

Practical Design of GNNs

소프트웨어 끈대 강의

노기섭 교수

(kafa46@gmail.com)

- From GNN Layers in Practice
 - Batch Normalization
 - Dropout
- Stacking GNN Layers
 - Over-smoothing & Receptive Field
 - GNN Layer Connectivity
 - Skip Connections
- Graph Manipulation
 - Graph Feature Augmentation
 - Graph Structure Manipulation

GNN Layers in Practice

Designing a Practical GNN Layer

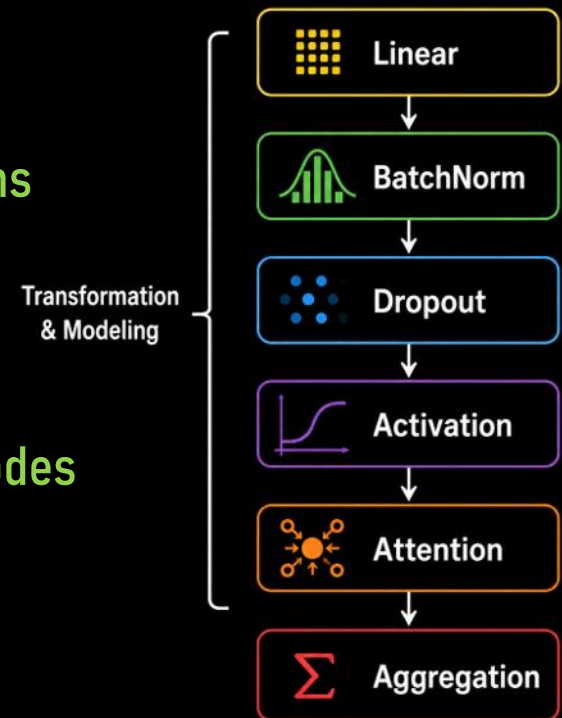
Classic GNN layers provide a strong starting point

- Basic GNNs mainly rely on transformation and neighborhood aggregation
- Their performance can often be improved with a more general layer design

Incorporate modern deep learning modules

- Linear transformation: transforms node features
- Batch Normalization: stabilizes feature distributions
- Dropout: provides regularization
- Activation: introduces non-linearity
- Attention: learns the importance of neighboring nodes
- Aggregation: combines neighborhood information

A Practical GNN Layer



Batch Normalization for Node Embeddings

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

N : number of nodes

D : embedding dimension

$X_{i,j}$: node i 's j -th feature

예를 들어 $D=64$ 라면 64개의 feature dimension 각각에 대해 별도의 mean과 variance가 계산

Step 1:

Compute the mean & variance over N embeddings

$$\mu_j = \frac{1}{N} \sum_{i=1}^N X_{i,j}$$

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

Step 2:

Normalize the feature using computed mean & variance

$$\hat{X}_{i,j} = \frac{X_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

$mean \approx 0, var \approx 1$

Step 3:

Learnable Scale & Shift

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

$$Y_{i,j} = \gamma_j \hat{X}_{i,j} + \beta_j$$



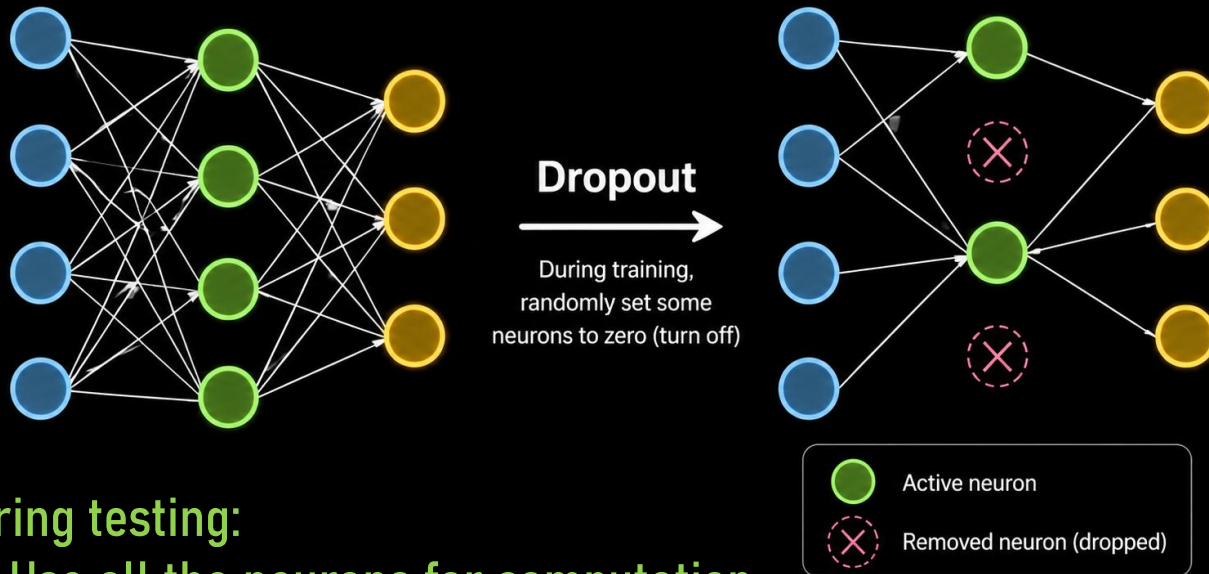
Output: $Y \in \mathbb{R}^{N \times D}$

Dropout: Regularization by Random Deactivation

Goal: Reduce overfitting in neural network

During Training:

- with probability p , randomly set neurons to zero (turn off)
- Importantly, a different set of neurons is randomly dropped at each training step.



During testing:

- Use all the neurons for computation

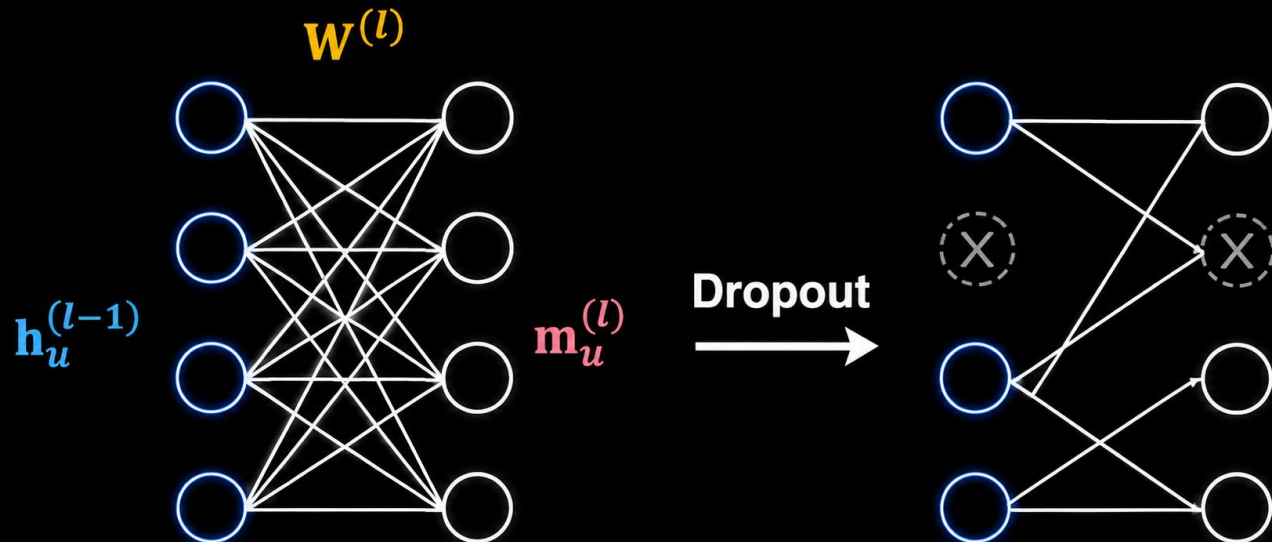
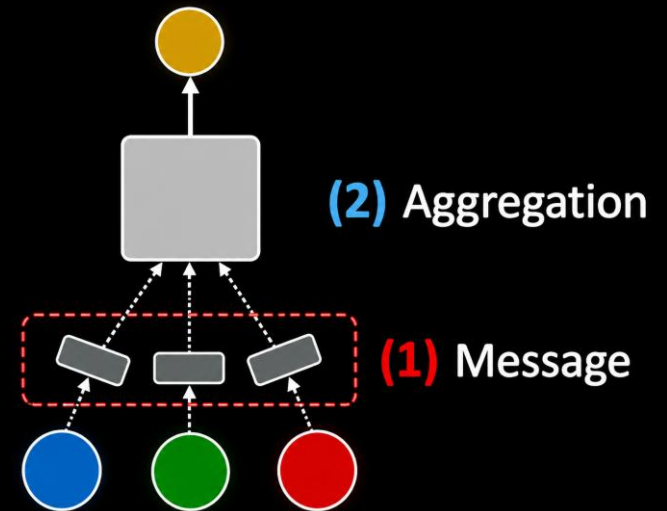
Dropout in GNN Message Passing

$$\mathbf{m}_u^l = \mathbf{W}^l \mathbf{h}_u^{l-1}$$

$\mathbf{h}_u^{l-1}$: Node representation of previous layer

$\mathbf{W}^l$: Learnable weight matrix

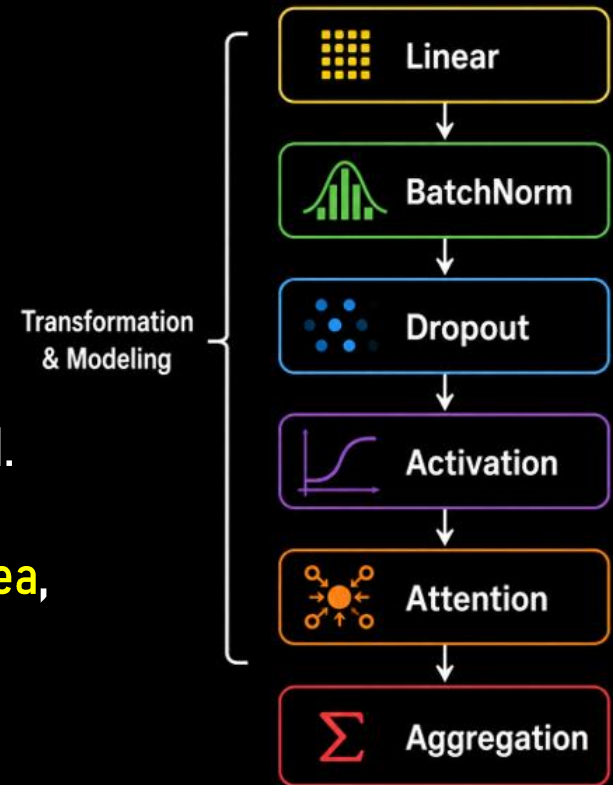
$\mathbf{m}_u^l$: Message for passing to neighbors



Designing a Modern GNN Layer

- **Practical GNN layers** combine standard deep-learning modules such as Linear, BatchNorm, Dropout, and Activation.
- **Attention and Aggregation** determine how information from neighboring nodes is combined.
- **GNN layer design remains an active research area**, with many possible architectural choices.

A Practical GNN Layer

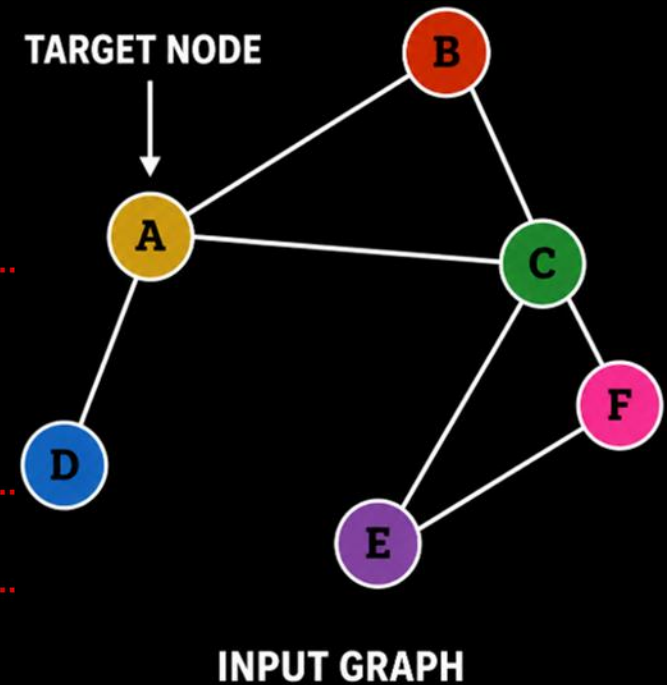
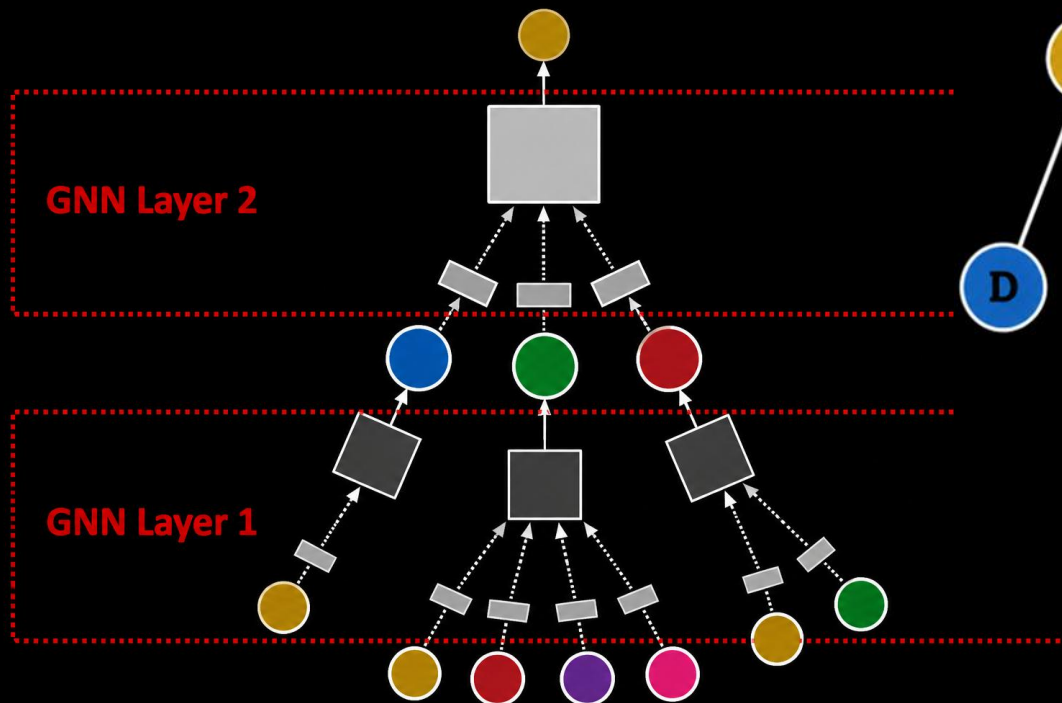


Stacking GNN Layers

Connecting Multiple GNN Layers

How to connect GNN layers into a GNN?

- Stack layers sequentially
- Ways of adding skip connections

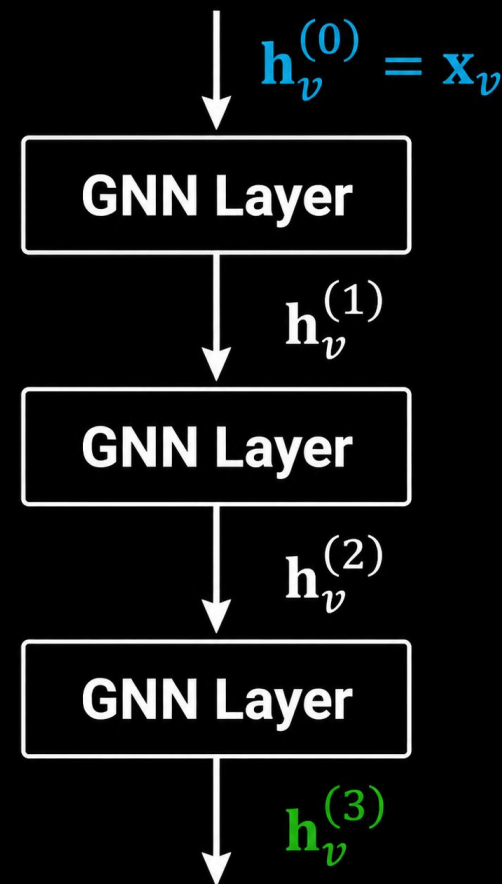


Stacking GNN Layers for Node Representation Learning

Standard way: Stack GNN layers sequentially

- **Input:** Initial raw node feature X_v
- **Output:** Node embeddings h_v^l after L GNN layers

One GNN layer
 $\approx$ one-hop information propagation



Over-Smoothing in Deep GNNs

The Problem:

As more GNN layers are stacked, node embeddings tend to become increasingly similar.

Why it Matters:

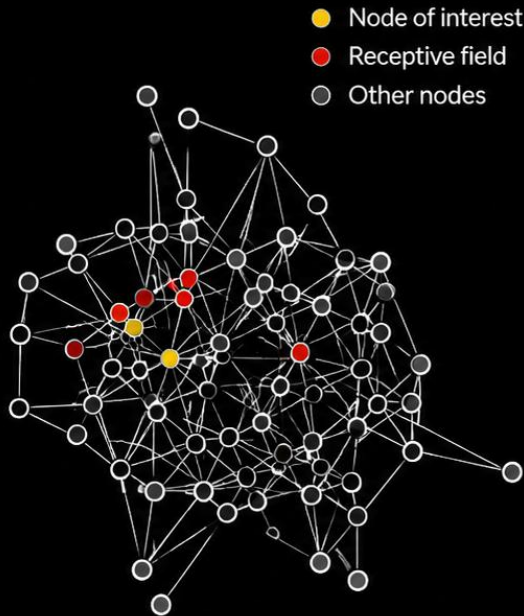
Over-smoothing makes it difficult to distinguish different nodes using their embeddings.

Question:

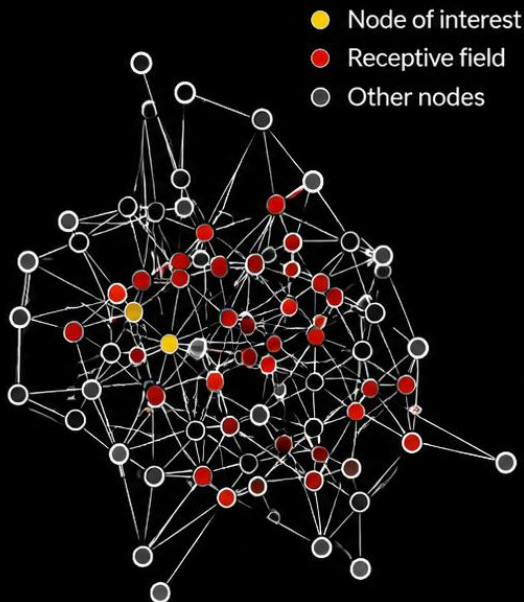
Why does stacking many GNN layers cause node representations to become similar?

GNN Depth and Receptive Field

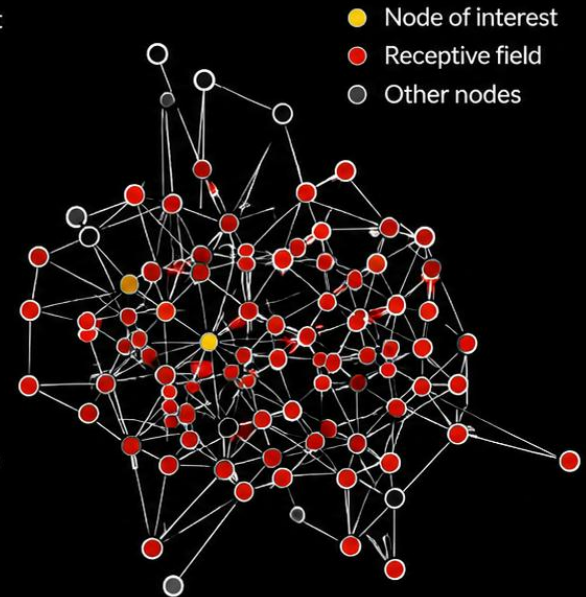
**Receptive field for
1-layer GNN**



**Receptive field for
2-layer GNN**



**Receptive field for
3-layer GNN**



Receptive Field:

The set of nodes whose information can influence the representation of a target node.

Effect of GNN Depth:

Each GNN layer propagates information by one additional hop.

Why Does Over-Smoothing Occur?

Receptive Field Overlap:

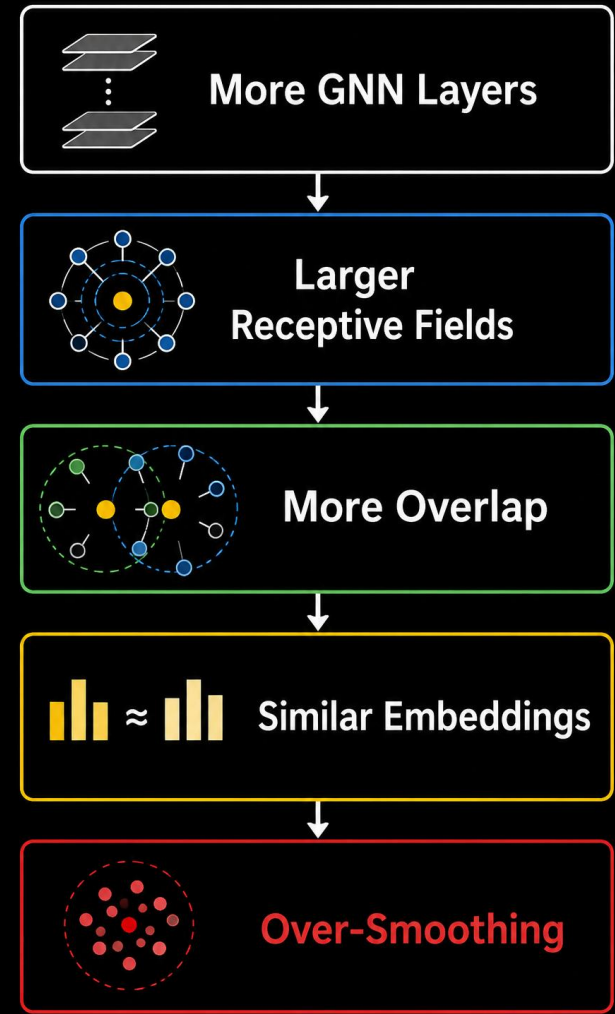
A node embedding is determined by the information within its receptive field.

As GNNs become deeper:

Receptive fields expand and increasingly overlap across nodes.

Consequence:

Nodes with highly overlapping receptive fields aggregate similar information and therefore learn similar embeddings.



Designing 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?**

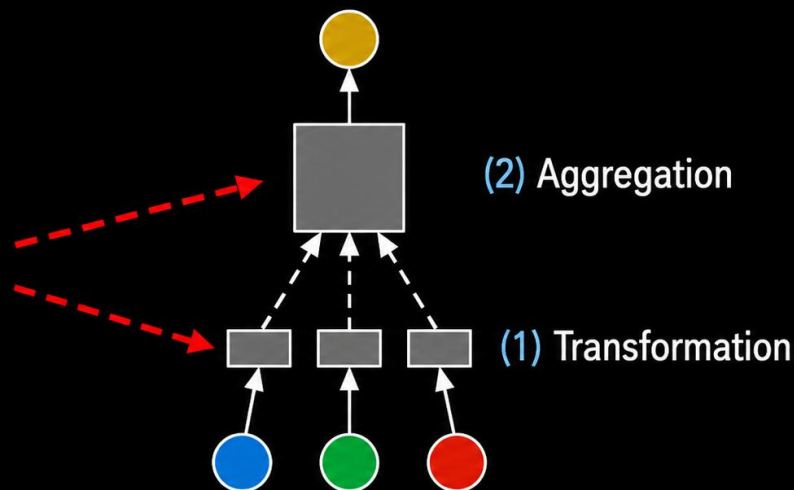
Increasing Expressive Power within Each GNN Layer (1/2)

How to make a shallow GNN more expressive?

Solution 1: Increase the expressive power within each GNN layer

- Each transformation or aggregation function only include one linear layer
- We can make aggregation / transformation become a deep neural network!

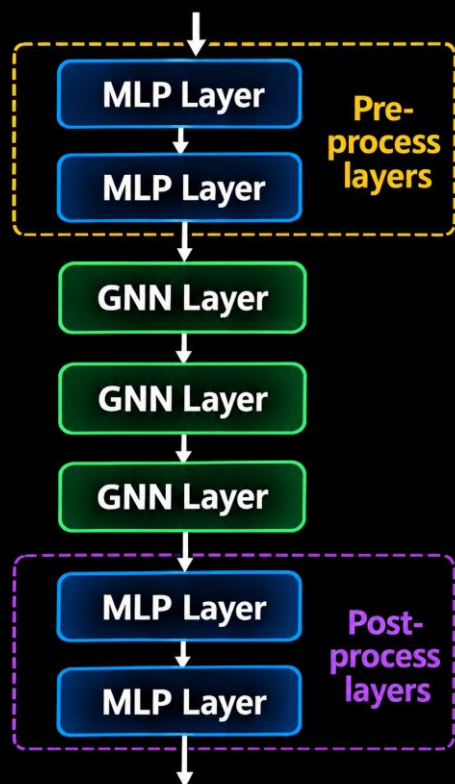
If needed, each box could include a **3-layer MLP**



Increasing Expressive Power within Each GNN Layer (2/2)

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

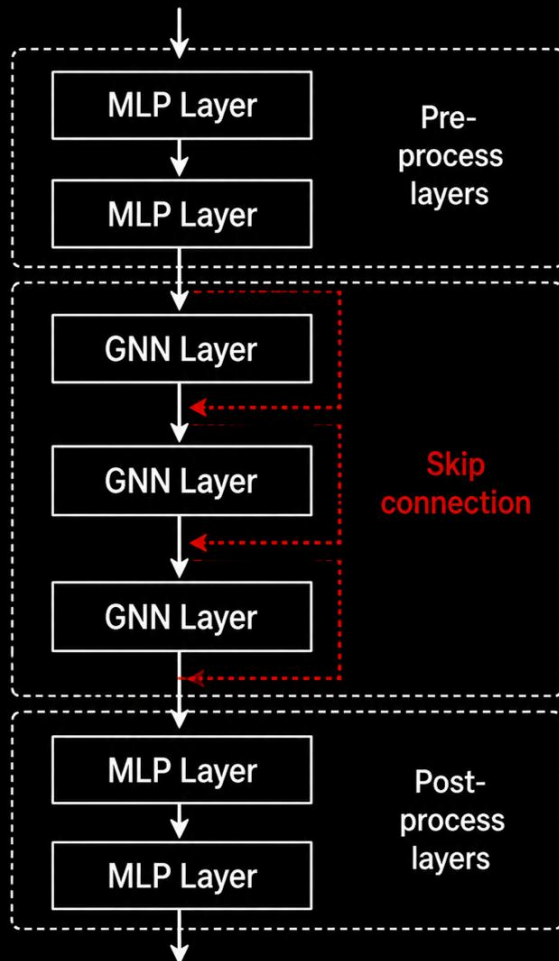
Important when encoding node features is necessary
E.g., when nodes represent images/text

Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

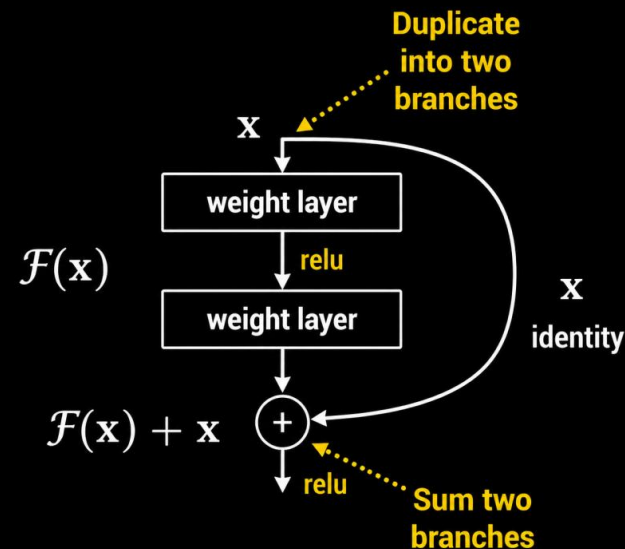
Skip Connections for Deep GNNs

What if my problem still requires many GNN layers?

Add skip connections!



- 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

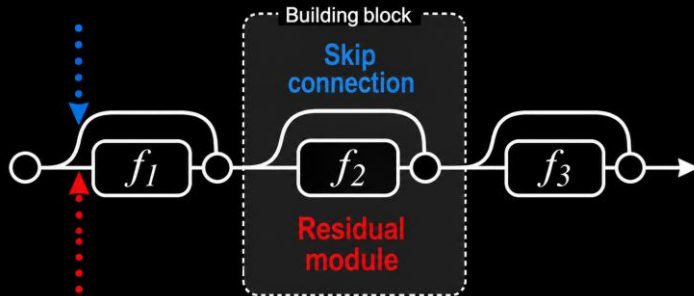


Skip Connections as a Mixture of Shallow and Deep GNNs

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

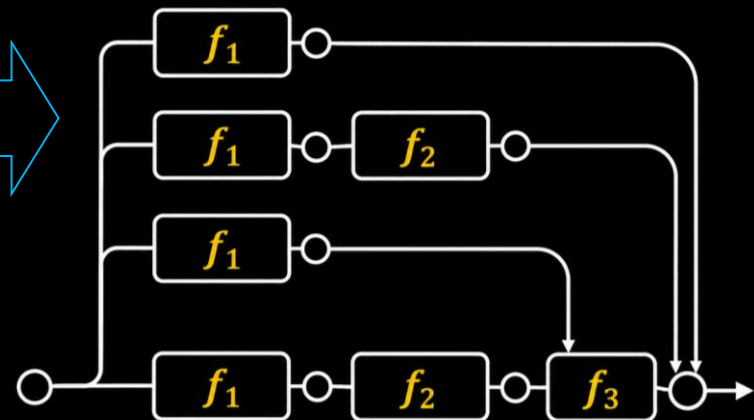
Path 2: skip this module



Path 1: include this module

All the possible paths:

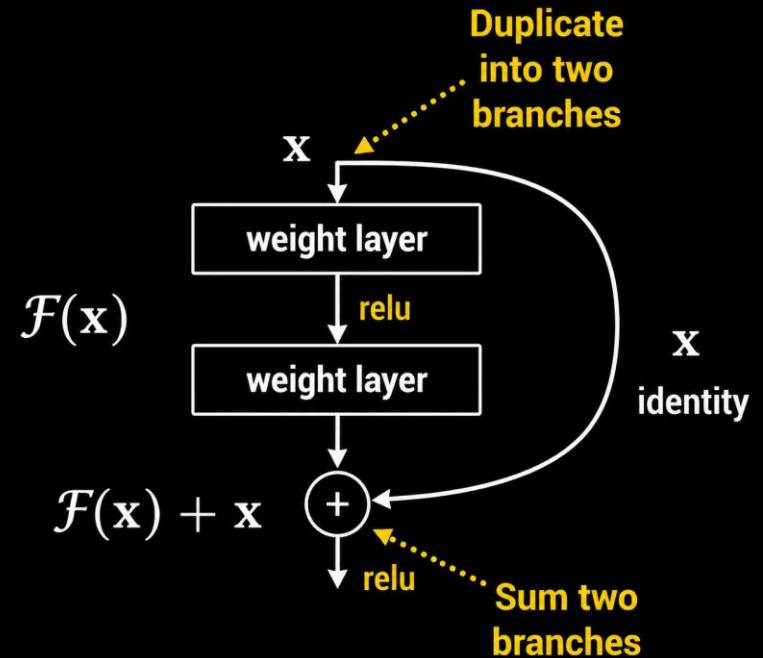
$$2 * 2 * 2 = 2^3 = 8$$



Example: GCN with Skip Connections

Standard GCN Layer

$$h_v^l = \sigma \left(\sum_{u \in N(v)} W^l \frac{h_u^{l-1}}{|N(v)|} \right)$$



GCN Layer with Skip Connection

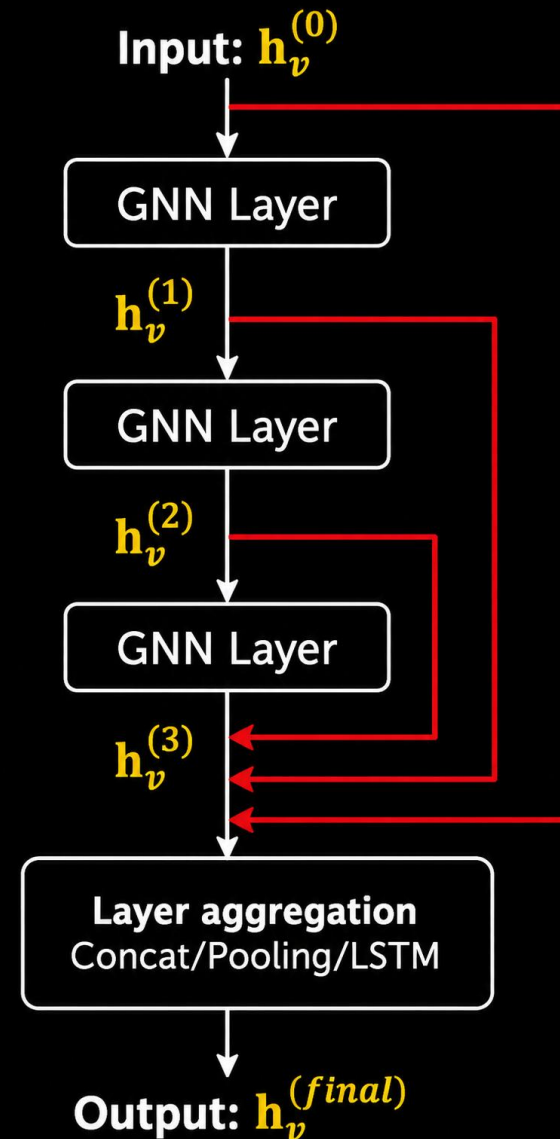
$$h_v^l = \sigma \left(\sum_{u \in N(v)} W^l \frac{h_u^{l-1}}{|N(v)|} + h_v^{l-1} \right)$$

$F(X) \quad + \quad X$

Skip Connections with Layer Aggregation

Directly skip to the last layer

The final layer directly aggregates from all the node embeddings in the previous layers



Graph Manipulation in GNNs

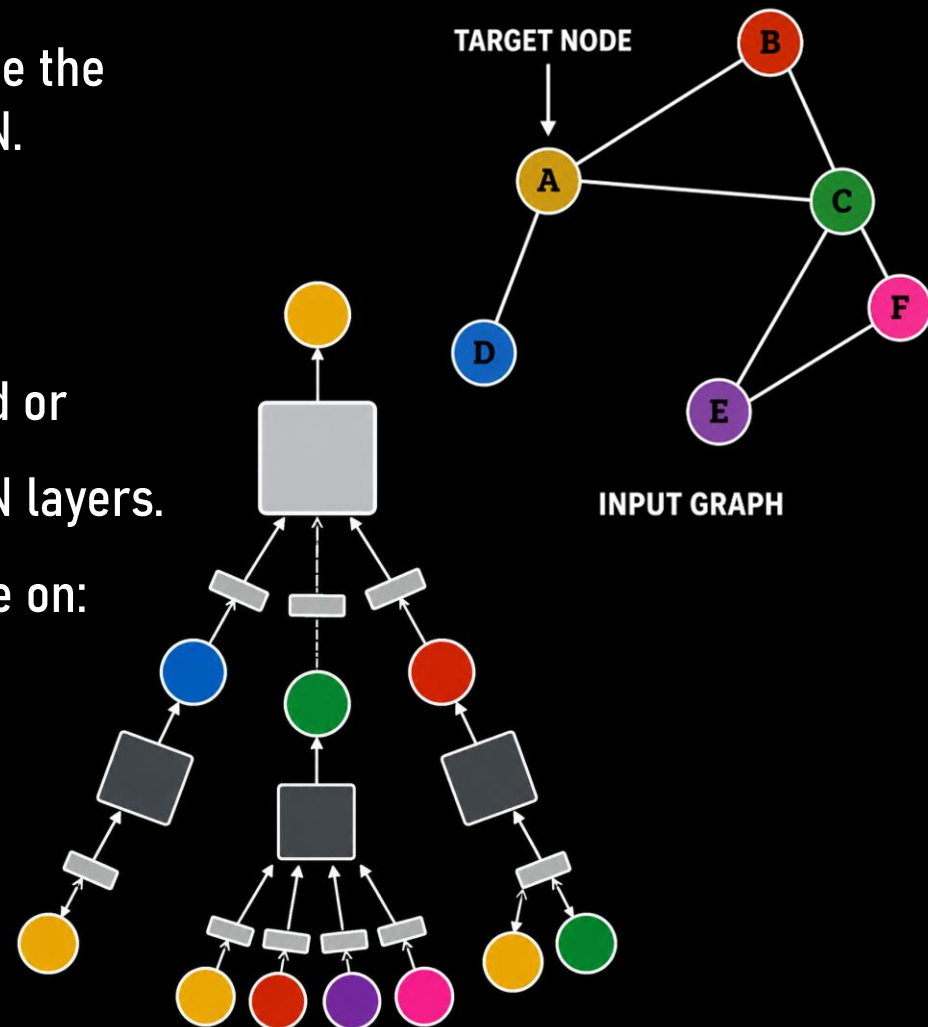
General GNN Framework

Key Idea:

Raw input graph does not have to be the computational graph used by a GNN.

Before Message Passing:

- The input graph can be modified or augmented before applying GNN layers.
- Graph manipulation can operate on:
 - Graph features
 - Graph structure



Why Manipulate Graphs?

Raw input graph = computational graph
Reasons for breaking this assumption

Real-world graphs are not always optimal for GNN message passing

- The given graph structure may not effectively support information propagation.

Graph structure affects how information flows

- GNNs propagate messages according to the connections between nodes.
- Poor connectivity can limit effective message passing.

Graph manipulation can improve GNN computation

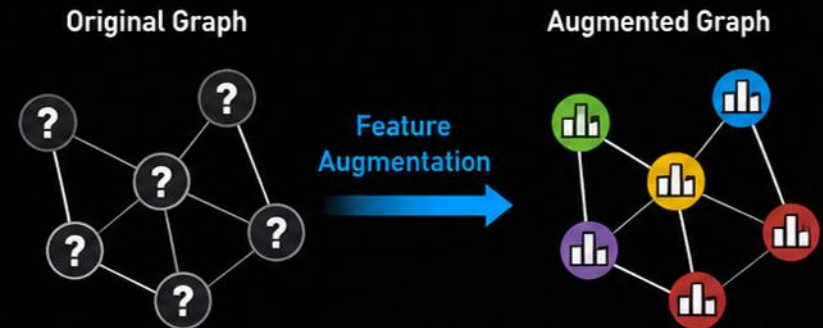
- Modify node/edge features
- Modify graph structure
- Control which neighbors participate in message passing

Graph Manipulation Approaches

Graph Feature Manipulation

Missing or insufficient features

→ **Feature augmentation**



Graph Structure Manipulation

Sparse graph

→ **Add virtual nodes / edges**



Dense graph

→ **Sample neighbors during message passing**



Large graph

→ **Sample subgraphs for efficient computation**



Adding Virtual Nodes and Edges (1/2)

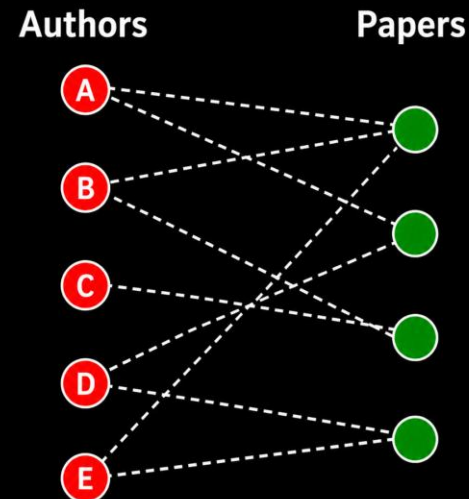
Motivation: Augment Sparse Graphs

Add virtual edges

- Common approach: Connect 2-hop neighbors via virtual edges
- Intuition: Instead of using adjacency matrix A for GNN computation, use $A + A^2$
 - A : one-hop paths
 - A^2 : two-hop paths

Use case: Bipartite graph

- Author-to-papers (they authored)
- 2-hop virtual edges make an author-author collaboration graph



Adding Virtual Nodes and Edges (2/2)

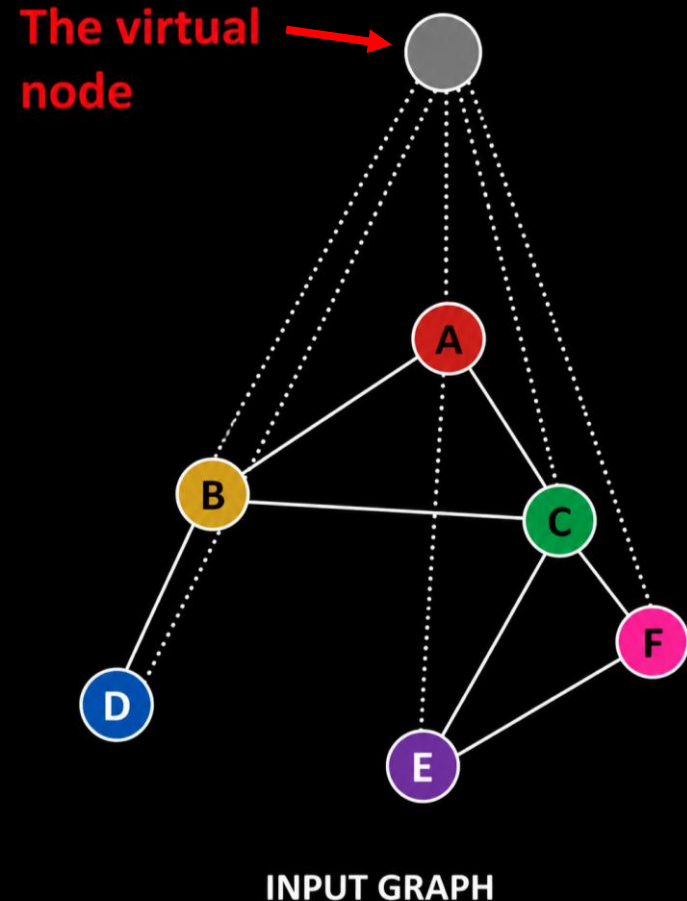
Motivation: Augment Sparse Graphs

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 n
 - After adding the virtual node, **all the nodes will have the distance of 2**
 - Node A – Virtual node – Node B

Benefits:

Enables efficient long-range message passing in sparse graphs

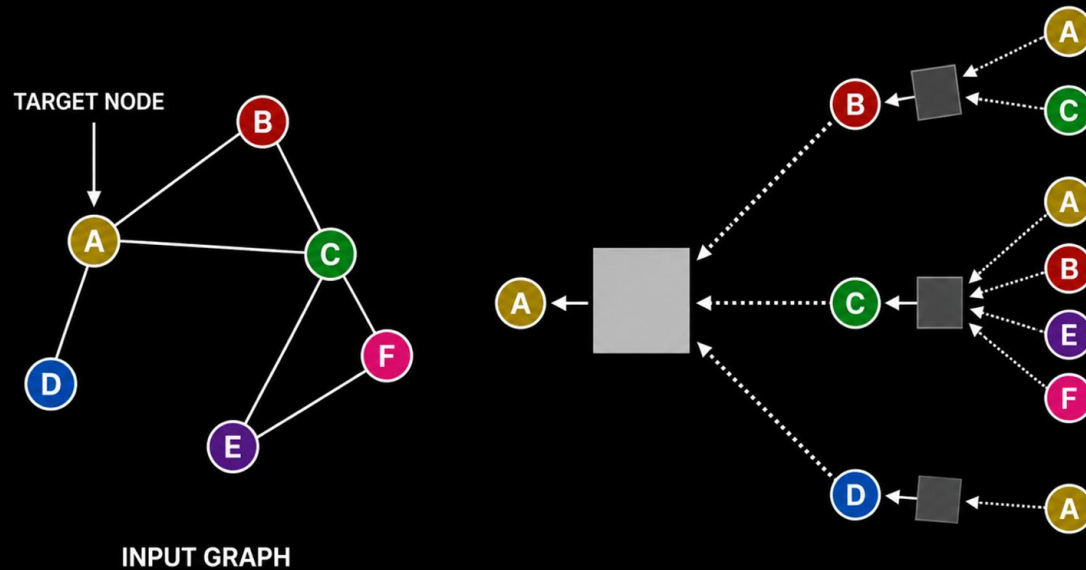


Node Neighborhood Sampling

Our approach so far:

All the neighbors are used for message passing

Problem: Dense/large graphs, high-degree nodes



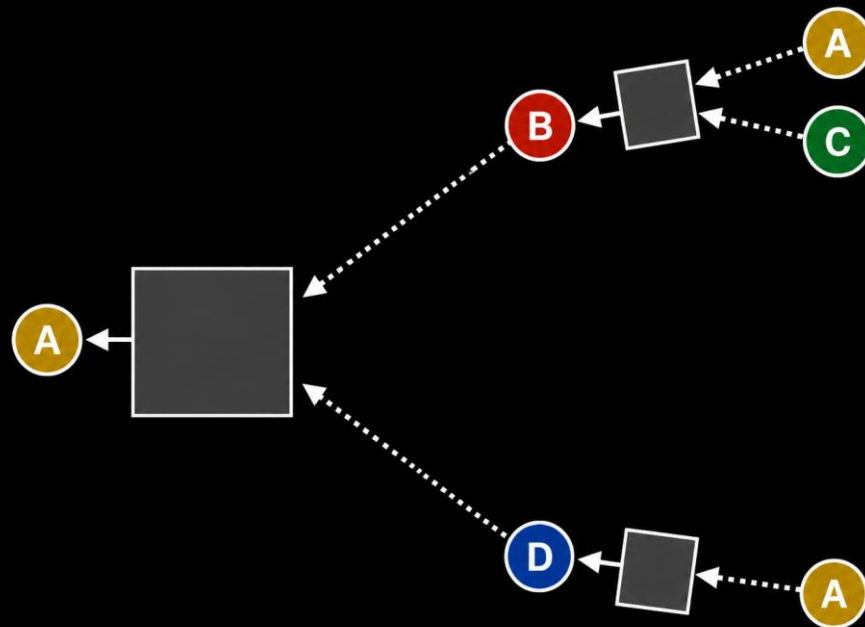
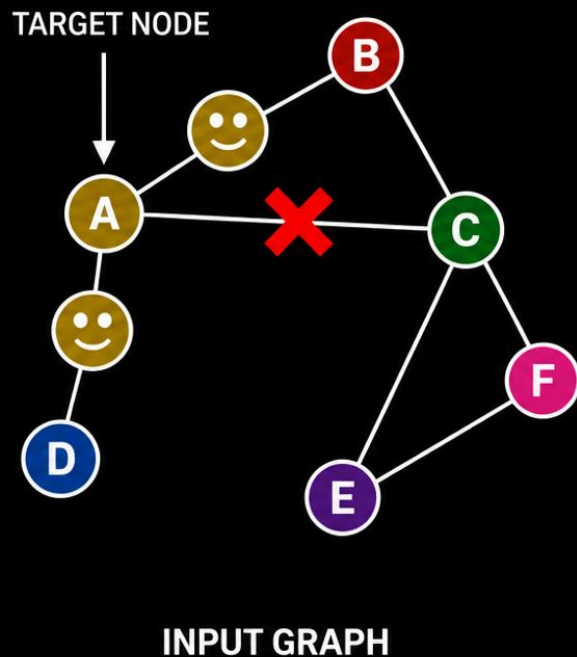
New idea:

(Randomly) determine a node's neighborhood for message passing

Neighborhood Sampling in GNN Layers (1/3)

Randomly choose 2 neighbors to pass messages

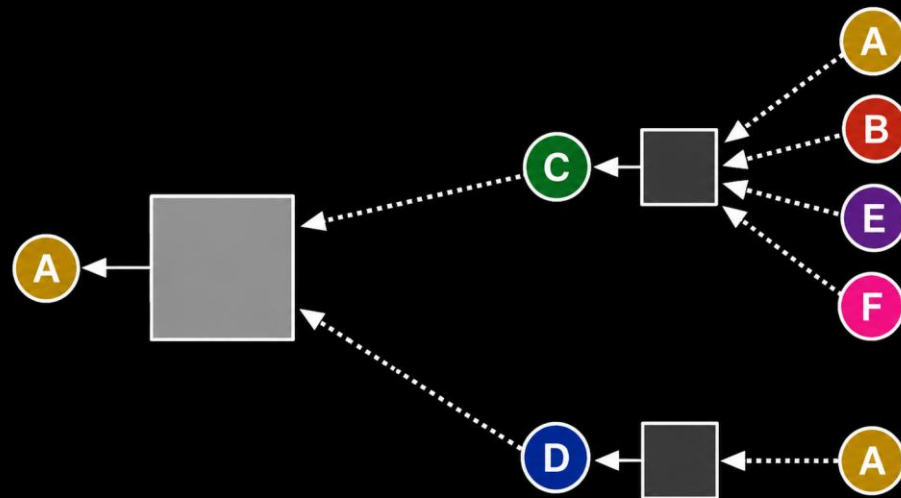
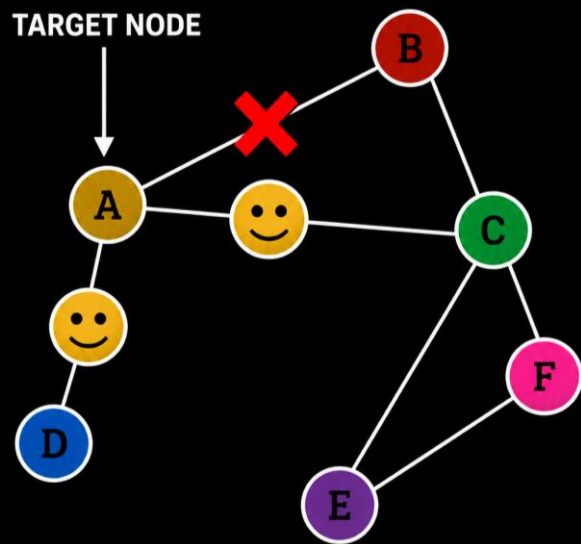
- Only nodes **B** and **D** will pass message to **A**



Neighborhood Sampling in GNN Layers (2/3)

Next time when we compute the embeddings, we can sample different neighbors

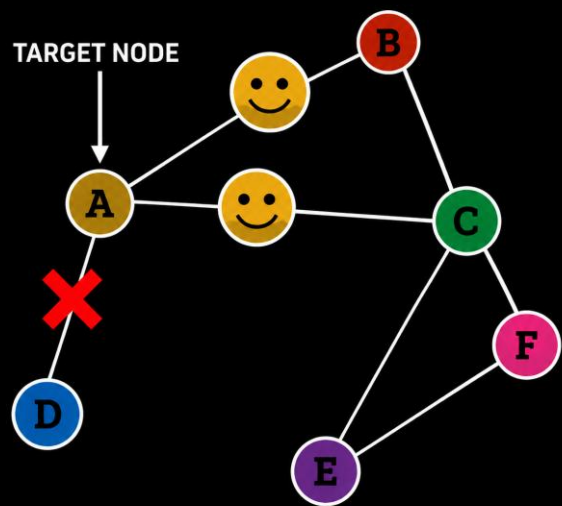
- Only nodes **C** and **D** will pass message to **A**



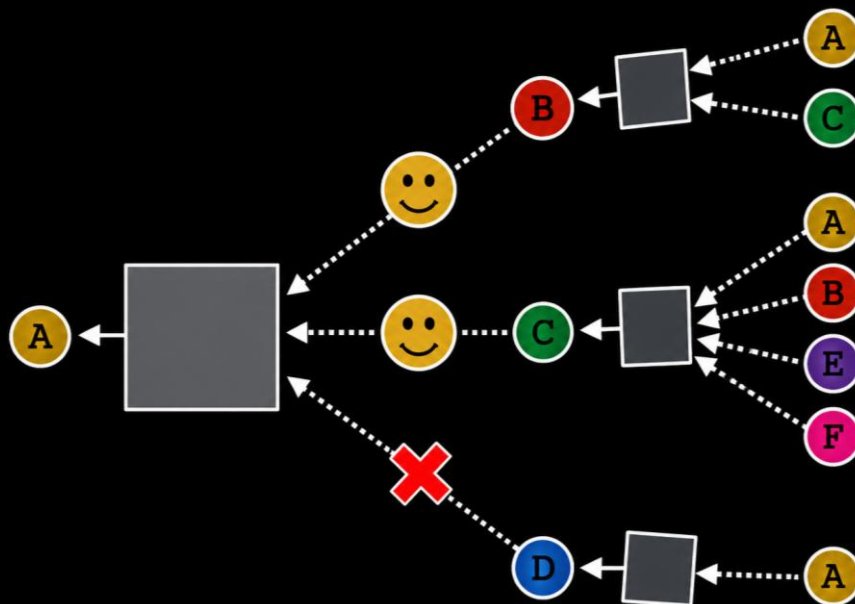
Neighborhood Sampling in GNN Layers (3/3)

In expectation, we can get embeddings like the case where all the neighbors are used

- Benefits: Greatly reduce computational cost
- And in practice it works great!



INPUT GRAPH





수고하셨습니다 ..^^..