Graph Neural Network

Zijian Chen

zijianc@bu.edu

Contents

- Part I: Introduction

  graphs; publication trend; why and how do we study graphs

- Part II: Basic Principles

  difference between graphs and images; message passing formulation; pitfalls and work-arounds

- Part III: Architectures and Training

  design of frameworks and training schemes for different tasks

- Part IV: Non-message-passing GNNs

  spectral GNNs; graph transformers

Graph Neural Network

Part I: Introduction

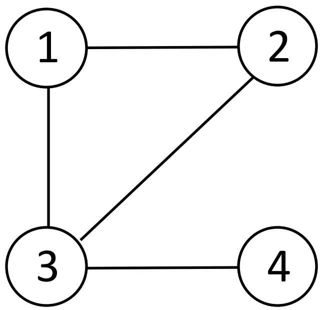
What is a Graph

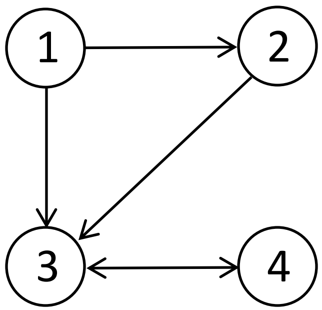
Rigorously, a graph is an ordered pair

$$G = (V, E)$$

node $\{1, 2, \dots, n\}$

edge $\{(i, j) : i, j \in V\}$





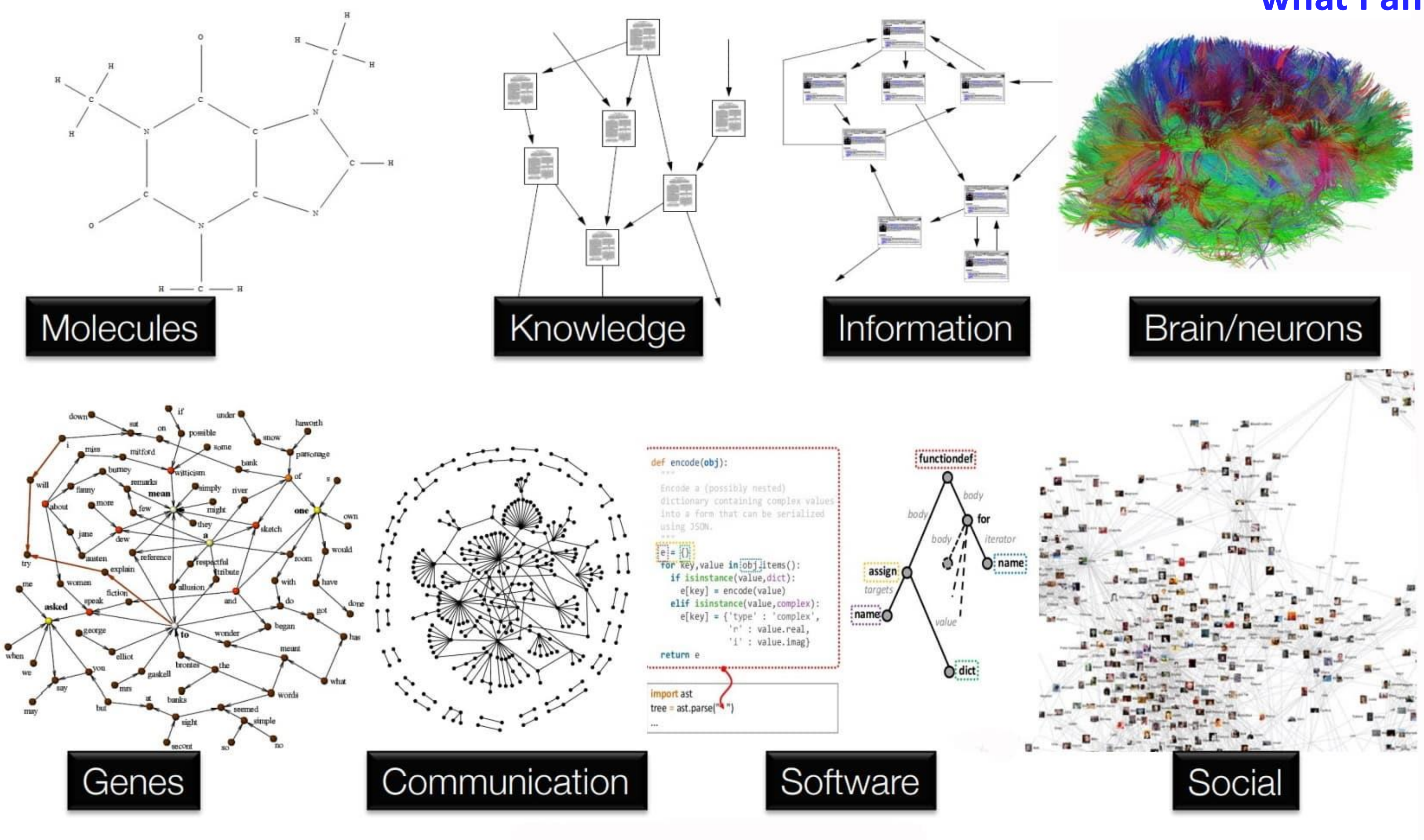
We use adjacency matrix and the Laplacian to algebraically represent the structure.

$$A = \begin{array}{c c} \begin{matrix} \;1 & \;\;2 & \;\;3 & \;\;4 \end{matrix} & \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \end{array}$$

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

Data as Graphs

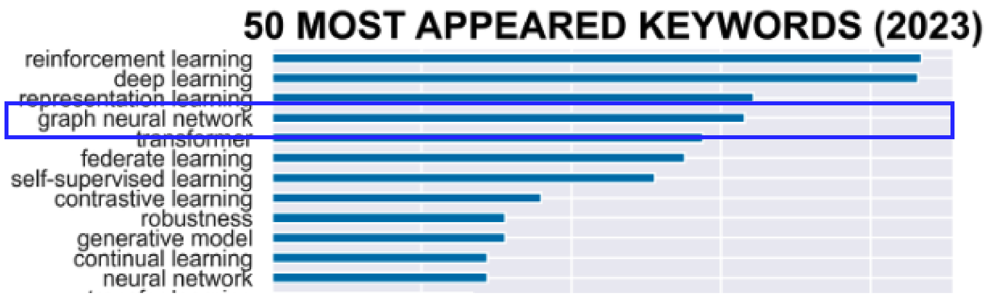
what I am doing

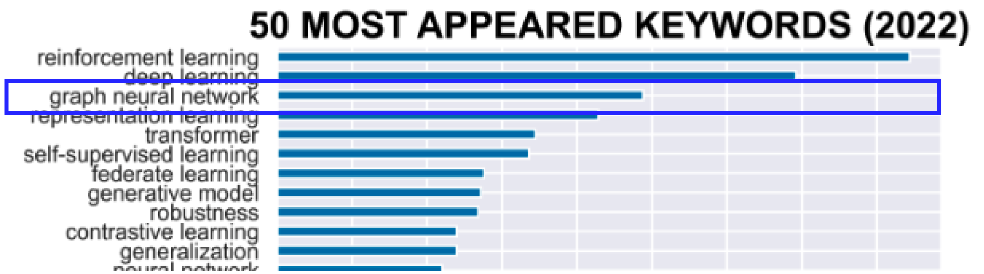


ICLR Publication Trend

Top 50 keywords *2024*

Keyword	Count
Large Language Models	318
Reinforcement Learning	201
Graph Neural Networks	123
Diffusion Models	112
Deep Learning	110
...	
Foundation Models	20
Learning Theory	19
Online Learning	19
Instruction Tuning	19
Variational Inference	19





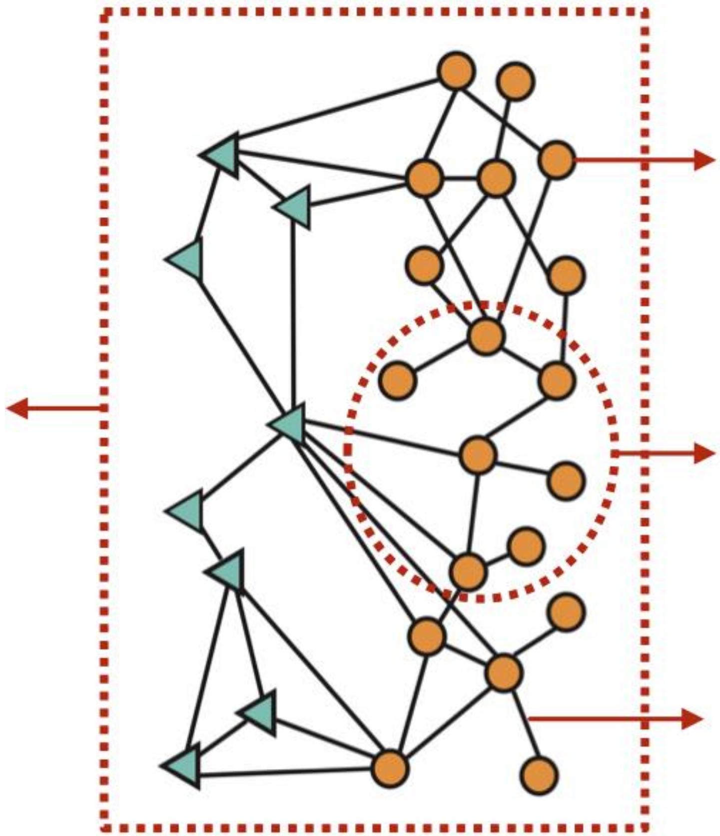
Different Types of Tasks

graph-level regression/
classification

Patient status

graph generation

Drug Discovery



node-level tasks
(mostly classification)

protein folding (predicting node 3D coordinates)

structure identification
subgraph property prediction

Google Map ETA

link prediction/
edge feature engineering

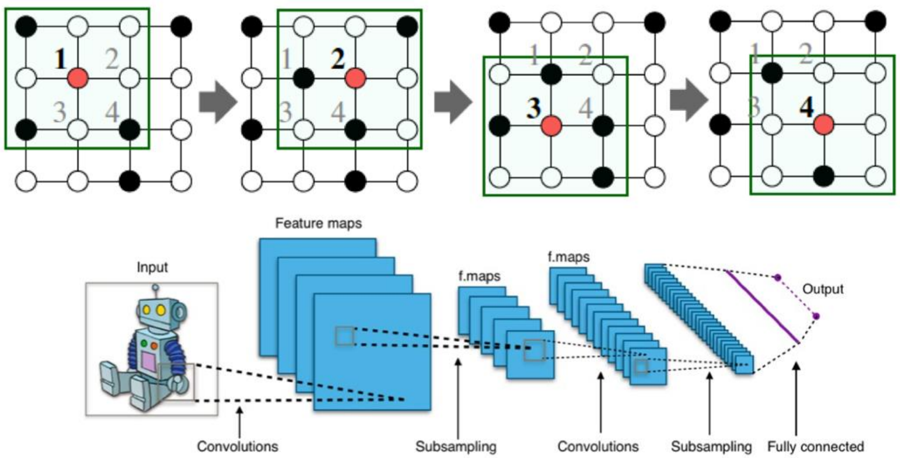
recommender systems
drug interaction

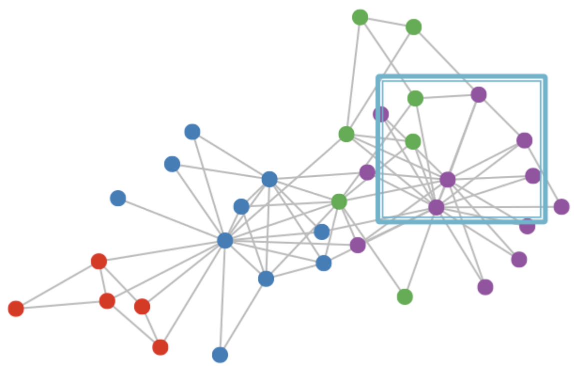
Graph Neural Network

Part II: Basic Principles

How are graphs different 1/2

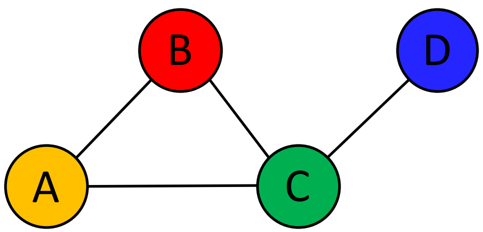
Observation 1: graphs do not have a fixed notion of locality or sliding window.



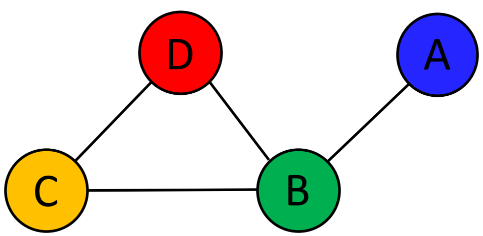


How are graphs different 2/2

Observation 2: graphs do not have a canonical node ordering.



$$A_1 = \begin{array}{cc} & \begin{array}{cccc} A & B & C & D \end{array} \\ & \left[\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right] \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array} \qquad X_1 = \begin{bmatrix} 0.11 & 0.14 \\ 0.22 & 0.23 \\ 0.33 & 0.35 \\ 0.44 & 0.48 \end{bmatrix}$$



$$A_2 = \begin{array}{cc} & \begin{array}{cccc} A & B & C & D \end{array} \\ & \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array} \qquad X_2 = \begin{bmatrix} 0.44 & 0.48 \\ 0.11 & 0.14 \\ 0.33 & 0.35 \\ 0.22 & 0.23 \end{bmatrix}$$

How do we want the output to be?

Invariance and Equivariance

Observation 2: graphs do not have a canonical node ordering.

Invariance: permuting the input, the output stays the same.

$$f(A_1, X_1) = f(A_2, X_2) \quad \text{or,} \quad f(A, X) = f(PAP^\top, PX)$$

Equivariance: permuting the input, the output also gets permuted accordingly.

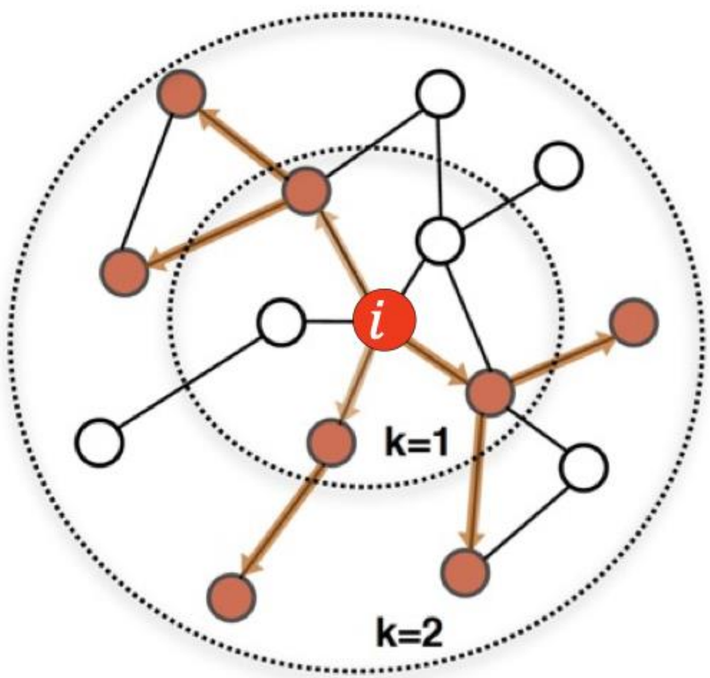
$$f(A_1, X_1) = Pf(A_2, X_2) \quad \text{or,} \quad Pf(A, X) = f(PAP^\top, PX)$$

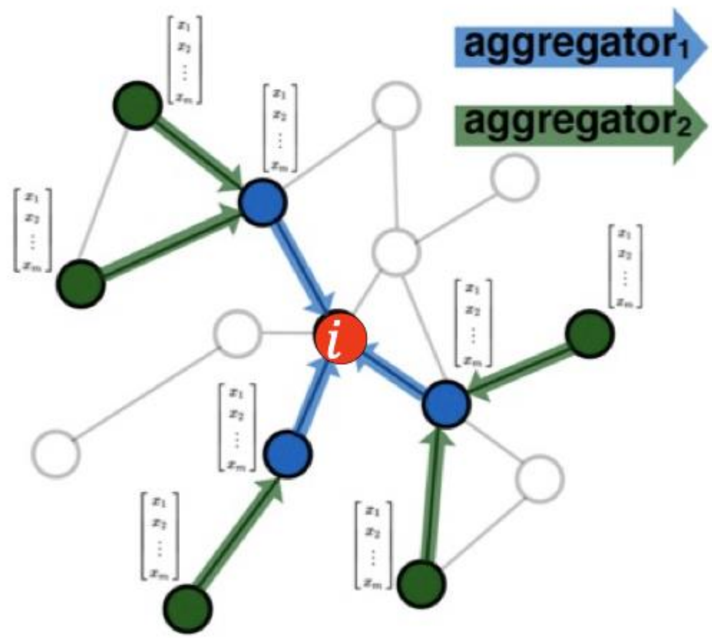
Traditional NN architectures, e.g., MLPs, fail for graphs, as switching the order of input will lead to different outputs.

Invariance/Equivariance can be achieved by passing and aggregating information from neighbors. This is the core of GNN.

Constructing a GNN

In each layer, a GNN aggregates neighboring node features.





Message Passing

Mathematically, we can write the message passing rule as

$$\mathbf{x}_i' = \gamma_{\Theta}\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}(i)} \phi_{\mathbf{\Theta}}\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{j,i}\right)\right)$$

Key ingredients:

- **Message**: each node computes a message.
- **Aggregation:** aggregate message from neighbors.
- **Update:** determine how to apply the aggregated message to target node.

Which part do you think is the hardest to implement?

Message Passing

Let's see a concrete example: (one of your homework questions!)

$$\mathbf{x}_i^{(l+1)} = \mathrm{relu}\left(W_i^{(l)} \mathbf{x}_i^{(l)} + \sum_{j \in \mathcal{N}(i)} e_{ij} W_j^{(l)} \mathbf{x}_j^{(l)} \right)$$

- **Message:** $W_i^{(l)} \mathbf{x}_i, \quad e_{ij} W_j^{(l)} \mathbf{x}_j^{(l)}$
- **Aggregation:** $\sum_{j \in \mathcal{N}(i)}$
- **Update:** $\mathrm{relu}(\cdots + \cdots)$

Is this formulation invariant or equivariant?

More Examples…

Almost all current cutting-edge GNN designs are MPNNs:

- vanilla GCN (2017)
- GAT
- GraphSAGE
- GIN
- PNA
- EGNN
- …

We will discuss non-message-passing designs later.

Implementation

🏠 / torch_geometric.nn / conv.MessagePassing

conv.MessagePassing

class **MessagePassing** (**aggr**: Optional[Union[str, List[str], Aggregation]] = **'sum'**, *, **aggr_kwargs**: Optional[Dict[str, Any]] = **None**, **flow**: str = **'source_to_target'**, **node_dim**: int = **-2**, **decomposed_layers**: int = **1**) [source]

propagate (**edge_index**: Union[Tensor, SparseTensor], **size**: Optional[Tuple[int, int]] = **None**, ****kwargs**: *Any*) → Tensor [source]

The initial call to start propagating messages.

message (**x_j**: Tensor) → Tensor [source]

Constructs messages from node j to node i in analogy to ϕ_{Θ} for each edge in `edge_index`. This function can take any argument as input which was initially passed to `propagate()`. Furthermore, tensors passed to `propagate()` can be mapped to the respective nodes i and j by appending `_i` or `_j` to the variable name, *.e.g.* `x_i` and `x_j`.

aggregate (**inputs**: Tensor, **index**: Tensor, **ptr**: Optional[Tensor] = **None**, **dim_size**: Optional[int] = **None**) → Tensor [source]

Aggregates messages from neighbors as $\bigoplus_{j \in \mathcal{N}(i)}$.

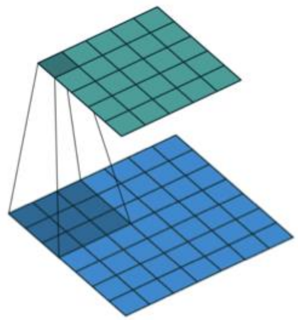
Takes in the output of message computation as first argument and any argument which was initially passed to `propagate()`.

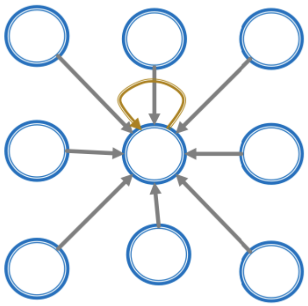
You will need to implement a light-weight version of this class in HW5

(official class: ~1000 lines of code)

CNN as a special case of GNN

Consider a CNN with 3x3 filter:





$$\mathbf{x}_i' = \sigma\Big(\sum_{j \in \mathcal{N}_{3\times 3}} W_j \mathbf{x}_j\Big)$$

$$\mathbf{x}_i' = \sigma\Big(\sum_{j \in \mathcal{N}(i)} W_j \mathbf{x}_j\Big)$$

You don't necessarily need weight sharing
& You can pick any neighbor you want

A Closer Look: Deep layers

Observation: Layer-k update gets info from nodes up to k-hops away.

Consider a simplified version of the general formulation:

$$\phi_{\Theta}\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{j,i}\right) = W\mathbf{x}_j \qquad \gamma_{\Theta}(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}(i)} (\cdot)) = \sigma((1-\alpha)U\mathbf{x}_i + \alpha \sum_{j \in \mathcal{N}(i)} W\mathbf{x}_j)$$

Then we will have

$$X^{(k+1)} = \sigma((1-\alpha)X^{(k)}U + \alpha A X^{(k)} W)$$

As $k \rightarrow \infty$, $X^{(k+1)} \rightarrow X^{(k)}$ This is called **over-smoothing.**

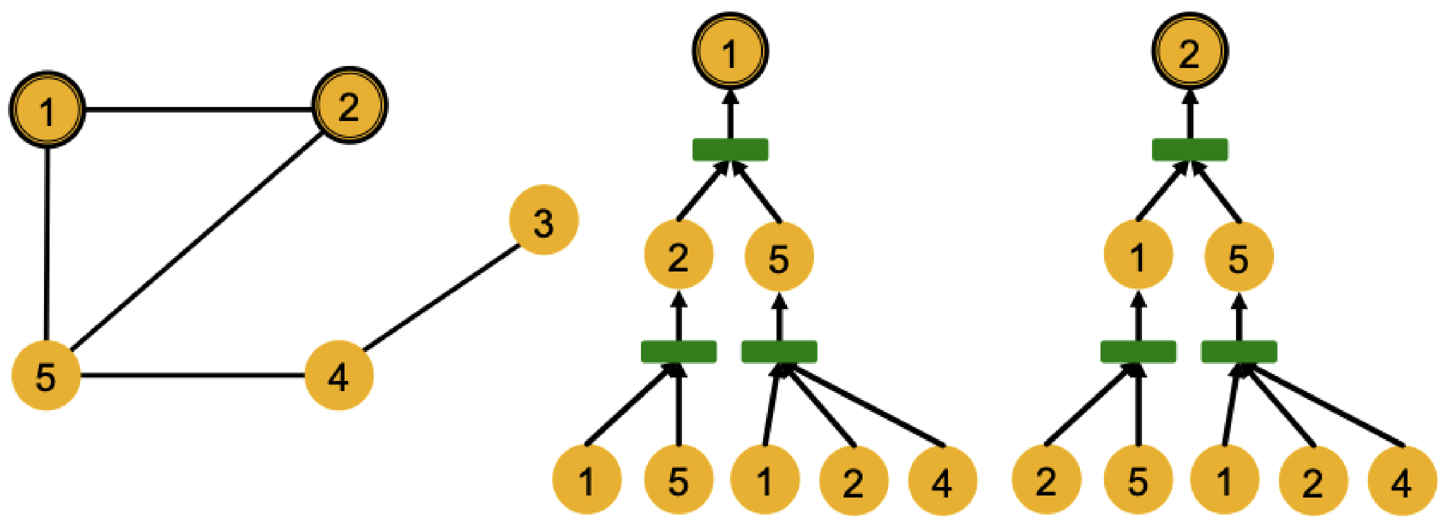
Are there specific choices that can avoid over-smoothing?

A Closer Look: Expressivity

A classical expressivity test is **Graph Isomorphism**.

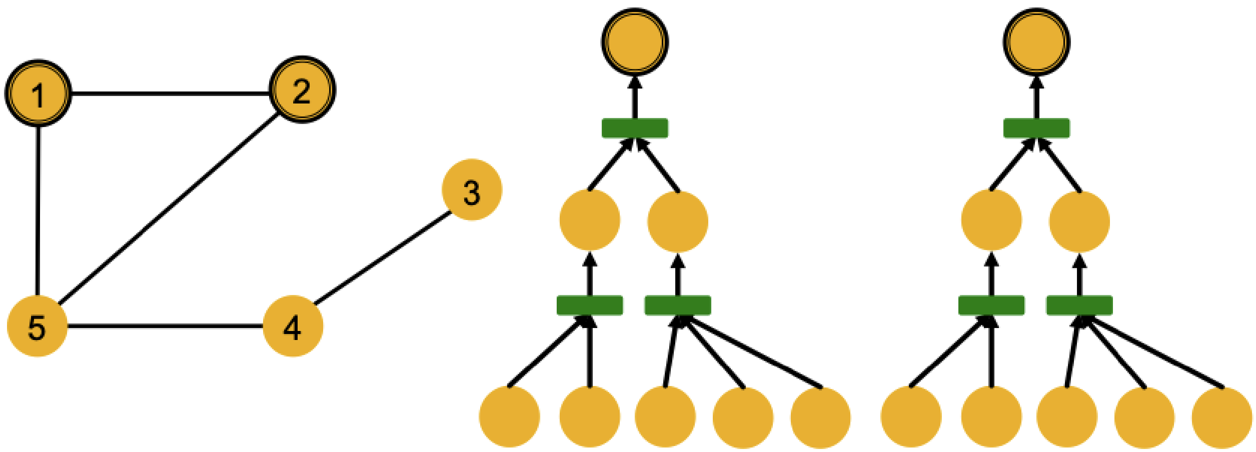
A simpler problem: *given a pair of nodes with different neighborhood structure, is there a GNN that can always tell them apart?*

Consider the extreme case where all nodes have the same feature. Computational graph for Node 1 and Node 2:



A Closer Look: Expressivity

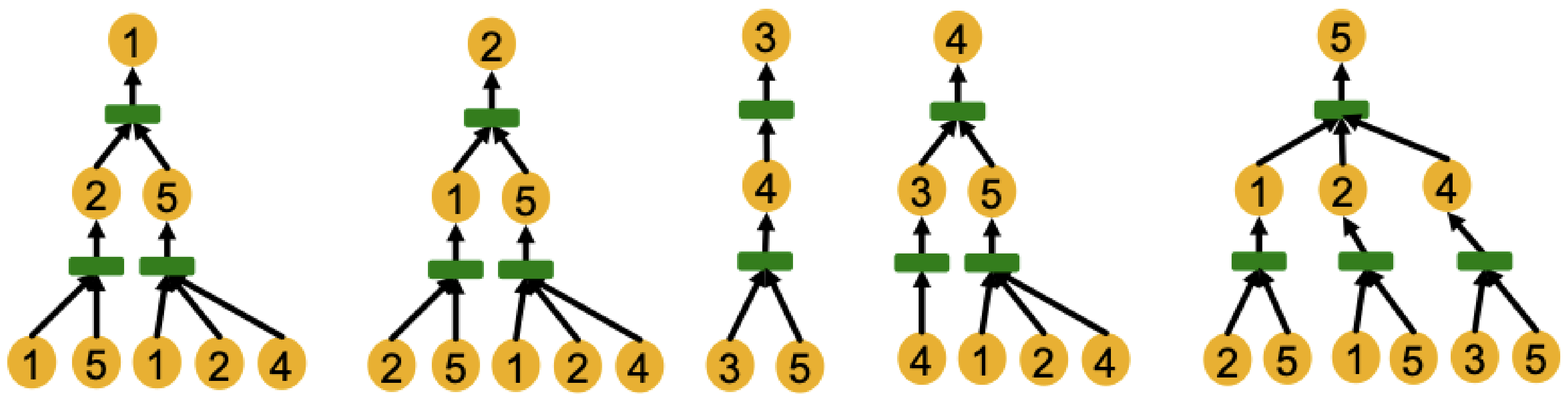
But GNN only see the node features but not IDs



So, the updated features of node 1 and node 2 are still identical.

A Closer Look: Expressivity

Computational graphs for all nodes:



should produce
identical features

should produce
different features

A Closer Look: Expressivity

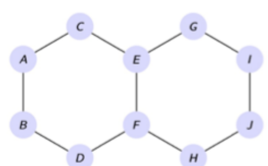
Conclusion: The expressive power of GNNs depend on the expressive power of the aggregation function. Injective function leads to the most expressive GNN.

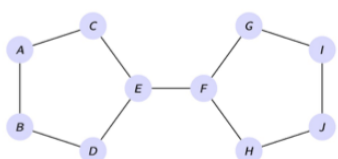
More in-depth Conclusion:

- MP-GNNs are at most as powerful as the *WL test* in distinguishing graph structures.
- One such GNN ("Graph Isomorphism Network", ICLR 2019):

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right).$$

- Examples that WL test (or equivalently, GIN) fails:
  - Certain special structures
  - Counting cycles in the graph





A Closer Look: Expressivity

Workarounds:

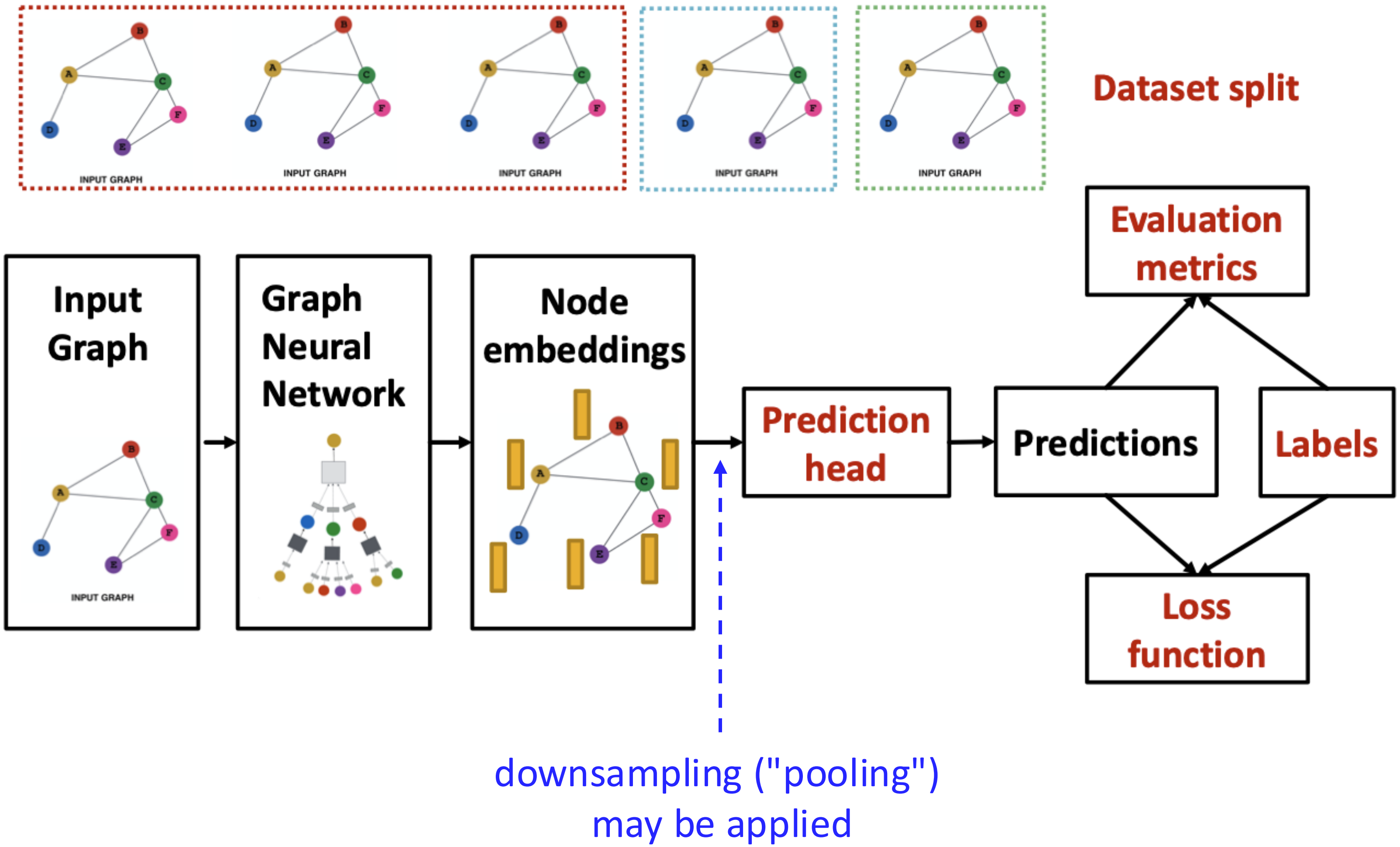
- Higher-order WL tests: e.g.,
  - 2-WL considers pairs of nodes – "hypergraphs"
- Positional/structural encodings, e.g.,
  - encode each node with a different ID
  - cycle counts as augmented node features
  - assigning anchor nodes and compute relative distance …
- Global attention/transformers
- …

MP-GNNs are not perfect, but in most cases, they are more than sufficient (in terms of performance).

Graph Neural Network

Part III: Architectures and Training

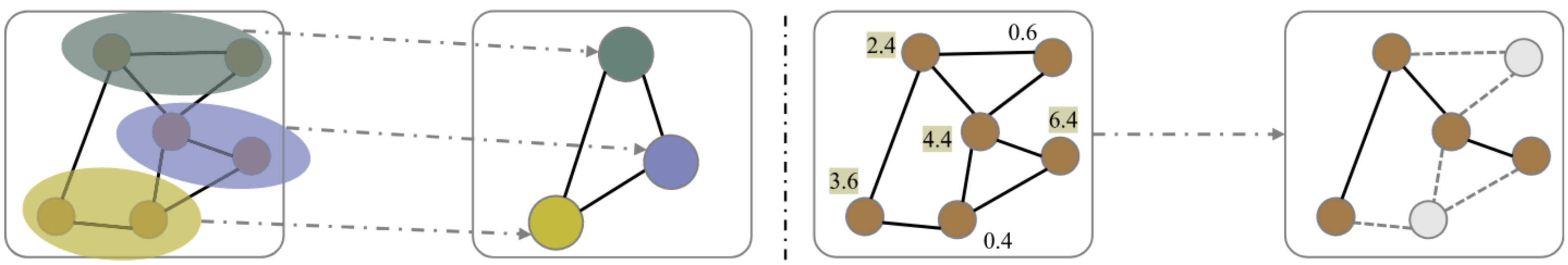
A Full GNN Framework



Graph Pooling

Goal: downsample the graph to obtain representations at a smaller scale.

Two typical forms:



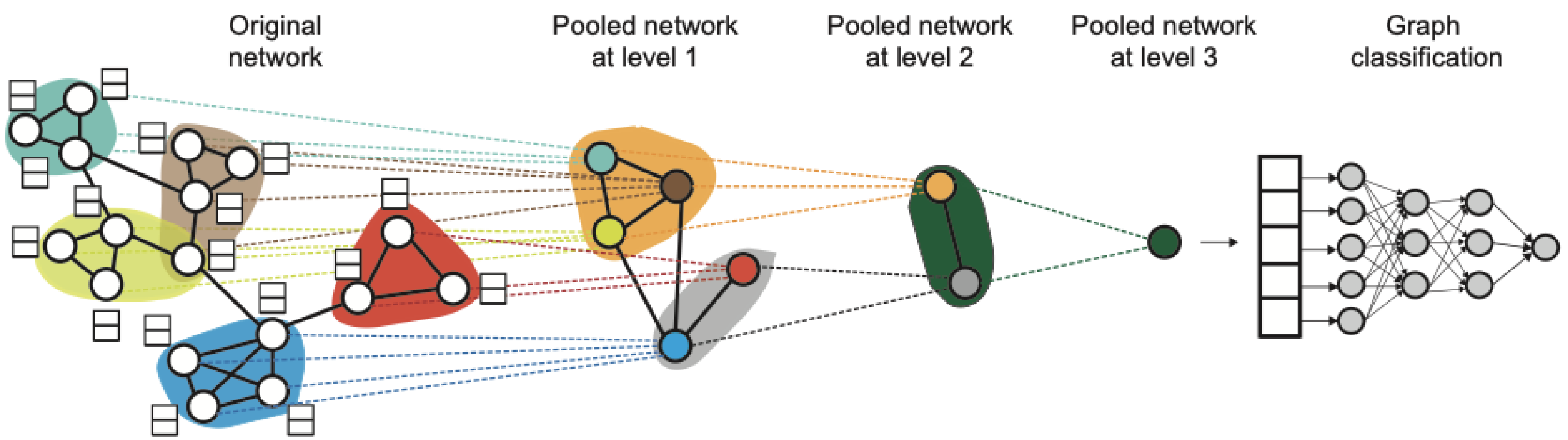
(a) Cluster-based pooling

(b) Selection-based pooling

DiffPool (NIPS 2018)
MinCutPool (ICML 2020)

Top-K Pool (ICML 2019)
SAGPool (ICML 2019)

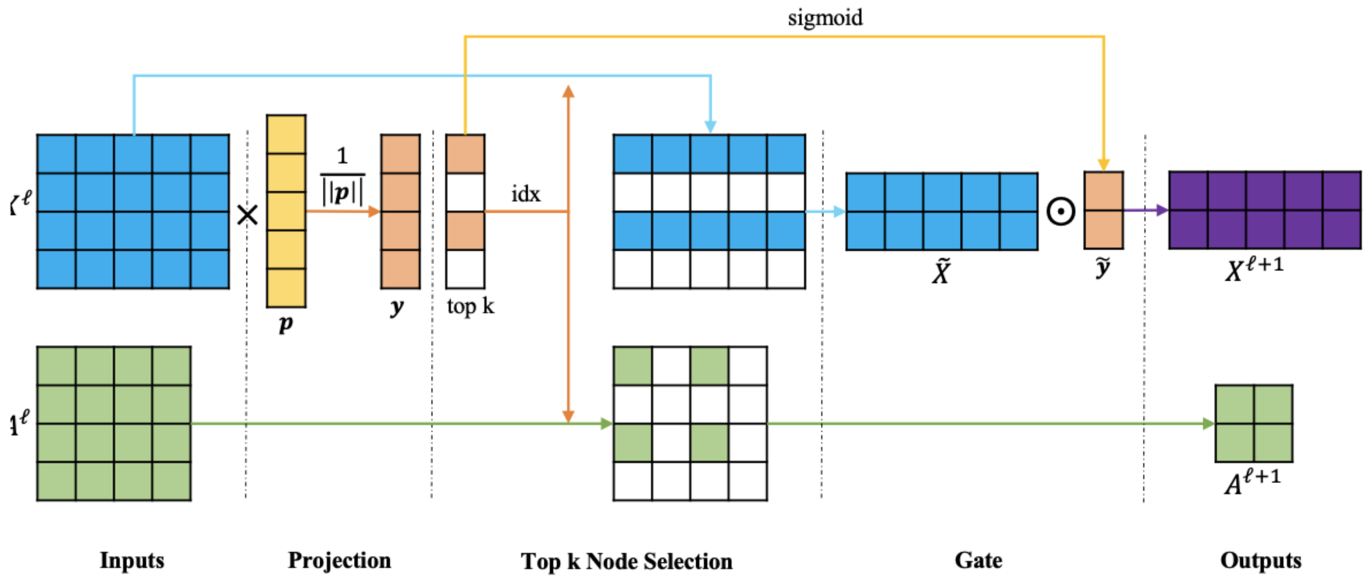
Graph Pooling: cluster-based



$$X^{(l+1)} = S^{(l)^T} Z^{(l)} \in \mathbb{R}^{n_{l+1} \times d},$$
$$A^{(l+1)} = S^{(l)^T} A^{(l)} S^{(l)} \in \mathbb{R}^{n_{l+1} \times n_{l+1}}.$$

$$L_{\mathrm{LP}} = \left\| A^{(l)}, S^{(l)} S^{(l)^T} \right\|_F$$
$$L_{\mathrm{E}} = \frac{1}{n} \sum_{i=1}^{n} H\left(S_i\right)$$

Graph Pooling: selection-based



$$\boldsymbol{y} = X^\ell \boldsymbol{p}^\ell / \|\boldsymbol{p}^\ell\|, \qquad \in \mathbb{R}^N$$

$$\text{idx} = \text{rank}(\boldsymbol{y}, k), \qquad \in \mathbb{R}^k$$

$$\tilde{\boldsymbol{y}} = \text{sigmoid}(\boldsymbol{y}(\text{idx})), \qquad \in \mathbb{R}^k$$

$$\tilde{X}^\ell = X^\ell(\text{idx}, :), \qquad \in \mathbb{R}^{k \times C}$$

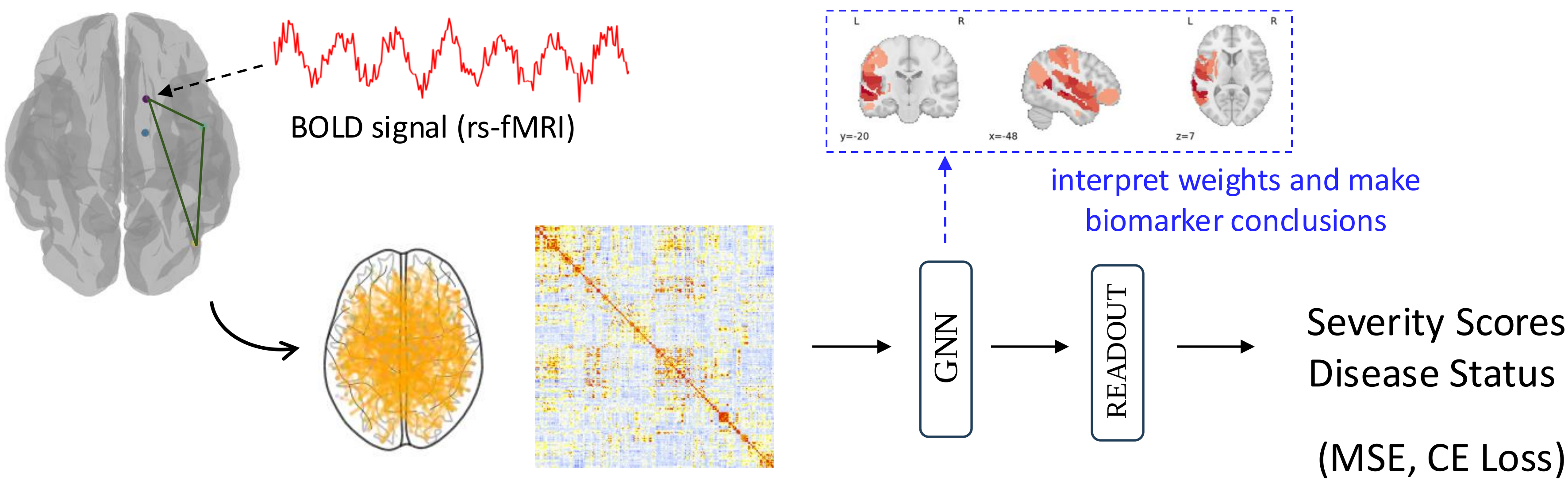
$$A^{\ell+1} = A^\ell(\text{idx}, \text{idx}), \qquad \in \mathbb{R}^{k \times k}$$

$$X^{\ell+1} = \tilde{X}^\ell \odot (\tilde{\boldsymbol{y}} \mathbf{1}_C^T), \qquad \in \mathbb{R}^{k \times C},$$

Supervised Learning

Directly train the model for a specific task with ground truth label given

For example, in neuroimaging, (mostly graph-level tasks)



Unsupervised Learning

The most common idea: similar nodes should have similar embeddings.

e.g., $$\mathcal{L} = \sum_{z_u, z_v} \mathrm{CE}(y_{uv}, z_u^\top z_v)$$

and it boils down to defining what kind of "similarity" you want.

Other design principles:

- maximizing information/entropy
- obeying flow constraints, such as curl-free, energy-preserving
- reflecting causal relationship
- …

Graph Neural Network

Part IV: Non-MP GNN

Spectral GNNs

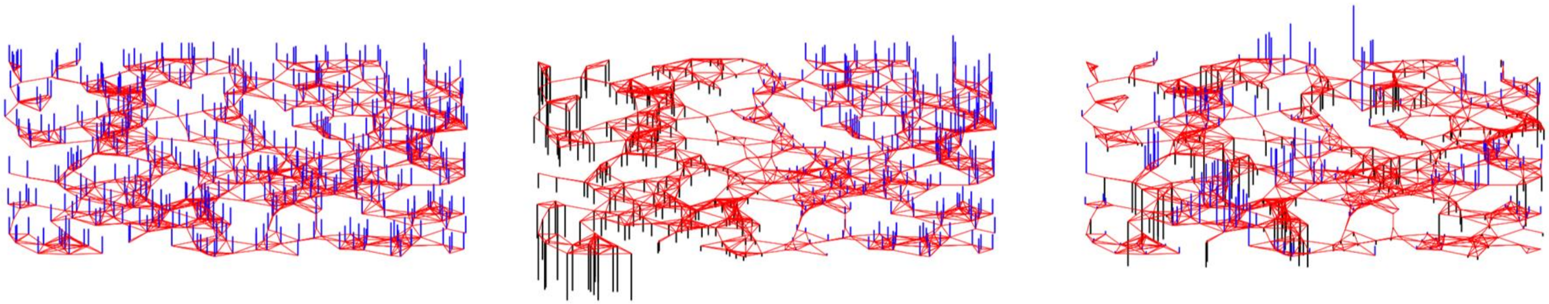
Spectral Domain of Graphs

Overview: in MPNNs, we focus on "the neighborhood of a node"

$$N_\delta(j) = \{i \in \Omega : W_{ij} > \delta\}$$

This is usually inefficient and cannot carry additional info.

New Idea: we move all operations to the spectral domain.



$\mathbf{u}_0$ $\mathbf{u}_1$ $\mathbf{u}_{50}$

Graph Fourier Transform

	Euclidean Space	Graphs
Fourier basis	eigen-functions of the Laplacian	eigen-functions of the graph Laplacian
Fourier transform	$\hat{f}(\omega) = \int f(t)\exp(-i\omega t)\mathrm{d}t$	$\hat{f}(\lambda_l) := \sum_{i=1}^{N} f(i) u_l(i)$
Convolution	$\mathcal{F}^{-1}\{\hat{f}(\omega)\hat{h}(\omega)\}$	$U\Big((U^\top f) \odot (U^\top h)\Big)$

$$f \xrightarrow{\text{FT}} U^\top f \underbrace{\xrightarrow{\text{filtering}} \hat{h}_\theta U^\top f}_{\text{aggregation happens in the frequency domain}} \xrightarrow{\text{IFT}} U\hat{h}_\theta U^\top f$$

Kernel Design Example 1/3

Pick $\hat{h}_\theta(\lambda_i) = \theta_i$, so the filter becomes

$$\hat{h}_\theta = \begin{bmatrix} \theta_1 & & \\ & \ddots & \\ & & \theta_N \end{bmatrix}$$

Performance on MNIST dataset:

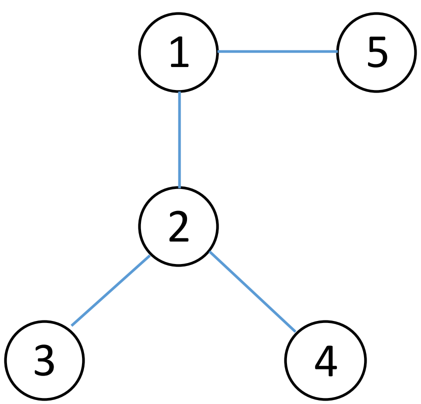
	method	Parameters	Error
	Nearest Neighbors	N/A	4.11
	400-FC800-FC50-10	$3.6 \cdot 10^5$	1.8
spatial	400-LRF1600-MP800-10	$7.2 \cdot 10^4$	1.8
spatial	400-LRF3200-MP800-LRF800-MP400-10	$1.6 \cdot 10^5$	**1.3**
spectral	400-SP1600-10 ($d_1 = 300$, $q = n$)	$3.2 \cdot 10^3$	2.6
spectral	400-SP1600-10 ($d_1 = 300$, $q = 32$)	$1.6 \cdot 10^3$	2.3
spectral	400-SP4800-10 ($d_1 = 300$, $q = 20$)	$5 \cdot 10^3$	1.8

	method	Parameters	Error
	Nearest Neighbors	N/A	19
	4096-FC2048-FC512-9	10^7	**5.6**
spatial	4096-LRF4620-MP2000-FC300-9	$8 \cdot 10^5$	**6**
spatial	4096-LRF4620-MP2000-LRF500-MP250-9	$2 \cdot 10^5$	6.5
spectral	4096-SP32K-MP3000-FC300-9 ($d_1 = 2048$, $q = n$)	$9 \cdot 10^5$	7
spectral	4096-SP32K-MP3000-FC300-9 ($d_1 = 2048$, $q = 64$)	$9 \cdot 10^5$	**6**

Kernel Design Example 1/3

Problems:

- diagonalizing the Laplacian takes $O(N^3)$ time
- # of parameters = N
- Not spatially localized: every dimension of the result is related to ALL nodes



$$L = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 3 & -1 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$U\hat{h}_\theta U^\top = \begin{bmatrix} 3.363 & -0.819 & -0.205 & -0.205 & -1.135 \\ -0.819 & 3.977 & -0.977 & -0.977 & -0.205 \\ -0.205 & -0.977 & 2.614 & -0.386 & -0.046 \\ -0.205 & -0.977 & -0.386 & 2.614 & -0.046 \\ -1.135 & -0.205 & -0.046 & -0.046 & 2.432 \end{bmatrix}, \quad \text{with } \hat{h}_\theta = \text{diag}\{1, 2, 3, 4, 5\}$$

Kernel Design Example 2/3

Instead, if we pick $\hat{h}_\theta(\lambda_i) = \theta_0 + \theta_1 \lambda_i + \cdots + \theta_K \lambda_i^K$

We will have

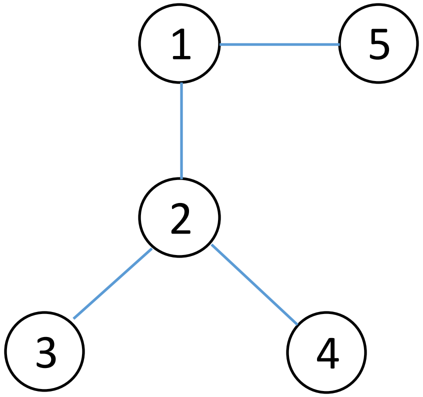
simpler
but still $O(N^3)$

$$U \hat{h}_\theta U^\top = U \left(\sum_{j=0}^{K} \theta_j \Lambda^j \right) U^\top = \sum_{j=0}^{K} \theta_j \left(U \Lambda^j U^\top \right) = \sum_{j=0}^{K} \theta_j L^j$$

only K+1 params

But it has spatial localization!

$$U \hat{h}_\theta U^\top = \begin{bmatrix} \theta_0 & & \\ & \ddots & \\ & & \theta_0 \end{bmatrix}$$



Kernel Design Example 2/3

Instead, if we pick $\hat{h}_\theta(\lambda_i) = \theta_0 + \theta_1 \lambda_i + \cdots + \theta_K \lambda_i^K$

We will have

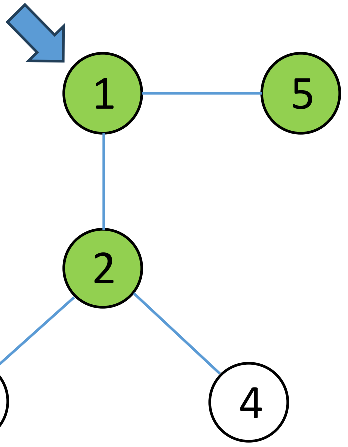
$$U \hat{h}_\theta U^\top = U \left(\sum_{j=0}^{K} \theta_j \Lambda^j \right) U^\top = \sum_{j=0}^{K} \theta_j \left(U \Lambda^j U^\top \right) = \sum_{j=0}^{K} \theta_j L^j$$

simpler
but still $O(N^3)$

only K+1 params

But it has spatial localization!

$$U \hat{h}_\theta U^\top = \begin{bmatrix} \boxed{\theta_0 + 2\theta_1 \quad -\theta_1 \quad 0 \quad 0 \quad -\theta_1} \\ -\theta_1 & \theta_0 + 3\theta_1 & -\theta_1 & -\theta_1 & 0 \\ 0 & -\theta_1 & \theta_0 + \theta_1 & 0 & 0 \\ 0 & -\theta_1 & 0 & \theta_0 + \theta_1 & 0 \\ -\theta_1 & 0 & 0 & 0 & \theta_0 + \theta_1 \end{bmatrix}$$



Kernel Design Example 2/3

Instead, if we pick $\hat{h}_\theta(\lambda_i) = \theta_0 + \theta_1 \lambda_i + \cdots + \theta_K \lambda_i^K$

We will have

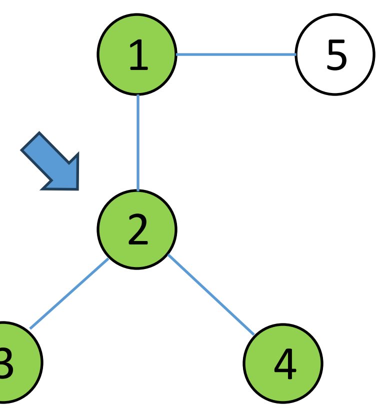
$$U \hat{h}_\theta U^\top = U \left(\sum_{j=0}^{K} \theta_j \Lambda^j \right) U^\top = \sum_{j=0}^{K} \theta_j \left(U \Lambda^j U^\top \right) = \sum_{j=0}^{K} \theta_j L^j$$

simpler
but still $O(N^3)$

only K+1 params

But it has spatial localization!

$$U \hat{h}_\theta U^\top = \begin{bmatrix} \theta_0 + 2\theta_1 & -\theta_1 & 0 & 0 & -\theta_1 \\ -\theta_1 & \theta_0 + 3\theta_1 & -\theta_1 & -\theta_1 & 0 \\ 0 & -\theta_1 & \theta_0 + \theta_1 & 0 & 0 \\ 0 & -\theta_1 & 0 & \theta_0 + \theta_1 & 0 \\ -\theta_1 & 0 & 0 & 0 & \theta_0 + \theta_1 \end{bmatrix}$$



Kernel Design Example 2/3

Instead, if we pick $\hat{h}_\theta(\lambda_i) = \theta_0 + \theta_1 \lambda_i + \cdots + \theta_K \lambda_i^K$

We will have

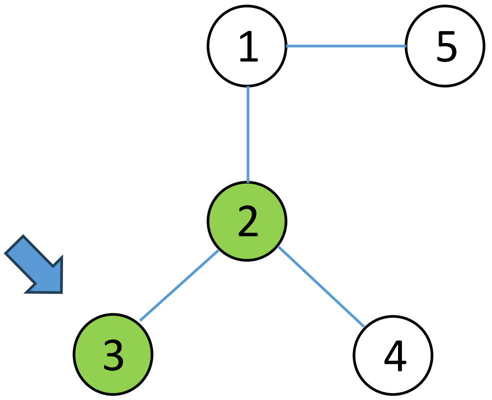
$$U \hat{h}_\theta U^\top = U \left(\sum_{j=0}^{K} \theta_j \Lambda^j \right) U^\top = \sum_{j=0}^{K} \theta_j \left(U \Lambda^j U^\top \right) = \sum_{j=0}^{K} \theta_j L^j$$

simpler
but still $O(N^3)$

only K+1 params

But it has spatial localization!

$$U \hat{h}_\theta U^\top = \begin{bmatrix} \theta_0 + 2\theta_1 & -\theta_1 & 0 & 0 & -\theta_1 \\ -\theta_1 & \theta_0 + 3\theta_1 & -\theta_1 & -\theta_1 & 0 \\ \boxed{0} & \boxed{-\theta_1} & \boxed{\theta_0 + \theta_1} & \boxed{0} & \boxed{0} \\ 0 & -\theta_1 & 0 & \theta_0 + \theta_1 & 0 \\ -\theta_1 & 0 & 0 & 0 & \theta_0 + \theta_1 \end{bmatrix}$$



Kernel Design Example 3/3

Based on this idea, a more efficient kernel choice is discovered:

$$\hat{h}_\theta(\lambda_i) = \theta_0 T_0(\tilde{\lambda}_i) + \theta_1 T_1(\tilde{\lambda}_i) + \cdots + \theta_K T_K(\tilde{\lambda}_i)$$ (Chebyshev polynomial)

This time, we don't even need to compute the power. Just do recursion:

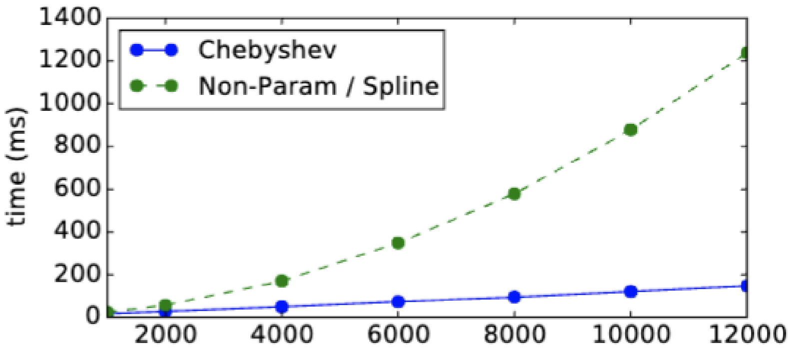
$$\bar{f}_k = T_k(\tilde{L})f \in \mathbb{R}^{n\times 1} \rightsquigarrow \bar{f}_k = 2\tilde{L}\bar{f}_{k-1} - \bar{f}_{k-2}$$ (Time complexity: $O(K|E|)$)

Dataset	Architecture	Accuracy Non-Param (2)	Spline (7) [4]	Chebyshev (4)
MNIST	GC10	95.75	97.26	97.48
MNIST	GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

Table 3: Classification accuracies for different types of spectral filters ($K = 25$).

Model	Architecture	Time (ms) CPU	GPU	Speedup
Classical CNN	C32-P4-C64-P4-FC512	210	31	6.77x
Proposed graph CNN	GC32-P4-GC64-P4-FC512	1600	200	8.00x

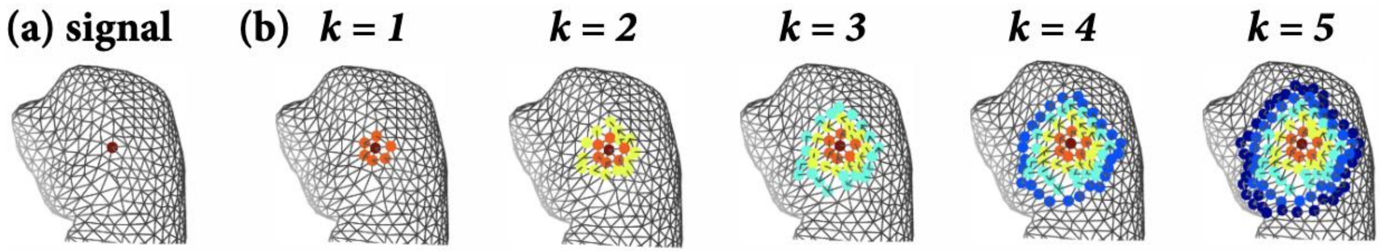
Table 4: Time to process a mini-batch of $S = 100$ MNIST images.



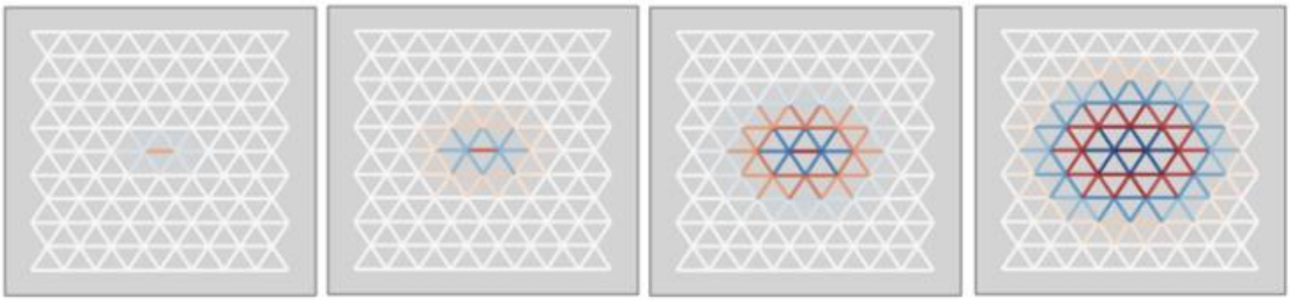
Different Laplacian

Different Laplacian variants can add different information.

e.g. 1 Laplace-Beltrami Operator on a compact manifold:



e.g. 2 Higher-order Laplacian that can encode edge info



Graph Transformers

Self-attn as Message Passing

Recall: for the self-attention update:

$$\mathrm{Attn}(X) = \mathrm{Softmax}(QK^\top)V \qquad Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

If we just focus on token 1:

$$z_1 = \sum_{j=1}^{N} \mathrm{Softmax}_j(q_1^\top k_j)v_j$$

We can see this as:

- Compute message from j: $v_j = W_V x_j, \quad k_j = W_K x_j$
- Compute Query from 1: $q_1 = W_Q x_1$
- Aggregate all messages: $\bigoplus(q_1, \{\phi_j\}) = \sum_{j=1}^{N} \mathrm{Softmax}_j\left(q_1^\top k_j\right) v_j$

Deviate a bit…

If you are already content with this discovery, you will have:

Published as a conference paper at ICLR 2018

GRAPH ATTENTION NETWORKS

Petar Veličković*
Department of Computer Science and Technology
University of Cambridge
petar.velickovic@cst.cam.ac.uk

Guillem Cucurull*
Centre de Visió per Computador, UAB
gcucurull@gmail.com

Arantxa Casanova*
Centre de Visió per Computador, UAB
ar.casanova.8@gmail.com

Adriana Romero
Montréal Institute for Learning Algorithms
adriana.romero.soriano@umontreal.ca

Pietro Liò
Department of Computer Science and Technology
University of Cambridge
pietro.lio@cst.cam.ac.uk

Yoshua Bengio
Montréal Institute for Learning Algorithms
yoshua.umontreal@gmail.com

Graph Transformers

To become a transformer, we still need position encoding.

Idea: we use the adjacency information. Just consider the eigenvectors of the Laplacian

$$L\phi = \lambda\phi$$

$$\begin{array}{c|c} & \begin{array}{ccccc} \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 \end{array} \\ \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} & \begin{bmatrix} \mathbf{0.58} & 0 & 0 & 0.77 & 0.30 \\ \mathbf{0.58} & 0 & 0 & -0.12 & -0.81 \\ \mathbf{0.58} & 0 & 0 & -0.64 & 0.51 \\ 0.00 & -\mathbf{0.71} & -0.71 & 0 & 0 \\ 0.00 & -\mathbf{0.71} & 0.71 & 0 & 0 \end{bmatrix} \end{array}$$

position encoding for node 2

What if we flip the sign?

Graph Transformers

Recall the (i,j) element from $QK^\top$, it describes how much token j contributes to the update of token i

What if the graph has edge features? Do we just overwrite them with the attention?

Idea: we just add them together…

Do Transformers Really Perform Bad for Graph Representation?

Chengxuan Ying[1,*], **Tianle Cai**[2], **Shengjie Luo**[3,*],
Shuxin Zheng[4,†], **Guolin Ke**[4], **Di He**[4,†], **Yanming Shen**[1], **Tie-Yan Liu**[4]
[1]Dalian University of Technology [2]Princeton University
[3]Peking University [4]Microsoft Research Asia
yingchengsyuan@gmail.com, tianle.cai@princeton.edu, luosj@stu.pku.edu.cn
{shuz[†], guoke, dihe[†], tyliu}@microsoft.com, shen@dlut.edu.cn

(NIPS 2020)