Stanford CS224W: Designing Powerful Graph Encoders

Breaking the Limits of the WL kernel

CS224W: Machine Learning with Graphs

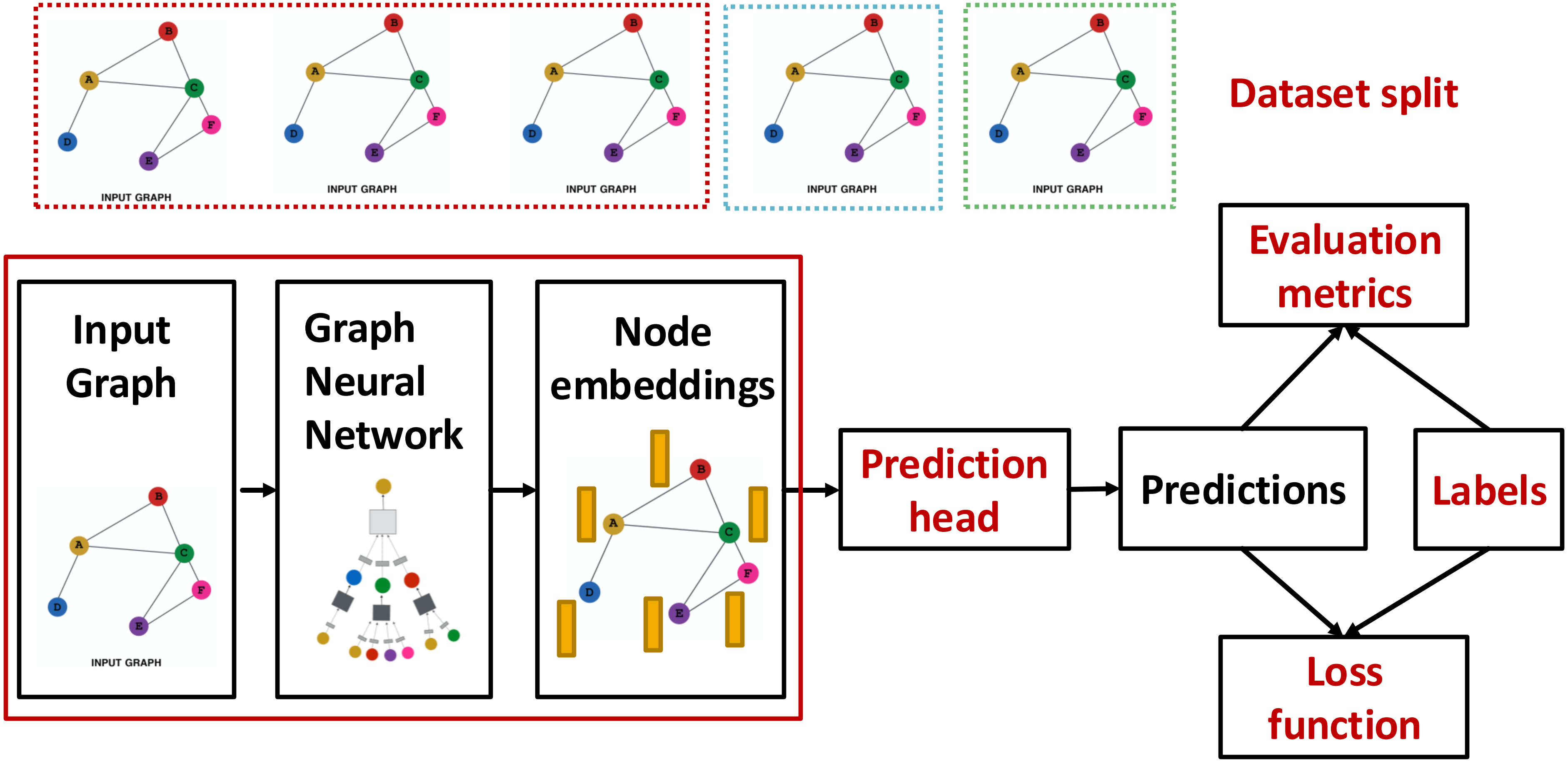
Charilaos Kanatsoulis and Jure Leskovec, Stanford University

http://cs224w.stanford.edu

Announcements

- **Homework 1 due Thursday, 10/16**
  - Late submissions accepted until end of day Monday, 10/20
- **Project Proposal due Tuesday, 10/21**
- **Colab 2 due Thursday, 10/23**

Recap: GNN Training Pipeline



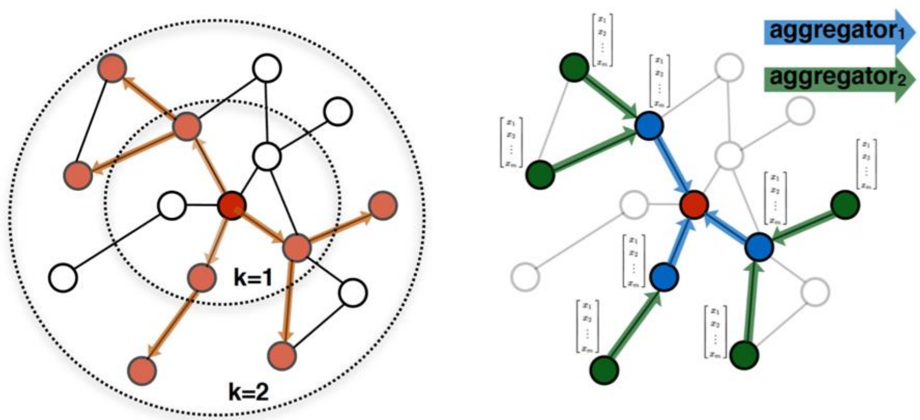
Today's lecture: Can we make GNN representation more expressive?

Stanford CS224W: Limitations of Graph Neural Networks

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

A “Perfect” GNN Model

- **A thought experiment:** What should a perfect GNN do?
  - A k-layer GNN embeds a node based on the K-hop neighborhood structure

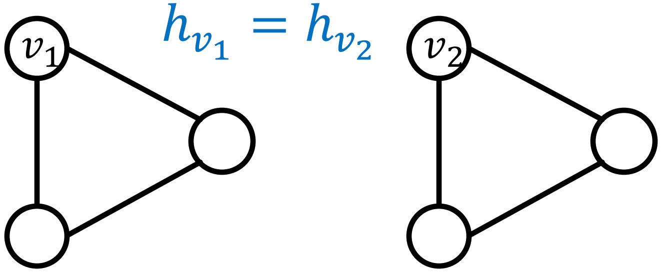


  - A perfect GNN should build **an injective function** between neighborhood structure (regardless of hops) and node embeddings

A "Perfect" GNN Model

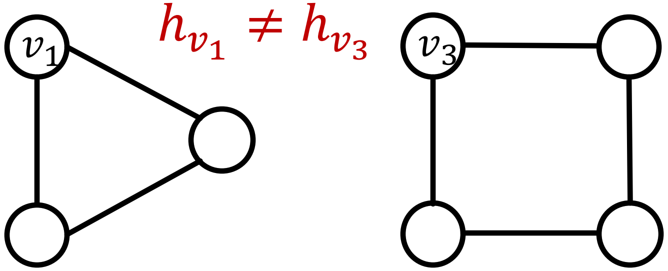
- **For a perfect GNN (ignore node attributes for now):**
  - **Observation 1:** If two nodes have the same neighborhood structure, they must have the same embedding

$$h_{v_1} = h_{v_2}$$



  - **Observation 2:** If two nodes have different neighborhood structure, they must have different embeddings

$$h_{v_1} \neq h_{v_3}$$



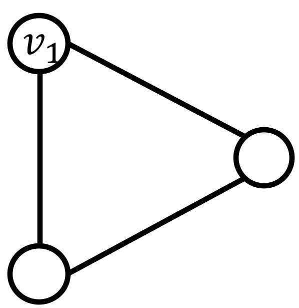
(Considering that attributes of all nodes are the same)

J. You, J. Gomes-Selman, R. Ying, J. Leskovec. [Identity-aware Graph Neural Networks](), AAAI 2021

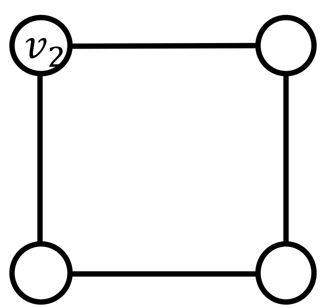
Imperfections of Existing GNNs

- **Observation 2 often cannot be satisfied:**
  - **The GNNs we have introduced so far are not perfect**
  - In previous lecture, we discussed that their expressive power is **upper bounded by the WL test**
  - For example, message passing GNNs **cannot count the cycle length:**

v_1 resides in a cycle with length 3

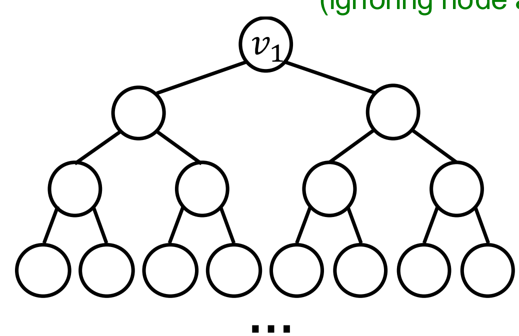


v_2 resides in a cycle with length 4



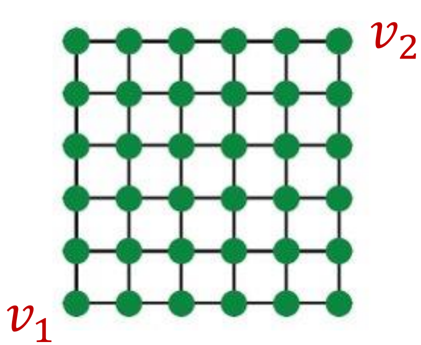
The computational graphs for nodes v_1 and v_2 are always the same

(ignoring node attributes)



Imperfections of Existing GNNs

- **Observation 1 could also have issues:**
  - Even though two nodes may have the same neighborhood structure, we may want to assign different embeddings to them
  - Because these nodes appear in **different positions in the graph**
  - We call these tasks **Position-aware tasks**
  - **Even a perfect GNN will fail for these tasks:**



A grid graph

NYC road network

Plan for the Lecture

We will resolve both issues by **building more expressive GNNs**

- **Fix issues in Observation 2:**
  - Build message passing GNNs that are more expressive than WL test
  - Example method: **Structurally-aware GNNs**
- **Fix issues in Observation 1:**
  - Create node embeddings based on their positions in the graph
  - Example method: **Position-aware GNNs**

Stanford CS224W: A Spectral Perspective of message-passing GNNs

CS224W: Machine Learning with Graphs
Charilaos Kanatsoulis and Jure Leskovec, Stanford University

http://cs224w.stanford.edu

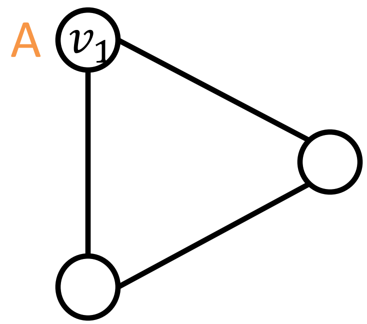
More Failure Cases for GNNs

- GNNs exhibit three levels of failure cases in structure-aware tasks:
  - Node level
  - Edge level
  - Graph level

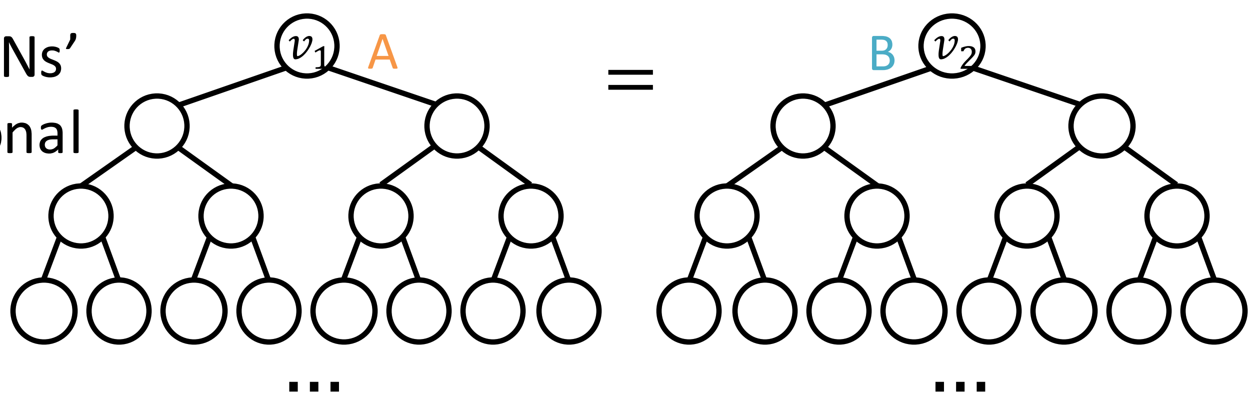
GNN Failure 1: Node-level Tasks

Different Inputs but the same computational graph → GNN fails

Example input graphs



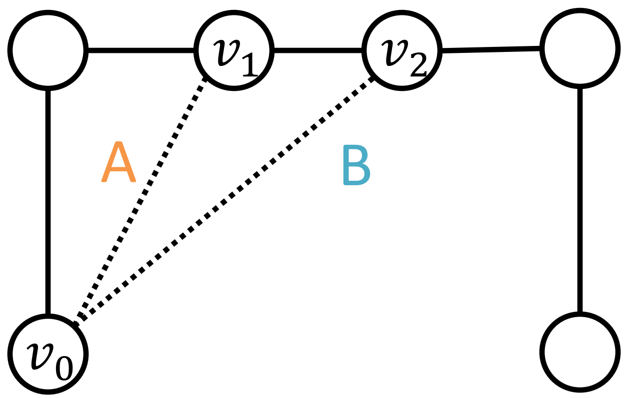
Existing GNNs' computational graphs



GNN Failure 2: Edge-level Tasks

Different Inputs but the same computational graph → GNN fails

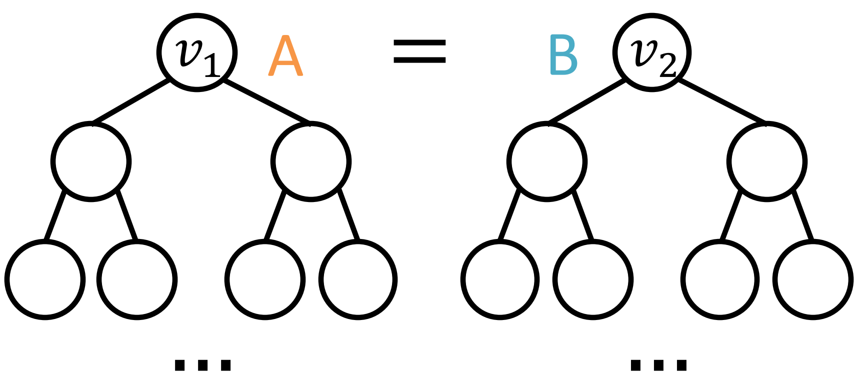
Example input graphs



Edge A and B share node v_0

We look at embeddings for v_1 and v_2

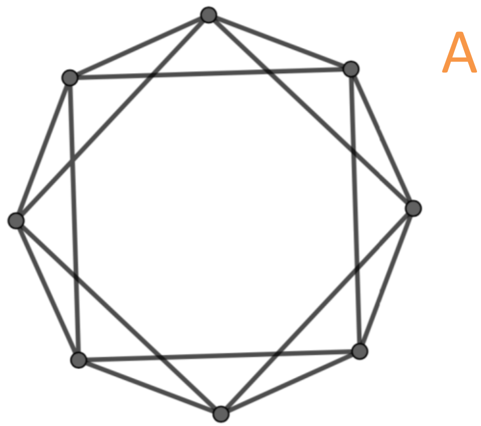
Existing GNNs' computational graphs

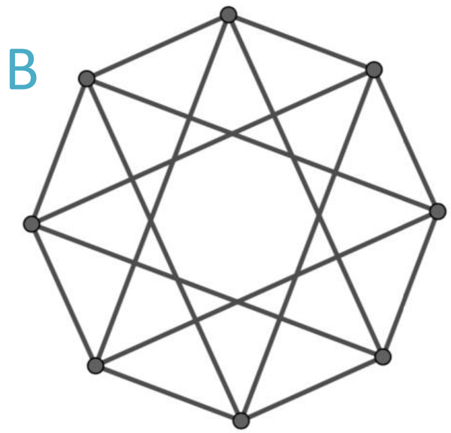


GNN Failure 3: Graph-level Tasks

Different Inputs but the same computational graph → GNN fails

Example input graphs



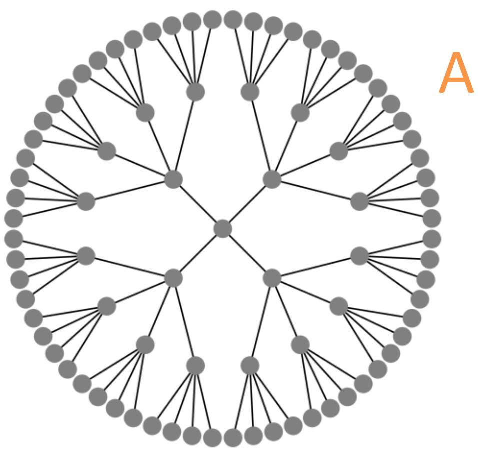


We look at embeddings for each node

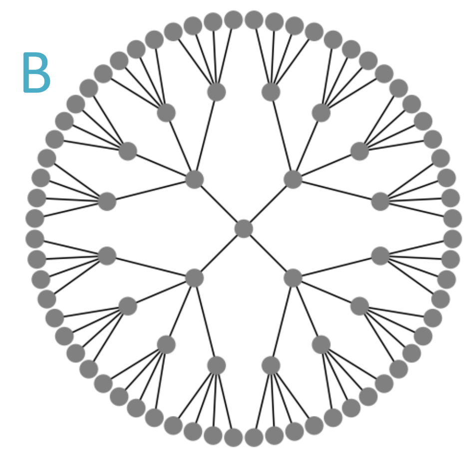
For each node:

For each node:

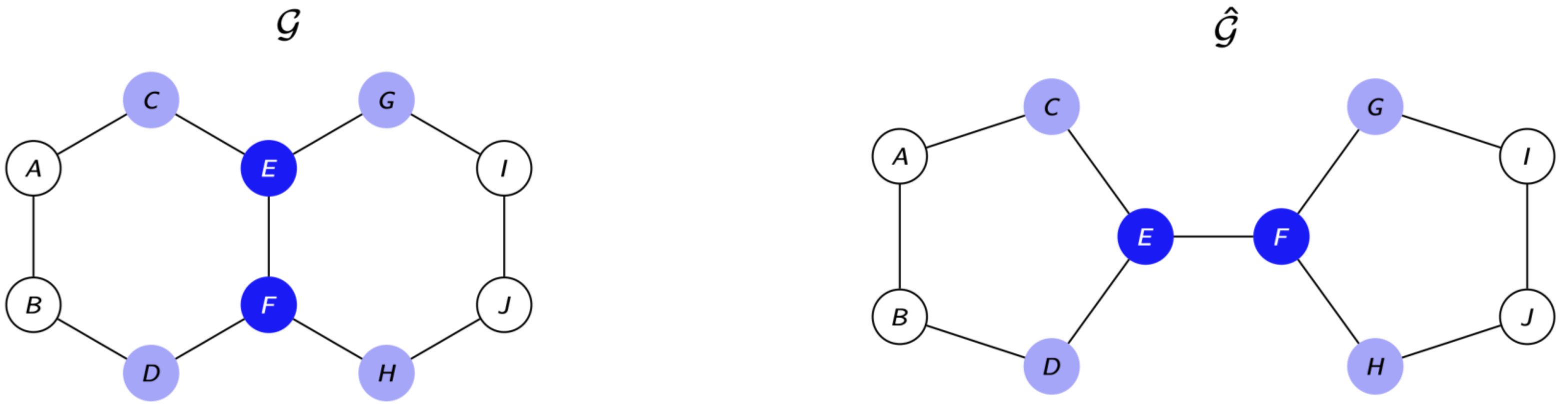
Existing GNNs' computational graphs



=



Limitation of the WL kernel



- **The WL kernel** colors **inherit the graph symmetries**.
- **Symmetric colors** are associated with limitations involving the **spectral decomposition of the graph**.

Matrix representation of GIN

- **Recall the GIN update:**

$$\boldsymbol{c}_v^{(l+1)} = \mathrm{MLP}\left((1+\epsilon)\, \boldsymbol{c}_v^{(l)} + \sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)} \right)$$

  - We can consider a single-layer MLP:

$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}^{(l)} (1+\epsilon)\, \boldsymbol{c}_v^{(l)} + \boldsymbol{W}^{(l)} \sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)} \right)$$

  - σ is a pointwise activation.

Matrix representation of GIN

- **Recall the GIN update:**

$$\boldsymbol{c}_v^{(l+1)} = \mathrm{MLP}\left((1+\epsilon)\, \boldsymbol{c}_v^{(l)} + \sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)} \right)$$

  - We can consider a single-layer MLP:

$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)} \boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)} \sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)} \right)$$

  - σ is a pointwise activation.

Matrix representation of GIN

- **Recall the GIN update:**

$$\boldsymbol{c}_v^{(l+1)} = \mathrm{MLP}\left((1+\epsilon)\, \boldsymbol{c}_v^{(l)} + \sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)} \right)$$

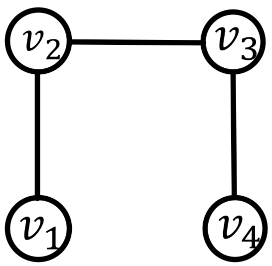
  - We can consider a single-layer MLP:

$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)} \boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)} \sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)} \right)$$

  - σ is a pointwise activation.
  - GraphSAGE with sum aggregation!
  - Sparse matrix-vector multiplication (see linear algebra recitation)

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**

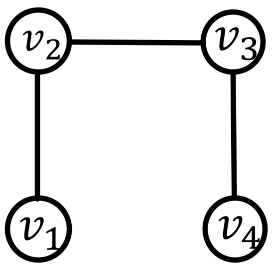


A =

0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



A =

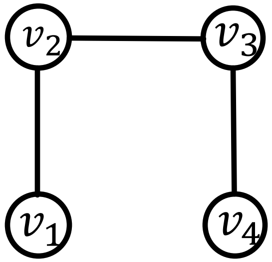
0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0

$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)}\boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)}\sum_{u\in\mathcal{N}(v)}\boldsymbol{c}_u^{(l)}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N\times d}, \quad \boldsymbol{C}^{(l)}[v,:] = \boldsymbol{c}_v^{(l)}$$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



A =

0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0

$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)}\boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)}\sum_{u\in\mathcal{N}(v)}\boldsymbol{c}_u^{(l)}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N\times d}, \quad \boldsymbol{C}^{(l)}[v,:] = \boldsymbol{c}_v^{(l)}$$

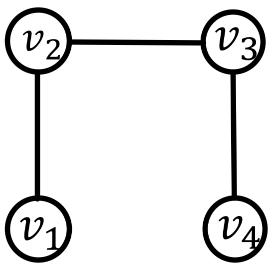
$\boldsymbol{A}\boldsymbol{C}^{(l)}$ **=**

0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0

$\boldsymbol{C}^{(l)}$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



A =

0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0

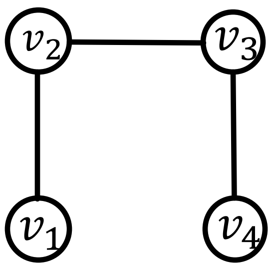
$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)}\boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)}\sum_{u\in\mathcal{N}(v)}\boldsymbol{c}_u^{(l)}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N\times d}, \quad \boldsymbol{C}^{(l)}[v,:] = \boldsymbol{c}_v^{(l)}$$

$$\boldsymbol{A}\boldsymbol{C}^{(l)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{C}^{(l)} = \begin{array}{c} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{l} \boldsymbol{c}_{v_1}^{(l)} \\ \boldsymbol{c}_{v_2}^{(l)} \\ \boldsymbol{c}_{v_3}^{(l)} \\ \boldsymbol{c}_{v_4}^{(l)} \end{array}$$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

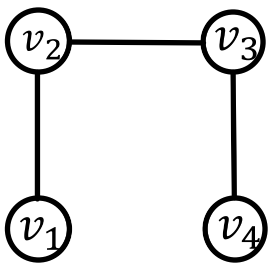
$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)}\boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)}\boxed{\sum_{u\in\mathcal{N}(v)}\boldsymbol{c}_u^{(l)}}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N\times d}, \quad \boldsymbol{C}^{(l)}\,[v,:] = \boldsymbol{c}_v^{(l)}$$

$$\boxed{\boldsymbol{A}\boldsymbol{C}^{(l)}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{C}^{(l)} = \begin{array}{c} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{l} \boldsymbol{c}_{v_1}^{(l)} \\ \boldsymbol{c}_{v_2}^{(l)} \\ \boldsymbol{c}_{v_3}^{(l)} \\ \boldsymbol{c}_{v_4}^{(l)} \end{array}$$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



A =

0	1	0	0
1	0	1	0
0	1	0	1
0	0	1	0

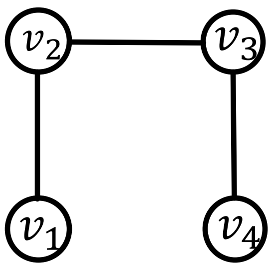
$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)} \boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)} \boxed{\sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)}}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N \times d}, \quad \boldsymbol{C}^{(l)}[v, :] = \boldsymbol{c}_v^{(l)}$$

$$\boldsymbol{A}\boldsymbol{C}^{(l)}[1, :] = \begin{array}{cc} \begin{bmatrix} 0 & \color{red}{1} & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{matrix} \boldsymbol{c}_{v_1}^{(l)} \\ \boxed{\boldsymbol{c}_{v_2}^{(l)}} \\ \boldsymbol{c}_{v_3}^{(l)} \\ \boldsymbol{c}_{v_4}^{(l)} \end{matrix} \end{array}$$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

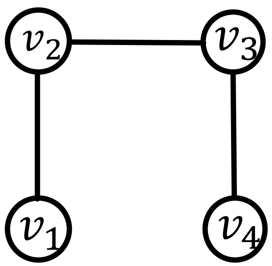
$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)}\boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)}\boxed{\sum_{u\in\mathcal{N}(v)}\boldsymbol{c}_u^{(l)}}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N\times d}, \quad \boldsymbol{C}^{(l)}[v,:] = \boldsymbol{c}_v^{(l)}$$

$$\boldsymbol{A}\boldsymbol{C}^{(l)}[2\,,:] = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} \boxed{\boldsymbol{c}_{v_1}^{(l)}} \\ \boldsymbol{c}_{v_2}^{(l)} \\ \boxed{\boldsymbol{c}_{v_3}^{(l)}} \\ \boldsymbol{c}_{v_4}^{(l)} \end{matrix}$$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

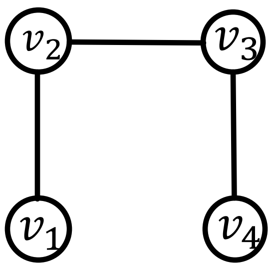
$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)} \boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)} \boxed{\sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)}}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N \times d}, \quad \boldsymbol{C}^{(l)}[v, :] = \boldsymbol{c}_v^{(l)}$$

$$\boldsymbol{A}\boldsymbol{C}^{(l)}[3\ ,\ :] \quad = \quad \begin{array}{c} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \boldsymbol{c}_{v_1}^{(l)} \\ \boxed{\boldsymbol{c}_{v_2}^{(l)}} \\ \boldsymbol{c}_{v_3}^{(l)} \\ \boxed{\boldsymbol{c}_{v_4}^{(l)}} \end{array}$$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

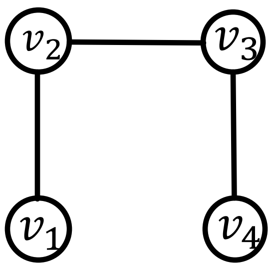
$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)}\boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)}\boxed{\sum_{u\in\mathcal{N}(v)}\boldsymbol{c}_u^{(l)}}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N\times d}, \quad \boldsymbol{C}^{(l)}[v,:] = \boldsymbol{c}_v^{(l)}$$

$$\boldsymbol{A}\boldsymbol{C}^{(l)}[4\,,:] \quad = \quad \begin{array}{c} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & \color{red}{1} & 0 \end{bmatrix} \end{array} \begin{array}{c} \boldsymbol{c}_{v_1}^{(l)} \\ \boldsymbol{c}_{v_2}^{(l)} \\ \boxed{\boldsymbol{c}_{v_3}^{(l)}} \\ \boldsymbol{c}_{v_4}^{(l)} \end{array}$$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)}\boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)}\boxed{\sum_{u\in\mathcal{N}(v)}\boldsymbol{c}_u^{(l)}}\right)$$

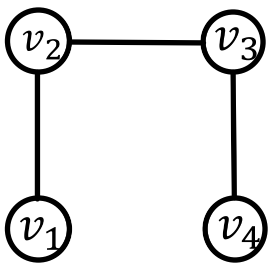
$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N\times d}, \quad \boldsymbol{C}^{(l)}[v,:] = \boldsymbol{c}_v^{(l)}$$

$$\boldsymbol{AC}^{(l)}[4\,,:] = \begin{array}{c} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \boldsymbol{c}_{v_1}^{(l)} \\ \boldsymbol{c}_{v_2}^{(l)} \\ \boxed{\boldsymbol{c}_{v_3}^{(l)}} \\ \boldsymbol{c}_{v_4}^{(l)} \end{array}$$

$$\boldsymbol{C}^{(l+1)} = \sigma\left(\boldsymbol{C}^{(l)}\boldsymbol{W}_0^{(l)} + \boldsymbol{AC}^{(l)}\boldsymbol{W}_1^{(l)}\right)$$

Neighborhood Aggregation as Matrix-vector Mult.

- **Key object:** Adjacency Matrix **A**



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{c}_v^{(l+1)} = \sigma\left(\boldsymbol{W}_0^{(l)}\boldsymbol{c}_v^{(l)} + \boldsymbol{W}_1^{(l)} \boxed{\sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)}}\right)$$

$$\boldsymbol{C}^{(l)} \in \mathbb{R}^{N \times d}, \quad \boldsymbol{C}^{(l)}[v,:] = \boldsymbol{c}_v^{(l)}$$

$$\boldsymbol{A}\boldsymbol{C}^{(l)}[4,:] \quad = \quad \begin{array}{c} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \boldsymbol{c}_{v_1}^{(l)} \\ \boldsymbol{c}_{v_2}^{(l)} \\ \boxed{\boldsymbol{c}_{v_3}^{(l)}} \\ \boldsymbol{c}_{v_4}^{(l)} \end{array}$$

$$\boldsymbol{C}^{(l+1)} = \sigma\left(\boldsymbol{C}^{(l)}\boldsymbol{W}_0^{(l)} + \boldsymbol{A}\boldsymbol{C}^{(l)}\boldsymbol{W}_1^{(l)}\right) = \sigma\left(\sum_{k=0}^{1} \boldsymbol{A}^k \boldsymbol{C}^{(l)} \boldsymbol{W}_k^{(l)}\right)$$

Matrix representation of GIN

- **Recall the GIN update:**

$$\boldsymbol{c}_v^{(l+1)} = \mathrm{MLP}\left((1+\epsilon)\, \boldsymbol{c}_v^{(l)} + \sum_{u \in \mathcal{N}(v)} \boldsymbol{c}_u^{(l)} \right)$$

  - We can consider a single-layer MLP
  - Write the color update in a matrix form:

$$\boldsymbol{C}^{(l+1)} = \sigma\left(\boldsymbol{C}^{(l)} \boldsymbol{W}_0^{(l)} + \boldsymbol{A}\boldsymbol{C}^{(l)} \boldsymbol{W}_1^{(l)} \right) = \sigma\left(\sum_{k=0}^{1} \boldsymbol{A}^k \boldsymbol{C}^{(l)} \boldsymbol{W}_k^{(l)} \right)$$

  - $\boldsymbol{C}^{(l)} \in \mathbb{R}^{N \times d}, \quad \boldsymbol{C}^{(l)}[v,:] = \boldsymbol{c}_v^{(l)}$
  - Where $A \epsilon \{0,1\}^{N \times N}$ is the adjacency matrix of the graph, i.e., $A[u,v] = 1$ if (u,v) is an edge and $A[u,v] = 0$ if (u,v) is not an edge.

Spectral Graph Representation

- **Let's compute the eigenvalue decomposition of the graph adjacency matrix A.**

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$$

- $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$ is the orthonormal matrix of eigenvectors
- $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues $\{\lambda_n\}_{n=1}^{N}$
  - The eigenvalue (**spectral**) decomposition of the adjacency is a **universal characterization** of the graph.
  - **Different graphs have different spectral decompositions**
  - The **number of cycles** in a graph can be viewed as **functions of eigenvalues and eigenvectors**, e.g.,

$$\#\text{triangles} = \text{diag}\left(\boldsymbol{A}^3\right) = \sum_{n=1}^{N} \lambda_n^3 \left|\boldsymbol{v}_n\right|^2$$

GNN as functions of eigenvectors

- **We can interpret GIN layers as MLPs operating on the eigenvectors:**

$$\boldsymbol{C}^{(l+1)} = \sigma\left(\sum_{k=0}^{1} \boldsymbol{A}^k \boldsymbol{C}^{(l)} \boldsymbol{W}_k^{(l)}\right)$$

- **If we replace A with the spectral decomposition** $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$

$$\boldsymbol{C}^{(l+1)} = \sigma\left(\sum_{k=0}^{1} \boldsymbol{A}^k \boldsymbol{C}^{(l)} \boldsymbol{W}_k^{(l)}\right) = \sigma\left(\sum_{k=0}^{1} \boldsymbol{V}\boldsymbol{\Lambda}^k \boldsymbol{V}^T \boldsymbol{C}^{(l)} \boldsymbol{W}_k^{(l)}\right)$$

GNN as functions of eigenvectors

- **We can interpret GIN layers as MLPs operating on the eigenvectors:**

$$\boldsymbol{C}^{(l+1)} = \sigma\left(\sum_{k=0}^{1} \boldsymbol{A}^k \boldsymbol{C}^{(l)} \boldsymbol{W}_k^{(l)}\right)$$

- **If we replace A with the spectral decomposition** $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$

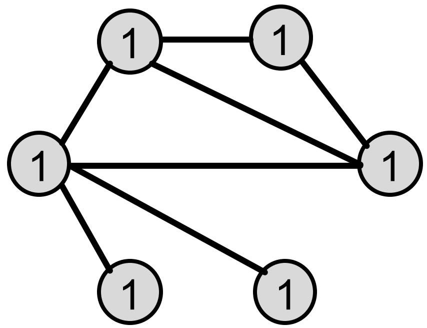
$$\boldsymbol{C}^{(l+1)} = \sigma\left(\sum_{k=0}^{1} \boldsymbol{A}^k \boldsymbol{C}^{(l)} \boldsymbol{W}_k^{(l)}\right) = \sigma\left(\sum_{k=0}^{1} \boldsymbol{V}\boldsymbol{\Lambda}^k \boxed{\boldsymbol{V}^T \boldsymbol{C}^{(l)}} \boldsymbol{W}_k^{(l)}\right)$$

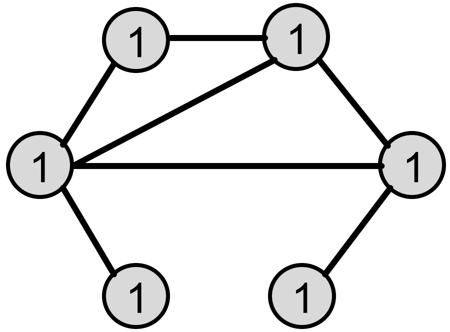
- The GNN update depends on the **eigenvalues, eigenvectors** and the **dot product between the eigenvectors and the colors at the previous level**.

Recall Color Refinement

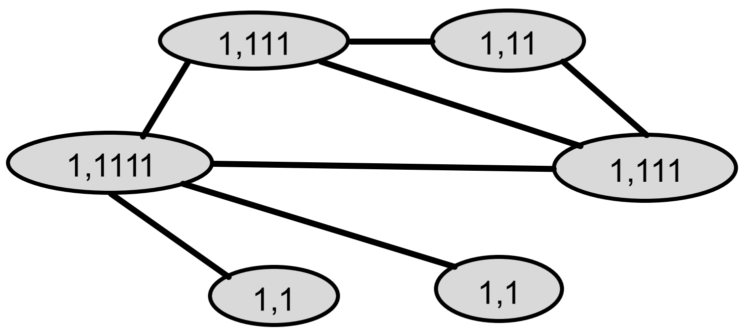
Example of color refinement given two graphs

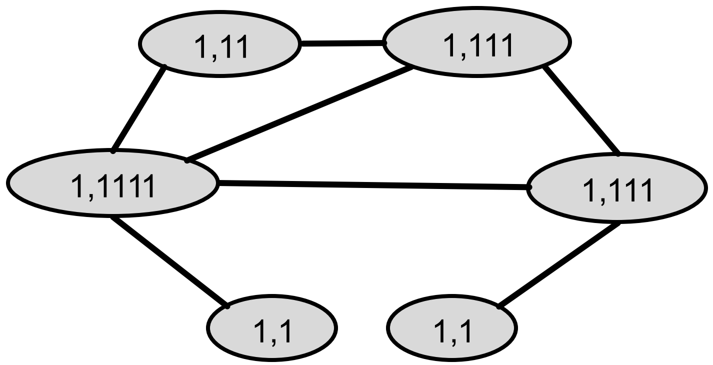
- Assign initial colors





- Aggregate neighboring colors





GNN with uniform initial colors

$$C^{(l+1)} = \sigma\left(\sum_{k=0}^{1} A^k C^{(l)} W_k^{(l)}\right) = \sigma\left(\sum_{k=0}^{1} V \Lambda^k \boxed{V^T C^{(l)}} W_k^{(l)}\right)$$

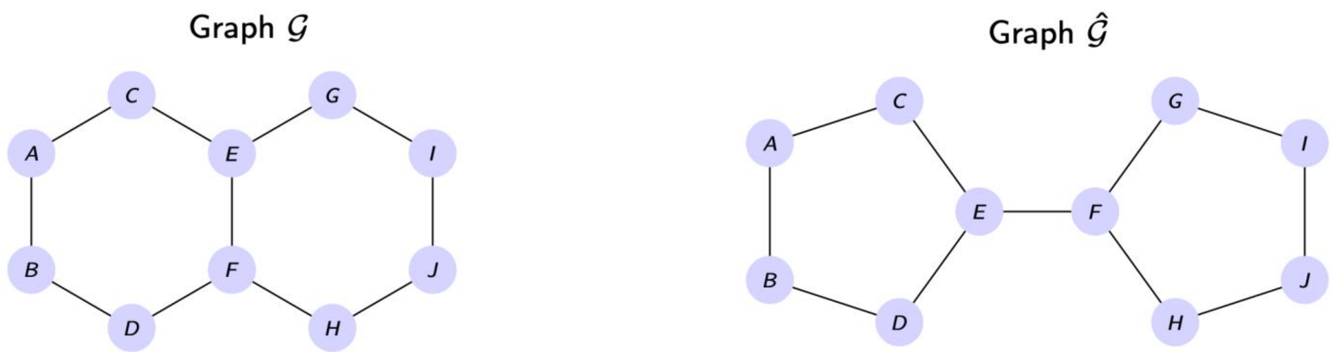
$$V^T C^{(0)} = V^T \mathbf{1}$$

$$V^T \mathbf{1} = [\langle v_1, \mathbf{1}\rangle, \ldots, \langle v_n, \mathbf{1}\rangle]$$

- The new node colors **depend** on the **eigenvectors that are not orthogonal to 1**.
- **Graphs with symmetries** admit **eigenvectors orthogonal to 1.**

Spectral limitation of the WL kernel

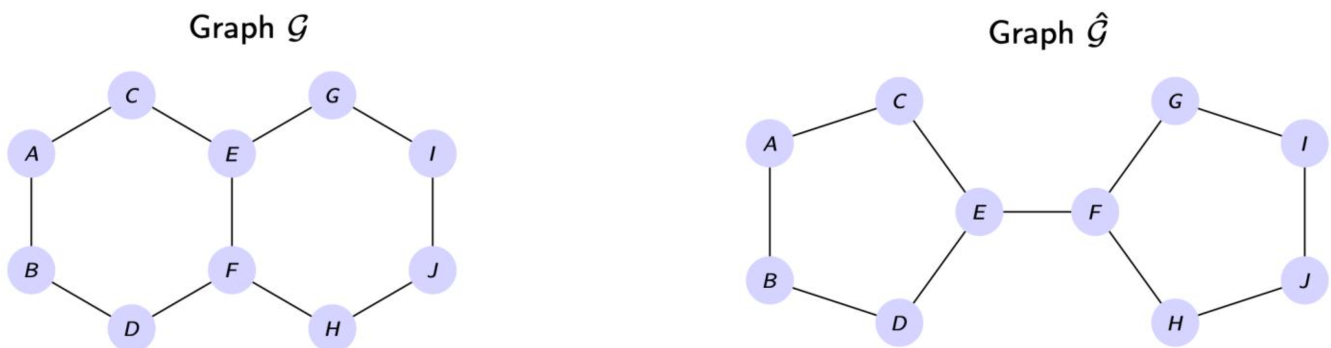
- **The WL kernel cannot distinguish between some basic graph structures, e.g.,**



$\mathcal{G}$	λ_n	2.303	1.618	1.303	1	0.618	-2.303	-1.618	-0.618	-1	-1.303
$\hat{\mathcal{G}}$	$\hat{\lambda}_n$	2.303	1.861	1	0.618	0.618	0.254	-1.303	-1.618	-1.618	-2.115

Spectral limitation of the WL kernel

- **The WL kernel cannot distinguish between some basic graph structures, e.g.,**



$\mathcal{G}$	λ_n	2.303	1.618	1.303	1	0.618	-2.303	-1.618	-0.618	-1	-1.303
	$\langle \boldsymbol{v}_n, \mathbf{1} \rangle$	3.048	0	0	-0.816	0	0	0	0	0	-0.210
$\hat{\mathcal{G}}$	$\hat{\lambda}_n$	2.303	$\boxed{1.861}$	1	0.618	0.618	0.254	-1.303	-1.618	-1.618	-2.115
	$\langle \hat{\boldsymbol{v}}_n, \mathbf{1} \rangle$	3.048	$\boxed{0}$	-0.816	0	0	0	-0.210	0	0	0

Spectral limitation of the WL kernel

- **The WL kernel cannot count basic graph structures:**

$$\#\text{triangles} = \text{diag}\left(\boldsymbol{A}^3\right) = \sum_{n=1}^{N} \lambda_n^3 \left|\boldsymbol{v}_n\right|^2$$

$\mathcal{G}$	λ_n	2.303	1.618	1.303	1	0.618	-2.303	-1.618	-0.618	-1	-1.303
	$\langle \boldsymbol{v}_n, \boldsymbol{1} \rangle$	3.048	0	0	-0.816	0	0	0	0	0	-0.210
$\hat{\mathcal{G}}$	$\hat{\lambda}_n$	2.303	$\boxed{1.861}$	1	0.618	0.618	0.254	-1.303	-1.618	-1.618	-2.115
	$\langle \hat{\boldsymbol{v}}_n, \boldsymbol{1} \rangle$	3.048	$\boxed{0}$	-0.816	0	0	0	-0.210	0	0	0

Limitations of the WL kernel

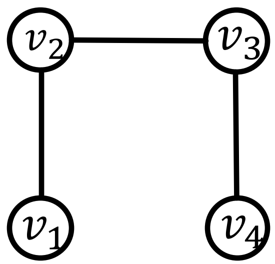
- **Summary: The limitations of the WL kernel are limitations of the initial node color.**
  - These limitations are well understood in the **spectral domain**.
    - Constant node colorings are orthogonal with adjacency eigenvectors and **critical spectral components** (eigenvalues and eigenvectors) **are omitted**.
  - In a high level, colors generated by the WL kernel obey the **same symmetries as graph structure**.
  - These joint symmetries **lock** the message-passing operations to limited representations.

Stanford CS224W: Feature Augmentation: Eigen-vector-based GNNs

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

Adjacency/Laplacian Eigenvector Feature Augmentation

- **What if we use eigenvectors as additional features?**
- **Key object:** Laplacian Matrix **L = Degrees - Adjacency**
  - Each graph has its own Laplacian matrix
  - Laplacian encodes the graph structure
  - Several Laplacian variants that add degree information differently



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

Degree of each node

Adjacency

Laplacian Eigenvector Feature Augmentation

- Laplacian matrix captures graph structure
- **Its eigenvectors inherit this structure**
- Eigenvectors with small eigenvalue = global structure, large eigenvalue = local symmetries

Refresher

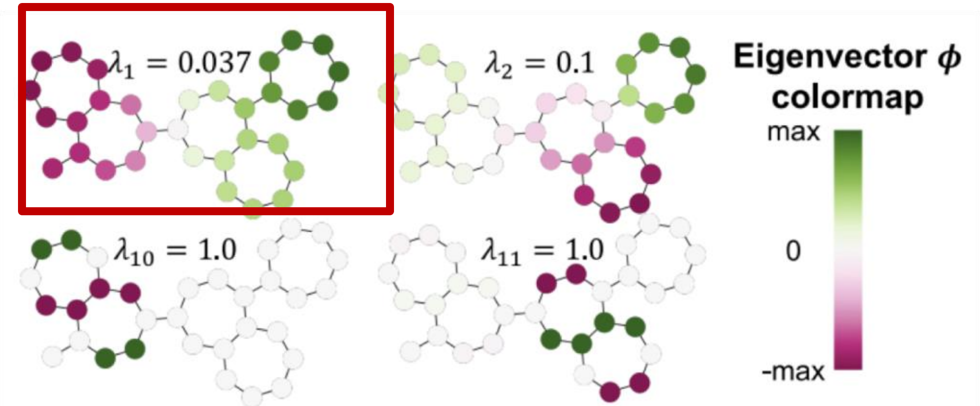
$\boxed{\text{Eigenvector: } v \text{ such that } Lv = \lambda v}$

$L: n \times n$ matrix

$v: n$ dimensional vector

λ: Scalar eigenvalue

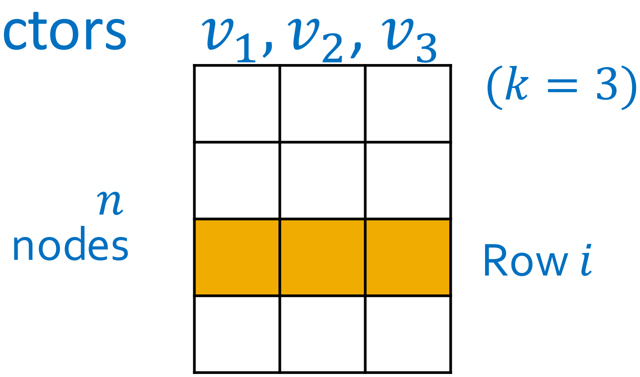
Visualize one eigenvector

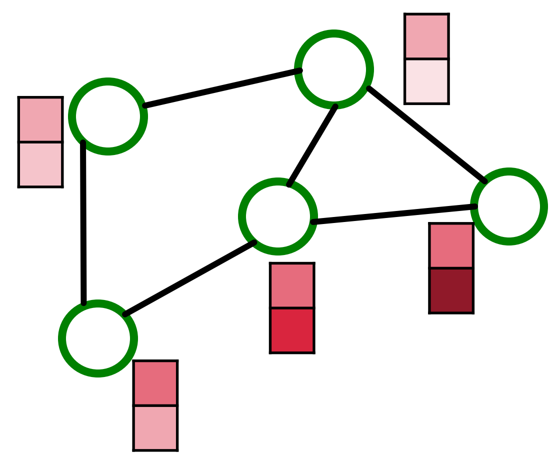


(Figure from Kreuzer* and Beaini* et al. 2021)

Laplacian Eigenvector Positional Encodings

- **Feature augmentation steps:**
  - 1. compute k eigenvectors
  - 2. Stack into matrix:
  - 3. ith row is a feature for node i





Laplacian Eigenvectors in Practice

- Task: given a graph, predict YES if it has a cycle, NO otherwise
- "PE" indicates feature augmentation with Laplacian Eigenvector

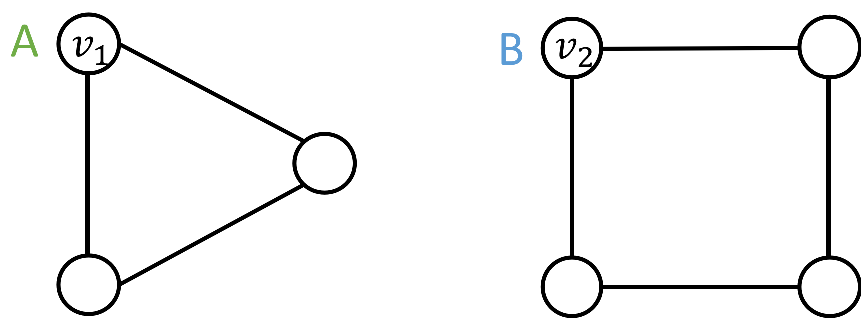
Train samples →			200	500	1000	5000
Model	L	**#Param**	**Test Acc±s.d.**			
GIN	4	100774	70.585±0.636	74.995±1.226	78.083±1.083	86.130±1.140
GIN-PE	4	102864	**86.720±3.376**	**95.960±0.393**	**97.998±0.300**	**99.570±0.089**
GatedGCN	4	103933	50.000±0.000	50.000±0.000	50.000±0.000	50.000±0.000
GatedGCN-PE	4	105263	**95.082±0.346**	**96.700±0.381**	**98.230±0.473**	**99.725±0.027**

Stanford CS224W: Feature Augmentation: Structurally-Aware GNNs

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

Our Approach

- **We use the following thinking:**
  - Two different inputs (nodes, edges, graphs) are labeled differently
  - A “failed” model will always assign the same embedding to them
  - A “successful” model will always assign different embeddings to them
  - **Embeddings are determined by GNN computational graphs:**

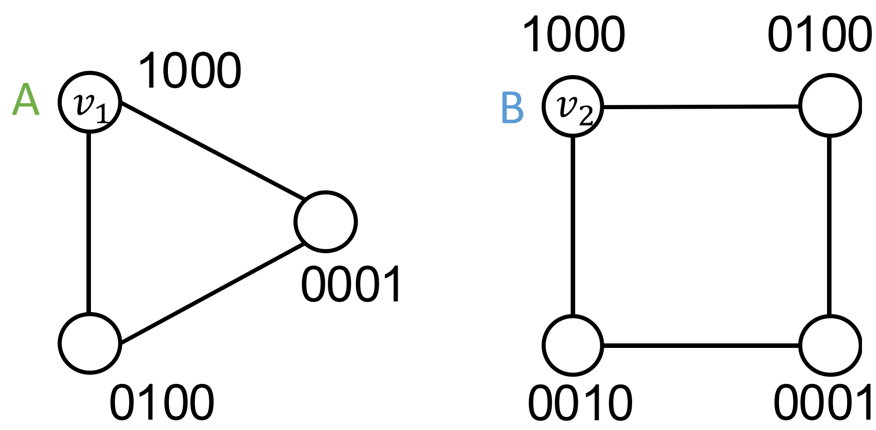


Two inputs: nodes v_1 and v_2
Different labels: A and B
Goal: assign different embeddings to v_1 and v_2

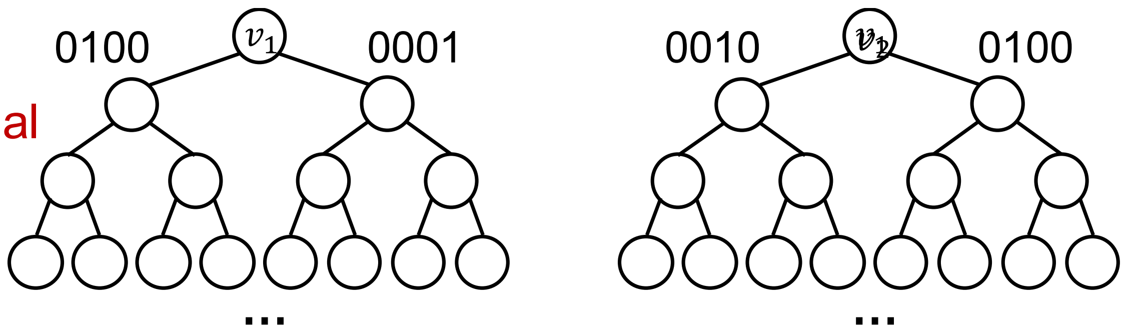
Naïve Solution is not Desirable

- **A naïve solution:** One-hot encoding
  - Encode each node with a different ID, then we can always differentiate different nodes/edges/graphs

Input graphs



Computational graphs

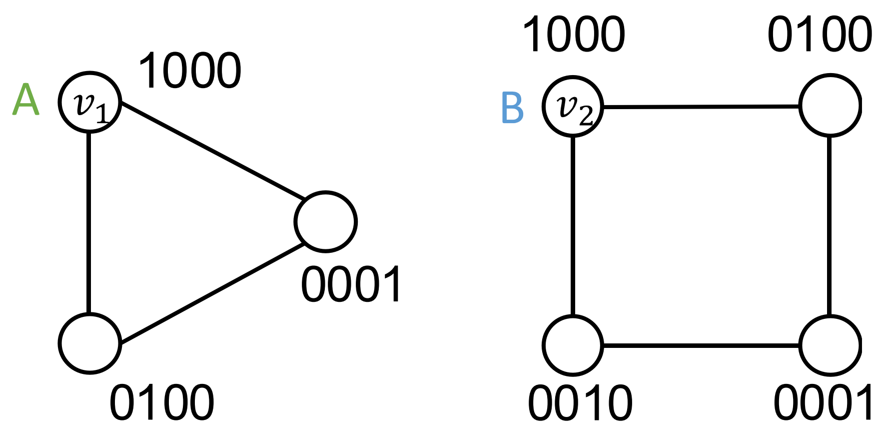


Computational graphs are clearly different if each node has a different ID

Naïve Solution is not Desirable

- **A naïve solution:** One-hot encoding
  - Encode each node with a different ID, then we can always differentiate different nodes/edges/graphs

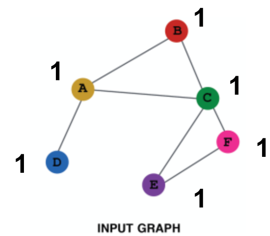
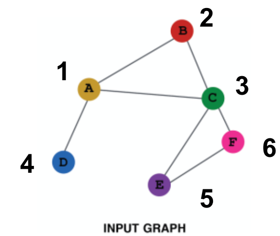
Input graphs



  - **Issues:**
    - **Not scalable**: Need $O(N)$ feature dimensions (N is the number of nodes)
    - **Not inductive:** Cannot generalize to new nodes/graphs

Feature Augmentation on Graphs

- Feature augmentation: **constant** vs. **one-hot**

	Constant node feature 	**One-hot node feature** 
Expressive power	**Medium**. All the nodes are identical, but GNN can still learn from the graph structure	**High**. Each node has a unique ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	**High**. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	**Low**. Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
Computational cost	**Low**. Only 1 dimensional feature	**High**. High dimensional feature, cannot apply to large graphs
Use cases	Any graph, inductive settings (generalize to new nodes)	Small graph, transductive settings (no new nodes)

Feature Augmentation on Graphs

Why do we need feature augmentation?

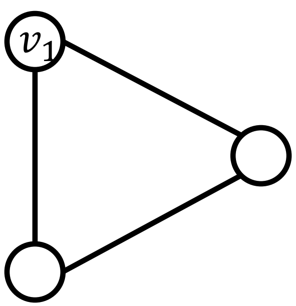
- **(2) Certain structures are hard to learn by GNN**
- **Solution:**
  - We can use cycle count as augmented node features

We start from cycle with length 0

Augmented node feature for v_1

[0, 0, 0, 1, 0, 0]

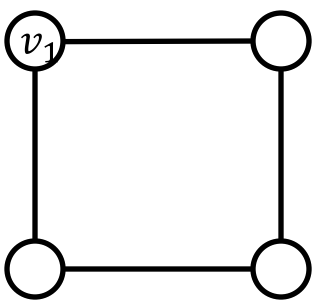
v_1 resides in a cycle with length 3



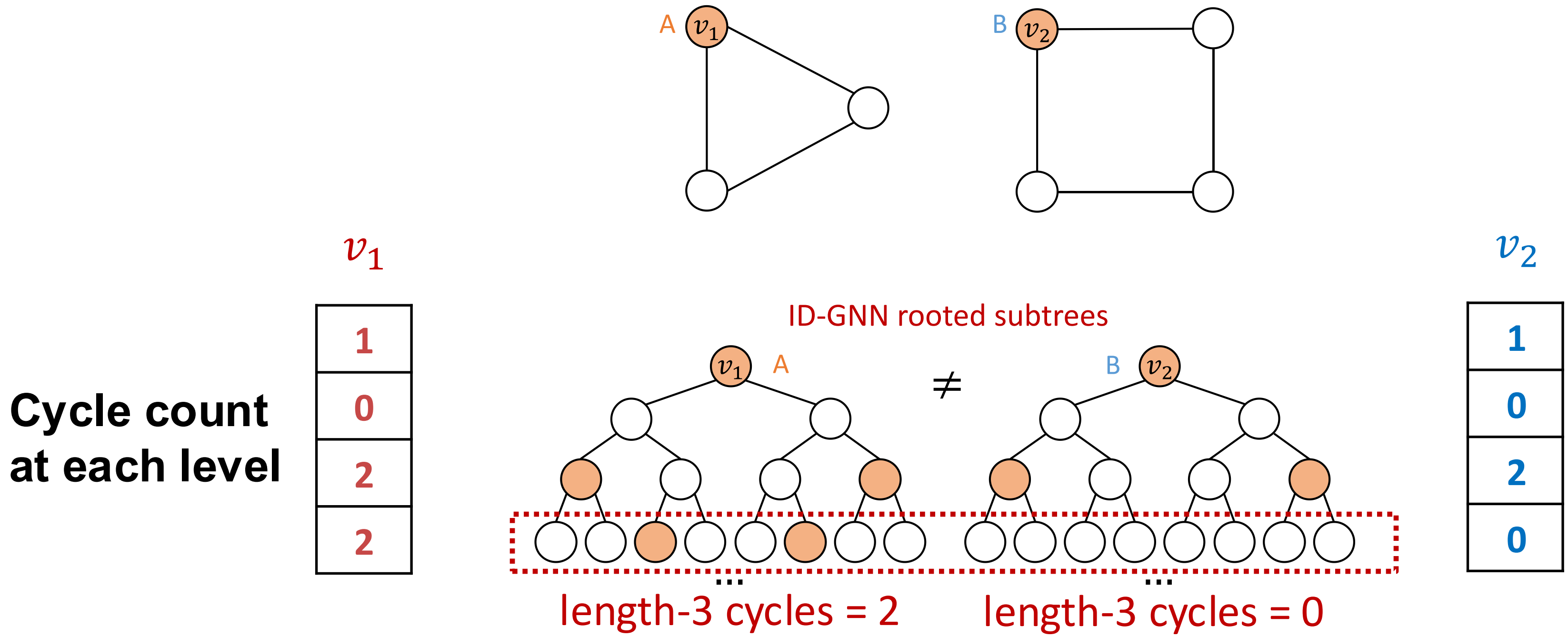
Augmented node feature for v_1

[0, 0, 0, 0, 1, 0]

v_1 resides in a cycle with length 4



ID-GNN-Fast



- Idea: Count cycles originating from a given node, use it as initial feature.
  - Include identity information as an **augmented node feature**
  - **Use cycle counts in each layer as an augmented node feature.** Also can be used together with **any GNN**

C. Kanatsoulis, A. Ribeiro. Graph Neural Network Are More Powerful Than we Think, ICASSP 2024

Closed loops as node features

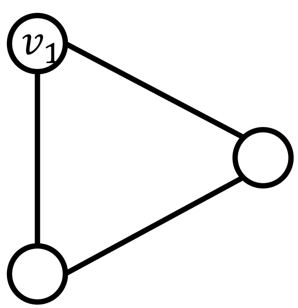
- We can also use the **diagonals of the adjacency powers** as augmented node features.
- They correspond to the **closed loops** each node is involved in.

$$C^{(0)} = \left[\operatorname{diag}\left(A^0\right), \operatorname{diag}\left(A^1\right), \operatorname{diag}\left(A^2\right), \operatorname{diag}\left(A^3\right), \ldots, \operatorname{diag}\left(A^{D-1}\right)\right] \in \mathbb{N}_0^{N \times D}$$

Augmented node feature for v_1

[1, 0, 2, 2, 6, 8]

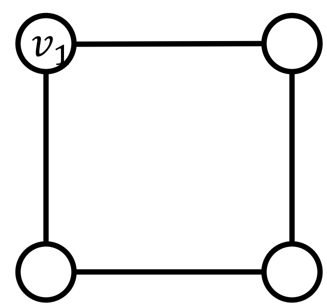
v_1 resides in a cycle with length 3



Augmented node feature for v_1

[1, 0, 2, 0, 8, 0]

v_1 resides in a cycle with length 4



Expressive Power

- **Theorem:** If two graphs have adjacency matrices with **different eigenvalues**, there exists a GNN with closed-loop initial node features that can **always tell them apart.**

- GNNs with **structural initial node features** can produce different representations for **almost all real-world graphs**.

- GIN with structural initial node features is **strictly more powerful** than the WL-kernel.

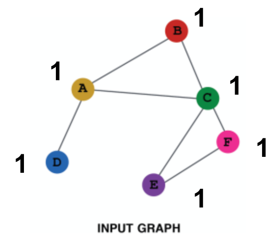
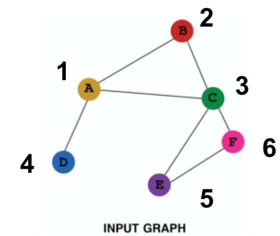
Feature Augmentation on Graphs

Why do we need feature augmentation?

- **(2) Certain structures are hard to learn by GNN**
- Other commonly used augmented features:
  - **Clustering coefficient**
  - **PageRank**
  - **Centrality**
  - **…**
- Any **feature we have introduced in Lecture 1 can be used!**

Feature Augmentation on Graphs

- Feature augmentation: **constant** vs. **Structure**

	Constant node feature 	**Structure-aware node feature** 
Expressive power	**Medium**. All the nodes are identical, but GNN can still learn from the graph structure	**High**. Each node has a structure-aware ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	**High**. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	**High**. Simple to generalize to new nodes: can count triangles or closed loops for any graph
Computational cost	**Low**. Only 1 dimensional feature	**Low/High**. Depending on the structures we are counting
Use cases	Any graph, inductive settings (generalize to new nodes)	Any graph, inductive settings (generalize new nodes)

Stanford CS224W: Position-aware Graph Neural Networks

CS224W: Machine Learning with Graphs

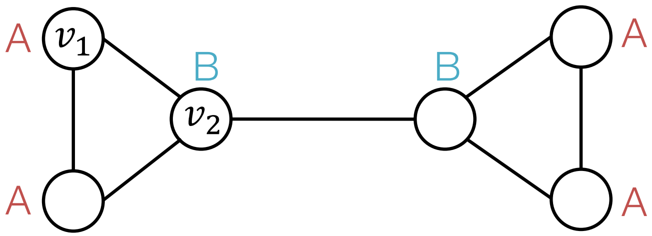
Jure Leskovec, Stanford University

http://cs224w.stanford.edu

Two Types of Tasks on Graphs

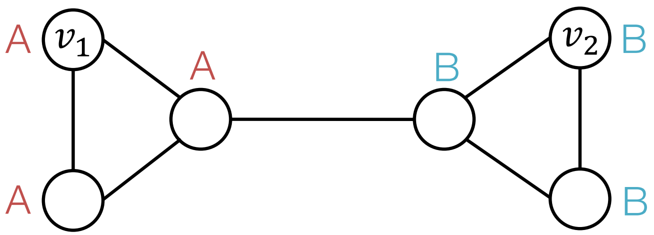
- **There are two types of tasks on graphs**

Structure-aware task



- Nodes are labeled by their **structural roles** in the graph

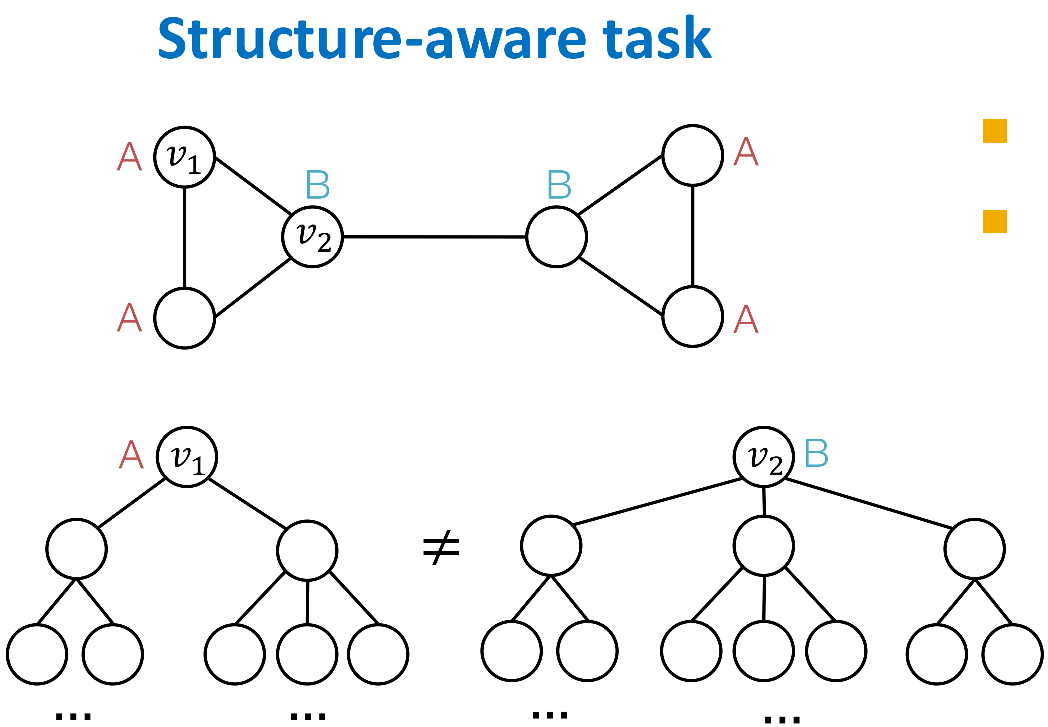
Position-aware task



- Nodes are labeled by their **positions** in the graph

Structure-aware Tasks

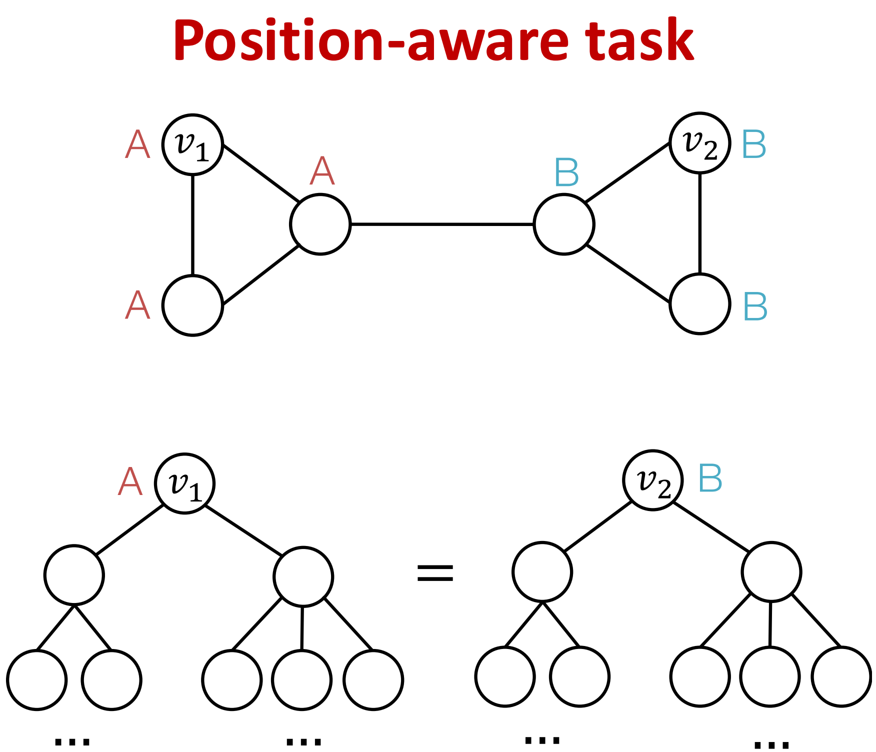
- **We showed how to design GNNs to work well for structure-aware tasks**



- GNNs work ☺
- Can differentiate v_1 and v_2 by using different computational graphs

Position-aware Tasks

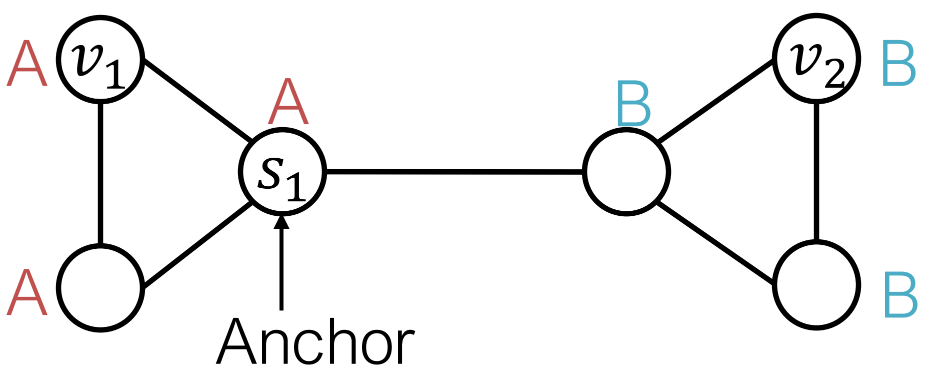
- **GNNs will always fail for position-aware tasks**



- GNNs fail ☹
- v_1 and v_2 will always have the same computational graph, **due to structure symmetry**
- **Can we define deep learning methods that are position-aware?**

Power of "Anchor"

- Randomly pick a node s_1 as an **anchor node**
- Represent v_1 and v_2 via their relative distances w.r.t. the anchor s_1, **which are different**
- An anchor node serves as **a coordinate axis**
  - Which can be used to **locate nodes in the graph**

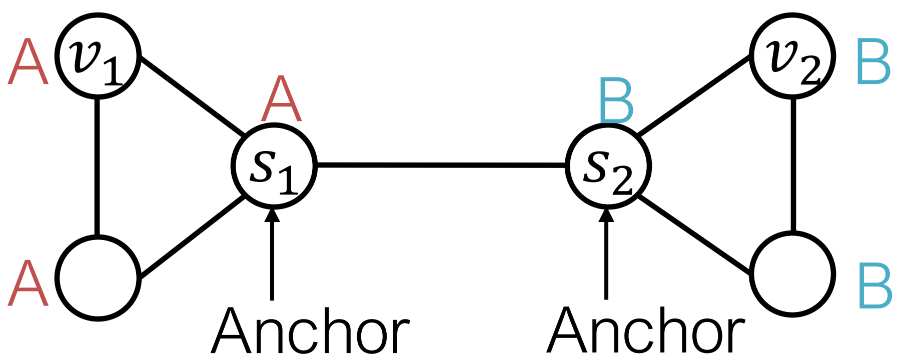


Relative Distances

	s_1
v_1	1
v_2	2

Power of "Anchors"

- Pick more nodes s_1, s_2 as **anchor nodes**
- **Observation:** More anchors can better characterize node position in different regions of the graph
- Many anchors –> Many coordinate axes

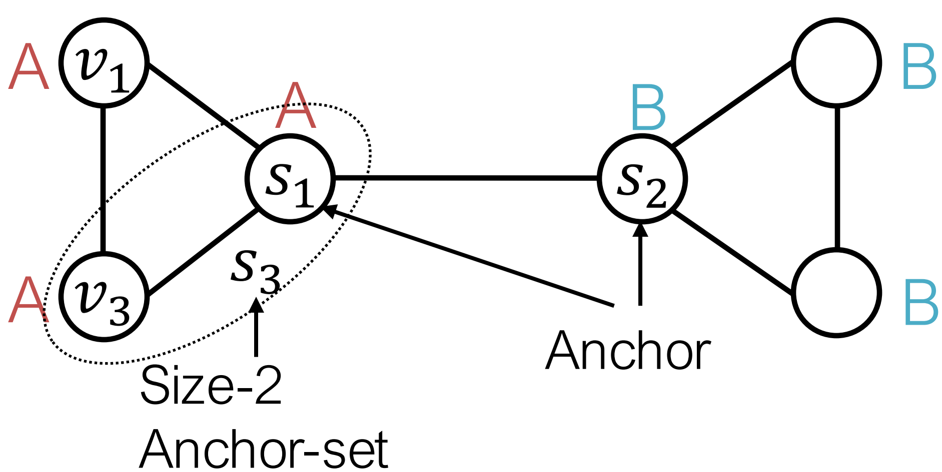


Relative Distances

	s_1	s_2
v_1	1	2
v_2	2	1

Power of "Anchor-sets"

- Generalize anchor from a single node to **a set of nodes**
  - We define distance to an anchor-set as the minimum distance to all the nodes in the ancho-set
- **Observation:** Large anchor-sets can sometimes provide more precise position estimate
  - We can save the total number of anchors



Relative Distances

	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

Anchor s_1, s_2 cannot differentiate node v_1, v_3, but anchor-set s_3 can

Anchor Set: Theory

- **Goal:** Embed the metric space (V, d) into the Euclidian space $\mathbb{R}^k$ such that the original distance metric is preserved.
  - For every node pairs $u, v \in V$, the Euclidian embedding distance $\|\boldsymbol{z}_u - \boldsymbol{z}_v\|_2$ is close to the original distance metric $d(u, v)$.

Anchor Set: Theory

- Bourgain Theorem [Informal] [Bourgain 1985]
  - Consider the following embedding function of node $v \in V$.

$$f(v) = \left(d_{\min}(v, S_{1,1}), d_{\min}(v, S_{1,2}), \dots, d_{\min}(v, S_{\log n, c\log n})\right) \in \mathbb{R}^{c \log^2 n}$$

  - where
    - c is a constant.
    - $S_{i,j} \subset V$ is chosen by including each node in V independently with probability $\frac{1}{2^i}$.
    - $d_{\min}(v, S_{i,j}) \equiv \min_{u \in S_{i,j}} d(v, u)$.
  - **The embedding distance produced by f is provably close to the original distance metric (V, d).**

Anchor Set: Theory

P-GNN follows the theory of Bourgain theorem

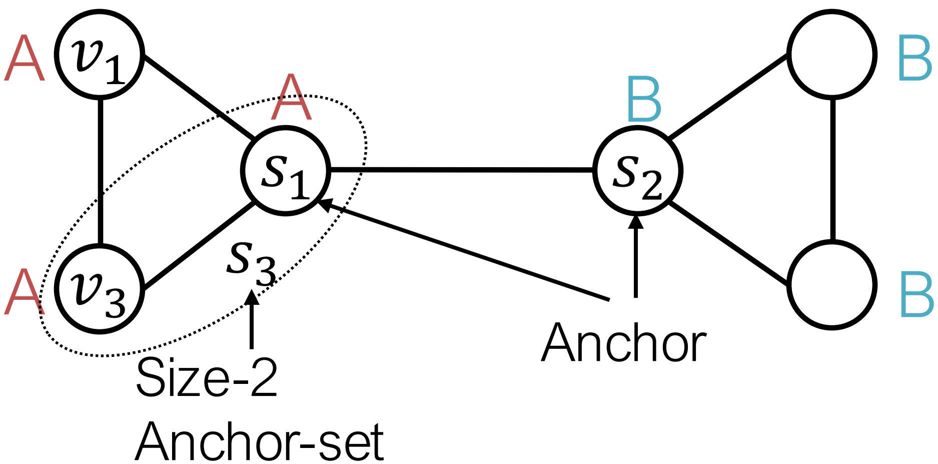
- First samples $O(\log^2 n)$ anchor sets $S_{i,j}$.
- Embed each node v via
  $\left(d_{\min}(v, S_{1,1}), d_{\min}(v, S_{1,2}), \dots, d_{\min}(v, S_{\log n, c\log n})\right) \in \mathbb{R}^{c \log^2 n}$.

P-GNN maintains the inductive capability

- During training, new anchor sets are *re-sampled* every time.
- P-GNN is learned to operate over the new anchor sets.
- At test time, given a new unseen graph, new anchor sets are sampled.

Position Information: Summary

- **Position encoding for graphs:** Represent a node's position by its distance to randomly selected anchor-sets
  - Each dimension of the position encoding is tied to an anchor-set



	s_1	s_2	s_3
v_1	1	2	1
v_3	1	2	0

v_1's Position encoding

v_3's Position encoding

How to Use Position Information

- **The simple way:** Use position encoding as an augmented node feature (works well in practice)
  - **Issue:** Since each dimension of position encoding is tied to a random anchor set, dimensions of positional encoding can be randomly permuted, without changing its meaning
  - Imagine you permute the input dimensions of a normal NN, the output will surely change

How to Use Position Information

- **The rigorous solution:** Requires a special NN that can maintain the **permutation invariant property of position encoding**
  - Permuting the input feature dimension will only result in the permutation of the output dimension, the value in each dimension won't change
  - Position-aware GNN paper has more details