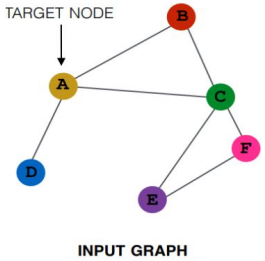
GNN Architectures

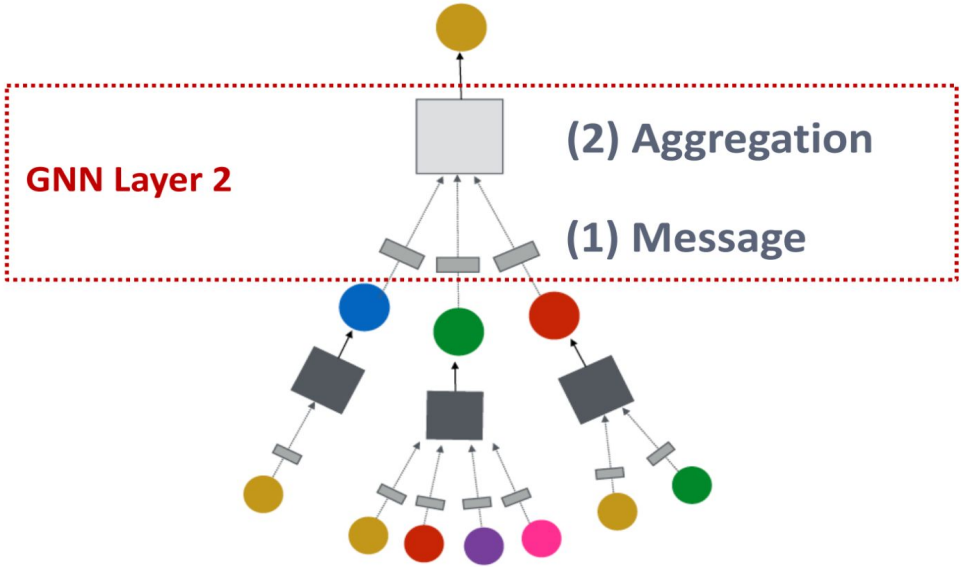
https://web.stanford.edu/class/cs224w/slides/04-GNN2.pdf

A GNN Layer



GNN Layer = Message + Aggregation

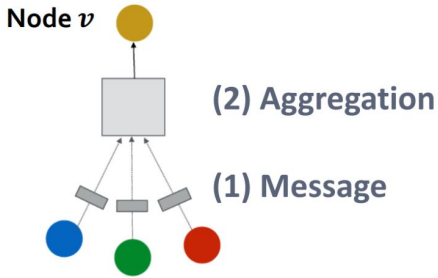
- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**

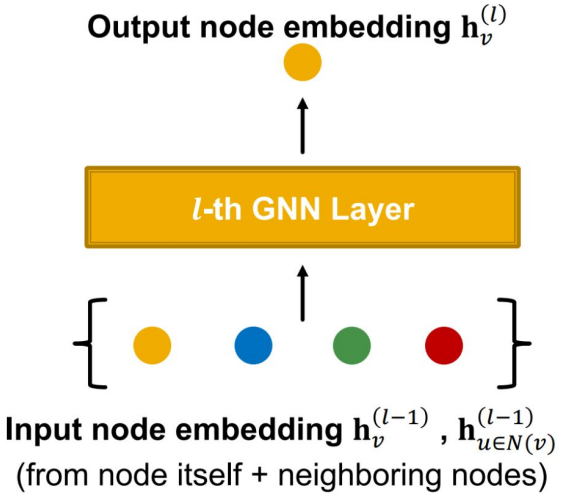


Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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**

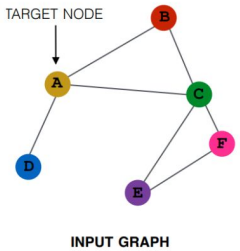


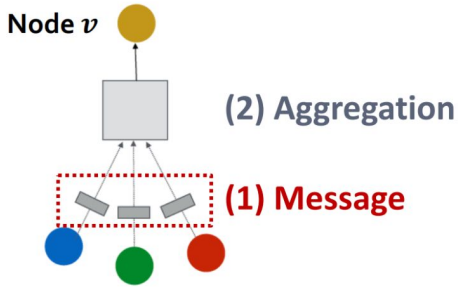


Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Message Computation

- **(1) Message computation**
  - **Message function:** $\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$
    - **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 $\mathbf{W}^{(l)}$





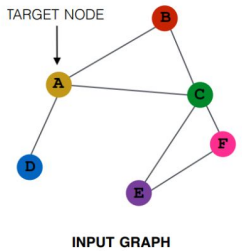
Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

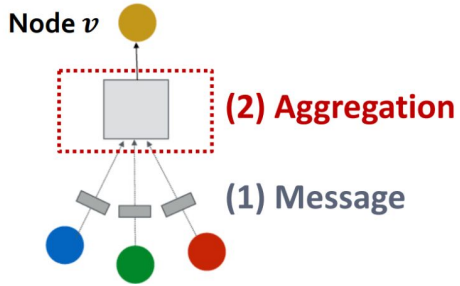
Message Aggregation

- **(2) Aggregation**
  - **Intuition:** Node v will aggregate the messages from its neighbors u:

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

  - **Example:** $\mathrm{Sum}(\cdot)$, $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$ aggregator
    - $\mathbf{h}_v^{(l)} = \mathrm{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$





Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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

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

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

Then aggregate from node itself

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

First aggregate from neighbors

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

A Single GNN Layer

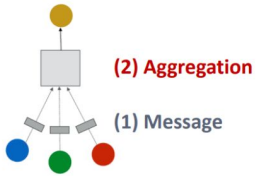
- **Putting things together:**
  - **(1) Message**: each node computes a message

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

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

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

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



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

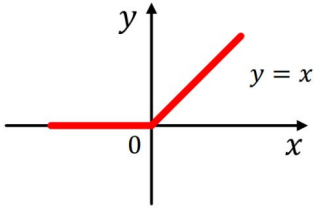
Activation (Non-linearity)

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

- **Rectified linear unit (ReLU)**

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

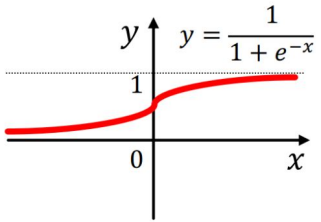
  - Most commonly used



- **Sigmoid**

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

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

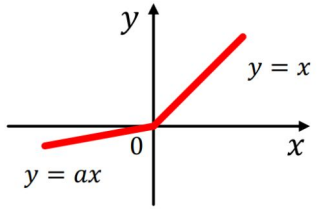


- **Parametric ReLU**

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

  a_i is a trainable parameter

  - Empirically performs better than ReLU



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Classical GNN Layers: GCN (1)

- **(1) Graph Convolutional Networks (GCN)**

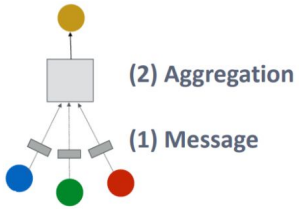
$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

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

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

Message

Aggregation

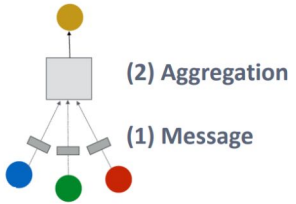


Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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.

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Classical GNN Layers: GraphSAGE

- **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?**

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

GraphSAGE: L2 Normalization

- **ℓ_2 Normalization:**
  - **Optional:** Apply ℓ_2 normalization to $\mathbf{h}_v^{(l)}$ at every layer
  - $\mathbf{h}_v^{(l)} \leftarrow \frac{\mathbf{h}_v^{(l)}}{\left\|\mathbf{h}_v^{(l)}\right\|_2} \; \forall v \in V$ 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

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Classical GNN Layers: GAT (1)

- **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
  - $\Longrightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph (node degree)
  - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Classical GNN Layers: GAT (1)

- **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
  - $\Longrightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph (node degree)
  - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Classical GNN Layers: GAT (2)

- **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.

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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 $\boldsymbol{h}_v^{(l)}$ 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

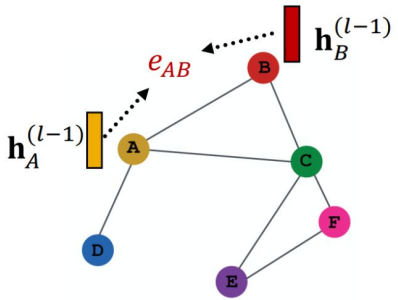
Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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:

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

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



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

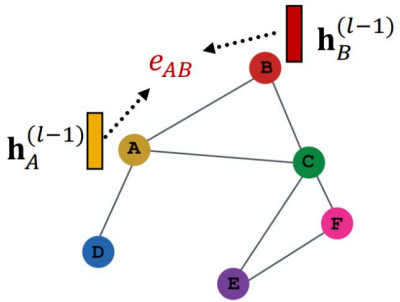
Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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:

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

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



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

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Attention Mechanism (2)

- **Normalize** e_{vu} into the **final attention weight** α_{vu}
  - Use the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:

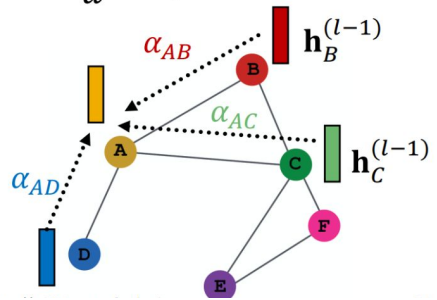
$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

- **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}:

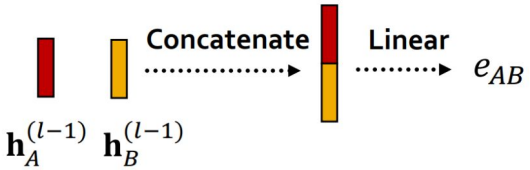
$$\mathbf{h}_A^{(l)} = \sigma\left(\alpha_{AB}\mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)} + \alpha_{AC}\mathbf{W}^{(l)}\mathbf{h}_C^{(l-1)} + \alpha_{AD}\mathbf{W}^{(l)}\mathbf{h}_D^{(l-1)}\right)$$



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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)



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

  - 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

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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):

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

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

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

  - **Outputs are aggregated:**
    - By concatenation or summation
    - $\mathbf{h}_v^{(l)} = \mathrm{AGG}(\mathbf{h}_v^{(l)}[1], \mathbf{h}_v^{(l)}[2], \mathbf{h}_v^{(l)}[3])$

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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

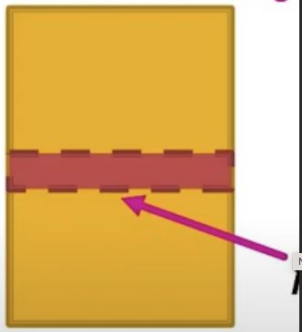
Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Matrix Notation for GNNs

GCN with Matrix Notation

- **Many aggregations can be performed efficiently by (sparse) matrix operations**
- Let $H^{(l)} = [h_1^{(l)} \dots h_{|V|}^{(l)}]^T$
- Then: $\sum_{u \in N_v} h_u^{(l)} = A_{v,:} H^{(l)}$
- Let D be diagonal matrix where $D_{v,v} = \text{Deg}(v) = |N(v)|$
  - The inverse of D: D^{-1} is also diagonal: $D_{v,v}^{-1} = 1/|N(v)|$
- **Therefore,**

Matrix of hidden embeddings



$$\sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|} \Rightarrow H^{(l+1)} = D^{-1} A H^{(l)}$$

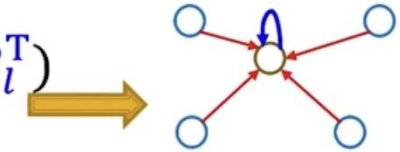
Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

GCN with Matrix Notation

- Re-writing update function in matrix form:

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}})$$

where $\tilde{A} = D^{-1}A$



$H^{(l)} = [h_1^{(l)} \dots h_{|V|}^{(l)}]^T$

  - Red: neighborhood aggregation
  - Blue: self transformation

- In practice, this implies that efficient sparse matrix multiplication can be used ($\tilde{A}$ is sparse)

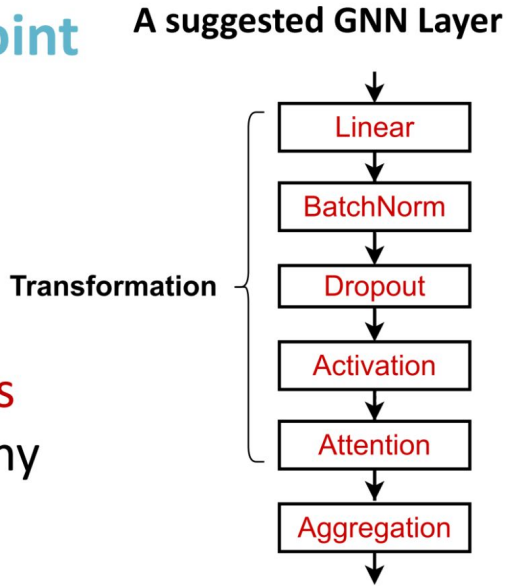
Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

GNN Layers in Practice

Caltech EAS AI BOOTCAMP

GNN Layer in Practice

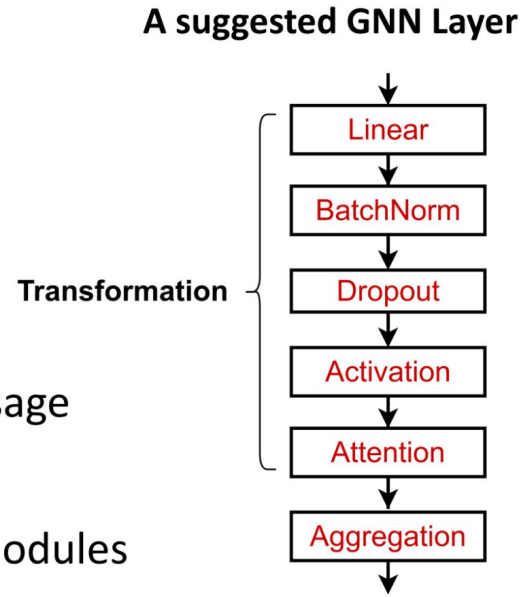
- **In practice, these classic GNN layers are a great starting point**
  - We can often get better performance by considering a general GNN layer design
  - Concretely, we can include modern deep learning modules that proved to be useful in many domains



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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

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

Step 1:
Compute the mean and variance over N embeddings

$$\boldsymbol{\mu}_j = \frac{1}{N}\sum_{i=1}^{N} \mathbf{X}_{i,j}$$

$$\boldsymbol{\sigma}_j^2 = \frac{1}{N}\sum_{i=1}^{N} \left(\mathbf{X}_{i,j} - \boldsymbol{\mu}_j\right)^2$$

Step 2:
Normalize the feature using computed mean and variance

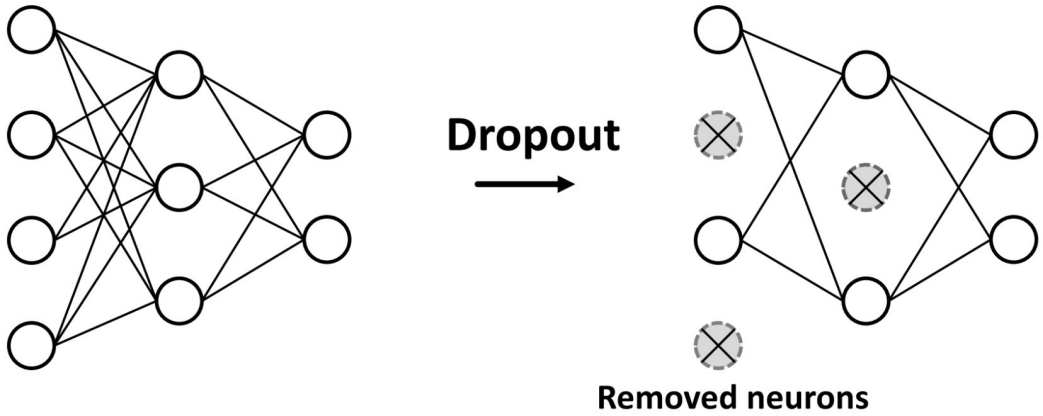
$$\widehat{\mathbf{X}}_{i,j} = \frac{\mathbf{X}_{i,j} - \boldsymbol{\mu}_j}{\sqrt{\boldsymbol{\sigma}_j^2 + \epsilon}}$$

$$\mathbf{Y}_{i,j} = \boldsymbol{\gamma}_j \widehat{\mathbf{X}}_{i,j} + \boldsymbol{\beta}_j$$

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Dropout

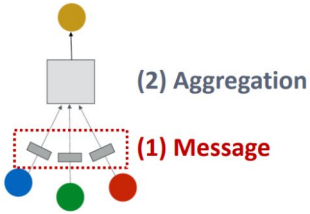
- **Goal**: Regularize a neural net to prevent overfitting.
- **Idea**:
  - **During training**: with some probability p, randomly set neurons to zero (turn off)
  - **During testing:** Use all the neurons for computation

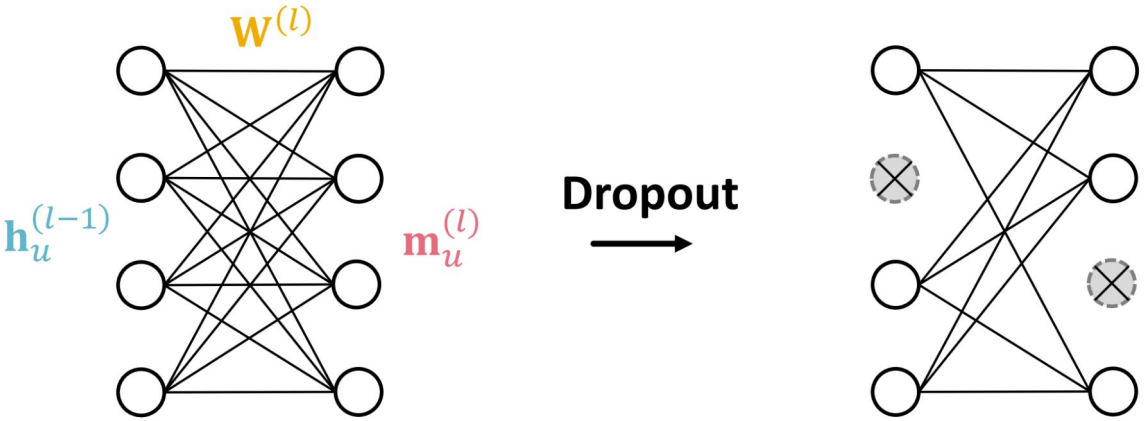


Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Dropout for GNNs

- In GNN, Dropout is applied to **the linear layer in the message function**
  - **A simple message function with linear layer:** $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$



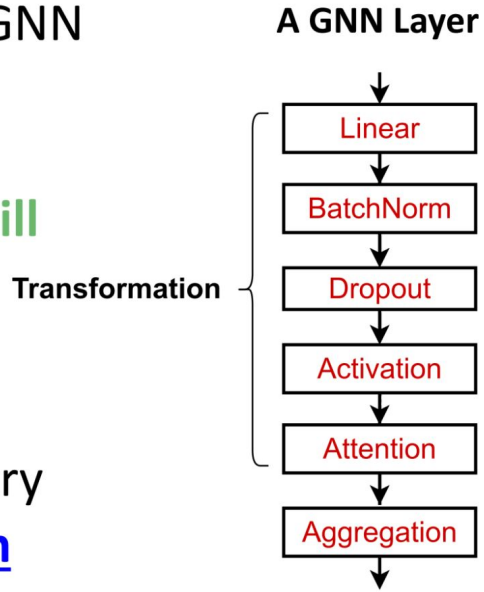


Visualization of a linear layer

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

GNN Layer in Practice

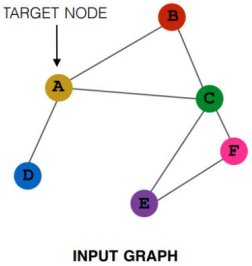
- **Summary:** Modern deep learning modules can be included into a GNN layer for better performance
- **Designing novel GNN layers is still an active research frontier!**
- **Suggested resources:** You can explore diverse GNN designs or try out your own ideas in **GraphGym**



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Stacking Layers of a GNN

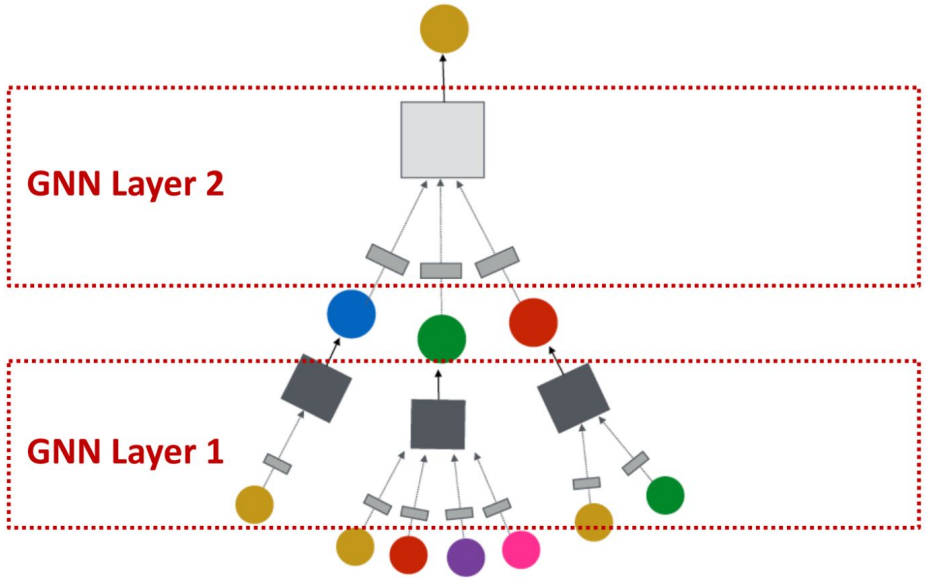
Stacking GNN Layers



How to connect GNN layers into a GNN?

- **Stack layers sequentially**
- **Ways of adding skip connections**

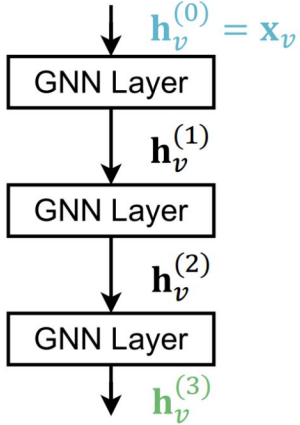
(3) Layer connectivity



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

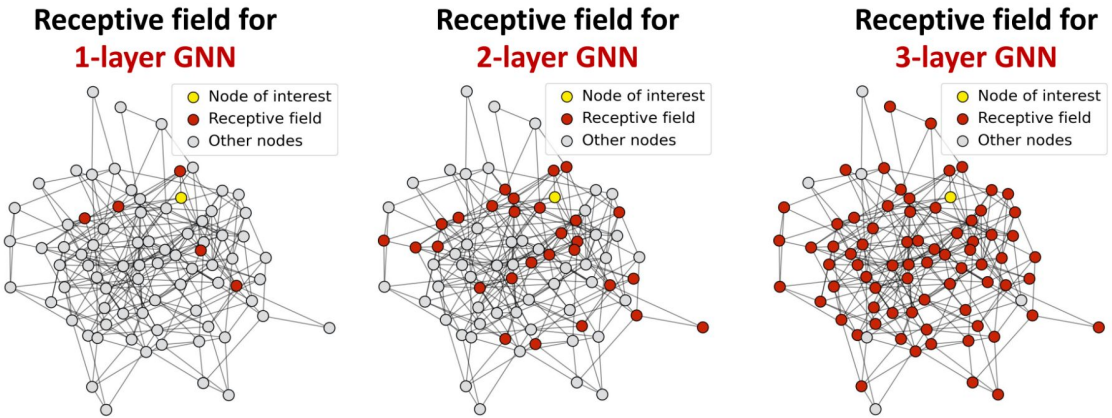
The Over-smoothing Problem

- **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?**

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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**

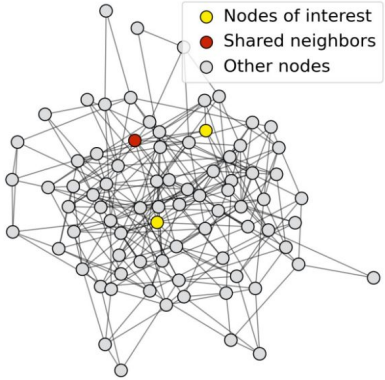


Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

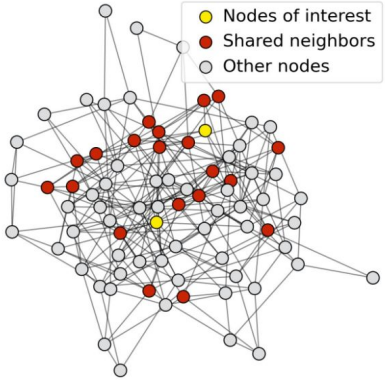
Receptive Field of a GNN

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

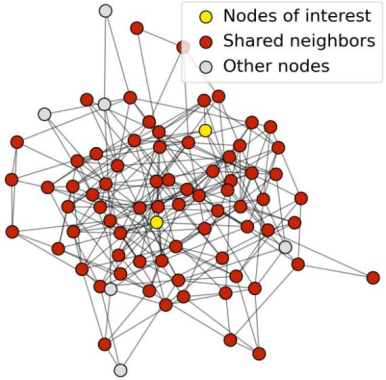
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes



3-hop neighbor overlap
Almost all the nodes!



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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?

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Design GNN Layer Connectivity

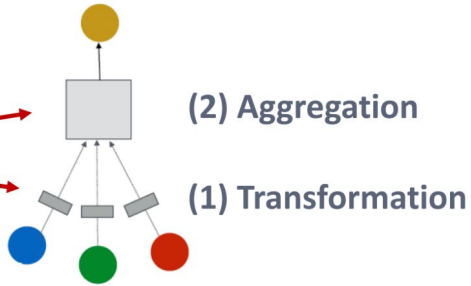
- **What do we learn from the over-smoothing problem?**
- **Lesson 1: Be cautious when adding GNN layers**
  - Unlike neural networks in other domains (CNN for image classification), **adding more GNN layers do not always help**
  - **Step 1: Analyze the necessary receptive field** to solve your problem. 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?

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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 include one linear layer
  - We can **make aggregation / transformation become a deep neural network**!

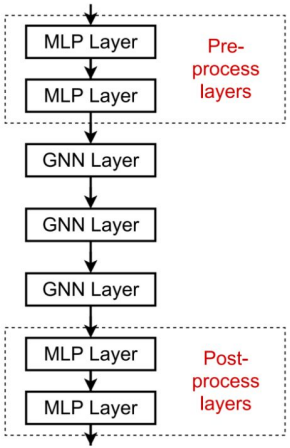
If needed, each box could include a 3-layer MLP



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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

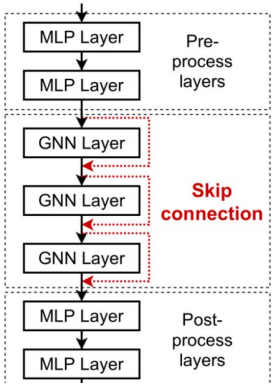
Post-processing layers: Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

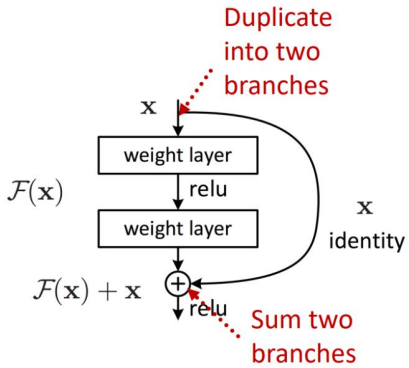
In practice, adding these layers works great!

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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:

$$\boldsymbol{F}(\mathbf{x})$$

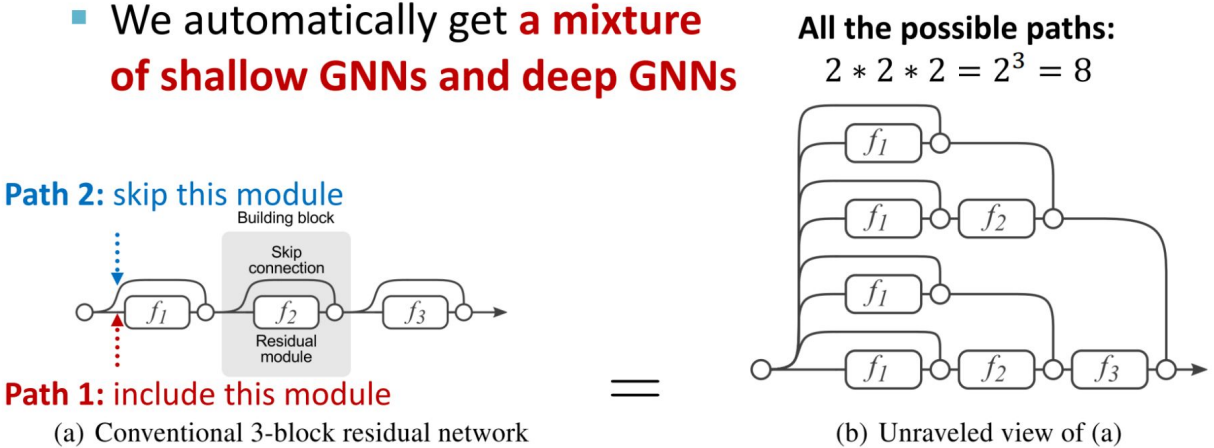
After adding shortcuts:

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

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Idea of Skip Connections

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



(a) Conventional 3-block residual network

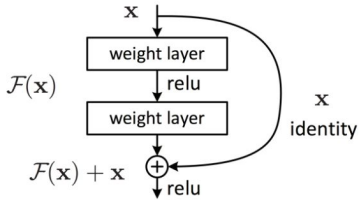
(b) Unraveled view of (a)

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Example: GCN with Skip Connections

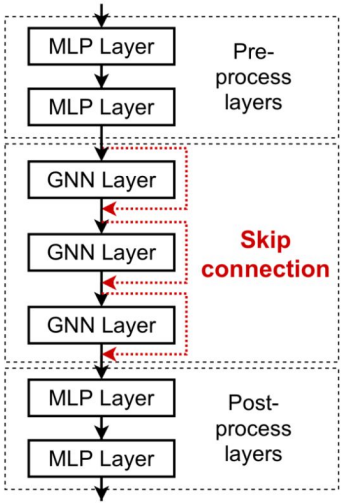
- **A standard GCN layer**
- $\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$

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



- **A GCN layer with skip connection**
- $\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} + \mathbf{h}_v^{(l-1)}\right)$

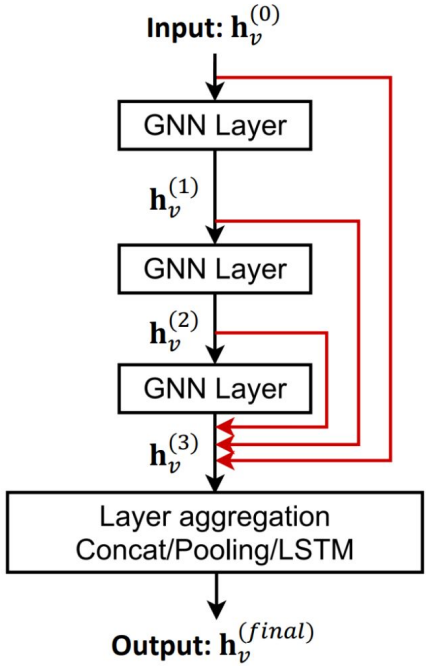
$F(\mathbf{x})$ + $\mathbf{x}$



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

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

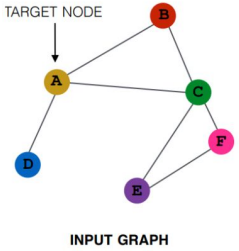


Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Graph Manipulation in GNNs

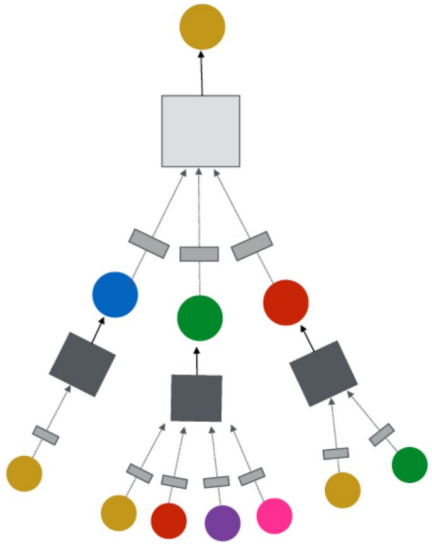
Caltech EAS AI BOOTCAMP

General GNN Framework



Idea: Raw input graph ≠ computational graph

- **Graph feature augmentation**
- **Graph structure manipulation**



(4) Graph manipulation

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Why Manipulate Graphs

Our assumption so far has been

- **Raw input graph = computational graph**

Reasons for breaking this assumption

- **Feature level:**
  - The input graph **lacks features** → feature augmentation
- **Structure level:**
  - The graph is **too sparse** → inefficient message passing
  - The graph is **too dense** → message passing is too costly
  - The graph is **too large** → cannot fit the computational graph into a GPU
- It's just **unlikely that the input graph happens to be the optimal computation graph** for embeddings

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Graph Manipulation Approaches

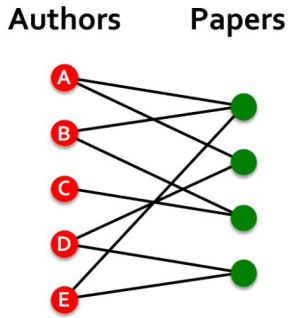
- **Graph Feature manipulation**
  - The input graph **lacks features** → **feature augmentation (will cover in Lecture 7)**
- **Graph Structure manipulation**
  - The graph is **too sparse** → **Add virtual nodes / edges**
  - The graph is **too dense** → **Sample neighbors when doing message passing**
  - The graph is **too large** → **Sample subgraphs to compute embeddings**
    - Will cover later in lecture: Scaling up GNNs

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Add Virtual Nodes / Edges

- **Motivation:** Augment sparse graphs
- **(1) Add virtual edges**
  - **Common approach:** Connect 2-hop neighbors via virtual edges
  - **Intuition:** Instead of using adj. matrix A for GNN computation, use $A + A^2$

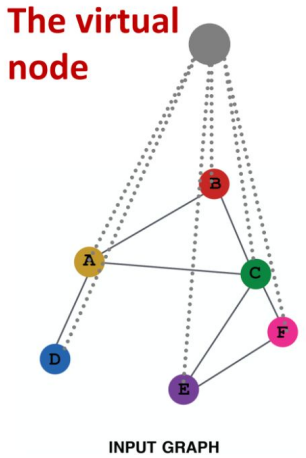
- **Use cases:** Bipartite graphs
  - Author-to-papers (they authored)
  - 2-hop virtual edges make an author-author collaboration graph



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Add Virtual Nodes / Edges

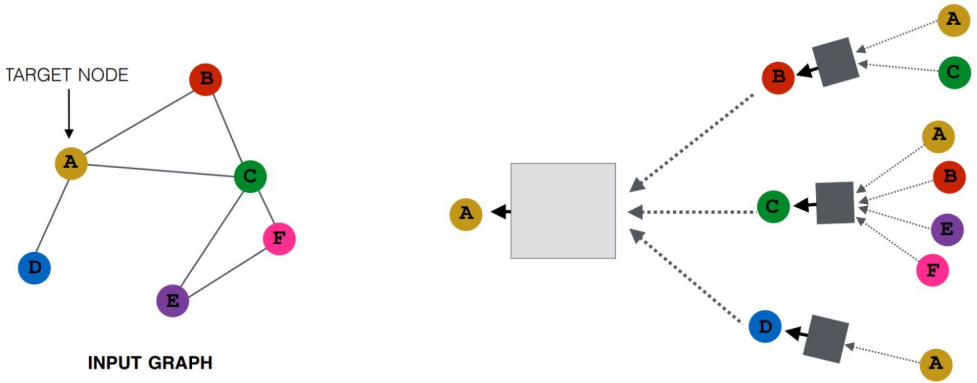
- **Motivation:** Augment sparse graphs
- **(2) Add virtual nodes**
  - The virtual node will connect to all the nodes in the graph
    - Suppose in a sparse graph, two nodes have shortest path distance of 10
    - After adding the virtual node, **all the nodes will have a distance of 2**
      - **Node A – Virtual node – Node B**
  - **Benefits:** Greatly **improves message passing in sparse graphs**



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Node Neighborhood Sampling

- **Our approach so far:**
  - All the neighbors are used for message passing
- **Problem: Dense/large graphs, high-degree nodes**

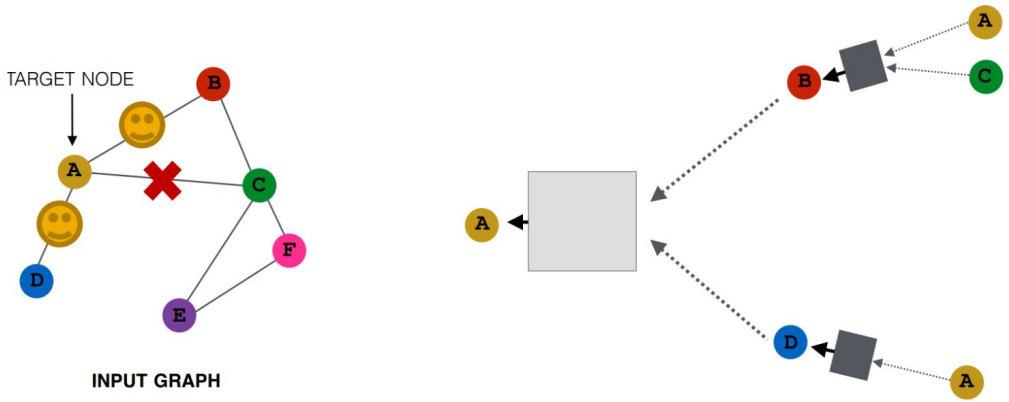


- **New idea:** (Randomly) determine a node's neighborhood for message passing

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Neighborhood Sampling Example

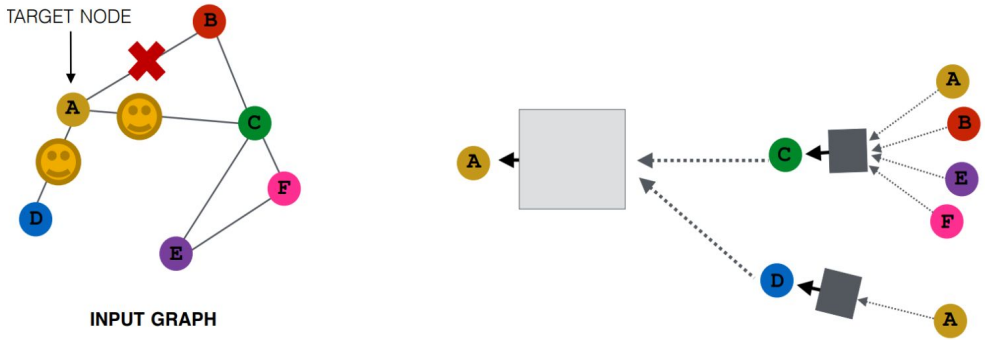
- **For example, we can randomly choose 2 neighbors to pass messages**
  - Only nodes B and D will pass message to A



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Neighborhood Sampling Example

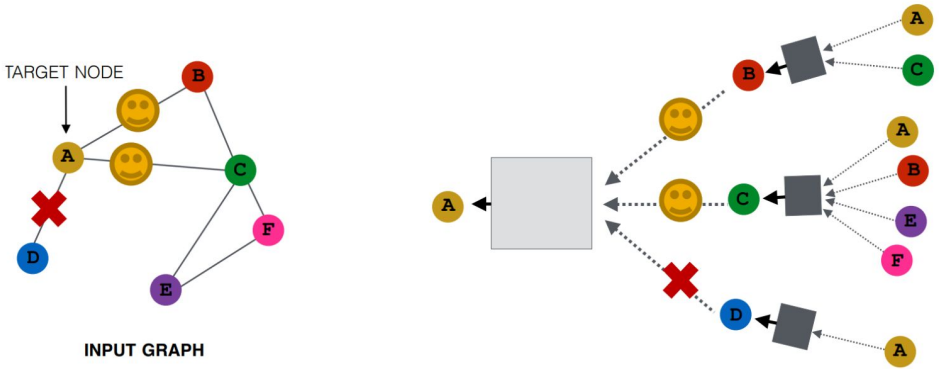
- **Next time when we compute the embeddings, we can sample different neighbors**
  - Only nodes C and D will pass message to A



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Neighborhood Sampling Example

- In expectation, we can get embeddings similar to the case where all the neighbors are used
  - **Benefits:** Greatly reduce computational cost
  - And in practice it works great!



Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf

Summary of the Lecture

- **Recap: A general perspective for GNNs**
  - **GNN Layer**:
    - Transformation + Aggregation
    - Classic GNN layers: GCN, GraphSAGE, GAT
  - **Layer connectivity**:
    - Deciding number of layers
    - Skip connections
  - **Graph Manipulation:**
    - Structure manipulation

Source: https://web.stanford.edu/class/cs246/slides/14-gnn.pdf