Scaling Up the Graph Neural Network (1)

ACMS 80770: Deep Learning with Graphs

Instructor: Navid Shervani-Tabar

Department of Applied and Comp Math and Stats

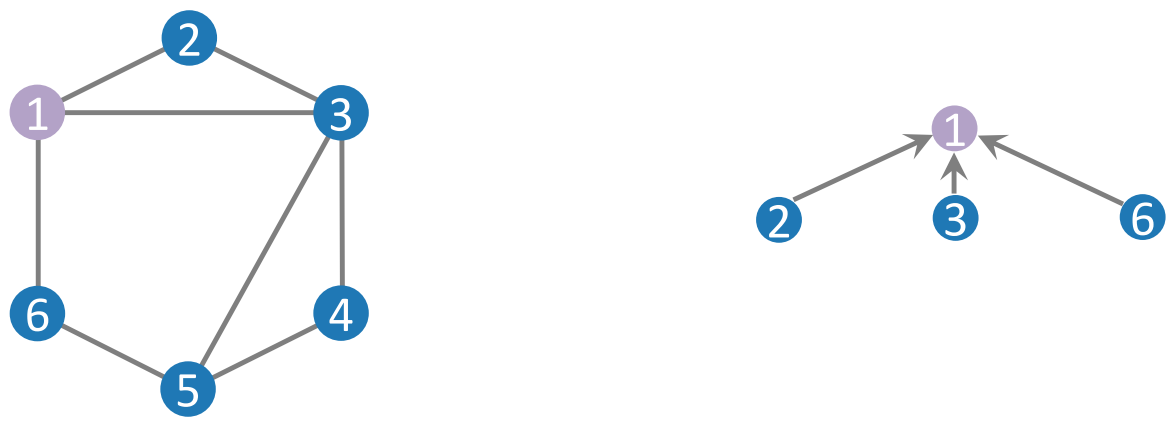
UNIVERSITY OF NOTRE DAME

Neural Message Passing

- In the previous lecture, we discussed the general **propagation rule** for a graph-based neural network.
- This propagation rule is based on the concept of **neural message passing**.

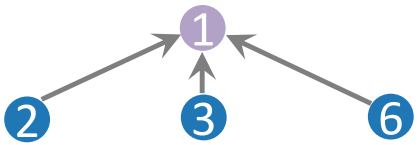
Neural Message Passing

- In the previous lecture, we discussed the general **propagation rule** for a graph-based neural network.
- This propagation rule is based on the concept of **neural message passing**.
- A single iteration of neural message passing defines a computational graph by unfolding a **subtree pattern** of height 1 rooted at each node v_i.



GNN Design

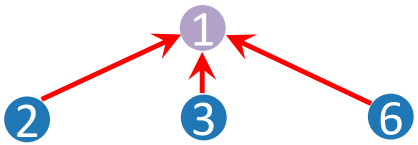
- In general, the **design space** of a graph neural network consist of the following elements:



GNN Design

- In general, the **design space** of a graph neural network consist of the following elements:
  - Aggregation operator

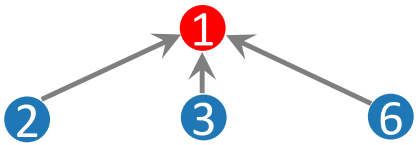
`aggregate`



GNN Design

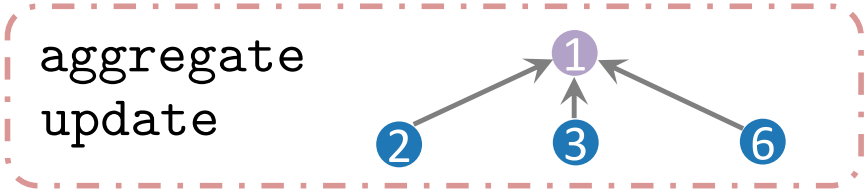
- In general, the **design space** of a graph neural network consist of the following elements:
  - Aggregation operator
  - Update operator

```
aggregate
update
```



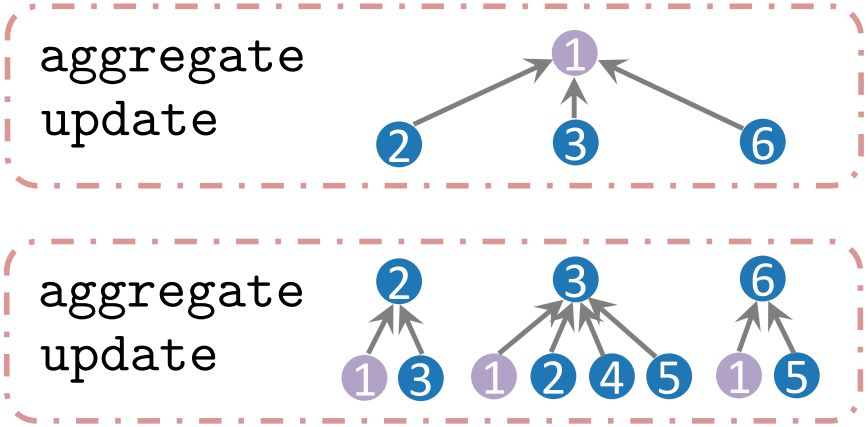
GNN Design

- In general, the **design space** of a graph neural network consist of the following elements:
  - Aggregation operator
  - Update operator



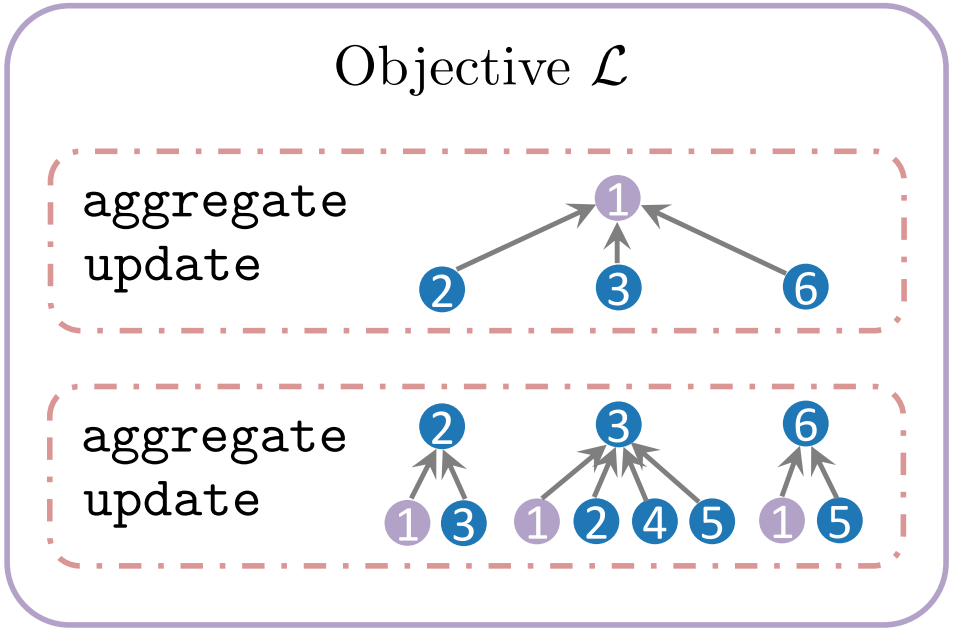
GNN Design

- In general, the **design space** of a graph neural network consist of the following elements:
  - Aggregation operator
  - Update operator
  - Staking layers



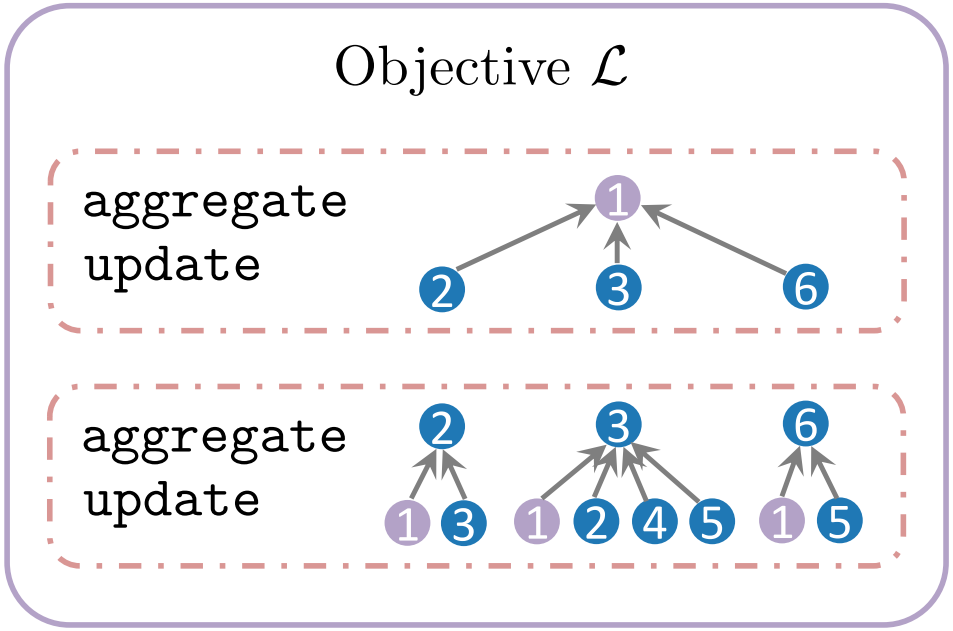
GNN Design

- In general, the **design space** of a graph neural network consist of the following elements:
  - Aggregation operator
  - Update operator
  - Staking layers
  - Objective function



GNN Design

- In general, the **design space** of a graph neural network consist of the following elements:
  - Aggregation operator
  - Update operator
  - Staking layers
  - Objective function



- In this lecture, we look at different approaches to **improve** the aggregation and update stages of the network.

Neighborhood Normalization

- The **degree** of nodes **varies** across the graph in many graph data structures.
- Suppose the **difference** between the degree of nodes is too high.
- In that case, it is reasonable to expect that the neighborhood messages $m_{N_i \to i}$ aggregated by summation **drastically differ** in magnitude.

$$\left\| \sum_{v_k \in N(v_i)} \boldsymbol{h}_k \right\| >> \left\| \sum_{v_k \in N(v_j)} \boldsymbol{h}_k \right\|$$

- This imbalance throughout the graph results in numerical **instabilities** during the training.

Neighborhood Normalization

- One remedy is to **normalize** the message.
- Given embeddings $\boldsymbol{h}_j$ of the neighbor nodes $v_j \in N(v_i)$ of node v_i with degree d_i, one can normalize the message as

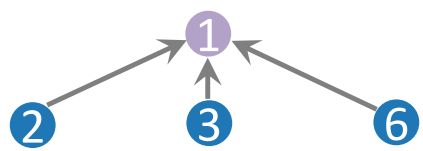
$$m_{N_i \to i} = \frac{1}{|d_i|} \sum_{v_j \in N(v_i)} \boldsymbol{h}_j$$

- Another way to alleviate the imbalance of the messages is through **symmetric normalization**.
- Mathematically put,

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

Set Pooling

- Aggregators are **multi-set functions**.
- We can improve them by **adding fully-connected layers** that extract features from input hidden node embeddings.

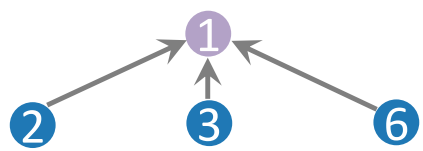


Set Pooling

- Aggregators are **multi-set functions**.
- We can improve them by **adding fully-connected layers** that extract features from input hidden node embeddings.
- An aggregation function would then take the form

$$m_{N_i \to i} = \sum_{v_j \in N(v_i)} f_{\boldsymbol{\theta}}(\boldsymbol{h}_j)$$

where $f_{\boldsymbol{\theta}}$ is a neural network parameterized by $\boldsymbol{\theta}$.

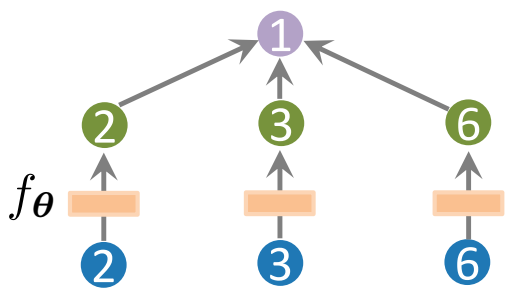


Set Pooling

- Aggregators are **multi-set functions**.
- We can improve them by **adding fully-connected layers** that extract features from input hidden node embeddings.
- An aggregation function would then take the form

$$m_{N_i \to i} = \sum_{v_j \in N(v_i)} f_{\boldsymbol{\theta}}(\boldsymbol{h}_j)$$

where $f_{\boldsymbol{\theta}}$ is a neural network parameterized by $\boldsymbol{\theta}$.

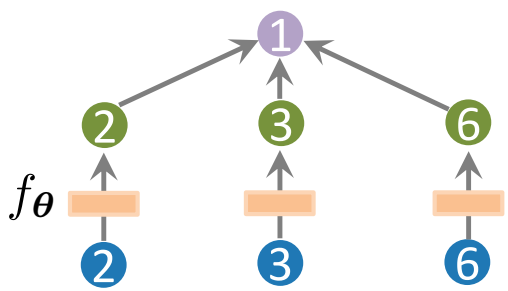


Set Pooling

- Further, we can add fully-connected layers to construct more distinctive feature representations **from the aggregated input** messages

$$m_{N_i \to i} = f_{\boldsymbol{\phi}}\Big(\sum_{v_j \in N(v_i)} f_{\boldsymbol{\theta}}(\boldsymbol{h}_j) \Big)$$

where $f_{\boldsymbol{\theta}}$ and $f_{\boldsymbol{\phi}}$ are NNs parameterized by $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$.

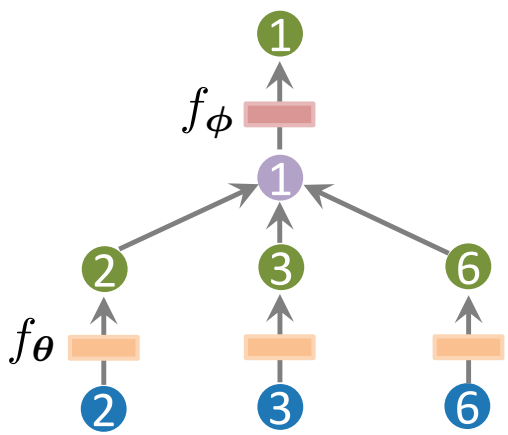


Set Pooling

- Further, we can add fully-connected layers to construct more distinctive feature representations **from the aggregated input** messages

$$m_{N_i \to i} = f_{\boldsymbol{\phi}}\Big(\sum_{v_j \in N(v_i)} f_{\boldsymbol{\theta}}(\boldsymbol{h}_j) \Big)$$

where $f_{\boldsymbol{\theta}}$ and $f_{\boldsymbol{\phi}}$ are NNs parameterized by $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$.



Set Pooling

- So far, we have used a **summation** as the aggregation function.

$$\texttt{aggregate}: \quad m_{N_i \to i}^{(k)} = \sum_{v_j \in N(v_i)} \boldsymbol{h}_j^{(k)}$$

Set Pooling

- So far, we have used a **summation** as the aggregation function.

$$\texttt{aggregate}: \quad m_{N_i \to i}^{(k)} = \sum_{v_j \in N(v_i)} \boldsymbol{h}_j^{(k)}$$

- Other **permutation invariant** functions that we can use include:
  - **Element-wise maximization**

$$\texttt{aggregate}: \quad m_{N_i \to i}^{(k)} = \max\left(\left\{\boldsymbol{h}_j^{(k)} \mid \forall v_j \in N\left(v_i\right)\right\}\right)$$

Set Pooling

- So far, we have used a **summation** as the aggregation function.

$$\texttt{aggregate}: \quad m_{N_i \to i}^{(k)} = \sum_{v_j \in N(v_i)} \boldsymbol{h}_j^{(k)}$$

- Other **permutation invariant** functions that we can use include:
  - **Element-wise maximization**

$$\texttt{aggregate}: \quad m_{N_i \to i}^{(k)} = \max\left(\left\{\boldsymbol{h}_j^{(k)} \mid \forall v_j \in N(v_i)\right\}\right)$$

  - **Element-wise minimization**

$$\texttt{aggregate}: \quad m_{N_i \to i}^{(k)} = \min\left(\left\{\boldsymbol{h}_j^{(k)} \mid \forall v_j \in N(v_i)\right\}\right)$$

Janossy Pooling

- Another approach is to define the aggregation step using functions that are **not permutation** invariant.
- As opposed to that, aggregation constructs messages using a **permutation sensitive** functions.

Janossy Pooling

- Another approach is to define the aggregation step using functions that are **not permutation** invariant.
- As opposed to that, aggregation constructs messages using a **permutation sensitive** functions.
- Let Π be the set of **all** possible **permutations** corresponding to the nodes v_j in the neighborhood $N(v_i)$ of node v_i.

Janossy Pooling

- Another approach is to define the aggregation step using functions that are **not permutation** invariant.
- As opposed to that, aggregation constructs messages using a **permutation sensitive** functions.
- Let Π be the set of **all** possible **permutations** corresponding to the nodes v_j in the neighborhood $N(v_i)$ of node v_i.
- Then, we can define the **aggregation** as

$$m_{N_i \to i} = \frac{1}{|\Pi|} \sum_{P \in \Pi} f_{\boldsymbol{\theta}}(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{|N(v_i)|})_P$$

where $f_{\boldsymbol{\theta}}$ is a permutation sensitive function, parameterized by $\boldsymbol{\theta}$.

Janossy Pooling

- Then, we can define the **aggregation** as

$$m_{N_i \to i} = \frac{1}{|\Pi|} \sum_{P \in \Pi} f_{\boldsymbol{\theta}}(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{|N(v_i)|})_P$$

  where $f_{\boldsymbol{\theta}}$ is a permutation sensitive function, parameterized by $\boldsymbol{\theta}$.

- In practice, it may **not be feasible** to compute all permutations $P \in \Pi$ of the neighborhood $N(v_i)$ for all nodes $v_i \in V$.

Janossy Pooling

- Then, we can define the **aggregation** as

$$m_{N_i \to i} = \frac{1}{|\Pi|} \sum_{P \in \Pi} f_{\boldsymbol{\theta}}(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{|N(v_i)|})_P$$

  where $f_{\boldsymbol{\theta}}$ is a permutation sensitive function, parameterized by $\boldsymbol{\theta}$.

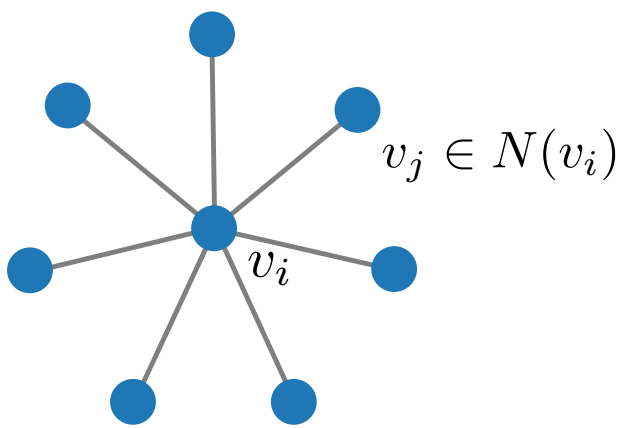
- In practice, it may **not be feasible** to compute all permutations $P \in \Pi$ of the neighborhood $N(v_i)$ for all nodes $v_i \in V$.
- An approximation to this uses only a few **randomly sampled** permutations to compute message.

$$m_{N_i \to i} = \frac{1}{T} \sum_{P \sim \Pi} f_{\boldsymbol{\theta}} \left(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{|N(v_i)|}\right)_P$$

  where T is the number of sampled permutations.

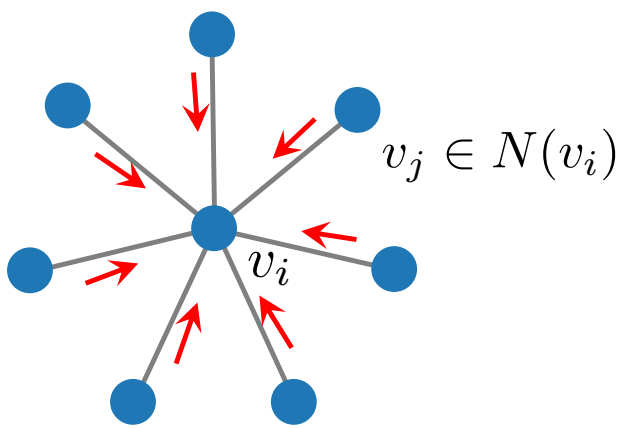
Directional Bias

- The aggregation methods discussed so far have treated all neighbors $N(v_i)$ of a node v_i **equally**.
- However, in practice, some neighbors may be more **informative** than the others.



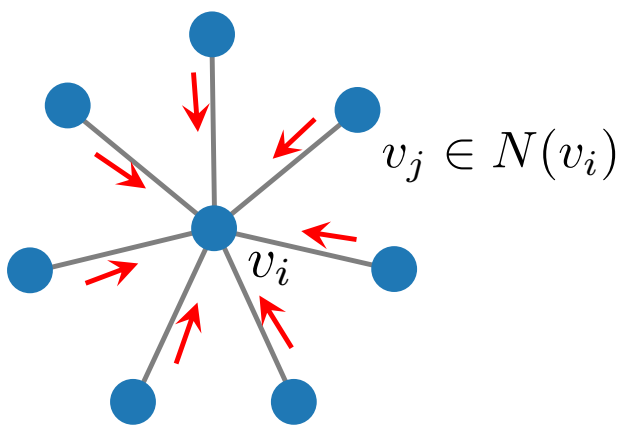
Directional Bias

- The aggregation methods discussed so far have treated all neighbors $N(v_i)$ of a node v_i **equally**.
- However, in practice, some neighbors may be more **informative** than the others.



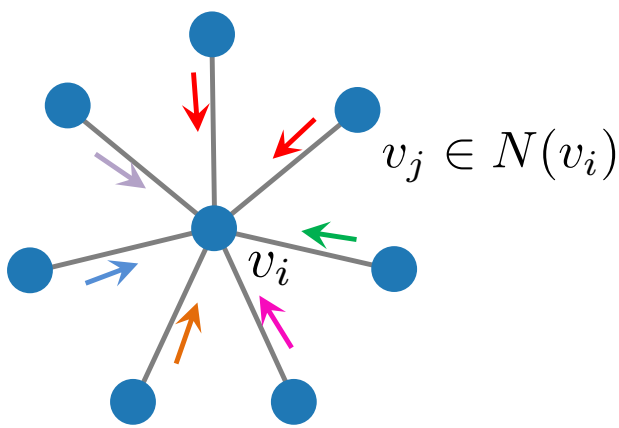
Directional Bias

- The aggregation methods discussed so far have treated all neighbors $N(v_i)$ of a node v_i **equally**.
- However, in practice, some neighbors may be more **informative** than the others.
- One idea is to modify the aggregation function to account for the **importance** of the neighbors when computing the neighborhood message $m_{N_i \to i}$.



Directional Bias

- The aggregation methods discussed so far have treated all neighbors $N(v_i)$ of a node v_i **equally**.
- However, in practice, some neighbors may be more **informative** than the others.
- One idea is to modify the aggregation function to account for the **importance** of the neighbors when computing the neighborhood message $m_{N_i \to i}$.

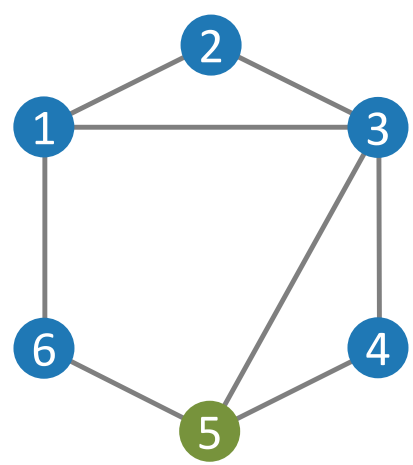


Directional Bias

- The aggregation methods discussed so far have treated all neighbors $N(v_i)$ of a node v_i **equally**.
- However, in practice, some neighbors may be more **informative** than the others.
- One idea is to modify the aggregation function to account for the **importance** of the neighbors when computing the neighborhood message $m_{N_i \to i}$.
- This can be done by assigning **learnable coefficients**, reflecting the importance of each neighbor v_j of node v_i.
- This approach is named **neighborhood attention**.

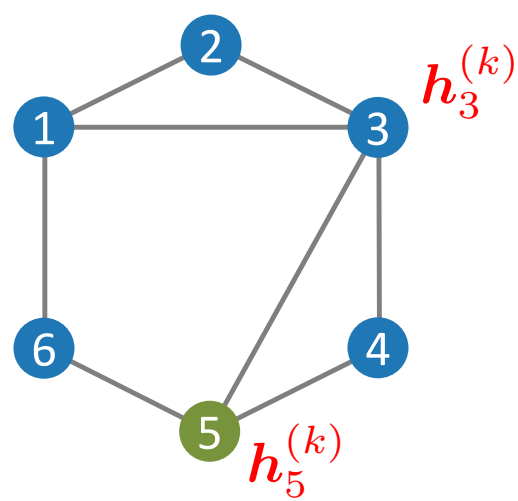
Directional Bias

- Consider node v_5 in the graph below.



Directional Bias

- Consider node v_5 in the graph below.
- Each node has an attribute vector $\boldsymbol{h}_i^{(k)} \in \mathbb{R}^{d_k}$.

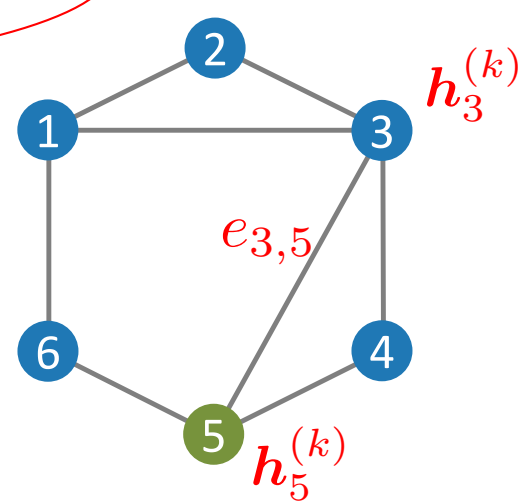


Directional Bias

- Consider node v_5 in the graph below.
- Each node has an attribute vector $\boldsymbol{h}_i^{(k)} \in \mathbb{R}^{d_k}$.
- An **attention mechanism** $f: \mathbb{R}^{d_k} \times \mathbb{R}^{d_k} \to \mathbb{R}$ takes embeddings $\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)} \in \mathbb{R}^{d_k}$ corresponding to nodes v_i and v_j as input and returns a score $e_{ij} \in \mathbb{R}$ as output

$$e_{ij} = f(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)})$$

Attention Mechanism



Directional Bias

- Consider node v_5 in the graph below.
- Each node has an attribute vector $\boldsymbol{h}_i^{(k)} \in \mathbb{R}^{d_k}$.
- An **attention mechanism** $f: \mathbb{R}^{d_k} \times \mathbb{R}^{d_k} \to \mathbb{R}$ takes embeddings $\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)} \in \mathbb{R}^{d_k}$ corresponding to nodes v_i and v_j as input and returns a score $e_{ij} \in \mathbb{R}$ as output

$$e_{ij} = f(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)})$$

- This score quantifies the importance of the corresponding edge (v_i, v_j) and is referred to as **attention coefficient**.
- However, these values depend on the magnitude of the node embeddings and hence, need to be **normalized** over the neighborhood $N(v_i)$.

Directional Bias

- One can normalize the set of scores $\{e_{ij} | v_j \in N(v_i)\}$ within neighborhood of each node v_i through a **softmax** function

$$\alpha_{ij}^{(k)} = \frac{\exp\left(f(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)})\right)}{\sum_{v_m \in N(v_i)} \exp\left(f(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_m^{(k)})\right)} = \frac{\exp\left(e_{ij}\right)}{\sum_{v_m \in N(v_i)} \exp\left(e_{im}\right)}$$

- The normalized attention weights α_{ij} can be used in the aggregation stage to account for the **importance** of different neighbors.

Directional Bias

- One can normalize the set of scores $\{e_{ij} | v_j \in N(v_i)\}$ within neighborhood of each node v_i through a **softmax** function

$$\alpha_{ij}^{(k)} = \frac{\exp\left(f(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)})\right)}{\sum_{v_m \in N(v_i)} \exp\left(f(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_m^{(k)})\right)} = \frac{\exp\left(e_{ij}\right)}{\sum_{v_m \in N(v_i)} \exp\left(e_{im}\right)}$$

- The normalized attention weights α_{ij} can be used in the aggregation stage to account for the **importance** of different neighbors.
  - For instance, for a summation aggregation function, the message has the form

$$m_{N_i \to i} = \sum_{v_j \in N(v_i)} \alpha_{ij} \boldsymbol{h}_j$$

Directional Bias

- There are various **attention mechanisms** to compute the attention coefficients.
  - One example is to use a function of the form

$$f\left(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)}\right) = \boldsymbol{a}^T \left[\boldsymbol{W}\boldsymbol{h}_i^{(k)} \oplus \boldsymbol{W}\boldsymbol{h}_j^{(k)}\right]$$

where $\boldsymbol{W} \in \mathbb{R}^{d_k \times d_{k+1}}$ and $\boldsymbol{a} \in \mathbb{R}^{2d_{k+1}}$ are trainable parameters.

Directional Bias

- There are various **attention mechanisms** to compute the attention coefficients.
- One example is to use a function of the form

$$f\left(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)}\right) = \boldsymbol{a}^T \left[\boldsymbol{W}\boldsymbol{h}_i^{(k)} \oplus \boldsymbol{W}\boldsymbol{h}_j^{(k)}\right]$$

  where $\boldsymbol{W} \in \mathbb{R}^{d_k \times d_{k+1}}$ and $\boldsymbol{a} \in \mathbb{R}^{2d_{k+1}}$ are trainable parameters.
- Another popular variant is **bilinear** attention model,

$$f\left(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)}\right) = \boldsymbol{h}_i^{(k)} \boldsymbol{W} \boldsymbol{h}_j^{(k)}$$

Directional Bias

- There are various **attention mechanisms** to compute the attention coefficients.
  - One example is to use a function of the form

$$f\left(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)}\right) = \boldsymbol{a}^T \left[\boldsymbol{W}\boldsymbol{h}_i^{(k)} \oplus \boldsymbol{W}\boldsymbol{h}_j^{(k)}\right]$$

where $\boldsymbol{W} \in \mathbb{R}^{d_k \times d_{k+1}}$ and $\boldsymbol{a} \in \mathbb{R}^{2d_{k+1}}$ are trainable parameters.

  - Another popular variant is **bilinear** attention model,

$$f\left(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)}\right) = \boldsymbol{h}_i^{(k)} \boldsymbol{W} \boldsymbol{h}_j^{(k)}$$

  - An alternative is using neural network models,

$$f\left(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)}\right) = f_{\boldsymbol{\theta}}(\boldsymbol{h}_i^{(k)}, \boldsymbol{h}_j^{(k)})$$

where $f_{\boldsymbol{\theta}}$ is a neural network parameterized by $\boldsymbol{\theta}$ that takes $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ as input and returns a scalar output.

Directional Bias

- Another approach is to use a **multi-headed** attention mechanism.
- In this approach, we learn multiple attention α_{ijk} corresponding to each pair of connected nodes (v_i, v_j).
- We compute each message corresponding to each of T attention heads using linear projection.

$$m^t_{N_i \to i} = \boldsymbol{W}_t \sum_{v_j \in N(v_i)} \alpha_{ijk} \boldsymbol{h}_j$$

- Then, the final message $m_{N_i \to i}$ is computed by concatenating all messages $m^t_{N_i \to i}$ corresponding to different attention heads.

$$m_{N_i \to i} = \left[m^1_{N_i \to i} \oplus \cdots \oplus m^T_{N_i \to i} \right]$$

Neighborhood Sampling

- In many practical scenarios, the node **degree** can be excessively **large**.
  - Instagram followers of celebrities.
- In such cases, **aggregating** information over neighbor nodes can become **hard**.

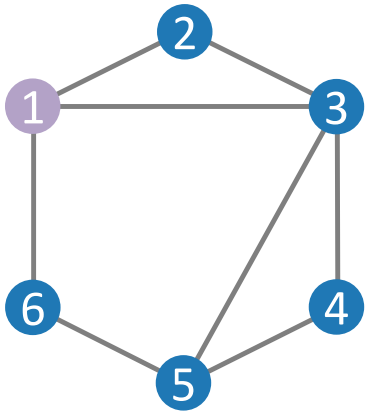
Neighborhood Sampling

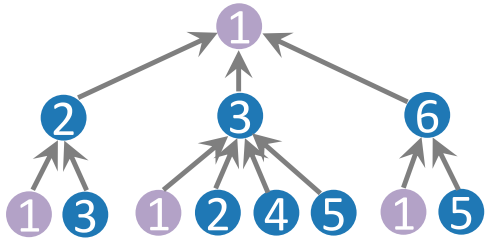
- In many practical scenarios, the node **degree** can be excessively **large**.
  - Instagram followers of celebrities.
- In such cases, **aggregating** information over neighbor nodes can become **hard**.
- To alleviate that, we can select a random **subset of neighbors** $N' \subseteq N(v_i)$ and aggregate information only from the sampled nodes.

$$m_{N_i' \to i}^{(k)} = \texttt{aggregate}\left(\left\{\boldsymbol{h}_j^{(k)} \mid v_j \in N'\right\}\right)$$

Neighborhood Sampling

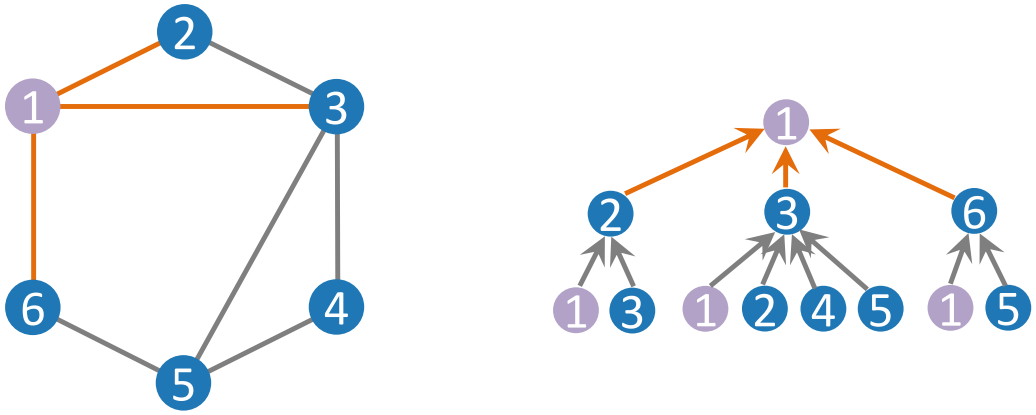
- Consider the graph below, with v_i colored in purple.





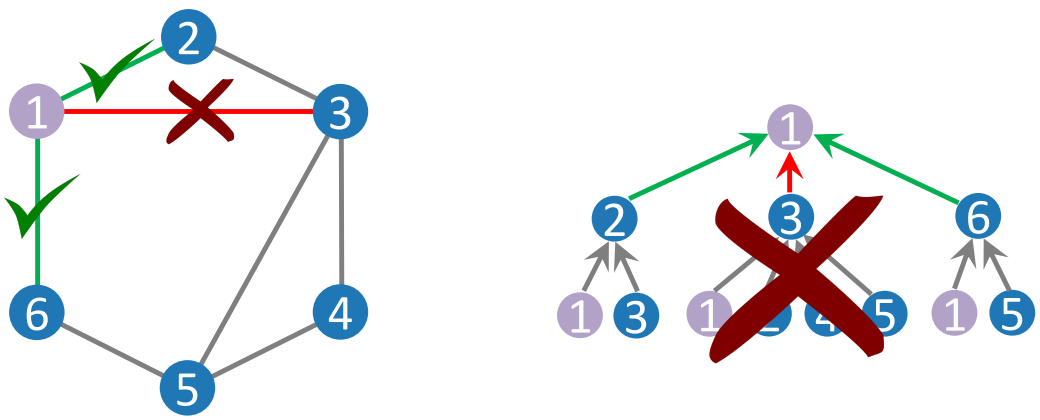
Neighborhood Sampling

- Consider the graph below, with v_i colored in purple.
- At each iteration k, we uniformly select a **fixed-size** neighborhood N' and aggregate embeddings only from $v_j \in N'$.



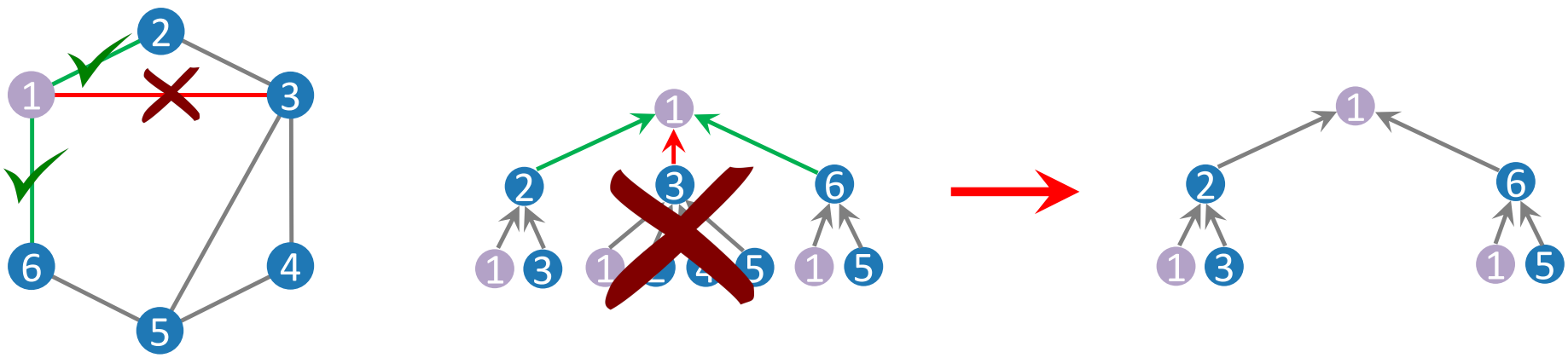
Neighborhood Sampling

- Consider the graph below, with v_i colored in purple.
- At each iteration k, we uniformly select a **fixed-size** neighborhood N' and aggregate embeddings only from $v_j \in N'$.



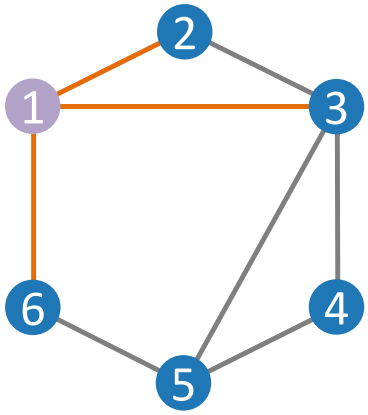
Neighborhood Sampling

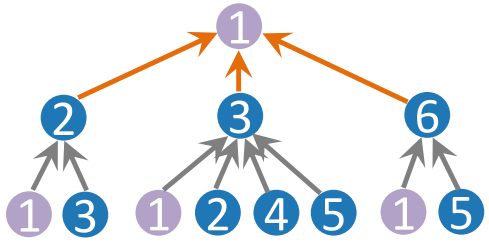
- Consider the graph below, with v_i colored in purple.
- At each iteration k, we uniformly select a **fixed-size** neighborhood N' and aggregate embeddings only from $v_j \in N'$.
- This results in a fixed memory footprint.
- However, we may then **overlook** some very important and informative neighbors.



Neighborhood Sampling

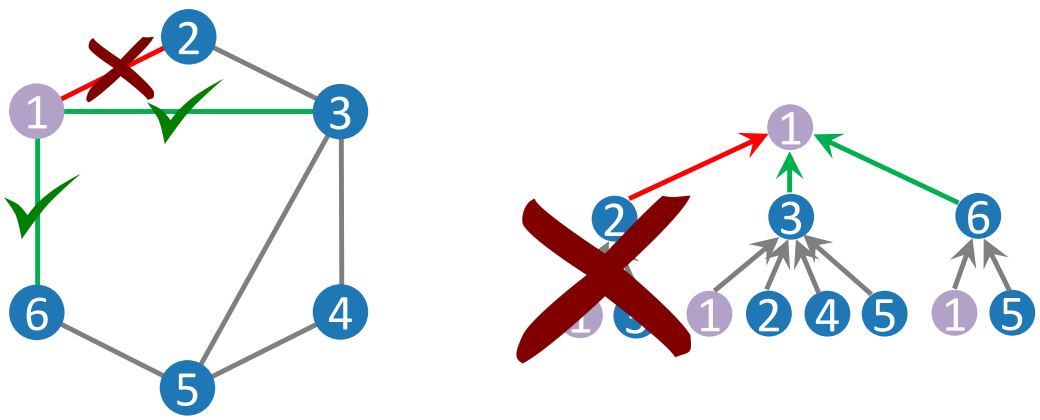
- Consider the graph below, with v_i colored in purple.
- Neighborhood sampling may result in loosing information from potentially highly informative neighbors.
- One solution is to sample a new neighborhood N' at each training iteration.





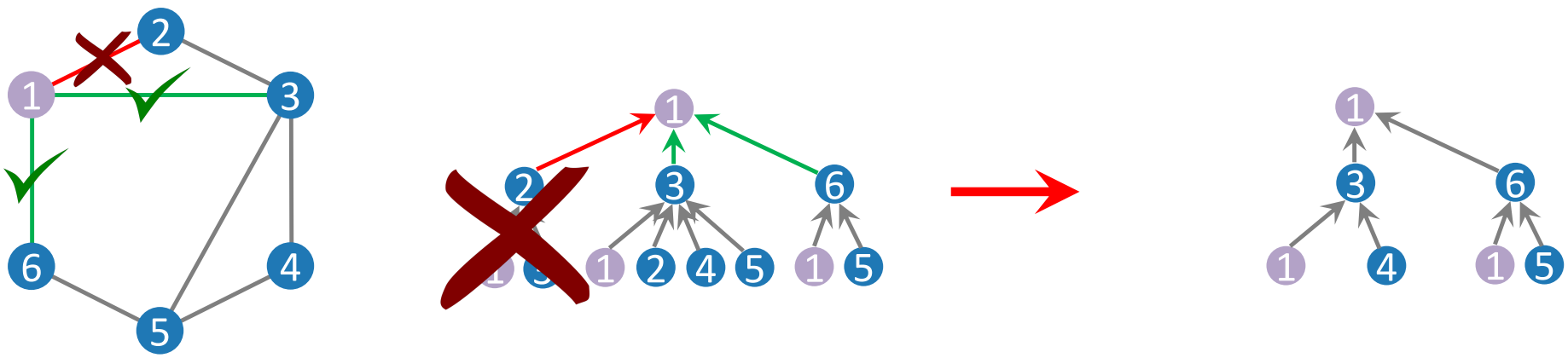
Neighborhood Sampling

- Consider the graph below, with v_i colored in purple.
- Neighborhood sampling may result in loosing information from potentially highly informative neighbors.
- One solution is to sample a new neighborhood N' at each training iteration.



Neighborhood Sampling

- Consider the graph below, with v_i colored in purple.
- Neighborhood sampling may result in loosing information from potentially highly informative neighbors.
- One solution is to sample a new neighborhood N' at each training iteration.



Update Methods

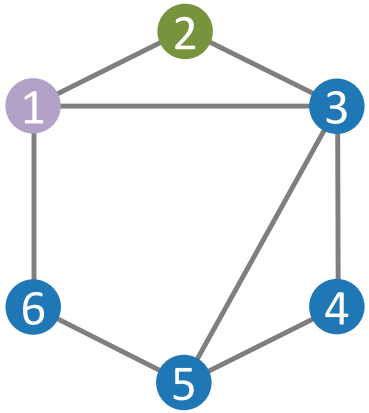
- So far, we have looked at different variations of aggregation steps.
- To achieve a higher discriminative power, GNN models should map different multisets to different representations.
- Therefore, aggregation step affects the expressivity of the GNNs.
- This has rendered aggregation the focus of GNN research.
- Next, we are looking at some issues with the update step and methods to address them.

Over-Smoothing

- When dealing with fully-connected neural networks, we can build more powerful models by **increasing the depth** of our model.
- However, this does not translate well to graph neural networks.

Over-Smoothing

- When dealing with fully-connected neural networks, we can build more powerful models by **increasing the depth** of our model.
- However, this does not translate well to graph neural networks.

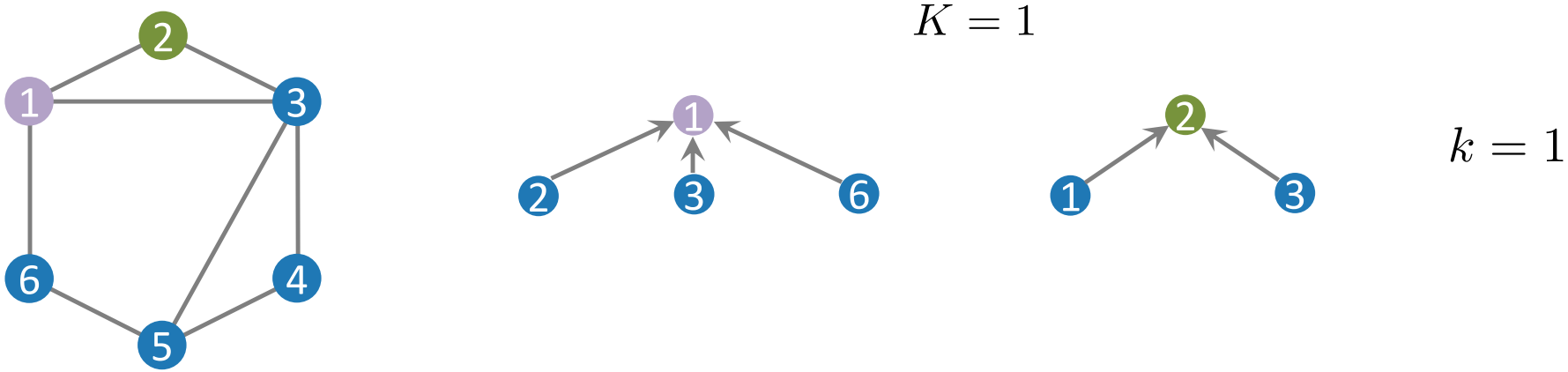


Over-Smoothing

- When dealing with fully-connected neural networks, we can build more powerful models by **increasing the depth** of our model.
- However, this does not translate well to graph neural networks.
- Recall that K iterations of the aggregation process in a neural message passing propagation rule uses an unfolded, height K **subtree pattern** rooted at node $v_i \in V$ to construct node embeddings $\boldsymbol{h}_i^{(K)}$.

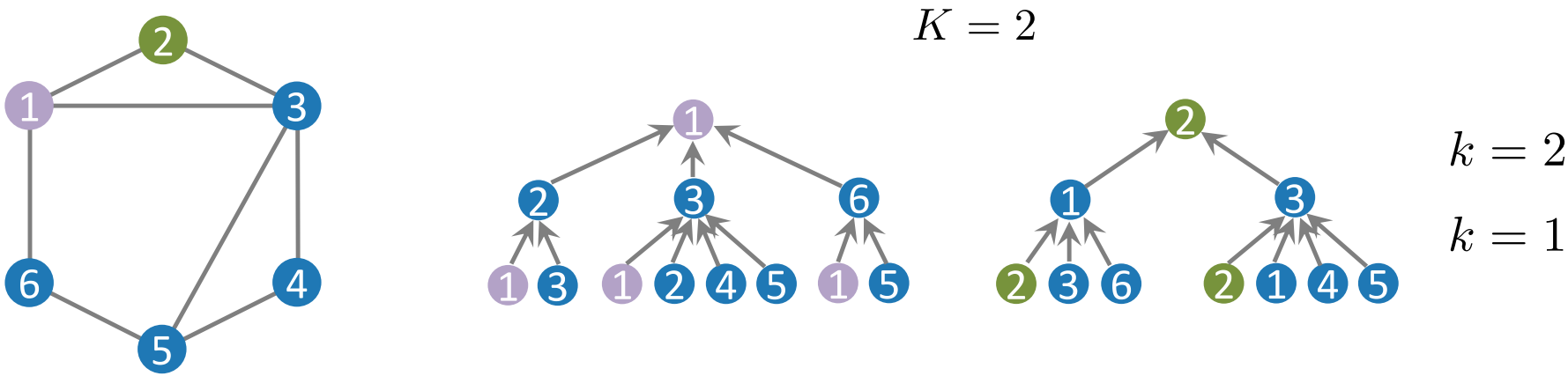
Over-Smoothing

- When dealing with fully-connected neural networks, we can build more powerful models by **increasing the depth** of our model.
- However, this does not translate well to graph neural networks.
- Recall that K iterations of the aggregation process in a neural message passing propagation rule uses an unfolded, height K **subtree pattern** rooted at node $v_i \in V$ to construct node embeddings $\boldsymbol{h}_i^{(K)}$.



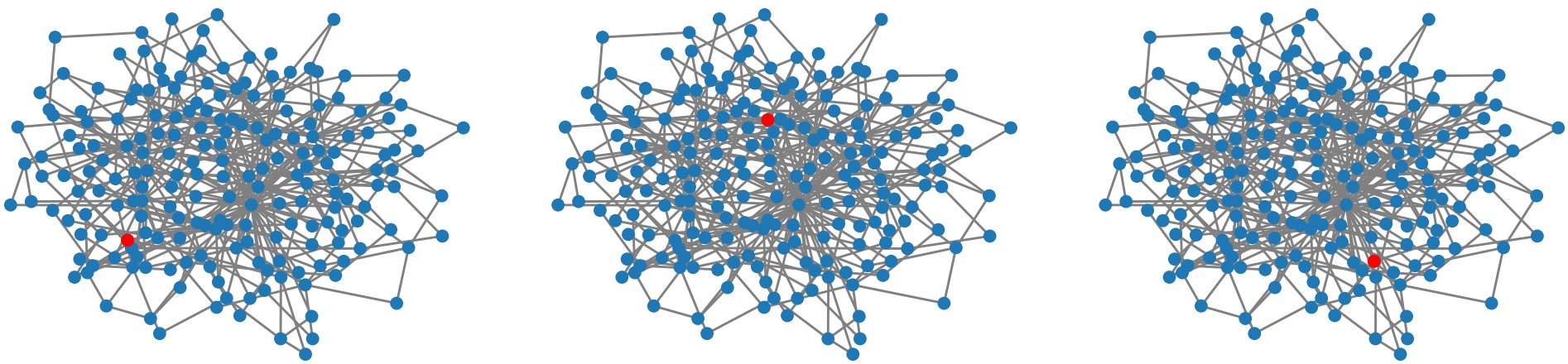
Over-Smoothing

- When dealing with fully-connected neural networks, we can build more powerful models by **increasing the depth** of our model.
- However, this does not translate well to graph neural networks.
- Recall that K iterations of the aggregation process in a neural message passing propagation rule uses an unfolded, height K **subtree pattern** rooted at node $v_i \in V$ to construct node embeddings $\boldsymbol{h}_i^{(K)}$.



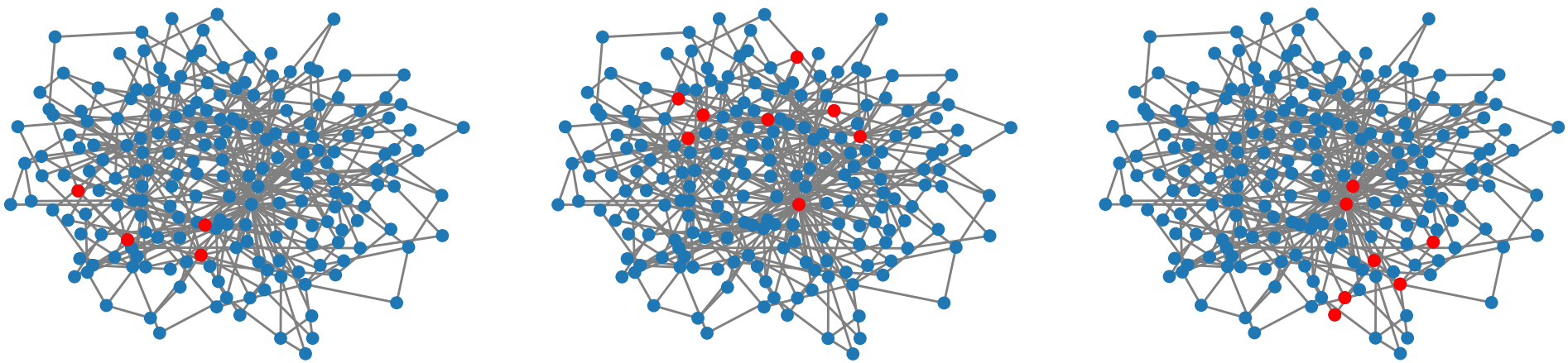
Over-Smoothing

- Consider the graph below, with v_i colored in red.



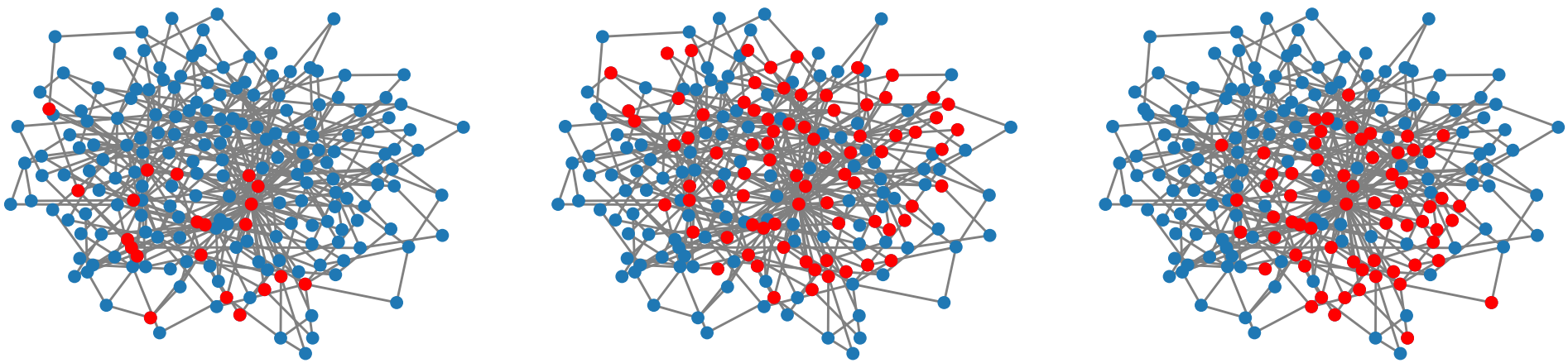
Over-Smoothing

- Consider the graph below, with v_i colored in red.
- In a single iteration of aggregation process, node v_i receives information from a neighborhood $N(v_i)$.



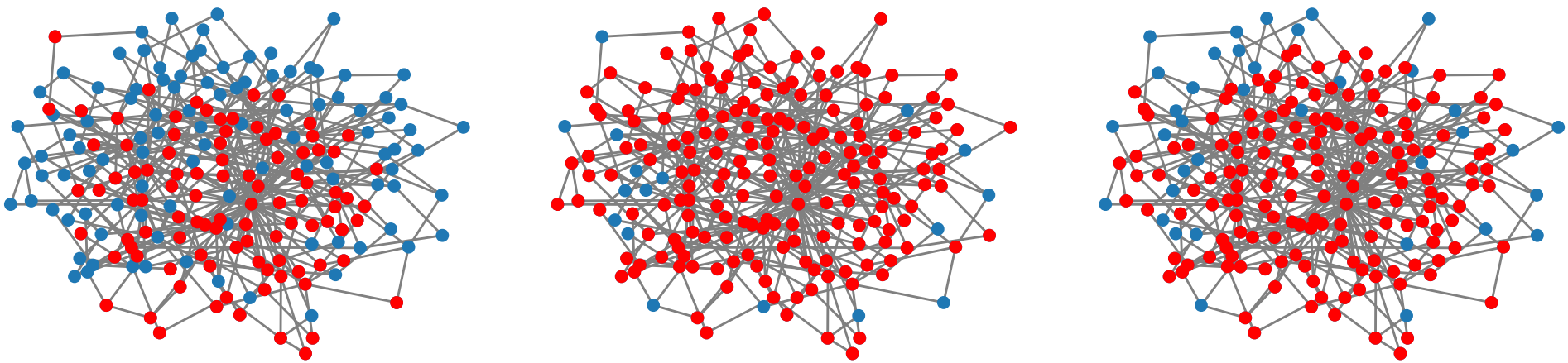
Over-Smoothing

- Consider the graph below, with v_i colored in red.
- In a single iteration of aggregation process, node v_i receives information from a neighborhood $N(v_i)$.
- In the second layer, these information are gathered from a 2 –hop neighborhood.



Over-Smoothing

- Consider the graph below, with v_i colored in red.
- In a single iteration of aggregation process, node v_i receives information from a neighborhood $N(v_i)$.
- In the second layer, these information are gathered from a 2 –hop neighborhood.
- The set of nodes that influence the embeddings $\boldsymbol{h}_i^{(k)}$ is called the **effective range** of nodes in the aggregation scheme.



Over-Smoothing

- The influence of each node embedding $\boldsymbol{h}_i^{(K)}$ by any node $v_j \in V$ in a K-layer graph neural network is quantified by the **influence score**, defined as

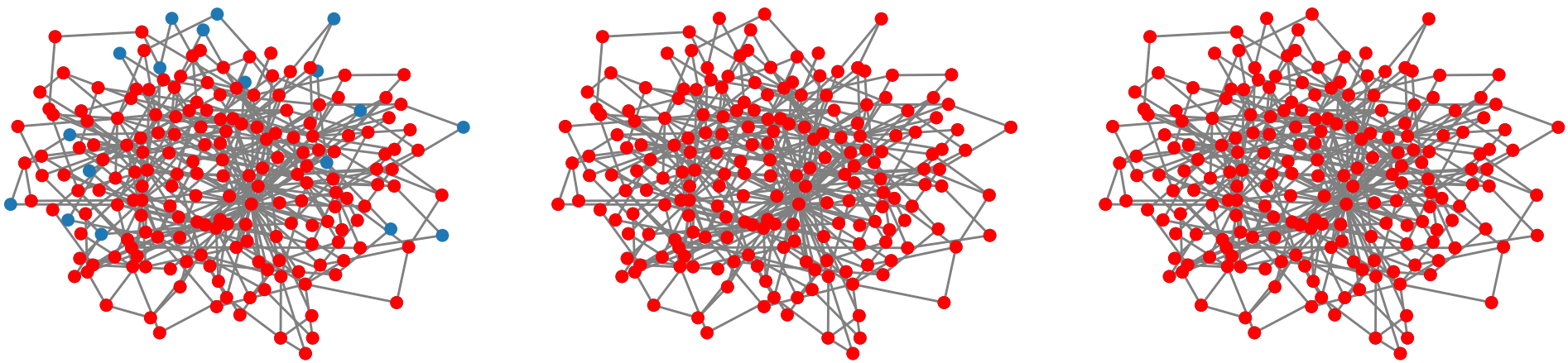
$$I_K(v_j, v_i) = \vec{\mathbf{1}}^T \left(\frac{\partial \boldsymbol{h}_i^{(K)}}{\partial \boldsymbol{h}_j^{(0)}} \right) \vec{\mathbf{1}}$$

Over-Smoothing

- The influence of each node embedding $\boldsymbol{h}_i^{(K)}$ by any node $v_j \in V$ in a K-layer graph neural network is quantified by the **influence score**, defined as

$$I_K(v_j, v_i) = \vec{\mathbf{1}}^T \left(\frac{\partial \boldsymbol{h}_i^{(K)}}{\partial \boldsymbol{h}_j^{(0)}} \right) \vec{\mathbf{1}}$$

- As the depth of GNN K increases, the effective range of different nodes **overlap** with each other.



Over-Smoothing

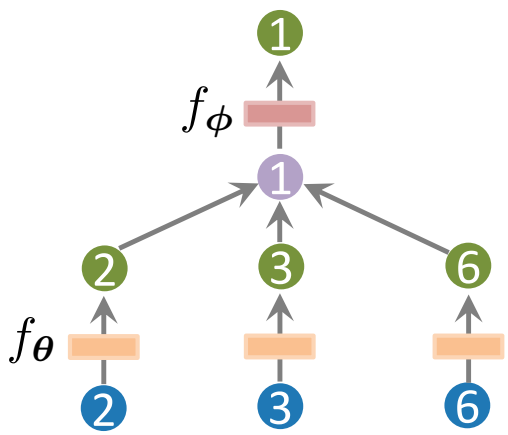
- The influence of each node embedding $\boldsymbol{h}_i^{(K)}$ by any node $v_j \in V$ in a K-layer graph neural network is quantified by the **influence score**, defined as

$$I_K(v_j, v_i) = \vec{\mathbf{1}}^T \left(\frac{\partial \boldsymbol{h}_i^{(K)}}{\partial \boldsymbol{h}_j^{(0)}} \right) \vec{\mathbf{1}}$$

- As the depth of GNN K increases, the effective range of different nodes **overlap** with each other.
- As a result, the node representations $\boldsymbol{h}_i^{(K)}$ for different nodes $v_i \in V$ become increasingly similar to each other.
- This is referred to as **over-smoothing**.
- Over-smoothing **degrades** learning and **limits** the aggregation scheme.

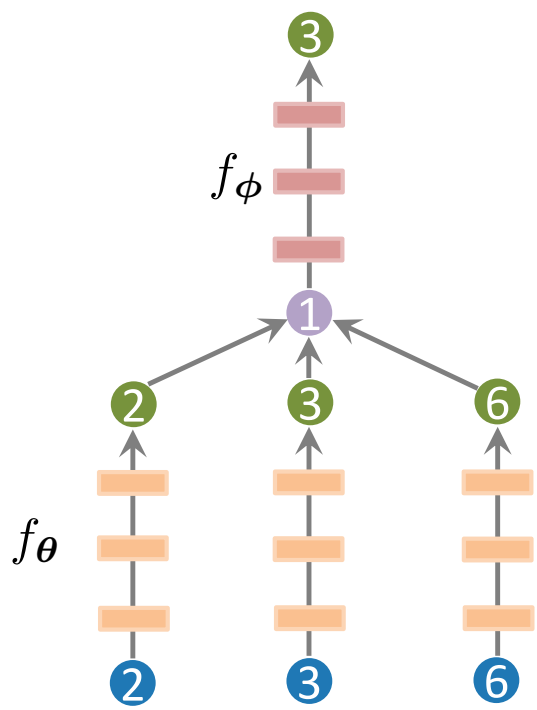
Adding Layers

- As a consequence of over-smoothing, the depth of GNN layers is limited.
- One solution to this is to use deeper fully-connected networks within each GNN layer.



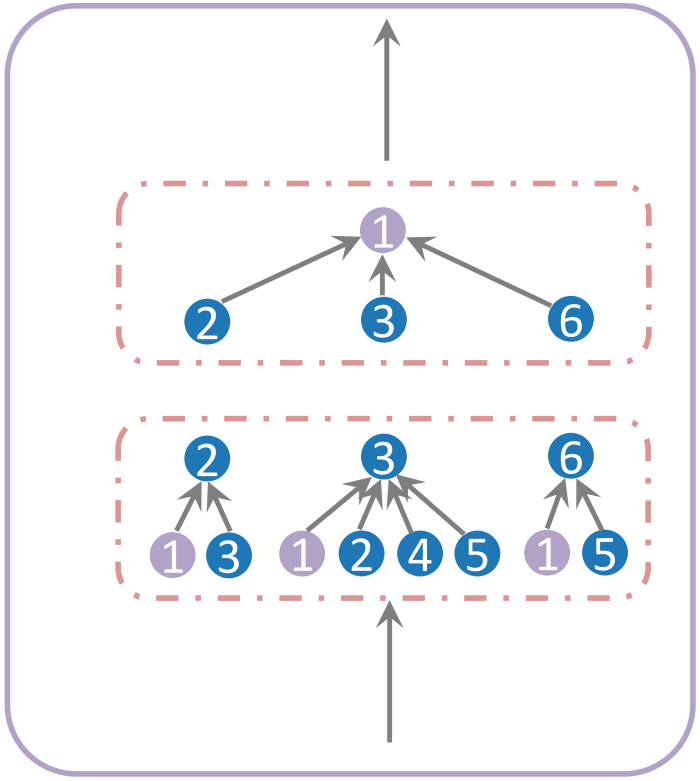
Adding Layers

- As a consequence of over-smoothing, the depth of GNN layers is limited.
- One solution to this is to use deeper fully-connected networks within each GNN layer.



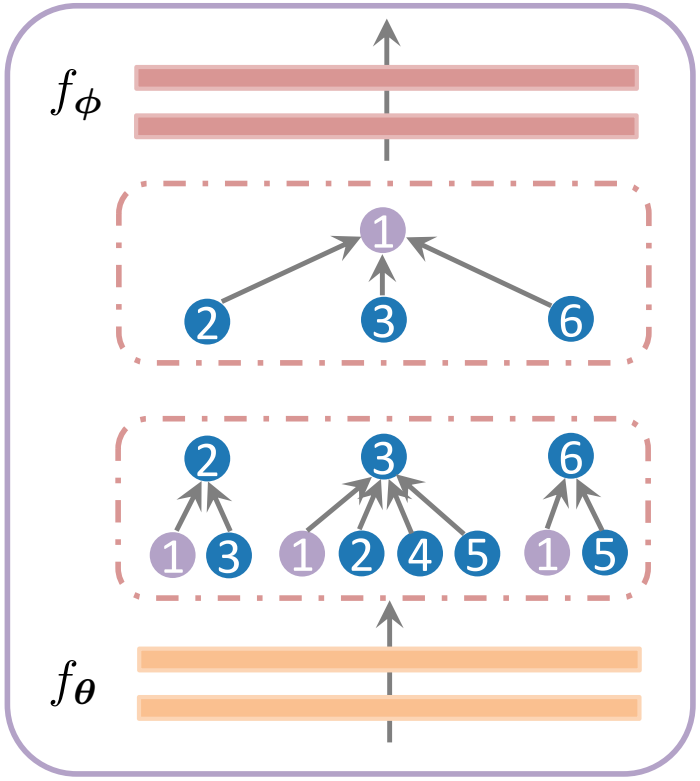
Adding Layers

- Another remedy is to apply deeper fully-connected networks before and after the GNN layers.



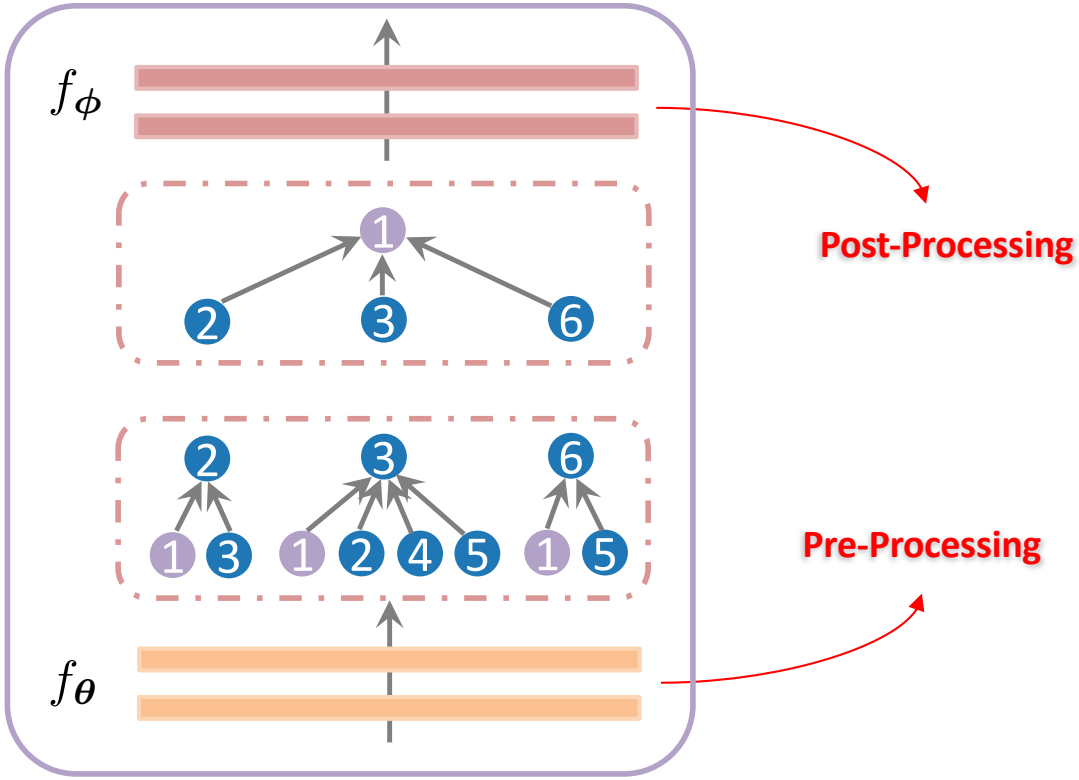
Adding Layers

- Another remedy is to apply deeper fully-connected networks before and after the GNN layers.



Adding Layers

- Another remedy is to apply deeper fully-connected networks before and after the GNN layers.



- These approaches are referred to as **pre-processing** and **post-processing**, respectively.

Skip Connection

- The **update** operator introduced here **combines** the message $m_{N_i \to i}$ from neighbors $N(v_i)$ of node v_i with the hidden activation $\boldsymbol{h}_i^{(k)}$ of node v_i to yield an **updated hidden activation** $\boldsymbol{h}_i^{(k+1)}$.
- While the embedding $\boldsymbol{h}_i^{(k)}$ is used in the update stage, after a few iterations, this information may be lost.
- Therefore, we may want to directly include the hidden activation $\boldsymbol{h}_i^{(k)}$ in $\boldsymbol{h}_i^{(k+1)}$.
- This is referred to as **skip connection**.

Skip Connection

- One approach to implement skip connections is by **concatenation**

$$\boldsymbol{h}_i^{(k+1)} = \left[\texttt{update}(\boldsymbol{h}_i^{(k)}, m_{N_i \to i}^{(k)}) \oplus \boldsymbol{h}_i^{(k)}\right]$$

Skip Connection

- One approach to implement skip connections is by **concatenation**

$$\boldsymbol{h}_i^{(k+1)} = \left[\texttt{update}(\boldsymbol{h}_i^{(k)}, m_{N_i \to i}^{(k)}) \oplus \boldsymbol{h}_i^{(k)}\right]$$

- Another approach is to instead linearly **interpolate** between these two terms.
- Mathematically put,

$$\boldsymbol{h}_i^{(k+1)} = \boldsymbol{\alpha} \odot \texttt{update}(\boldsymbol{h}_i^{(k)}, m_{N_i \to i}^{(k)}) + (1 - \boldsymbol{\alpha}) \odot \boldsymbol{h}_i^{(k)}$$

where $\alpha \in [0,1]^d$ is a gating vector with learnable parameters.

Summary

- General GNN framework
- Aggregation methods
  - Neighborhood normalization
  - Set pooling
  - Janossy pooling
  - Directional Bias
  - Neighborhood sampling
- Update methods
  - Over-smoothing
  - Adding layers
  - Skip Connections