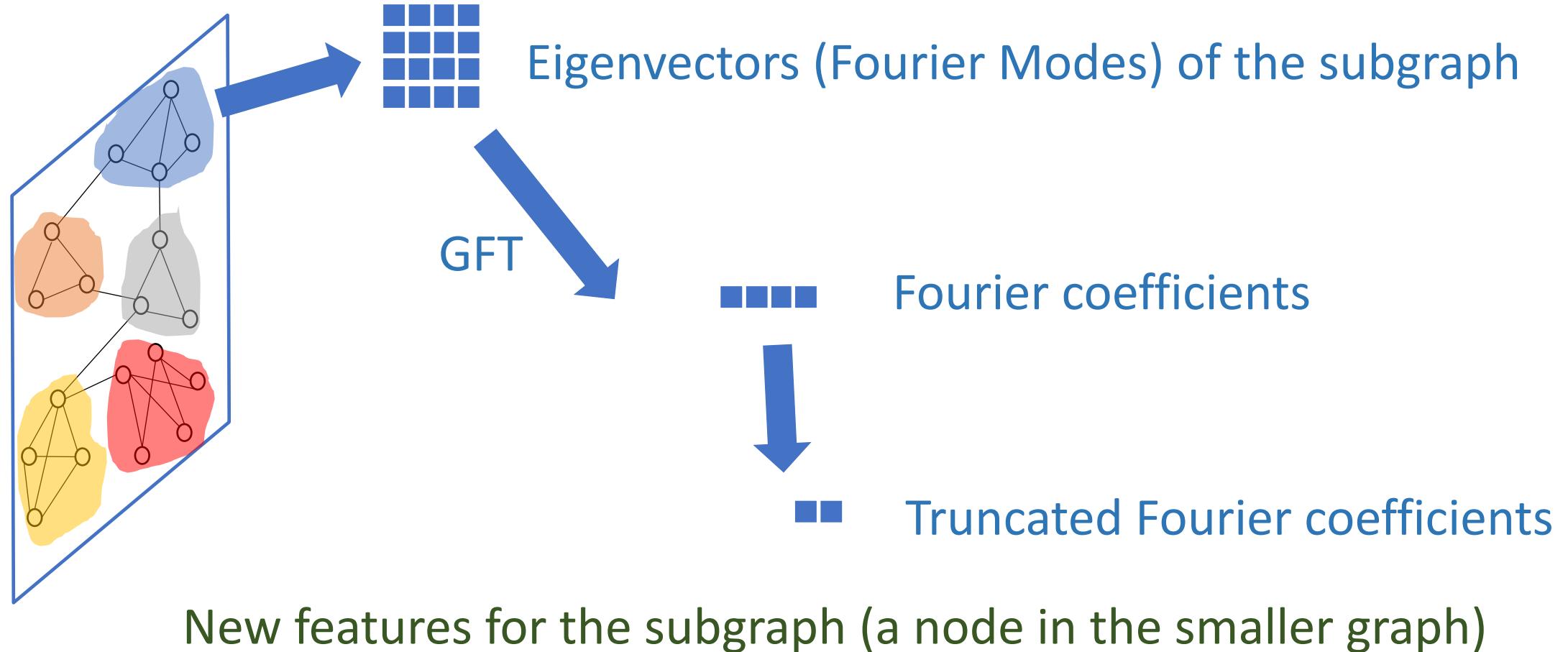


# Eigenpooling: Truncated Fourier Coefficients



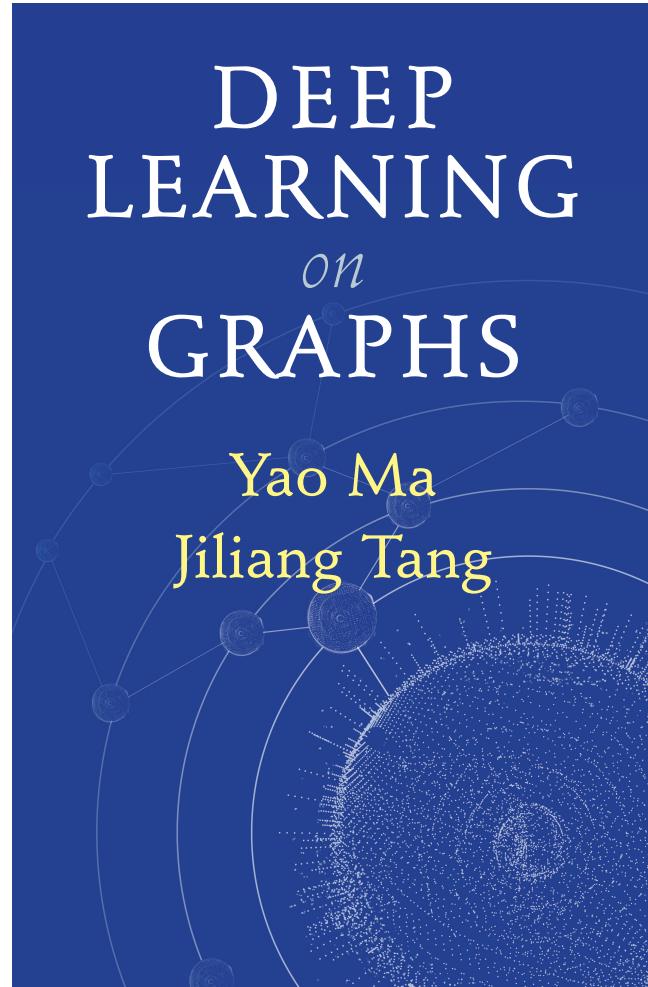


# Eigenpooling: Truncated Fourier Coefficients





# Book: Deep Learning on Graphs



[https://cse.msu.edu/~mayao4/dlg\\_book/](https://cse.msu.edu/~mayao4/dlg_book/)

# Acknowledgements



Thanks for this great opportunity and our funding support from MSU, AERA, NSF, ARO, DARPA and industrial collaborators.

**DSE Lab@MSU**

