Scaling Up the Graph Neural Network (2)

ACMS 80770: Deep Learning with Graphs

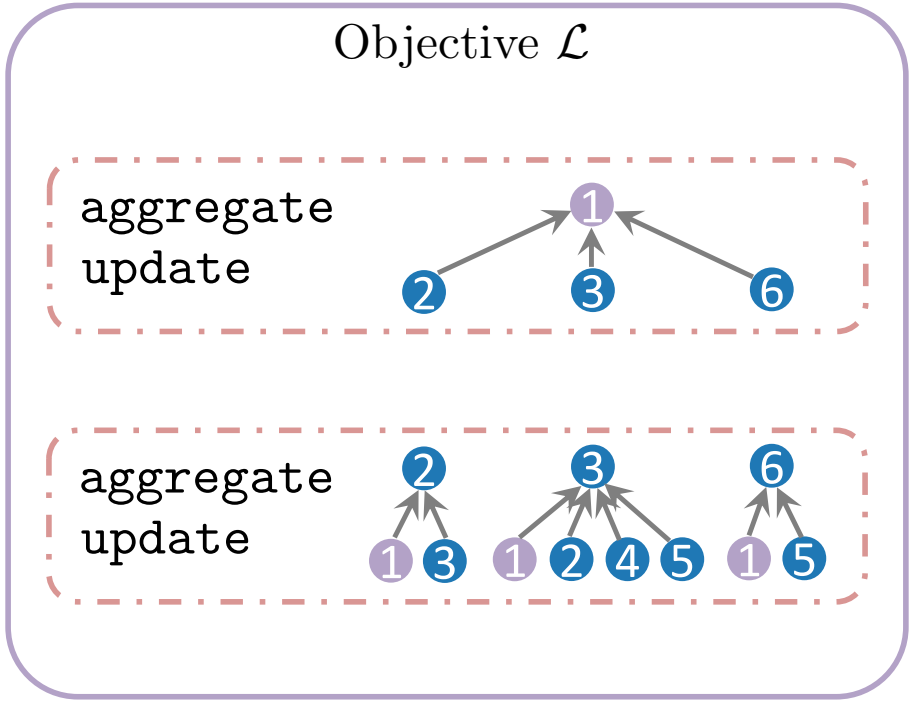
Instructor: Navid Shervani-Tabar

Department of Applied and Comp Math and Stats

UNIVERSITY OF NOTRE DAME

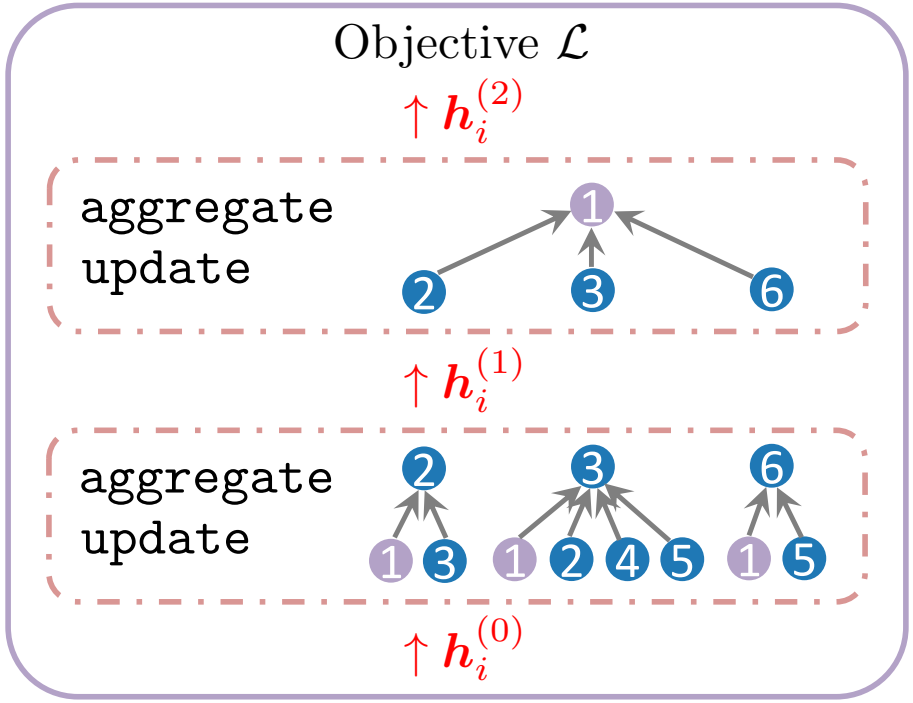
Jumping Knowledge Connection

- In the examples so far, we have directly used the embedding in the last layer $\boldsymbol{h}_i^{(K)}$ as node-level feature representation $\boldsymbol{z}_i$.



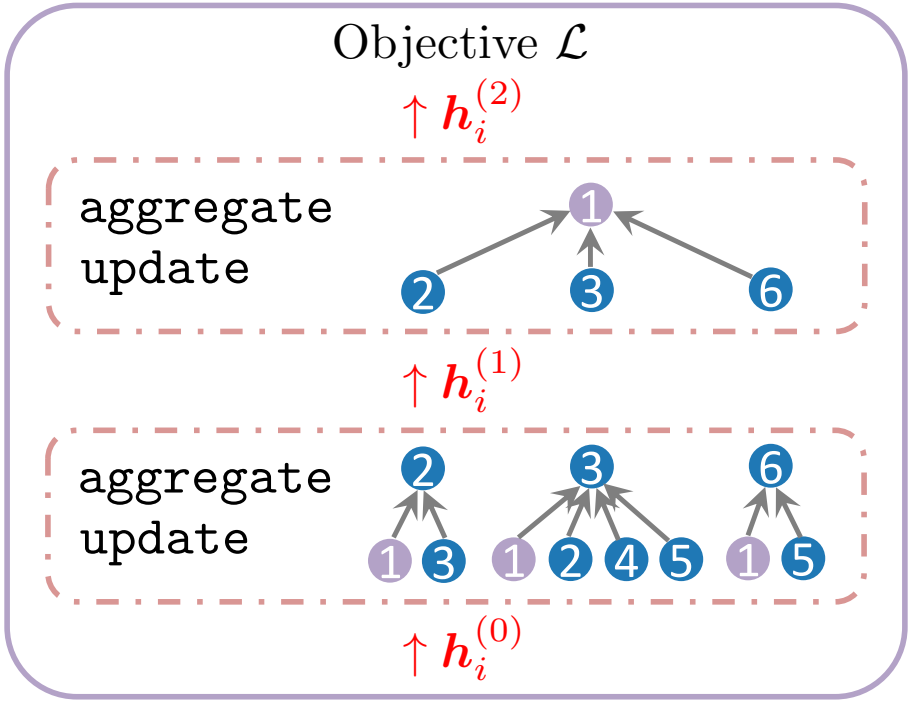
Jumping Knowledge Connection

- In the examples so far, we have directly used the embedding in the last layer $\boldsymbol{h}_i^{(K)}$ as node-level feature representation $\boldsymbol{z}_i$.



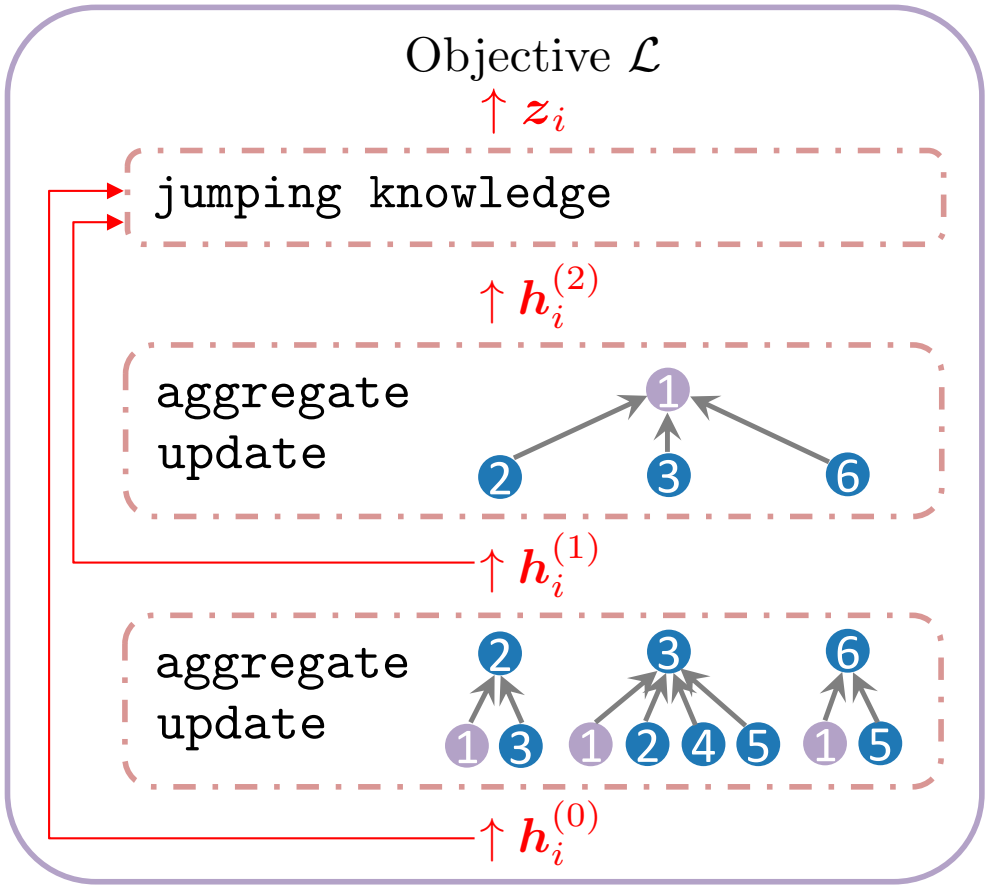
Jumping Knowledge Connection

- In the examples so far, we have directly used the embedding in the last layer $\boldsymbol{h}_i^{(K)}$ as node-level feature representation $\boldsymbol{z}_i$.
- However, over-smoothing causes the local information of the node contained in node embedding to be washed out after a few iterations of GNN.



Jumping Knowledge Connection

- One way to avoid this is by explicitly combining the node embeddings $\boldsymbol{h}_i^{(k)}$ at different layers k to construct final feature representation $\boldsymbol{z}_i$ for different nodes $v_i \in V$.



Jumping Knowledge Connection

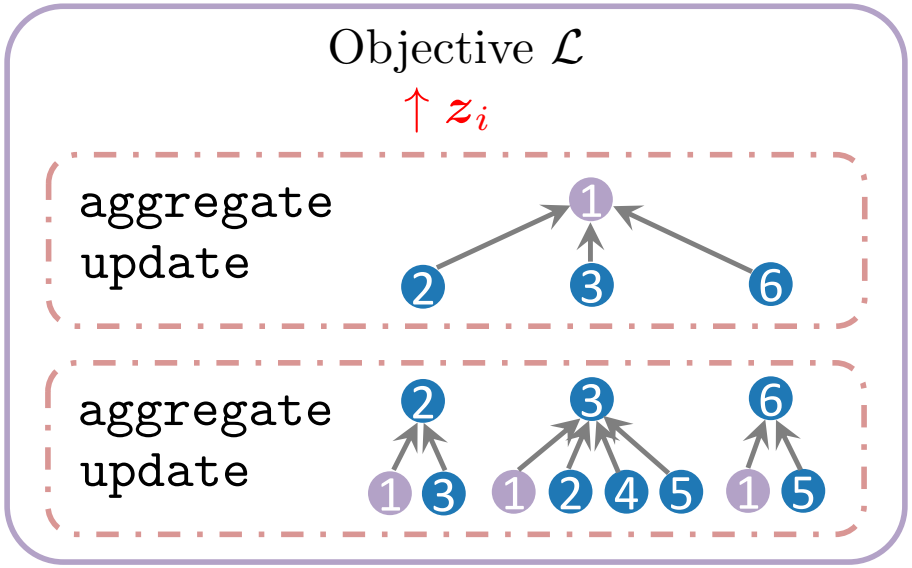
- One way to avoid this is by explicitly combining the node embeddings $\boldsymbol{h}_i^{(k)}$ at different layers k to construct final feature representation $\boldsymbol{z}_i$ for different nodes $v_i \in V$.
- This approach is referred to as Jumping knowledge
- Mathematically put

$$\boldsymbol{z}_i = f_{\text{JK}}\left(\boldsymbol{h}_i^{(0)} \oplus \cdots \oplus \boldsymbol{h}_i^{(K)}\right)$$

where f_{JK} is a differentiable function, $\oplus$ denotes concatenation, and K is the number of GNN layers.

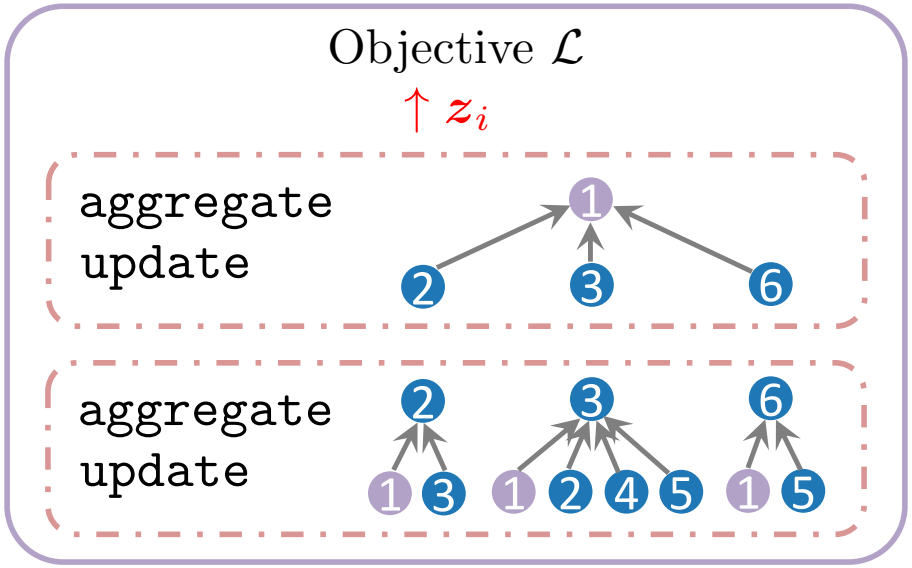
Graph Pooling

- Our discussion on GNNs has, so far, been focused on extracting **node-level features z_i** for nodes $v_i \in V$.



Graph Pooling

- Our discussion on GNNs has, so far, been focused on extracting **node-level features** $\boldsymbol{z}_i$ for nodes $v_i \in V$.
- However, many practical scenarios involve **graph-level** tasks.
- To that end, we need to construct **graph-level embeddings** $\boldsymbol{z}_G$ that represent the whole graph G.

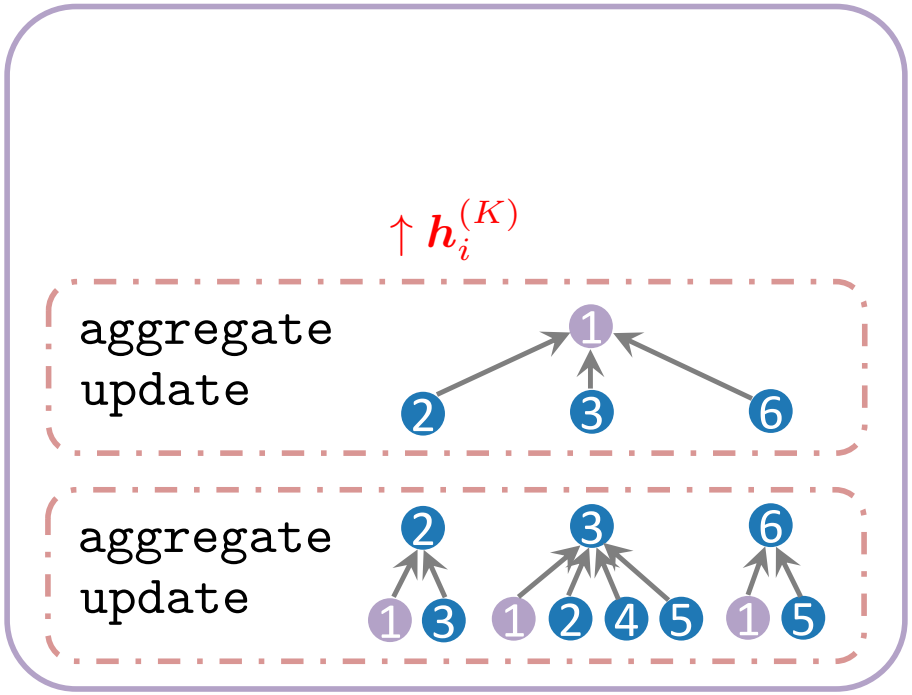


Graph Pooling

- Our discussion on GNNs has, so far, been focused on extracting **node-level features** $\boldsymbol{z}_i$ for nodes $v_i \in V$.
- However, many practical scenarios involve **graph-level** tasks.
- To that end, we need to construct **graph-level embeddings** $\boldsymbol{z}_G$ that represent the whole graph G.
- The set of approaches to construct graph-level embeddings from node-level embeddings are referred to as **graph pooling**.
- There are **two** general methods for graph pooling
  - **Multi-Set pooling** approaches
  - **Graph coarsening** approaches.

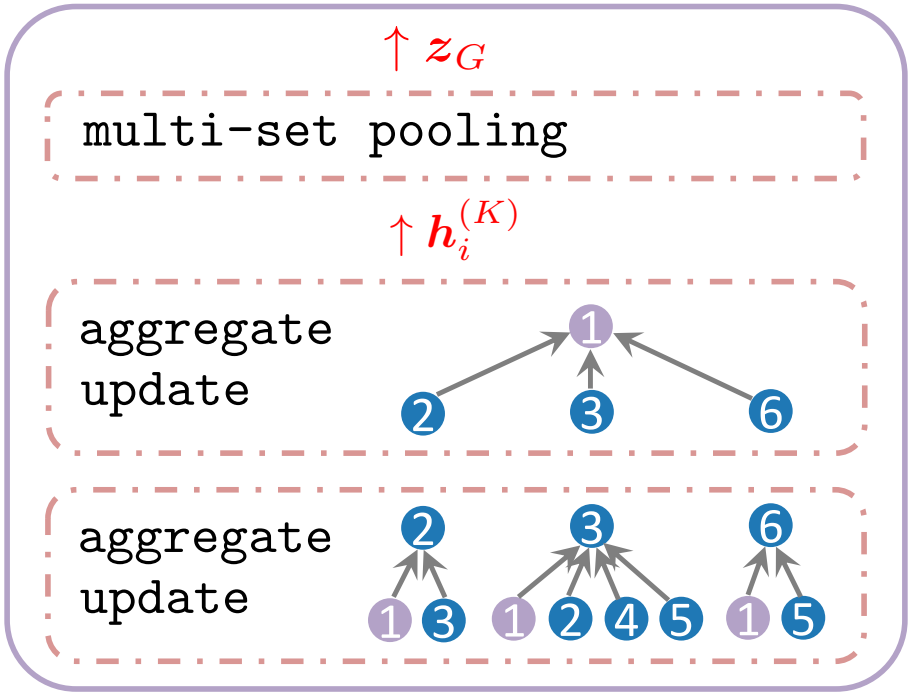
Multi-set Pooling

- In the multi-set pooling approach, a **pooling function** f_{p} maps node embeddings $\boldsymbol{z}_i$ to a graph-level embedding $\boldsymbol{z}_G$.



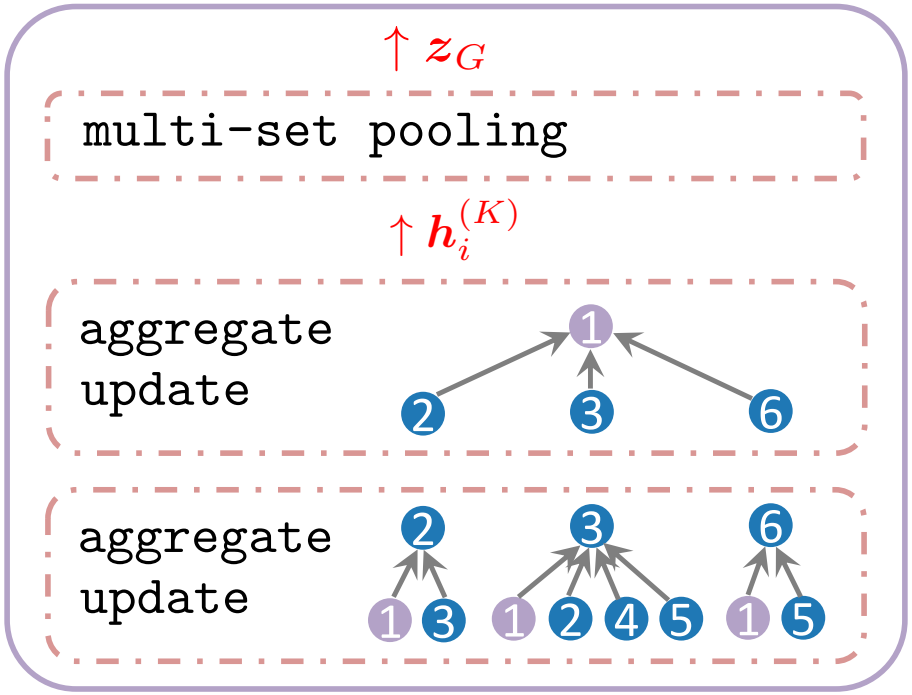
Multi-set Pooling

- In the multi-set pooling approach, a **pooling function** f_{p} maps node embeddings $\boldsymbol{z}_i$ to a graph-level embedding $\boldsymbol{z}_G$.



Multi-set Pooling

- In the multi-set pooling approach, a **pooling function** f_{p} maps node embeddings $\boldsymbol{z}_i$ to a graph-level embedding $\boldsymbol{z}_G$.
- This approach is **similar** to **constructing messages** from the multi-set of node embeddings in the neighborhood during message-passing.



Multi-set Pooling

- In the multi-set pooling approach, a **pooling function** f_{p} maps node embeddings $\boldsymbol{z}_i$ to a graph-level embedding $\boldsymbol{z}_G$.
- This approach is **similar** to **constructing messages** from the multi-set of node embeddings in the neighborhood during message-passing.
- In practice, any function used in the aggregation step can be used for multi-set pooling.
  - An example of such function is sum or average function

$$\boldsymbol{z}_G = \frac{\sum_{v_i \in V} \boldsymbol{z}_i}{f_n(|V|)}$$

where f_n is a differentiable normalization function.

Multi-set Pooling

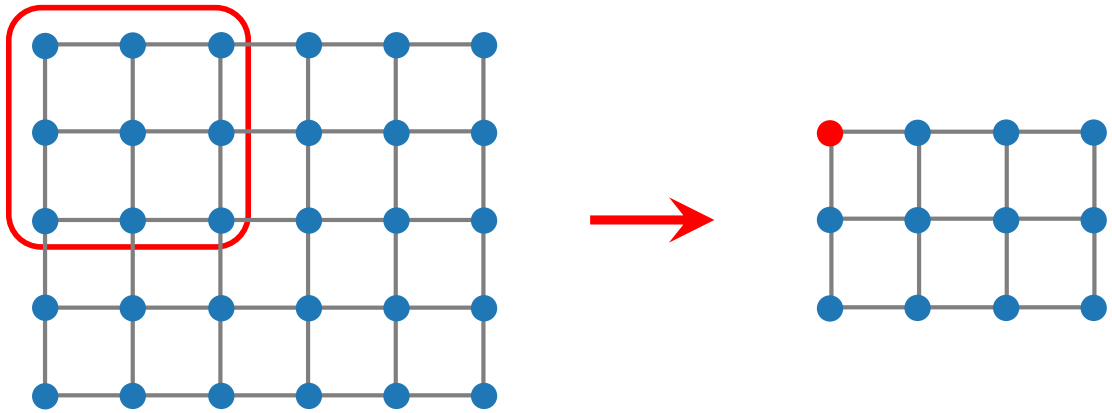
- One issue with the multi-set pooling approaches is that they construct graph-level representations from a multi-set of **individual node** embedding vectors and **disregard the graph** connectivity.

Graph Coarsening

- One issue with the multi-set pooling approaches is that they construct graph-level representations from a multi-set of **individual node** embedding vectors and **disregard the graph** connectivity.
- To address this issue, one can perform **successive graph coarsening** operations to construct graph-level features.
- This method is inspired by **pooling** in the **CNN** models.

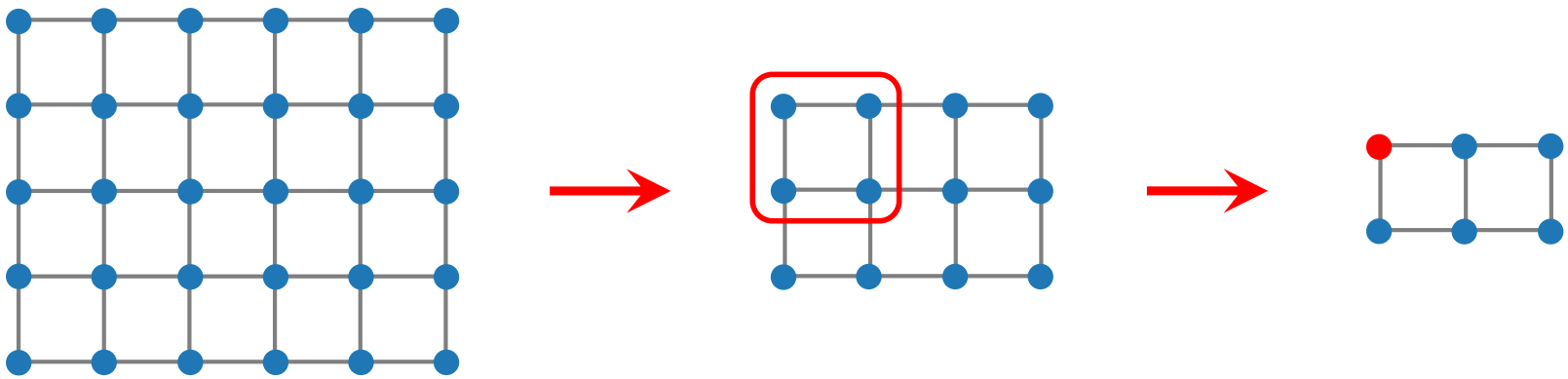
Graph Coarsening

- One issue with the multi-set pooling approaches is that they construct graph-level representations from a multi-set of **individual node** embedding vectors and **disregard the graph** connectivity.
- To address this issue, one can perform **successive graph coarsening** operations to construct graph-level features.
- This method is inspired by **pooling** in the **CNN** models.



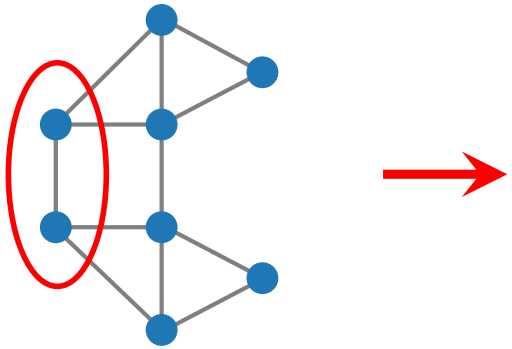
Graph Coarsening

- One issue with the multi-set pooling approaches is that they construct graph-level representations from a multi-set of **individual node** embedding vectors and **disregard the graph** connectivity.
- To address this issue, one can perform **successive graph coarsening** operations to construct graph-level features.
- This method is inspired by **pooling** in the **CNN** models.



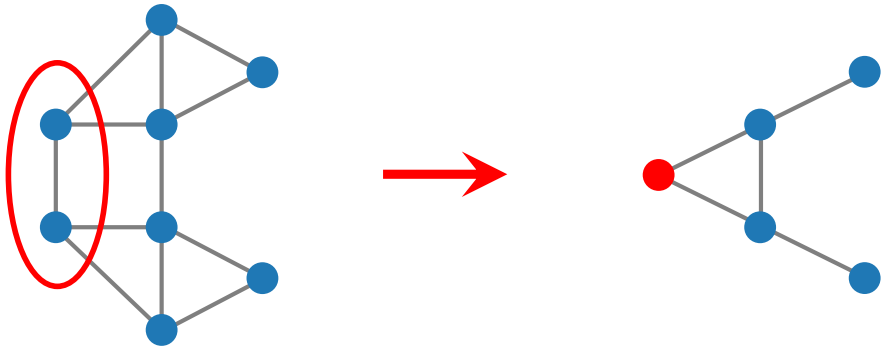
Graph Coarsening

- In each iteration, coarsening operator **clusters** sets of connected nodes to construct **super-nodes** for a coarsened graph.



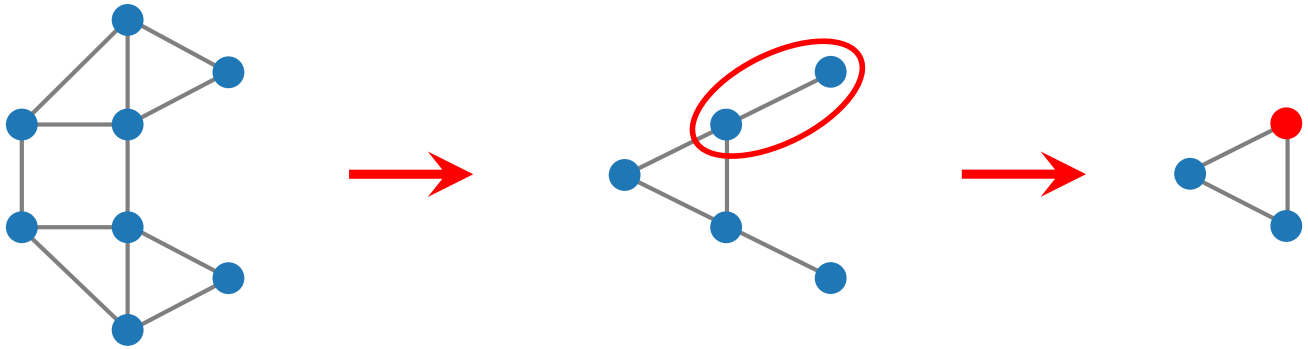
Graph Coarsening

- In each iteration, coarsening operator **clusters** sets of connected nodes to construct **super-nodes** for a coarsened graph.



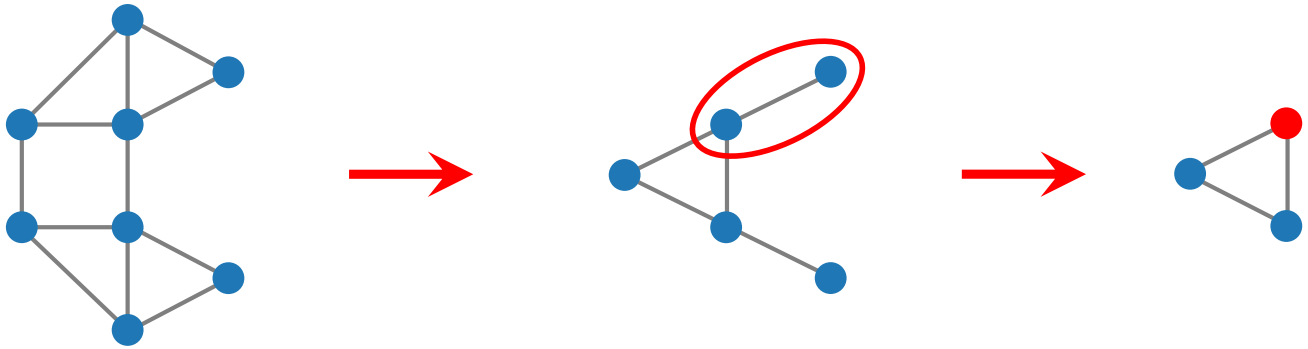
Graph Coarsening

- In each iteration, coarsening operator **clusters** sets of connected nodes to construct **super-nodes** for a coarsened graph.



Graph Coarsening

- In each iteration, coarsening operator **clusters** sets of connected nodes to construct **super-nodes** for a coarsened graph.



- Let f_c be a coarsening function such that

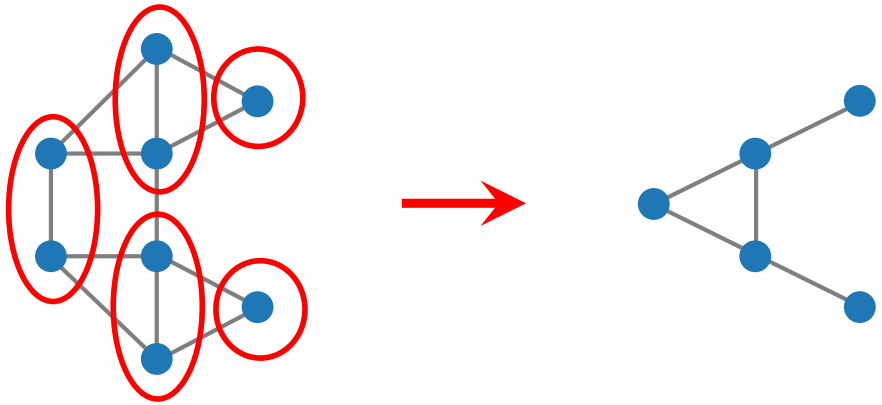
$$f_c : G \times \mathbb{R}^{|V| \times d} \longrightarrow \mathbb{R}^{+,|V| \times C}$$

Graph Coarsening

- The coarsening function f_c maps graph G and its set of node embeddings $\boldsymbol{h}_i \in \mathbb{R}^d$ to an **assignment matrix** $\boldsymbol{S} \in \mathbb{R}^{+,|V|\times C}$ which assigns each **node** v_i to one of the C **clusters**.
- Each row $\boldsymbol{S}_i \in \mathbb{R}^{+,C}$ represents strength of **association** of node v_i to each of the C different clusters over the graph.

Graph Coarsening

- The coarsening function f_c maps graph G and its set of node embeddings $\boldsymbol{h}_i \in \mathbb{R}^d$ to an **assignment matrix** $\boldsymbol{S} \in \mathbb{R}^{+,|V| \times C}$ which assigns each **node** v_i to one of the C **clusters**.
- Each row $\boldsymbol{S}_i \in \mathbb{R}^{+,C}$ represents strength of **association** of node v_i to each of the C different clusters over the graph.
- Given an assignment matrix, one can **coarsen** the graph and represent it using C **nodes**, instead.



Graph Coarsening

- The coarsening function f_c maps graph G and its set of node embeddings $\boldsymbol{h}_i \in \mathbb{R}^d$ to an **assignment matrix** $\boldsymbol{S} \in \mathbb{R}^{+,|V|\times C}$ which assigns each **node** v_i to one of the C **clusters**.
- Each row $\boldsymbol{S}_i \in \mathbb{R}^{+,C}$ represents strength of **association** of node v_i to each of the C different clusters over the graph.
- Given an assignment matrix, one can **coarsen** the graph and represent it using C **nodes**, instead.
- This coarsened representation, roughly speaking, depicts the graph on a **lower resolution**.
- The adjacency matrix for the coarsened graph G' is

$$\mathbf{A}' = \boldsymbol{S}^T \mathbf{A} \boldsymbol{S}$$

where $\mathbf{A}' \in \mathbb{R}^{+,C\times C}$.

Graph Coarsening

- Elements on the coarsened adjacency matrix $\mathbf{A}'$ may **not be binary anymore**.
- Therefore, non-zero elements on $\mathbf{A}'$ represent the **strength** of relation between different clusters.

Graph Coarsening

- Elements on the coarsened adjacency matrix $\mathbf{A}'$ may **not be binary anymore**.
- Therefore, non-zero elements on $\mathbf{A}'$ represent the **strength** of relation between different clusters.
- Similarly, we can construct **coarsened feature** representations analogue to the original graph
- The coarsened features $\boldsymbol{H}'$ would be

$$\boldsymbol{H}' = \boldsymbol{S}^T \boldsymbol{H}$$

where $\boldsymbol{H}' \in \mathbb{R}^{C \times d}$.

Graph Coarsening

- Elements on the coarsened adjacency matrix $\mathbf{A}'$ may **not be binary anymore**.
- Therefore, non-zero elements on $\mathbf{A}'$ represent the **strength** of relation between different clusters.
- Similarly, we can construct **coarsened feature** representations analogue to the original graph
- The coarsened features $\boldsymbol{H}'$ would be

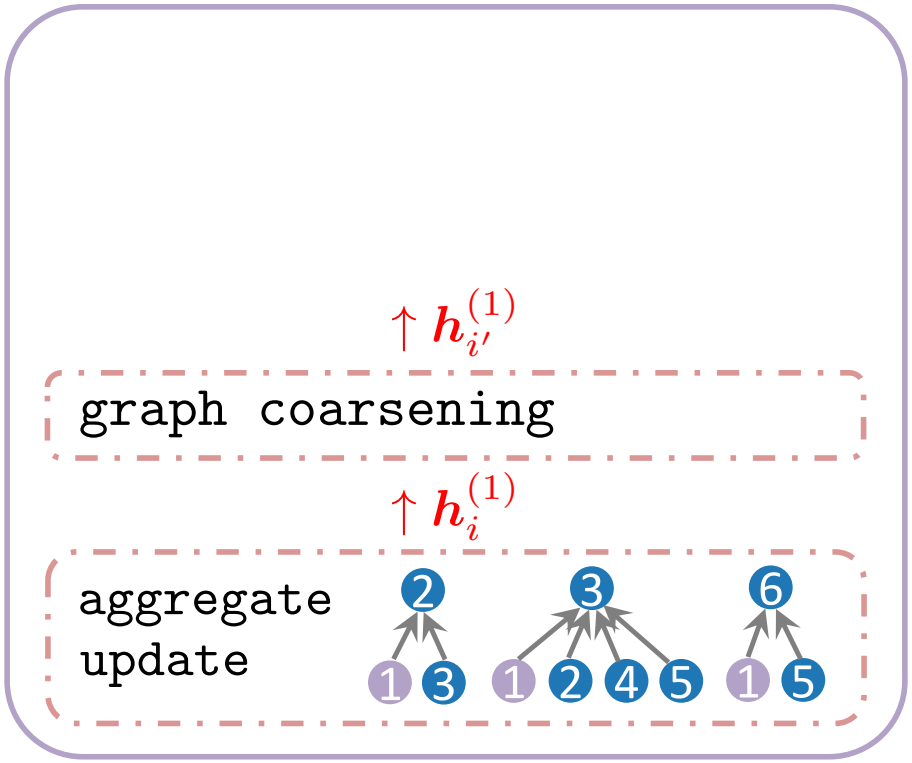
$$\boldsymbol{H}' = \boldsymbol{S}^T \boldsymbol{H}$$

where $\boldsymbol{H}' \in \mathbb{R}^{C \times d}$.

- Each row $\boldsymbol{H}_i'$ of the updated feature matrix represents an aggregation of feature representation of the corresponding cluster $\mathcal{A}_i$.

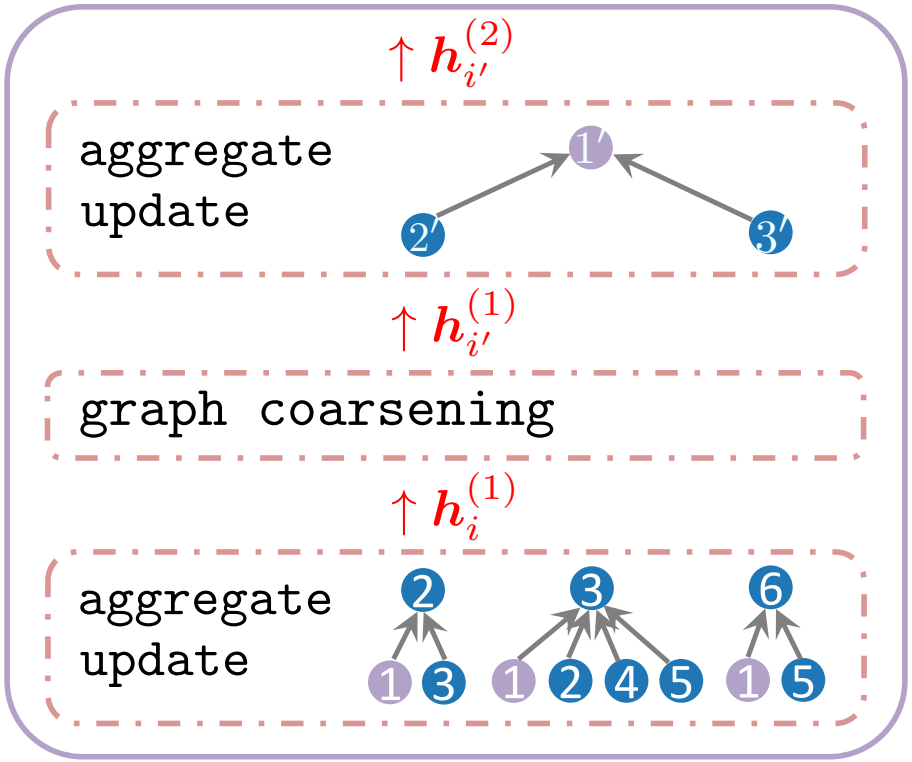
Graph Coarsening

- After **coarsening** the graph, one can apply a **new GNN layer** on the updated graph G'.
- We can continue this step on the output embedding $\boldsymbol{H}'$ until all node embeddings are aggregated into one vector to construct a graph-level feature representation.



Graph Coarsening

- After **coarsening** the graph, one can apply a **new GNN layer** on the updated graph G'.
- We can continue this step on the output embedding $\boldsymbol{H}'$ until all node embeddings are **aggregated into one vector** to construct a graph-level feature representation.



Graph Coarsening

- After **coarsening** the graph, one can apply a **new GNN layer** on the updated graph G'.
- We can continue this step on the output embedding $\boldsymbol{H}'$ until all node embeddings are **aggregated into one vector** to construct a graph-level feature representation.
- **Many approaches** like spectral clustering can be used to coarsen the graph in the pooling step.
- However, in order to train the model using backprop, the clustering function f_c used in GNNs needs to be **differentiable**.
- Alternatively, one can instead coarsen the graph by **removing nodes** in graph, rather than pooling them into a cluster.

GNN Architectures

- We have considered **aggregate** and **update** steps as the core functions to build GNNs.

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}(\{h_j^{(k)} | v_j \in N(v_i)\})$$

$$h_i^{(k+1)} = \texttt{update}(h_i^{(k)}, m_{N_i \to i}^{(k)})$$

- We have further looked at different methods to **enhance** them.

GNN Architectures

- We have considered **aggregate** and **update** steps as the core functions to build GNNs.

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}(\{h_j^{(k)} | v_j \in N(v_i)\})$$

$$h_i^{(k+1)} = \texttt{update}(h_i^{(k)}, m_{N_i \to i}^{(k)})$$

- We have further looked at different methods to **enhance** them.
- Now we look at four most popular GNN architectures
  - Graph Convolution Network (**GCN**)
  - Graph SAmple and aggreGatE (**GraphSAGE**)
  - Graph Attention Network (**GAT**)
  - Graph Isomorphism Network (**GIN**)

GCN

- In Graph Convolution Network (GCN),
  - **Aggregate:**

$$m_{N_i \to i}^{(k-1)} = \sum_{v_j \in N(v_i) \cup v_i} \frac{\boldsymbol{h}_j^{(k-1)}}{\sqrt{d_i d_j}}$$

  - Mean function is used as multi-set aggregate function.
  - Aggregation includes self-loop.
  - **Update:**

$$\boldsymbol{h}_i^{(k)} = \sigma\left(\mathbf{W}^{(k-1)} m_{N_i \to i}^{(k-1)}\right)$$

  - Update function is a single layer fully-connected network

GraphSAGE

- In Graph SAmple and aggreGatE (GraphSAGE)
  - **Aggregate:**

$$m_{N_i \to i}^{(k-1)} = \max\left(\left\{\sigma\left(\mathbf{W}_{agg}\boldsymbol{h}_j^{(k-1)} + \mathbf{b}\right) \middle| \forall v_j \in N(v_i)\right\}\right)$$

  - Aggregate function is trainable.
  - Max pooling is used as multi-set aggregate function.
  - Aggregate function uses neighborhood sampling.

  - **Update:**

$$\boldsymbol{h}_i^{(k)} = \sigma\left(\mathbf{W}^{(k)}\left[\boldsymbol{h}_i^{(k-1)} \oplus m_{N_i \to i}^{(k-1)}\right]\right)$$

  - Update function is a single layer fully-connected network.

GAT

- In Graph Attention Network (GAT)
  - **Aggregate:**

$$m_{N_i \to i}^{(k-1)} = \sum_{v_j \in N(v_i)} \frac{\exp(f_{att}^{(k-1)}(v_i, v_j))}{\sum_{v_k \in N(v_i)} \exp(f_{att}^{(k-1)}(v_i, v_k))} \boldsymbol{h}_j^{(k-1)}$$

- Aggregate function uses attention mechanism

$$f_{att}^{(k-1)}(v_i, v_j) = \sigma(\boldsymbol{a}^T [\mathbf{W}^{(k-1)} \boldsymbol{h}_i^{(k-1)} \oplus \mathbf{W}^{(k-1)} \boldsymbol{h}_j^{(k-1)}])$$

- Sum function is used as multi-set aggregate function.

  - **Update:**

$$\boldsymbol{h}_i^{(k)} = \sigma\left(\mathbf{W}^{(k-1)} m_{N_i \to i}^{(k-1)}\right)$$

- Update function is a single layer fully-connected network

GIN

- In Graph Isomorphism Network (GIN),
  - **Aggregate:**

$$m_{N_i \to i}^{(k-1)} = \sum_{v_j \in N(v_i)} \boldsymbol{h}_j^{(k-1)}$$

- Sum function is used as multi-set aggregate function.

  - **Update:**

$$\boldsymbol{h}_i^{(k)} = \mathrm{MLP}^{(k-1)}\left(\left(1+\epsilon^{(k-1)}\right)\boldsymbol{h}_i^{(k-1)} + m_{N_i \to i}^{(k-1)}\right)$$

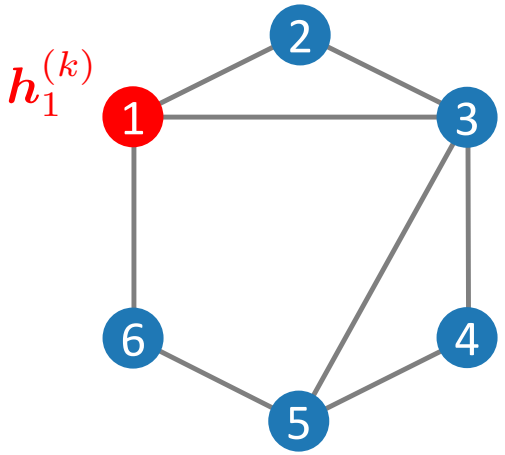
- ϵ shows the importance of the target node compared to the neighbors and can be a fix or trainable parameter.
- Update function is a deep neural network.

Generalized Message Passing

- One common point between all these methods is that they are based on neural message passing algorithms that propagate **node-level** information.

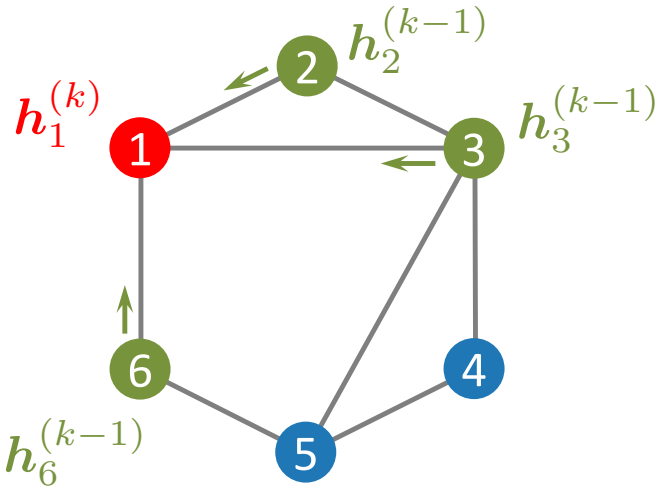
Generalized Message Passing

- One common point between all these methods is that they are based on neural message passing algorithms that propagate **node-level** information.



Generalized Message Passing

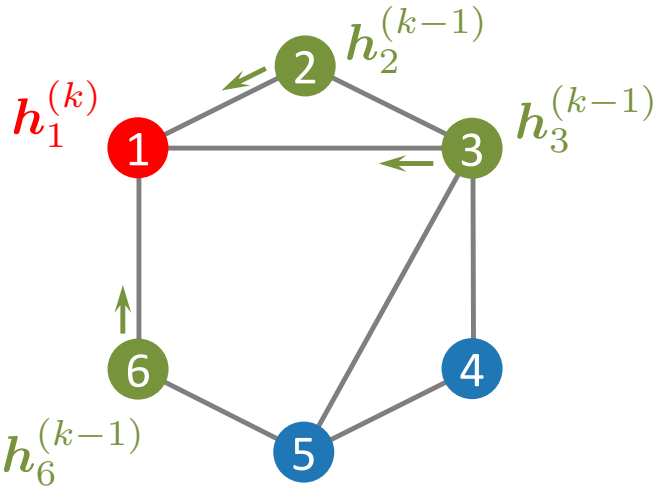
- One common point between all these methods is that they are based on neural message passing algorithms that propagate **node-level** information.



Generalized Message Passing

- One common point between all these methods is that they are based on neural message passing algorithms that propagate **node-level** information.

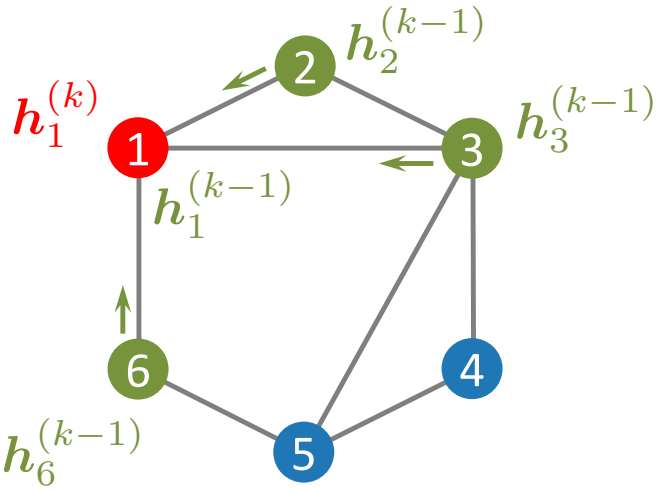
$$m_{N_i \to i}^{(k)} = \texttt{aggregate}\left(\left\{h_j^{(k-1)} \mid v_j \in N\left(v_i\right)\right\}\right)$$



Generalized Message Passing

- One common point between all these methods is that they are based on neural message passing algorithms that propagate **node-level** information.

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}\left(\left\{h_j^{(k-1)} \mid v_j \in N\left(v_i\right)\right\}\right)$$

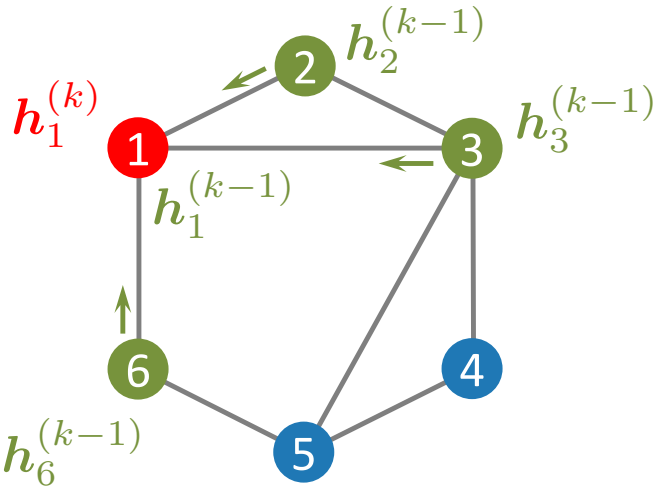


Generalized Message Passing

- One common point between all these methods is that they are based on neural message passing algorithms that propagate **node-level** information.

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}\left(\left\{h_j^{(k-1)} \mid v_j \in N\left(v_i\right)\right\}\right)$$

$$h_i^{(k)} = \texttt{update}\left(h_i^{(k-1)}, m_{N_i \to i}^{(k)}\right)$$

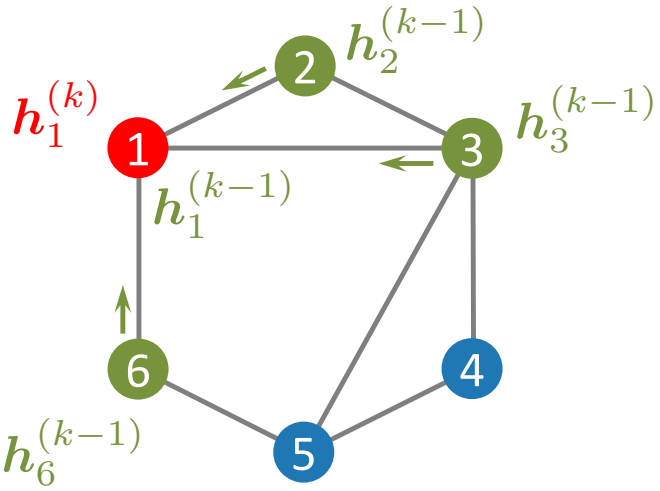


Generalized Message Passing

- One common point between all these methods is that they are based on neural message passing algorithms that propagate **node-level** information.

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}\left(\left\{h_j^{(k-1)} \mid v_j \in N\left(v_i\right)\right\}\right)$$

$$h_i^{(k)} = \texttt{update}\left(h_i^{(k-1)}, m_{N_i \to i}^{(k)}\right)$$



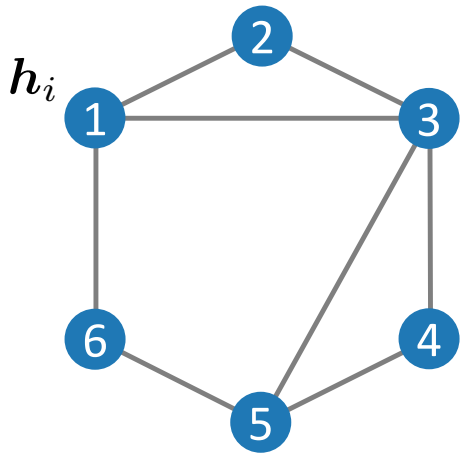
- However, one can integrate **edge-level** and **graph-level** embeddings into neural message passing propagation rule.

Generalized Message Passing

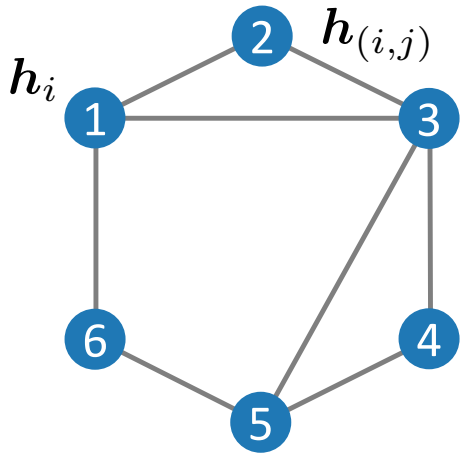
- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

Generalized Message Passing



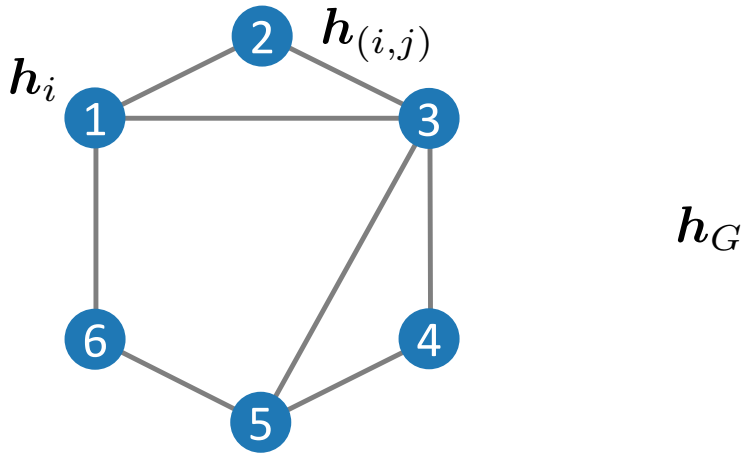
Generalized Message Passing



h_G

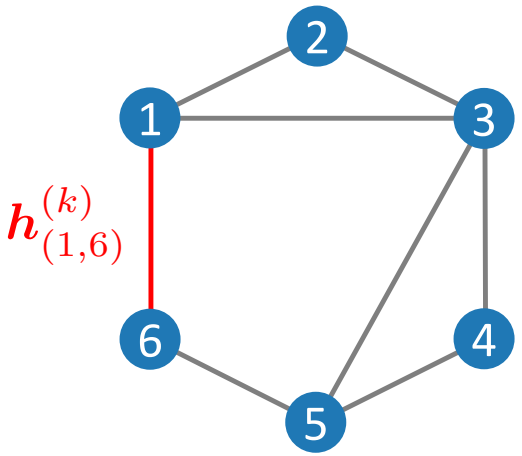
Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.



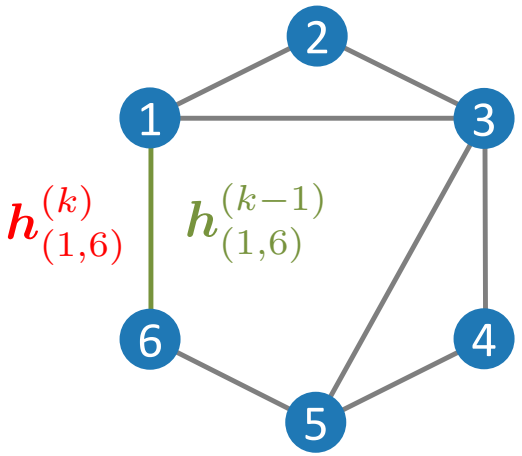
Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.



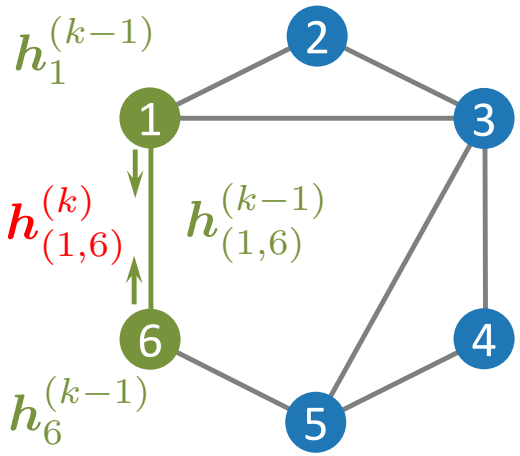
Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.



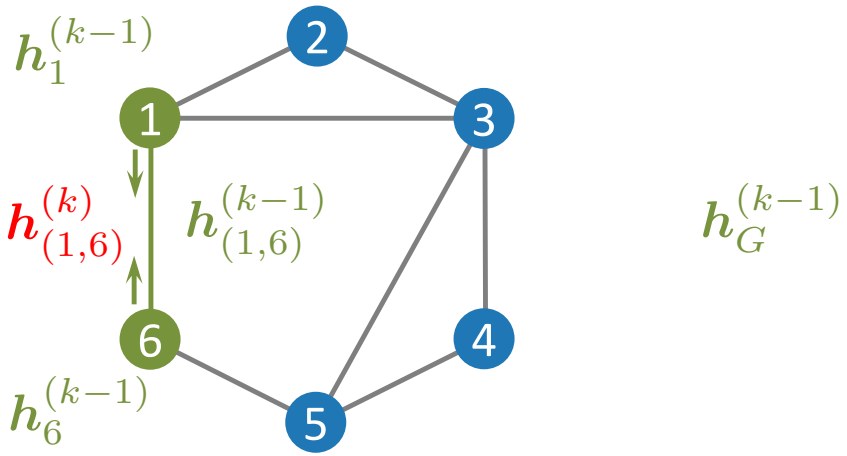
Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.



Generalized Message Passing

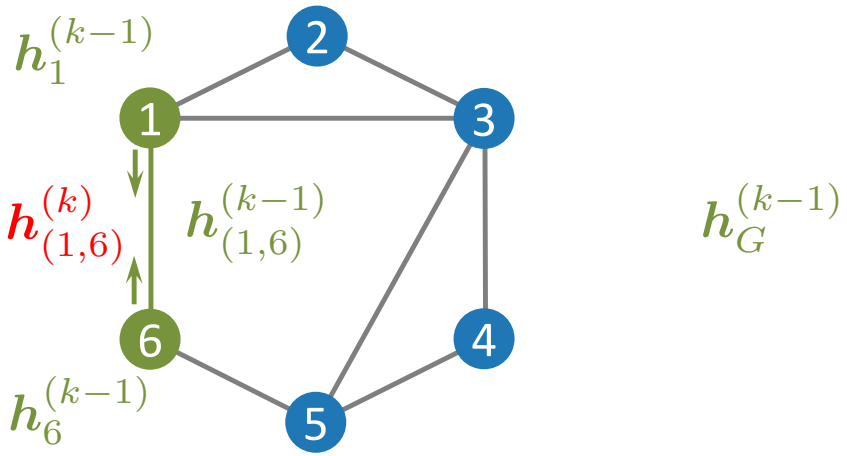
- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.



Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

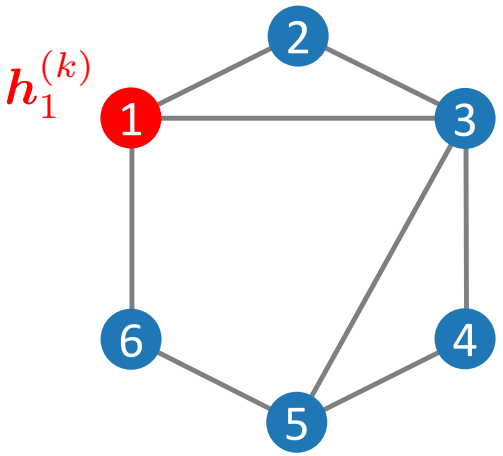
$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$



Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

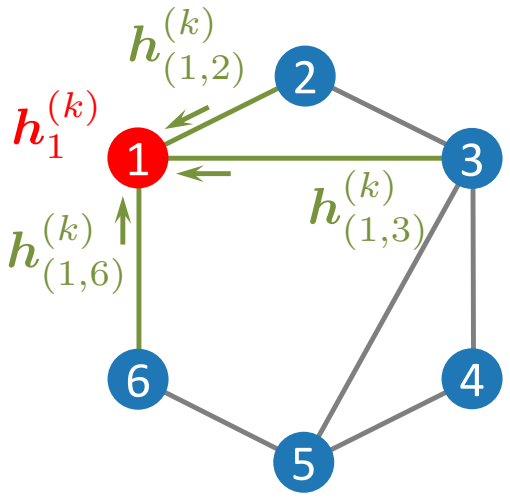
$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$



Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

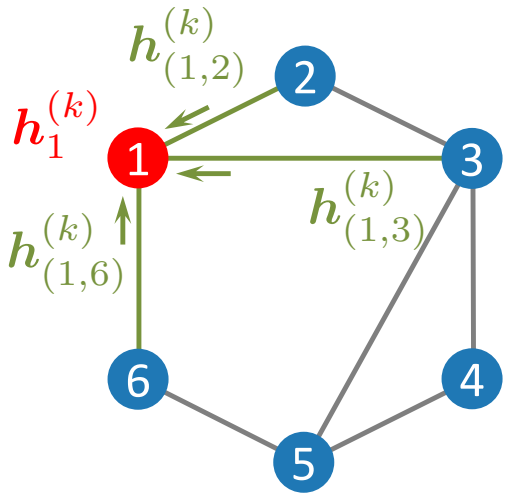


Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}_{\text{node}}\left(\left\{\boldsymbol{h}_{(i,j)}^{(k)} \,\middle|\, \forall v_j \in \mathcal{N}(v_i)\right\}\right)$$

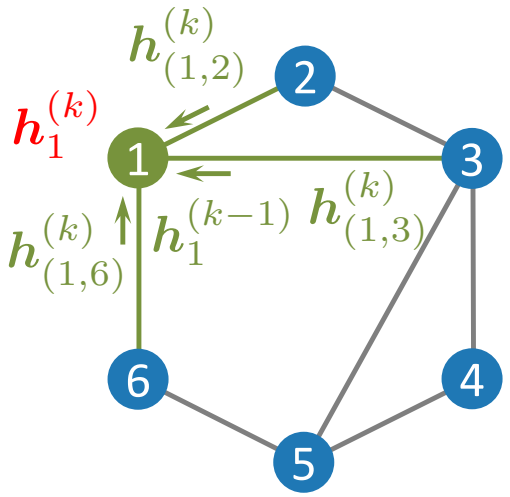


Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}_{\text{node}}\left(\left\{\boldsymbol{h}_{(i,j)}^{(k)} \,\middle|\, \forall v_j \in \mathcal{N}(v_i)\right\}\right)$$

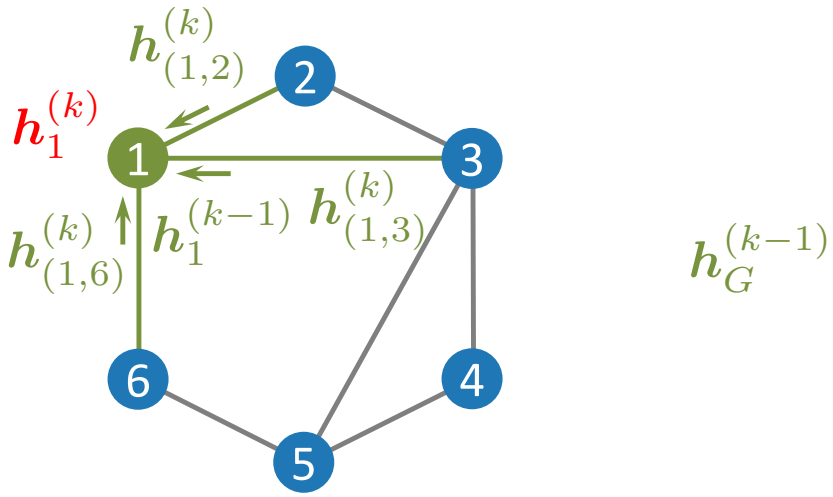


Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}_{\text{node}}\left(\left\{\boldsymbol{h}_{(i,j)}^{(k)} \middle| \forall v_j \in \mathcal{N}(v_i)\right\}\right)$$



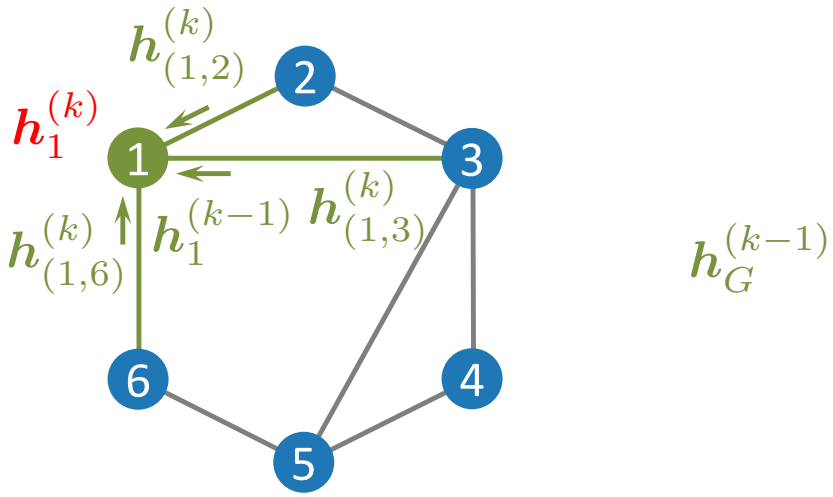
Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}_{\text{node}}\left(\left\{\boldsymbol{h}_{(i,j)}^{(k)} \,\middle|\, \forall v_j \in \mathcal{N}(v_i)\right\}\right)$$

$$\boldsymbol{h}_{i}^{(k)} = \texttt{update}_{\text{node}}\left(\boldsymbol{h}_{i}^{(k-1)}, m_{N_i \to i}^{(k)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$



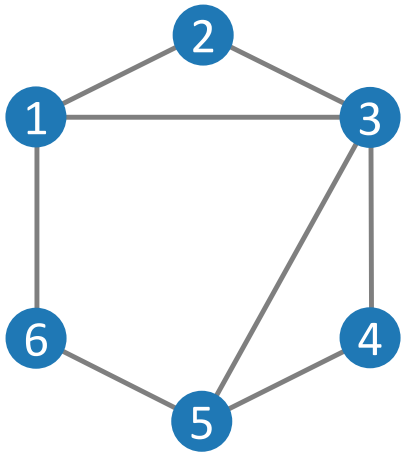
Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}_{\text{node}}\left(\left\{\boldsymbol{h}_{(i,j)}^{(k)} \,\middle|\, \forall v_j \in \mathcal{N}(v_i)\right\}\right)$$

$$\boldsymbol{h}_{i}^{(k)} = \texttt{update}_{\text{node}}\left(\boldsymbol{h}_{i}^{(k-1)}, m_{N_i \to i}^{(k)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$



$\boldsymbol{h}_{G}^{(k)}$

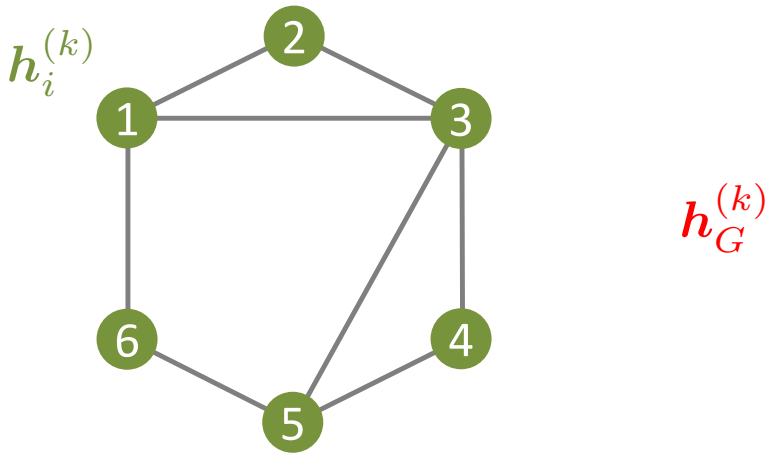
Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}_{\text{node}}\left(\left\{\boldsymbol{h}_{(i,j)}^{(k)} \,\middle|\, \forall v_j \in \mathcal{N}(v_i)\right\}\right)$$

$$\boldsymbol{h}_{i}^{(k)} = \texttt{update}_{\text{node}}\left(\boldsymbol{h}_{i}^{(k-1)}, m_{N_i \to i}^{(k)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$



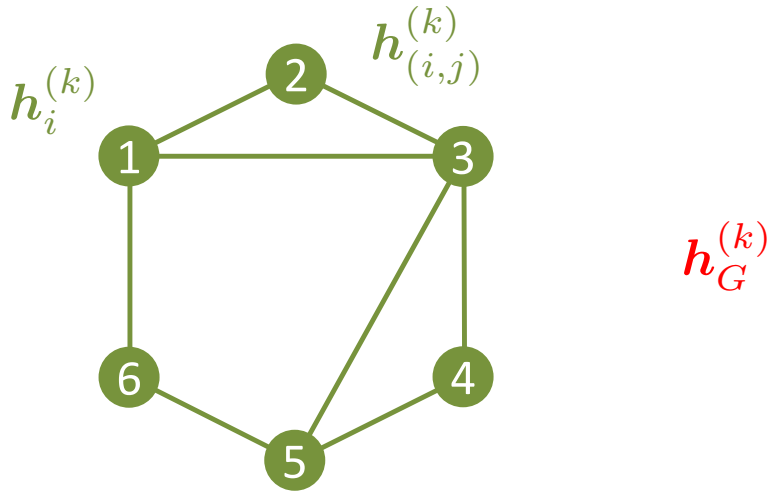
Generalized Message Passing

- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}_{\text{node}}\left(\left\{\boldsymbol{h}_{(i,j)}^{(k)} \middle| \forall v_j \in \mathcal{N}(v_i)\right\}\right)$$

$$\boldsymbol{h}_{i}^{(k)} = \texttt{update}_{\text{node}}\left(\boldsymbol{h}_{i}^{(k-1)}, m_{N_i \to i}^{(k)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$



Generalized Message Passing

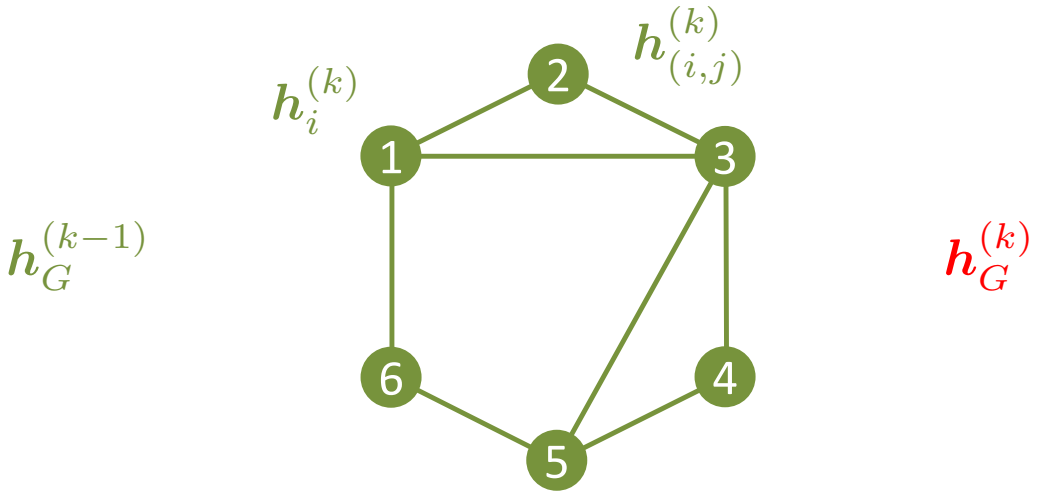
- In the **generalized** algorithm, we use **distinct** update functions to update node, edge, and graph-level embeddings.

$$\boldsymbol{h}_{(i,j)}^{(k)} = \texttt{update}_{\text{edge}}\left(\boldsymbol{h}_{(i,j)}^{(k-1)}, \boldsymbol{h}_{i}^{(k-1)}, \boldsymbol{h}_{j}^{(k-1)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$m_{N_i \to i}^{(k)} = \texttt{aggregate}_{\text{node}}\left(\left\{\boldsymbol{h}_{(i,j)}^{(k)} \middle| \forall v_j \in \mathcal{N}(v_i)\right\}\right)$$

$$\boldsymbol{h}_{i}^{(k)} = \texttt{update}_{\text{node}}\left(\boldsymbol{h}_{i}^{(k-1)}, m_{N_i \to i}^{(k)}, \boldsymbol{h}_{\mathcal{G}}^{(k-1)}\right)$$

$$\boldsymbol{h}_{\mathcal{G}}^{(k)} = \texttt{update}_{\text{graph}}\left(\boldsymbol{h}_{\mathcal{G}}^{(k-1)}, \left\{\boldsymbol{h}_{i}^{(k)} \middle| \forall v_i \in V\right\}, \left\{\boldsymbol{h}_{(i,j)}^{(k)} \middle| \forall (v_i, v_j) \in E\right\}\right)$$



Summary

- Jumping Knowledge Connections
- Graph pooling
  - Multi-Set pooling
  - Graph coarsening
- Popular GNN models
  - GAT
  - GraphSAGE
  - GCN
  - GIN
- General message passing