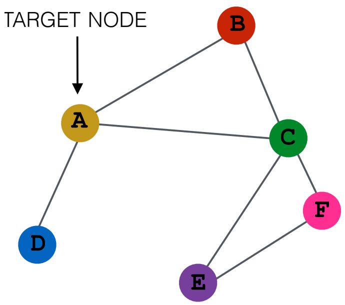
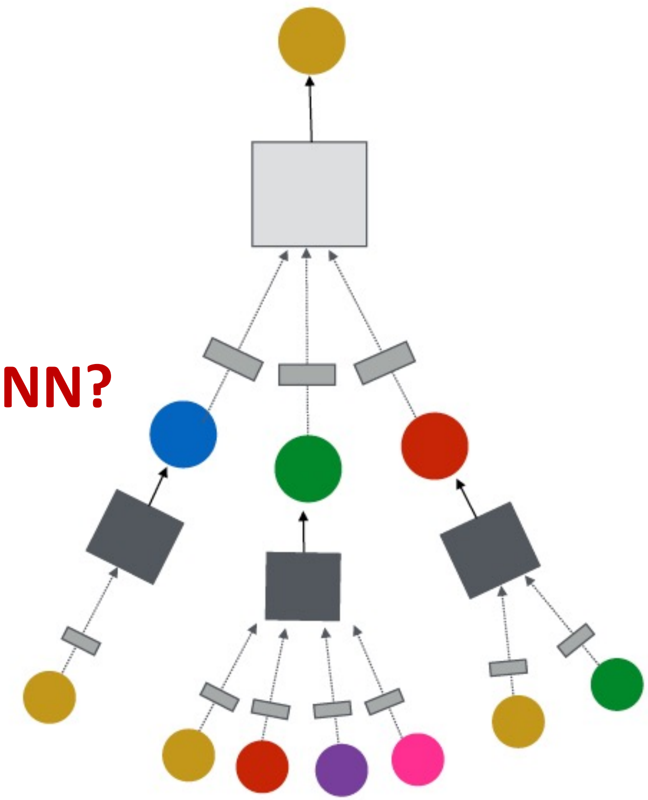
Prediction with GNNs

A General GNN Framework (4)



INPUT GRAPH

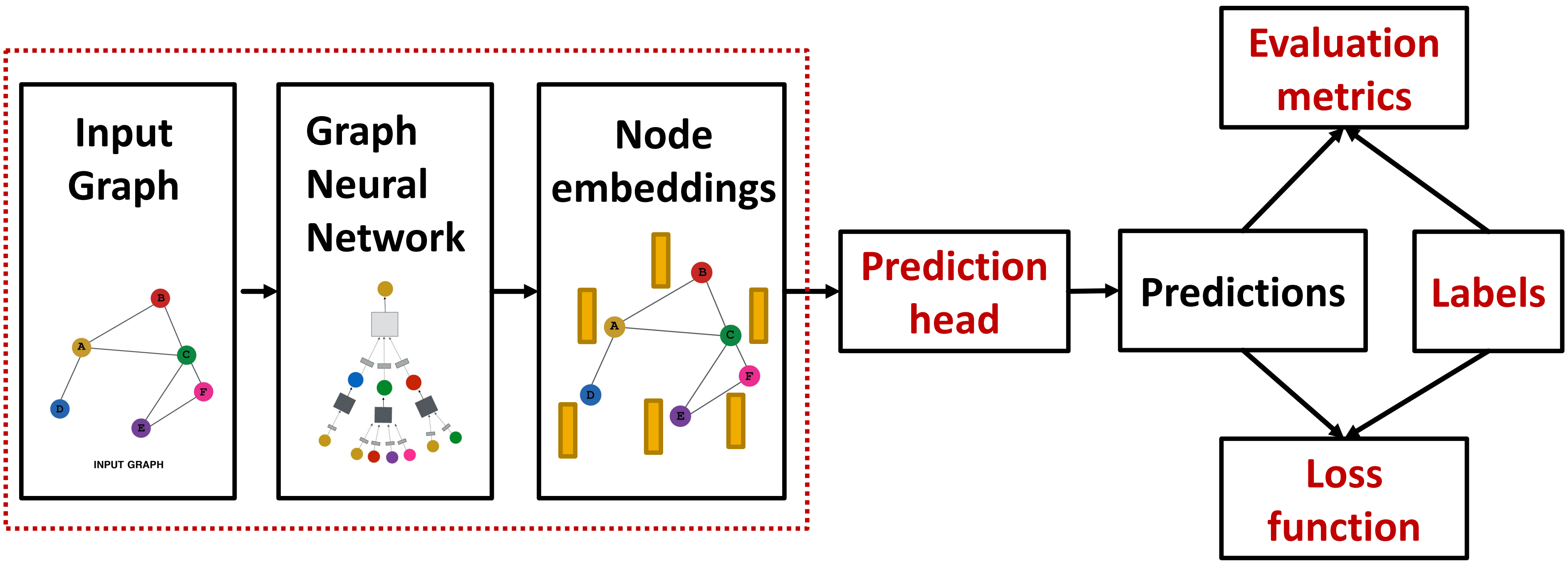
(5) Learning objective



Next: How do we train a GNN?

GNN Training Pipeline

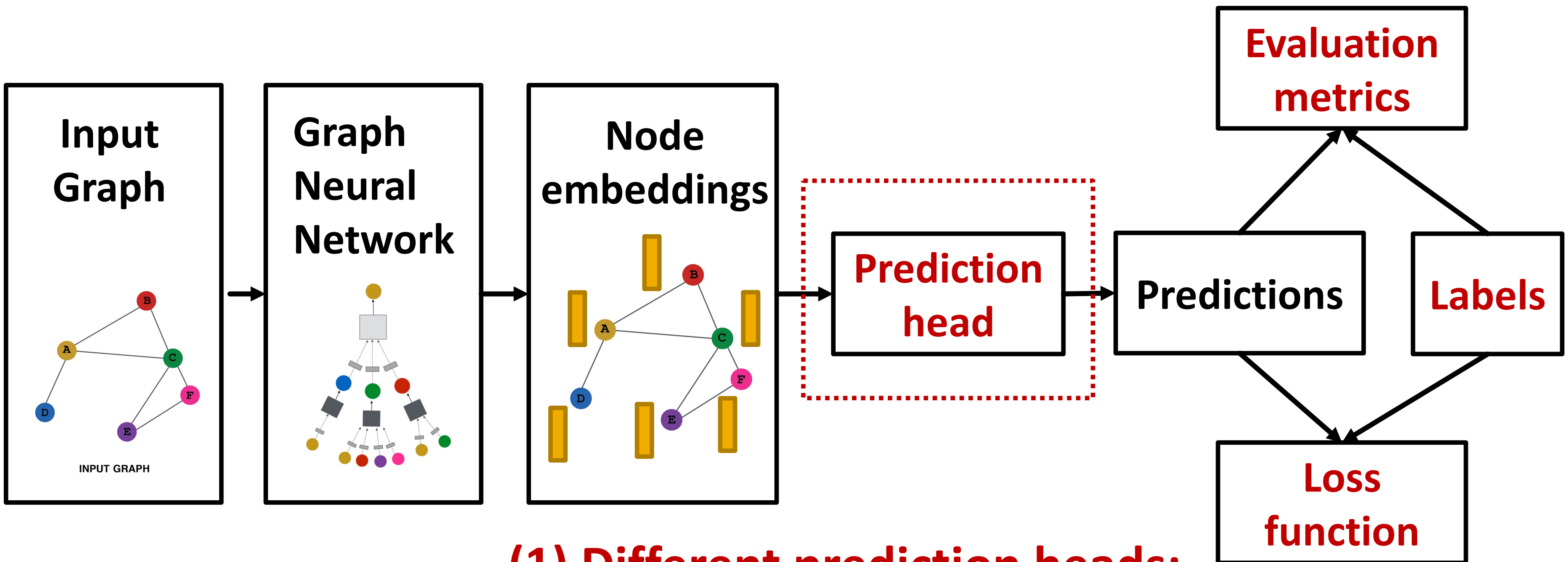
So far what we have covered



Output of a GNN: set of node embeddings

$$\{\mathbf{h}_v^{(L)}, \forall v \in G\}$$

GNN Training Pipeline (1)

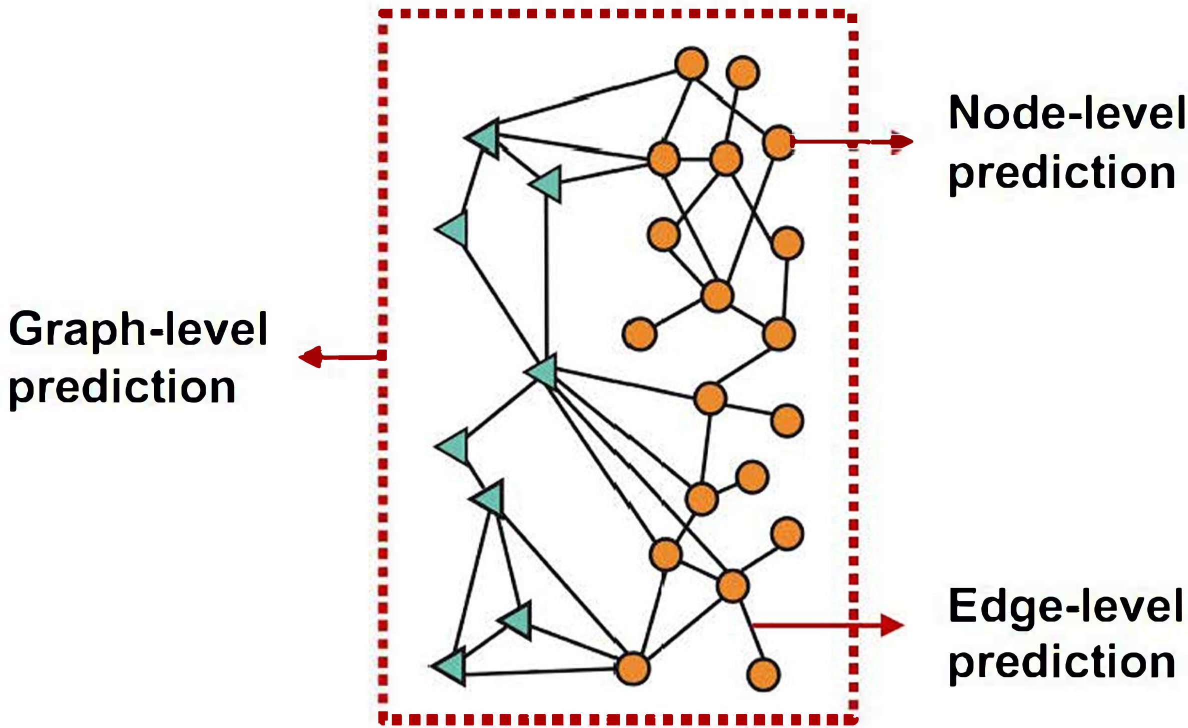


(1) Different prediction heads:

- Node-level tasks
- Edge-level tasks
- Graph-level tasks

GNN Prediction Heads

- **Idea:** Different task levels require different prediction heads

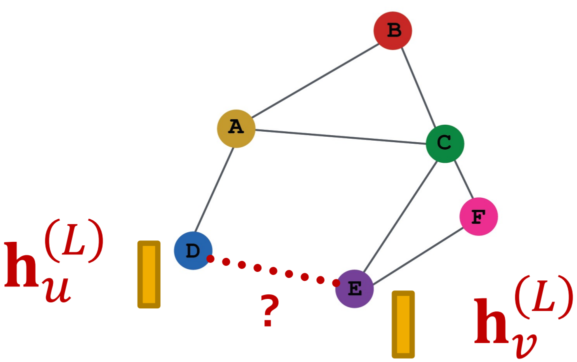


Prediction Heads: Node-level

- **Node-level prediction**: We can directly make prediction using node embeddings!
- After GNN computation, we have d-dim node embeddings: $\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\}$
- Suppose we want to make k-way prediction
  - Classification: classify among k categories
  - Regression: regress on k targets
- $\widehat{\boldsymbol{y}}_{\boldsymbol{v}} = \mathrm{Head}_{\mathrm{node}}(\mathbf{h}_v^{(L)}) = \mathbf{W}^{(H)}\mathbf{h}_v^{(L)}$
  - $\mathbf{W}^{(H)} \in \mathbb{R}^{k*d}$: We map node embeddings from $\mathbf{h}_v^{(L)} \in \mathbb{R}^d$ to $\widehat{\boldsymbol{y}}_v \in \mathbb{R}^k$ so that we can compute the loss

Prediction Heads: Edge-level

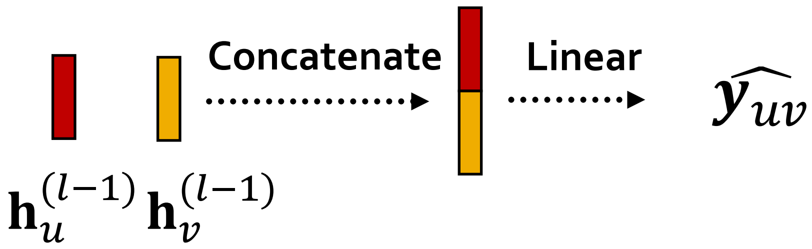
- **Edge-level prediction**: Make prediction using pairs of node embeddings
- Suppose we want to make k-way prediction
- $\widehat{\boldsymbol{y}}_{\boldsymbol{uv}} = \mathrm{Head}_{\mathrm{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$



- What are the options for $\mathrm{Head}_{\mathrm{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$?

Prediction Heads: Edge-level

- **Options for** $\mathrm{Head}_{\mathrm{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$:
- **(1) Concatenation + Linear**
  - We have seen this in graph attention



  - $\widehat{\boldsymbol{y}}_{\boldsymbol{uv}} = \mathrm{Linear}(\mathrm{Concat}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}))$
  - Here $\mathrm{Linear}(\cdot)$ will map $2d$-dimensional embeddings (since we concatenated embeddings) to k-dim embeddings (k-way prediction)

Prediction Heads: Edge-level

- **Options for** $\mathrm{Head}_{\mathrm{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)})$:
- **(2) Dot product**
  - $\widehat{\boldsymbol{y}}_{\boldsymbol{uv}} = (\mathbf{h}_u^{(L)})^T \mathbf{h}_v^{(L)}$
  - **This approach only applies to 1-way prediction** (e.g., link prediction: predict the existence of an edge)
  - **Applying to $\boldsymbol{k}$-way prediction:**
    - Similar to **multi-head attention**: $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(k)}$ trainable

$$\widehat{\boldsymbol{y}}_{\boldsymbol{uv}}^{(\mathbf{1})} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(1)} \mathbf{h}_v^{(L)}$$

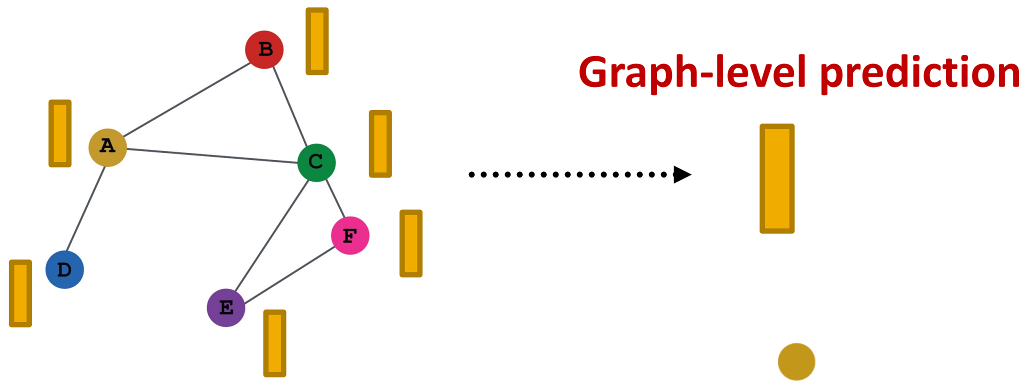
$$\ldots$$

$$\widehat{\boldsymbol{y}}_{\boldsymbol{uv}}^{(\boldsymbol{k})} = (\mathbf{h}_u^{(L)})^T \mathbf{W}^{(k)} \mathbf{h}_v^{(L)}$$

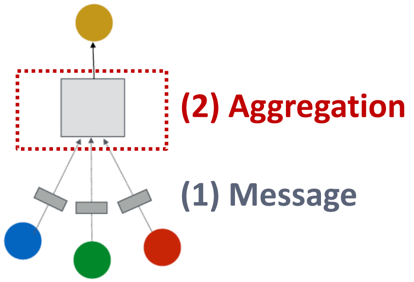
$$\widehat{\boldsymbol{y}}_{uv} = \mathrm{Concat}(\widehat{\boldsymbol{y}}_{\boldsymbol{uv}}^{(\mathbf{1})}, \ldots, \widehat{\boldsymbol{y}}_{\boldsymbol{uv}}^{(\boldsymbol{k})}) \in \mathbb{R}^k$$

Prediction Heads: Graph-level

- **Graph-level prediction**: Make prediction using all the node embeddings in our graph
- Suppose we want to make k-way prediction
- $\widehat{\boldsymbol{y}}_G = \mathrm{Head}_{\mathrm{graph}}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$



- $\mathrm{Head}_{\mathrm{graph}}(\cdot)$ is similar to $\mathrm{AGG}(\cdot)$ in a GNN layer!



Prediction Heads: Graph-level

- Options for $\text{Head}_{\text{graph}}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$
- **(1) Global mean pooling**

$$\widehat{\boldsymbol{y}}_G = \text{Mean}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- **(2) Global max pooling**

$$\widehat{\boldsymbol{y}}_G = \text{Max}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- **(3) Global sum pooling**

$$\widehat{\boldsymbol{y}}_G = \text{Sum}(\{\mathbf{h}_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- These options work great for small graphs
- **Can we do better for large graphs?**

Issue of Global Pooling

- **Issue:** Global pooling over a (large) graph will lose information
- **Toy example:** we use 1-dim node embeddings
  - Node embeddings for G_1: $\{-1, -2, 0, 1, 2\}$
  - Node embeddings for G_2: $\{-10, -20, 0, 10, 20\}$
  - Clearly G_1 and G_2 have very different node embeddings → Their structures should be different
- **If we do global sum pooling:**
  - **Prediction for** G_1: $\hat{y}_G = \mathrm{Sum}(\{-1, -2, 0, 1, 2\}) = 0$
  - **Prediction for** G_2: $\hat{y}_G = \mathrm{Sum}(\{-10, -20, 0, 10, 20\}) = 0$
  - We cannot differentiate G_1 and G_2!

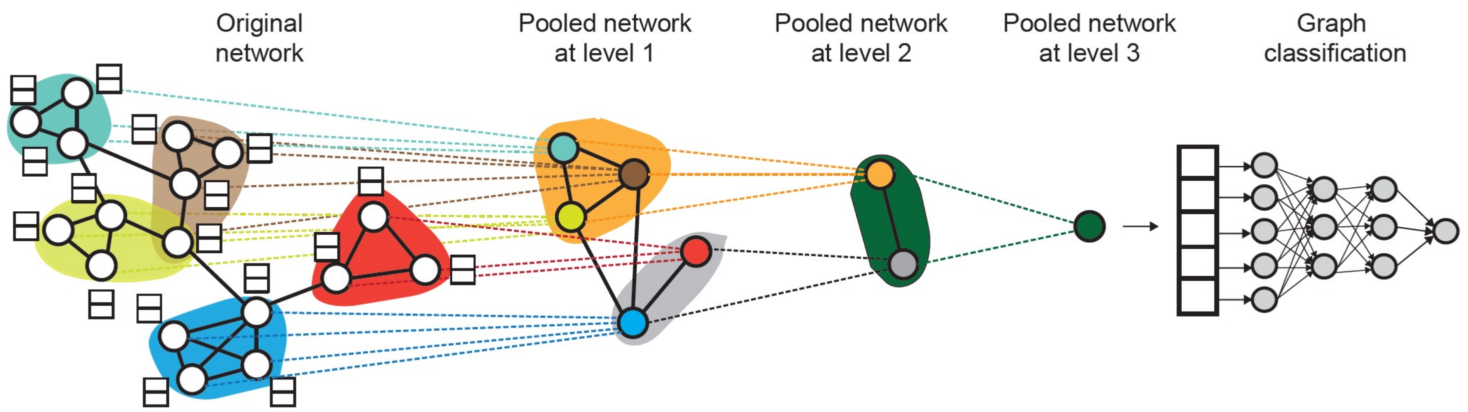
Hierarchical Global Pooling

- **A solution:** Let's aggregate all the node embeddings **hierarchically**
  - **Toy example:** We will aggregate via $\text{ReLU}\big(\text{Sum}(\cdot)\big)$
    - We first separately aggregate the first 2 nodes and last 3 nodes
    - Then we aggregate again to make the final prediction
  - G_1 node embeddings: $\{-1, -2, 0, 1, 2\}$
    - **Round 1**: $\hat{y}_a = \text{ReLU}\big(\text{Sum}(\{-1, -2\})\big) = 0$, $\hat{y}_b = \text{ReLU}\big(\text{Sum}(\{0,1, 2\})\big) = 3$
    - **Round 2:** $\hat{y}_G = \text{ReLU}\big(\text{Sum}(\{y_a, y_b\})\big) = \mathbf{3}$
  - G_2 node embeddings: $\{-10, -20, 0, 10, 20\}$
    - **Round 1**: $\hat{y}_a = \text{ReLU}\big(\text{Sum}(\{-10, -20\})\big) = 0$, $\hat{y}_b = \text{ReLU}\big(\text{Sum}(\{0,10, 20\})\big) = 30$
    - **Round 2:** $\hat{y}_G = \text{ReLU}\big(\text{Sum}(\{y_a, y_b\})\big) = \mathbf{30}$

Now we can differentiate G_1 and G_2 !

Hierarchical Pooling In Practice

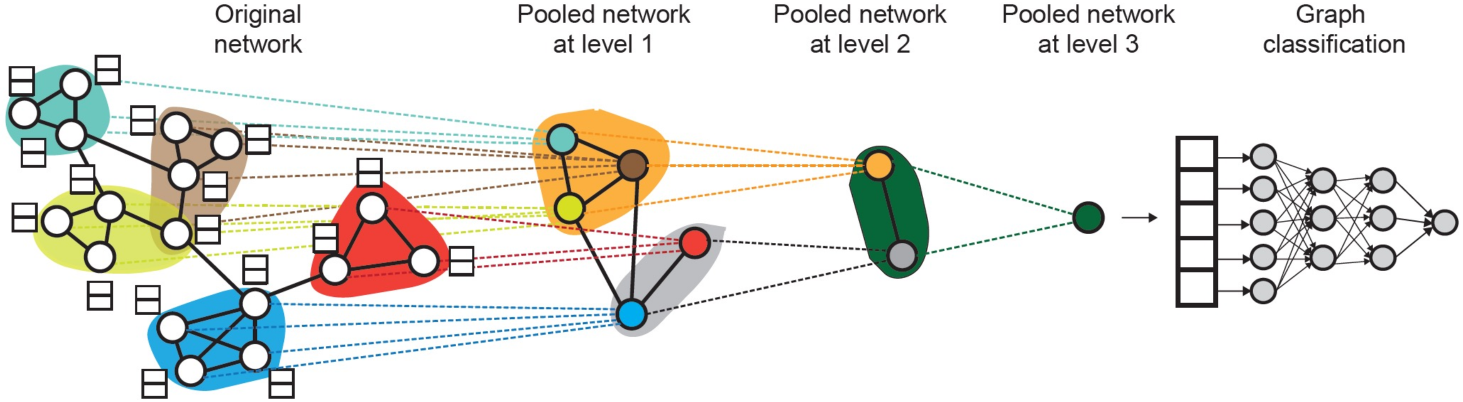
- **DiffPool idea:**
  - **Hierarchically pool node embeddings**



  - **Leverage 2 independent GNNs at each level**
    - **GNN A:** Compute node embeddings
    - **GNN B:** Compute the cluster that a node belongs to
  - **GNNs A and B at each level can be executed in parallel**

Hierarchical Pooling In Practice

- **DiffPool idea:**



  - **For each Pooling layer**
    - Use clustering assignments from **GNN B** to aggregate node embeddings generated by **GNN A**
    - Create a **single new node** for each cluster, maintaining edges between clusters to generated a new **pooled** network
  - **Jointly train GNN A and GNN B**