

Note to other teachers and users of these slides: We would be delighted if you found our material useful for giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. If you make use of a significant portion of these slides in your own lecture, please include this message, or a link to our web site: <http://cs224w.Stanford.edu>

Stanford CS224W: A General Perspective on Graph Neural Networks

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

Charilaos Kanatsoulis, Stanford University

<http://cs224w.stanford.edu>



Today: A General GNN Framework

(5) Learning objective

GNN Layer 2

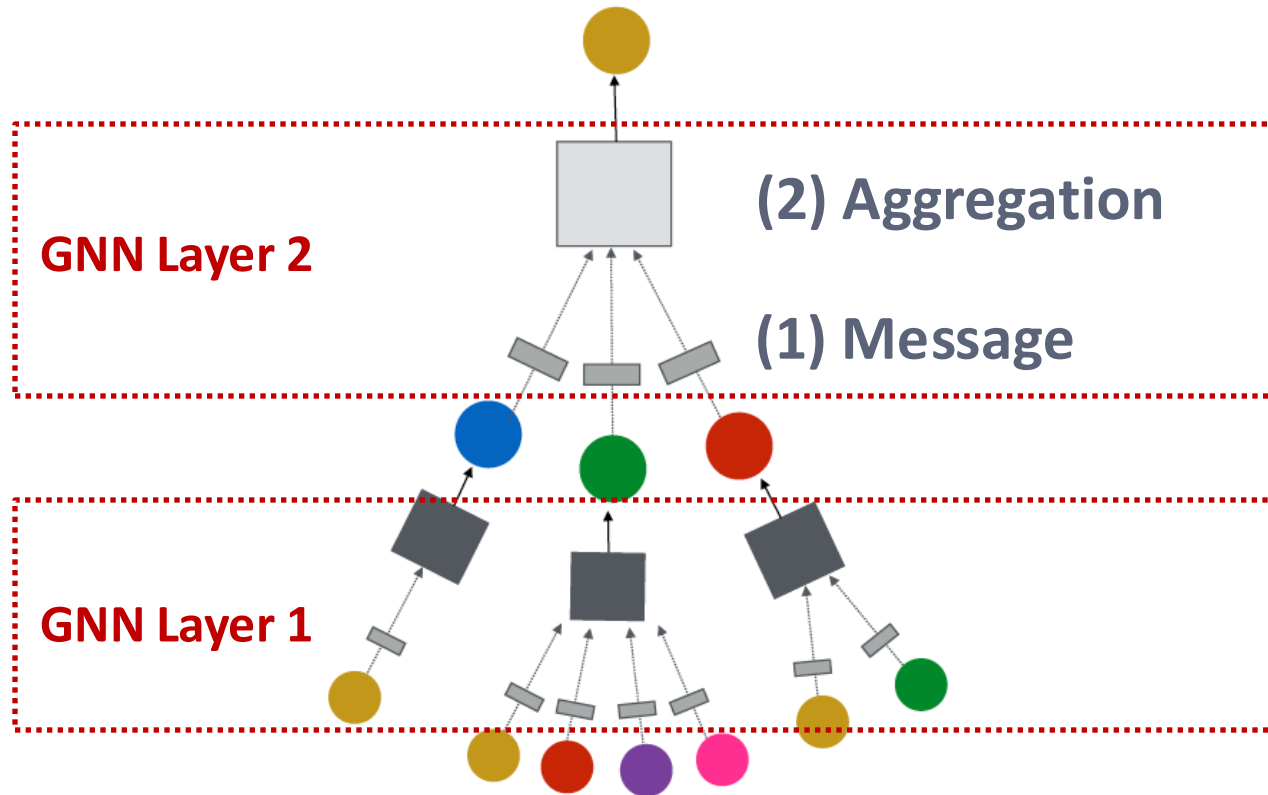
(2) Aggregation

(1) Message

(3) Layer connectivity

GNN Layer 1

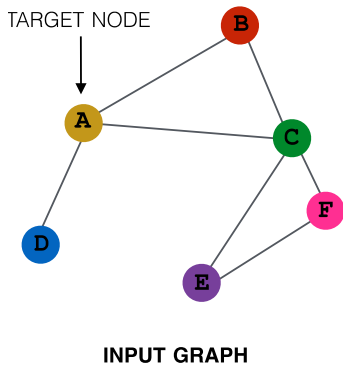
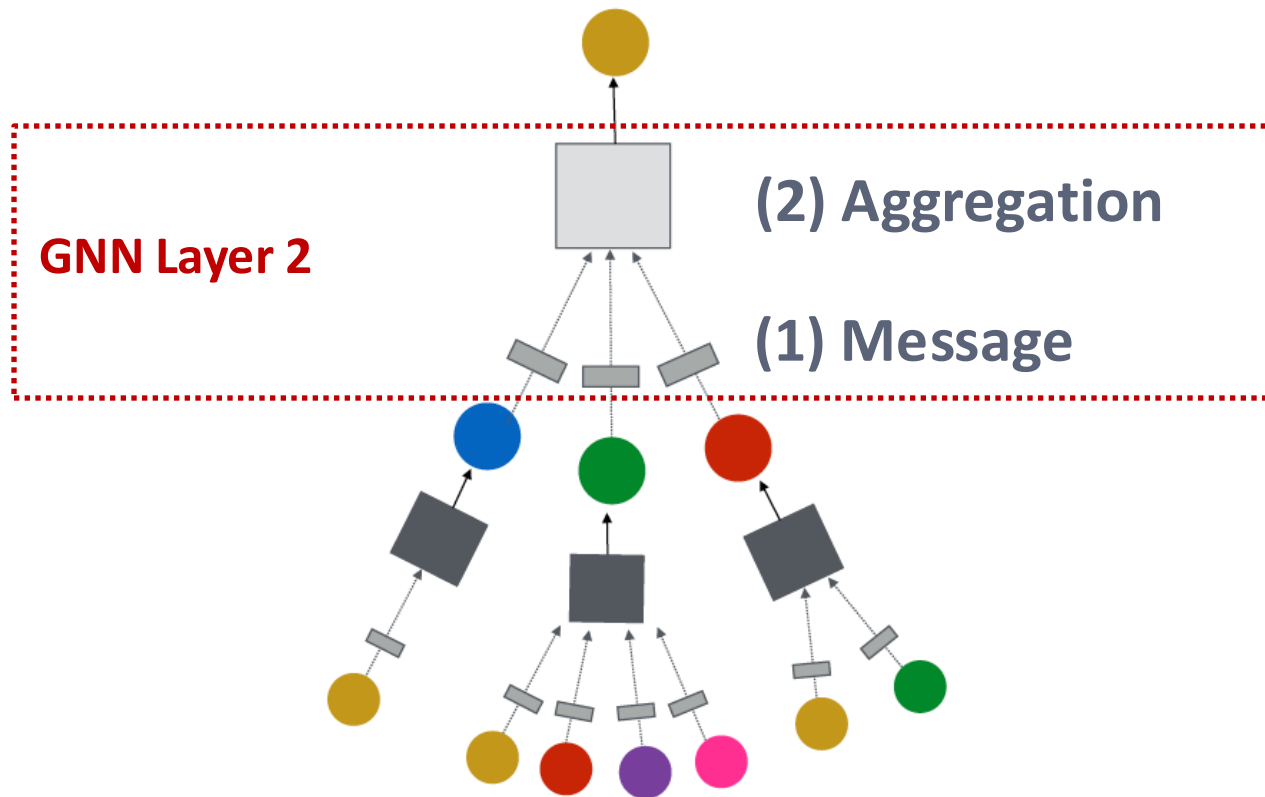
(4) Graph augmentation



A General GNN Framework (1)

GNN Layer = Message + Aggregation

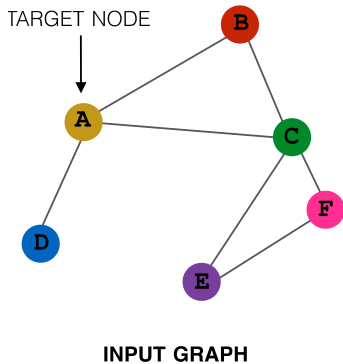
- Different instantiations under this perspective
- GCN, GraphSAGE, GAT, ...



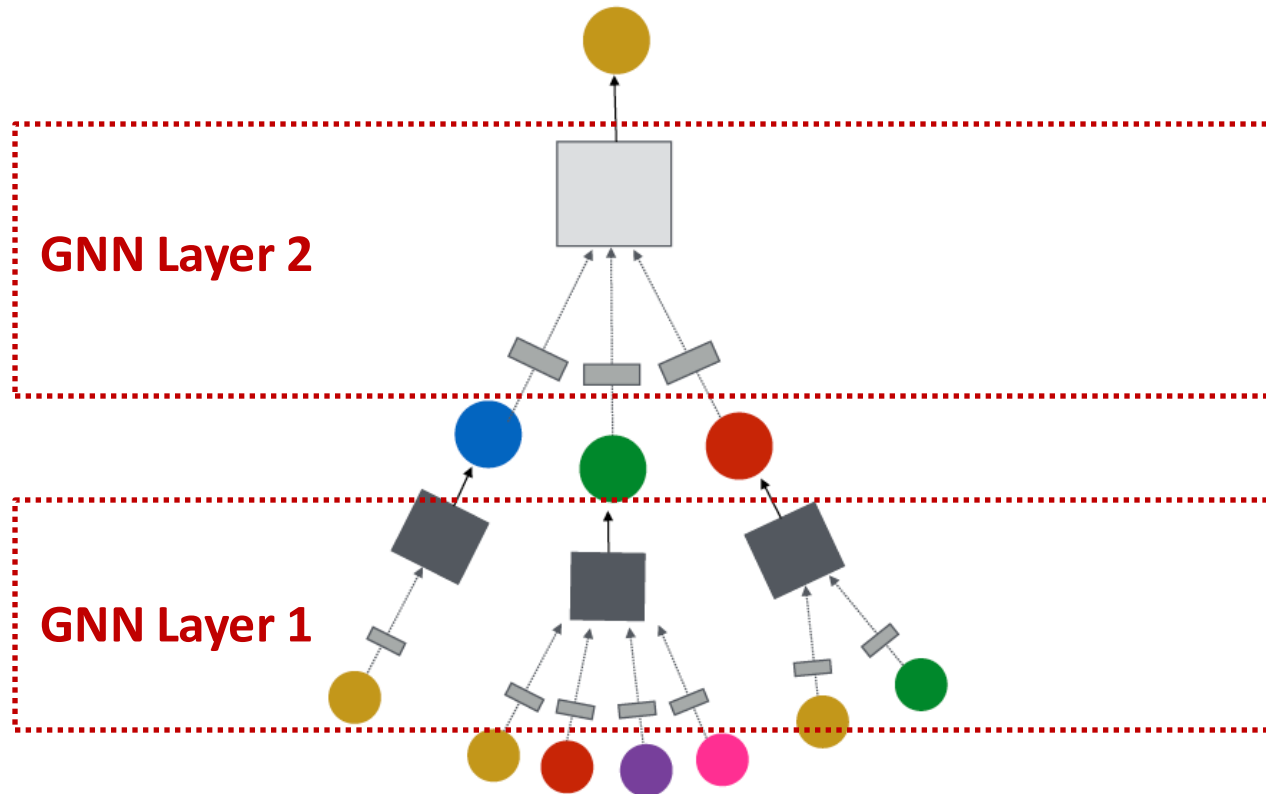
A General GNN Framework (2)

Connect GNN layers into a GNN

- Stack layers sequentially
- Ways of adding skip connections



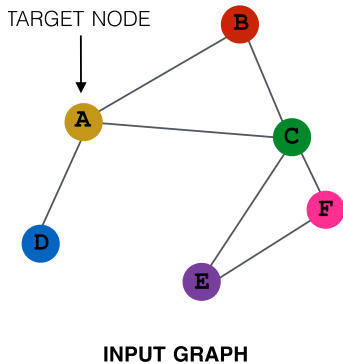
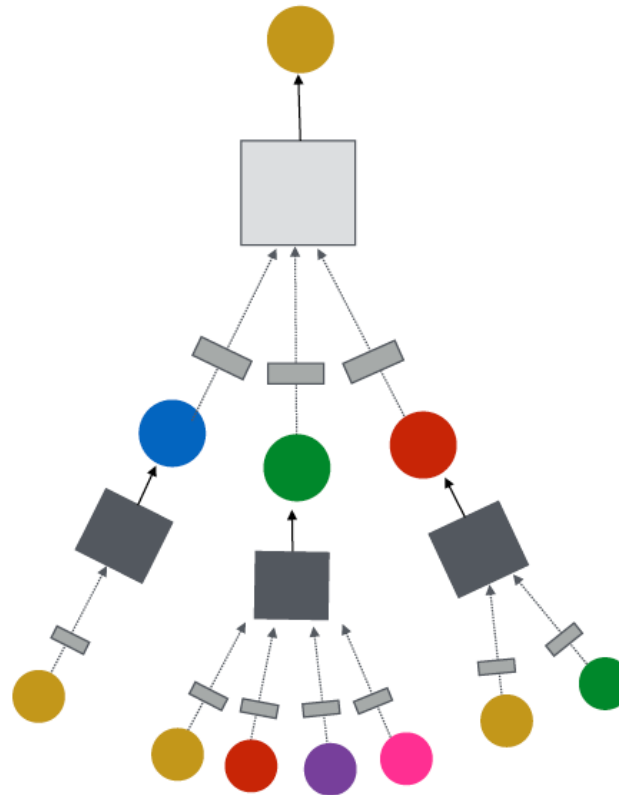
(3) Layer connectivity



A General GNN Framework (3)

Idea: Raw input graph $\neq$ computational graph

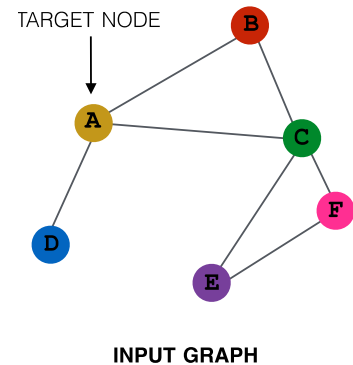
- Graph feature augmentation
- Graph structure augmentation



(4) Graph augmentation

A General GNN Framework (4)

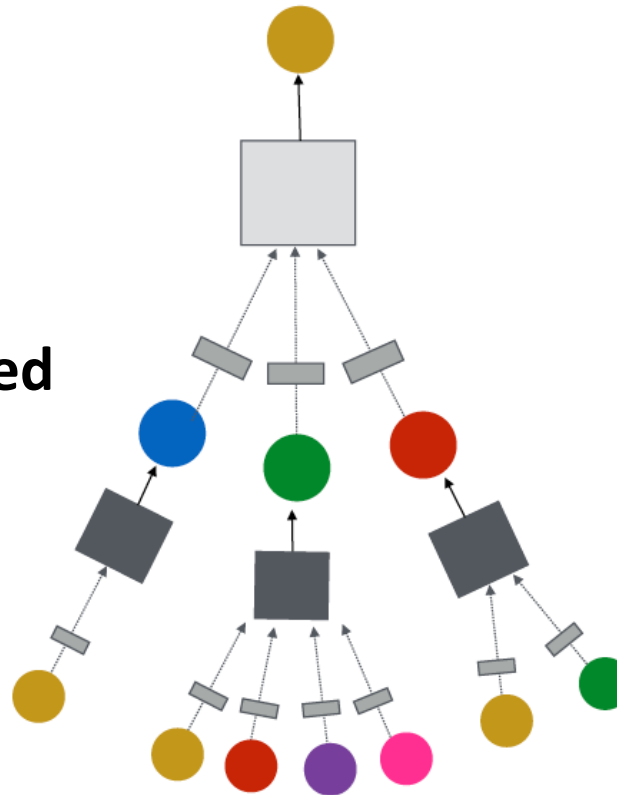
(5) Learning objective



How do we train a GNN:

- Supervised/Unsupervised objectives
- Node/Edge/Graph level objectives

(We will discuss all these later in the class)



GNN Framework: Summary

(5) Learning objective

GNN Layer 2

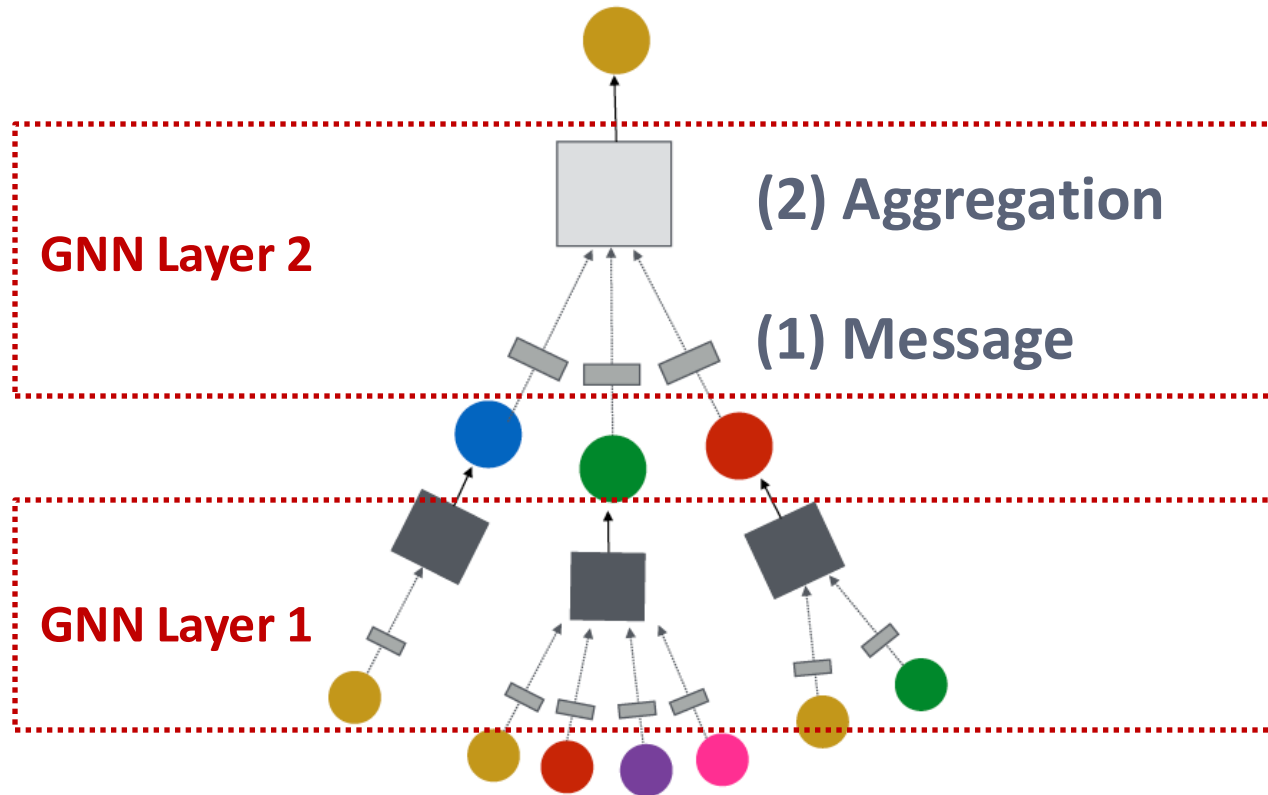
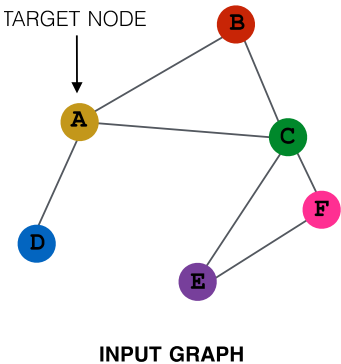
(2) Aggregation

(1) Message

(3) Layer connectivity

GNN Layer 1

(4) Graph augmentation



Stanford CS224W: A Single Layer of a GNN

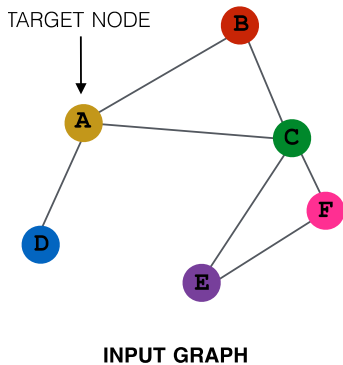
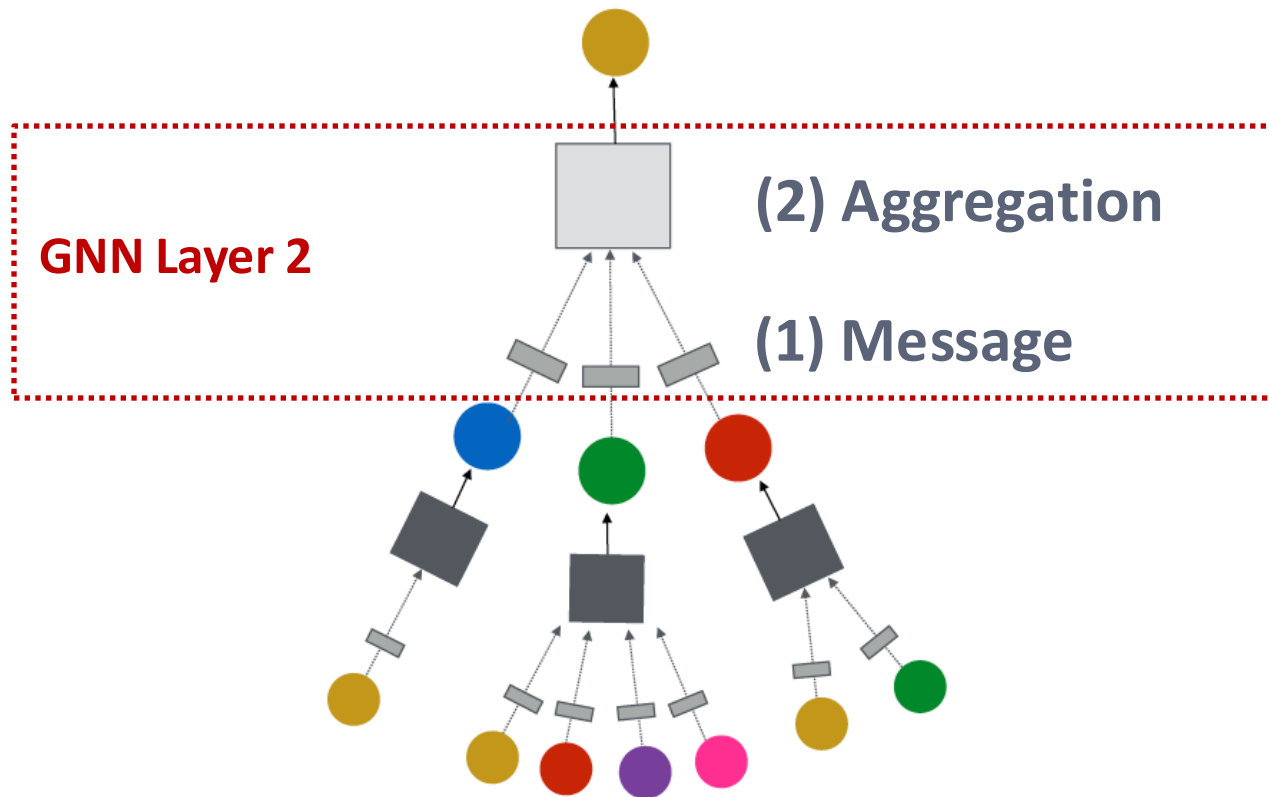
CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
<http://cs224w.stanford.edu>



A GNN Layer

GNN Layer = Message + Aggregation

- Different instantiations under this perspective
- GCN, GraphSAGE, GAT, ...



A Single GNN Layer

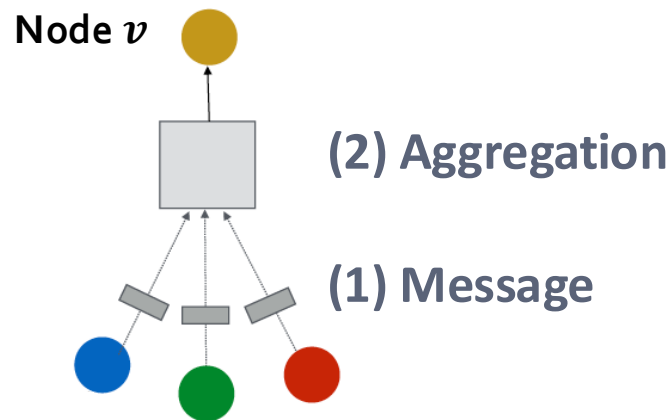
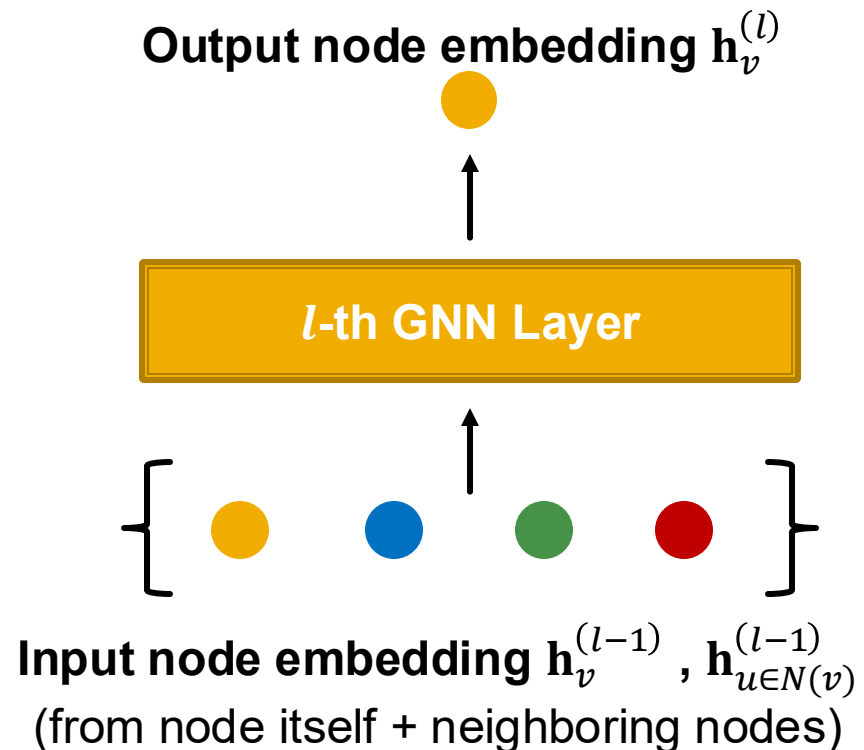
- **Idea of a GNN Layer:**

- Compress a set of vectors into a single vector

- **Two-step process:**

- (1) Message

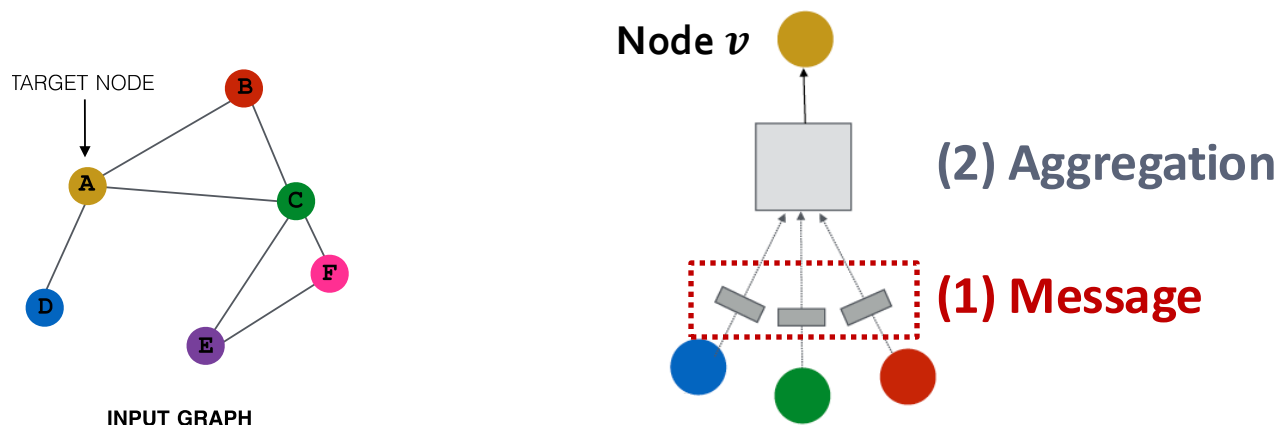
- (2) Aggregation



Message Computation

■ (1) Message computation

- **Message function:** $\mathbf{m}_u^{(l)} = \text{MSG}^{(l)} \left(\mathbf{h}_u^{(l-1)} \right)$
 - **Intuition:** Each node will create a message, which will be sent to other nodes later
 - **Example:** A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$
 - Multiply node features with weight matrix $\mathbf{W}^{(l)}$



Message Aggregation

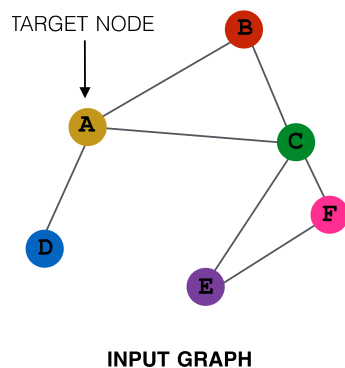
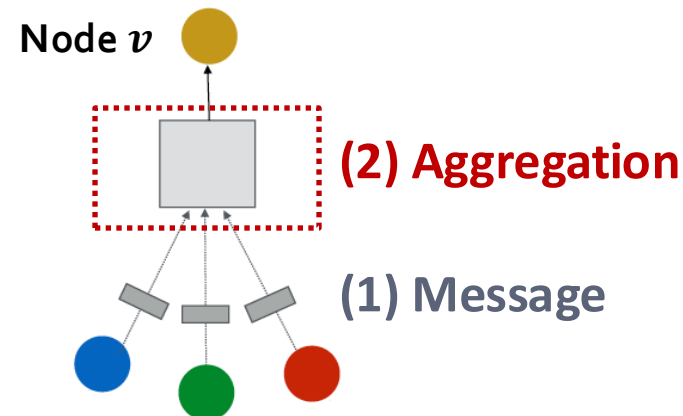
■ (2) Aggregation

- **Intuition:** Node v will aggregate the messages from its neighbors u :

$$\mathbf{h}_v^{(l)} = \text{AGG}^{(l)} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right)$$

- **Example:** Sum($\cdot$), Mean($\cdot$) or Max($\cdot$) aggregator

- $\mathbf{h}_v^{(l)} = \text{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$



Message Aggregation: Issue

- **Issue:** Information from node v itself **could get lost**

- Computation of $\mathbf{h}_v^{(l)}$ does not directly depend on $\mathbf{h}_v^{(l-1)}$

- **Solution:** Include $\mathbf{h}_v^{(l-1)}$ when computing $\mathbf{h}_v^{(l)}$

- **(1) Message:** compute message from node v itself

- Usually, a **different message computation** will be performed

$$\text{●} \text{●} \text{●} \quad \mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)} \qquad \text{●} \quad \mathbf{m}_v^{(l)} = \mathbf{B}^{(l)} \mathbf{h}_v^{(l-1)}$$

- **(2) Aggregation:** After aggregating from neighbors, we can **aggregate the message from node v itself**

- Via **concatenation** or **summation**

Then aggregate from node itself

$$\mathbf{h}_v^{(l)} = \text{CONCAT} \left(\underbrace{\text{AGG} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right)}_{\text{First aggregate from neighbors}} \underbrace{\left[\mathbf{m}_v^{(l)} \right]}_{\text{Then aggregate from node itself}} \right)$$

A Single GNN Layer

■ Putting things together:

- **(1) Message**: each node computes a message

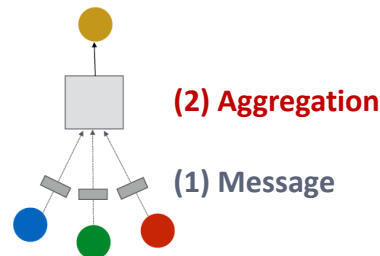
$$\mathbf{m}_u^{(l)} = \text{MSG}^{(l)} \left(\mathbf{h}_u^{(l-1)} \right), u \in \{N(v) \cup v\}$$

- **(2) Aggregation**: aggregate messages from neighbors

$$\mathbf{h}_v^{(l)} = \text{AGG}^{(l)} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\}, \mathbf{m}_v^{(l)} \right)$$

- **Nonlinearity (activation)**: Adds expressiveness

- Often written as $\sigma(\cdot)$. Examples: $\text{ReLU}(\cdot)$, $\text{Sigmoid}(\cdot)$, ...
- Can be added to **message** or **aggregation**



Activation (Non-linearity)

Apply activation to i -th dimension of embedding $\mathbf{x}$

- **Rectified linear unit (ReLU)**

$$\text{ReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0)$$

- Most commonly used

- **Sigmoid**

$$\sigma(\mathbf{x}_i) = \frac{1}{1 + e^{-\mathbf{x}_i}}$$

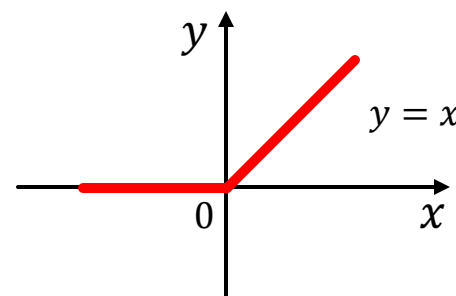
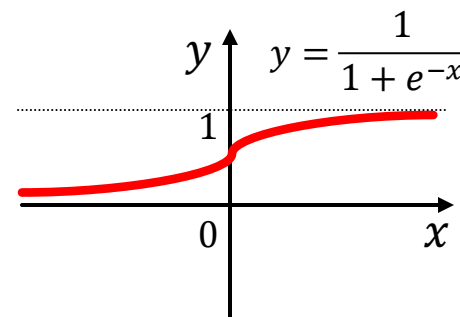
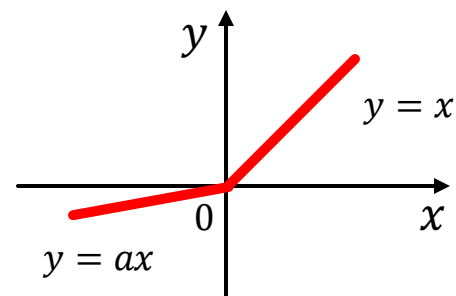
- Used only when you want to restrict the range of your embeddings

- **Parametric ReLU**

$$\text{PReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0) + a_i \min(\mathbf{x}_i, 0)$$

a_i is a trainable parameter

- Empirically performs better than ReLU



Classical GNN Layers: GCN (1)

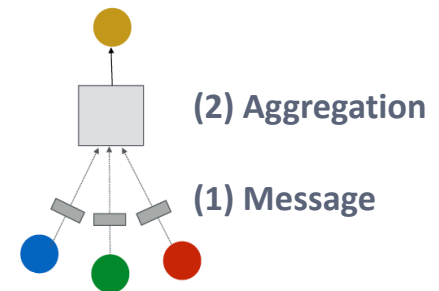
■ (1) Graph Convolutional Networks (GCN)

$$\mathbf{h}_v^{(l)} = \sigma \left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$

■ How to write this as Message + Aggregation?

$$\mathbf{h}_v^{(l)} = \sigma \left(\underbrace{\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}}_{\text{Aggregation}} \right)$$

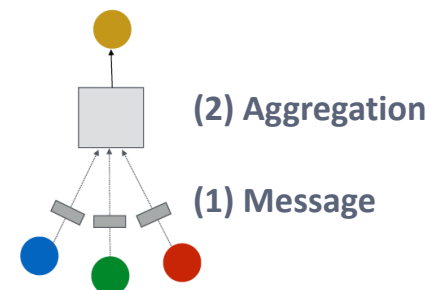
Message



Classical GNN Layers: GCN (2)

■ (1) Graph Convolutional Networks (GCN)

$$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$



■ Message:

- Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

Normalized by node degree
(In the GCN paper they use a slightly different normalization)

■ Aggregation:

- **Sum** over messages from neighbors, then apply activation

- $\mathbf{h}_v^{(l)} = \sigma \left(\text{Sum} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right) \right)$

In GCN the input graph is assumed to have self-edges that are included in the summation.

Classical GNN Layers: GraphSAGE

■ (2) GraphSAGE

$$\mathbf{h}_v^{(l)} = \sigma \left(\mathbf{W}^{(l)} \cdot \text{CONCAT} \left(\mathbf{h}_v^{(l-1)}, \text{AGG} \left(\left\{ \mathbf{h}_u^{(l-1)}, \forall u \in N(v) \right\} \right) \right) \right)$$

■ Two-stage aggregation

- **Stage 1:** Aggregate from node neighbors

$$\mathbf{h}_{N(v)}^{(l)} \leftarrow \text{AGG} \left(\left\{ \mathbf{h}_u^{(l-1)}, \forall u \in N(v) \right\} \right)$$

- **Stage 2:** Further aggregate over the node itself

$$\mathbf{h}_v^{(l)} \leftarrow \sigma \left(\mathbf{W}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)}) \right)$$

- **Message** is computed within the $\text{AGG}(\cdot)$

- **How to write this as Message + Aggregation?**

GraphSAGE Neighbor Aggregation

- **Mean:** Take a weighted average of neighbors

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}$$

Aggregation Message computation

- **Pool:** Transform neighbor vectors and apply symmetric vector function $\text{Mean}(\cdot)$ or $\text{Max}(\cdot)$

$$\text{AGG} = \text{Mean}(\{\text{MLP}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$$

Aggregation Message computation

- **LSTM:** Apply LSTM to reshuffled of neighbors

$$\text{AGG} = \text{LSTM}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))])$$

Aggregation

GraphSAGE: L2 Normalization

■ ℓ_2 Normalization:

- **Optional:** Apply ℓ_2 normalization to $\mathbf{h}_v^{(l)}$ at every layer

- $\mathbf{h}_v^{(l)} \leftarrow \frac{\mathbf{h}_v^{(l)}}{\|\mathbf{h}_v^{(l)}\|_2} \quad \forall v \in V$ where $\|u\|_2 = \sqrt{\sum_i u_i^2}$ (ℓ_2 -norm)

- Without ℓ_2 normalization, the embedding vectors have different scales (ℓ_2 -norm) for vectors
- In some cases (not always), normalization of embedding results in performance improvement
- After ℓ_2 normalization, all vectors will have the same ℓ_2 -norm

Classical GNN Layers: GAT (1)

■ (3) Graph Attention Networks

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Attention weights

■ In GCN / GraphSAGE

- $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u 's message to node v
- $\Rightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph (node degree)
- $\Rightarrow$ **All neighbors $u \in N(v)$ are equally important to node v**

Classical GNN Layers: GAT (1)

■ (3) Graph Attention Networks

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{m}_u^{(l)}\right)$$

Attention weights

■ In GCN / GraphSAGE

- $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u 's message to node v
- $\Rightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph (node degree)
- $\Rightarrow$ **All neighbors $u \in N(v)$ are equally important to node v**

Classical GNN Layers: GAT (2)

■ (3) Graph Attention Networks

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Attention weights

Not all node's neighbors are equally important

- **Attention** is inspired by cognitive attention.
- The **attention** α_{vu} focuses on the important parts of the input data and fades out the rest.
 - **Idea:** the NN should devote more computing power on that small but important part of the data.
 - Which part of the data is more important depends on the context and is learned through training.

Graph Attention Networks

Can we do better than simple neighborhood aggregation?

Can weighting factors α_{vu} be learned?

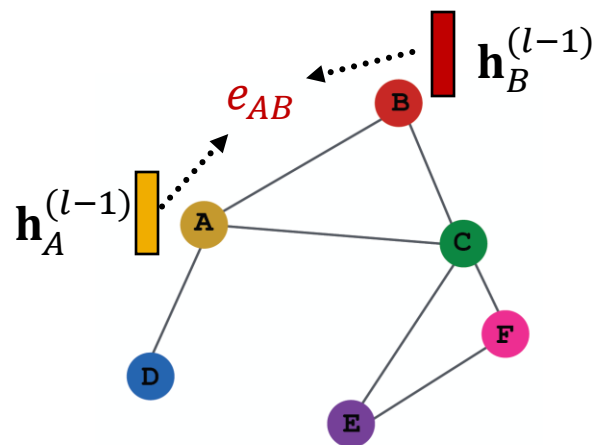
- **Goal:** Specify **arbitrary importance** to different neighbors of each node in the graph
- **Idea:** Compute embedding $\mathbf{h}_v^{(l)}$ of each node in the graph following an **attention strategy**:
 - Nodes attend over their neighborhoods' message
 - Implicitly specifying different weights to different nodes in a neighborhood

Attention Mechanism (1)

- Let α_{vu} be computed as a byproduct of an **attention mechanism a** :
 - (1) Let a compute **attention coefficients e_{vu}** across pairs of nodes u, v based on their messages:

$$e_{vu} = a(\mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)} \mathbf{h}_v^{(l-1)})$$

- e_{vu} indicates the importance of u 's message to node v



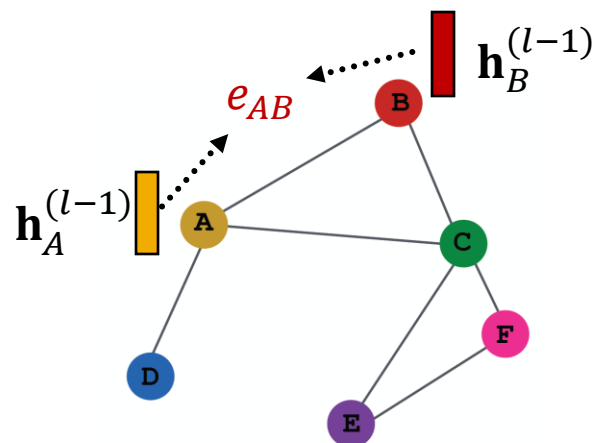
$$e_{AB} = a(\mathbf{W}^{(l)} \mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)} \mathbf{h}_B^{(l-1)})$$

Attention Mechanism (1)

- Let α_{vu} be computed as a byproduct of an **attention mechanism a** :
 - (1) Let a compute **attention coefficients e_{vu}** across pairs of nodes u, v based on their messages:

$$e_{vu} = a(\mathbf{m}_u^{(l)}, \mathbf{m}_v^{(l)})$$

- e_{vu} indicates the importance of u 's message to node v



$$e_{AB} = a(\mathbf{m}_A^{(l)}, \mathbf{m}_B^{(l)})$$

Attention Mechanism (2)

- **Normalize** e_{vu} into the **final attention weight** α_{vu}

- Use the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:

$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

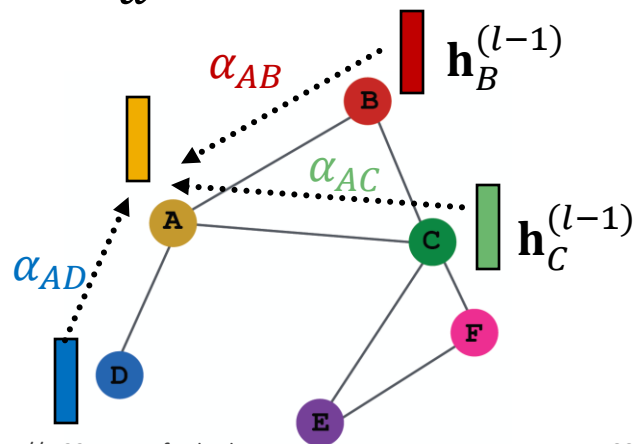
- **Weighted sum** based on the **final attention weight**

α_{vu} :

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Weighted sum using α_{AB} , α_{AC} , α_{AD} :

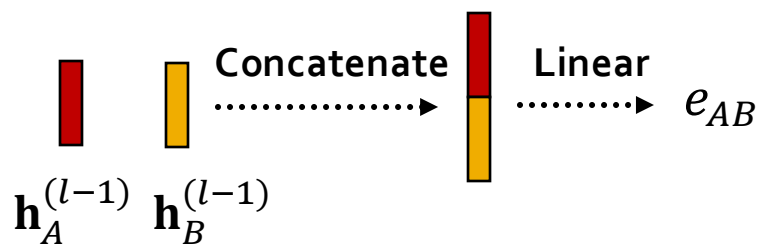
$$\mathbf{h}_A^{(l)} = \sigma(\alpha_{AB} \mathbf{W}^{(l)} \mathbf{h}_B^{(l-1)} + \alpha_{AC} \mathbf{W}^{(l)} \mathbf{h}_C^{(l-1)} + \alpha_{AD} \mathbf{W}^{(l)} \mathbf{h}_D^{(l-1)})$$



Attention Mechanism (3)

■ What is the form of attention mechanism a ?

- The approach is agnostic to the choice of a
 - E.g., use a simple single-layer neural network
 - a have trainable parameters (weights in the Linear layer)



$$\begin{aligned} e_{AB} &= a\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right) \\ &= \text{Linear}\left(\text{Concat}\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)\right) \end{aligned}$$

- Parameters of a are trained jointly:
 - Learn the parameters together with weight matrices (i.e., other parameter of the neural net $\mathbf{W}^{(l)}$) in an end-to-end fashion

Attention Mechanism (4)

- **Multi-head attention:** Stabilizes the learning process of attention mechanism

- Create **multiple attention scores** (each replica with a different set of parameters):

$$\mathbf{h}_v^{(l)}[1] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^1 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

$$\mathbf{h}_v^{(l)}[2] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^2 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

$$\mathbf{h}_v^{(l)}[3] = \sigma(\sum_{u \in N(v)} \alpha_{vu}^3 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

- **Outputs are aggregated:**

- By concatenation or summation

- $\mathbf{h}_v^{(l)} = \text{AGG}(\mathbf{h}_v^{(l)}[1], \mathbf{h}_v^{(l)}[2], \mathbf{h}_v^{(l)}[3])$

Benefits of Attention Mechanism

- **Key benefit:** Allows for (implicitly) specifying **different importance values (α_{vu}) to different neighbors**
- **Computationally efficient:**
 - Computation of attentional coefficients can be parallelized across all edges of the graph
 - Aggregation may be parallelized across all nodes
- **Storage efficient:**
 - Sparse matrix operations do not require more than $O(V + E)$ entries to be stored
 - **Fixed** number of parameters, irrespective of graph size
- **Localized:**
 - Only **attends over local network neighborhoods**
- **Inductive capability:**
 - It is a shared *edge-wise* mechanism
 - It does not depend on the global graph structure

Stanford CS224W: GNN Layers in Practice

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
<http://cs224w.stanford.edu>

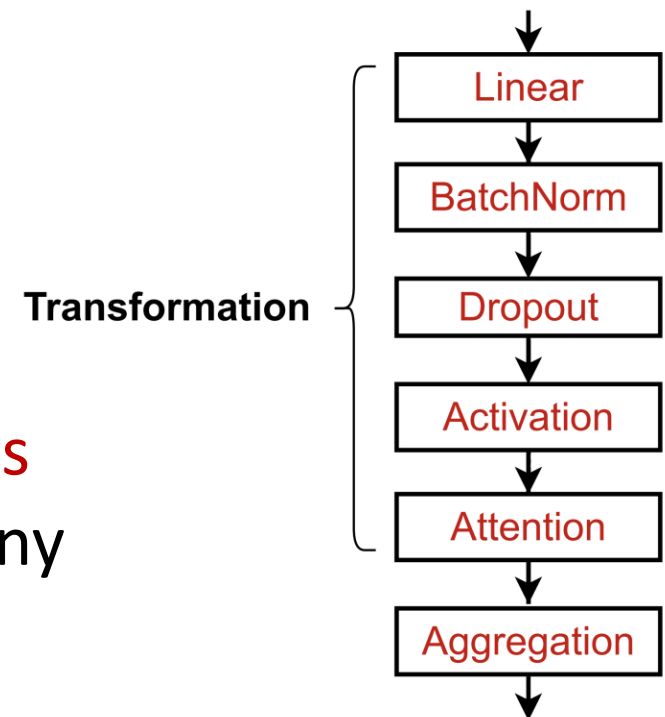


GNN Layer in Practice

- In practice, these classic GNN layers are a great starting point

- We can often get better performance by considering a general GNN layer design
- Concretely, we can include modern deep learning modules that proved to be useful in many domains

A suggested GNN Layer



GNN Layer in Practice

- Many modern deep learning modules can be incorporated into a GNN layer

- **Batch Normalization:**

- Stabilize neural network training

- **Dropout:**

- Prevent overfitting

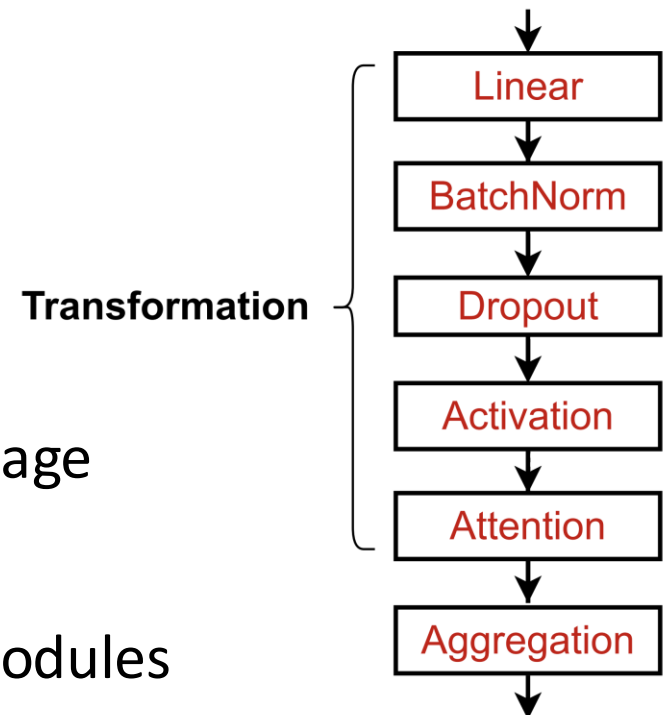
- **Attention/Gating:**

- Control the importance of a message

- **More:**

- Any other useful deep learning modules

A suggested GNN Layer



Batch Normalization

- **Goal:** Stabilize neural networks training
- **Idea:** Given a batch of inputs (node embeddings)
 - Re-center the node embeddings into zero mean
 - Re-scale the variance into unit variance

Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$
 N node embeddings

Trainable Parameters:
 $\gamma, \beta \in \mathbb{R}^D$

Output: $\mathbf{Y} \in \mathbb{R}^{N \times D}$
Normalized node embeddings

Step 1:
Compute the mean and variance over N embeddings

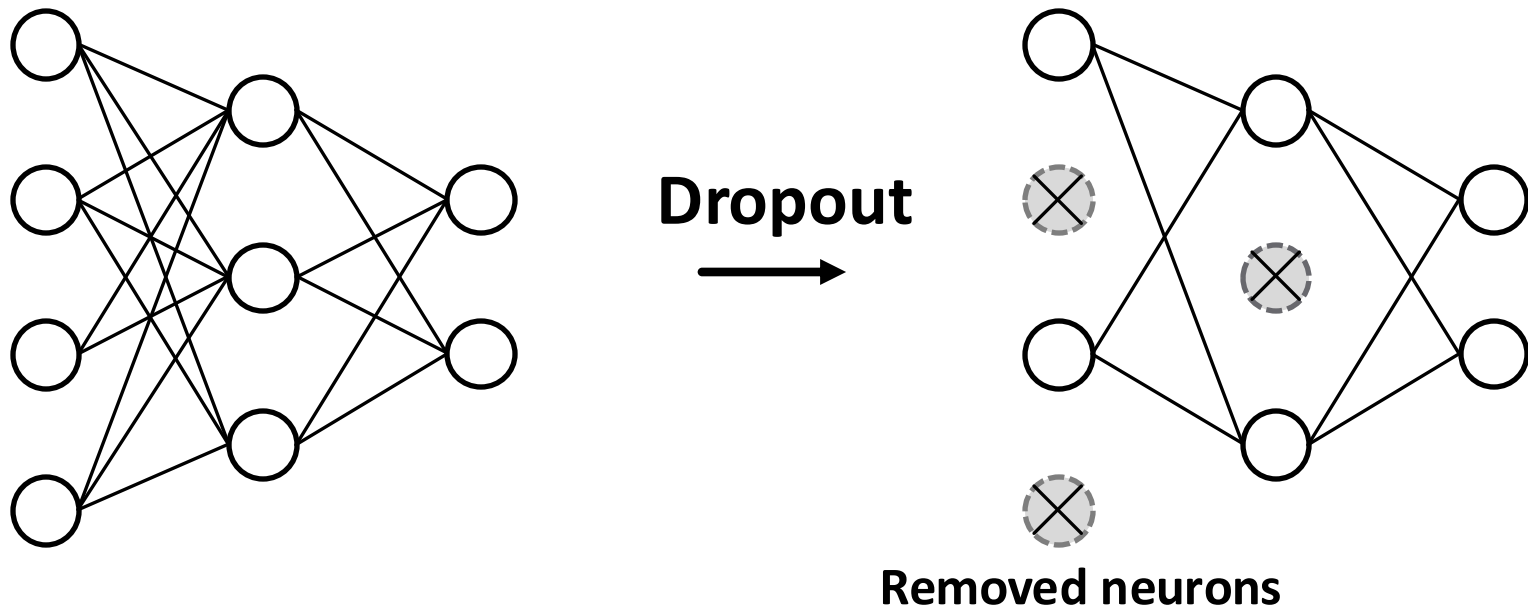
$$\mu_j = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_{i,j}$$
$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_{i,j} - \mu_j)^2$$

Step 2:
Normalize the feature using computed mean and variance

$$\hat{\mathbf{x}}_{i,j} = \frac{\mathbf{x}_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$
$$\mathbf{y}_{i,j} = \gamma_j \hat{\mathbf{x}}_{i,j} + \beta_j$$

Dropout

- **Goal:** Regularize a neural net to prevent overfitting.
- **Idea:**
 - **During training:** with some probability p , randomly set neurons to zero (turn off)
 - **During testing:** Use all the neurons for computation

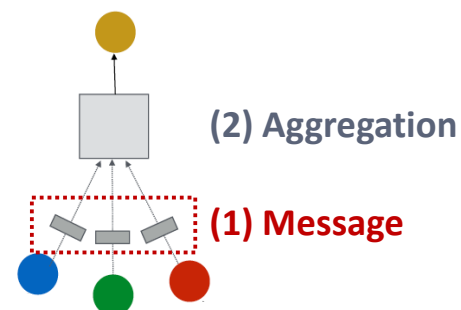
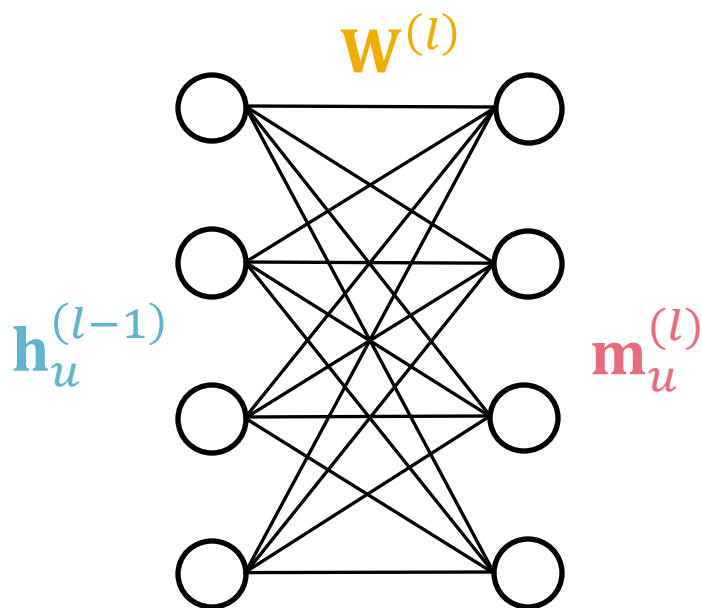


Dropout for GNNs

- In GNN, Dropout is applied to **the linear layer in the message function**

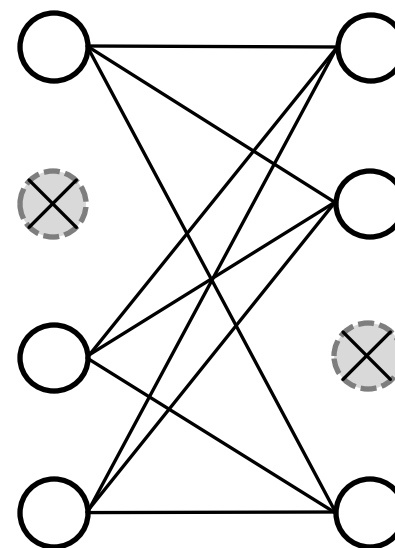
- A simple message function with linear

layer: $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$



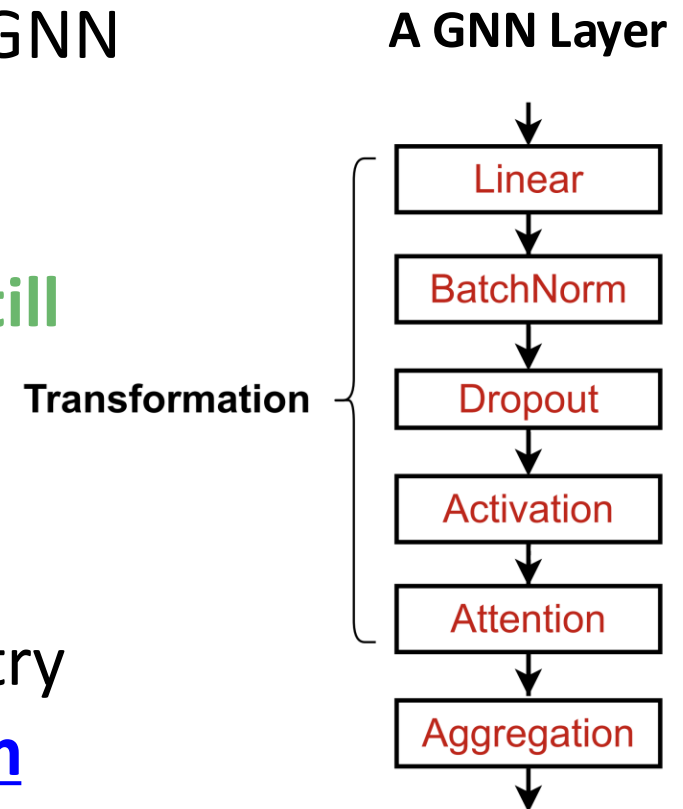
Visualization of a linear layer

Dropout
→



GNN Layer in Practice

- **Summary:** Modern deep learning modules can be included into a GNN layer for better performance
- **Designing novel GNN layers is still an active research frontier!**
- **Suggested resources:** You can explore diverse GNN designs or try out your own ideas in [GraphGym](#)



Stanford CS224W: Stacking Layers of a GNN

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

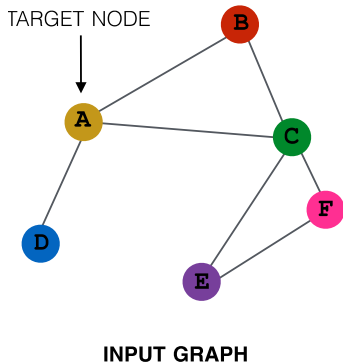
<http://cs224w.stanford.edu>



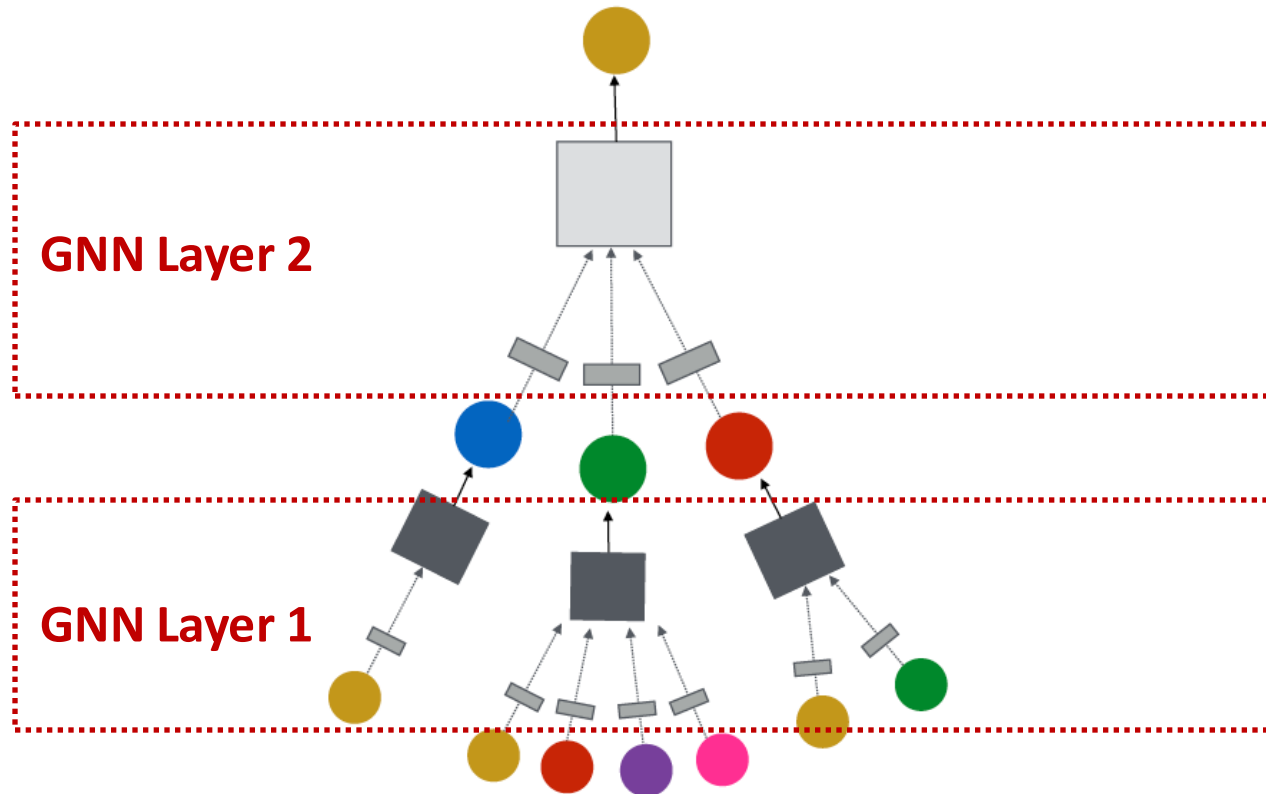
Stacking GNN Layers

How to connect GNN layers into a GNN?

- Stack layers sequentially
- Ways of adding skip connections

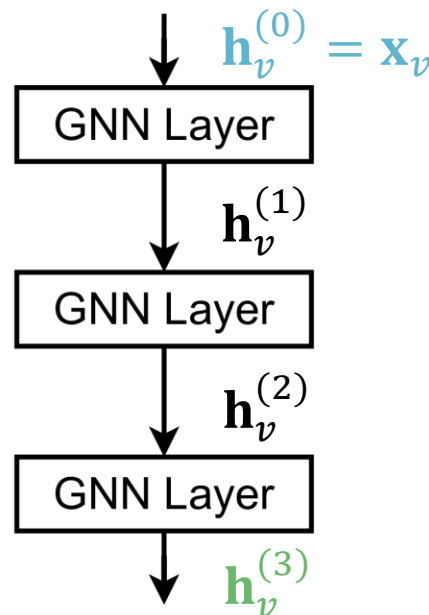


(3) Layer connectivity



Stacking GNN Layers

- **How to construct a Graph Neural Network?**
 - **The standard way:** Stack GNN layers sequentially
 - **Input:** Initial raw node feature $\mathbf{x}_v$
 - **Output:** Node embeddings $\mathbf{h}_v^{(L)}$ after L GNN layers



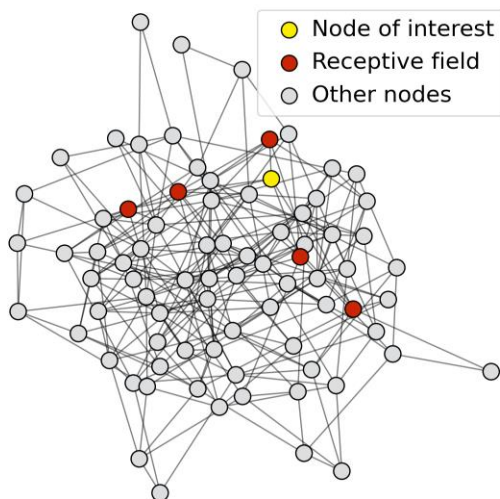
The Over-smoothing Problem

- The issue of stacking many GNN layers
 - GNN suffers from **the over-smoothing problem**
- **The over-smoothing problem: all the node embeddings converge to the same value**
 - This is bad because we **want to use node embeddings to differentiate nodes**
- **Why does the over-smoothing problem happen?**

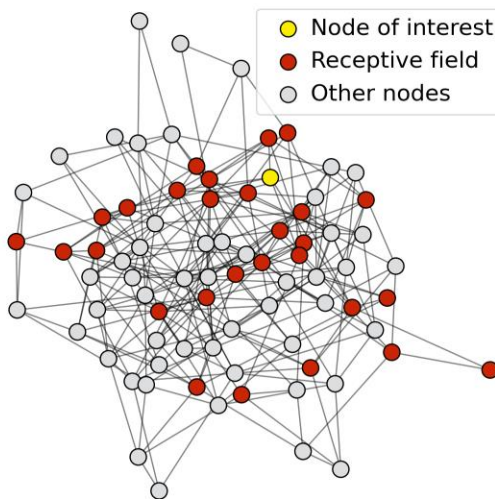
Receptive Field of a GNN

- **Receptive field:** the set of nodes that determine the embedding of a node of interest
 - In a K -layer GNN, each node has a receptive field of K -hop neighborhood

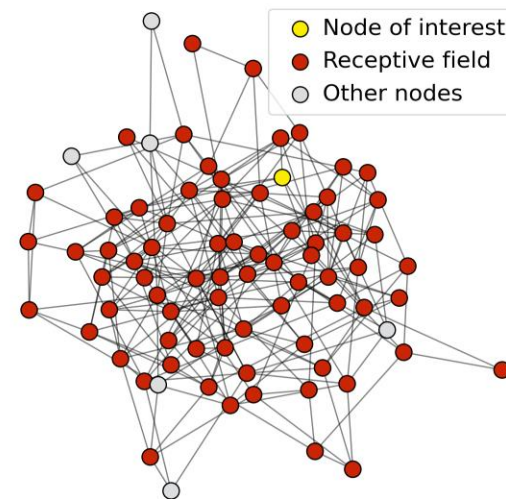
Receptive field for
1-layer GNN



Receptive field for
2-layer GNN



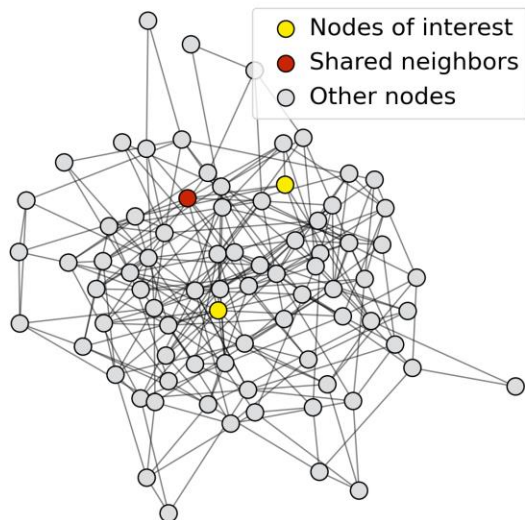
Receptive field for
3-layer GNN



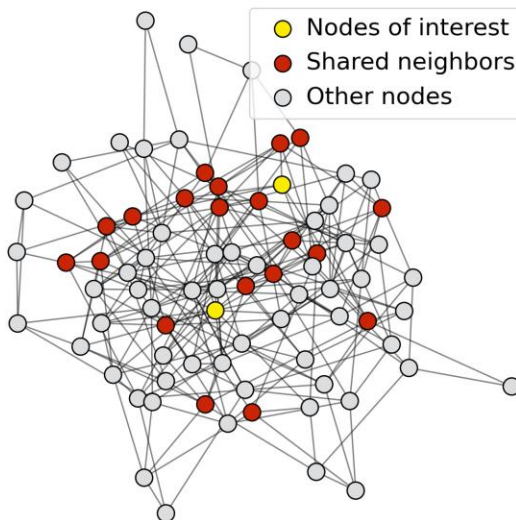
Receptive Field of a GNN

- **Receptive field overlap for two nodes**
 - **The shared neighbors quickly grow** when we increase the number of hops (num of GNN layers)

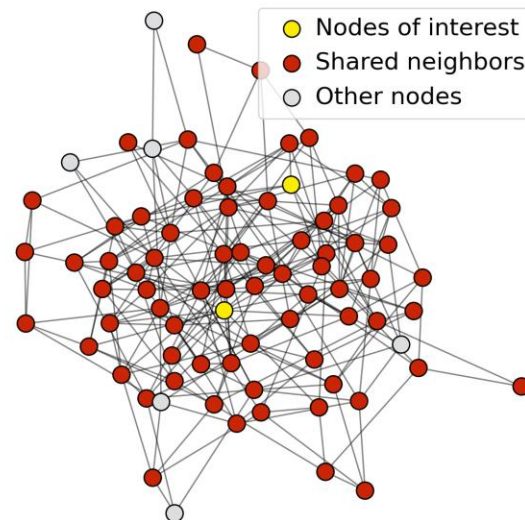
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes



3-hop neighbor overlap
Almost all the nodes!



Receptive Field & Over-smoothing

- We can explain over-smoothing via the notion of the receptive field
 - We know the embedding of a node is determined by its receptive field
 - If two nodes have highly-overlapped receptive fields, then their embeddings are highly similar
 - Stack many GNN layers → nodes will have highly-overlapped receptive fields → node embeddings will be highly similar → suffer from the over-smoothing problem
- Next: how do we overcome over-smoothing problem?

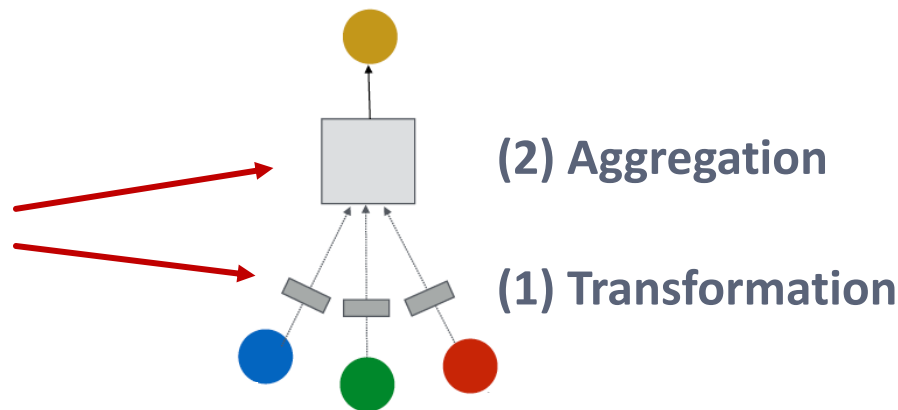
Design GNN Layer Connectivity

- **What do we learn from the over-smoothing problem?**
- **Lesson 1: Be cautious when adding GNN layers**
 - Unlike neural networks in other domains (CNN for image classification), **adding more GNN layers do not always help**
 - **Step 1: Analyze the necessary receptive field** to solve your problem. E.g., by computing the diameter of the graph
 - **Step 2:** Set number of GNN layers L to be a bit more than the receptive field we like. **Do not set L to be unnecessarily large!**
- **Question:** How to enhance the expressive power of a GNN, **if the number of GNN layers is small?**

Expressive Power for Shallow GNNs

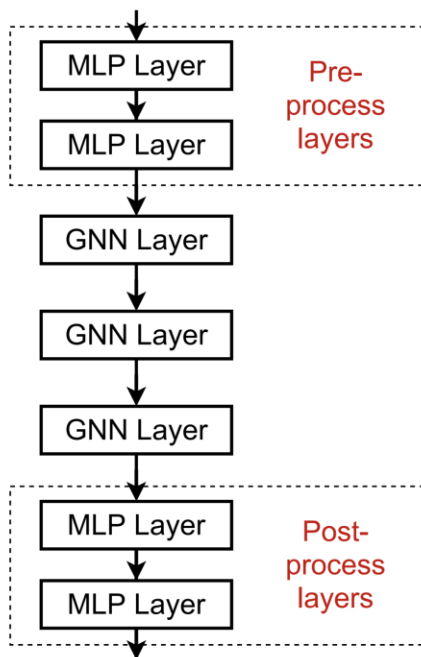
- **How to make a shallow GNN more expressive?**
- **Solution 1:** Increase the expressive power **within** each GNN layer
 - In our previous examples, each transformation or aggregation function only includes one linear layer
 - We can **make aggregation / transformation become a deep neural network!**

If needed, each box could include a **3-layer MLP**



Expressive Power for Shallow GNNs

- **How to make a shallow GNN more expressive?**
- **Solution 2:** Add layers that do not pass messages
 - A GNN does not necessarily only contain GNN layers
 - E.g., we can add **MLP layers** (applied to each node) before and after GNN layers, as **pre-process layers** and **post-process layers**



Pre-processing layers: Important when encoding node features is necessary.

E.g., when nodes represent images/text

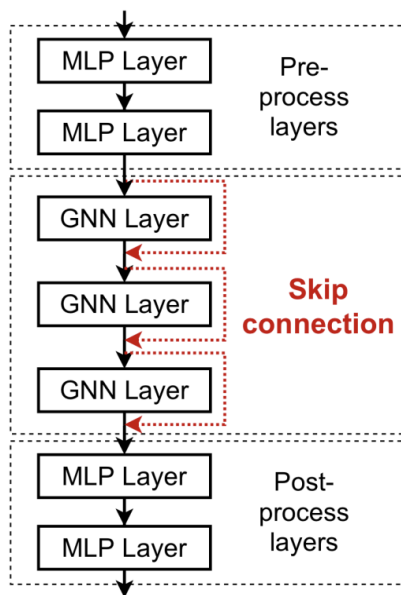
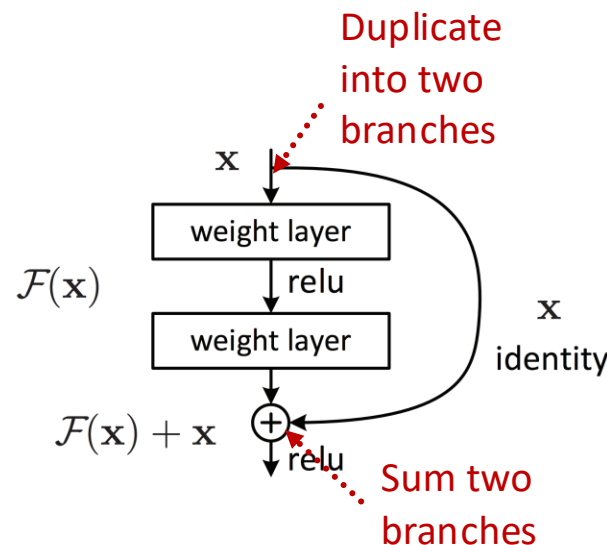
Post-processing layers: Important when reasoning / transformation over node embeddings are needed

E.g., graph classification, knowledge graphs

In practice, adding these layers works great!

Design GNN Layer Connectivity

- What if my problem still requires many GNN layers?
- Lesson 2: Add skip connections in GNNs
 - Observation from over-smoothing: Node embeddings in earlier GNN layers can sometimes better differentiate nodes
 - Solution: We can increase the impact of earlier layers on the final node embeddings, **by adding shortcuts in GNN**



Idea of skip connections:

Before adding shortcuts:

$$F(x)$$

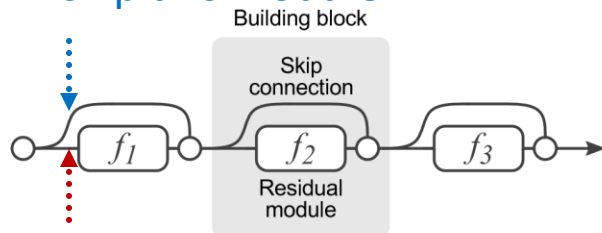
After adding shortcuts:

$$F(x) + x$$

Idea of Skip Connections

- **Why do skip connections work?**
 - **Intuition:** Skip connections create **a mixture of models**
 - N skip connections $\rightarrow 2^N$ possible paths
 - Each path could have up to N modules
 - We automatically get **a mixture of shallow GNNs and deep GNNs**

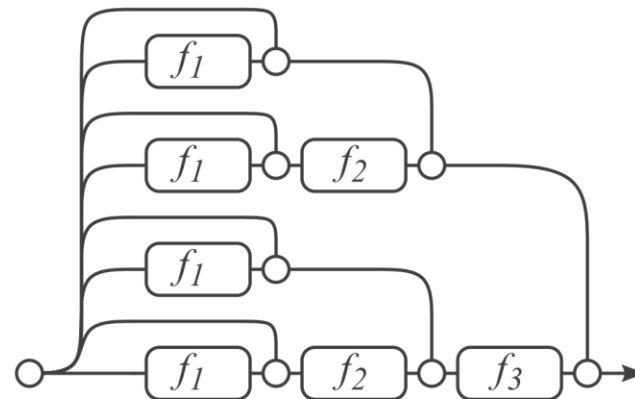
Path 2: skip this module



Path 1: include this module

(a) Conventional 3-block residual network

=



(b) Unraveled view of (a)

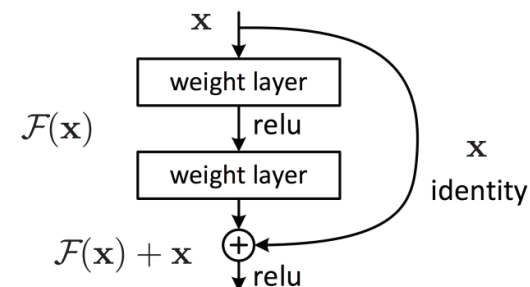
Veit et al. Residual Networks Behave Like Ensembles of Relatively Shallow Networks, ArXiv 2016

Example: GCN with Skip Connections

- A standard GCN layer

- $$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$

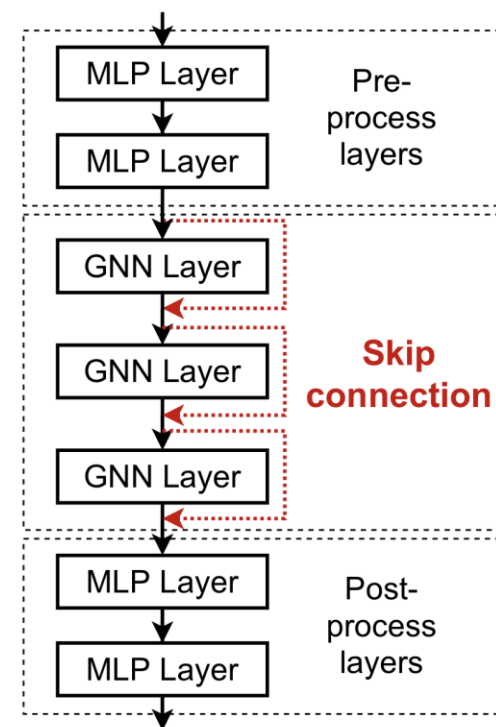
This is our $F(\mathbf{x})$



- A GCN layer with skip connection

- $$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} + \mathbf{h}_v^{(l-1)} \right)$$

$F(\mathbf{x}) \quad + \quad \mathbf{x}$



Other Options of Skip Connections

- **Other options:** Directly skip to the last layer
 - The final layer directly **aggregates from the all the node embeddings** in the previous layers

