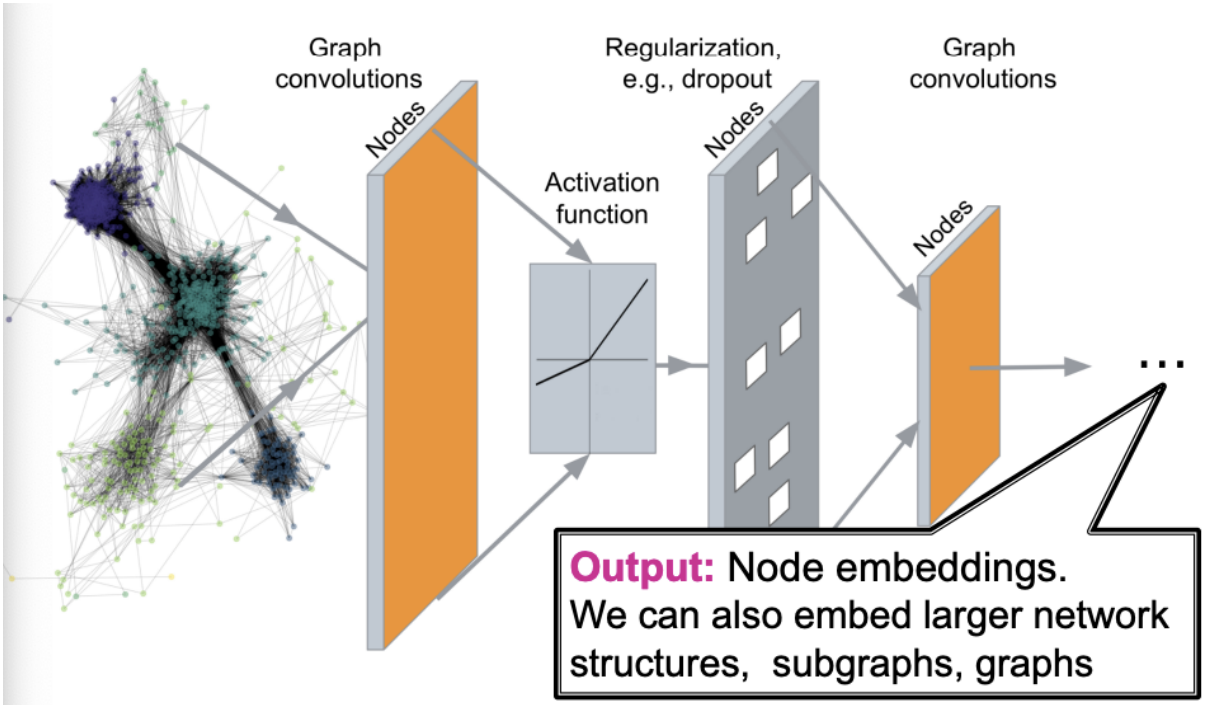
9. Theory of Graph Neural Networks

Contents

---

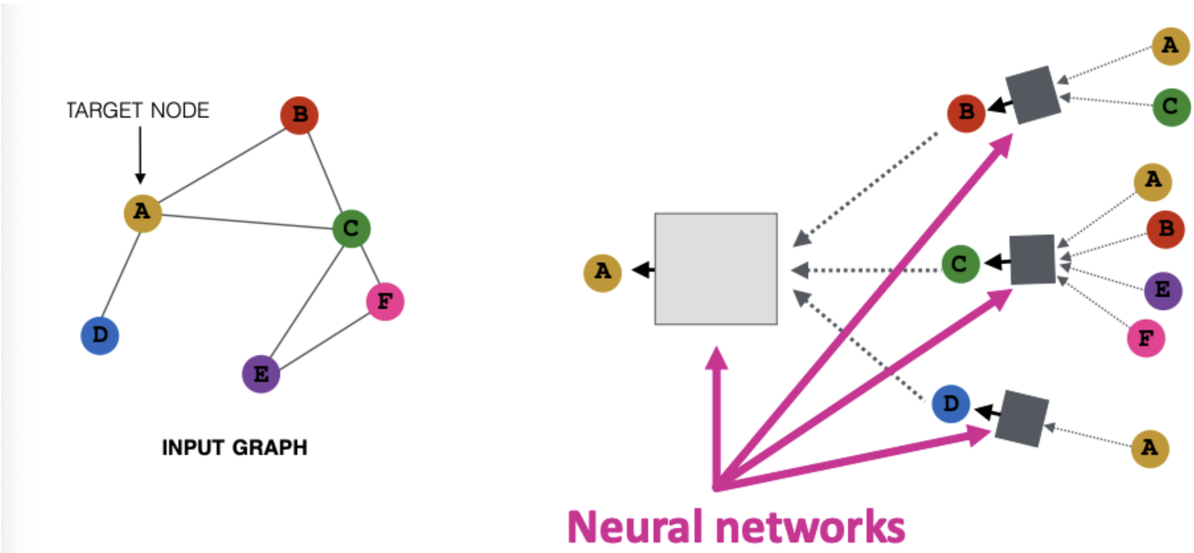
1. How Expressive are Graph Neural Networks?

Recap : Graph Neural Networks



Idea : Aggregate Neightbors

- **Key idea** : Generate node embeddings based on **local network neighborhoods**
- **Intuition** : Nodes aggregate information from their neighbors using neural networks



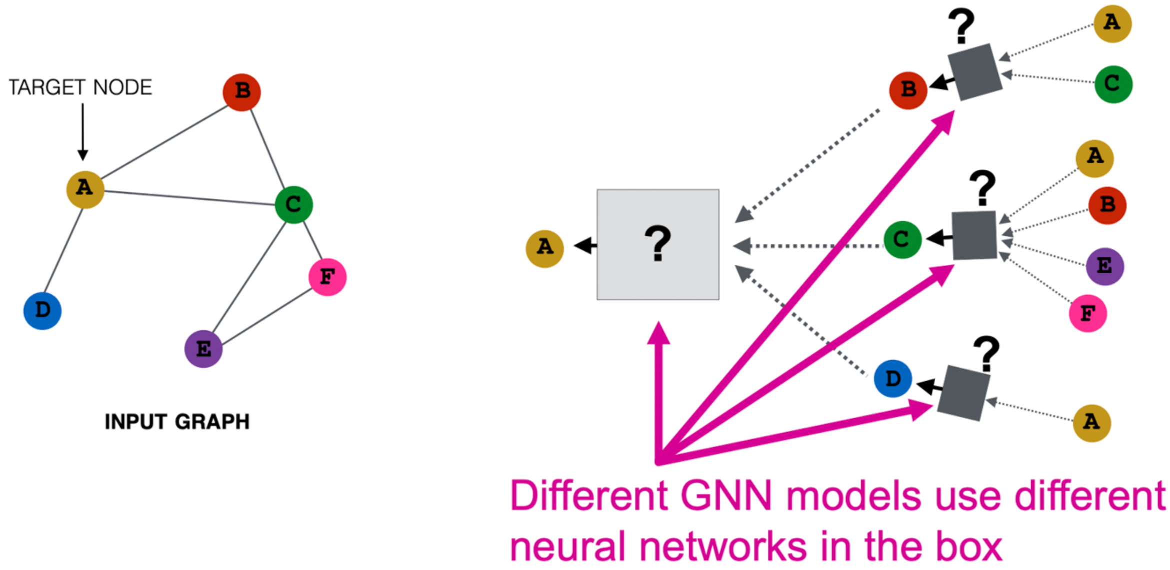
Theory of GNNs

How powerful are GNNs?

- **Many GNN models have been proposed (e.g., GCN, GAT, GraphSAGE, design space).**
- **What is the expressive power (ability to distinguish different graph structures) of these GNN models?**
- **How to design a maximally expressive GNN model?**

Background : Many GNN Models

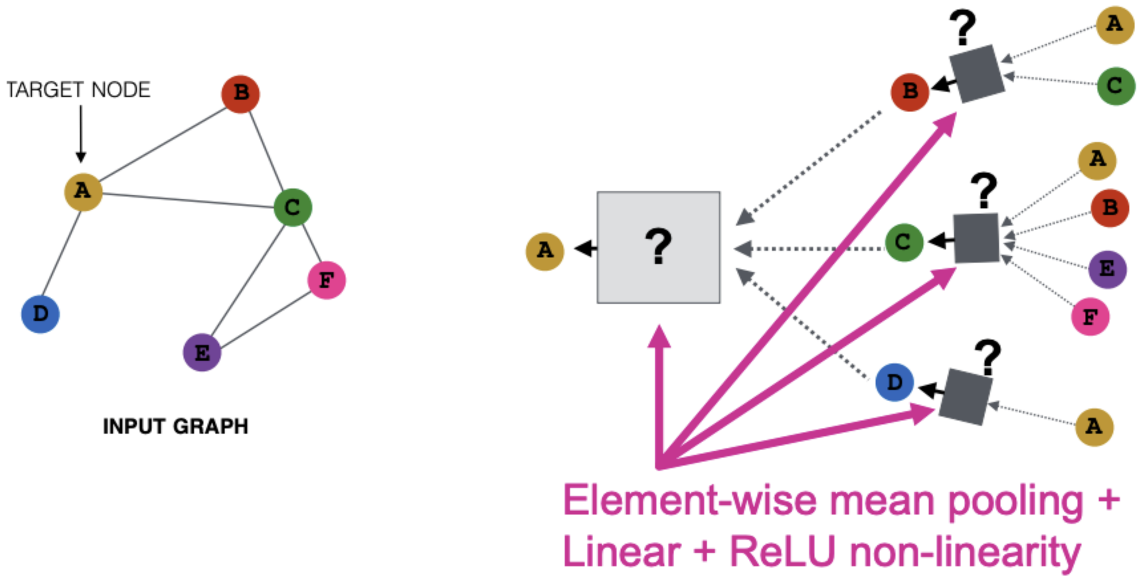
- **Many GNN models have been proposed :**
  - **GCN, GraphSAGE, GAT, Design Space etc.**



GNN Model Example (1)

- GCN (mean-pool) [Kipf and Welling ICLR 2017]

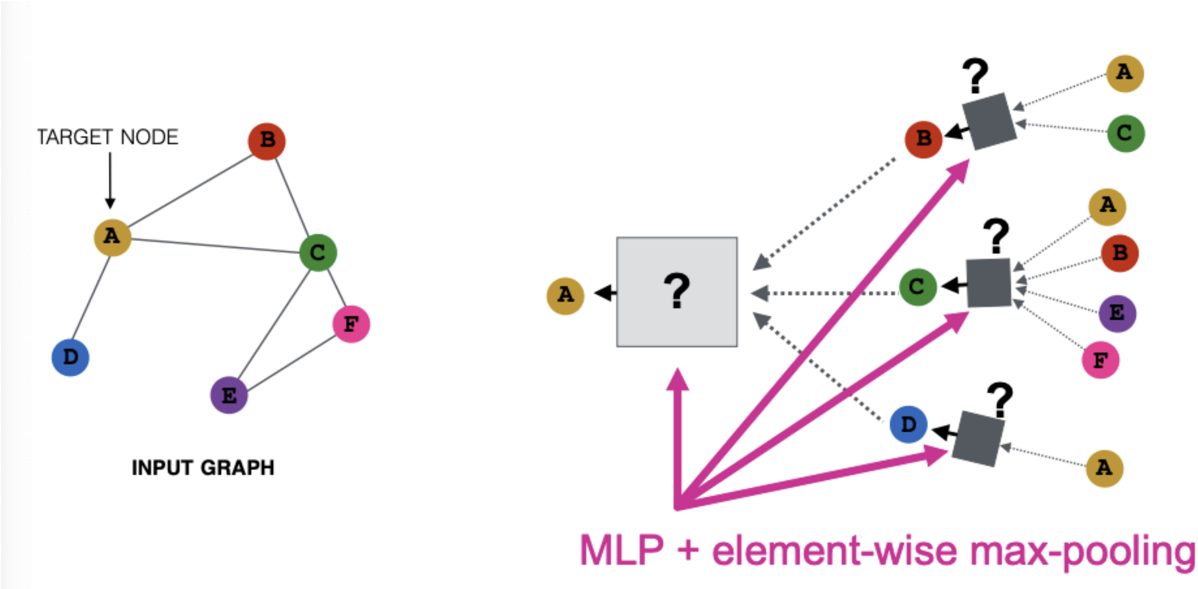
https://arxiv.org/abs/1609.02907



GNN Model Example (2)

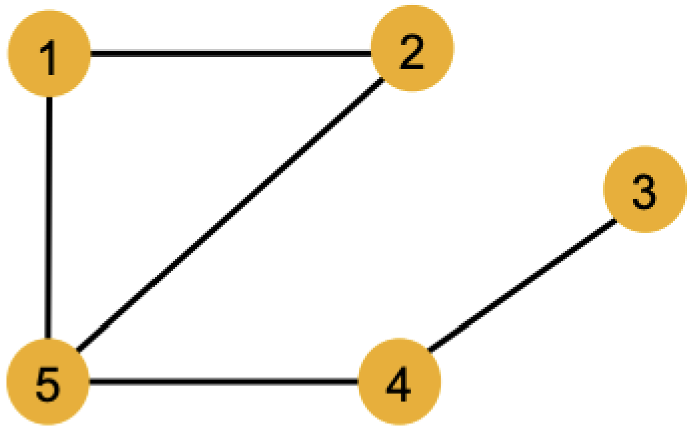
- GraphSAGE(max-pool) [Hamilton et al. NeurIps 2017]

https://arxiv.org/abs/1706.02216



Note : Node Colors

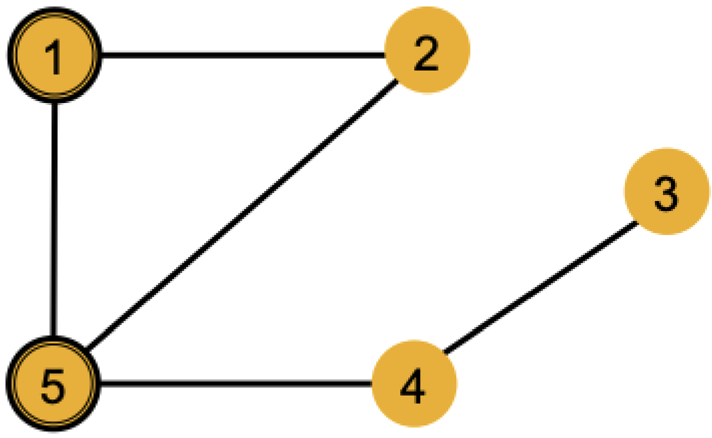
- We use node same/different **colors** to represent nodes with same/different features
    - For example, the graph below assumes all the nodes **share the same feature.**



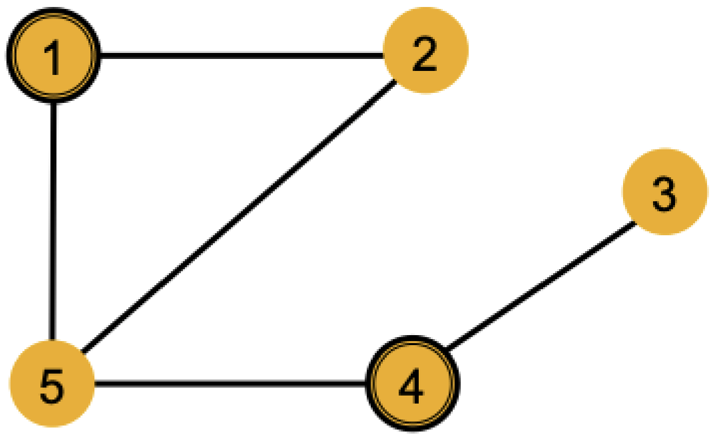
- Key question : **How well can a GNN distinguish different graph structures?**

Local Neighborhood Structures

- We specifically consider **local neighborhood structures** around each node in a graph.
  - Example : **Nodes 1 and 5 have different** neighborhood structures because they have different node degrees.



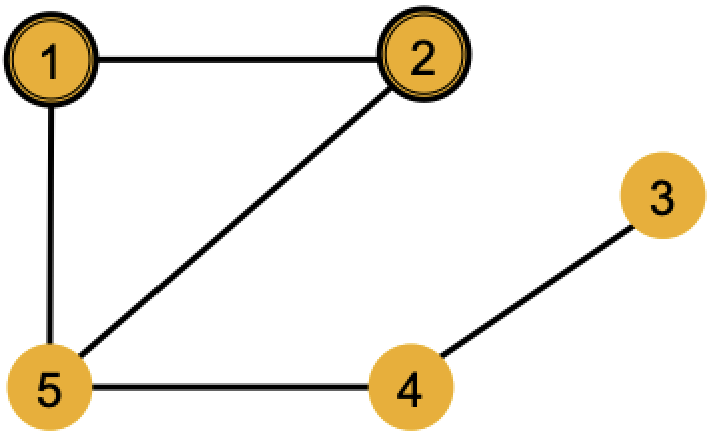
  - Example : **Nodes 1 and 4** both have the same node degree of 2. However, they still have **different** neighborhood structures because **their neighbors have different node degrees.**



→ Node 1 has neighbors of degrees 2 and 3.

→ Node 4 has neighbors of degrees 1 and 3.

  - Example : **Nodes 1 and 2** have the **same** neighborhood structure because **they are symmetric within the graph.**

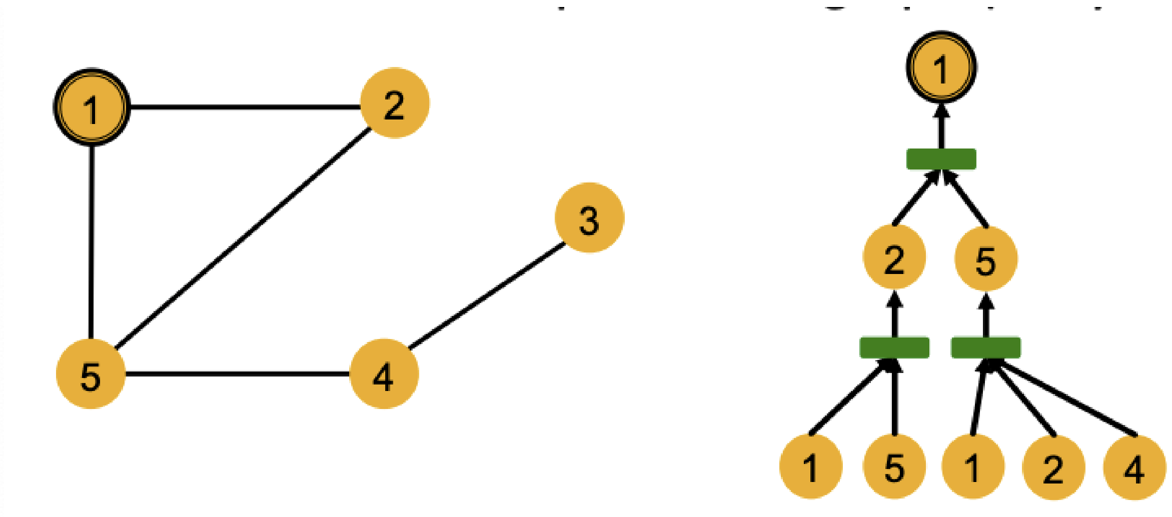


→ Node 1 has neighbors of degrees 2 and 3.
→ Node 2 has neighbors of degrees 2 and 3.
→ And even if we go a step deeper to 2^{nd} hop neighbors, both nodes have the same degrees (Node 4 of degree 2)

- **Key question : Can GNN node embeddings distinguish different node's local neighborhood structures?**
  - If so, When? If not, when will a GNN fail?
- **Next** : **We need to understand how a GNN captures local neighborhood structures.**
  - **Key concept : Computational graph**

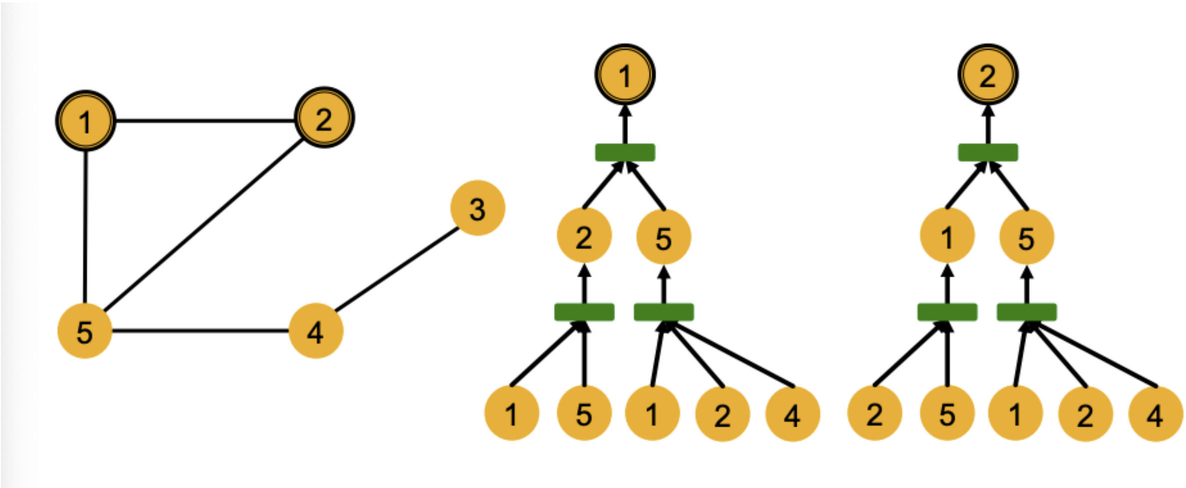
Computational Graph (1)

- **In each layer, a GNN aggregates neighboring node embeddings.**
- **A GNN generates node embeddings through a computational graph defined by the neighborhood.**
  - **Ex** : Node 1's computational graph (2-layer GNN)



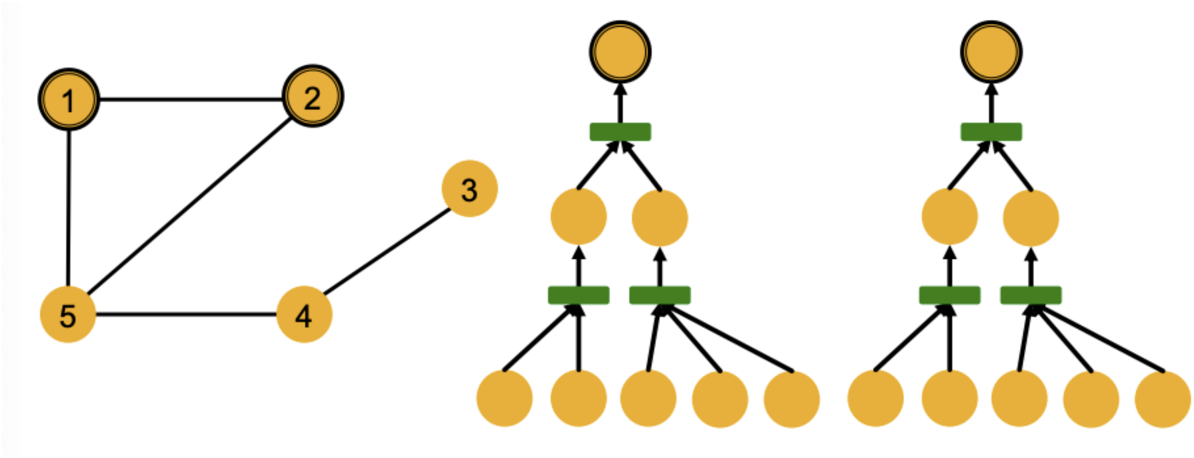
Computational Graph (2)

- **Ex** : Nodes 1 and 2's computational graphs

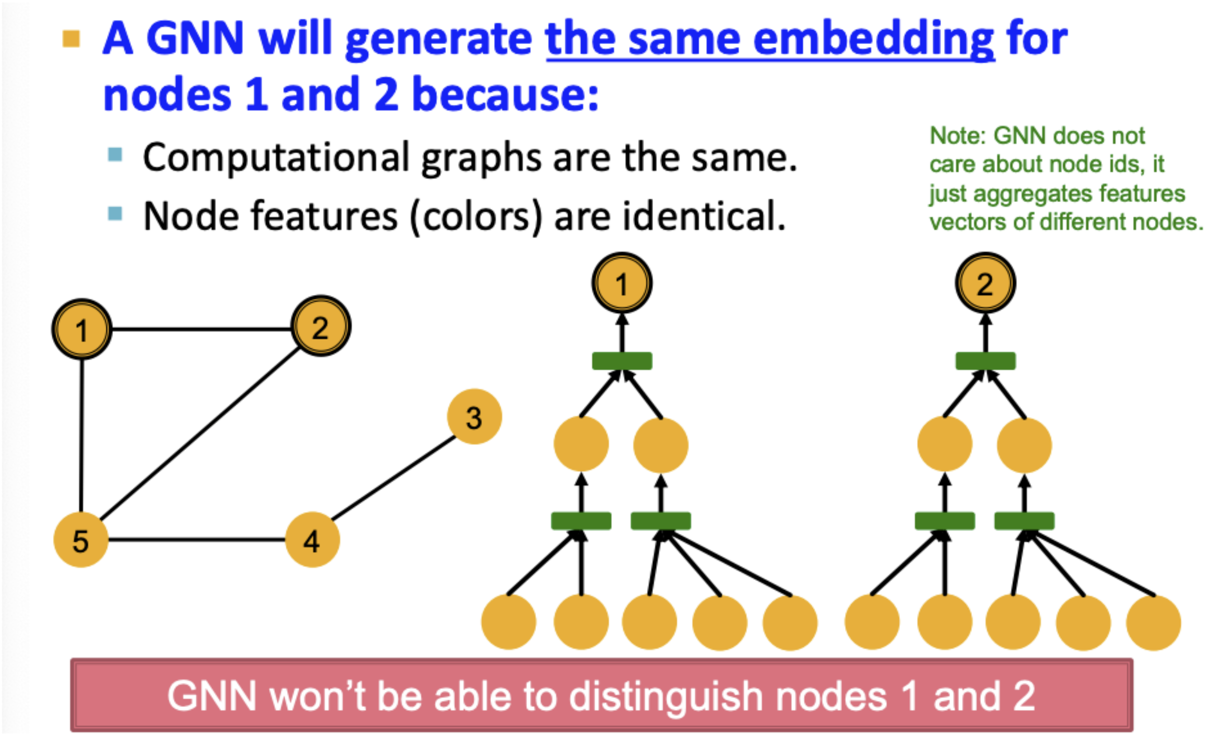


Computational Graph (3)

- **Ex** : Nodes 1 and 2's computational graphs.
- **But GNN only sees node features(Not IDs):**

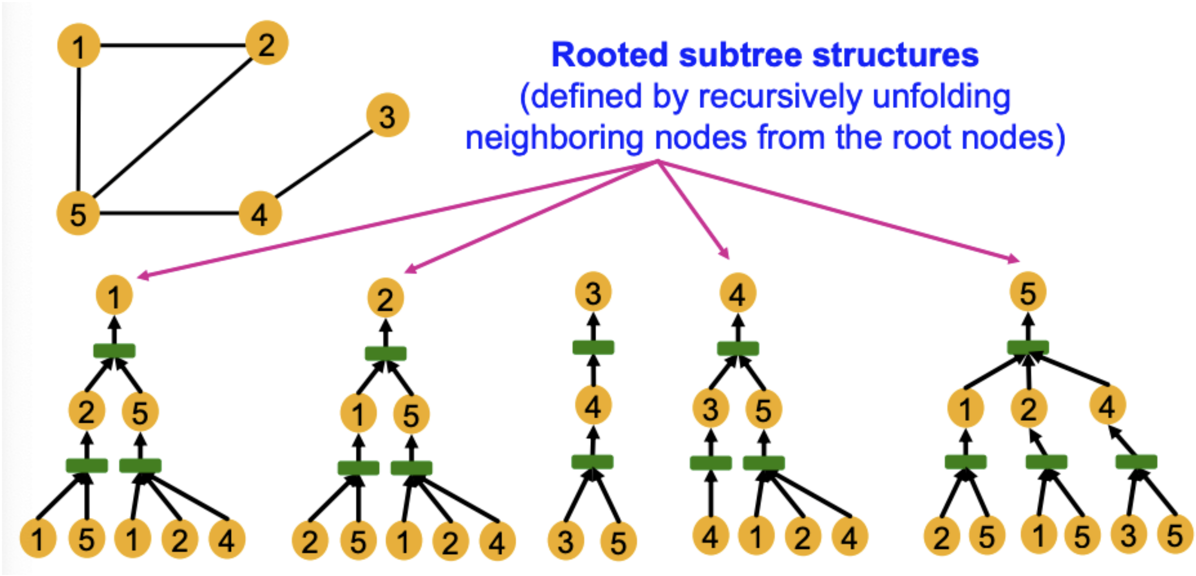


Computational Graph (4)

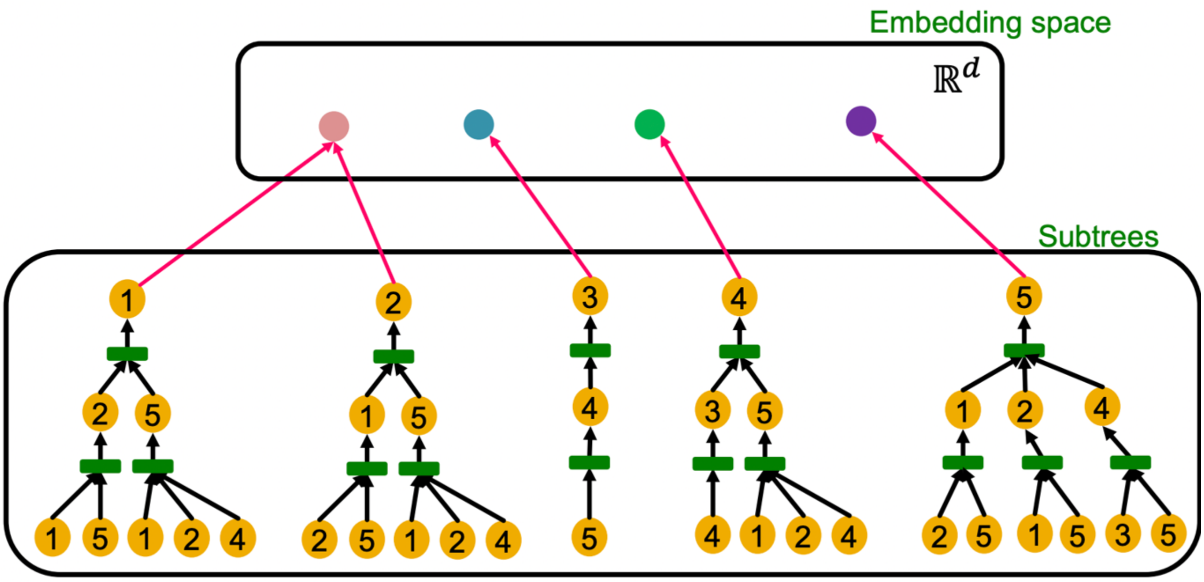


Computational Graph

- **In general, different local neighborhoods define different computational graphs**
- Computational graphs are identical to **rooted subtree structures** around each node.

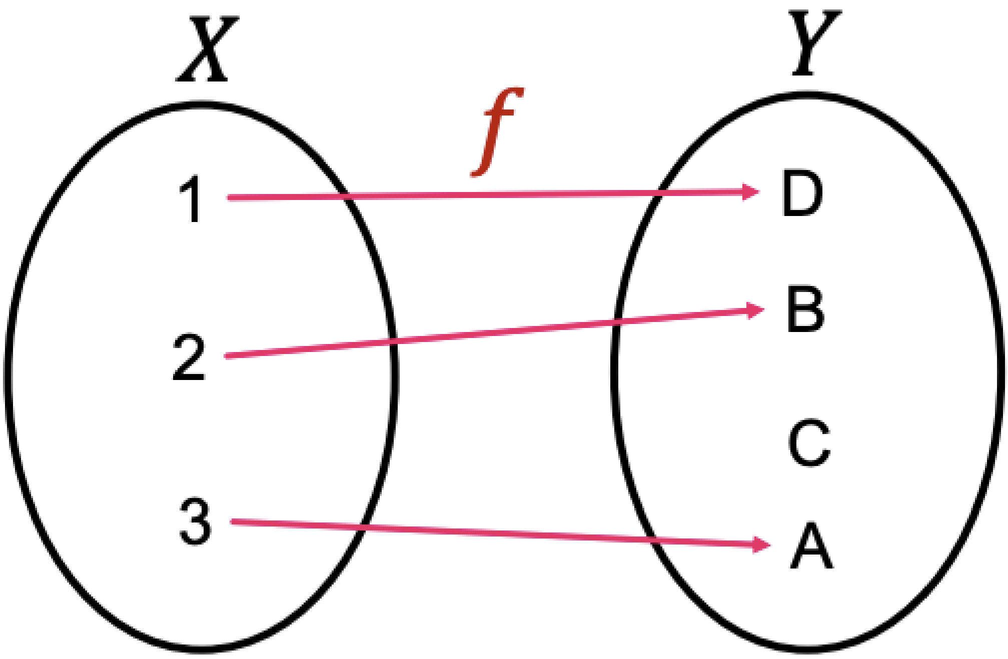


- GNN's node embeddings capture **rooted subtree structures.**
- Most expressive GNN maps different **rooted subtrees** into different node embeddings(represented by different colors).



Recall : Injective Function

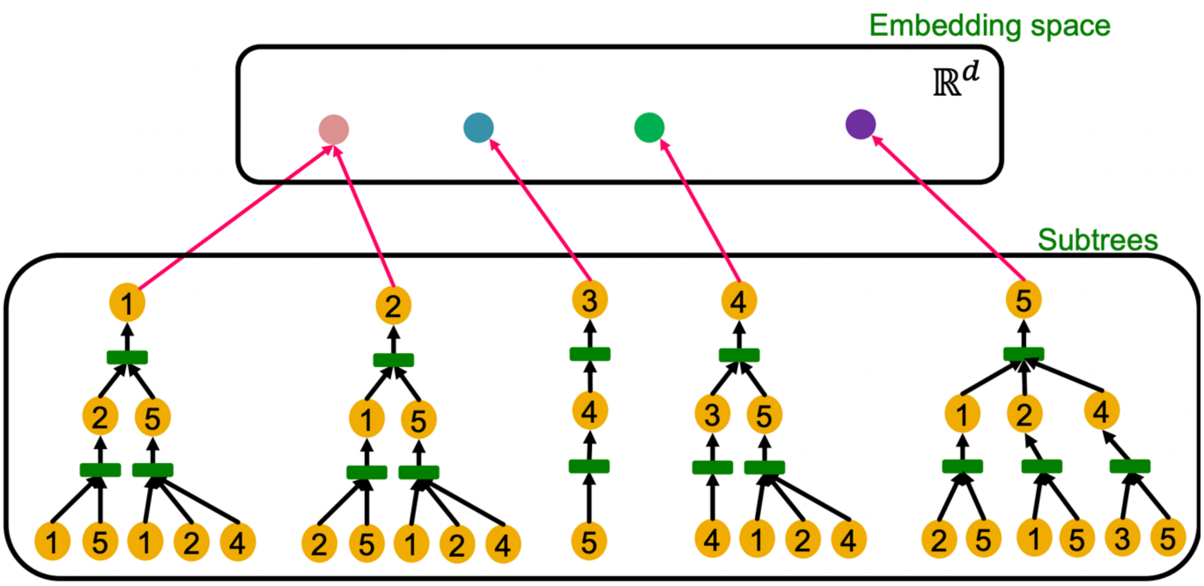
- **Function f : X → Y is injective if it maps different elements into different outputs.**
- Intuition : f **retains all the information about input.**



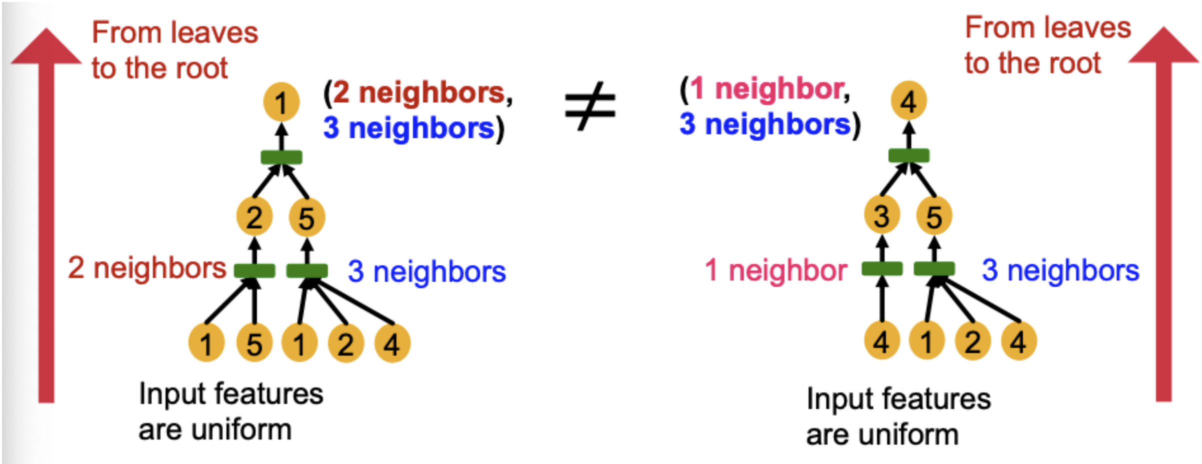
- 즉 injective하다는 뜻은 단사 함수(일대일 함수)란 뜻으로 정의역의 서로 다른 원소를 공역의 서로 다른 원소로 대응시키는 함수를 말한다.

How Expressive is a GNN?

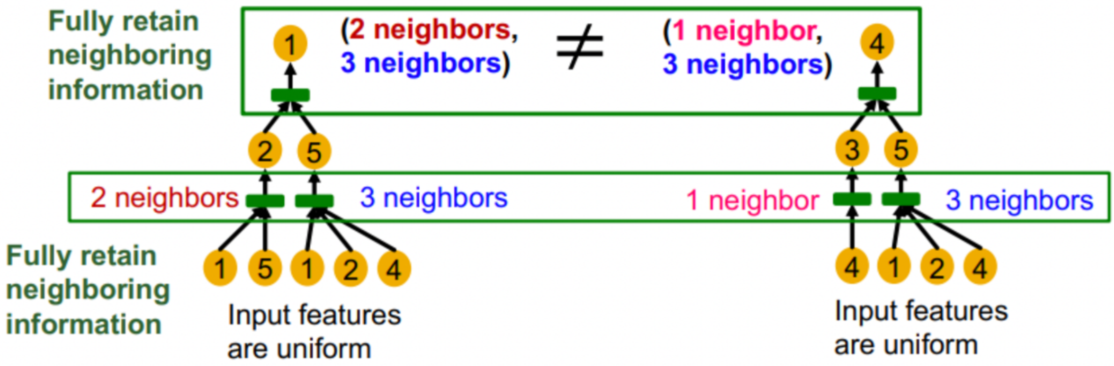
- Most expressive GNN should map subtrees to the node embeddings **injectively.**



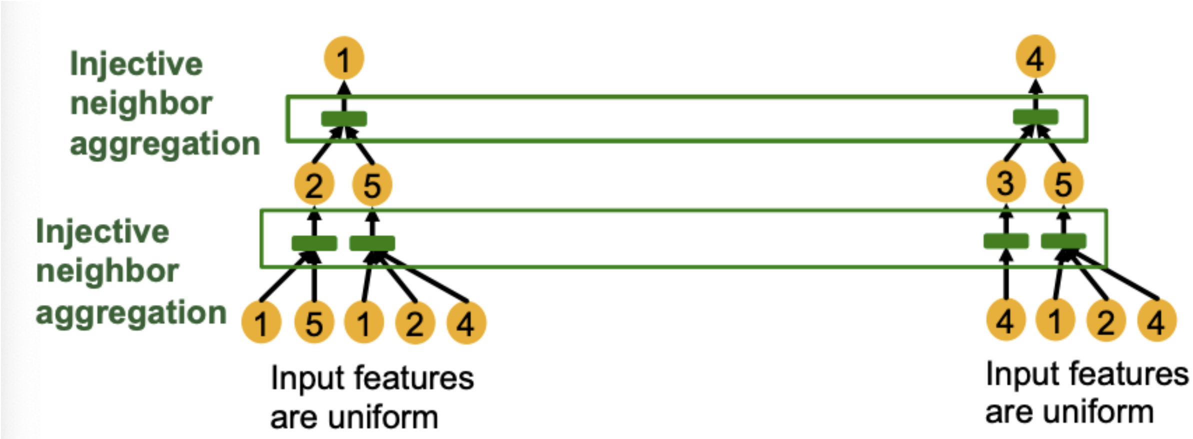
- **Key observation** : Subtrees of the same depth can be recursively characterized from the leaf nodes to the root nodes.



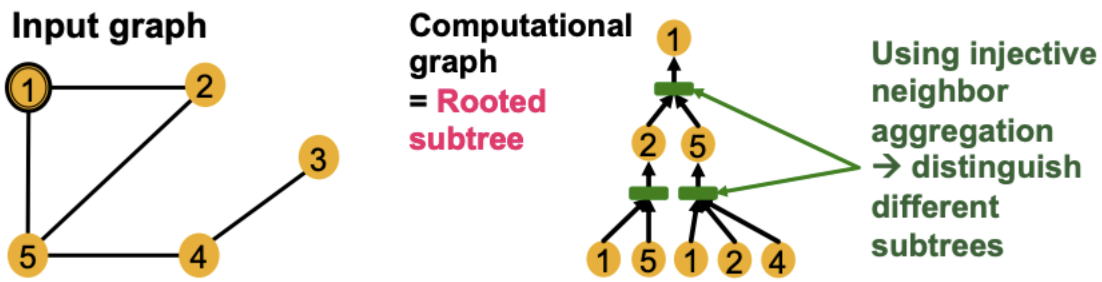
- If each step of GNN's aggregation **can fully retain the neighboring information**, the generated node embeddings can distinguish different rooted subtrees.



- In other words, most expressive GNN would use an **injective neighbor aggregation** function ant each step.
    - Maps different neighbors to different embeddings.



- Summary so far
  - To generate a node embedding, GNNs use a computational graph corresponding to a **subtree rooted around each node.**



  - GNN can fully distinguish different subtree structures if **every step of its neighbor aggregation is injective.**

---

2. Designing the Most Powerful Graph Neural Network

Expressive Power of GNNs

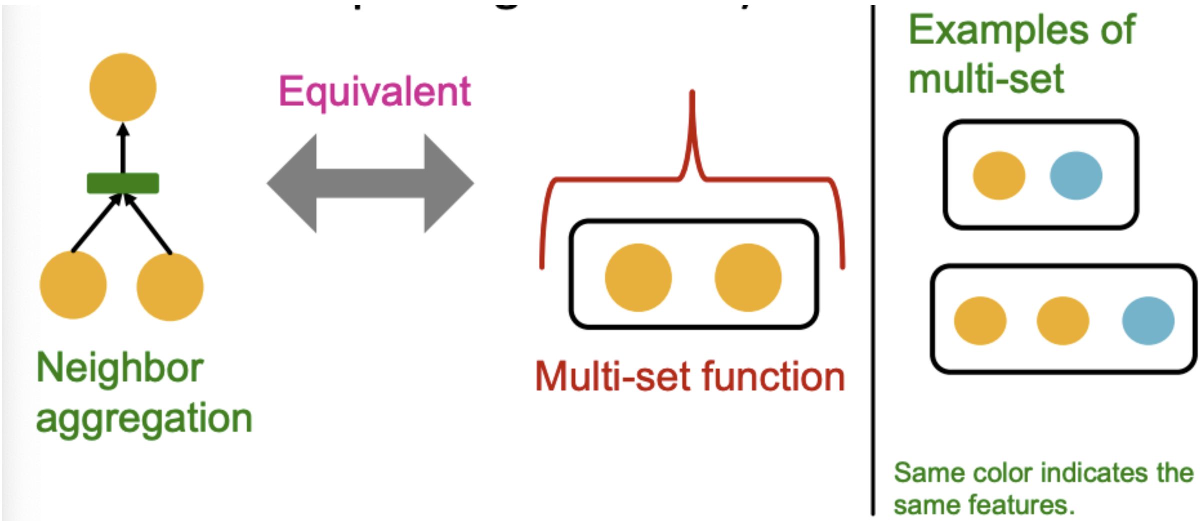
- **Key observation** **: Expressive power of GNNs can be characterized by that of neighbor aggregation functions they use.**
  - A more expressive aggregation function leads to a more expressive a GNN.

- **Injective aggregation function** leads to the most expressive GNN.

- **Next :**
  - Theoretically analyze expressive power of aggregation functions.

Neighbor Aggregation

- **Observation** : **Neighbor aggregation** can be abstracted as **a function over a multi-set**(a set with repeating elements).



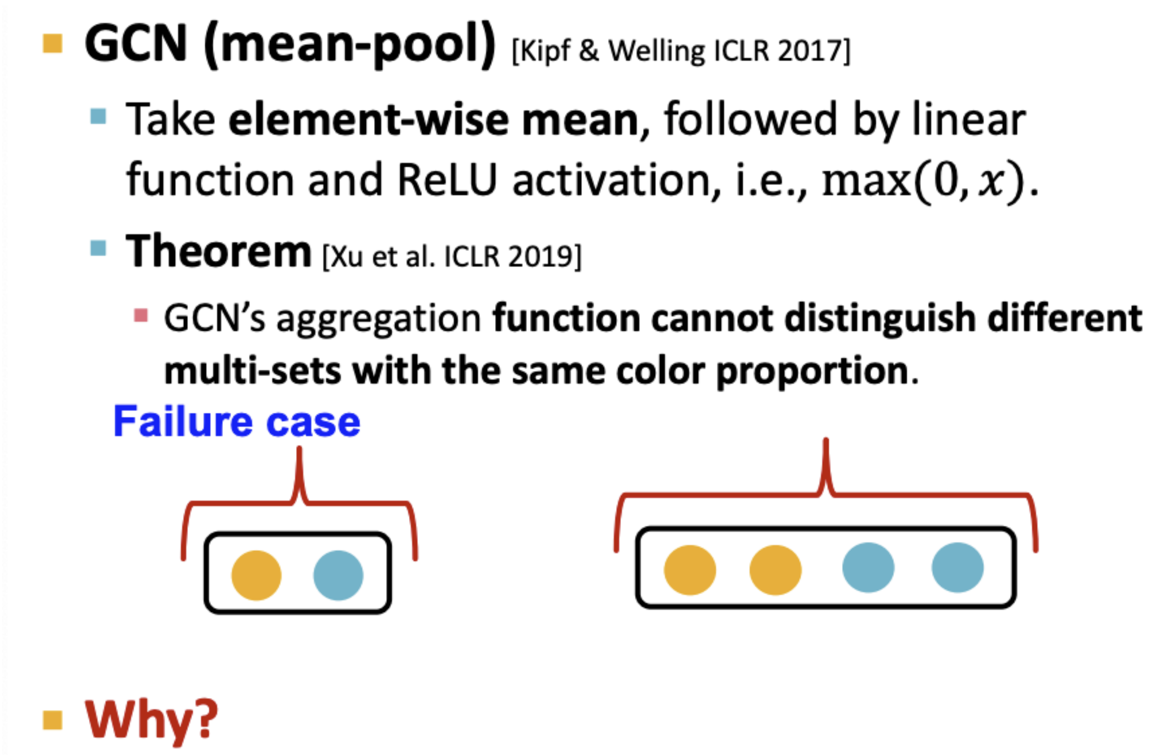
- **Next**: **We analyze aggregation functions of two popular GNN models**
  - **GCN** (mean-pool) [Kipf & Welling, ICLR 2017]
    - Uses **element-wise** mean pooling over neighboring node features

$$\mathrm{Mean}(\{x_u\}_{u\in N(v)})$$

  - **GraphSAGE** (max-pool) [Hamilton et al. NeurIPS 2017]
    - Uses **element-wise** max pooling over neighboring node features

$$\mathrm{Max}(\{x_u\}_{u\in N(v)})$$

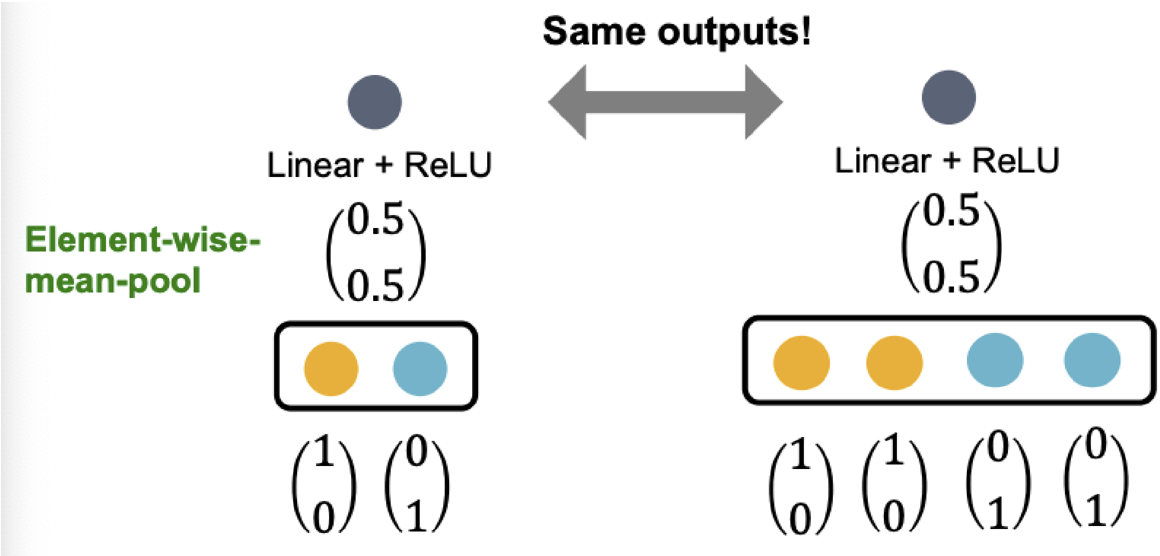
Neighbor Aggregation : Case Study

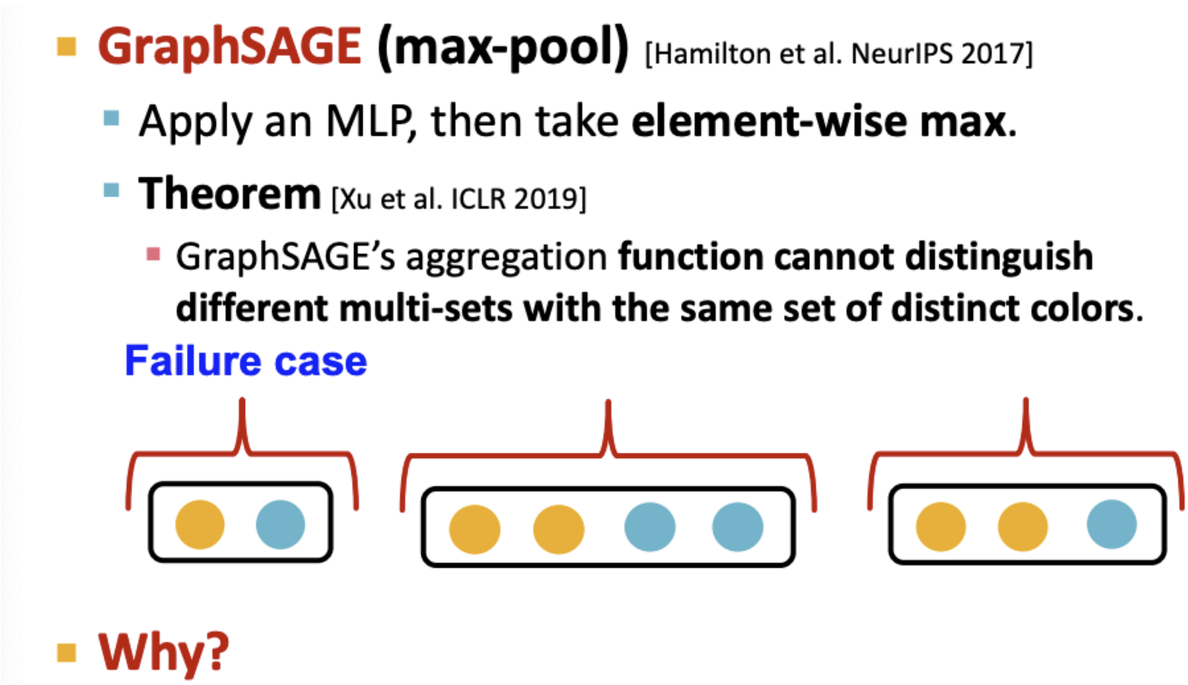


- For simplicity, we assume node colors are represented by one-hot encoding.
  - Example) If there are two distinct colors:

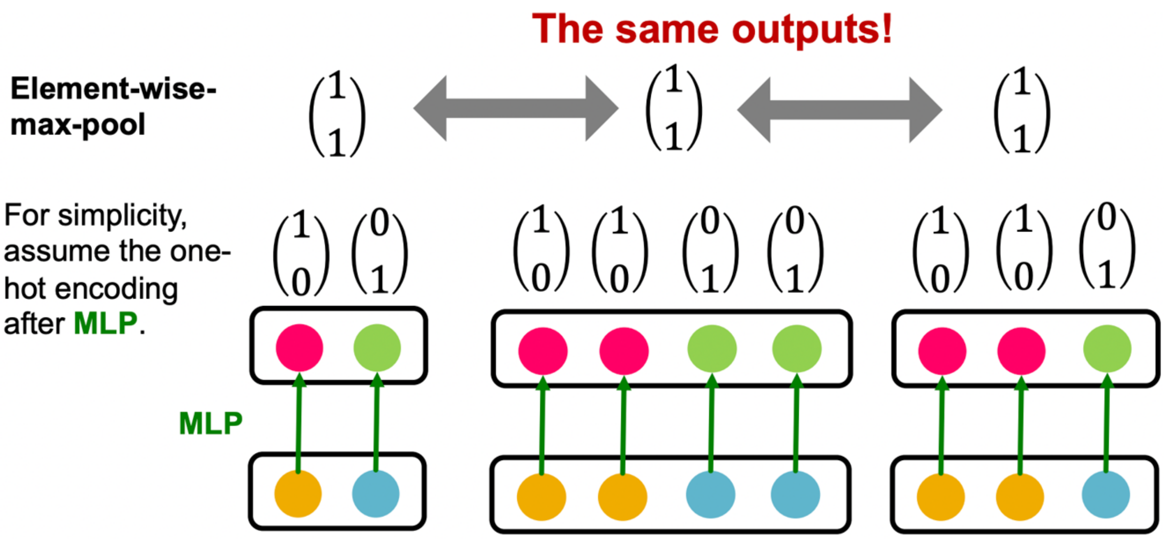
  - This assumption is sufficient to illustrate how GCN fails.

- Failure case illustration





- Failure case illustration



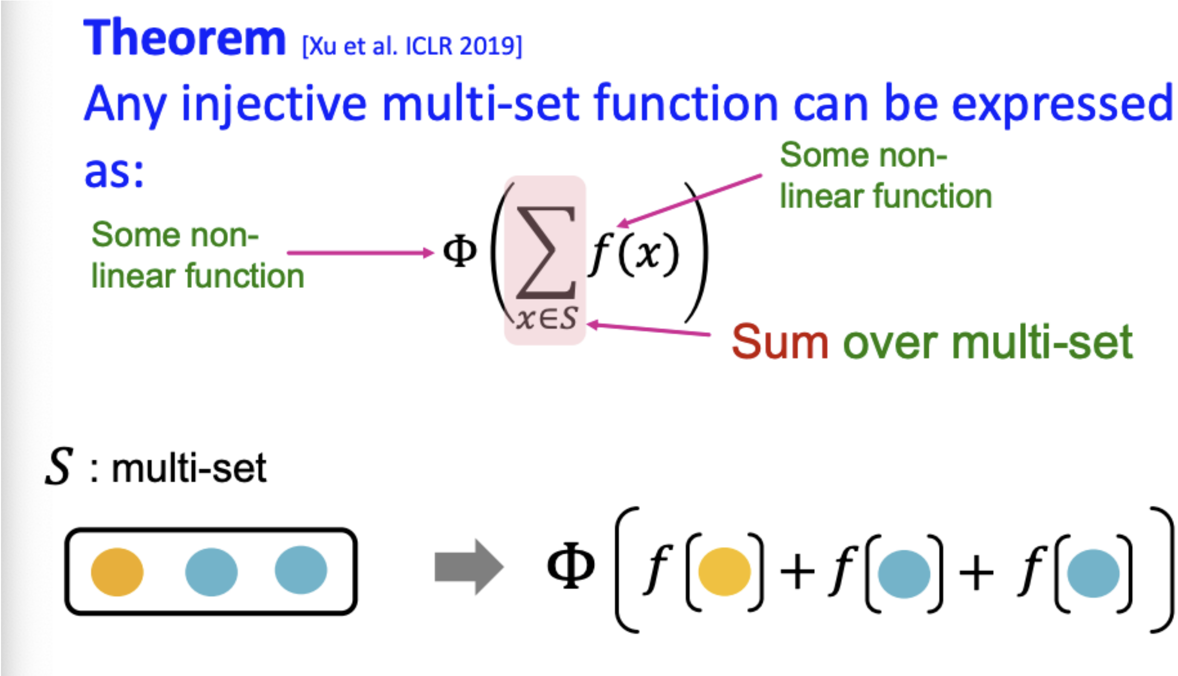
Summary So Far

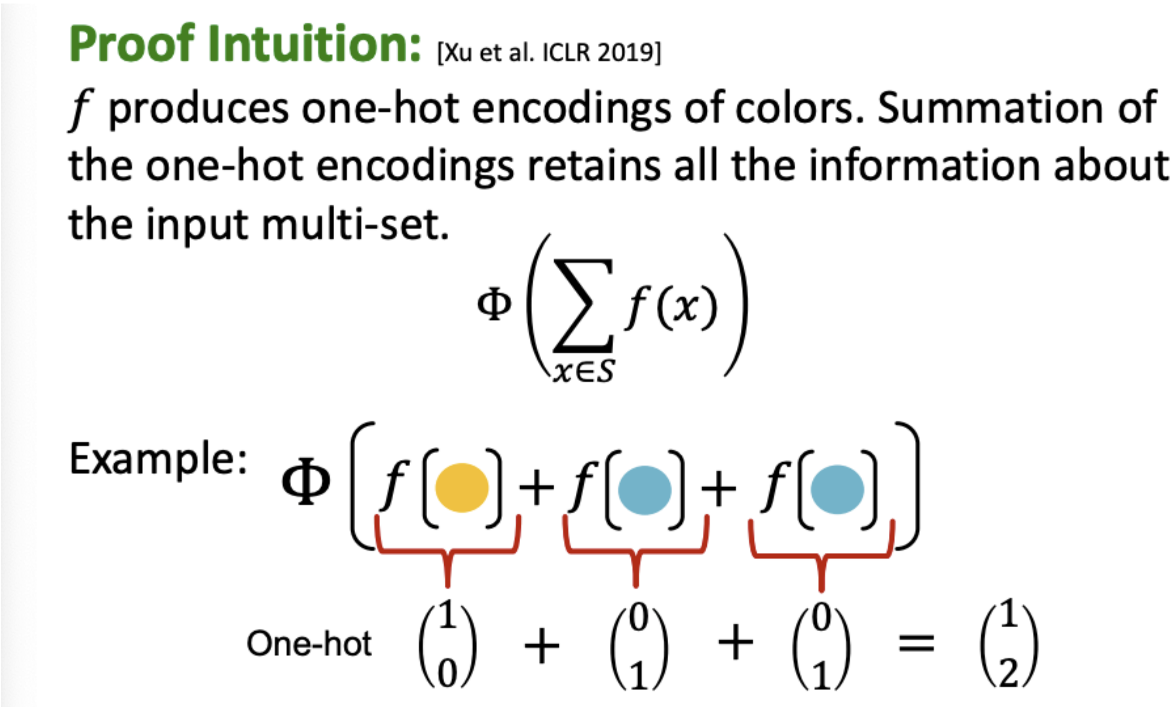
- We analyzed the **expressive power of GNNs.**
- **Main takeaways:**
  - Expressive power of GNNs can be characterized by that of the neighbor aggregation function.
  - Neighbor aggregation is a function over multi-sets(sets with repeating elements)
  - GCN and GraphSAGE's aggregation functions fail to distinguish some basic multi-sets; hence **not injective.**
  - Therefore, GCN and GraphSAGE are **not** maximally powerful GNNs.

Designing Most Expressive GNNs

- Our goal **: Design maximally powerful GNNs in the class of message-passing GNNs.**
- This can be achieved by designing injective **neighbor aggregation function over multi-sets.**
- **Here, we design a** neural network **that can model** injective **multiset function.**

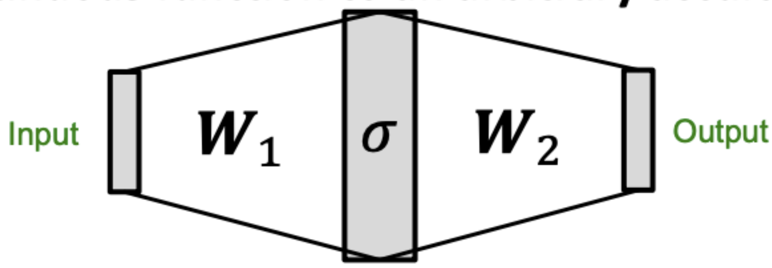
Injective Multi-Set Function





Universal Approximation Theorem

- **How to model $\boldsymbol{\Phi}$ and $\boldsymbol{f}$ in $\boldsymbol{\Phi}\left(\sum_{x\in S} \boldsymbol{f}(\boldsymbol{x})\right)$?**
- We use a Multi-Layer Perceptron (MLP).
- **Theorem: Universal Approximation Theorem** [Hornik et al., 1989]
  - 1-hidden-layer MLP with sufficiently-large hidden dimensionality and appropriate non-linearity $\sigma(\cdot)$ (including ReLU and sigmoid) can **approximate any continuous function to an arbitrary accuracy**.



Most Expressive GNN

- **Graph Isomorphism Network (GIN)** [Xu et al. ICLR 2019]
  - Apply an MLP, element-wise **sum**, followed by another MLP.

$$\mathrm{MLP}_{\Phi}\left(\sum_{x\in S} \mathrm{MLP}_{f}(x)\right)$$

- **Theorem** [Xu et al. ICLR 2019]
  - GIN's neighbor aggregation function is injective.
- **No failure cases!**
- **GIN is THE most expressive GNN in the class of message-passing GNNs!**

Full Model of GIN

- **So far : We have described the neighbor aggregation part of GIN.**
- We now describe the full model of GIN by relating it to **WL graph Kernel** (traditional way of obtaining graph-level features).
    - **We will see how GIN is a "neural network" version of the WL graph Kernel.**

Relation to WL Graph Kernel

Recall : Color refinement algorithm in WL kernel.

- **Given : A graph G with a set of nodes V.**
    - Assign an initial color $c^{(0)}(v)$ to each node v.
    - Iteratively refine node colors by

$$\boxed{c^{(k+1)}(v) = \text{HASH}\left(c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right),}$$

where HASH maps different inputs to different colors.

    - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood

The Complete GIN Model

- GIN uses a neural network to model the injective HASH function.

$$\boxed{c^{(k+1)}(v) = \text{HASH}\left(c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right)}$$

- Specifically, we will model the injective function over the tuple:

$$\left(\boxed{c^{(k)}(v)}, \boxed{\left\{c^{(k)}(u)\right\}_{u\in N(v)}}\right)$$

Root node features

Neighboring node colors

Theorem (Xu et al. ICLR 2019)

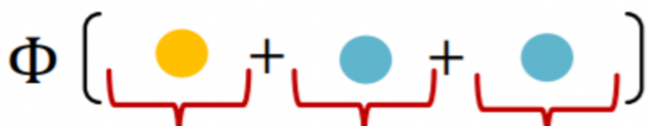
Any injective function over the tuple

Root node feature $\left(\boxed{c^{(k)}(v)}, \boxed{\left\{c^{(k)}(u)\right\}_{u\in N(v)}}\right)$ Neighboring node features

can be modeled as

$$\mathrm{MLP}_{\Phi}\left((1+\epsilon)\cdot \mathrm{MLP}_f(c^{(k)}(v)) + \sum_{u\in N(v)} \mathrm{MLP}_f(c^{(k)}(u))\right)$$

where ϵ is a learnable scalar.

Example: 

$$\begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}1\\2\end{pmatrix}$$

- We only need Φ to ensure the injectivity.

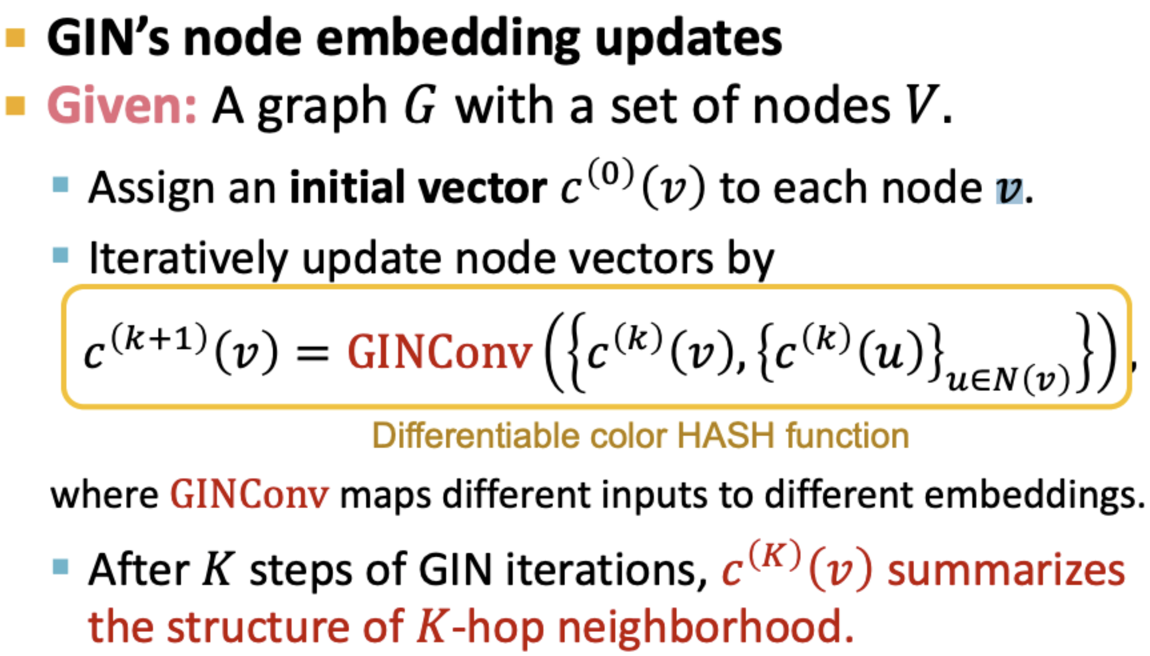
$$\mathrm{GINConv}\left(\boxed{c^{(k)}(v)}, \boxed{\left\{c^{(k)}(u)\right\}_{u\in N(v)}}\right) = \mathrm{MLP}_{\Phi}\left((1+\epsilon)\cdot c^{(k)}(v) + \sum_{u\in N(v)} c^{(k)}(u)\right)$$

Root node features

Neighboring node features

This MLP can provide "one-hot" input feature for the next layer.

- **WL Kernel**은 모든 노드에 동일한 **초깃값을 설정한 뒤 이웃 노드의 정보를 aggregation하여 반복적으로 해쉬 테이블을 업데이트** 하는 방식이다. <Lecture2>
- 이는 단사함수이며 **GIN은 해쉬 테이블을 MIP로 변형**하여 사용하는 것과 동일하다.
- MLP_ϕ는 자기 자신과 이웃 노드의 벡터를 종합하여 다음 레이어로 전달하며 input과 동일한 형태(one-hot)로 만들어주어 레이어를 거듭해도 정보가 보존될 수 있도록 만들어준다.
- GIN은 저차원 벡터로 구성되어 코사인 유사도 등을 통해 유사도를 계산하기 용이하며, MLP가 학습가능한 파라미터로 구성되어 있기 때문에 downstream task에 맞춰 fine tuning 할 수 있다는 장점이 있다.



GIN and WL Graph Kernel

- **GIN can be understood as differentiable neural version of the WL graph Kernel:**

	Update target	Update function
WL Graph Kernel	Node colors (one-hot)	HASH
GIN	Node embeddings (low-dim vectors)	GINConv

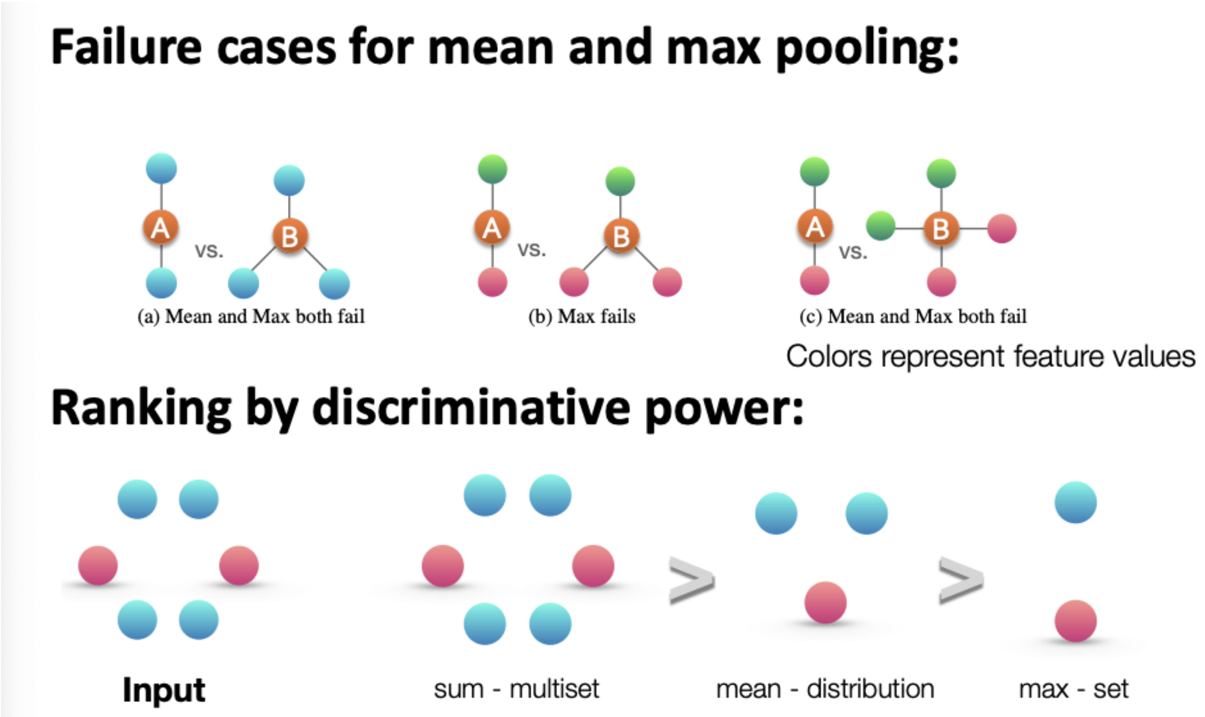
Expressive Power of GIN

- **Because of the relation between GIN and the WL graph kernel, their expressive is exactly the same.**
    - If two graphs can be distinguished by GIN, they can be also distinguished by the WL kernel, and vice versa.
- How powerful is this?
    - WL kernel has been both theoretically and empirically shown to distinguish most of the real world graphs [Cai et al. 1992].
    - Hence, GIN is also powerful enough to distinguish most of the real graphs!

Summary of the Lecture

- We design a neural network that can model **injective multi-set function.**
- We use the neural network for neighbor aggregation function and arrive at **GIN---the**
  most expressive GNN model.
- The key is to use **element-wise sum pooling**, instead of mean-/max-pooling.
- GIN is closely related to the WL graph kernel.
- Both GIN and WL graph kernel can distinguish most of the real graphs!

The Power of Pooling



Imporving GNN’s Power