

Message Passing Neural Networks

소프트웨어 끈대 강의

노기섭 교수

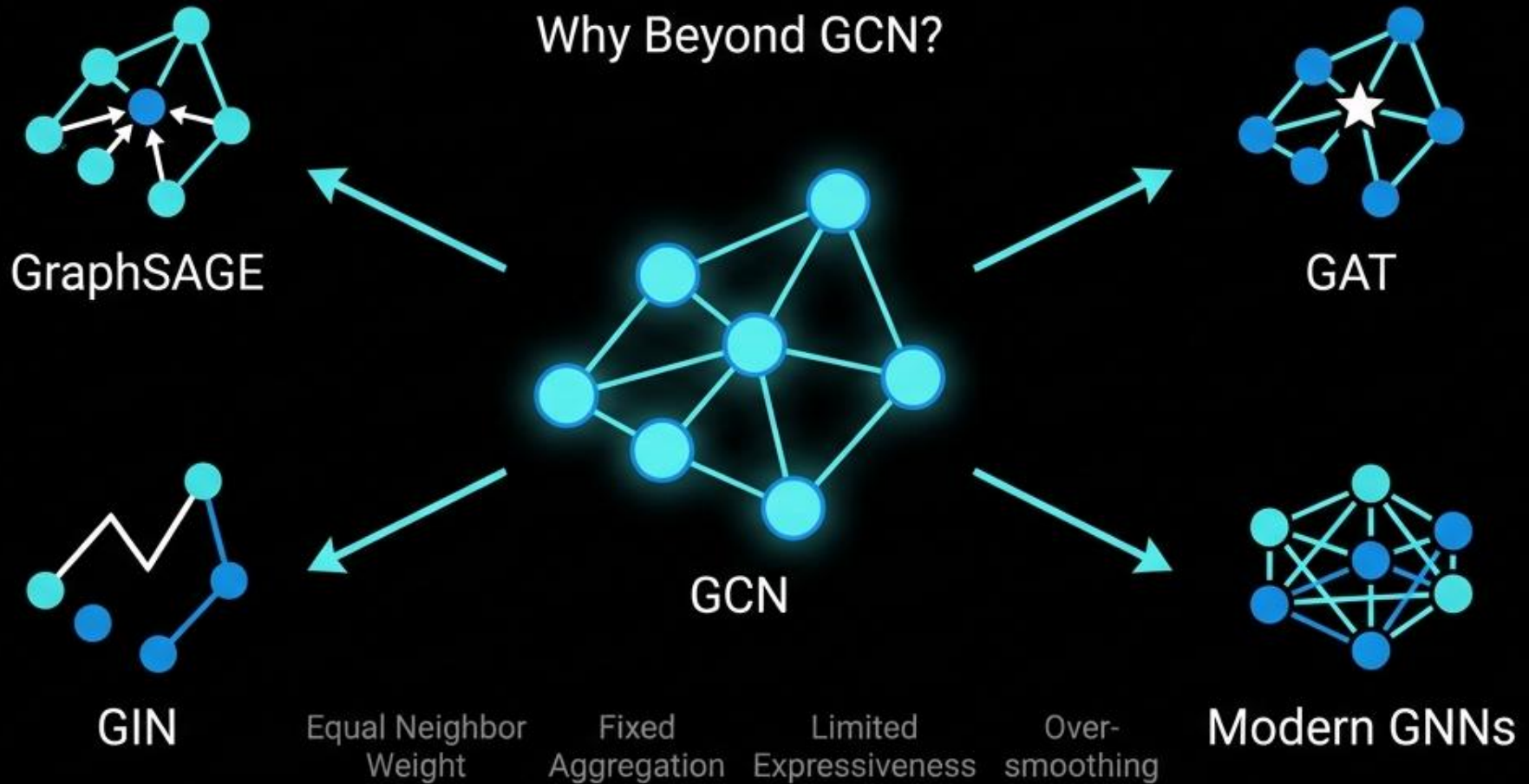
(kafa46@gmail.com)

- Why Beyond GCN?
- A Unified Message Passing Framework
- Message, Aggregation, and Update Functions
- Comparative Analysis of Representative GNN Models
 - GCN
 - GraphSAGE
 - GAT

Why Beyond GCN?

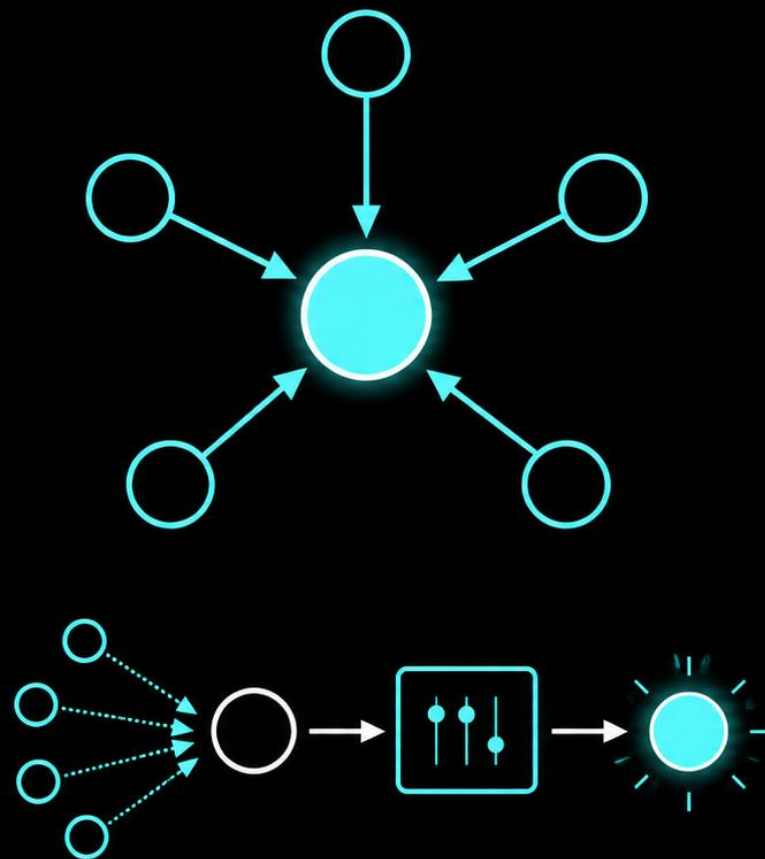
Why Beyond GCN?

Why Beyond GCN?

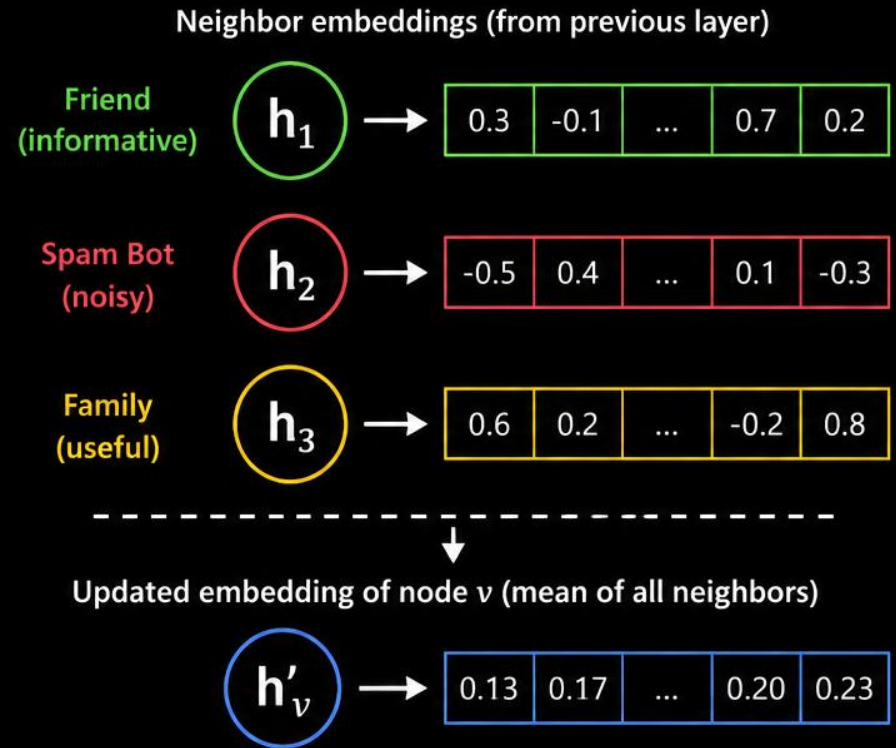
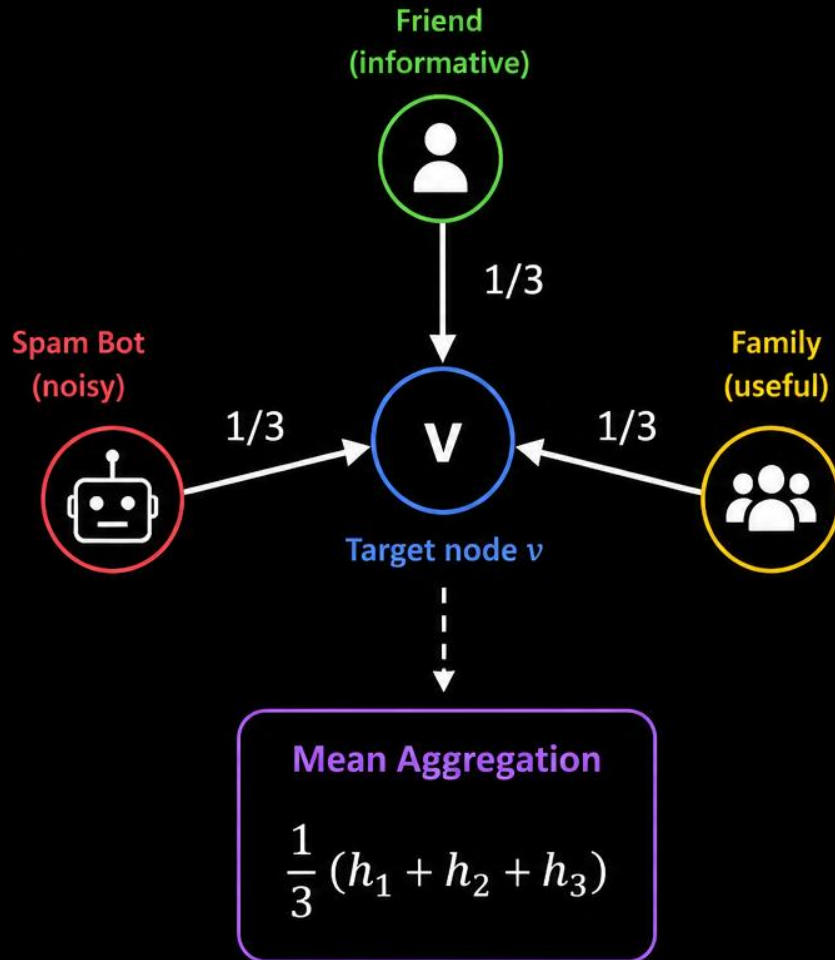


Review: What Does GCN Do?

- Aggregate information from neighboring nodes
- Share the same transformation across all nodes
- Update node representations layer by layer
- Repeat across multiple GCN layers



Limitation 1: Equal Importance of Neighbors

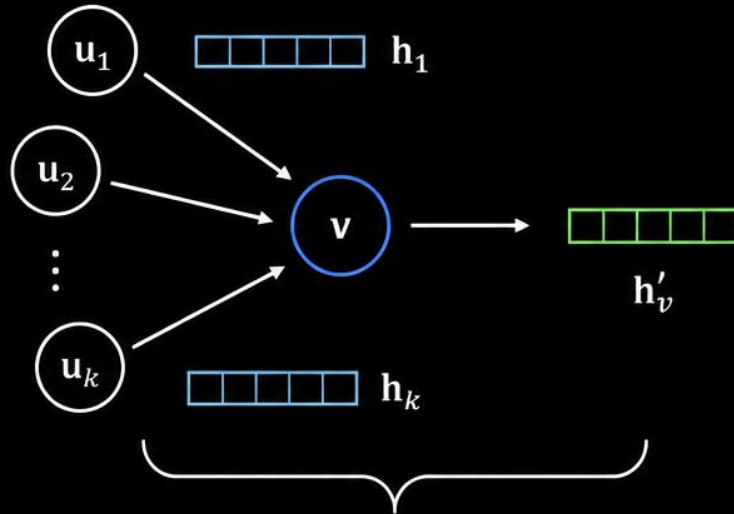


Problem

Important neighbors
↓
treated exactly the same as
↓
irrelevant or noisy neighbors

Limitation 2: Fixed Aggregation

GCN uses only Mean Aggregation



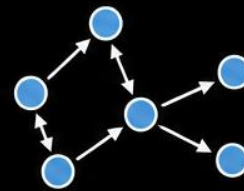
Mean Aggregation

$$h'_v = \frac{1}{|N(v)|} \sum_{u \in N(v)} h_u$$

Fixed Aggregation Rule

Different graphs require different aggregation strategies

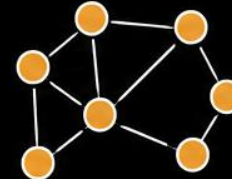
Citation Network



Mean



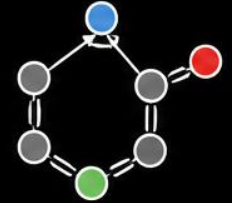
Social Network



Max



Molecular Graph



Sum

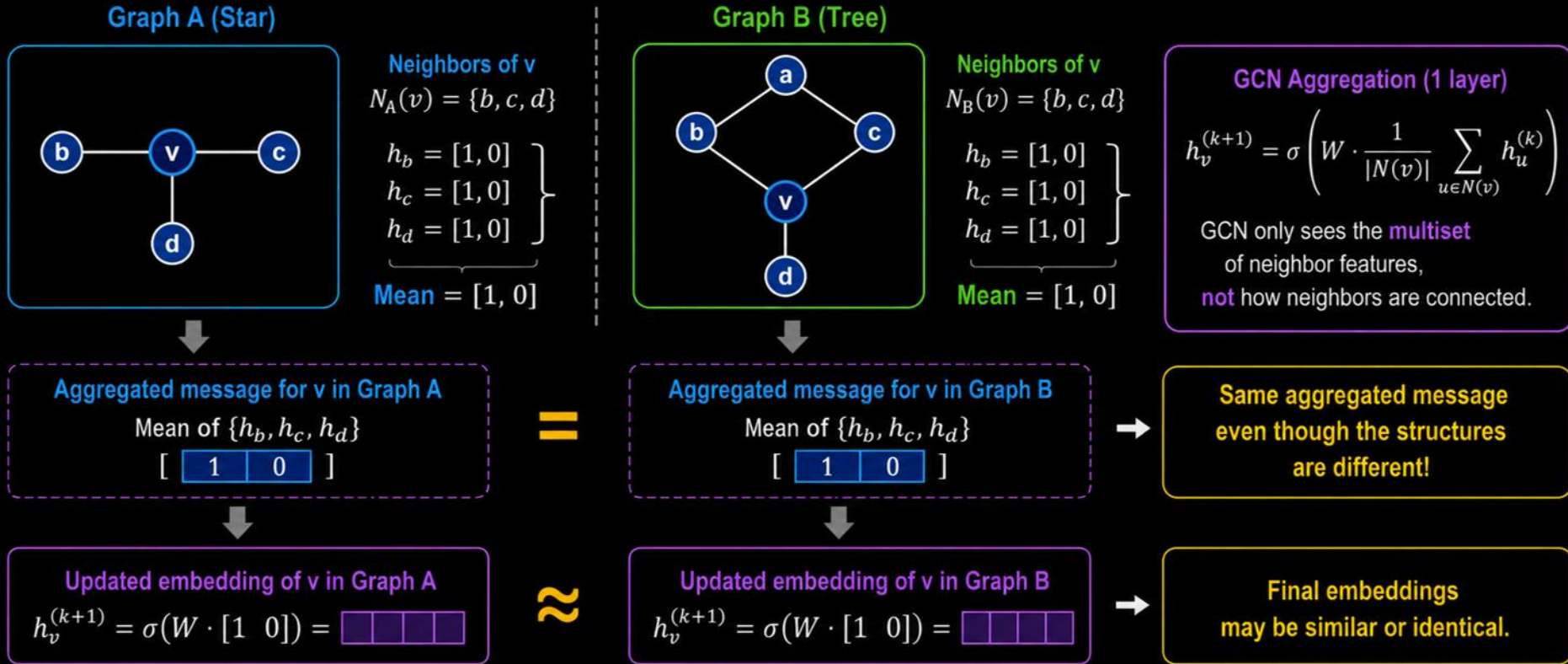


Different graphs
Different aggregation

Problem

One aggregation function
cannot fit
every graph

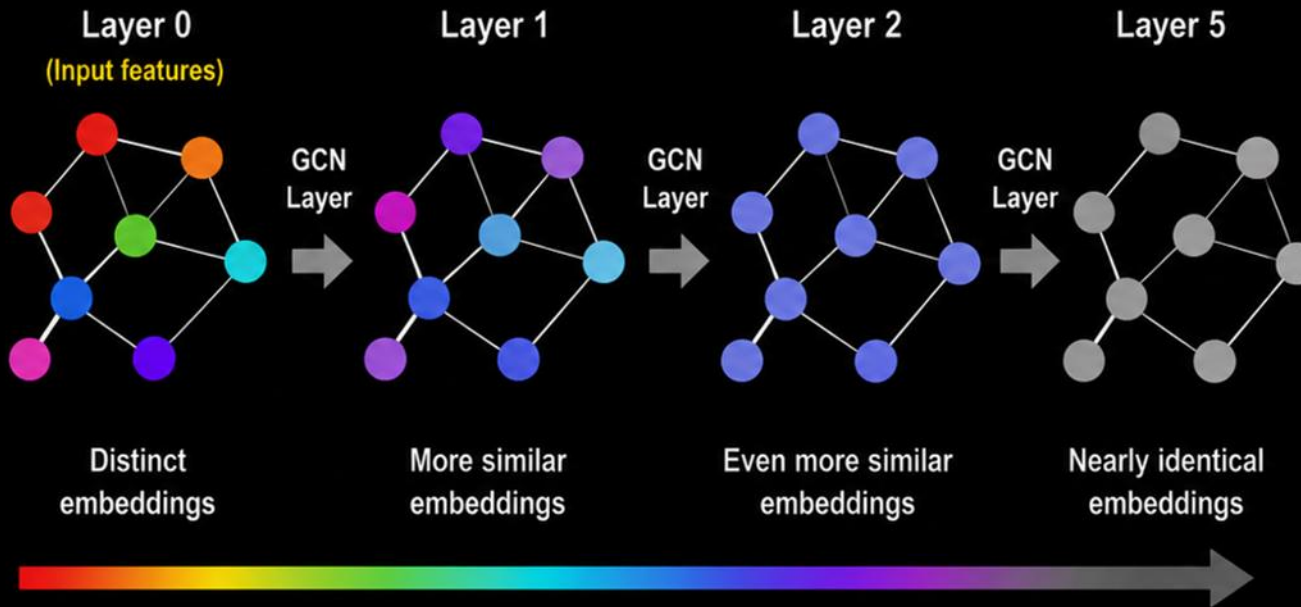
Limitation 3: Limited Expressive Power



- Because GCN uses only **MEAN** of neighbor features,
- GCN **ignores how neighbors are connected**
→ **lose** important **structural information**

Limitation 4: Over-Smoothing

As GNN layers increase, node embeddings become **increasingly similar**, losing their ability to distinguish nodes.



! Problem

Over many layers, different nodes become **indistinguishable**.



- Node embeddings converge to a **similar value**.
- GNNs **lose discriminative power** on deep architectures.

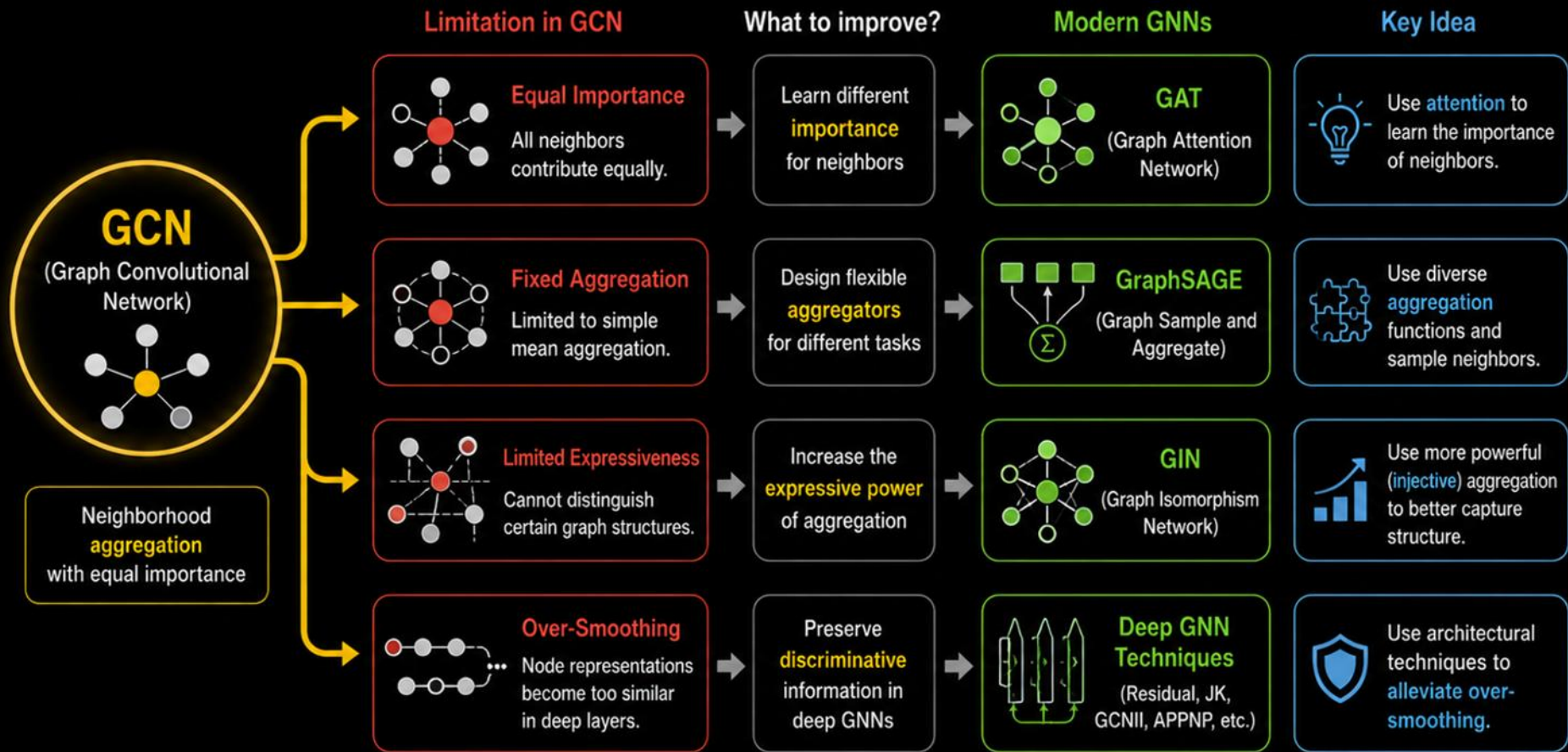


Too many GNN layers → **Over-smoothing**
→ **Loss of node identity** and poor performance.

From GCN to Modern GNNs

Modern GNNs do **NOT** replace GCN.

They **EXTEND & IMPROVE** it by **addressing its key limitations**.

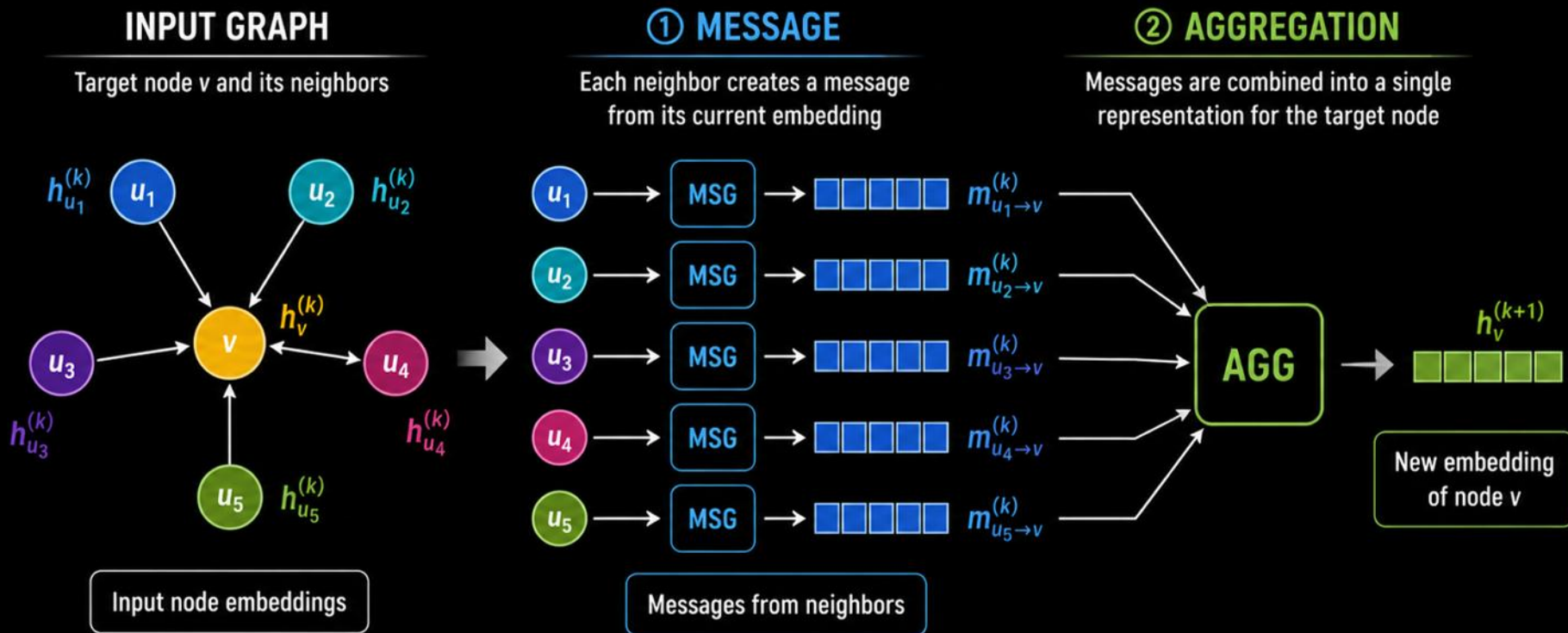


A Unified Message Passing Framework

A Unified Message Passing Framework

GNN Layer: Compress a set of vectors into single vector!

- Two-step process:** Message Computation + Aggregation



Two-step Process: (1) Message Computation

Message Function:

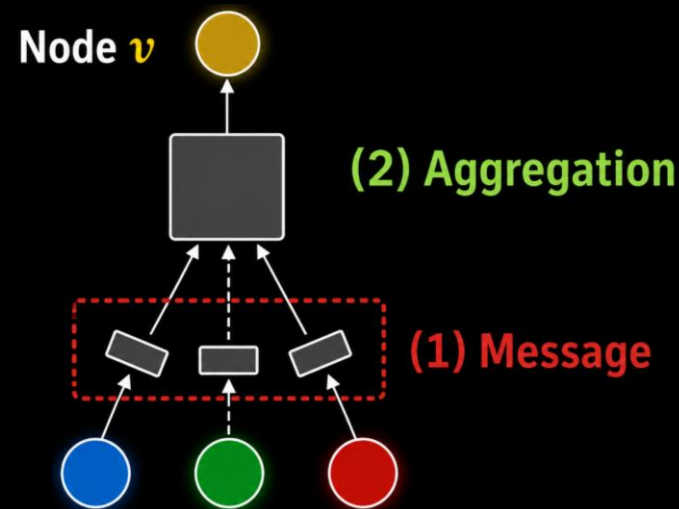
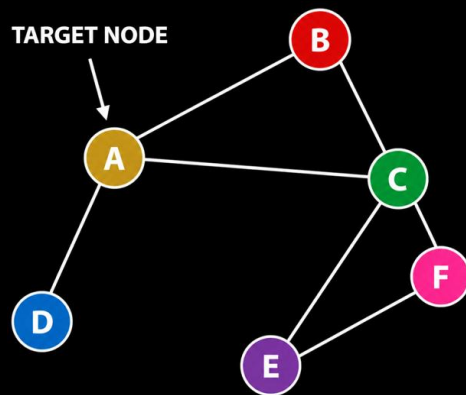
$$m_u^l = MSG^l(h_u^{l-1})$$

- **Intuition:** Each node will create a message, which will be sent to other nodes later

- **Example:**
A linear layer

$$m_u^l = W^l h_u^{l-1}$$

Multiply node features with weight matrix W^l



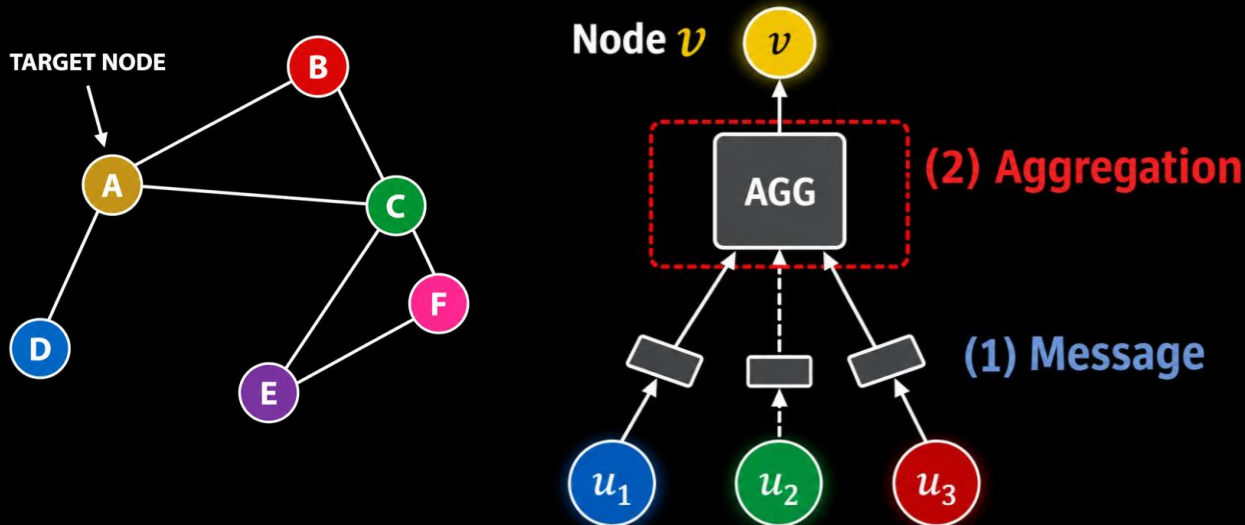
Two-step Process: (2) Aggregation

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

$$h_v^l = AGG^l(\{m_y^l, u \in N(v)\})$$

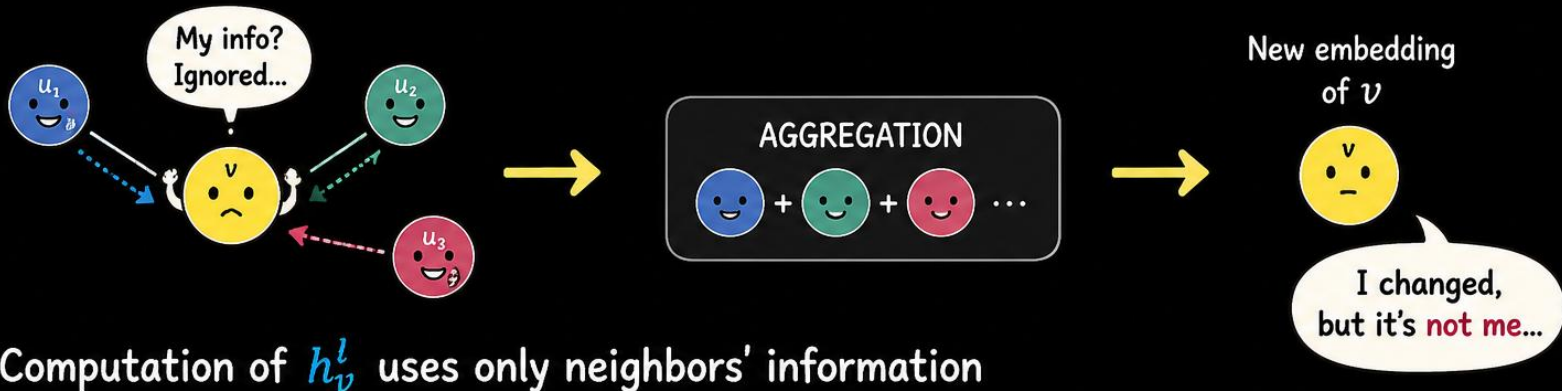
- **Example:** $Sum(\cdot)$, $Mean(\cdot)$, or $Max(\cdot)$

$$h_v^l = Sum(\{m_y^l, u \in N(v)\})$$

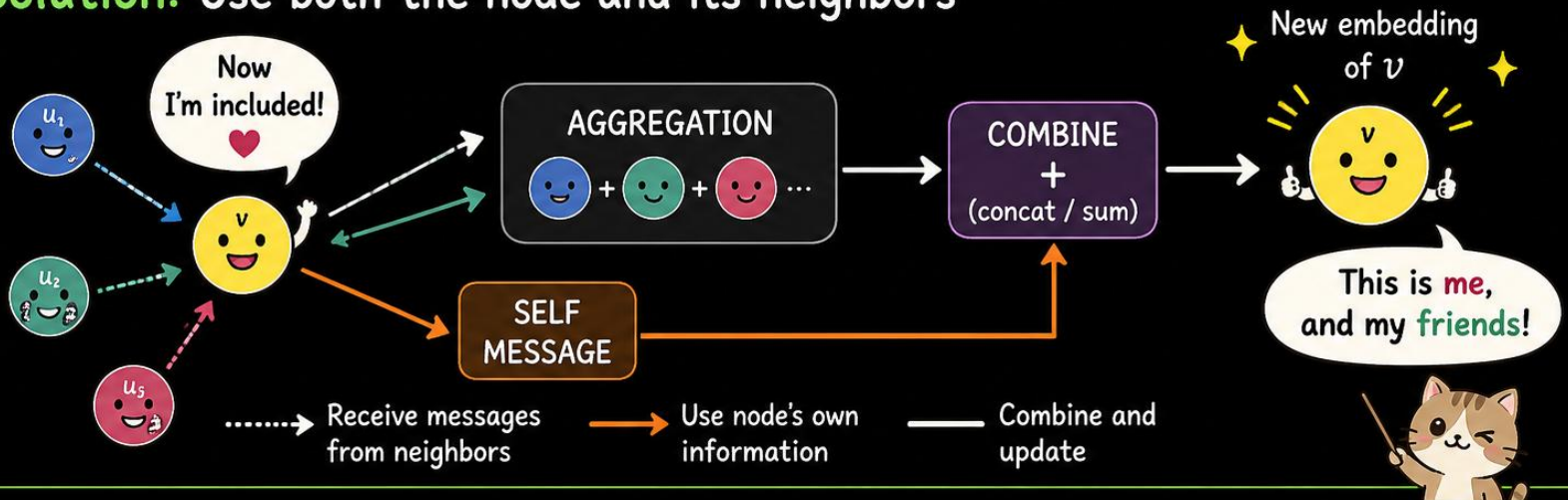


Issue on Message Aggregation

Issue: The node's own information is not used.



Solution: Use both the node and its neighbors



Issue on Message Aggregation – Formal Solution

(1) Message: compute message from node v itself

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



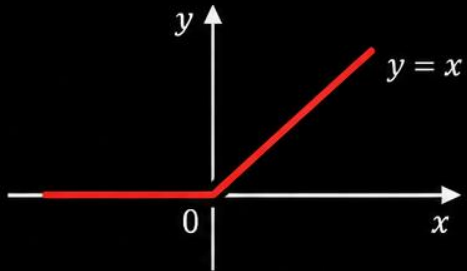
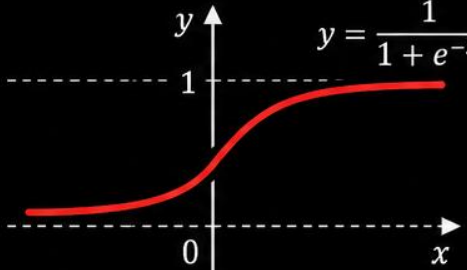
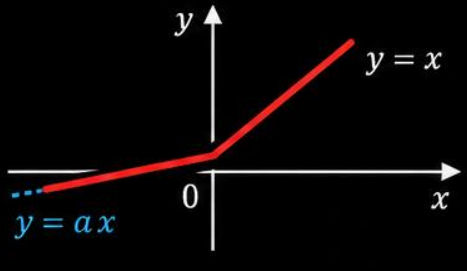
(2) Aggregation: After aggregating from neighbors, we can aggregate the message from node v itself

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



$$h_v^{(l)} = \text{CONCAT} \left(\underbrace{\text{AGG} \left(\left\{ m_u^{(l)}, u \in N(v) \right\} \right)}_{\text{First aggregate from neighbors}}, \underbrace{m_v^{(l)}}_{\text{Then aggregate from node itself}} \right)$$

Nonlinearity (activation): Adds expressiveness

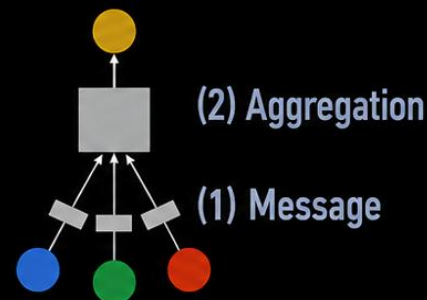
Function	Formula	When to Use	Shape
 ReLU Rectified Linear Unit	$\text{ReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0)$ Outputs 0 for negative values, keeps positive values unchanged.	 Most commonly used	
 Sigmoid	$\sigma(\mathbf{x}_i) = \frac{1}{1 + e^{-x_i}}$ Maps any real value to (0, 1). Smooth and bounded.	 Use when you want to restrict the range of your embeddings	
 PReLU Parametric ReLU	$\text{PReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0) + a_i \min(\mathbf{x}_i, 0)$ Allows a small, learnable slope a_i for negative values.	 Empirically performs better than ReLU	

Message + Aggregation + Nonlinearity

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

How to write this as Message + Aggregation?

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

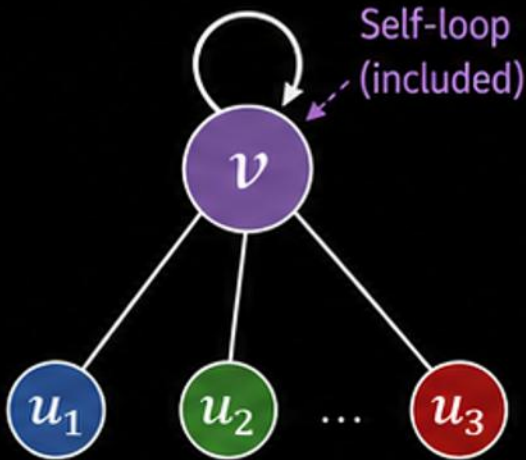


Comparative Analysis of Representative GNN Models

Recap: Graph Convolutional Networks 1/2

1 Neighbors & Self-Loop

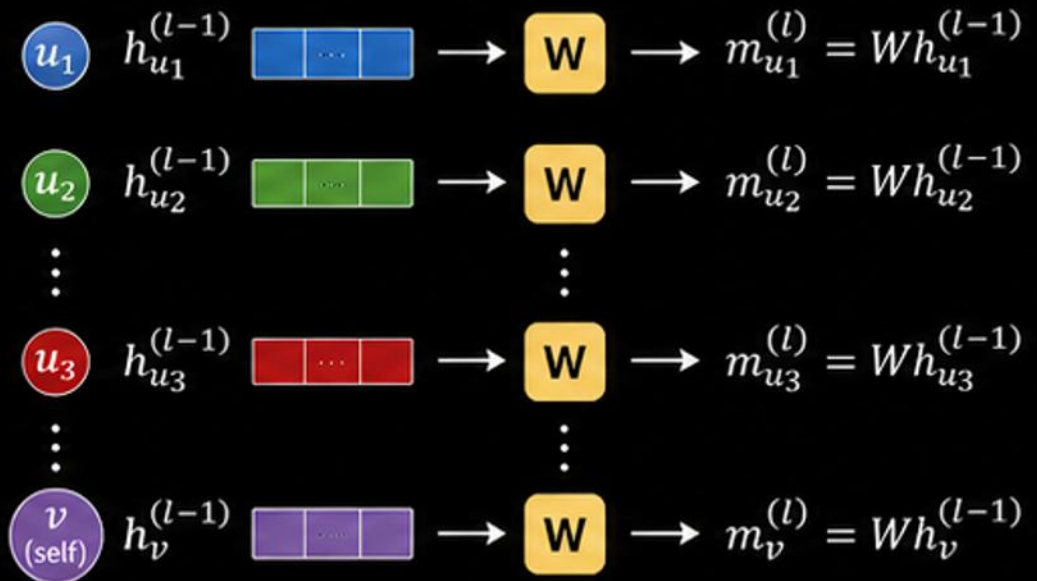
Node v collects messages from its neighbors and itself.



$N(v) \cup \{v\}$
Neighbors of v plus itself

2 Message Computation (Linear Transform)

Each neighbor (and itself) transforms its feature using the same shared weight W .

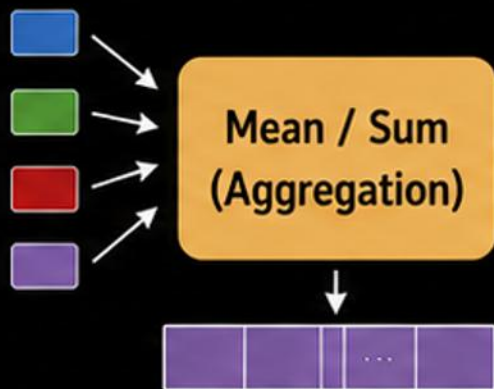


Same linear transform W applied to all (neighbors and self).

Recap: Graph Convolutional Networks 2/2

3 Aggregation (Mean/Sum)

Aggregate all messages (including self-message) using mean (or sum).

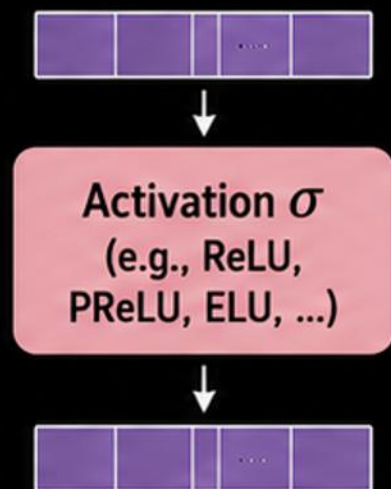


$$a_v^{(l)} = \text{AGG}\left(\left\{m_u^{(l)}, \forall u \in N(v) \cup \{v\}\right\}\right)$$

Include self-message via self-loop.

4 Activation (Non-linearity)

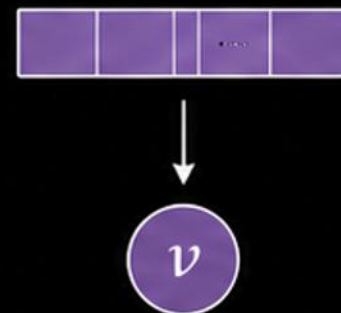
Apply non-linear activation function $\sigma(\cdot)$.



$$h_v^{(l)} = \sigma\left(a_v^{(l)}\right)$$

5 Updated Embedding

The result is the new embedding of node v at layer l .



$h_v^{(l)}$: new embedding of node v

GraphSAGE (1/4)

GraphSAGE (Graph Sample And aggreGate) <https://arxiv.org/abs/1706.02216>

Introduced in the paper "Inductive Representation Learning on Large Graphs" (NeurIPS 2017), is a graph neural network (GNN) model designed to efficiently learn node embeddings on large-scale graphs.

William Hamilton, Rex Ying, Jure Leskovec
CS, Stanford University, CA, USA

Feature	GCN	GraphSAGE
Self Feature	Self-loop	Concatenate
Aggregator	Mean	Mean / Pool / LSTM
Learning	Transductive	Inductive
Normalization	Degree Normalization	Optional L2 Normalization
Scalability	Moderate	High
Main Strength	Simple & Effective	Flexible & Scalable

GraphSAGE (2/4) - Embedding

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

Two-stage Aggregation

Stage 1: Aggregate from node neighbors

$$h_{N(v)}^{l-1} \leftarrow \text{AGG}(\{h_u^{l-1}, \forall u \in N(v)\})$$

Stage 2: Combine with the node itself

$$\text{CONCAT}(h_v^h, h_{N(v)}^{l-1})$$

Message is computed with $\text{AGG}(\cdot)$

- GCN: The node itself is included in the aggregation.
- GraphSAGE: The node itself is concatenated with the aggregated neighbor representation.

GraphSAGE (3/4) - Aggregation Approaches

Mean Aggregator

$$AGG = \sum_{u \in N(v)} \frac{h_u^{h-1}}{|N(v)|}$$

- Very simple
- Similar to GCN aggregation
- No complex transformation

Pooling Aggregator

Can apply $Max(\cdot)$ instead

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

- More expressive than the Mean aggregator.
- Uses a nonlinear transformation to extract important patterns.
- Remains permutation-invariant by applying Mean or Max aggregation.

LSTM Aggregator

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

- Learns complex interactions among neighboring nodes.
- However, LSTM is sensitive to the input order and is not inherently permutation-invariant.
- To reduce this issue, the neighbors are randomly shuffled before being processed by the LSTM.

GraphSAGE (4/4) - L2 Normalization

Why normalization?

- Embedding magnitudes may vary across layers.
- L2 normalization keeps all embeddings at the same scale.
- This helps stabilize training and may improve performance.

$$h_v^l = \frac{h_v^l}{\|h_v^l\|_2}, \quad \text{where} \quad \|\cdot\|_2 = \sqrt{\sum_i u_i^2}$$

What does it help?

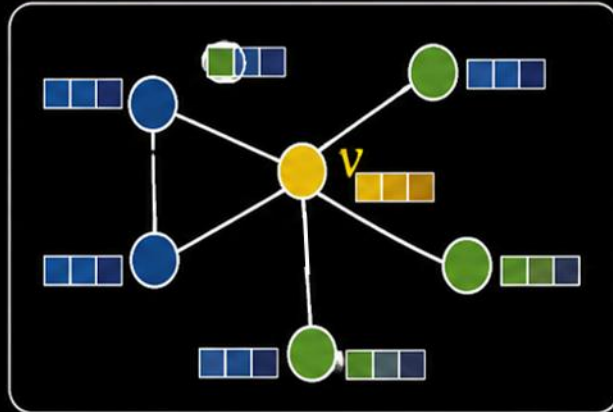
- Optionally apply L2 normalization after each layer.
- It rescales every embedding vector to have unit length.
- This stabilizes training by keeping all embeddings on the same scale.
- The vector direction is preserved, while only its magnitude is normalized.
- Whether it improves performance depends on the dataset.

Why GraphSAGE is Inductive?

GraphSAGE learns a **parameterized aggregation function** that generates **embeddings** based on node features & local neighbors

Once learned, the same function can embed **unseen nodes** or **even unseen graphs**.

1. Training Graph (seen during training)

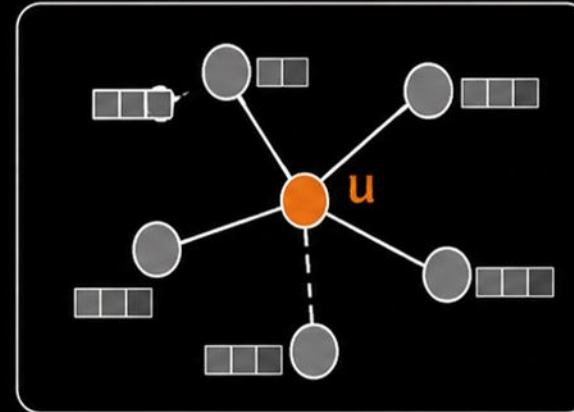


- Training nodes (with labels)
- Training nodes (context)
- Target node v

GraphSAGE
Aggregator
(learned)

Learn parameters
of the aggregator

2. New / Unseen Graph (not seen during training)



- Unseen nodes
- New node u (no label)
- Node features (available)

Generate embedding
for new node u



Because the aggregator is defined on features and local neighborhoods, GraphSAGE can **generalize to new nodes and new graphs**.

Graph Attention Networks (GAT)

GAT (Graph Attention Networks) <https://arxiv.org/abs/1710.10903>

Introduced in the paper "Graph Attention Networks" (ICLR 2018), GAT is a graph neural network (GNN) model that **learns the importance of neighboring nodes** using an **attention mechanism**.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio

(Cambridge Univ.; Centre de Visio per Computador, UAB; Montreal Institute for Learning Algorithms)

Feature	GCN	GraphSAGE	GAT
Self Feature	Self-loop	Concatenate	Self-loop
Neighbor Weight	Fixed	Fixed	Learnable Attention
Aggregator	Mean	Mean/Pool/LSTM	Attention-weighted Sum
Learning	Transductive	Inductive	Transductive
Normalization	Degree Normalization	Optional L2 Normalization	Softmax Attention
Scalability	Moderate	High	Moderate
Main Strength	Simple & Effective	Flexible & Scalable	Adaptive Neighbor Importance

Graph Attention Networks (GAT)

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

GCN & GraphSAGE

$$\alpha_{uv} = \frac{1}{|N(v)|}$$

The weighting factor (importance) of node u 's message to node v

Defined explicitly based on the structural properties of the graph (Node Degree)

All neighbors $u \in N(v)$: **equally important** to node v

→ This is problematic

Graph Attention Networks (GAT)

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

Attention Weights

Not all node's neighbors are equally important!

Attention:

α focuses on the important parts of the input data & fades out the rest.

Idea:

NN should devote more small computing but important part of data

- Which part of data is more important depends on context
- The importance is learned through training

Graph Attention Networks (GAT)

Research Question:

Can we do better than simple neighborhood aggregations?

Goal:

Specify arbitrary importance to different neighbors of each node in the graph

Idea:

Compute embedding h_v^l of each node in the graph following **attention strategy**!

- Nodes attend over their neighborhoods' message
- Implicitly specifying different weights to different nodes in a neighborhood

Attention Mechanism in Graphs (1/2)

Use an attention mechanism a to compute the attention weight α_{vu}

For each pair of connected nodes u and v ,
compute an attention score:

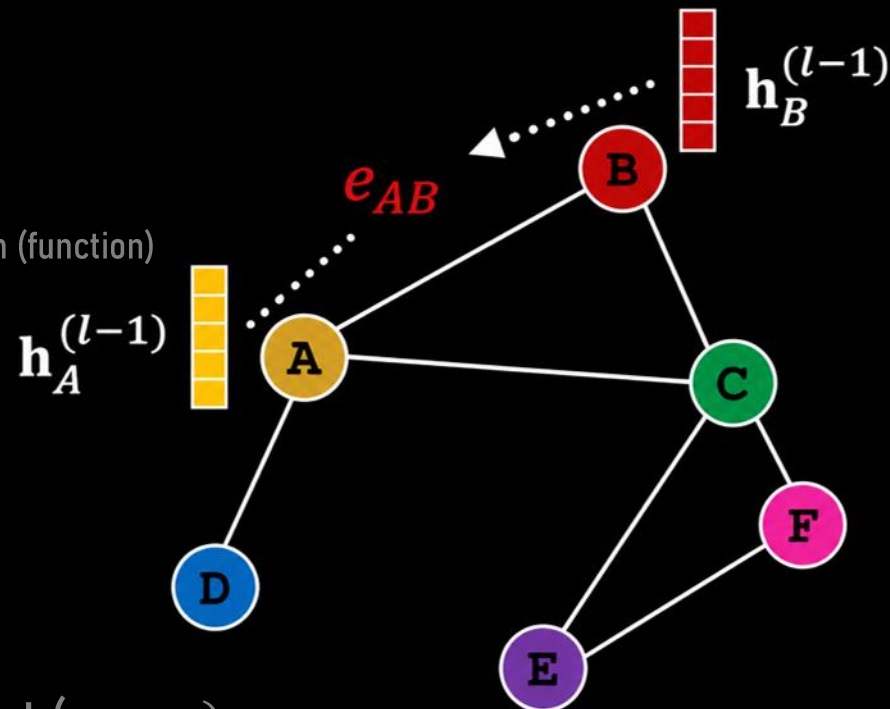
$$e_{vu} = a(W^l h_u^{l-1}, W^l h_v^{l-1})$$

$a(\cdot, \cdot)$: attention mechanism (function)

e_{uv} measures how important
node u 's message to node v

Note:

- e_{uv} is scalar score,
- u : source (neighbor), v : target ($u \rightarrow v$)



Attention Mechanism in Graphs (2/2)

Transform a node feature to message:

F' is the output dimension

$$\mathbf{m}_u^l = W^l h_u^{l-1}, \text{ where } m_u^l \in \mathbb{R}^{F'}$$

$$\mathbf{m}_v^l = W^l h_v^{l-1}, \text{ where } m_v^l \in \mathbb{R}^{F'}$$

Compute raw attention score:

$$e_{vu} = a(m_v^l, m_u^l)$$

$$= \text{LeakyLeRU}(\mathbf{a}^T [\mathbf{m}_v^l || \mathbf{m}_u^l]), \text{ where } \mathbf{a} \in \mathbb{R}^{2 \times F'}$$

Where $||$ denote concatenate operator,
 $\mathbf{a}$ is learnable attention parameter vector

Two node features



Concatenate



Dot product with a



One score (scalar)

Attention Weight & Weighted Aggregation

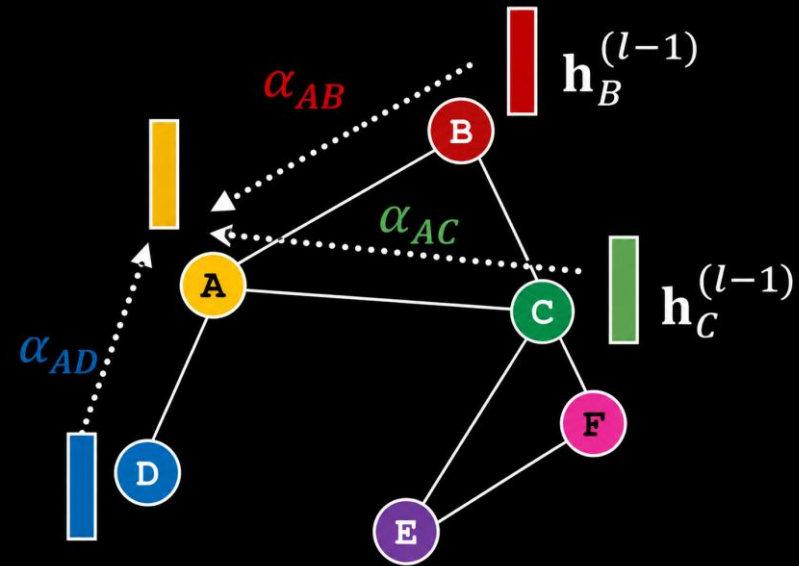
Raw score $\rightarrow$ Attention weight

$$e_{vu} = \text{LeakyLeRU}(\mathbf{a}^T [\mathbf{m}_v^l || \mathbf{m}_u^l]) , \text{ where } \mathbf{a} \in \mathbb{R}^{2 \times F'}$$

Applying softmax so that $\sum_{u \in N(v)} \alpha_{vu} = 1$

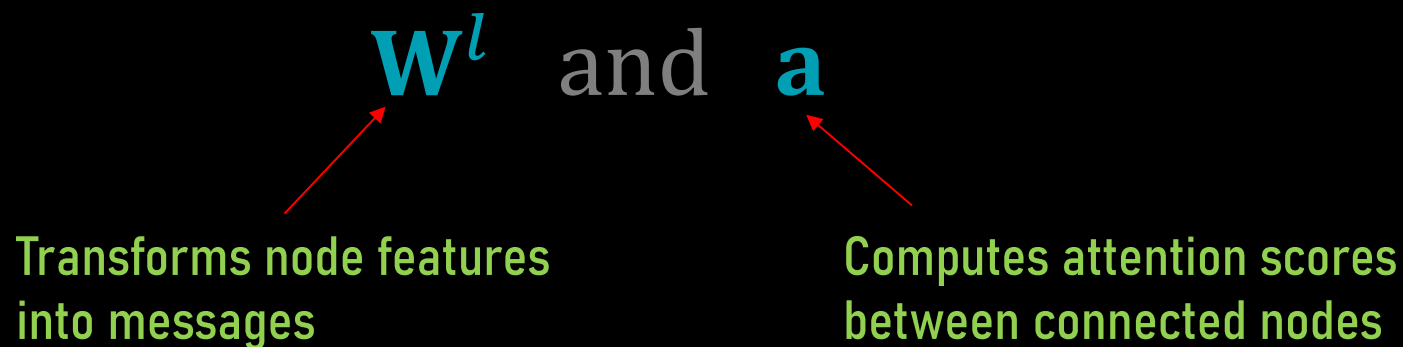
$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

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

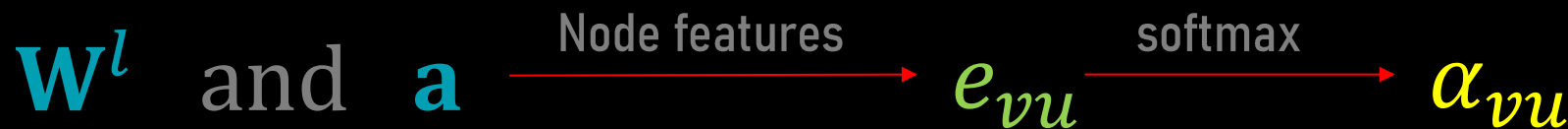


How Are Attention Parameters Learned?

GAT has two main types of learnable parameters:



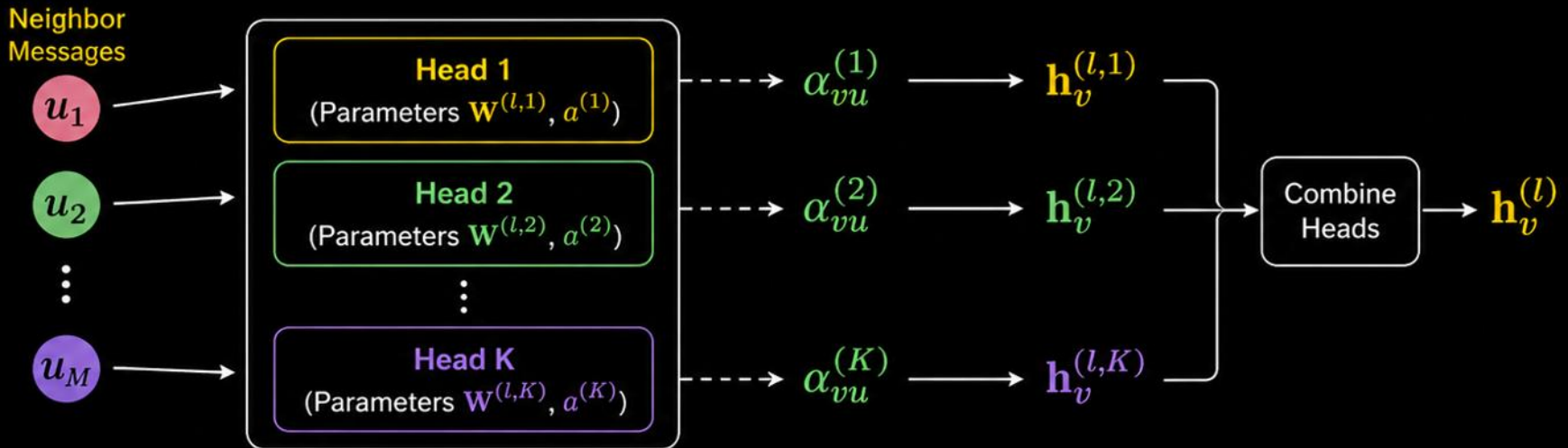
Both parameters are updated together through **backpropagation** to minimize the final task loss



Multi-Head Attention in GAT

Use multiple attention heads to learn different neighbor importance.

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



- Each head has its own **learnable parameters** and attention weights.
- Multiple head outputs are **combined** to produce the final representation.

Why Use Attention in GNNs?

Different Importance for Different Neighbors

- Learns which neighbors are more important for each target node.

$$\alpha_{vu_1} \neq \alpha_{vu_2} \neq \alpha_{vu_3}$$

Computationally Efficient

- Attention scores for different edges can be computed in parallel.

Localized

- Attention is computed only over the local neighborhood $N(v)$.

Inductive

- The same attention mechanism can be applied to unseen nodes and graphs.



수고하셨습니다 ..^^..