Expressivity of GNNs

StructML - Tutorial 8

Outline

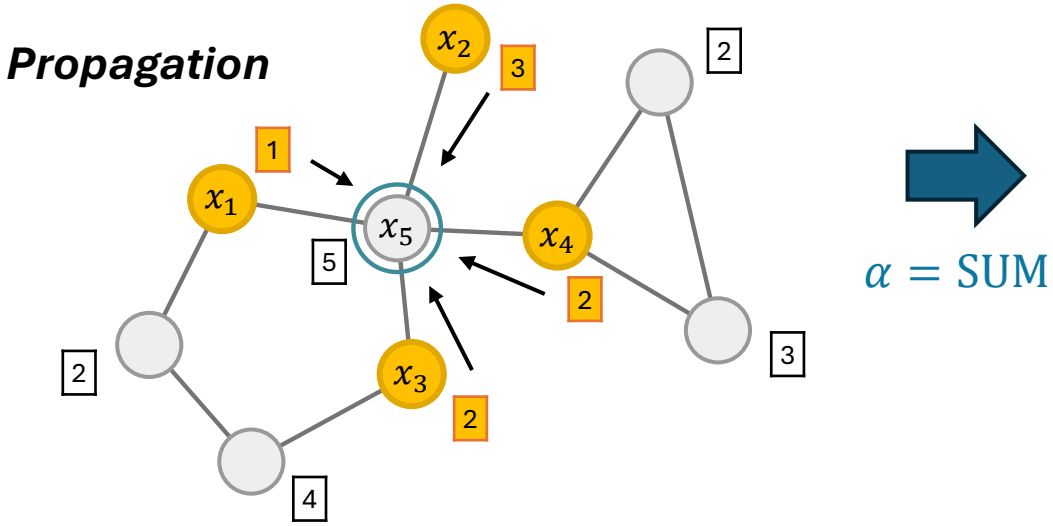
- Isomorphism & 1-WL Test
- GIN - Graph Isomorphism Network
- Compare to Known Architectures
- Beyond 1-WL

Reminder: Message Passing

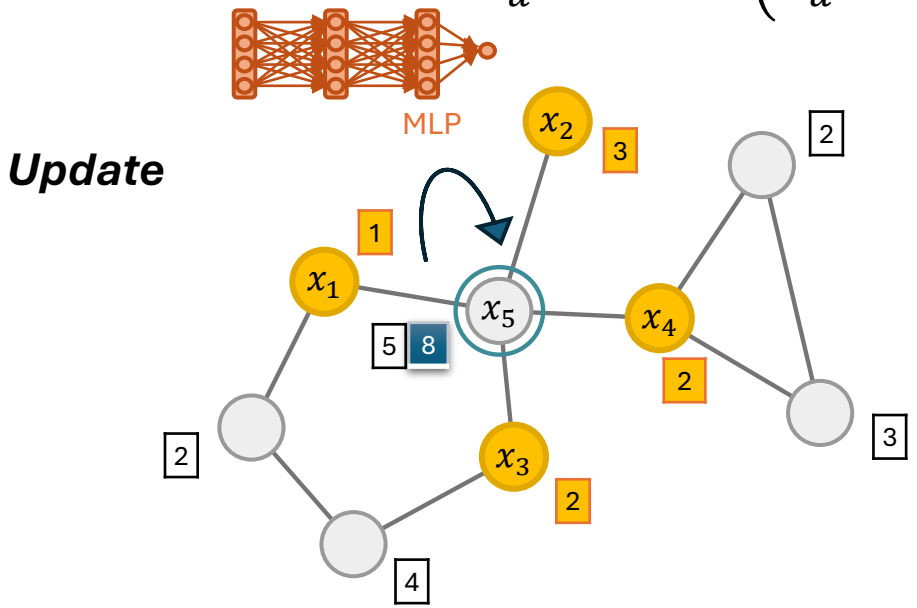
- $N(u)$ - neighbors of u
- $\mathrm{h}_u^{(k)}$ - embedding at "step" k
- $M^{(k)}$ - message function
- $\alpha^{(k)}$ - aggregation function
- $U^{(k)}$ - update function

multiset

$$\mathrm{m}_u^{(k)} := \alpha^{(k)}\left(\left\{\!\left\{ M^{(k)}\left(\mathrm{h}_u^{(k-1)}, \mathrm{h}_v^{(k-1)}\right) \mid v \in N(u) \right\}\!\right\}\right)$$



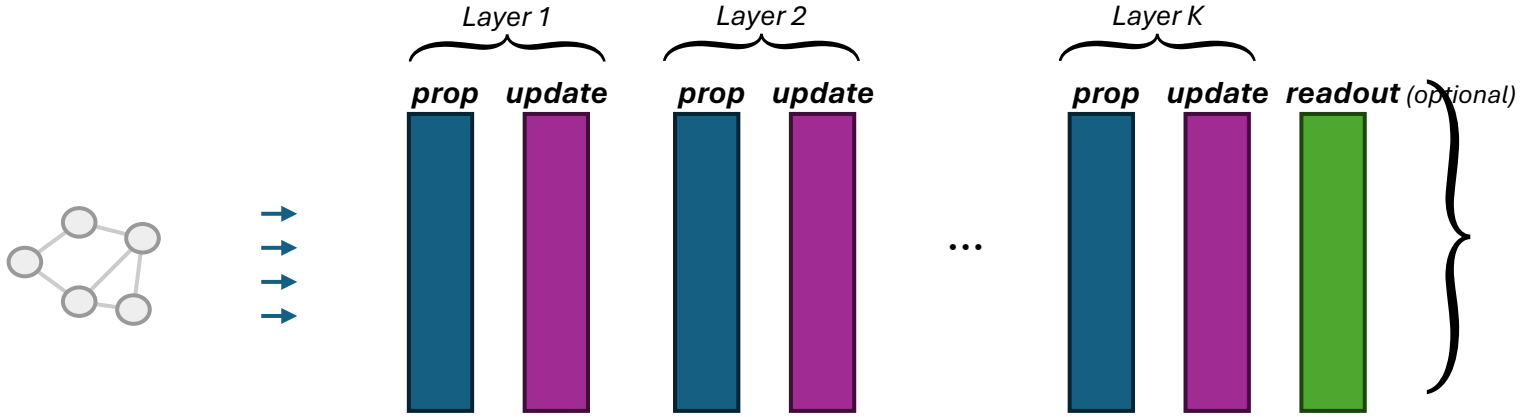
$$\mathrm{h}_u^{(k)} := U^{(k)}\left(\mathrm{h}_u^{(k-1)}, m_u^{(k)}\right)$$



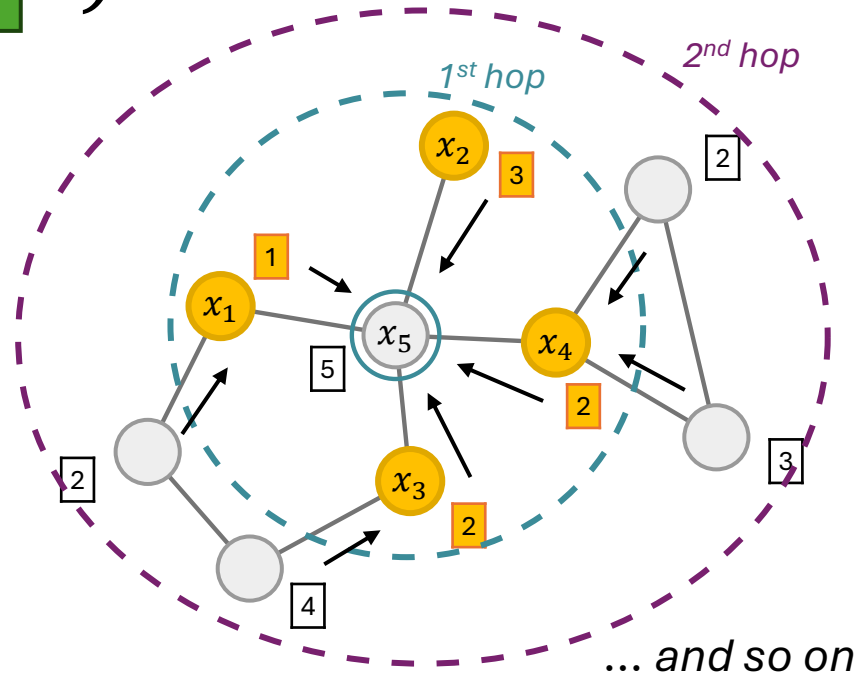
Reminder: Message Passing Neural Network

MPNN = Message Passing GNN

- K Layers



Today: *What is the expressive power of MPNN?*



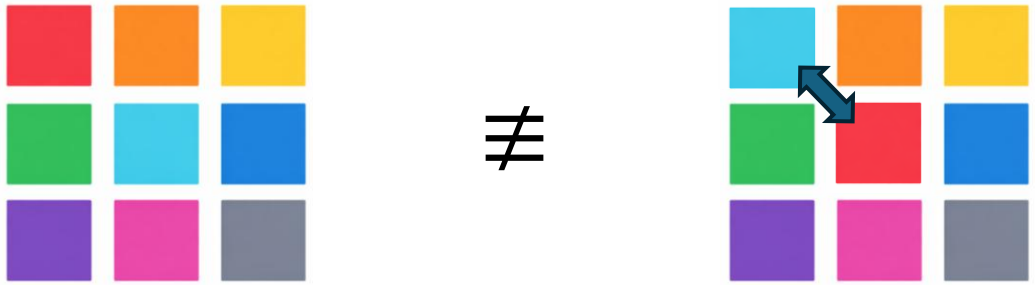
Outline

- Isomorphism & 1-WL Test
- GIN - Graph Isomorphism Network
- Compare to Known Architectures
- Beyond 1-WL

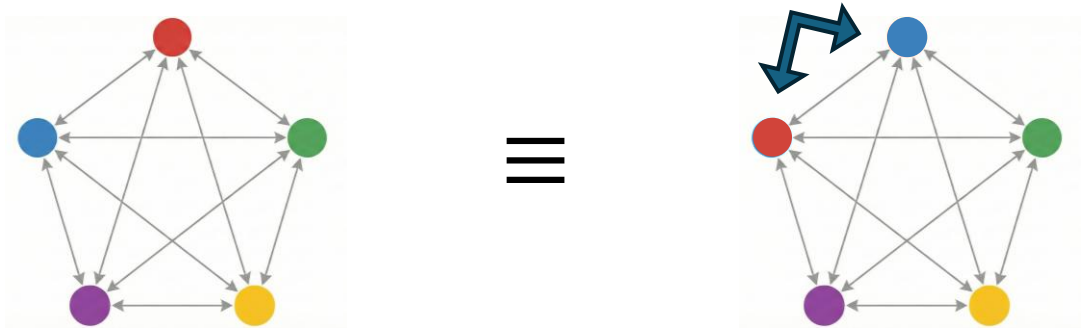
Data Structure in GNNs

Reminder: In text / images, every input has an order

- Each element ⇒ unique position



- Not in graphs!



Graph Isomorphism

Reminder: Two graphs $G = (V, E)$ and $G' = (V', E')$ are **isomorphic** if there is a bijection $\varphi : V \rightarrow V'$ such that for all nodes $u, v \in V$ it holds:

$$(u, v) \in E \Leftrightarrow \big(\varphi(u), \varphi(v)\big) \in E'$$

- Denoted: $G \cong G'$
- Assumption: Homogeneous

Optimally:

For graph-level (invariance)

$GNN(G) = GNN(G') \Leftrightarrow G \cong G'$

For node-level (equivariance)

$\exists$ permutation P s.t.

$P \cdot GNN(G) = GNN(G') \Leftrightarrow G \cong G'$

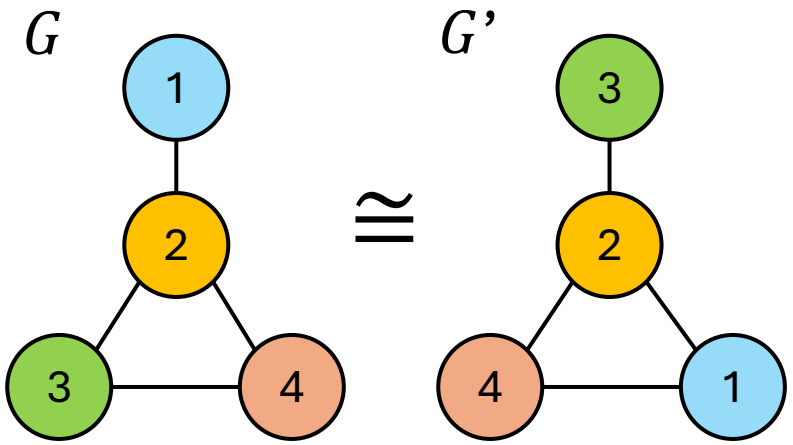
Isomorphic Graphs - Example

$G = (V, E), G' = (V', E')$

$V = V' = \{1,2,3,4\}$

$E = \{(1,2), (2,3), (3,4), (2,4)\}$

$E' = \{(3,2), (2,4), (1,4), (1,2)\}$



$\varphi: V \to V'$:

$1 \mapsto 3$
$2 \mapsto 2$
$3 \mapsto 4$
$4 \mapsto 1$

$(u, v) \in E \Leftrightarrow \big(\varphi(u), \varphi(v)\big) \in E'$:

$(1,2) \in E \Leftrightarrow \big(\varphi(1), \varphi(2)\big) \in E' \Leftrightarrow (3,2) \in E'$ ✓

…

The Graph Isomorphism Problem

So now we need to decide -

- Given two graphs – are they isomorphic?

For graph-level (invariance)

$$GNN(G) = GNN(G') \Leftrightarrow G \cong G'$$

But, how?

🔨 Open problem – no polynomial time algorithm yet no proof for NP-completeness

Weisfeiler Lehman Test

Yet, there is an algorithm for deciding isomorphism – 1-WL test:

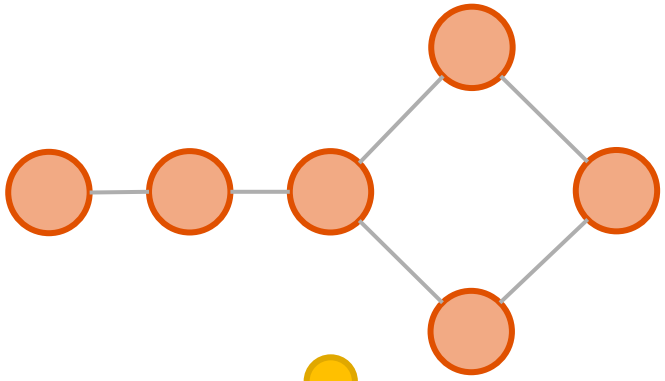
1. Initialize $\kappa_0(v) = 0$ for all $v \in V$
2. Having constructed κ_i, define κ_{i+1} as follows for all nodes $v \in V$:

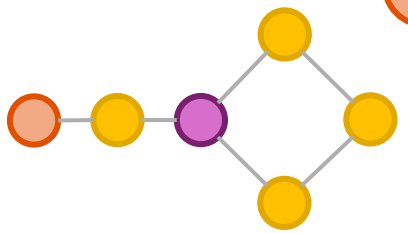
$$\kappa_{i+1}(v) = HASH(\kappa_i(v), \{\{\kappa_i(u) \mid u \in N(v)\}\})$$

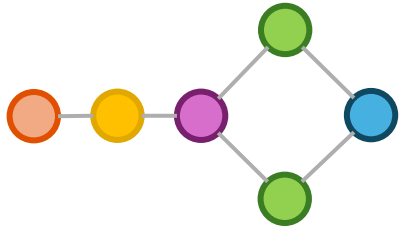
3. Stop when κ_{i+1} and κ_i are equivalent

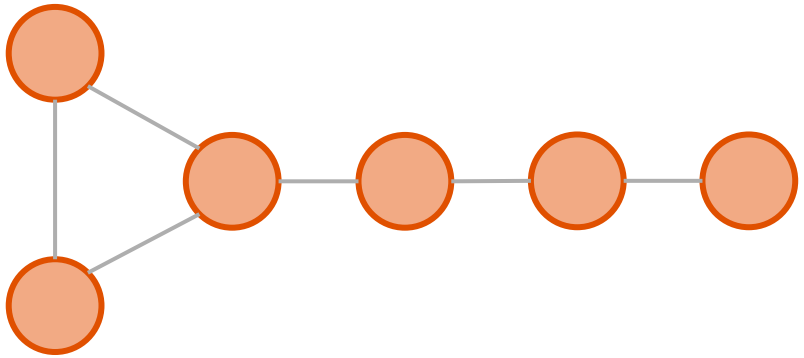
$$\text{Output } G \cong G' \Leftrightarrow 1\text{-WL}(G) = 1\text{-WL}(G')$$

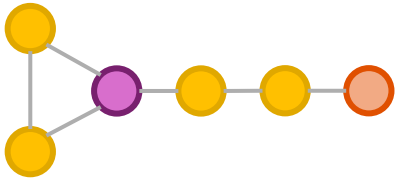
Example: Weisfeiler Lehman Test

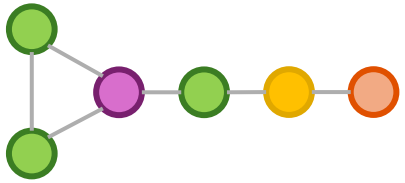


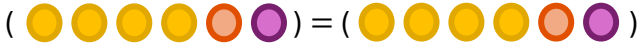


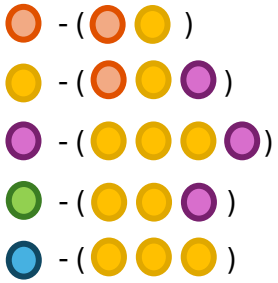


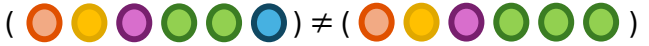






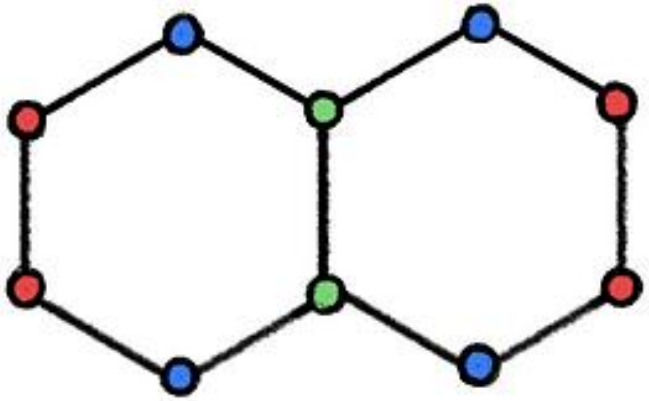


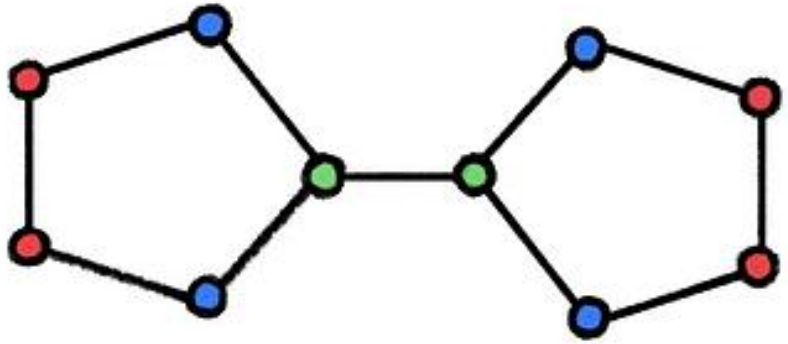


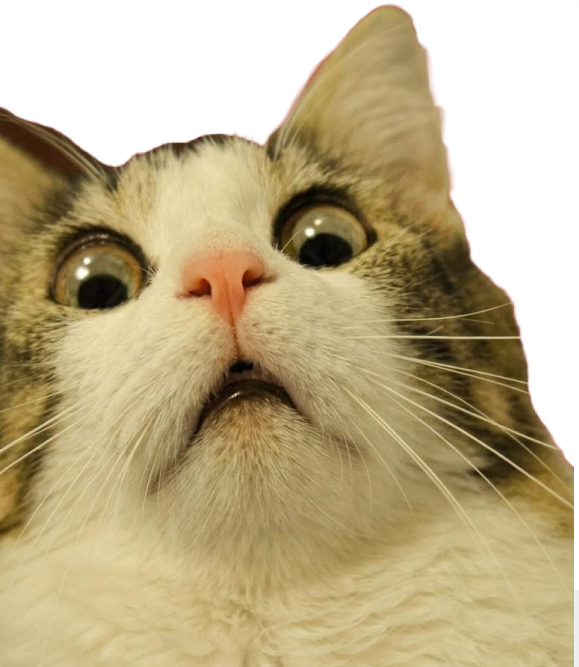


⇒ *non-isomorphic*

Example: Weisfeiler Lehman Test



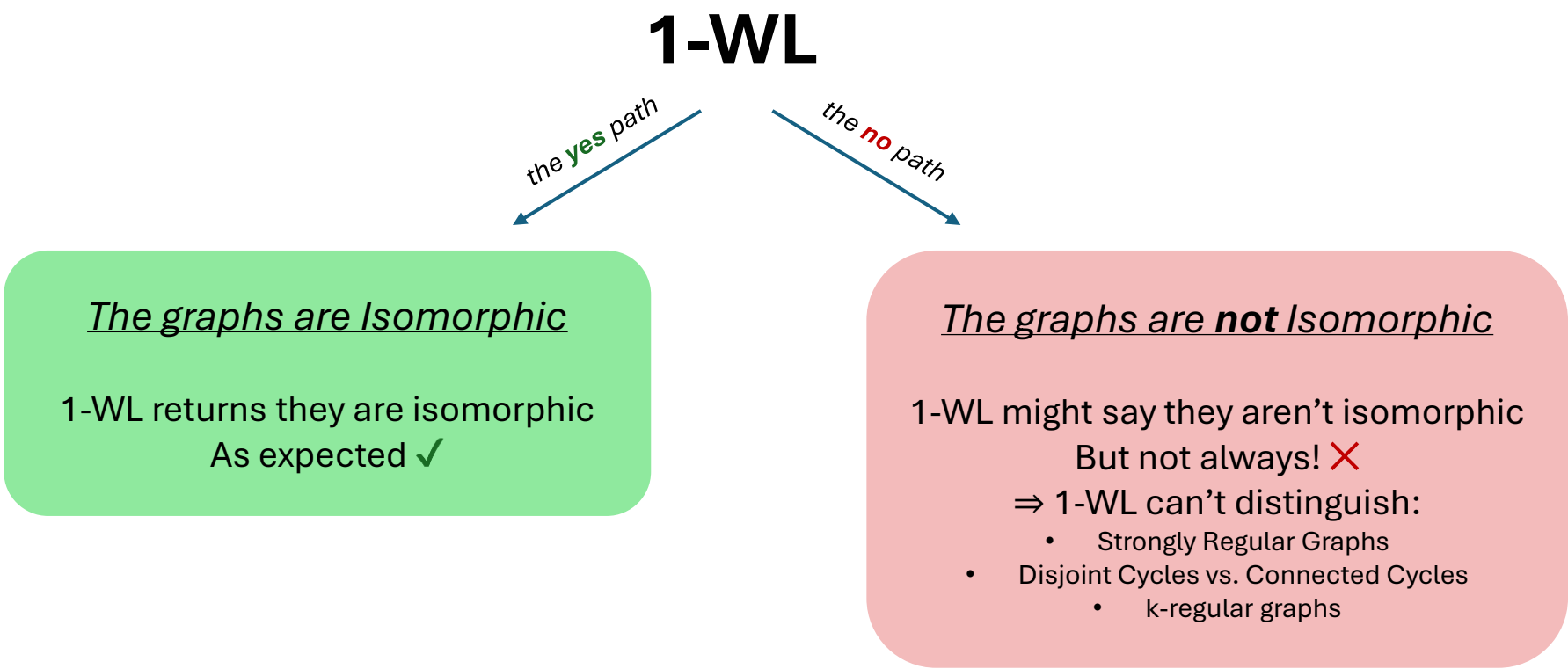




- Same 1-WL
- Non-isomorphic

https://medium.com/data-science/expressive-power-of-graph-neural-networks-and-the-weisefeiler-lehman-test-b883db3c7c49

1-WL Limitation

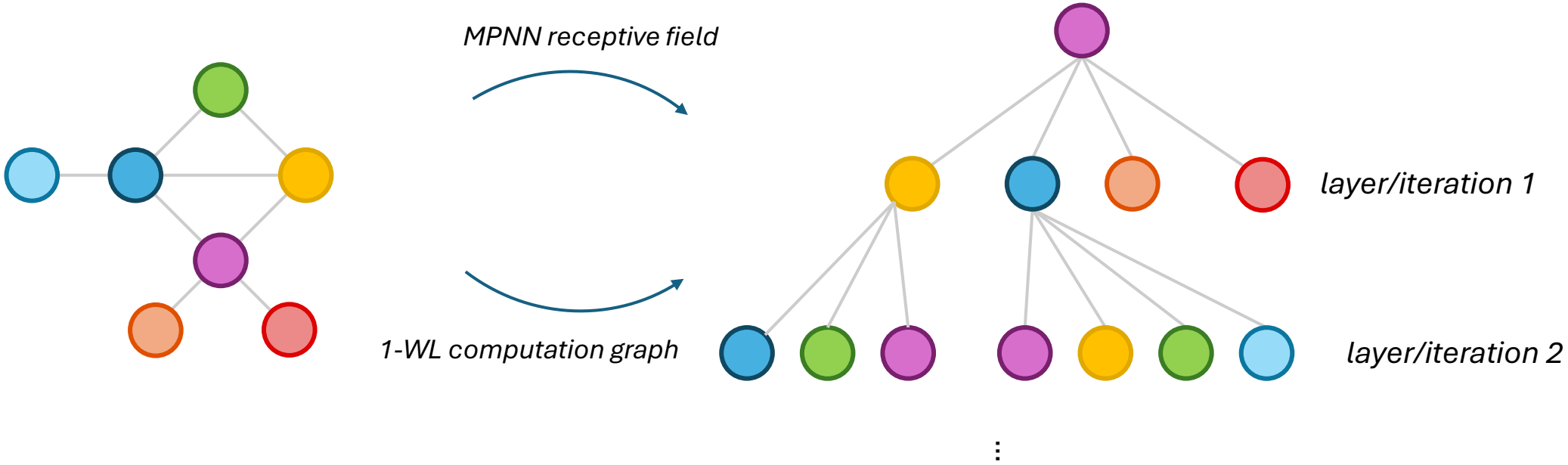


⇒ 1-WL expressivity is limited

- **Expressivity** = the ability to distinguish non-isomorphic graphs

1-WL & MPNN

Result: MPNN expressive power is limited by 1-WL expressivity



Can we reach the upper bound?

Reference: How Powerful are Graph Neural Networks?

Outline

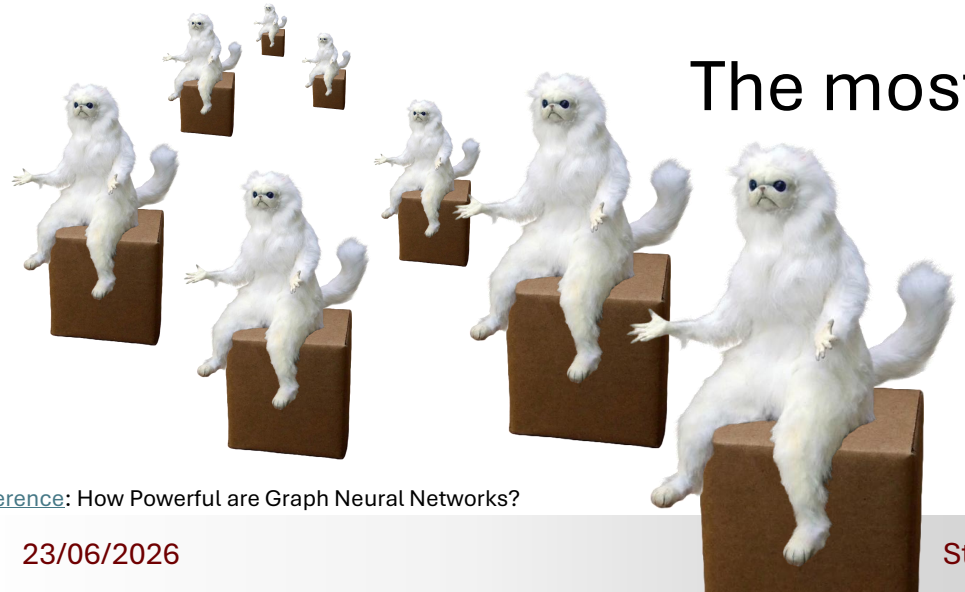
- Isomorphism & 1-WL Test
- GIN - Graph Isomorphism Network
- Compare to Known Architectures
- Beyond 1-WL

GIN

Graph Isomorphism Network (GIN):

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

- Aggregation: SUM
- $h_v^{(k)}$ - embedding of node v at layer k
- $\epsilon^{(k)}$ - learnable scalar at layer k



The most expressive 1-WL MPNN

But... Why?

Reference: How Powerful are Graph Neural Networks?

Message Passing Must Be Injective

- *Requirement*: For the MPNN expressivity to match that of 1-WL, message aggregation and update must be injective.

  Why? To match 1-WL expressivity, which follows from the hash function being injective.

Message Passing

$$h_v^{(k)} = U^{(k)}\left(h_v^{(k-1)}, \alpha^{(k)}\left(\left(h_v^{(k-1)}, \{\{h_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\}\right)\right)\right)$$

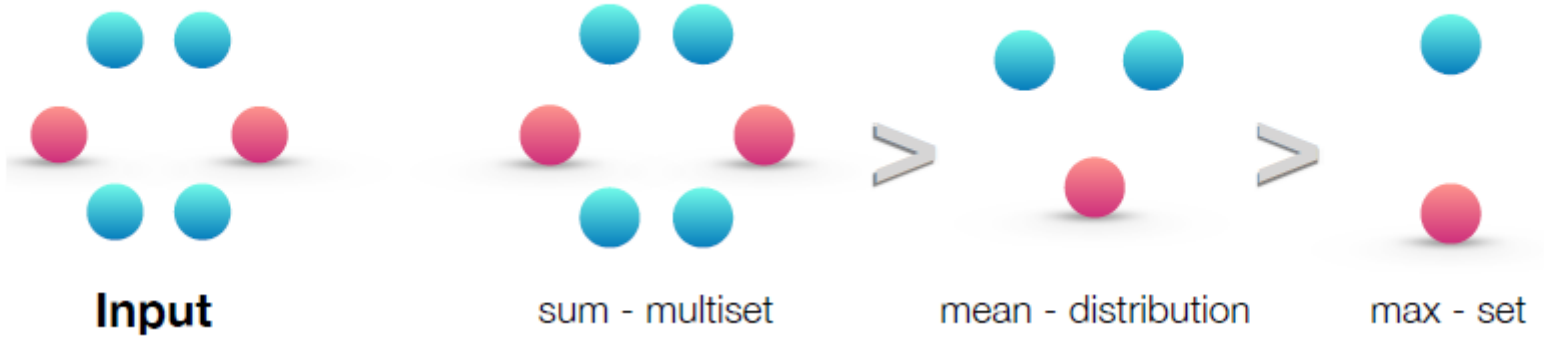
1-WL

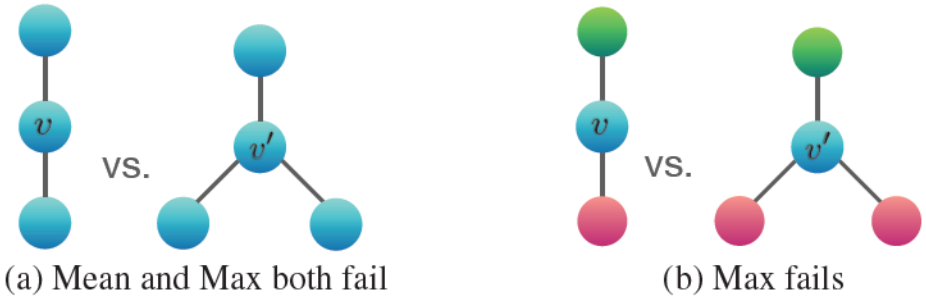
$$\kappa_{i+1}(v) = HASH(\kappa_i(v), \{\{\kappa_i(u) \mid u \in N(v)\}\})$$

injective

Types of Aggregations

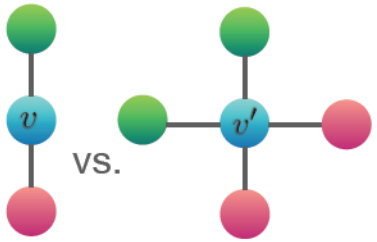
- Intuition: Why SUM is better than MEAN/MAX?





(a) Mean and Max both fail

(b) Max fails



(c) Mean and Max both fail

Reference: How Powerful are Graph Neural Networks?

Types of Aggregations

- And indeed, SUM is **injective**
- But wait, $1 + 4 = 2 + 3$. So how come it's injective?
  - We represent different values with different vectors
    - i.e – one hot encoding: $1 = [1,0,0,0]$ $2 = [0,1,0,0]$ $3 = [0,0,1,0]$ $4 = [0,0,0,1]$
    - Embeddings act the same way
  - And now $1 + 4 \neq 2 + 3$:

$$\text{"}1+4\text{"} = [[1,0,0,0]] + [[0,0,0,1]] = [[1,0,0,1]] \neq [0,1,1,0] = [0,1,0,0] + [0,0,1,0] = \text{"}2 + 3\text{"}$$

$\Rightarrow \mathrm{SUM}$ is injective, while $\mathrm{MEAN}/\mathrm{MAX}$ aren't

Reference: How Powerful are Graph Neural Networks?

The Injective Update

- The update function $U^{(k)}$ - over the aggregated messages & self node
  → should be injective as well
  - Let's just use sum again and get: $U^{(k)}\left(\mathrm{h}_u^{(k-1)}, m_u^{(k)}\right) = \mathrm{MLP}^{(k)}\left(\mathrm{h}_u^{(k-1)} + m_u^{(k)}\right)$
  - But we don't operate on multisets anymore! Sum is not enough:

$$5 + \mathrm{SUM}(\{\{2,3\}\}) = 2 + \mathrm{SUM}(\{\{5,3\}\})$$ 👎

  - So we scale the first value by $\left(1 + \epsilon^{(k)}\right)$ and get:

$$\left(1 + \epsilon^{(k)}\right) \cdot 5 + \mathrm{SUM}(\{\{2,3\}\}) \neq \left(1 + \epsilon^{(k)}\right) \cdot 2 + \mathrm{SUM}(\{\{5,3\}\})$$ 👍

GIN in total:

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

Outline

- Isomorphism & 1-WL Test
- GIN - Graph Isomorphism Network
- Compare to Known Architectures
- Beyond 1-WL

Examining known architectures

What aggregation function is used in GCN?

$$\mathbf{h_u^{(k)}} = \sigma\left(\sum_{v\in\widehat{N}(u)} \frac{1}{\sqrt{\widehat{d_u}\cdot\widehat{d_v}}}\mathbf{h_v^{(k-1)}}\mathbf{W^{(k)}}\right)$$

→ Normalized average
⇒ not injective 👎

What aggregation function is used in GraphSAGE?

$$\alpha^{(k)} = \text{MEAN/MAX/LSTM}$$

→ None is injective 👎

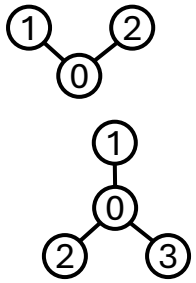
⇒ Both are ***not*** *1-WL MPNNs* (≡ reaching the 1-WL limit)

Experiment: Who Can Distinguish?

Create graphs

```
edge_index_A = torch.tensor([[1, 2, 0, 0],
                             [0, 0, 1, 2]])
x_A = torch.ones(3, 1)

edge_index_B = torch.tensor([[1, 2, 3, 0, 0, 0],
                             [0, 0, 0, 1, 2, 3]])
x_B = torch.ones(4, 1)
```



Init GCN/GraphSAGE/GIN

```
from torch_geometric.nn import GCNConv, SAGEConv, GINConv
from torch.nn import Sequential, Linear, ReLU

gcn = GCNConv(1, 16)
sage = SAGEConv(1, 16, aggr='max')
gin = GINConv(Linear(1, 16))
```

Compare outcomes

Specific node

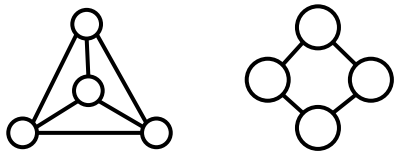
```
gcn_diff = torch.norm(gcn(x_A, edge_index_A)[0] - gcn(x_B,
edge_index_B)[0])
sage_diff = torch.norm(sage(x_A, edge_index_A)[0] - sage(x_B,
edge_index_B)[0])
gin_diff = torch.norm(gin(x_A, edge_index_A)[0] - gin(x_B,
edge_index_B)[0])

print(f"GCN (Mean) difference: {gcn_diff.item():.4f}")
print(f"SAGE (Max) difference: {sage_diff.item():.4f}")
print(f"GIN (Sum) difference: {gin_diff.item():.4f}")
```

output

```
GCN (Mean) difference: 0.2405
SAGE (Max) difference: 0.0000
GIN (Sum) difference: 0.6923
```

Experiment: Who Can Distinguish?



Create graphs

```
edges_C4 = [[0,1], [1,2], [2,3], [3,0]]
edges_C4 += [[v,u] for u,v in edges_C4]
edge_index_C4 = torch.tensor(edges_C4).t()
x_C4 = torch.ones(4, 1)

edges_K4 = [[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]
edges_K4 += [[v,u] for u,v in edges_K4]
edge_index_K4 = torch.tensor(edges_K4).t()
x_K4 = torch.ones(4, 1)
```

Init GCN/GraphSAGE/GIN

```
from torch_geometric.nn import GCNConv, SAGEConv, GINConv
from torch.nn import Sequential, Linear, ReLU

gcn = GCNConv(1, 16)
sage = SAGEConv(1, 16, aggr='mean')
gin = GINConv(Linear(1, 16))
```

Compare outcomes

Compare all nodes

```
def get_graph_emb(model, x, edge_index):
    return model(x, edge_index).sum(dim=0)
gcn_diff = torch.norm(get_graph_emb(gcn, x_C4,
edge_index_C4) - get_graph_emb(gcn, x_K4, edge_index_K4))
sage_diff = torch.norm(get_graph_emb(sage, x_C4,
edge_index_C4) - get_graph_emb(sage, x_K4, edge_index_K4))
gin_diff = torch.norm(get_graph_emb(gin, x_C4,
edge_index_C4) - get_graph_emb(gin, x_K4, edge_index_K4))

print(f"GCN diff:  {gcn_diff.item():.4f}")
print(f"SAGE diff: {mean_diff.item():.4f}")
print(f"GIN diff:  {gin_diff.item():.4f}")
```

output

```
GCN diff:  0.0000
SAGE diff: 0.0000
GIN diff:  2.9857
```

Outline

- Isomorphism & 1-WL Test
- GIN - Graph Isomorphism Network
- Compare to Known Architectures
- Beyond 1-WL

Breaking the 1-WL Limit

Can we design a GNN that breaks the 1-WL expressivity boundary?

Yes

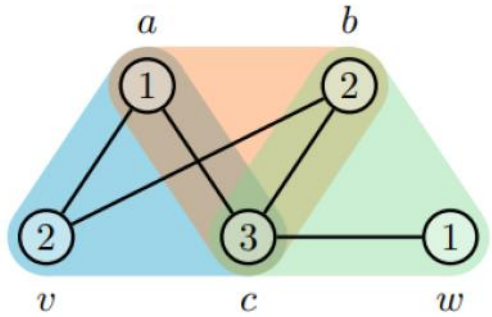
k-WL: Breaking the 1-WL Limit

- 1-WL test can be generalized into ***k*-WL**:
  - Instead of looking at each node at a time, look at a k-tuple of nodes

1. Initialize a color $\kappa_0(\mathbf{v})$ for each $\mathbf{v} = (v_1, v_2, \dots, v_k) \in V^k$ by the pattern of its induced subgraph
2. Having constructed κ_i, define κ_{i+1} as follows for all nodes $v \in V$:

$$\kappa_{i+1}(\mathbf{v}) = \mathrm{HASH}\left(\kappa_i(\mathbf{v}),\ \{\{\left(\kappa_i(\mathbf{v_{1\leftarrow w}}), \dots, \kappa_i(\mathbf{v_{k\leftarrow w}})\right) \mid w \in V\}\} \right)$$

3. Stop when κ_{i+1} and κ_i are equivalent



$k = 3$

$\Rightarrow$ Highly inefficient ($O(n^k)$) $\Rightarrow$

There are GNN architectures utilizing k-WL test for improving expressivity

- 1-2-3 GNN
- PPGN
- …

Identity-aware Graph Neural Networks (ID-GNN)

A simpler approach:

- Instead of assuming homogeneous graph – annotate the root node
- Making it heterogeneous (labeled nodes)

For each node $v \in V$

1. Extract the K-hop neighborhood of v
2. For each node u in the K-hop apply adapted message passing:

- For each layer $k = 1 \dots K$:

$$s_{u,v} = \begin{cases} 1, & u = v \\ 0, & u \neq v \end{cases}$$

$$m_{u,v}^{(k)} = \mathrm{AGG}^{(k)}\left(\{ h_{w,v}^{(k-1)} \mid w \in \mathcal{N}(u) \} \right)$$

$$h_{u,v}^{(k)} = \mathrm{UPDATE}^{(k)}\left(h_{u,v}^{(k-1)}, m_{u,v}^{(k)}, s_{u,v} \right)$$

- Final embedding: $h_u^{(K)} = h_{u,u}^{(K)}$

Expressivity:

- *Lower Bound*: $>$ 1-WL
- *Upper Bound*: $\leq$ 3-WL

Complexity:

- Time: $\mathcal{O}(N \cdot d^K)$
- Space: $\mathcal{O}(N \cdot d^K)$
  - N nodes, d avg. degree
- ID-GNN-Fast (Approx. version):
  - Time: $\mathcal{O}(E)$
  - Space: $\mathcal{O}(N)$

Bypass: Structural encodings

💡 *Idea*: Instead of increasing expressivity through architectural modifications meant for *discovering* structures, just hand over those structures through input node features

- We make the graph more heterogeneous using two types of features:

Positional Encodings

Features capturing the spatial position of the node in the Euclidean space
⇒ i.e: **embedding**

Structural Encodings

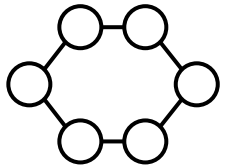
Features capturing the local topology and neighbourhood connectivity of the node within the graph
⇒ i.e: **GDV** (e.g., triangles count, etc)

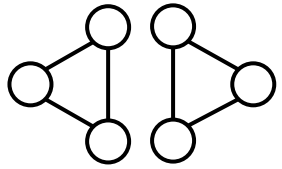
Distinguishing Code

Create Graphs

```
edges_hex = [[0,1], [1,2], [2,3], [3,4], [4,5], [5,0]]
edges_hex += [[v, u] for u, v in edges_hex]
edge_index_hex = torch.tensor(edges_hex).t()
x_hex = torch.ones(6, 1)

edges_tri = [[0,1], [1,2], [2,0], [3,4], [4,5], [5,3]]
edges_tri += [[v, u] for u, v in edges_tri]
edge_index_tri = torch.tensor(edges_tri).t()
x_tri = torch.ones(6, 1)
```





Define GIN model

```
class SimpleGIN(torch.nn.Module):
    def __init__(self, in_dim):
        super().__init__()
        self.conv1 = GINConv(Sequential(Linear(in_dim, 16), ReLU(), Linear(16, 16)))
        self.conv2 = GINConv(Sequential(Linear(16, 16), ReLU(), Linear(16, 16)))
        self.conv3 = GINConv(Sequential(Linear(16, 16), ReLU(), Linear(16, 16)))

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = self.conv2(x, edge_index)
        x = self.conv3(x, edge_index)
        return x
```

Run with simple GIN

```
model_std = SimpleGIN(in_dim=1)
std_diff = torch.norm(model_std(x_hex, edge_index_hex)[0] -
model_std(x_tri, edge_index_tri)[0])
print(f"Standard GIN diff (Hexagon vs Triangle):
{std_diff.item():.4f}")
```

output

```
Standard GIN diff (Hexagon vs Triangle): 0.0000
```

👎

Run with structural encodings

```
se_feature_hex = torch.zeros(6, 1)
x_hex_se = torch.cat([x_hex, se_feature_hex], dim=1)
se_feature_tri = torch.ones(6, 1)
x_tri_se = torch.cat([x_tri, se_feature_tri], dim=1)
model_se = SimpleGIN(in_dim=2)
se_diff = torch.norm(model_se(x_hex_se, edge_index_hex)[0] -
model_se(x_tri_se, edge_index_tri)[0])
print(f"PE/SE diff (Hexagon vs Triangle):       {se_diff.item():.4f}")
```

output

```
PE/SE diff (Hexagon vs Triangle):       0.7082
```

👍

Summary

Expressivity of GNNs

Key Measurement: 1-WL Test