Introduction to Graph Neural Networks

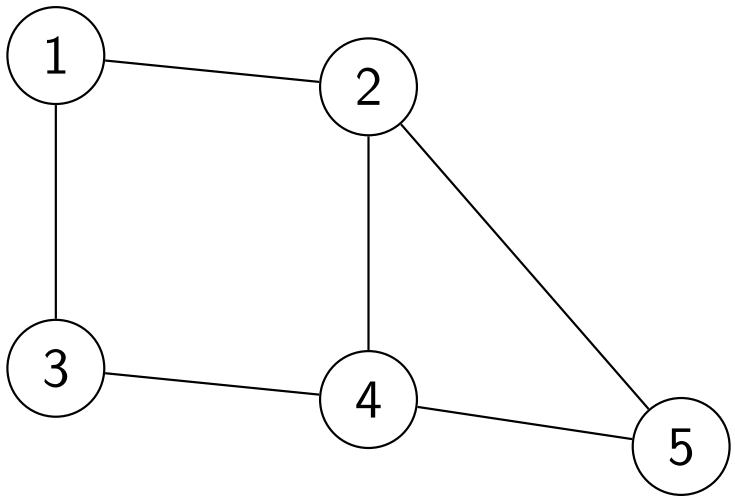
Giannis Nikolentzos

University of Peloponnese, Greece

HIAS Summer School in AI
July 1, 2024

What is a Graph?

A graph is a mathematical structure used to represent a set of **objects** and their **relationships**.



Graph $G = (V, E)$

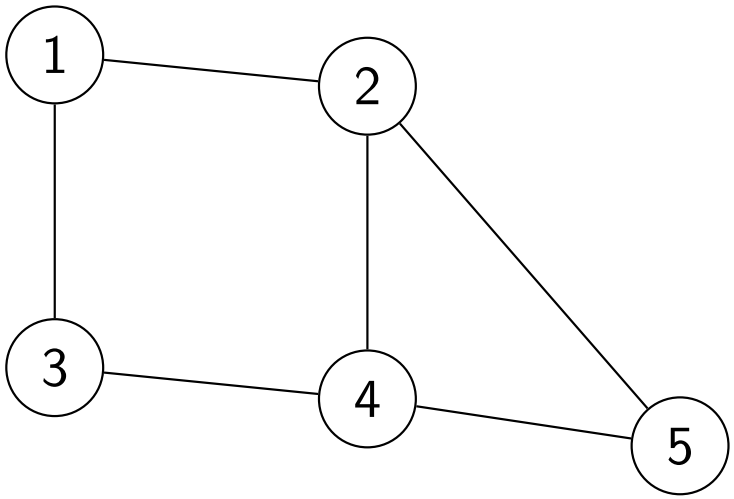
$V = \{1, 2, 3, 4, 5\}$

$E = \{(1, 2), (1, 3), (2, 4), (3, 4), (2, 5), (4, 5)\}$

How do we Represent Graphs on a Computer?

Adjacency matrix $\mathbf{A}$:

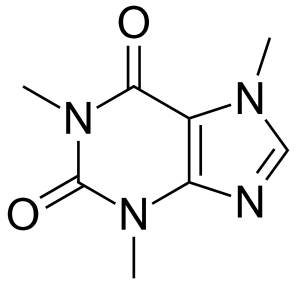
- an $n \times n$ matrix where n is the number of nodes of the graph
- if the i-th and j-th node of the graph are connected by an edge, then $\mathbf{A}_{i,j} = 1$, otherwise $\mathbf{A}_{i,j} = 0$



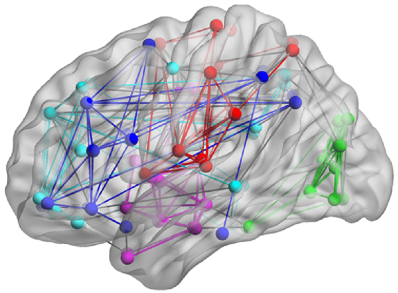
$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Machine Learning on Graphs

- Graphs are everywhere!!



Molecule



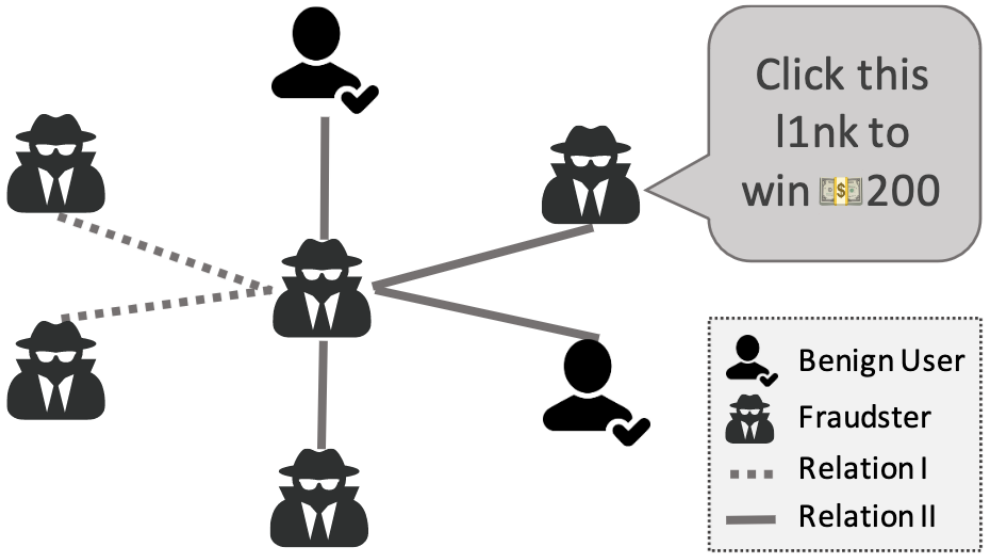
Brain network

Social network

- Many problems cannot be solved by conventional techniques
  - Need for machine learning algorithms
- Common learning tasks:
  - Node-level tasks:
    - node classification
    - node regression
  - Graph-level tasks:
    - graph classification
    - graph regression
  - Other tasks:
    - link prediction
    - community detection

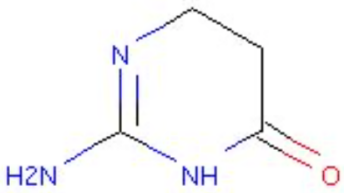
Motivation - Fraud Detection in Social Networks

Perform **node classification** to predict whether a user is fraudster or not **[Dou et al., CIKM'20]**

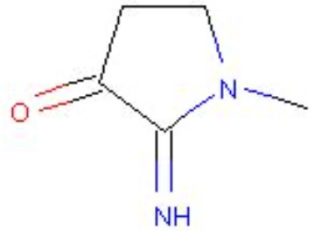


Motivation - Molecular Property Prediction

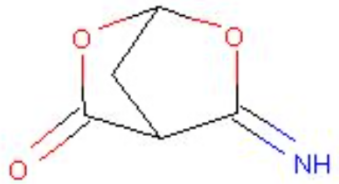
12 targets corresponding to molecular properties: ['mu', 'alpha', 'HOMO', 'LUMO', 'gap', 'R2', 'ZPVE', 'U0', 'U', 'H', 'G', 'Cv']



SMILES: NC1=NCCC(=O)N1
Targets: [2.54 64.1 -0.236 -2.79e-03 2.34e-01 900.7 0.12 -396.0 -396.0 -396.0 -396.0 26.9]



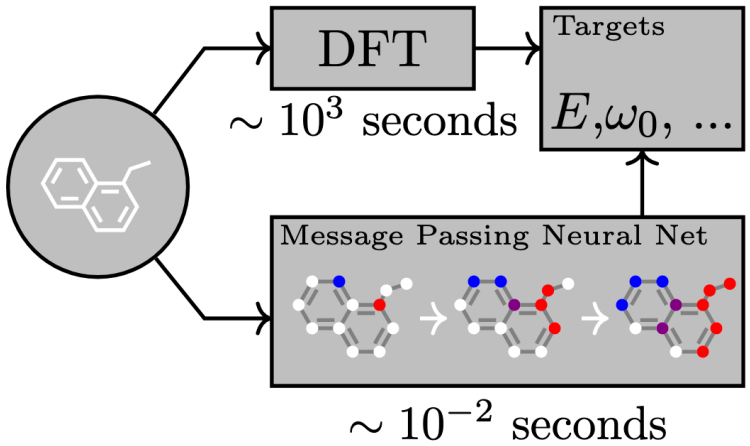
SMILES: CN1CCC(=O)C1=N
Targets: [4.218 68.69 -0.224 -0.056 0.168 914.65 0.131 -379.959 -379.951 -379.95 -379.992 27.934]



SMILES: N=C1OC2CC1C(=O)O2
Targets: [4.274 61.94 -0.282 -0.026 0.256 887.402 0.104 -473.876 -473.87 -473.869 -473.907 24.823]

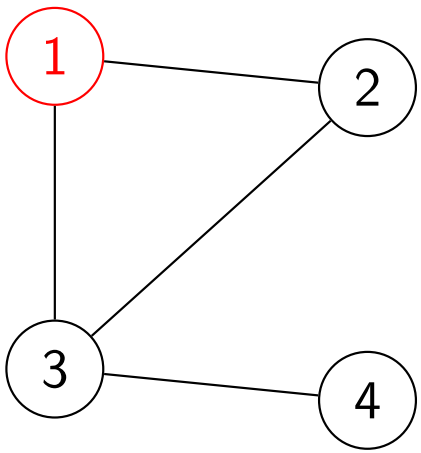
SMILES: C1N2C3C4C5OC13C2C5
Targets: [? ? ? ? ? ? ? ? ? ? ? ?]

Perform **graph regression** to predict the values of the properties **[Gilmer et al., ICML'17]**



Can we Solve Node-Level Tasks with Standard Architectures?

- Represent each node by the corresponding row of the adjacency matrix
- Feed the vectors to an MLP



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

☹ These vectors explicitly capture only **first-order proximity**!

Can we Solve Graph-Level Tasks with Standard Architectures?

- We can transform the adjacency matrix into a vector (by concatenating its rows) and feed the vectors to an MLP
- Or treat the adjacency matrix as an image and feed it to a CNN
- Or represent the graph as a sequence of nodes and feed the rows of the adacency matrix to an RNN
- But...

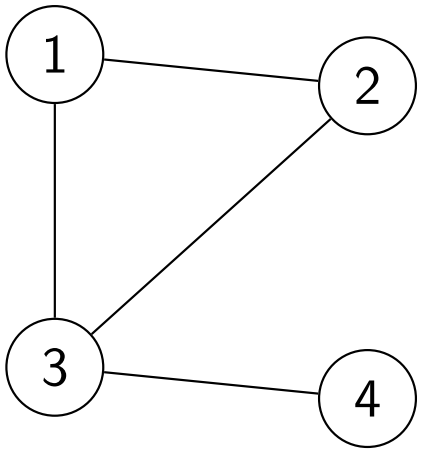
Can we Solve Graph-Level Tasks with Standard Architectures?

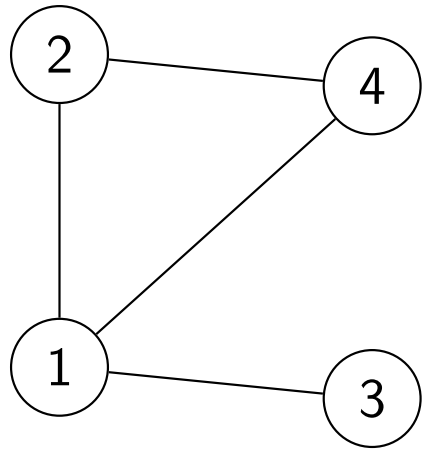
- We can transform the adjacency matrix into a vector (by concatenating its rows) and feed the vectors to an MLP
- Or treat the adjacency matrix as an image and feed it to a CNN
- Or represent the graph as a sequence of nodes and feed the rows of the adacency matrix to an RNN
- But...
- **Permutations** of the adjacency matrix (i.e., reorderings of the nodes) represent the same graph
- Thus, model output needs to be the **same for all permutations** of the adjacency matrix

Can we Solve Graph-Level Tasks with Standard Architectures?

For example, the next two adjacency matrices represent the same graph

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





Two graphs G_1 and G_2 are **isomorphic** if there exists a bijection f between their nodes such that there is an edge between nodes v and u in G_1 if and only if there is an edge between nodes $f(v)$ and $f(u)$ in G_2

Message Passing Neural Networks

- Consist of a series of message passing layers
- Within each layer, the representation of each node $\mathbf{h}_v^{(t)}$ is updated based on its previous representation and the representations of its neighbors:

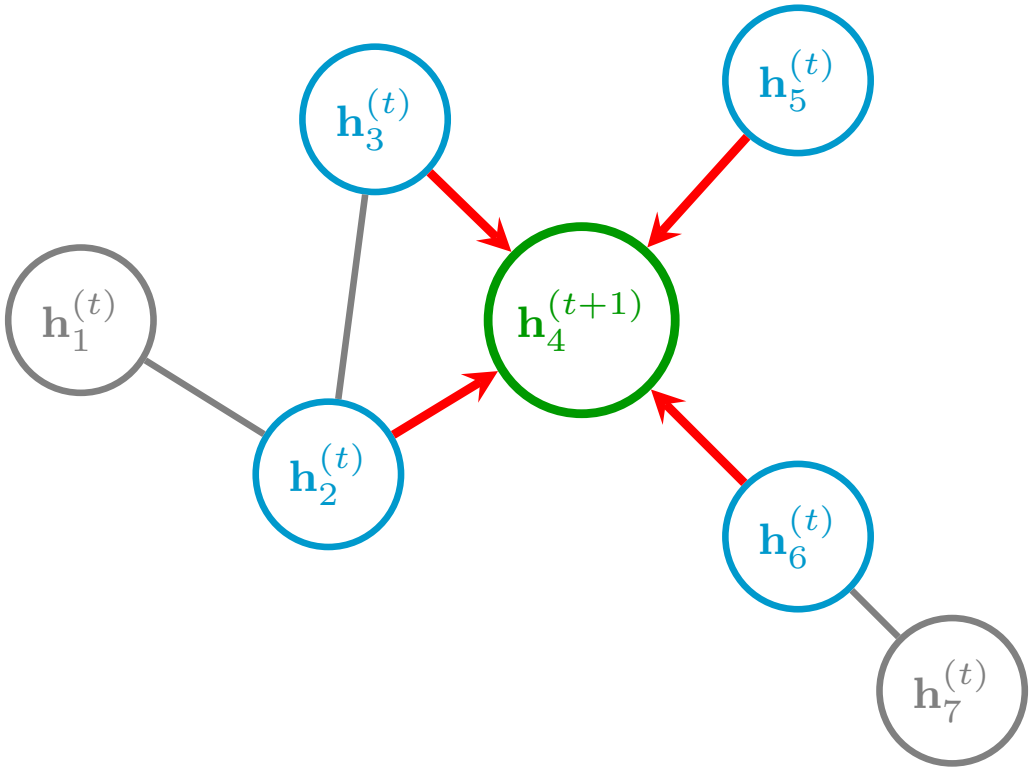
$$\mathbf{m}_v^{(t+1)} = \mathrm{AGGREGATE}\left(\left\{\!\left\{\mathbf{h}_u^{(t)} \middle| u \in \mathcal{N}(v)\right\}\!\right\}\right)$$

$$\mathbf{h}_v^{(t+1)} = \mathrm{COMBINE}\left(\mathbf{h}_v^{(t)}, \mathbf{m}_v^{(t+1)}\right)$$

  where $\mathcal{N}(v)$ is the set of neighbors of v, and $\mathrm{AGGREGATE}$ and $\mathrm{COMBINE}$ are message functions and node update functions respectively

- * a node's neighbors have **no natural ordering**
  - the $\mathrm{AGGREGATE}$ function operates over an unordered multiset of vectors $\rightarrow$ must be invariant to permutations of the neighbors
- Representations of last layer $\mathbf{h}_v^{(T)}$ typically followed by one or more fully-connected layers

Example of Message Passing



$$\mathbf{m}_4^{(t+1)} = \text{AGGREGATE}\left(\left\{\!\!\left\{\mathbf{h}_2^{(t)}, \mathbf{h}_3^{(t)}, \mathbf{h}_5^{(t)}, \mathbf{h}_6^{(t)}\right\}\!\!\right\}\right)$$

$$\mathbf{h}_4^{(t+1)} = \text{COMBINE}\left(\mathbf{h}_4^{(t)}, \mathbf{m}_4^{(t+1)}\right)$$

Graph Convolutional Network (GCN)

Each message passing layer of the GCN model **[Kipf and Welling, ICLR'17]** is defined as follows:

$$\mathbf{h}_v^{(t+1)} = \mathrm{ReLU}\left(\mathbf{W}^{(t)} \frac{1}{1+d(v)} \mathbf{h}_v^{(t)} + \sum_{u \in \mathcal{N}(v)} \mathbf{W}^{(t)} \frac{1}{\sqrt{(1+d(v))(1+d(u))}} \mathbf{h}_u^{(t)}\right)$$

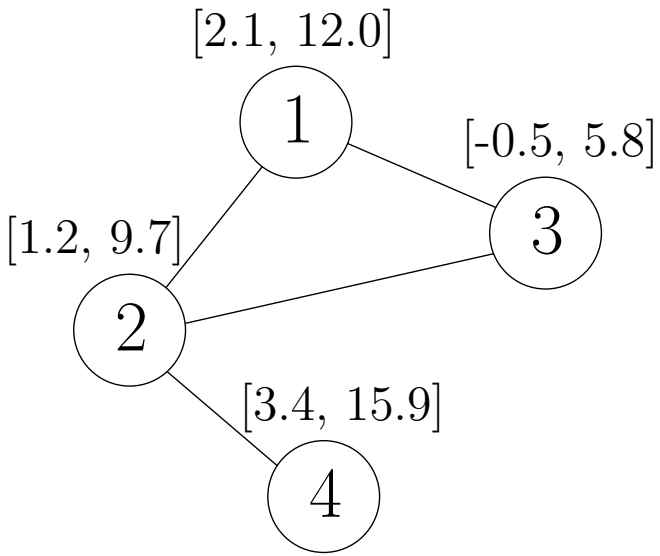
where $d(v)$ is the degree of node v

In matrix form, the above is equivalent to:

$$\mathbf{H}^{(t+1)} = \mathrm{ReLU}\left(\hat{\mathbf{A}}\, \mathbf{H}^{(t)}\, \mathbf{W}^{(t)}\right)$$

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\, \tilde{\mathbf{A}}\, \tilde{\mathbf{D}}^{-\frac{1}{2}}$, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ is a diagonal matrix such that $\tilde{\mathbf{D}}_{ii} = \sum_{j=1}^{n} \tilde{\mathbf{A}}_{ij}$

Example of Message Passing Layer of GCN (1/2)



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

$$\mathbf{X} = \begin{bmatrix} 2.1 & 12.0 \\ 1.2 & 9.7 \\ -0.5 & 5.8 \\ 3.4 & 15.9 \end{bmatrix}$$

We compute matrices $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{D}}$:

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

$$\tilde{\mathbf{D}} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Example of Message Passing Layer of GCN (2/2)

And then matrix $\hat{\mathbf{A}}$:

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} = \begin{bmatrix} 0.333 & 0.288 & 0.333 & 0 \\ 0.288 & 0.25 & 0.288 & 0.353 \\ 0.333 & 0.288 & 0.333 & 0 \\ 0 & 0.353 & 0 & 0.5 \end{bmatrix}$$

The parameters of the message passing layer are as follows:

$$\mathbf{W} = \begin{bmatrix} 1.064 & 0.211 & -0.557 \\ -1.282 & 0.614 & 0.996 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -1.177 & -0.540 & 1.331 \end{bmatrix}$$

The representations of the first message passing layer are computed as follows:

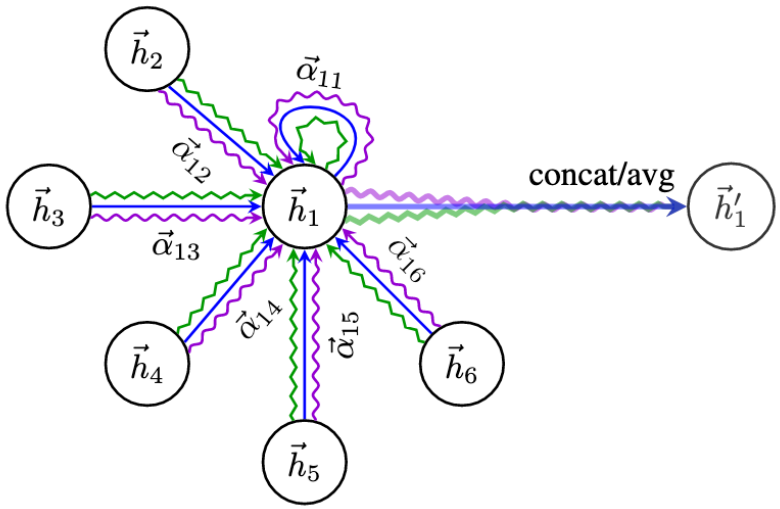
$$\mathbf{H} = \textsc{ReLU}\Big(\hat{\mathbf{A}}\big(\mathbf{X}\mathbf{W} + \mathbf{b}\big)\Big) = \begin{bmatrix} 0 & 5.024 & 9.466 \\ 0 & 7.859 & 13.588 \\ 0 & 5.024 & 9.466 \\ 0 & 6.971 & 11.281 \end{bmatrix}$$

Graph Attention Network (GAT)

- Messages from some neighbors may be **more important** than messages from others!!
- GAT applies self-attention on the nodes **[Veličković et al., ICLR'18]**
- For nodes $v_j \in \mathcal{N}(v_i)$, computes attention coefficients that indicate the importance of node v_j's features to node v_i:

$$\alpha_{ij}^{(t)} = \frac{\exp\Big(\text{LeakyReLU}\big(\mathbf{a}^\top[\mathbf{W}^{(t)}\mathbf{h}_i^{(t)}||\mathbf{W}^{(t)}\mathbf{h}_j^{(t)}]\big)\Big)}{\sum_{k\in\mathbf{N}_i}\exp\Big(\text{LeakyReLU}\big(\mathbf{a}^\top[\mathbf{W}^{(t)}\mathbf{h}_i^{(t)}||\mathbf{W}^{(t)}\mathbf{h}_k^{(t)}]\big)\Big)}$$

where $[\cdot||\cdot]$ denotes concatenation of two vectors and $\mathbf{a}$ is a trainable vector



Graph Attention Network (GAT)

Then the representations of the nodes are updated as follows:

$$\mathbf{h}_i^{(t+1)} = \sigma\Big(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(t)} \mathbf{W}^{(t)} \mathbf{h}_j^{(t)} \Big)$$

In matrix form, the above is equivalent to:

$$\mathbf{H}^{(t+1)} = \sigma\Big(\big(\mathbf{A} \odot \mathbf{T}^{(t)} \big) \mathbf{H}^{(t)} \mathbf{W}^{(t)} \Big)$$

where $\odot$ denotes elementwise product and is matrix such that $\mathbf{T}_{ij}^{(t)} = \alpha_{ij}^{(t)}$

More than one attention mechanisms can be employed by concatenating/averaging their respective node representations:

$$\mathbf{h}_i^{(t+1)} = \sigma\Big(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} [\alpha_k^{(t)}]_{ij} \mathbf{W}_k^{(t)} \mathbf{h}_j^{(t)} \Big)$$

where $[\alpha_k^{(t)}]_{ij}$ are the attention coefficients computed by the k^{th} attention mechanism, and $\mathbf{W}_k^{(t)}$ is the corresponding weight matrix

Can we use Message Passing Neural Networks to Compute Graph Representations?

We can utilize a **readout function**!

Step 1: Within each message passing layer, the representation of each node $\mathbf{h}_v^{(t)}$ is updated based on its previous representation and the representations of its neighbors:

$$\mathbf{m}_v^{(t+1)} = \mathrm{AGGREGATE}\left(\left\{\!\!\left\{\mathbf{h}_u^{(t)} \middle| u \in \mathcal{N}(v)\right\}\!\!\right\}\right)$$

$$\mathbf{h}_v^{(t+1)} = \mathrm{COMBINE}\left(\mathbf{h}_v^{(t)}, \mathbf{m}_v^{(t+1)}\right)$$

where $\mathcal{N}(v)$ is the set of neighbors of v, and $\mathrm{AGGREGATE}$ and $\mathrm{COMBINE}$ are message functions and node update functions respectively

Step 2: The readout step computes a feature vector for the entire graph using some permutation invariant readout function $\mathrm{READOUT}$:

$$\mathbf{h}_G = \mathrm{READOUT}\left(\left\{\!\!\left\{\mathbf{h}_v^{(T)} \middle| v \in V\right\}\!\!\right\}\right)$$

How Can we Build Message Passing Neural Networks for Learning Graph Representations?

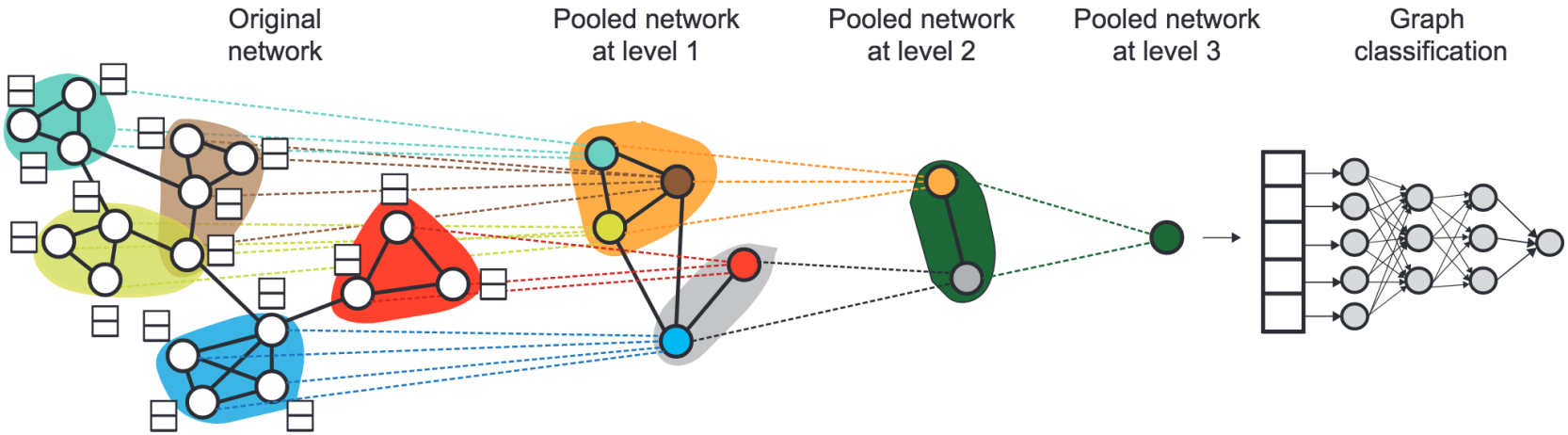
1. Take a message passing neural network that can produce node representations
2. Add a **readout function** to the model. Examples of functions:
   - sum aggerator: computes the sum of the representations of the nodes of the graph
   $$\mathbf{h}_G = \sum_{v \in V} \mathbf{h}_v^{(T)}$$
   - mean aggerator: computes the sum of the representations of the nodes of the graph
   $$\mathbf{h}_G = \frac{1}{n} \sum_{v \in V} \mathbf{h}_v^{(T)}$$
   - max aggerator: an elementwise max-pooling operation is applied to the representations of the nodes of the graph
   $$\mathbf{h}_G = \max \left(\left\{ \left\{ \mathbf{h}_v^{(T)} \right) \middle| v \in V \right\} \right)$$
   where $\max$ denotes the elementwise max operator

Differentiable Graph Pooling

The DiffPool model **[Ying et al, NIPS'18]**:

- learns **hierarchical pooling** analogous to CNNs
- sets of nodes are pooled hierarchically
- soft assignment of nodes to next-level nodes

A different GNN is learned at every level of abstraction



Differentiable Graph Pooling

- Each DiffPool layer coarsens the input graph:

$$\mathbf{X}^{(t+1)} = \mathbf{S}^{(t)^\top} \mathbf{Z}^{(t)}$$
$$\mathbf{A}^{(t+1)} = \mathbf{S}^{(t)^\top} \mathbf{A}^{(t)} \mathbf{S}^{(t)}$$

  where $\mathbf{A}^{(t+1)}$ is the coarsened adjacency matrix, and $\mathbf{X}^{(t+1)}$ is a matrix of embeddings for each node/cluster

- Matrix $\mathbf{S}^{(t)} \in \mathbb{R}^{n_t \times n_{t+1}}$ provides a soft assignment of each node at layer t to a cluster in the next coarsened layer $t+1$

- The assignment and embedding matrices are generated by two separate message passing neural networks:

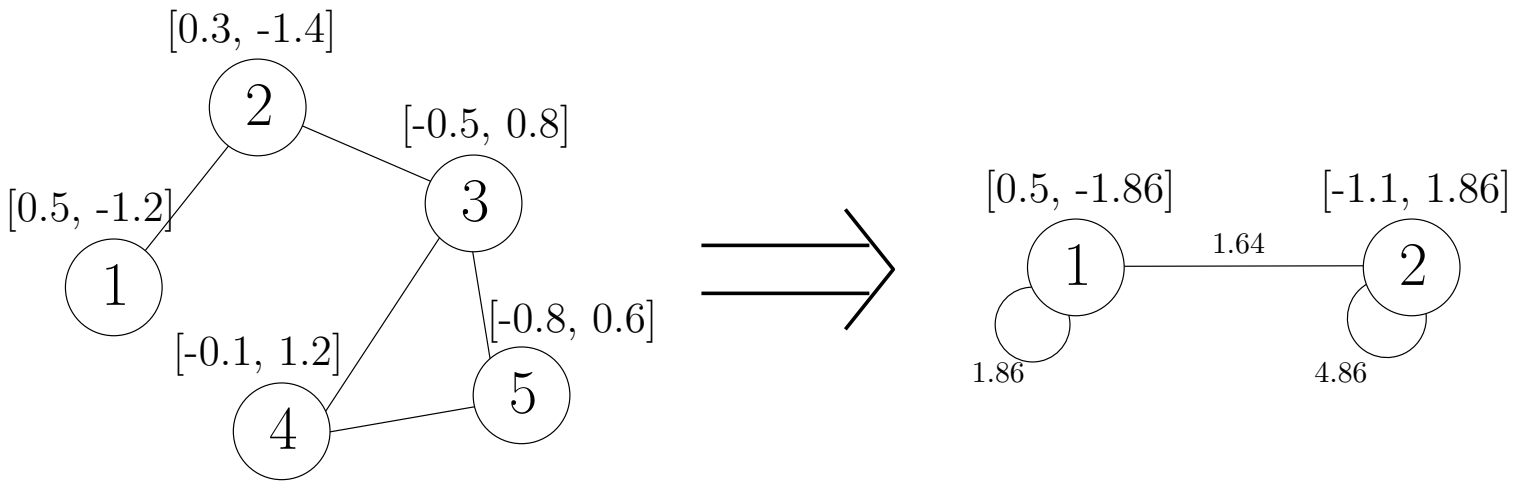
$$\mathbf{Z}^{(t)} = \mathsf{GNN}^{(t)}_{\mathsf{embed}}(\mathbf{A}^{(t)}, \mathbf{X}^{(t)})$$
$$\mathbf{S}^{(t)} = \mathsf{softmax}\big(\mathsf{GNN}^{(t)}_{\mathsf{pool}}(\mathbf{A}^{(t)}, \mathbf{X}^{(t)})\big)$$

  where the softmax function is applied in a row-wise fashion

Example of Coarsening Procedure of DiffPool

$$\mathbf{A}^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \mathbf{Z}^{(1)} = \begin{bmatrix} 0.5 & -1.2 \\ 0.3 & -1.4 \\ -0.5 & 0.8 \\ -0.1 & 1.2 \\ -0.8 & 0.6 \end{bmatrix} \quad \mathbf{S}^{(1)} = \begin{bmatrix} 0.9 & 0.1 \\ 0.8 & 0.2 \\ 0.2 & 0.8 \\ 0.1 & 0.9 \\ 0.1 & 0.9 \end{bmatrix}$$

$$\mathbf{X}^{(2)} = \mathbf{S}^{(1)\top}\mathbf{Z}^{(1)} = \begin{bmatrix} 0.5 & -1.86 \\ -1.1 & 1.86 \end{bmatrix} \quad \mathbf{A}^{(2)} = \mathbf{S}^{(1)\top}\mathbf{A}^{(1)}\mathbf{S}^{(1)} = \begin{bmatrix} 1.86 & 1.64 \\ 1.64 & 4.86 \end{bmatrix}$$



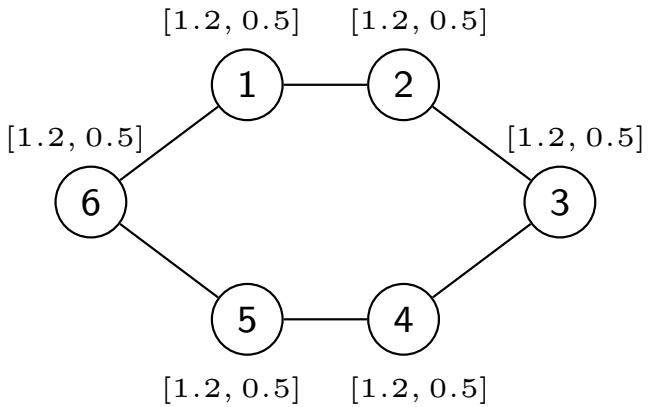
How Powerful are Message Passing Neural Networks?

- Can message passing models map all non-isomorphic graphs to different representations?

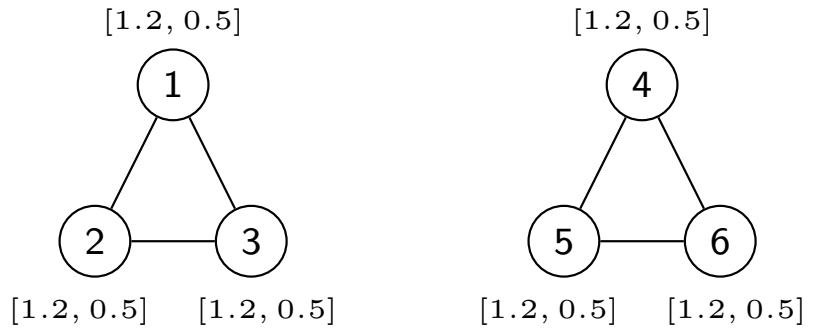
How Powerful are Message Passing Neural Networks?

- Can message passing models map all non-isomorphic graphs to different representations?

- Consider the following two graphs:



G_1



G_2

- All standard message passing models will map G_1 and G_2 to the **same vector**!!

- Those models are at most as powerful as the Weisfeiler-Leman (WL) test of isomorphism **[Morris et al., AAAI'19; Xu et al., ICLR'19; Nikolentzos et al., Neural Networks 130]**

Which Model is Equally Powerful to the WL Test?

The $\mathrm{AGGREGATE}$, $\mathrm{COMBINE}$ and $\mathrm{READOUT}$ functions of a message passing model are injective $\Rightarrow$ The model is as powerful as the WL test

The $\mathrm{AGGREGATE}$ and $\mathrm{READOUT}$ functions operate on multisets of node representations

Question: Are commonly-employed $\mathrm{AGGREGATE}$ and $\mathrm{READOUT}$ functions injective or not?

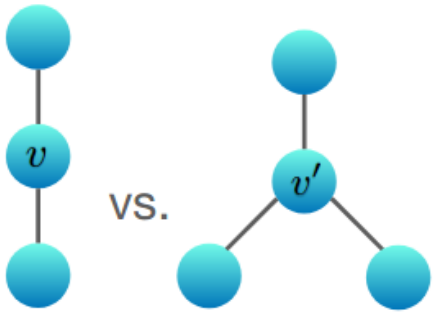
Which Model is Equally Powerful to the WL Test?

The $\mathrm{AGGREGATE}$, $\mathrm{COMBINE}$ and $\mathrm{READOUT}$ functions of a message passing model are injective $\Rightarrow$ The model is as powerful as the WL test

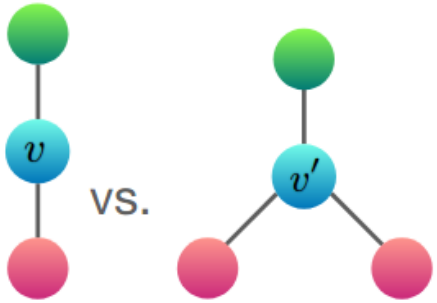
The $\mathrm{AGGREGATE}$ and $\mathrm{READOUT}$ functions operate on multisets of node representations

Question: Are commonly-employed $\mathrm{AGGREGATE}$ and $\mathrm{READOUT}$ functions injective or not?

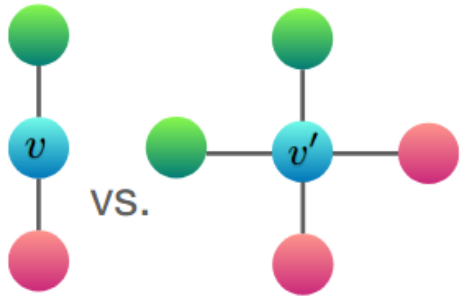
Turns out that mean and max functions are **not** injective!



(a) Mean and Max both fail



(b) Max fails



(c) Mean and Max both fail

On the other hand, **sum aggregators** can represent **injective**, in fact, universal functions over multisets

Graph Isomorphism Network (GIN)

GIN is a message passing neural network that **[Xu et al., ICLR'19]**:

- models injective multiset functions for the neighborhood and node aggregation
- has the same power as the Weisfeiler-Lehman test

Step 1: GIN updates node representations as follows:

$$\mathbf{h}_v^{(t+1)} = \mathsf{MLP}^{(t)}\left(\left(1+\epsilon^{(t)}\right)\mathbf{h}_v^{(t)} + \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(t)}\right)$$

where $\epsilon^{(t)}$ is a learnable scalar

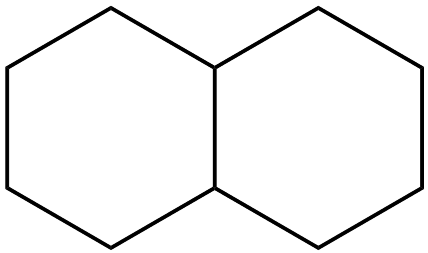
Step 2: Utilizes the following graph-level readout function which uses information from all iterations of the model:

$$\mathbf{h}_G = \sum_{v \in G} \mathbf{h}_v^{(T)}$$

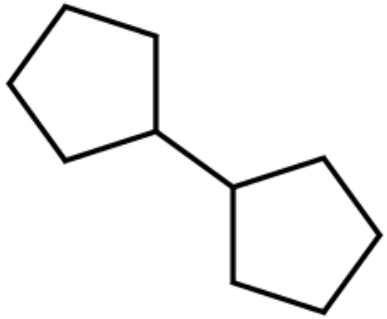
Why Do We Care About the Expressive Power?

- Non-isomorphic graphs that are not distinguished by a model are mapped to the same feature vector!!
- Therefore, there are cases where the model **cannot assign different labels to different graphs**
- Consider, for example, the following two chemical compounds

Decalin



Bicyclopentyl



- The above two compounds cannot be distinguished by the WL algorithm ☹ GNNs that are not more powerful than WL cannot embed the two compounds into different representations

How can we Build More Powerful Graph Models?

A very active field of research!

There are models that:

- perform message passing between subsets of nodes (instead of nodes)
  - k-GNN **[Morris et al., AAAI'19]**
- extract and process subgraphs
  - k-hop **[Nikolentzos et al., Neural Networks 130]**
  - node-deleted subgraphs **[Cotta et al., NeurIPS'21]**
- use extended neighborhoods
  - paths emanating from nodes **[Michel et al., ICML'23]**
- consider all possible permutations of nodes
  - RelationalPooling **[Murphy et al., ICML'19]**
  - CLIP **[Dasoulas et al., IJCAI'20]**
- utilize invariant and equivariant linear layers
  - k-order graph networks **[Maron et al., ICLR'19]**

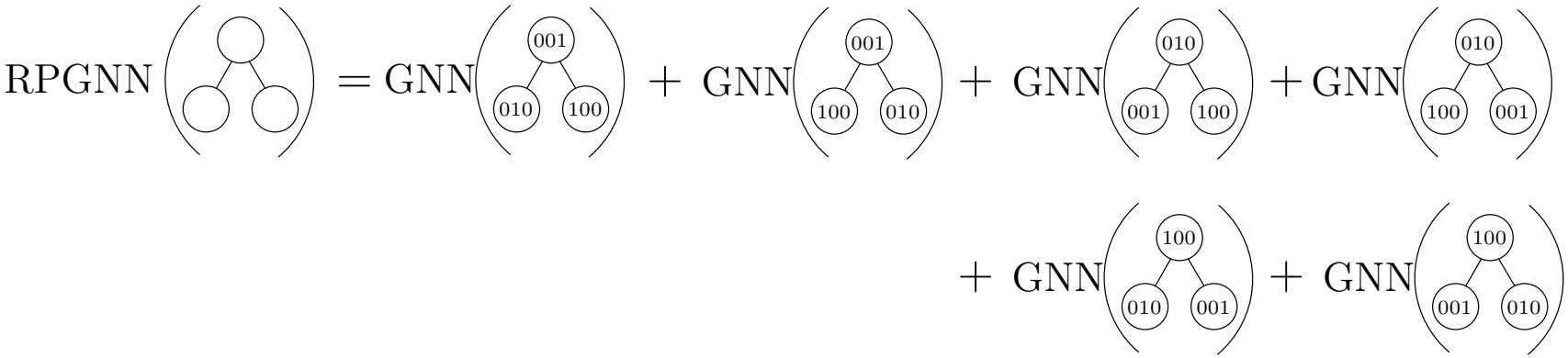
Relational Pooling

Idea: increase a model's expressive power by considering **all possible permutations** of nodes!

- Given graph G consisting of n nodes, let $\mathbf{A} \in \mathbb{R}^{n \times n}$, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote the adjacency matrix and matrix of node features of G, respectively
- Then, a representation for the entire graph is produced as follows:

$$\mathbf{h}_G = \frac{1}{n!} \sum_{\mathbf{P} \in \Pi} f(\mathbf{P}\,\mathbf{A}\,\mathbf{P}^\top, \mathbf{P}\,\mathbf{X})$$

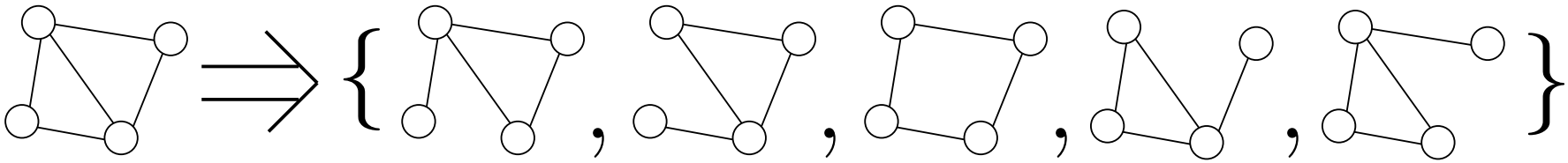
  where Π is the set of $n \times n$ permutation matrices
- Example of an RP model **[Murphy et al., ICML'19]**:
  (1) add unique IDs as node features
  (2) use any message passing GNN model
  (3) sum over all permutations of IDs



Subgraph GNNs

Idea: We can decompose a graph into a set of subgraphs and process those subgraphs

- Step 1: Extract subgraphs from a graph and represent the graph as a set of its subgraphs

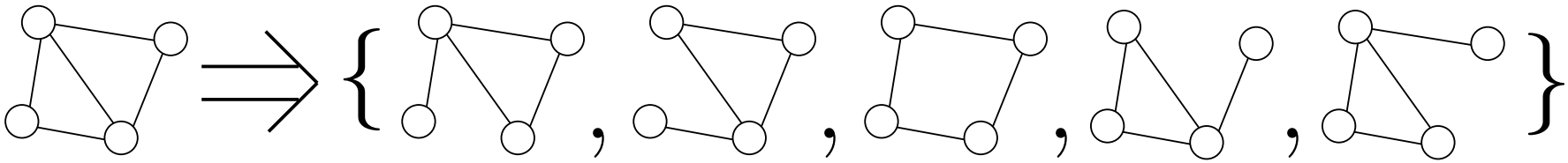


- Step 2: We can generate a representation for the graph by mapping the set of subgraphs into a vector

Subgraph GNNs

Idea: We can decompose a graph into a set of subgraphs and process those subgraphs

- Step 1: Extract subgraphs from a graph and represent the graph as a set of its subgraphs

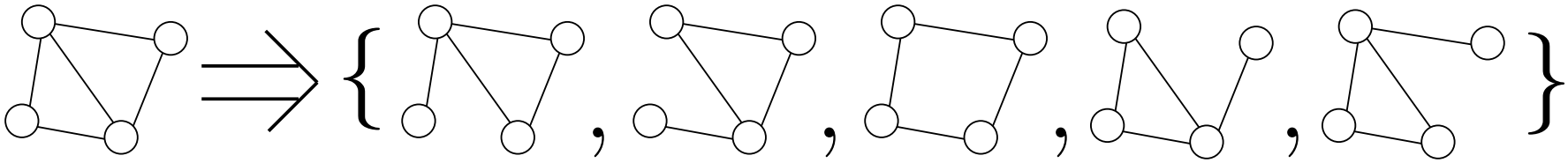


- Step 2: We can generate a representation for the graph by mapping the set of subgraphs into a vector
- However, two main challenges arise:
  1. How to extract subgraphs from a given graph?
     $\hookrightarrow$ different models propose different policies
  2. How to process sets of subgraphs?
     $\hookrightarrow$ each subgraph can be mapped into a vector and then, set of vectors mapped into a single representation

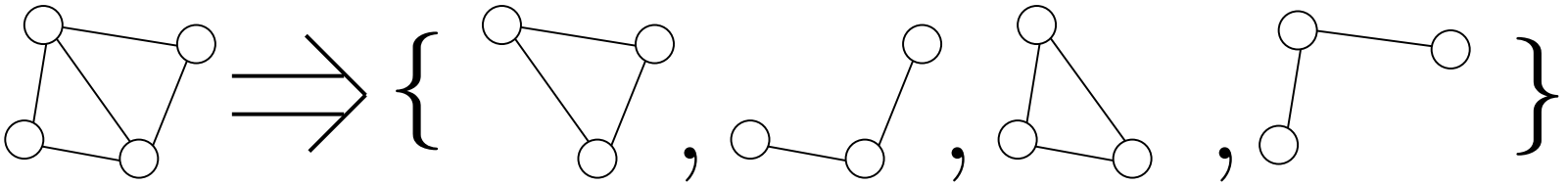
How to Extract Subgraphs from a Given Graph?

Different policies can be employed **[Bevilacqua et al., ICLR'22]**

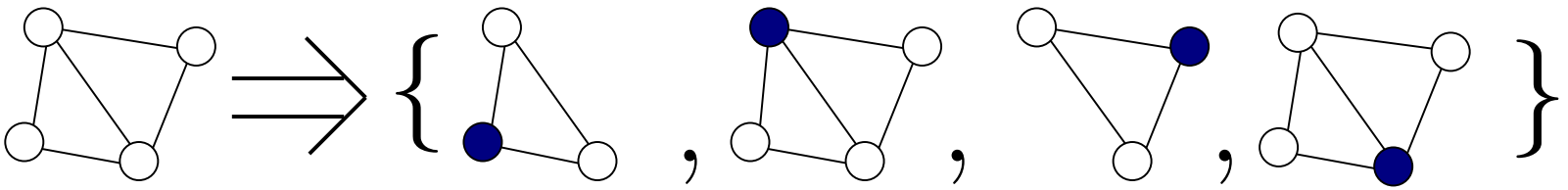
- Edge-deleted subgraphs



- Node-deleted subgraphs

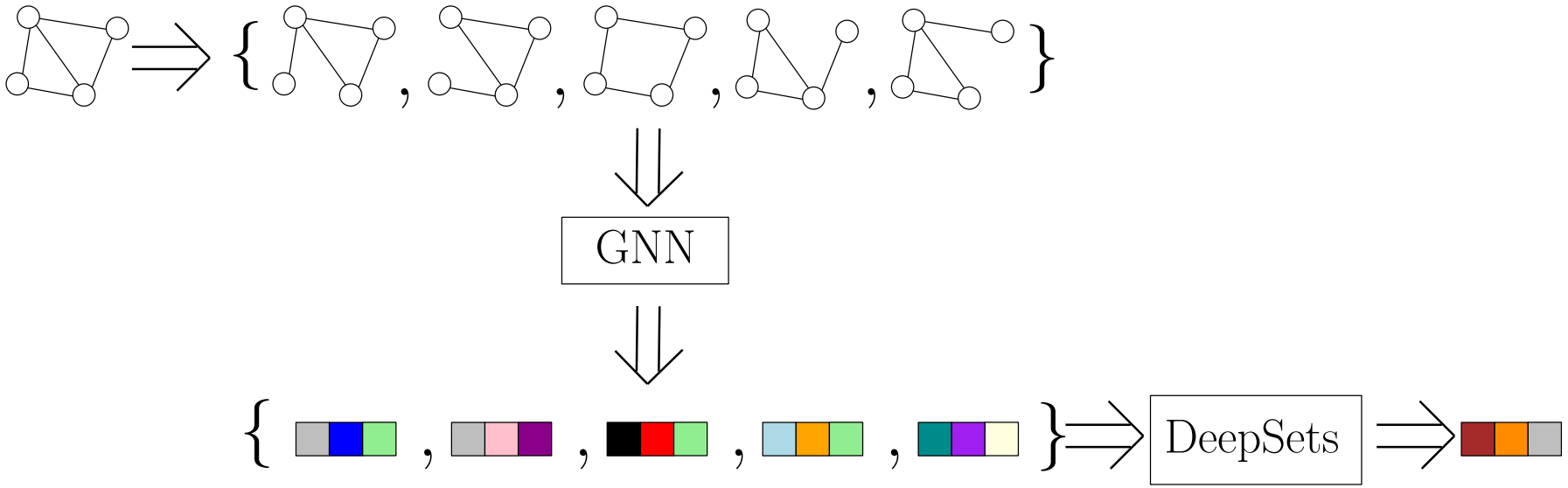


- Ego-networks (rooted)



How to Process Sets of Subgraphs?

- Use some message passing GNN model (e.g., GIN) to obtain a vector representation for each subgraph
- Then, use DeepSets to obtain a final representation for entire graph



Transformer

Transformer has become a dominant architecture in many domains (e.g., natural language processing, computer vision)

The Transformer architecture consists of a composition of Transformer layers

- Each Transformer layer has two parts:
  (i) a self-attention module
  (ii) a position-wise feed-forward network (FFN)
- Let $\mathbf{H} = [\mathbf{h}_1^\top, \ldots, \mathbf{h}_n^\top]^\top \in \mathbb{R}^{n \times d}$ denote the input of self-attention module where d is the hidden dimension and $\mathbf{h}_i \in \mathbb{R}^{1 \times d}$ is the hidden representation at position i
- The input $\mathbf{H}$ is projected by three matrices $\mathbf{W}_Q \in \mathbb{R}^{d \times d_K}$, $\mathbf{W}_K \in \mathbb{R}^{d \times d_K}$ and $\mathbf{W}_V \in \mathbb{R}^{d \times d_V}$ to the corresponding representations $\mathbf{Q}, \mathbf{K}, \mathbf{V}$:

$$\mathbf{Q} = \mathbf{H}\,\mathbf{W}_Q, \qquad \mathbf{K} = \mathbf{H}\,\mathbf{W}_K, \qquad \mathbf{V} = \mathbf{H}\,\mathbf{W}_V$$

- The self-attention is then calculated as

$$\mathbf{A} = \frac{\mathbf{Q}\,\mathbf{K}^\top}{\sqrt{d_K}}$$

$$\text{Attn}(\mathbf{H}) = \text{softmax}(\mathbf{A})\,\mathbf{V}$$

Graphormer

- Suppose we trivially apply a Transformer to graph data
- ☹ For each node v_i, self-attention only calculates the semantic similarity between v_i and other nodes, without considering the structural information of the graph
- **Idea:** incorporate structural information of graphs into the model
  But what type of structural information?

Graphormer

- Suppose we trivially apply a Transformer to graph data
- ☹ For each node v_i, self-attention only calculates the semantic similarity between v_i and other nodes, without considering the structural information of the graph
- **Idea:** incorporate structural information of graphs into the model
  But what type of structural information?
- In a graph, different nodes may have **different importance**, e.g., celebrities are considered to be more influential than the rest of the users in a social network
- Graphormer uses Centrality Encoding to capture the node importance in the graph **[Ying et al., NeurIPS'21]**:

$$\mathbf{h}_v^{(0)} = \mathbf{x}_v + \mathbf{z}_{\deg(v)}$$

where $\mathbf{x}_v$ is the vector of initial node features of v and $\mathbf{z}_{\deg(v)}$ is a learnable embedding vector specified by the degree of v

Graphormer

- Suppose we trivially apply a Transformer to graph data
- ☹ For each node v_i, self-attention only calculates the semantic similarity between v_i and other nodes, without considering the structural information of the graph
- **Idea:** incorporate structural information of graphs into the model
  But what type of structural information?
- In a graph, some nodes are closer to a given node than other nodes
- Graphormer models such structural information as follows **[Ying et al., NeurIPS'21]**:

$$\mathbf{A}_{ij} = \frac{(\mathbf{W}_Q \, \mathbf{h}_{v_i})^\top (\mathbf{W}_K \, \mathbf{h}_{v_j})}{\sqrt{d}} + \mathbf{b}_{\phi(v_i, v_j)}$$

  where ϕ is a function that measures the shortest path distance between two nodes

Experimental Evaluation

Experiments on two molecular property prediction datasets:

(i) ZINC-12K dataset

- a graph regression dataset
- consists of $12,000$ molecules
- the task is to predict the constrained solubility of molecules, an important chemical property for designing generative GNNs for molecules

(ii) ogbg-molhiv

- a binary graph classification dataset from the Open Graph Benchmark (OGB)
- consists of $41,127$ molecules
- the task is to predict whether a molecule inhibits HIV virus replication or not

All experiments are conducted using available train/val/test splits

Graph Regression Results

Table: Mean absolute error ($\pm$ standard deviation) of the different methods on the ZINC12K dataset. K denotes the number of employed layers.

	K	**ZINC-12K** $\downarrow$
GCN **[Kipf and Welling, ICLR'17]**	16	0.278 $\pm$ 0.003
GraphSAGE **[Hamilton et al., NeurIPS'17]**	16	0.398 $\pm$ 0.002
MoNet **[Monti et al., CVPR'17]**	16	0.292 $\pm$ 0.006
GAT **[Velickovic et al., ICLR'18]**	16	0.384 $\pm$ 0.007
GIN **[Xu et al., ICLR'19]**	5	0.387 $\pm$ 0.015
RingGNN **[Chen et al., NeurIPS'19]**	2	0.353 $\pm$ 0.019
PPGN **[Maron et al., NeurIPS'19]**	3	0.256 $\pm$ 0.054
GNNML3 **[Balcilar et al., ICML'21]**	NA	0.161 $\pm$ 0.006
Graphormer **[Ying et al., NeurIPS'21]**	NA	0.122 $\pm$ 0.006
CIN **[Bodnar et al., NeurIPS'21]**	NA	**0.079** $\pm$ 0.006
ESAN **[Bevilacqua et al., ICLR'22]**	NA	0.102 $\pm$ 0.003
KP-GIN **[Feng et al., NeurIPS'22]**	NA	0.093 $\pm$ 0.007
AgentNet **[Martinkus et al., ICLR'23]**	NA	0.258 $\pm$ 0.033
PathNN **[Gaspard et al., ICML'23]**	4	0.090 $\pm$ 0.004

Graph Classification Results

Table: ROC-AUC score (± standard deviation) of the different methods on the ogbg-molhiv dataset.

	ogbg-molhiv ↑
GCN **[Kipf and Welling, ICLR'17]**	76.06 ± 0.97
GIN **[Xu et al., ICLR'19]**	75.58 ± 1.40
GSN **[Bouritsas et al., TPAMI 45(1)]**	77.99 ± 1.00
HIMP **[Fey et al., arXiv:2006.12179]**	78.80 ± 0.82
PNA **[Corso et al., NeurIPS'20]**	79.05 ± 1.32
DGN **[Beaini et al., ICML'21]**	79.70 ± 0.97
Graphormer **[Ying et al., NeurIPS'21]**	80.51 ± 0.53
CIN **[Bodnar et al., NeurIPS'21]**	**80.94** ± 0.57
ESAN **[Bevilacqua et al., ICLR'22]**	78.00 ± 1.42
E-SPN **[Abboud et al., LOG'23]**	77.10 ± 1.20
GRWNN **[Nikolentzos and Vazirgiannis, AISTATS'23]**	78.38 ± 0.99
WLHN **[Nikolentzos et al., AISTATS'23]**	78.41 ± 0.31
AgentNet **[Martinkus et al., ICLR'23]**	78.33 ± 0.69
PathNN **[Gaspard et al., ICML'23]**	79.17 ± 1.09

Thank you!

$\mathbb{X}$:	@giannis_nikole
Slides:	`https://users.uop.gr/~nikolentzos/files/slides_hias.pdf`
Contact:	lastname@uop.gr