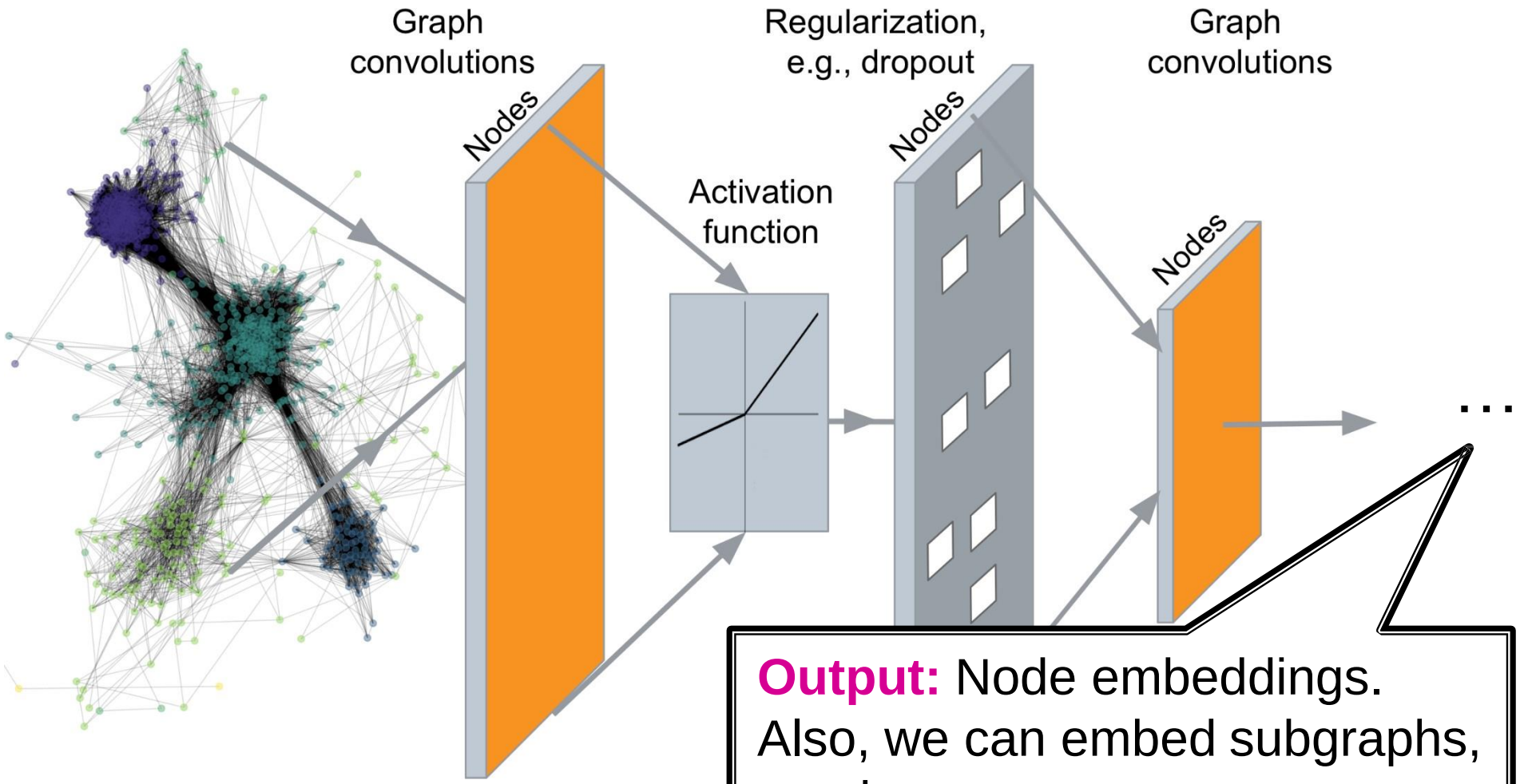
COMP4222 Machine Learning with Structured Data

Graph Isomorphism Network

Instructor: Yangqiu Song

Slides credits: Jure Laskovec

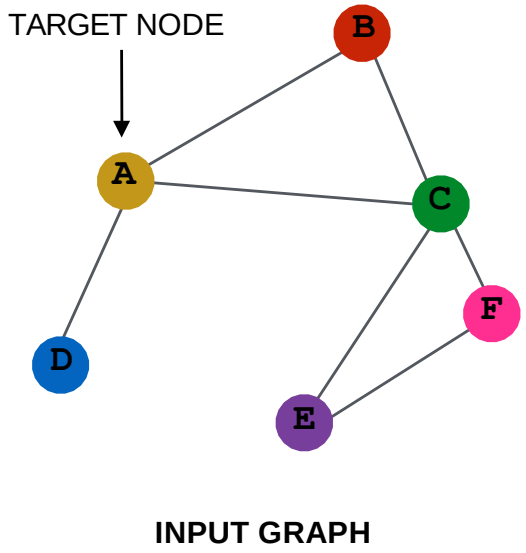
Recap: Deep Graph Encoders

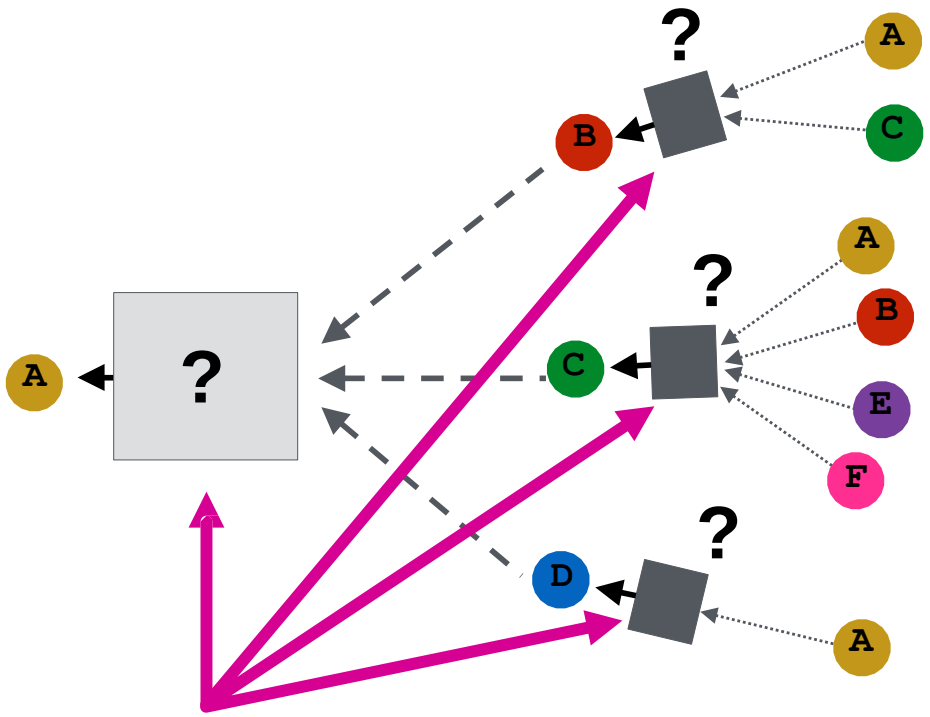


Output: Node embeddings.
Also, we can embed subgraphs, graphs

Recap: Aggregate from Neighbors

- **Intuition:** Nodes aggregate information from their neighbors using neural networks





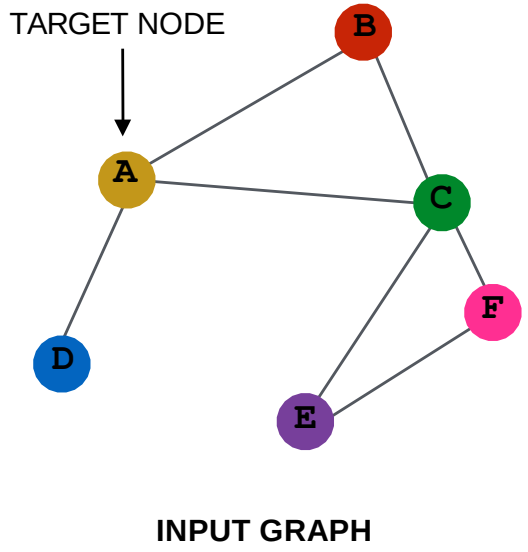
Neural networks

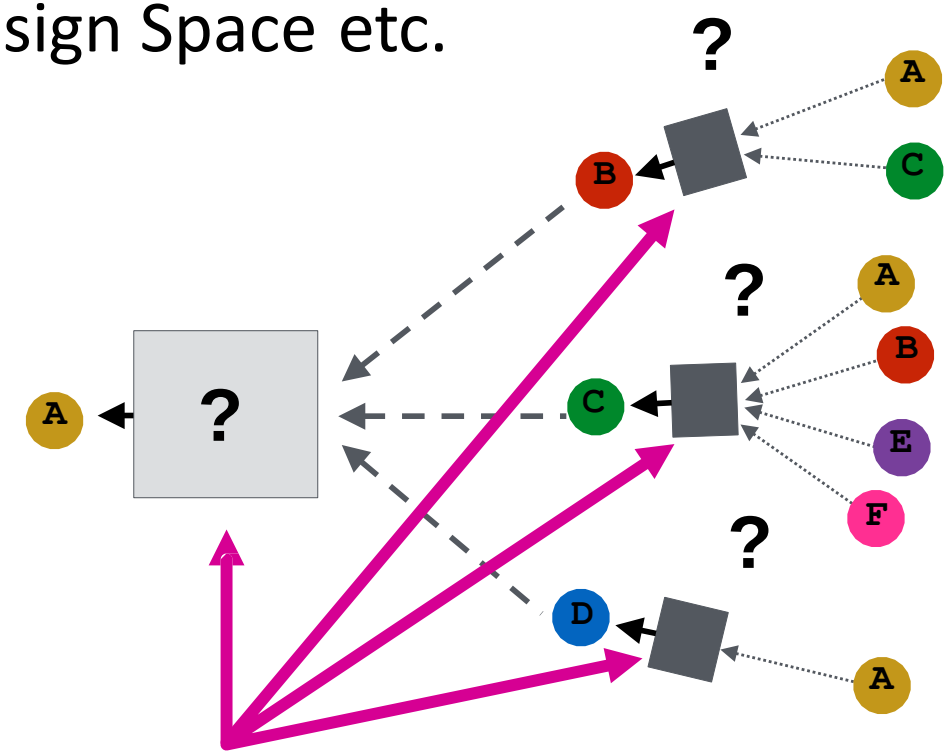
How powerful are GNNs?

- 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?

Many GNN Models

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

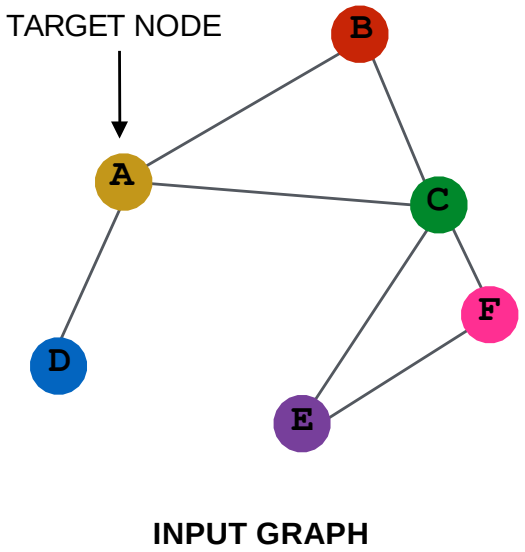


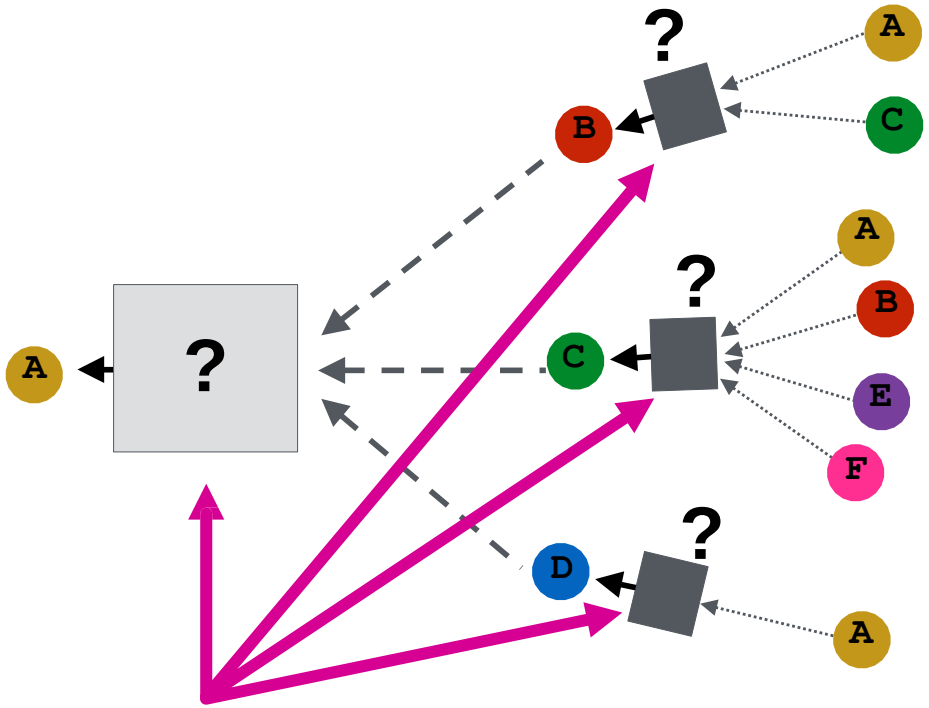


Different GNN models use different neural networks in the box

GNN Model Example

- **GCN (mean-pool)**

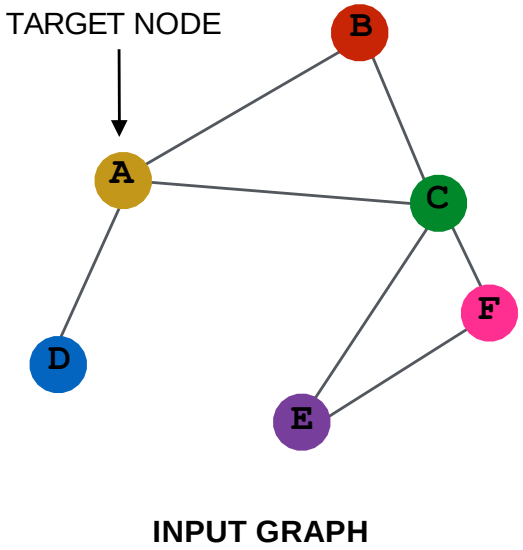


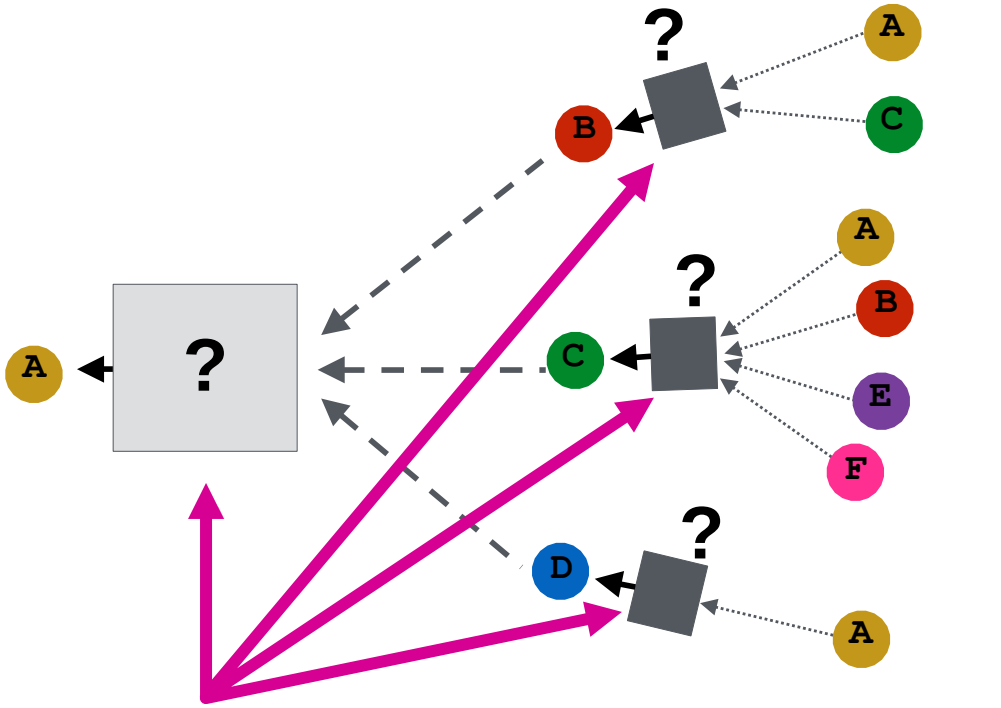


Element-wise mean pooling +
Linear + ReLU non-linearity

GNN Model Example

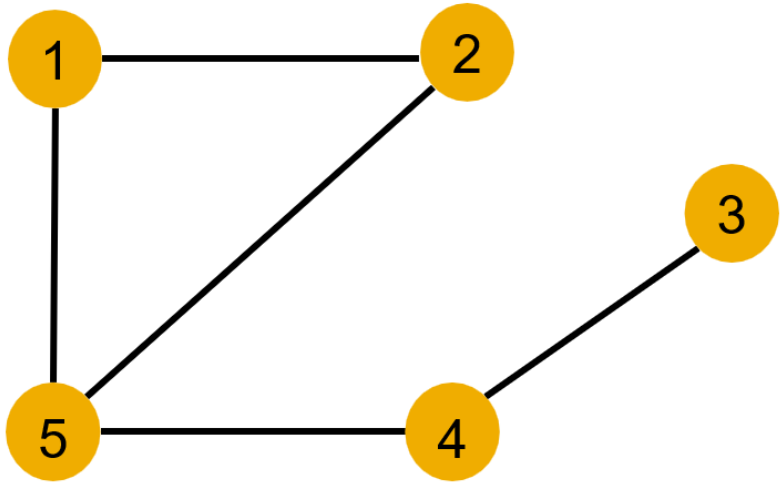
- **GraphSAGE (max-pool)**





Note: Note Colors

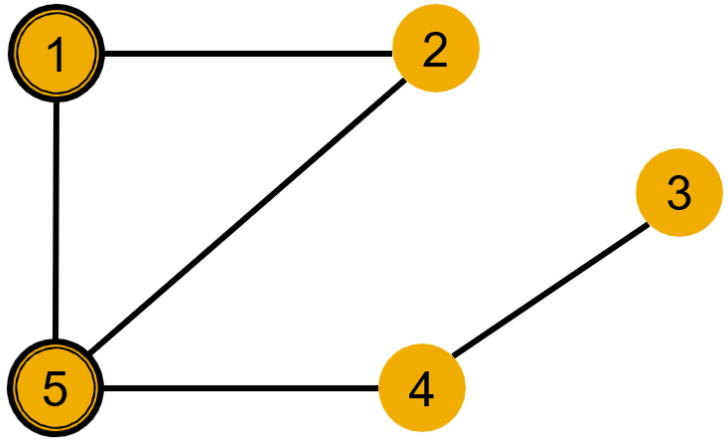
- We use node same/different **colors** to represent nodes with same/different features.
  - For example, the graph below assumes all the nodes share the same feature.



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

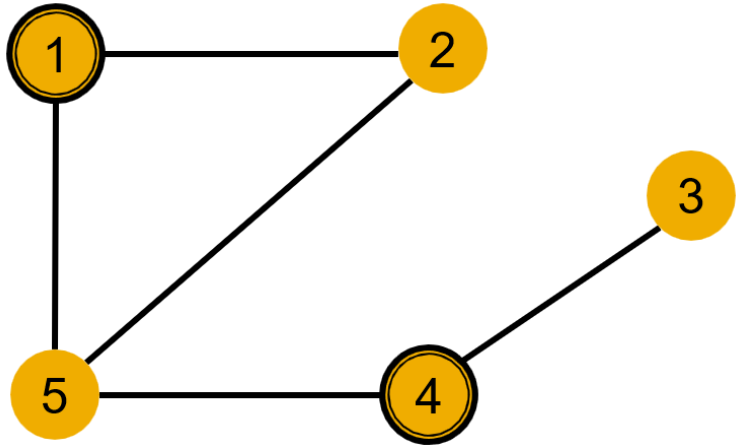
Local Neighborhood Structures

- 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.



Local Neighborhood Structures

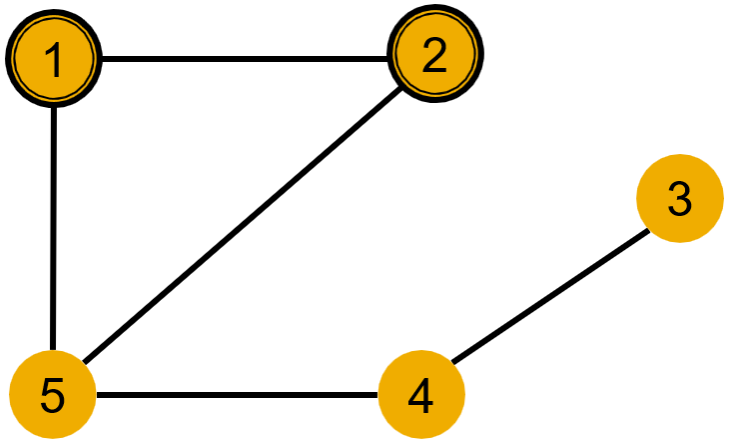
- We specifically consider **local neighborhood structures** around each node in a graph.
  - **Example:** Nodes 1 and 4 both have the same node degree of 2. However, they still have **different** neighborhood structures because **their neighbors have different node degrees.**



Node 1 has neighbors of degrees 2 and 3.
Node 4 has neighbors of degrees 1 and 3.

Local Neighborhood Structures

- We specifically consider **local neighborhood structures** around each node in a graph.
  - **Example:** Nodes 1 and 2 have the **same** neighborhood structure because **they are symmetric within the graph.**



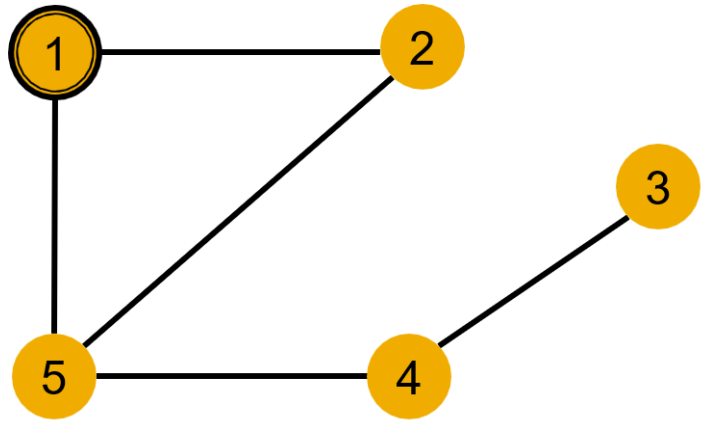
Node 1 has neighbors of degrees 2 and 3.
Node 2 has neighbors of degrees 2 and 3.
And even if we go a step deeper to 2nd hop neighbors, both nodes have the same degrees (Node 4 of degree 2)

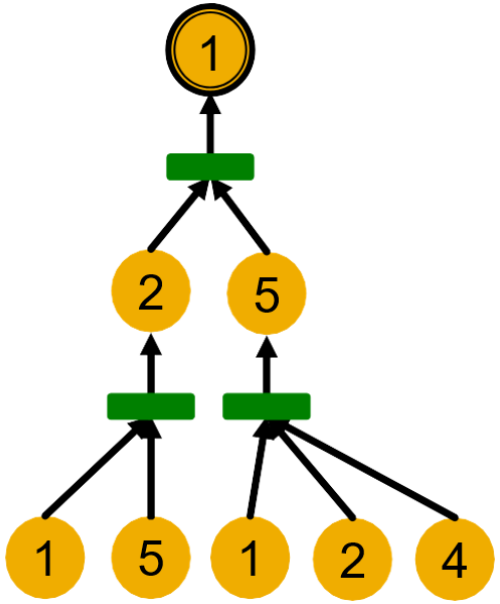
Local Neighborhood Structures

- **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

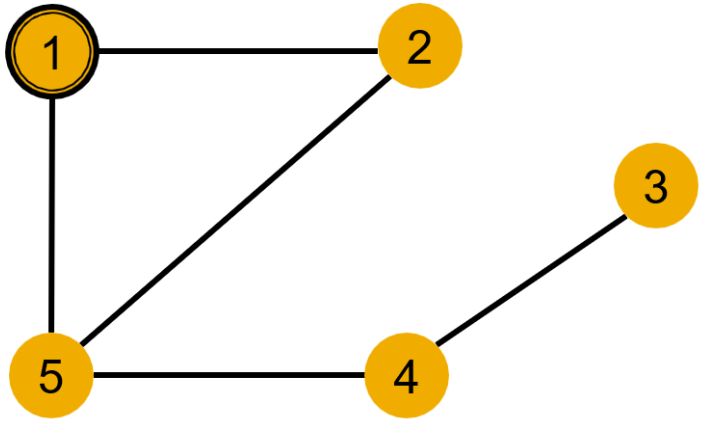
- **In each layer, a GNN aggregates neighboring node embeddings.**
- A GNN generates node embeddings through a
  - computational graph defined by the neighborhood.
    - **Ex:** Node 1's computational graph (2-layer GNN)

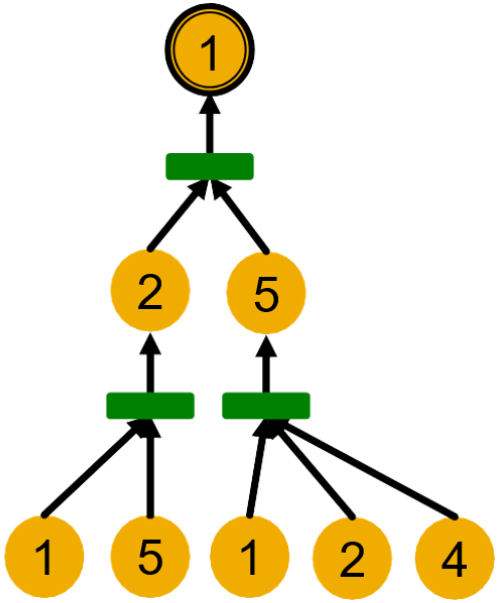


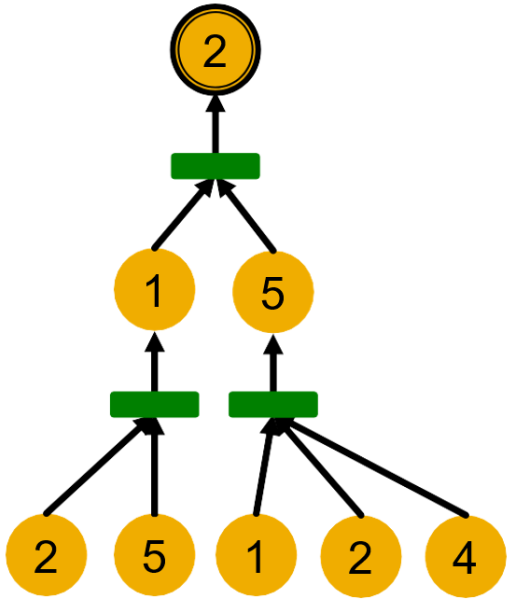


Computational Graph

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

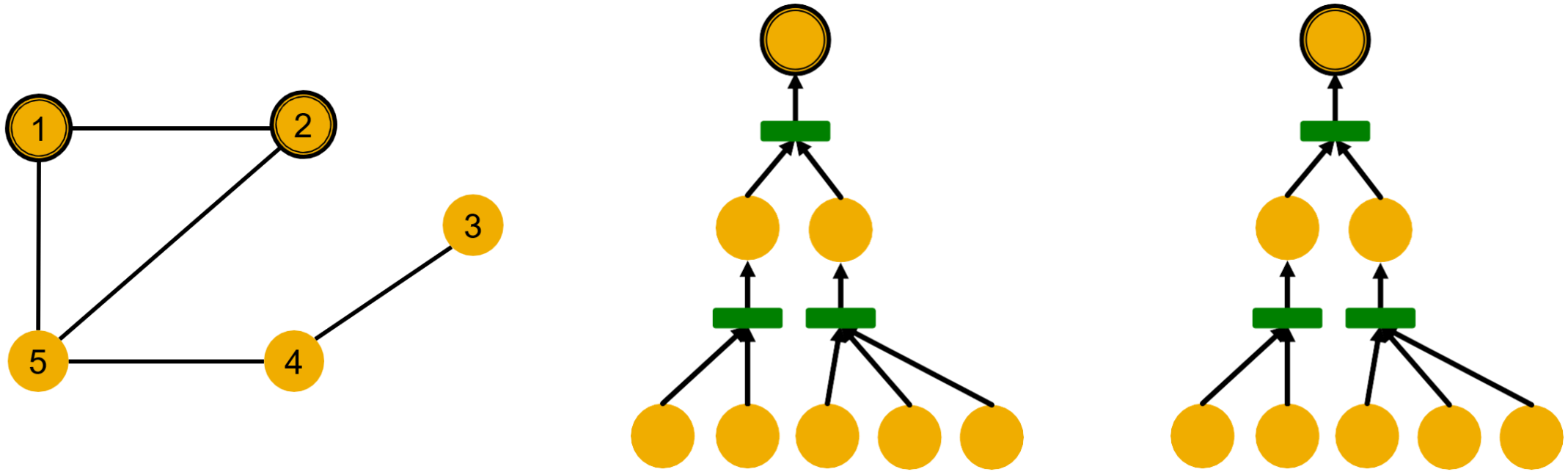






Computational Graph

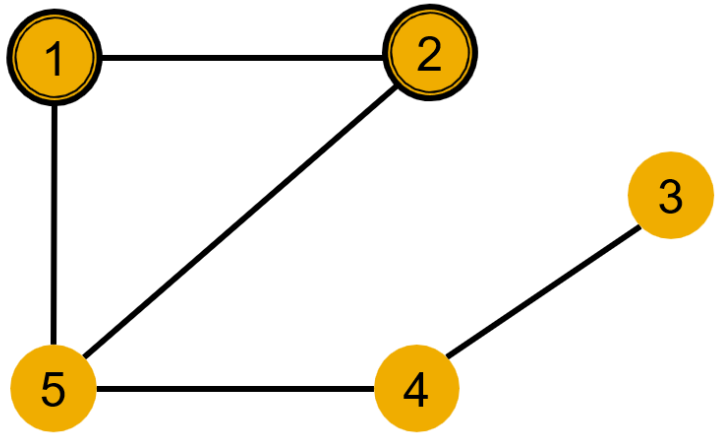
- **Ex:** Nodes 1 and 2's computational graphs.
- **But GNN only sees node features (not IDs):**

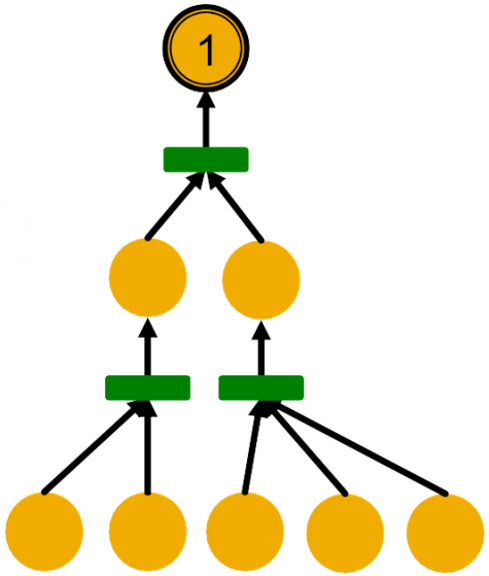


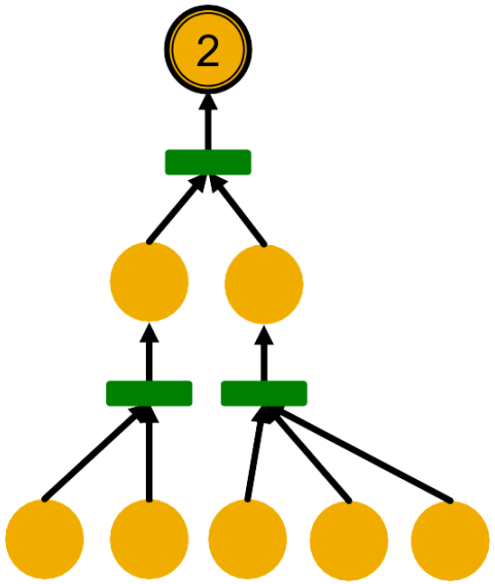
Computational Graph

- **A GNN will generate the same embedding for nodes 1 and 2 because:**
- Computational graphs are the same.
- Node features (colors) are identical.

Note: GNN does not care about node ids, it just aggregates features vectors of different nodes.



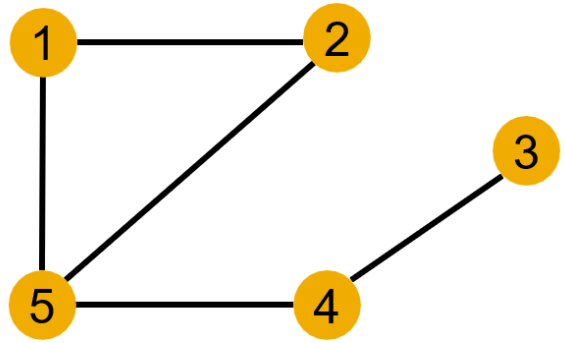


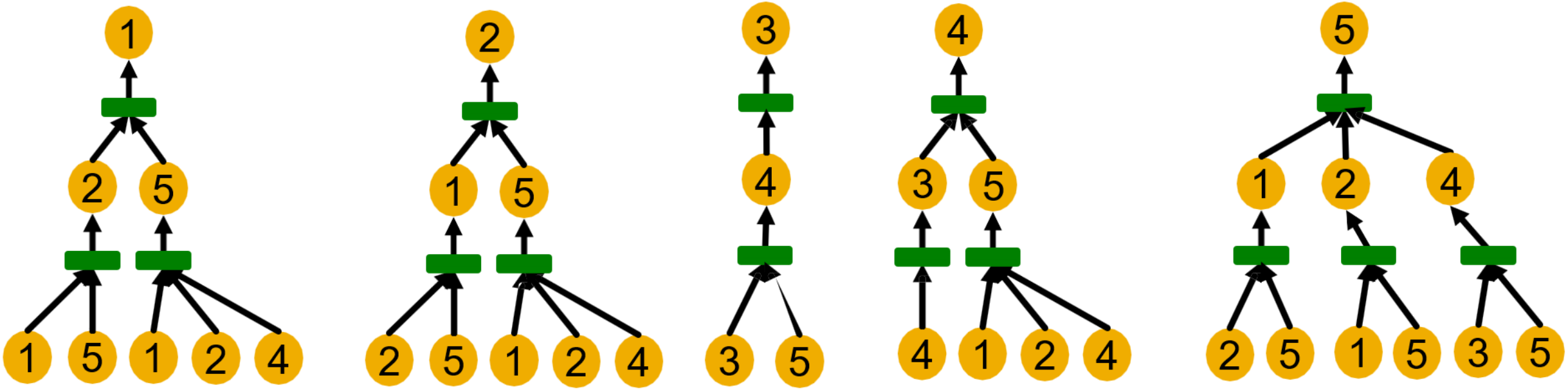


GNN won't be able to distinguish nodes 1 and 2

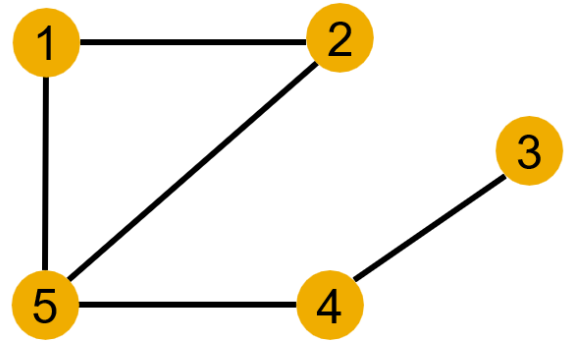
Computational Graph

- **In general, different local neighborhoods define different computational graphs**





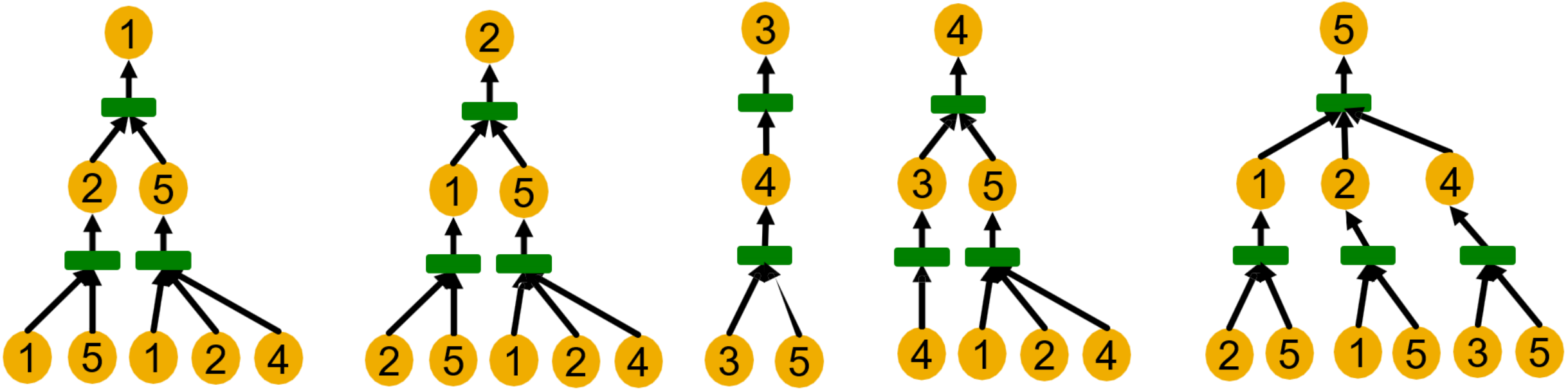
Computational Graph



- Computational graphs are identical to **rooted subtree structures** around each node.

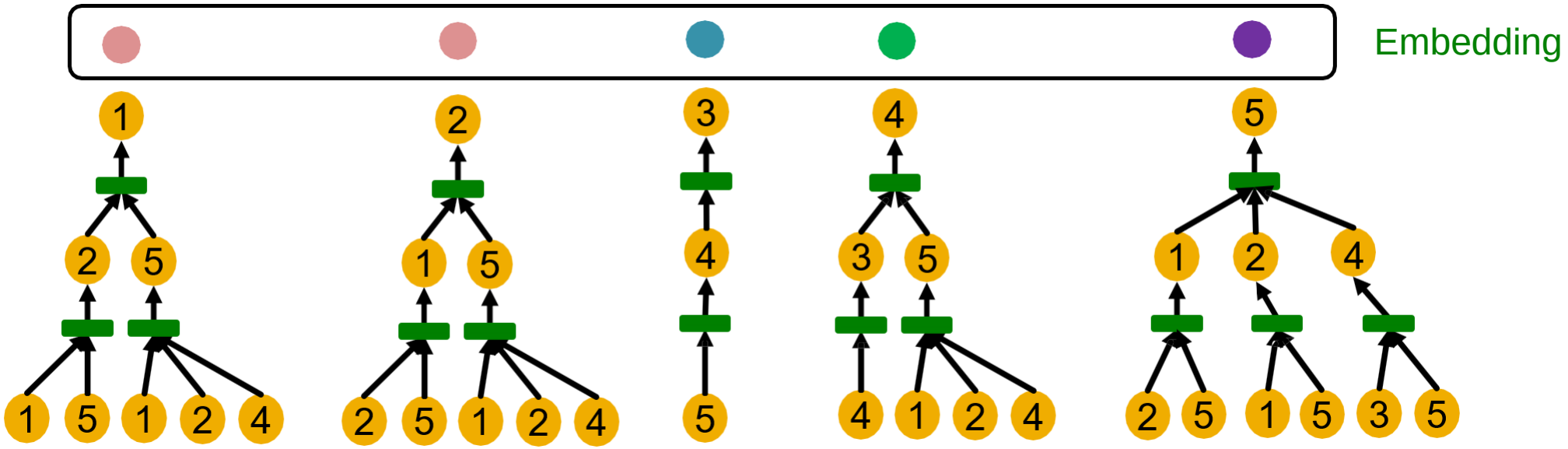
Rooted subtree structures

(defined by recursively unfolding neighboring nodes from the root nodes)



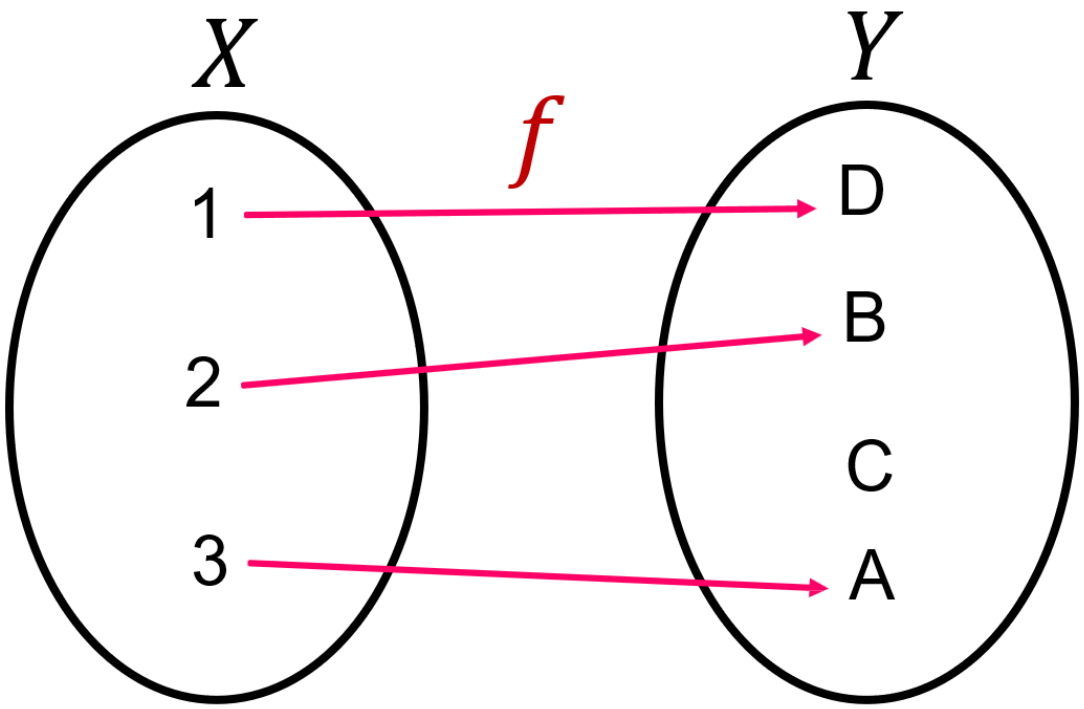
Computational Graph

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



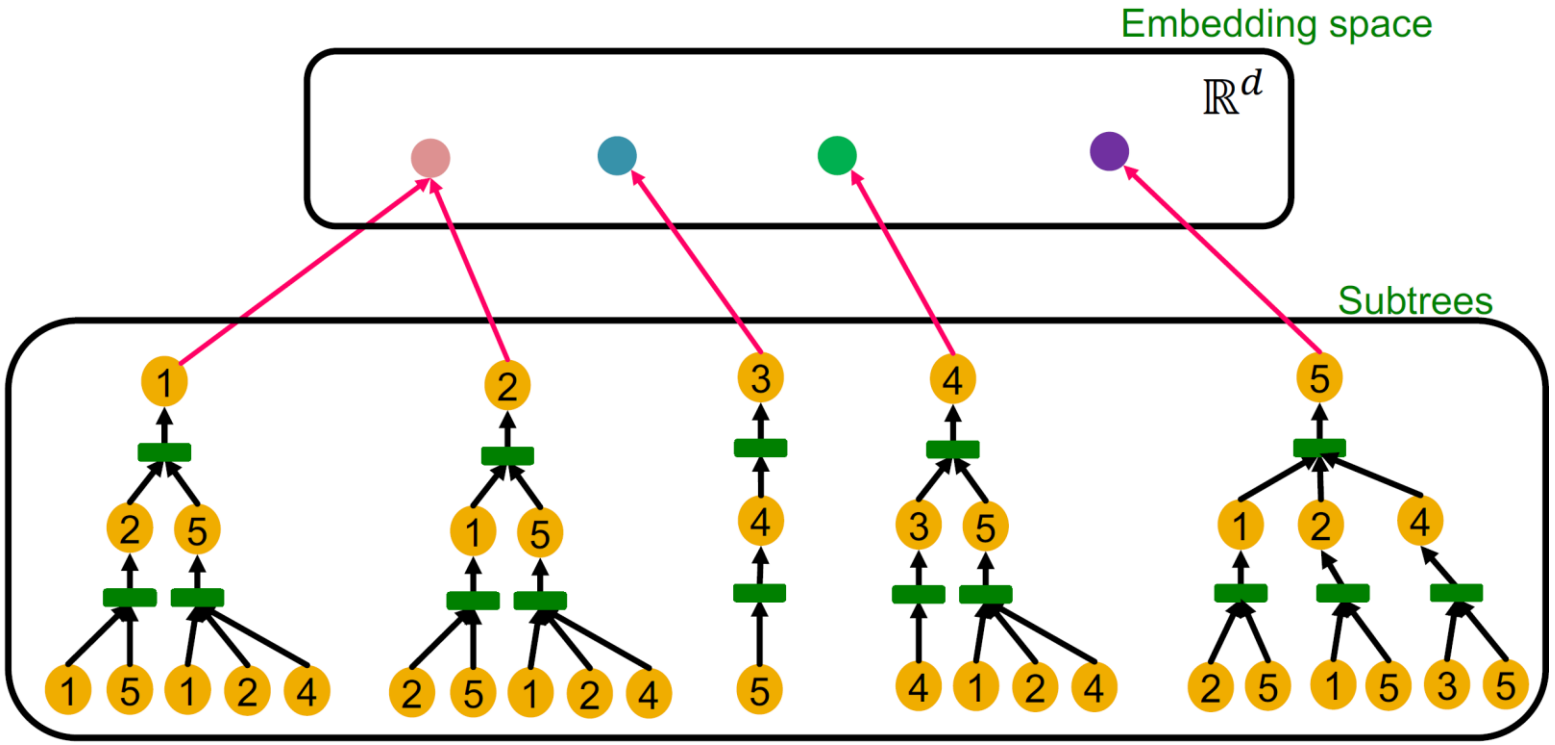
Injective Function

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



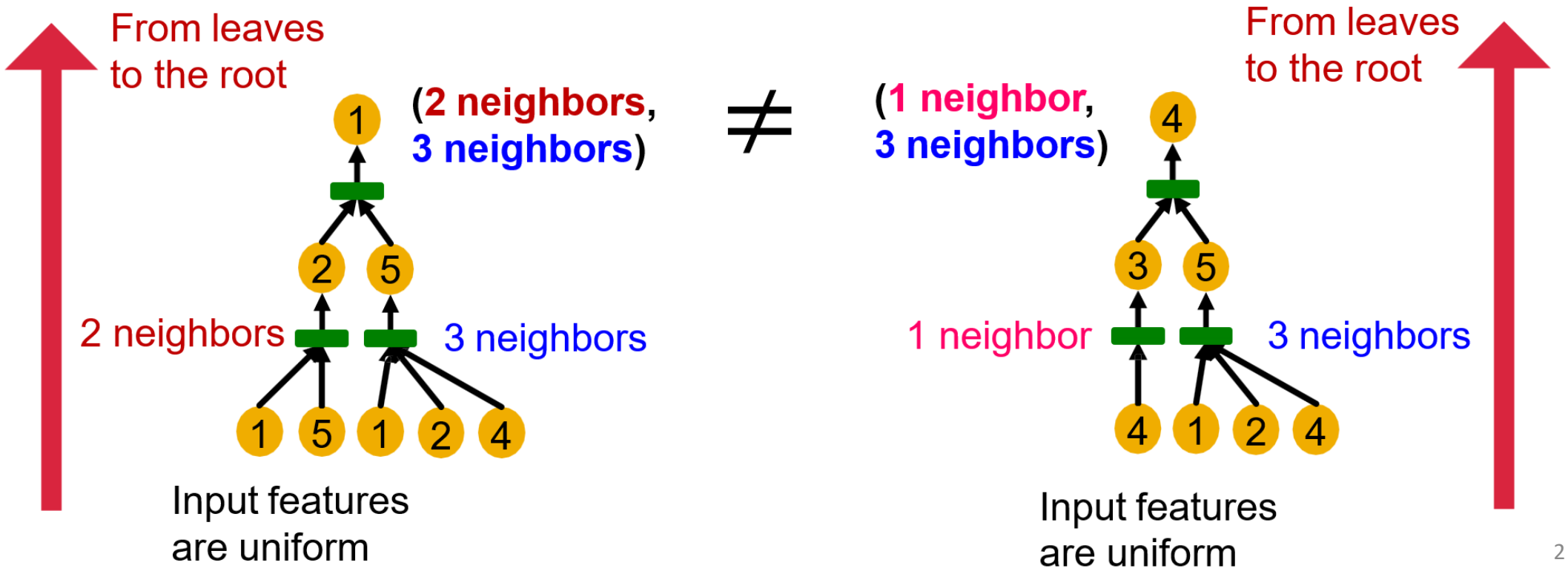
How Expressive is a GNN?

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



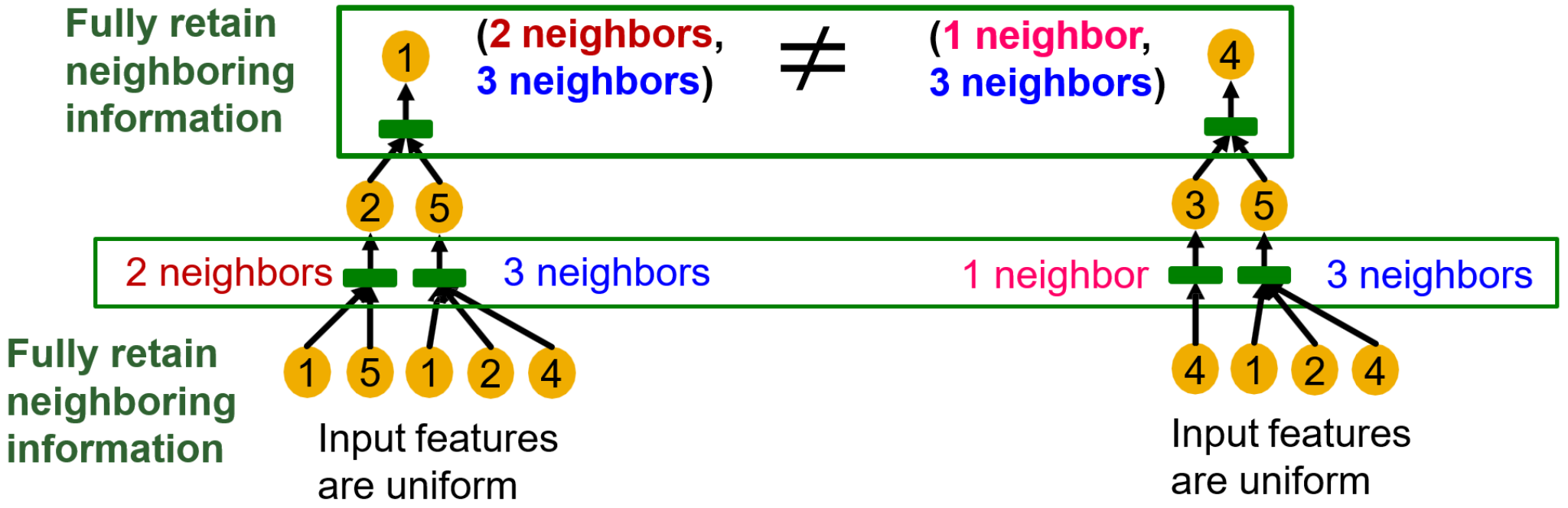
How Expressive is a GNN?

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



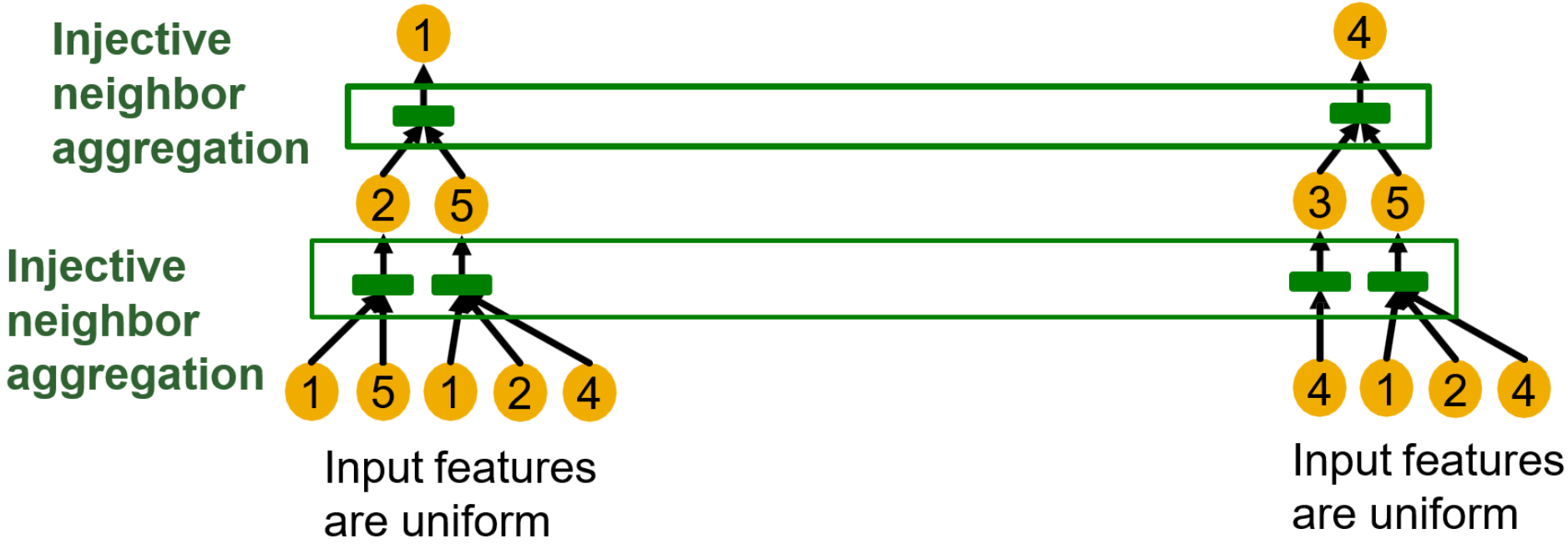
How Expressive is a GNN?

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



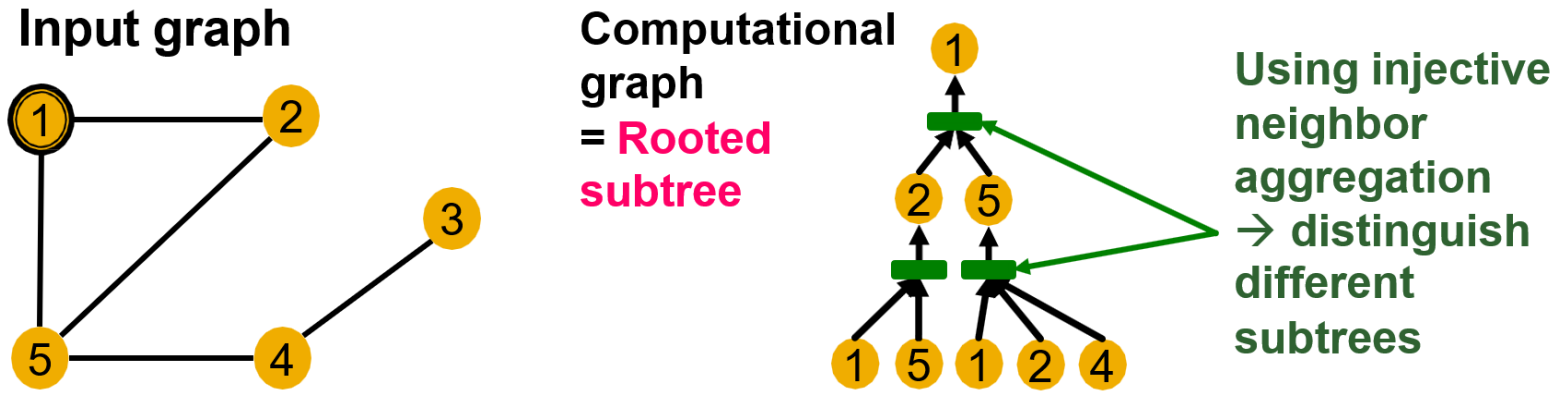
How Expressive is a GNN?

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



How Expressive is a GNN?

- **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.**

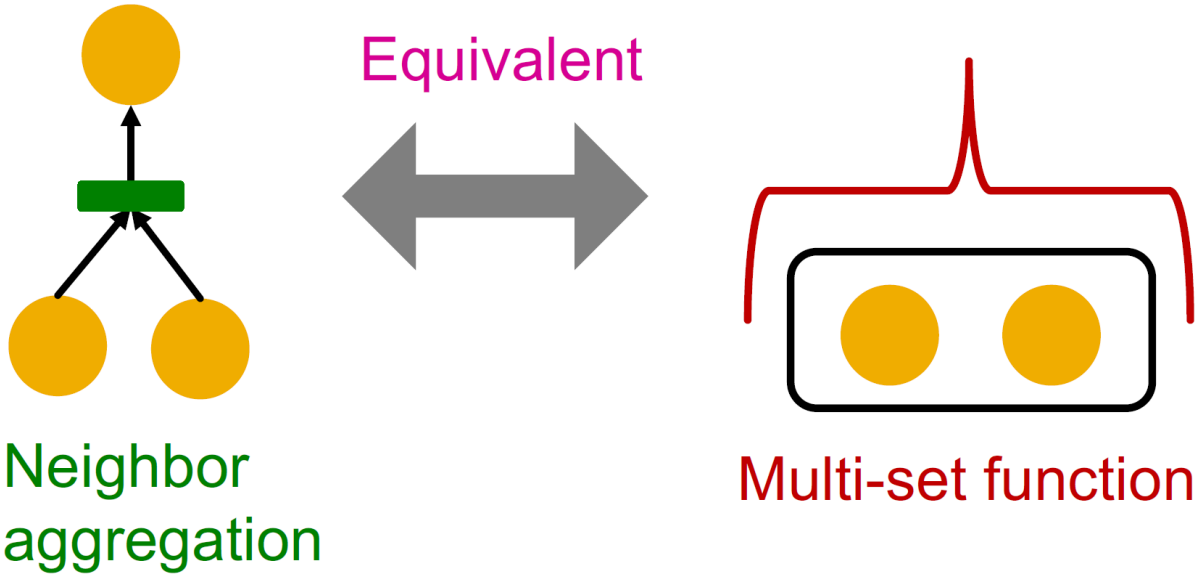
Design the Most Powerful GNN

Expressive Power

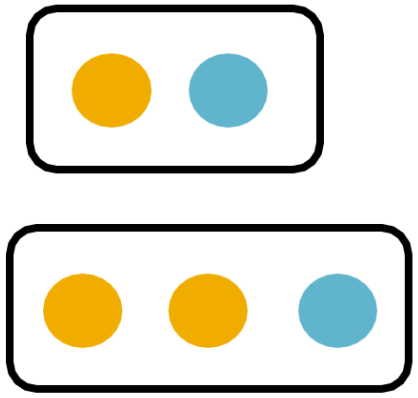
- **Key observation**: **Expressive power of GNNs can be characterized by that of neighbor aggregation functions they use.**
  - A more expressive aggregation function leads to a more expressive a GNN.
  - Injective aggregation function leads to the most expressive GNN.
- **Next:**
  - Theoretically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



Examples of multi-set



Same color indicates the same features.

Neighbor Aggregation

- **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

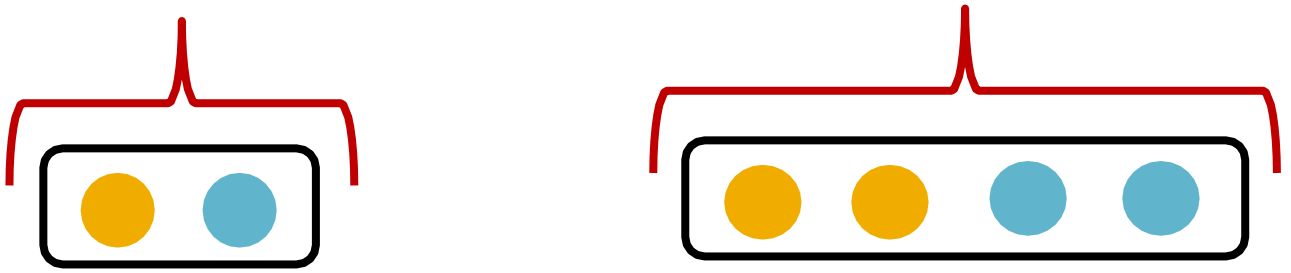
$$\text{Mean}(\{x_u\}_{u \in N(v)})$$

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

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

Neighbor Aggregation: GCN Case Study

- **GCN (mean-pool)** [Kipf & Welling ICLR 2017]
  - Take **element-wise mean**, followed by linear function and ReLU activation, i.e., $\max(0, x)$.
  - **Theorem** [Xu et al. ICLR 2019]
    - GCN's aggregation **function cannot distinguish different multi-sets with the same color proportion**.
  - **Failure case**

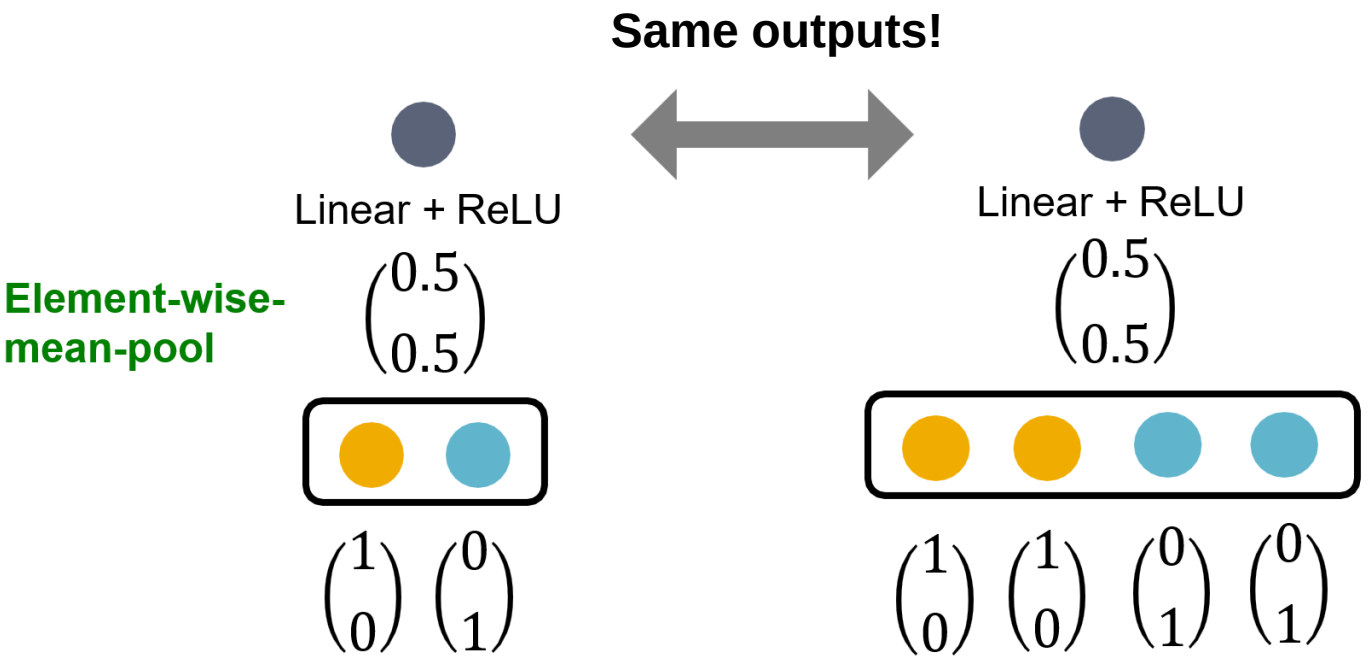


Neighbor Aggregation: GCN Case Study

- 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.

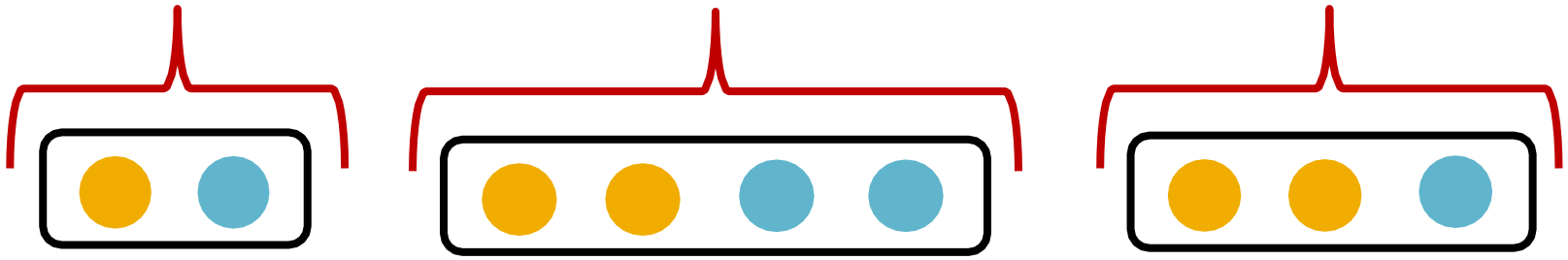
Neighbor Aggregation: GCN Case Study

- **GCN (mean-pool)** [Kipf & Welling ICLR 2017]
  - **Failure case illustration**



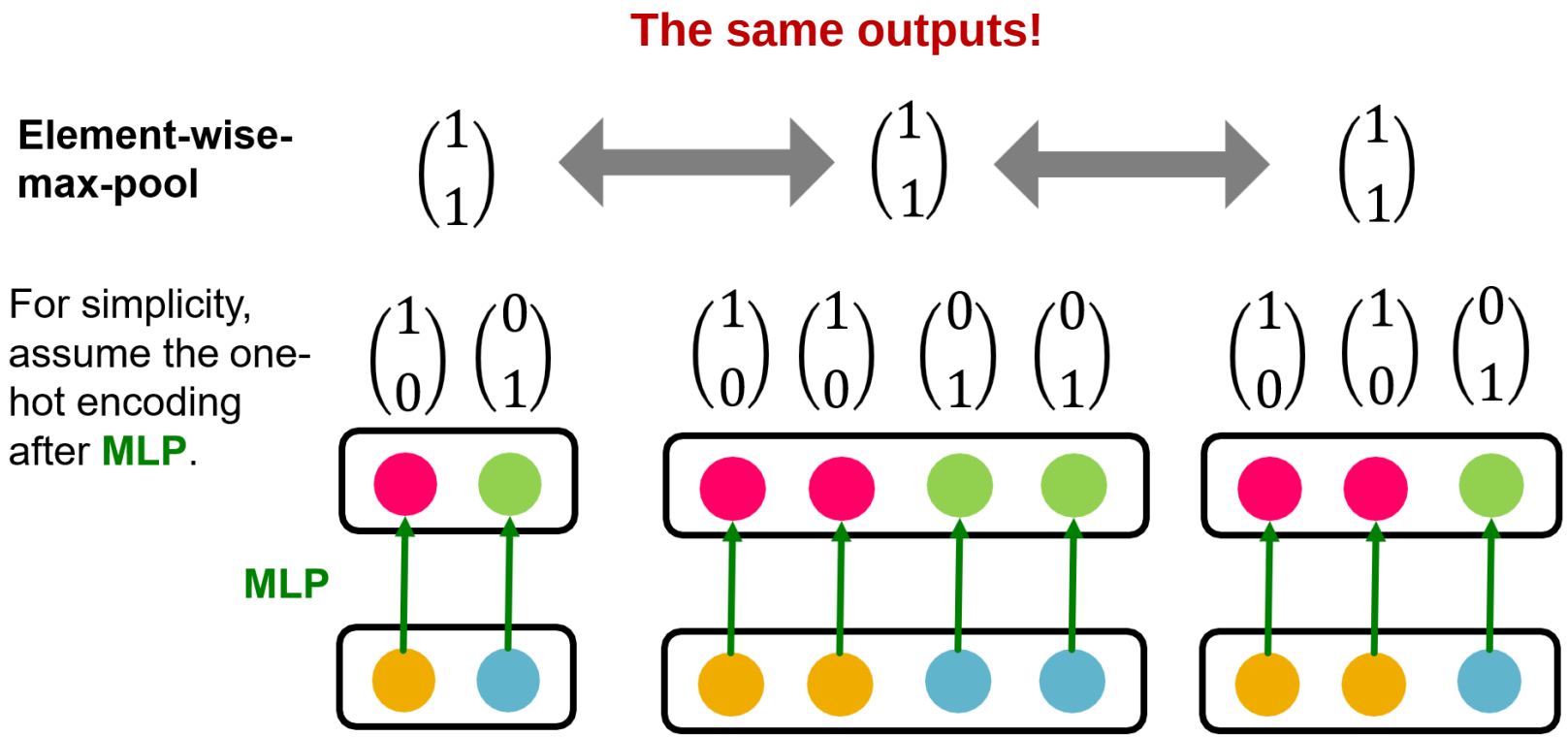
Neighbor Aggregation: GraphSAGE Case Study

- **GraphSAGE (max-pool)** [Hamilton et al. NeurIPS 2017]
  - Apply an MLP, then take **element-wise max**.
  - **Theorem** [Xu et al. ICLR 2019]
    - GraphSAGE's aggregation **function cannot distinguish different multi-sets with the same set of distinct colors**.
  - **Failure case**



Neighbor Aggregation: GraphSAGE Case Study

- **GraphSAGE (max-pool)** [Hamilton et al. NeurIPS 2017]
  - **Failure case illustration**



Summary So Far

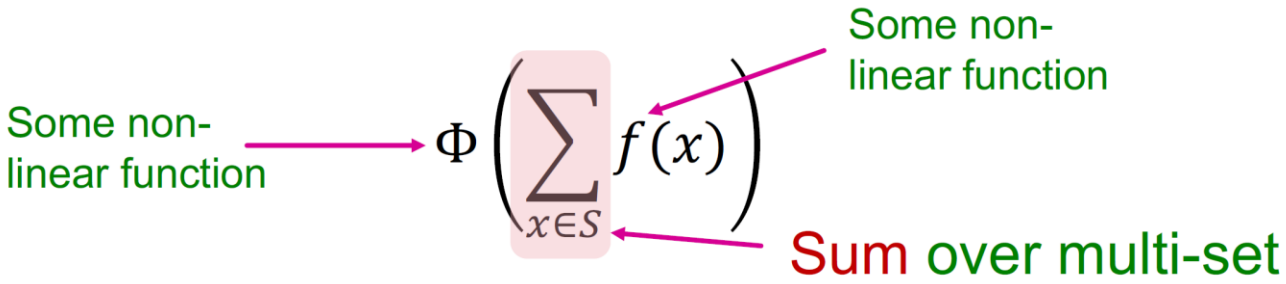
- We analyzed the **expressive power of GNNs.**
- **Main takeaways**:
  - Expressive power of GNNs can be characterized by that of the neighbor aggregation function.
  - Neighbor aggregation is a function over multi-sets (sets with repeating elements)
  - GCN and GraphSAGE's aggregation functions fail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

Designing Most Expressive GNNs

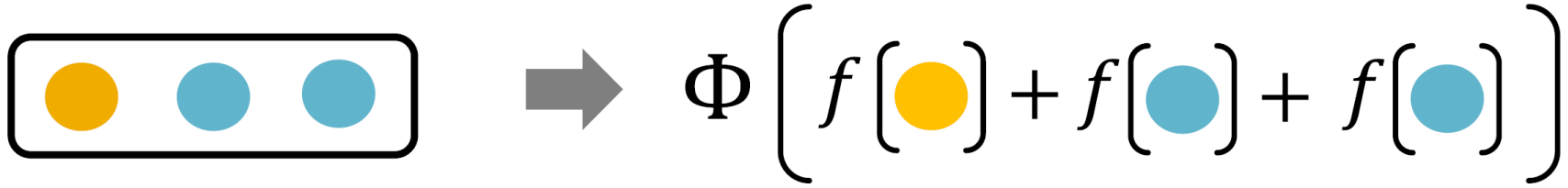
- **Our goal**: **Design maximally powerful GNNs in the class of message-passing GNNs.**
- This can be achieved by designing **injective neighbor aggregation function over multi-sets**.
- **Here, we design a neural network that can model injective multiset function.**

Injective Multi-Set Function

- Theorem [Xu et al. ICLR 2019]
- Any injective multi-set function can be expressed as:



S : multi-set

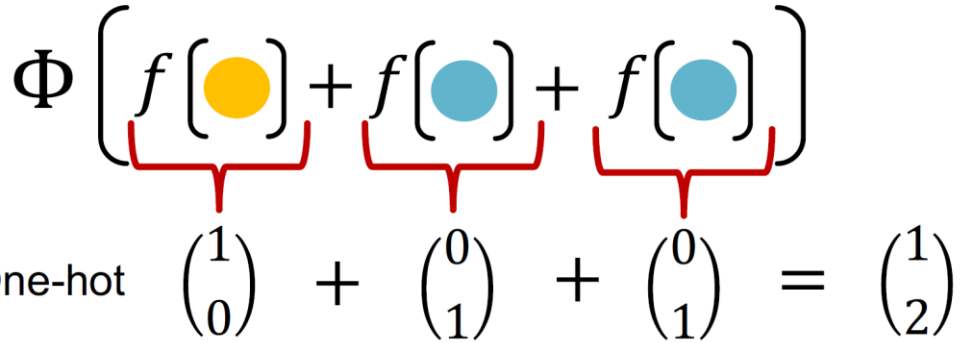


Injective Multi-Set Function

- **Proof Intuition:** [Xu et al. ICLR 2019]
- f produces one-hot encodings of colors. Summation of the one-hot encodings retains all the information about the input multi-set.

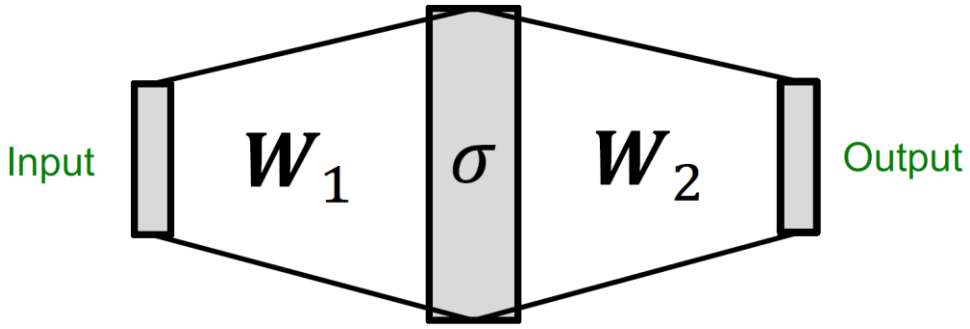
$$\Phi\left(\sum_{x \in S} f(x)\right)$$

Example:



Universal Approximation Theorem

- **How to model Φ and f in Φ $\sum_{x \in S} f(x)$?**
  - 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**.



Injective Multi-Set Function

- **We have arrived at a neural network that can model any injective multiset function.**

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

- In practice, MLP hidden dimensionality of 100 to 500 is sufficient.

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.

Weisfeiler-Lehman Isomorphism Test

- Determining whether two graphs are isomorphic when the correspondance is not provided is a challenging problem
- Weisfeiler-Lehman Isomorphism Test
  - Produces for each graph a canonical form.
  - If the canonical forms of two graphs are not equivalent, then the graphs are definitively not isomorphic.
- Weisfeiler-Lehman kernel
  - WL kernel has been both theoretically and empirically shown to distinguish most of the real-world graphs [Cai et al. 1992].

Weisfeiler-Lehman Kernel

- **Goal**: Design an efficient graph feature descriptor $\phi(G)$
- **Idea**: Use neighborhood structure to iteratively enrich node vocabulary.
  - Generalized version of **Bag of node degrees** since node degrees are one-hop neighborhood information.
- **Algorithm to achieve this**:
  - **Color refinement**

https://www.jmlr.org/papers/volume12/shervashidze11a/shervashidze11a.pdf

Color Refinement

- **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

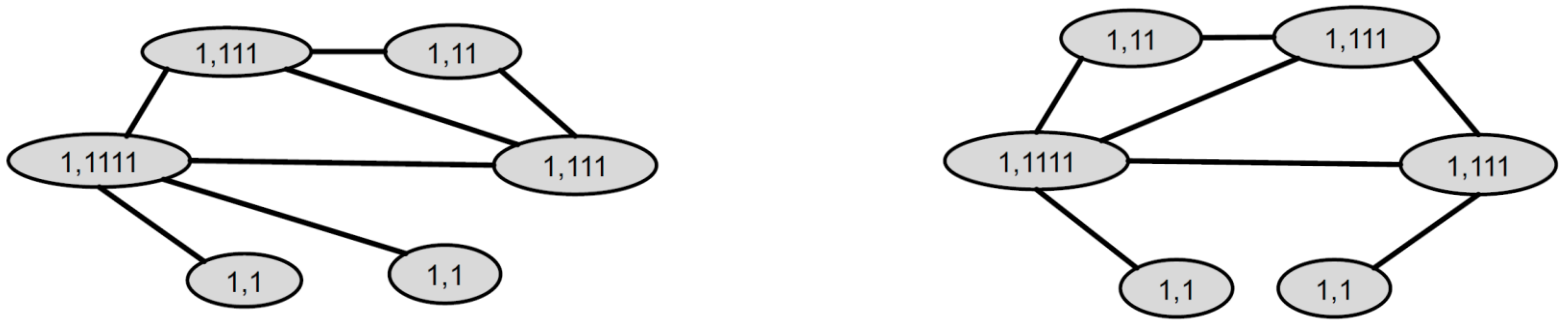
$$c^{(k+1)}(v) = \text{HASH}\left(c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right),$$

- where HASH maps different inputs to different colors.
  - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood

Color Refinement (1)

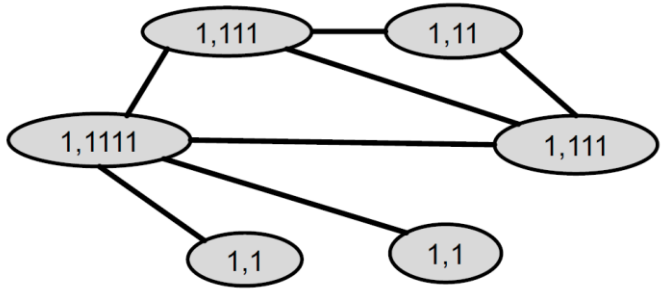
- **Example of color refinement given two graphs**
- Assign initial colors

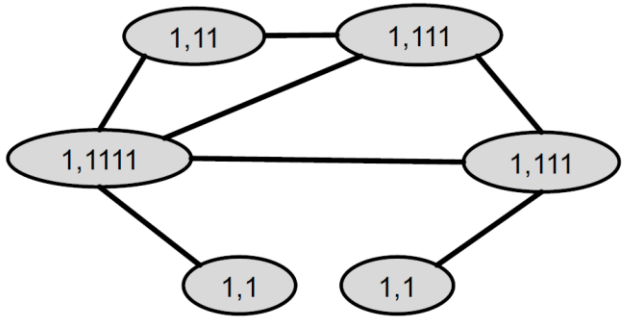
- Aggregate neighboring colors



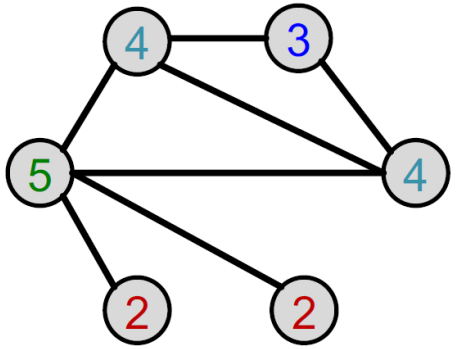
Color Refinement (2)

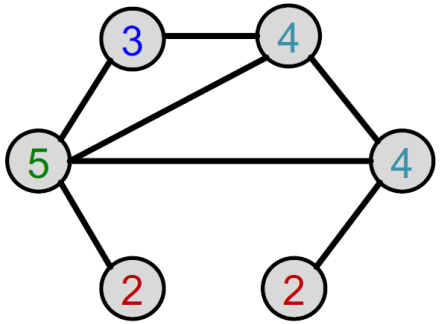
- **Example of color refinement given two graphs**
- Aggregated colors:





- **Injectively** HASH the aggregated colors



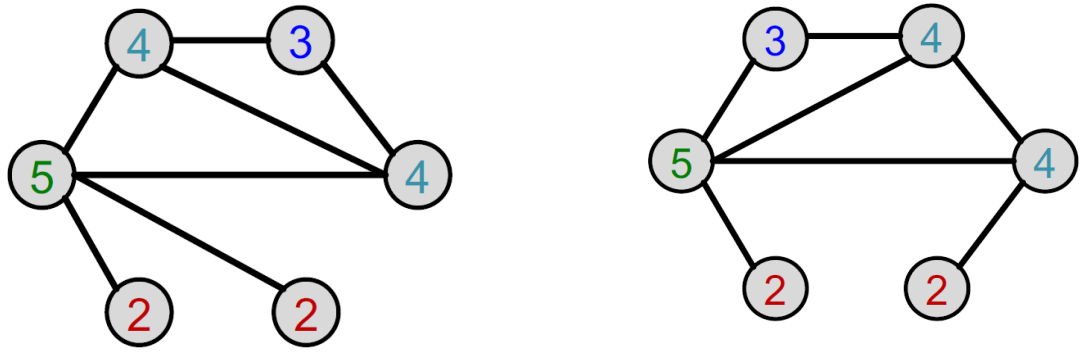


HASH table: **Injective!**

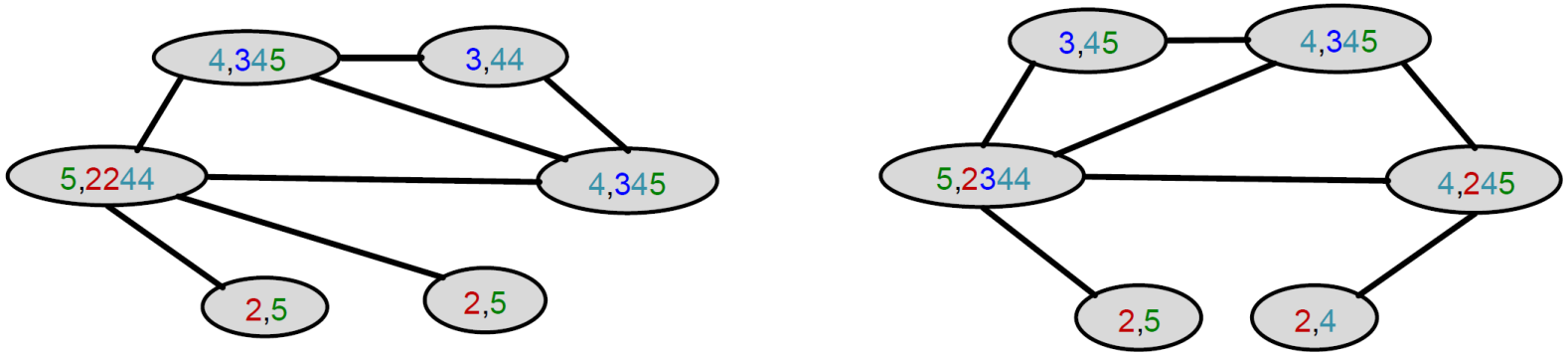
1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Color Refinement (3)

- Example of color refinement given two graphs
- Colors

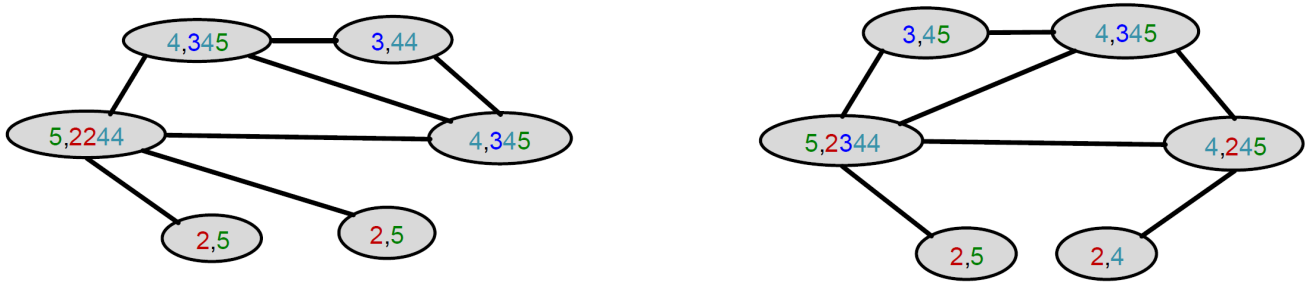


- Aggregated colors

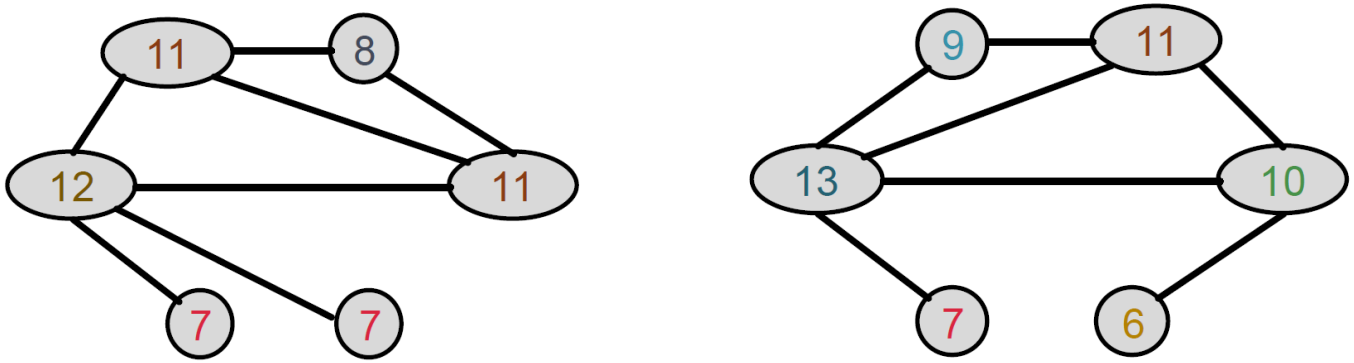


Color Refinement (4)

- **Example of color refinement given two graphs**
- Aggregated colors:



- **Injectively** HASH the aggregated colors

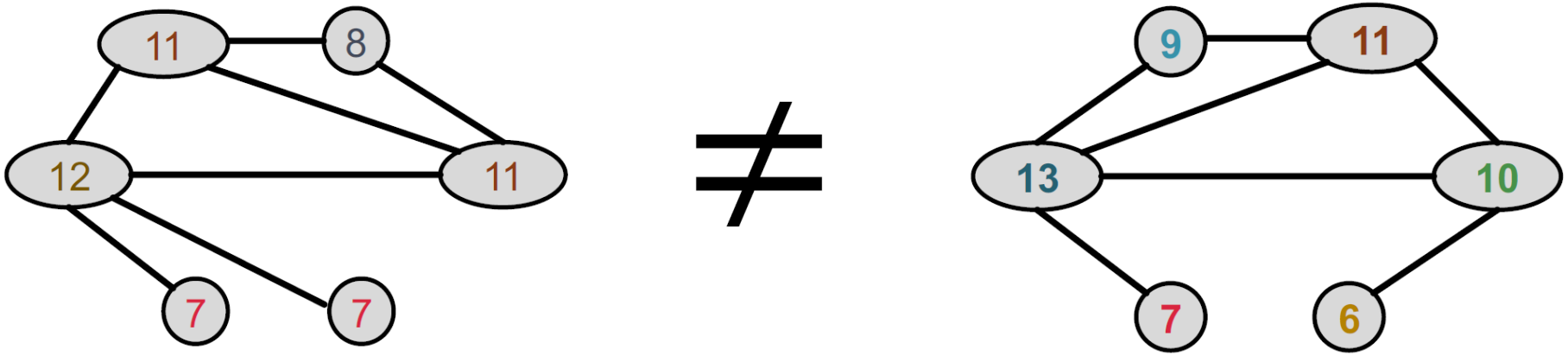


Hash table

2,4	-->	6
2,5	-->	7
3,44	-->	8
3,45	-->	9
4,245	-->	10
4,345	-->	11
5,22344	-->	12
5,2344	-->	13

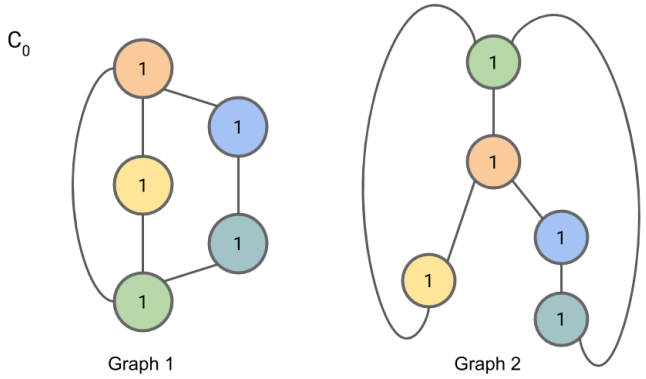
Color Refinement (5)

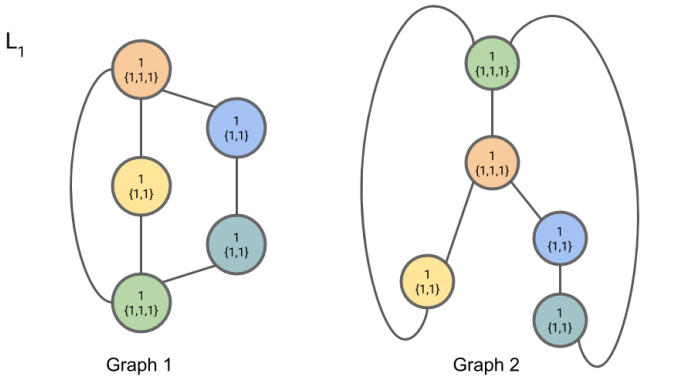
- **Example of color refinement given two graphs**
  - Process continues until a stable coloring is reached

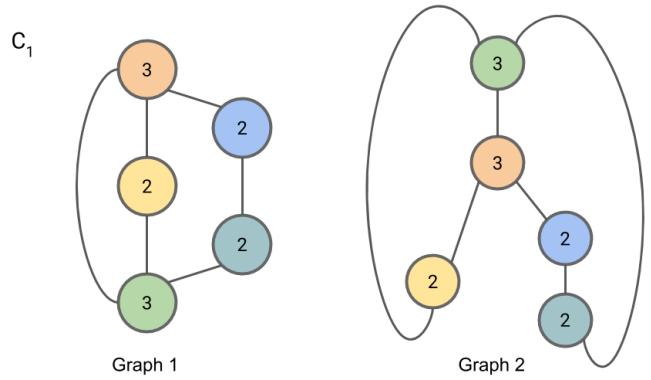


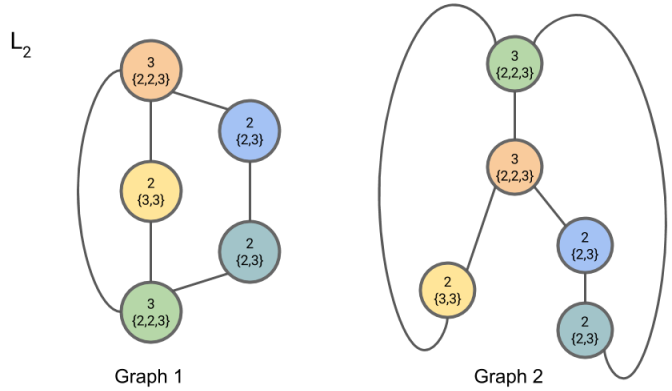
Another Example

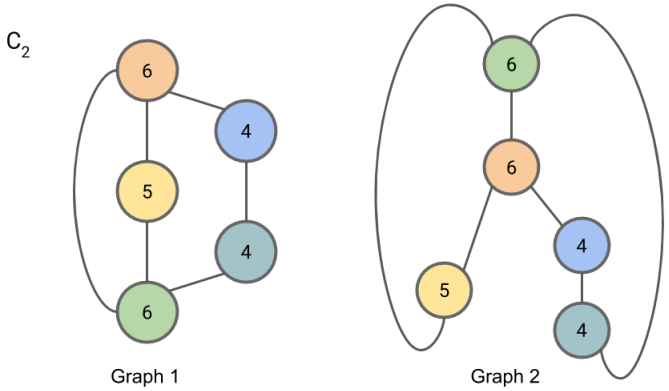
https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/

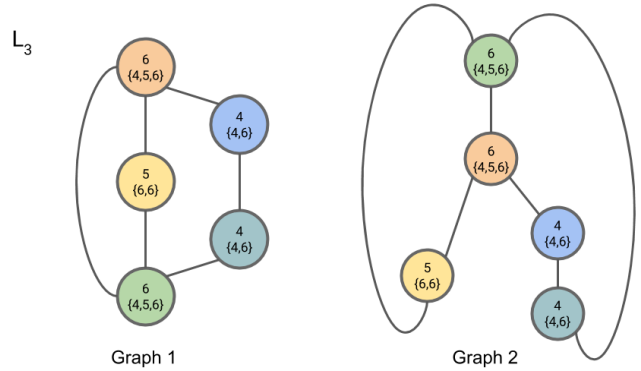


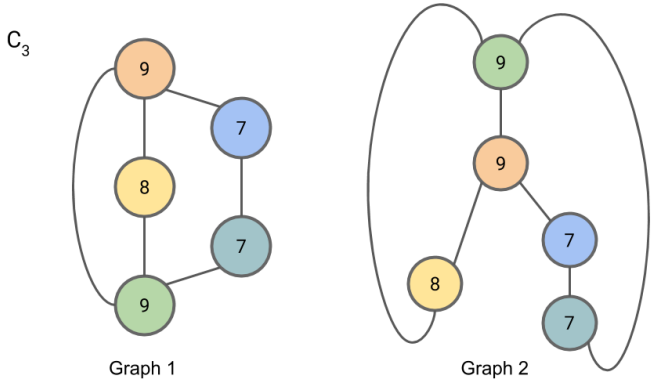








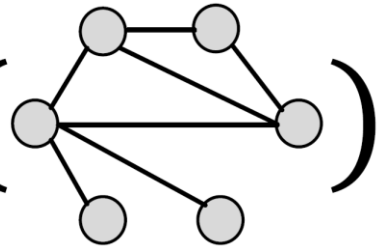




Since the partition of nodes by compressed label has not changed from C2 to C3, we may terminate the algorithm here.

Weisfeiler-Lehman Kernel

- After color refinement, WL kernel counts number of nodes with a given color.

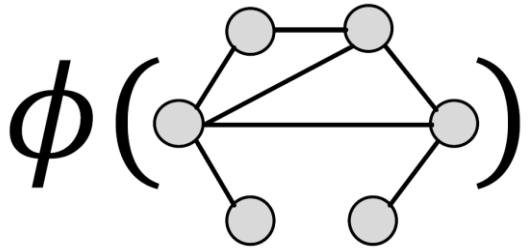
ϕ()

Colors

1,2,3,4,5,6,7,8,9,10,11,12,13

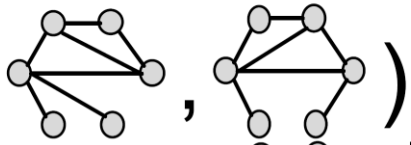
= [6,2,1,2,1,0,2,1,0, 0, 0, 2, 1]

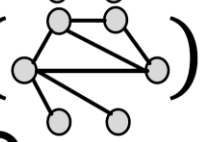
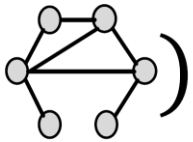
Counts

ϕ()

1,2,3,4,5,6,7,8,9,10,11,12,13

= [6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

K()

= ϕ()^T ϕ()

= 49

Weisfeiler-Lehman Kernel

- WL kernel is **computationally efficient**
  - The time complexity for color refinement at each step is linear in #(edges), since it involves aggregating neighboring colors.
- When computing a kernel value, only colors appeared in the two graphs need to be tracked.
  - Thus, #(colors) is at most the total number of nodes.
- Counting colors takes linear-time w.r.t. #(nodes).
- In total, time complexity is **linear in #(edges)**.

The Complete GIN Model

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

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

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

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

Root node features

Neighboring node colors

The Complete GIN Model

- **Theorem** (Xu et al. ICLR 2019)
- Any injective function over the tuple

Root node feature

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

Neighboring node features

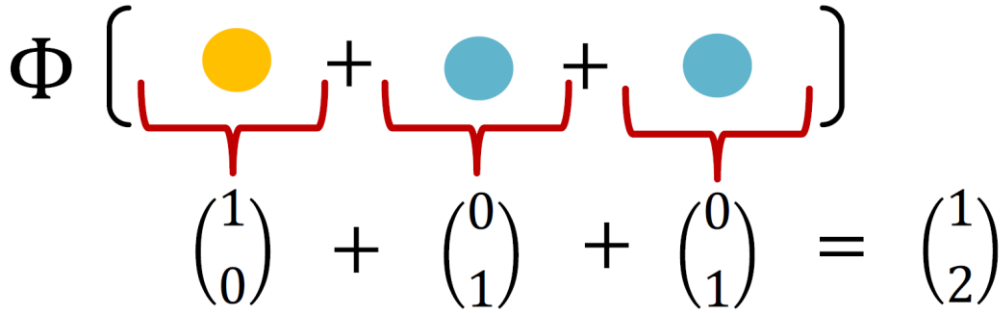
- can be modeled as

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

- where ϵ is a learnable scalar.

The Complete GIN Model

- If input feature $c^{(0)}(v)$ is represented as one-hot, **direct summation is injective**.
- Example:



- We only need Φ to ensure the injectivity.

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

Root node features

Neighboring node features

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

The Complete GIN Model

- **GIN's node embedding updates**
- **Given:** A graph G with a set of nodes V.
  - Assign an **initial vector** $c^{(0)}(v)$ to each node v.
  - Iteratively update node vectors by

Differentiable color HASH function

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

  - where GINConv maps different inputs to different embeddings.
- After K steps of GIN iterations, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood.

GNN 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

- **Advantages of GIN over the WL graph kernel are:**
  - Node embeddings are **low-dimensional**; hence, they can capture the fine-grained similarity of different nodes.
  - Parameters of the update function can be **learned for the downstream tasks.**

Expressive Power of GIN

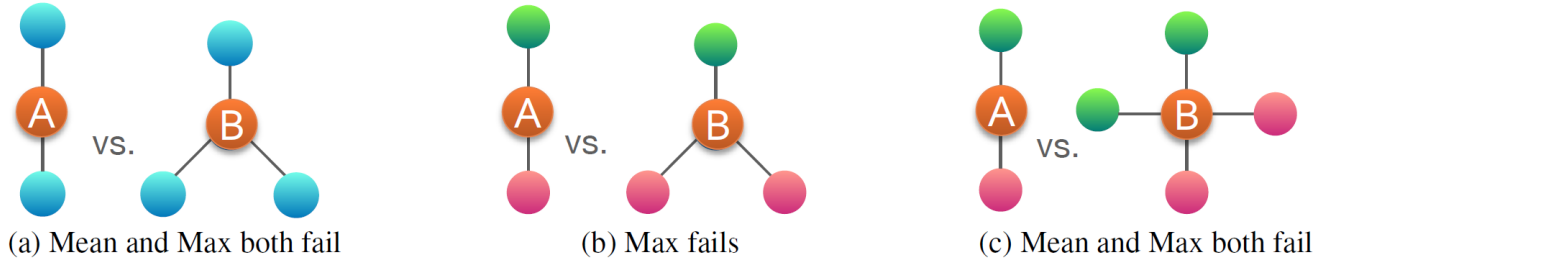
- **Because of the relation between GIN and the WL graph kernel, their expressive is exactly the same.**
  - If two graphs can be distinguished by GIN, they can be also distinguished by the WL kernel, and vice versa.
- **How powerful is this?**
  - WL kernel has been both theoretically and empirically 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

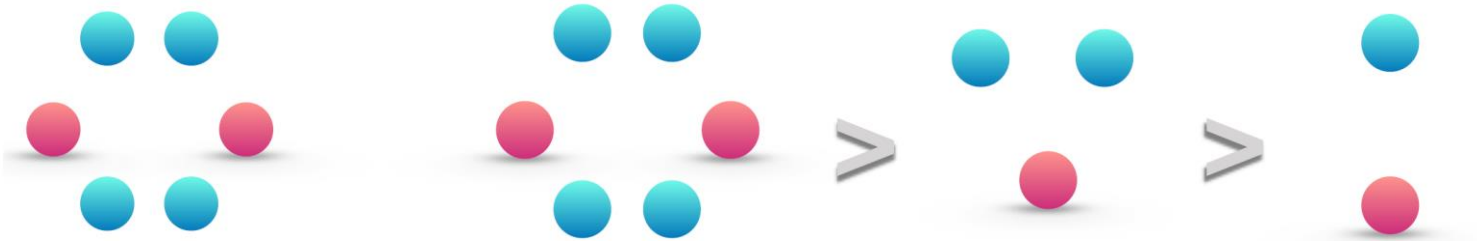
- **Failure cases for mean and max pooling:**



(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

Colors represent feature values

- **Ranking by discriminative power:**



Input sum - multiset mean - distribution max - set