

Graph Convolutional Networks

소프트웨어 끈대 강의

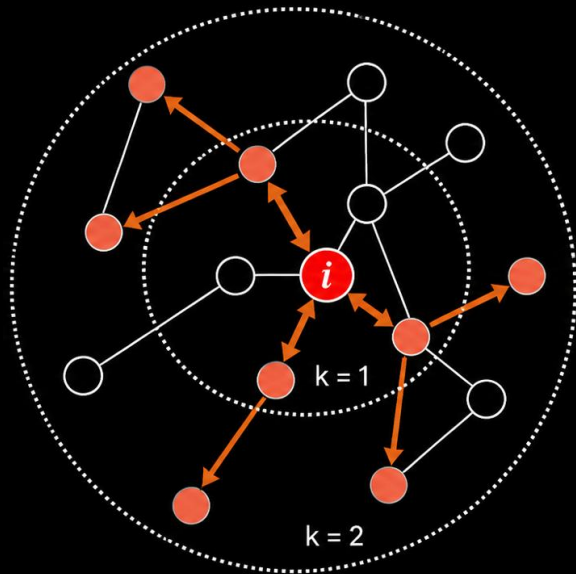
노기섭 교수

(kafa46@gmail.com)

- GCN Formulation
- Training GCN
 - Supervised
 - Unsupervised
- Inductive Capabilities
 - New graph
 - New Node
- GCN vs. CNN

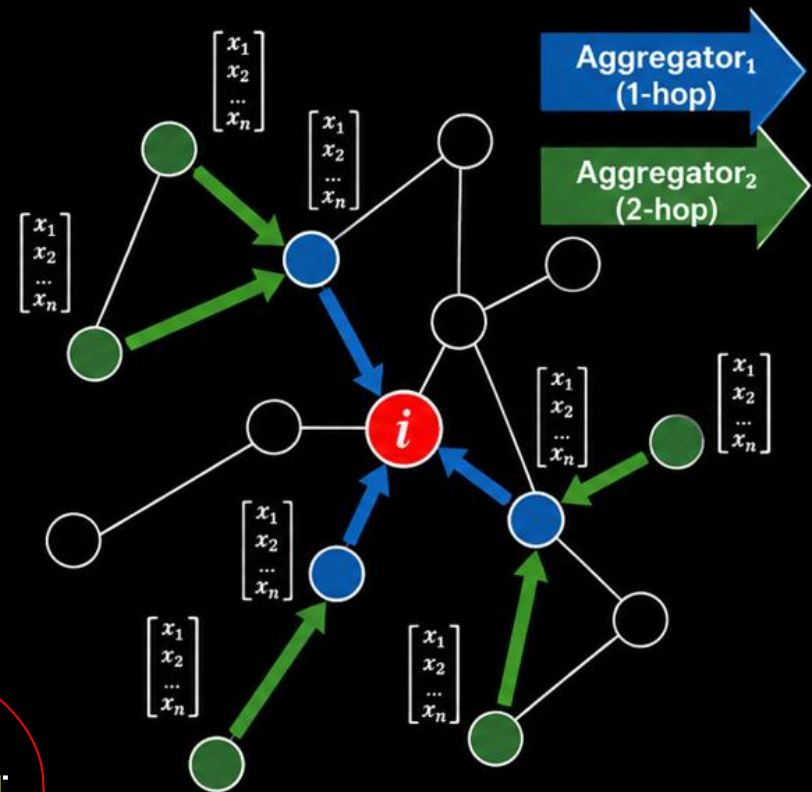
GCN Formulation

Node's neighborhood defines a computation graph



Determine node computation graph

Message Passing:
Propagating Information
Across the Graph

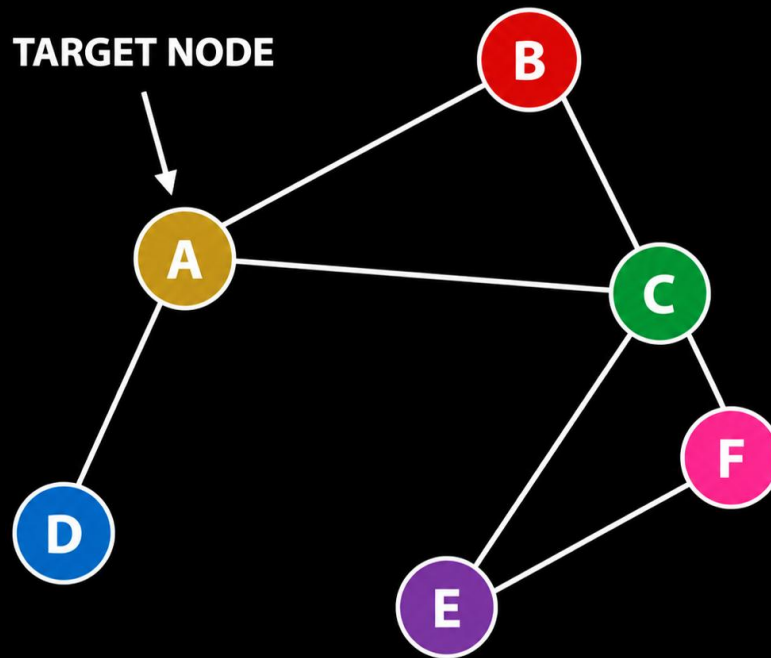


Propagate & transform
information

Aggregate Neighbors

Generate node embeddings based on local network neighborhoods

노드 임베딩은 그래프 전체를 보는 것이 아니라, 주변(Local Neighborhood) 정보만 이용하여 계산



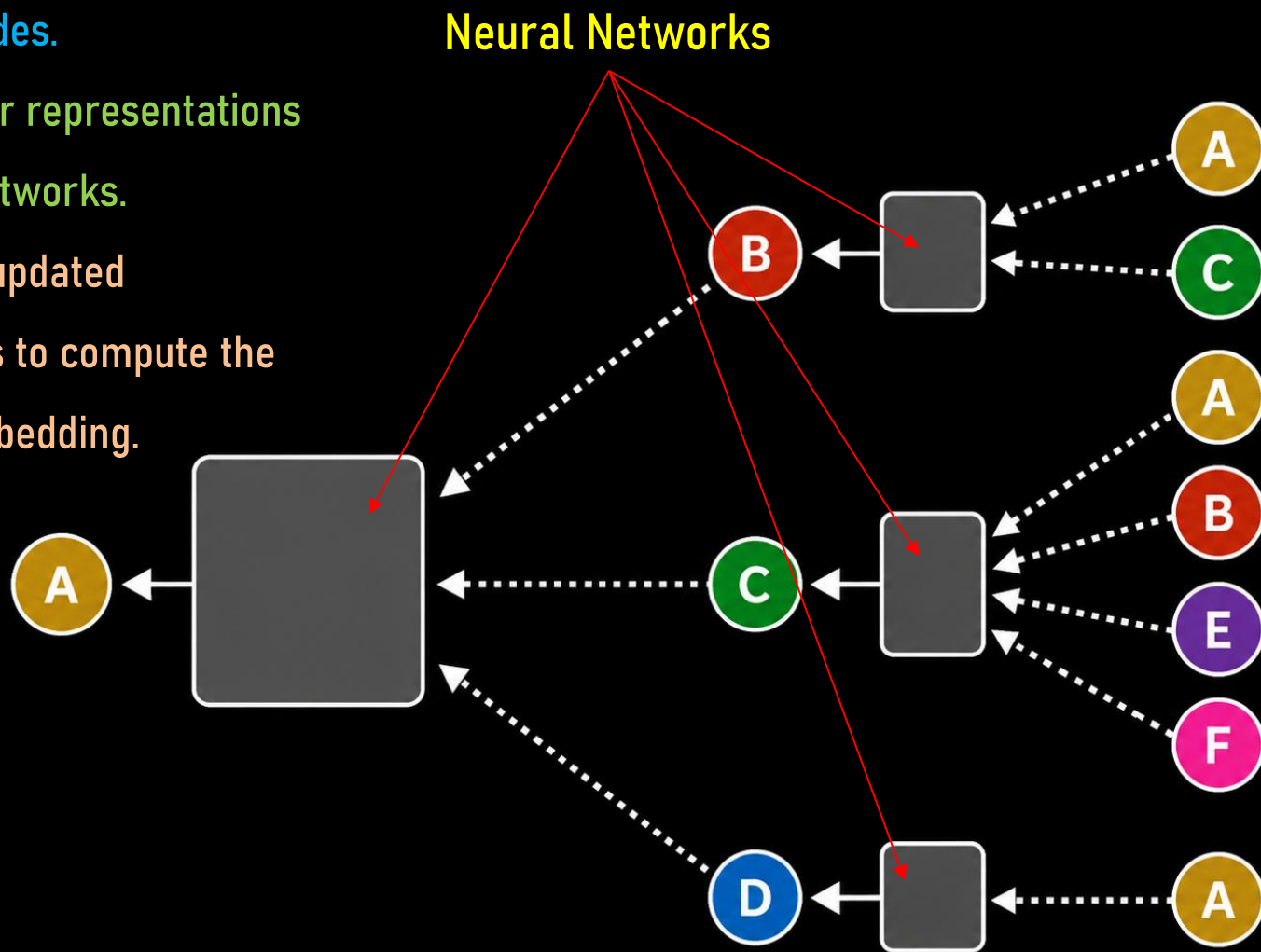
Input Graph

Local Neighborhood

- 1-hop Neighbors
- 2-hop Neighbors

How Are Node Embeddings Computed?

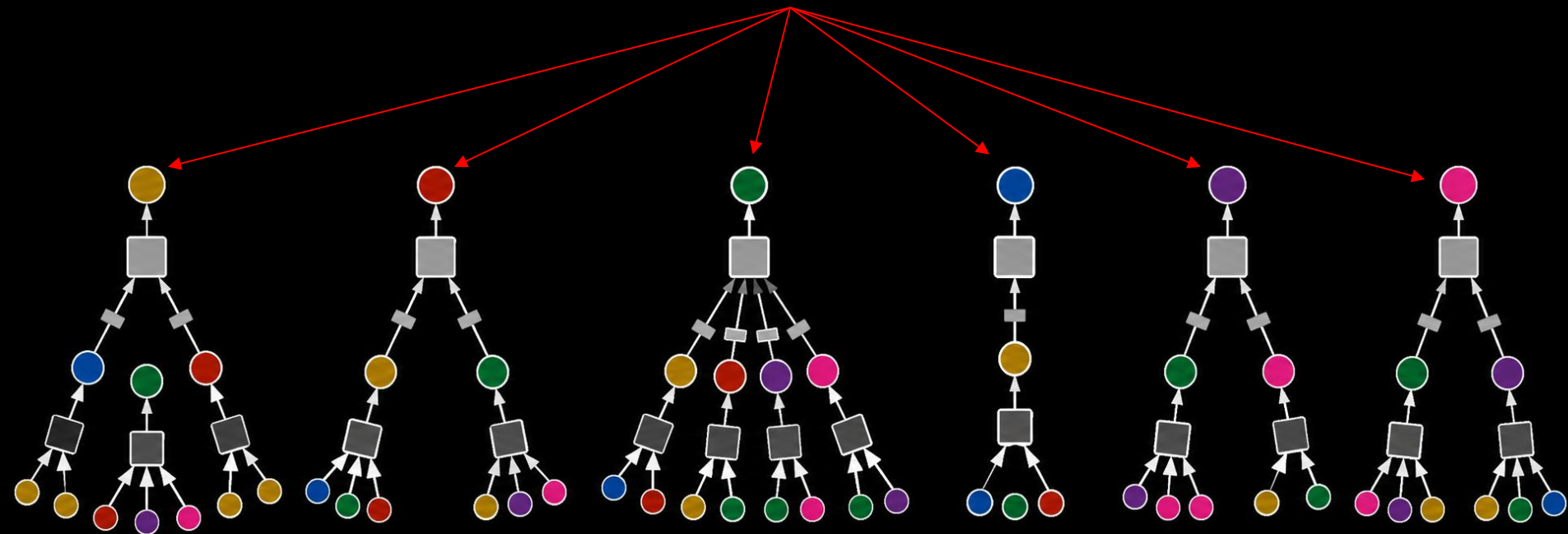
- Collect messages from neighboring nodes.
- Update neighbor representations using neural networks.
- Aggregate the updated representations to compute the target node embedding.



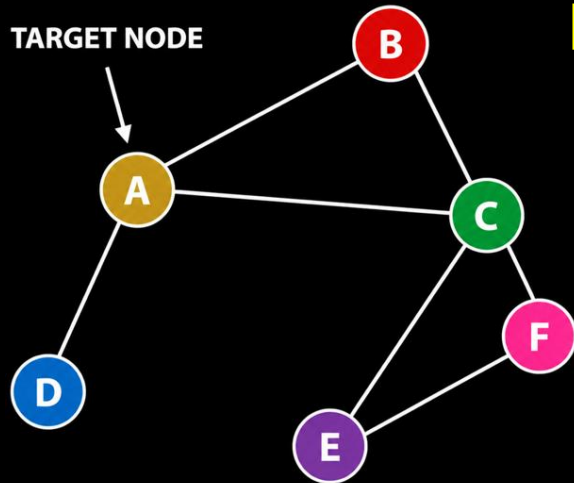
Each Node Defines Its Own Computation Graph

- Each node builds its own computation graph from its local neighborhood.
- The computation graph determines how messages are propagated.
- Node embeddings are learned through neighborhood aggregation.

Every node defines a computation graph based on its neighborhood!



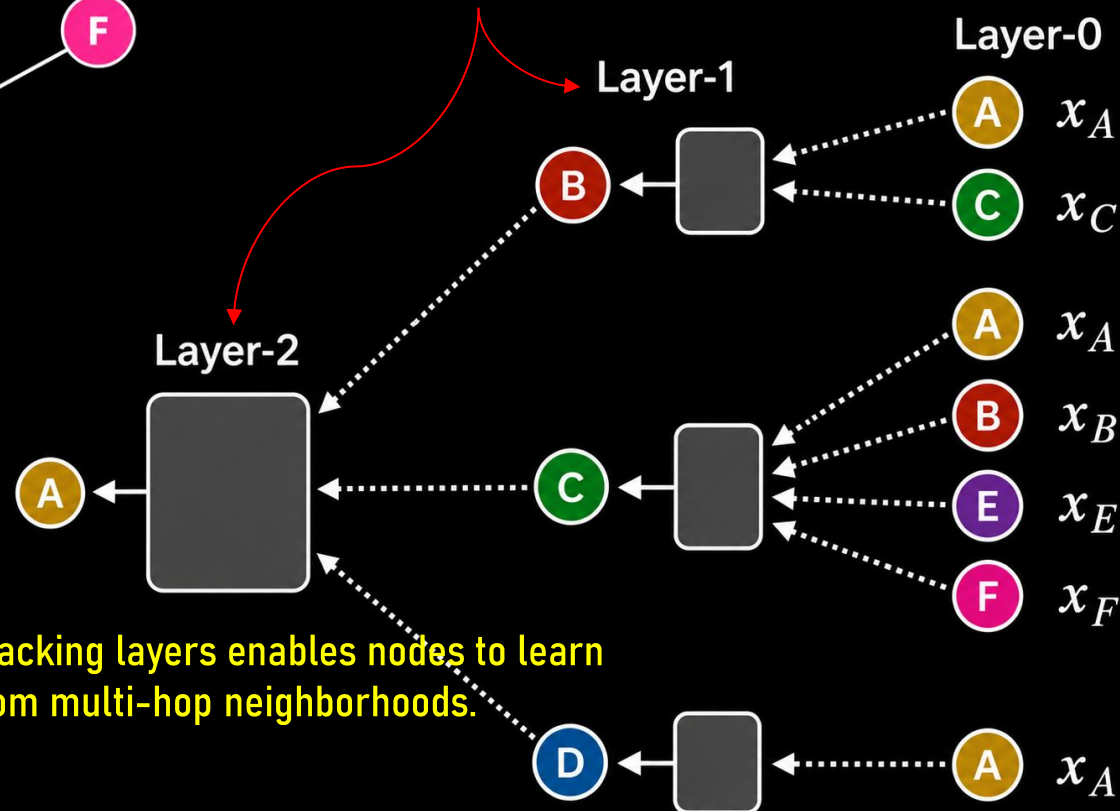
Deep GNNs: Expanding the Receptive Field



Model can be of arbitrary depth!

Each GNN layer aggregates information from one-hop neighbors.

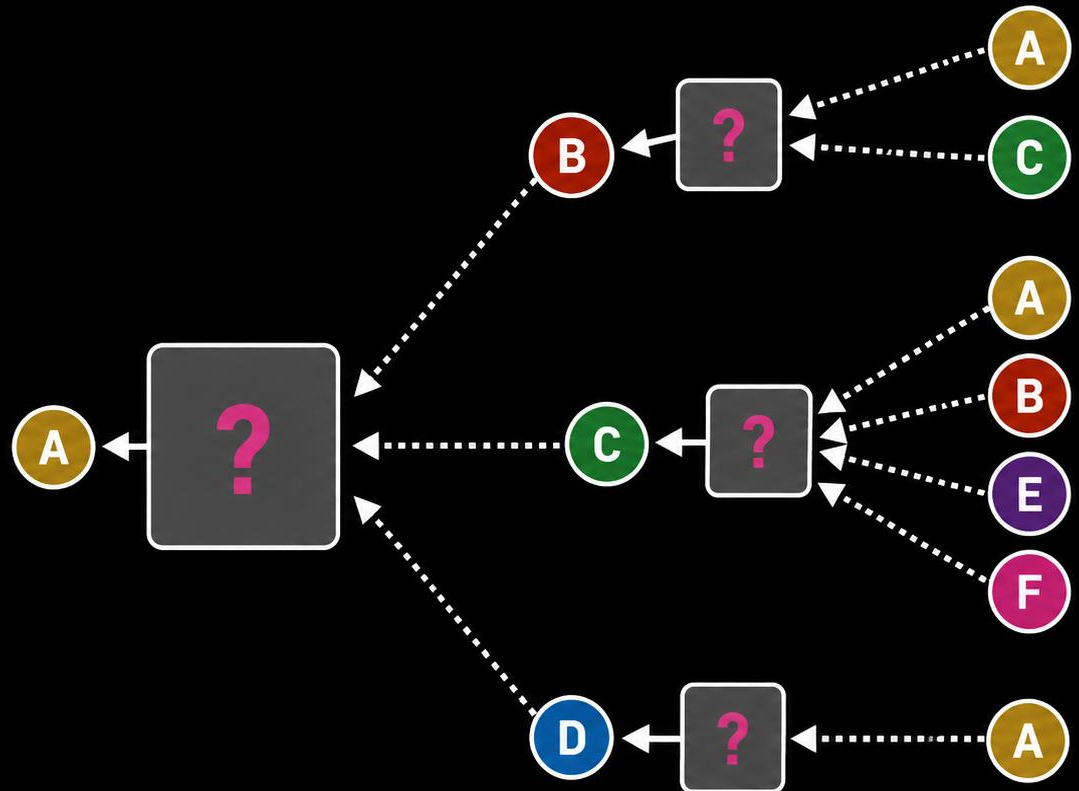
Layer 0 uses the original node features as input.



What's Inside the Aggregation Box?

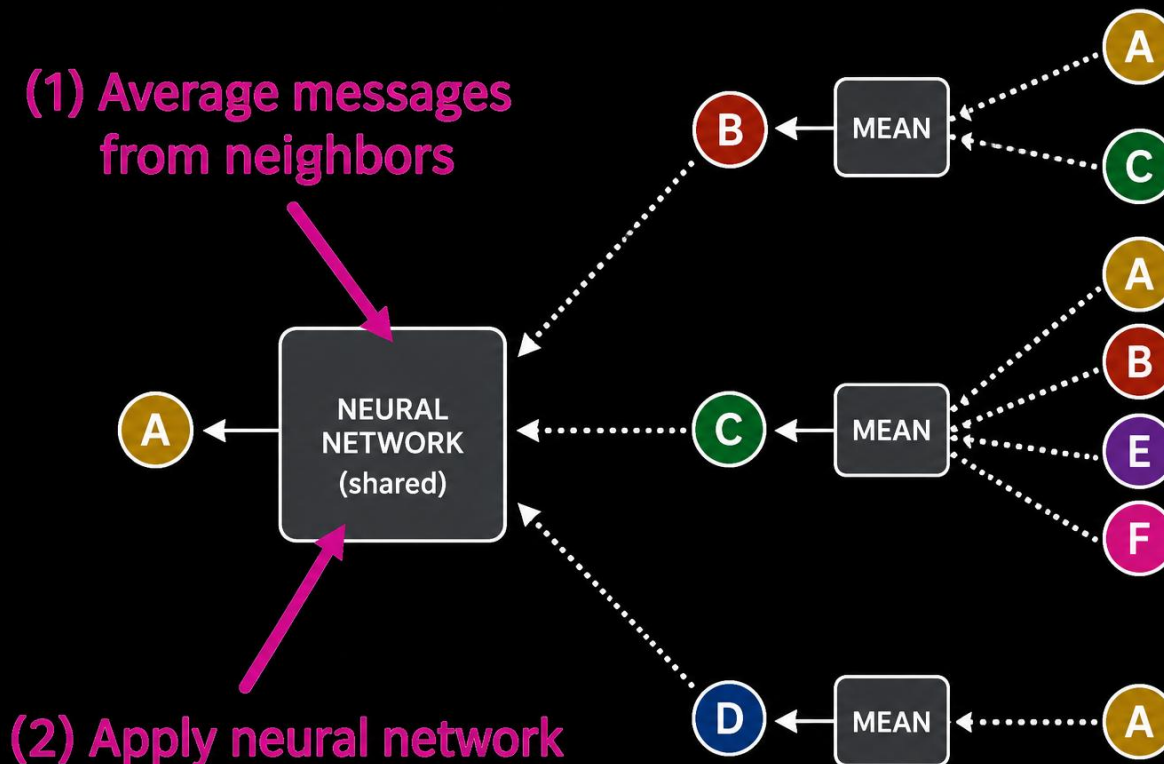
- Neighborhood aggregation is the core operation of GNNs.
- Different GNN models use different aggregation functions.
- Possible Algorithms (Architectures)
 - GCN
 - GraphSAGE
 - GAT
 - GIN

**What is
in the box?**



Mean Aggregation in Graph Convolutional Networks (GCNs)

Aggregate neighbor features using mean pooling,
then transform the aggregated representation
with a shared neural network to produce a new node embedding.



Mathematical Formulation of GCN

Basic approach: Average neighbor messages and apply a neural network

$h_v^0 = x_v$ Initial 0-th layer embeddings are equal to node features

$$h_v^{k+1} = \sigma \left(W_k \sum_{u \in \mathcal{N}(v)} \frac{h_u^k}{|\mathcal{N}(v)|} + B_k h_v^k \right), \forall k \in \{0, 1, \dots, K-1\}$$

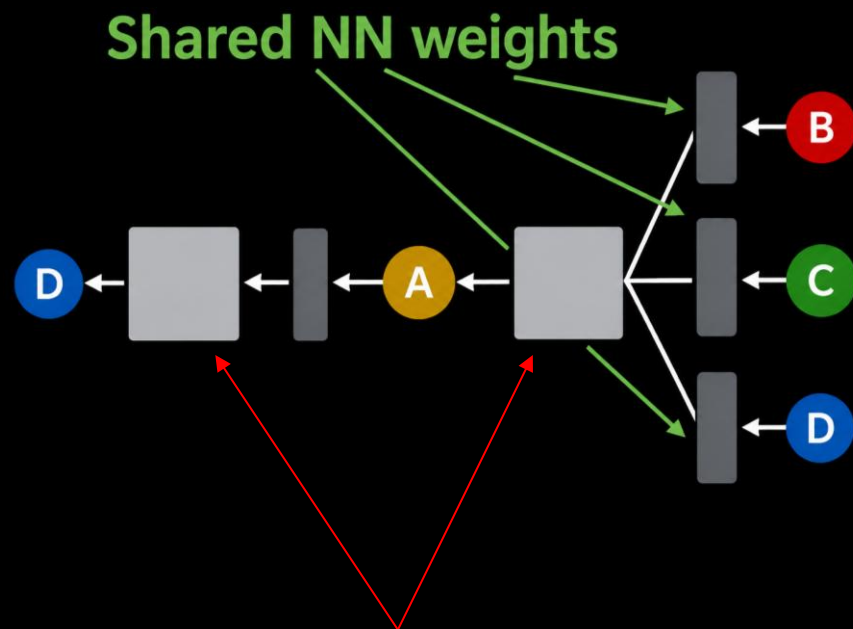
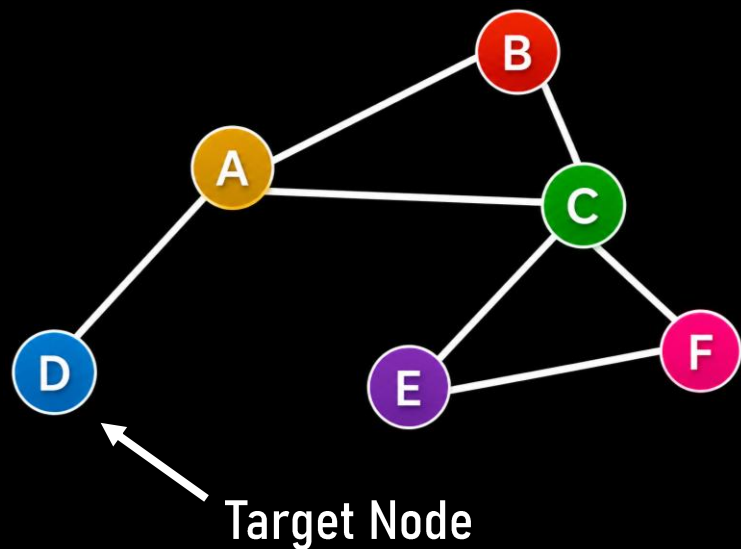
Annotations:

- Non-linearity** (yellow arrow pointing to σ)
- Average of neighbor's pervious layer embeddings** (green arrow pointing to $\sum_{u \in \mathcal{N}(v)} \frac{h_u^k}{|\mathcal{N}(v)|}$)
- Embedding of v at layer k** (red arrow pointing to $B_k h_v^k$)

$z_v = h_v^K$ Embedding after K layers of neighborhood aggregation

Permutation Invariance in GCN

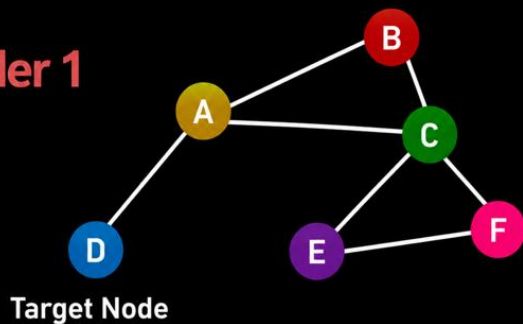
Given a node, the GCN that computes its embedding is permutation invariant.



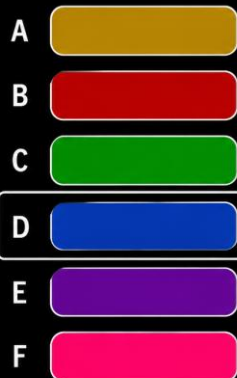
Average of neighbor's previous layer embeddings - Permutation invariant

Permutation Equivariance in GCN

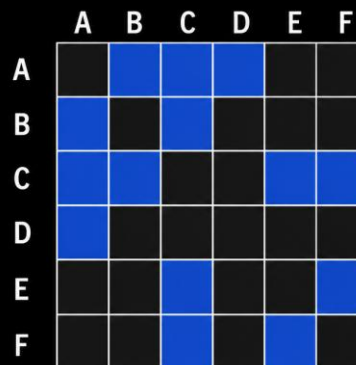
Order 1



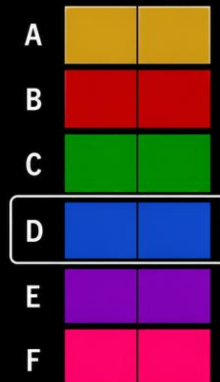
Node feature X_1



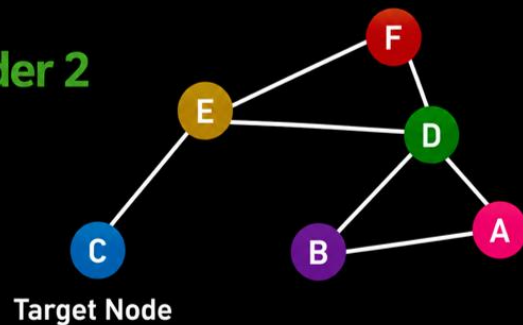
Adjacency matrix A_1



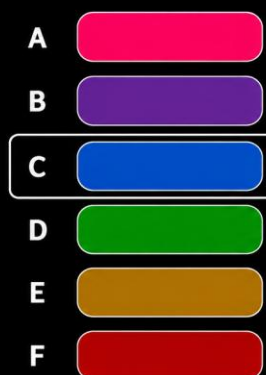
Embeddings H_1



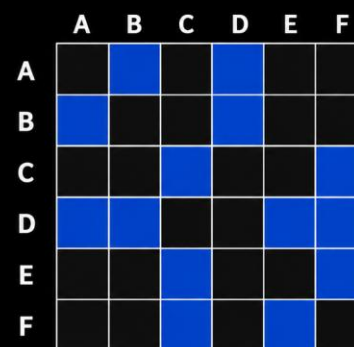
Order 2



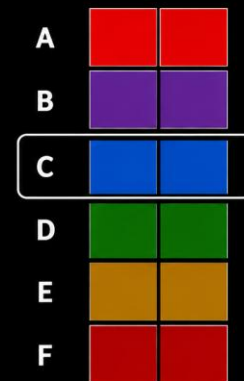
Node feature X_2



Adjacency matrix A_2



Embeddings H_2



Blue = connected (1) Black = not connected (0)

GCN Computations are all Permutation Equivariant

Considering all nodes in a graph,
GCN computation is **permutation equivariant!**

- The rows of input node features and output embeddings are aligned.

We know: For a fixed node identity, its embedding is unchanged under node reordering.

- After permutation, the location of a given node in the input node feature matrix is changed, and the output embedding of a given node stays the same.
- Same node identity $\rightarrow$ Same embedding

Mathematical Formulation of GCN

$h_v^0 = x_v$ Initial 0-th layer embeddings are equal to node features

Trainable Parameters (What we learn)

$$h_v^{k+1} = \sigma \left(W_k \sum_{u \in \mathcal{N}(v)} \frac{h_u^k}{|\mathcal{N}(v)|} + B_k h_v^k \right), \forall k \in \{0, 1, \dots, K-1\}$$

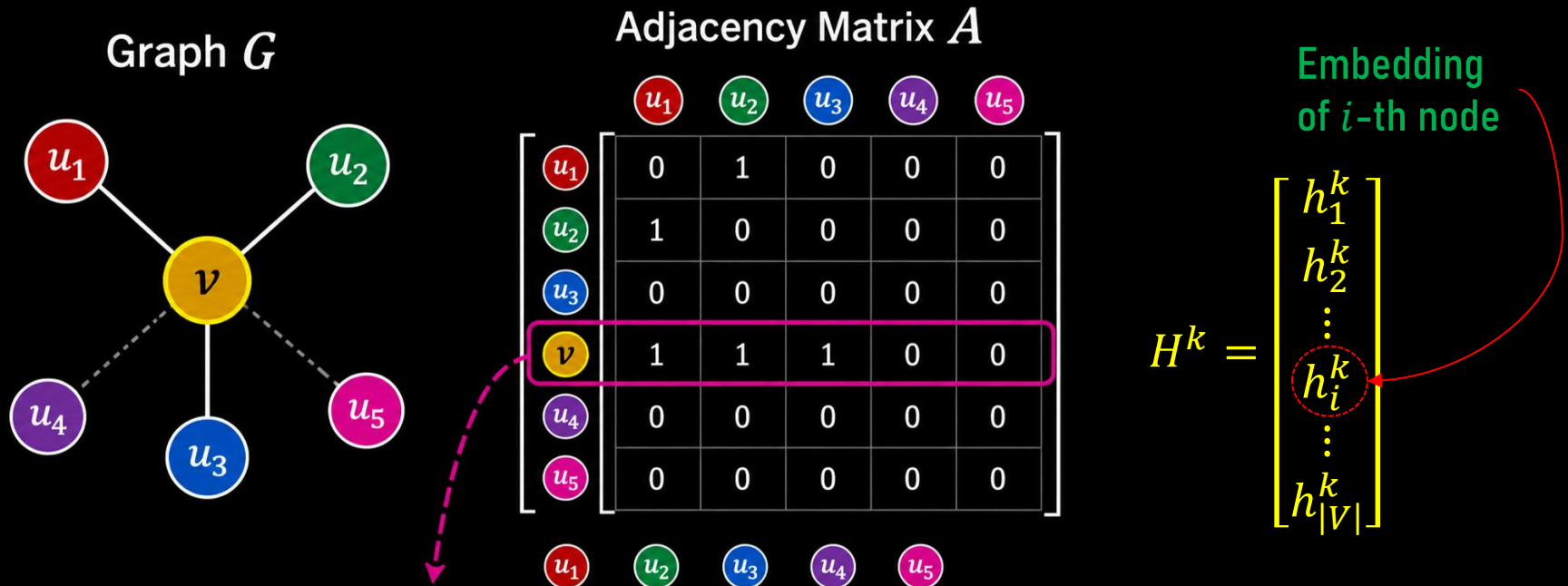
$z_v = h_v^K$ Final node embedding

h_v^k Hidden representation of node v at layer k

W_k : Weight matrix for neighborhood aggregation

B_k : Weight matrix for transforming hidden vector of self-node

Neighborhood Aggregation with the Adjacency Matrix



v -th row of A

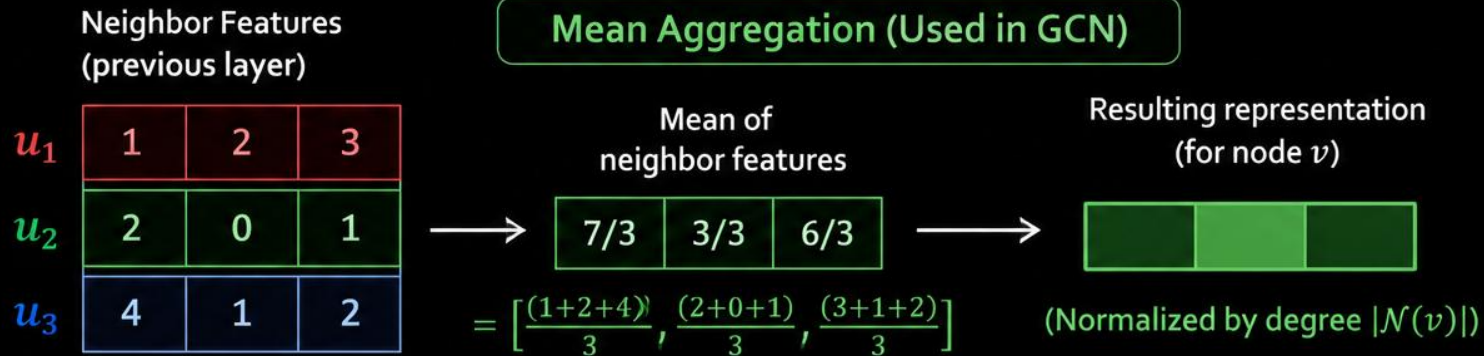
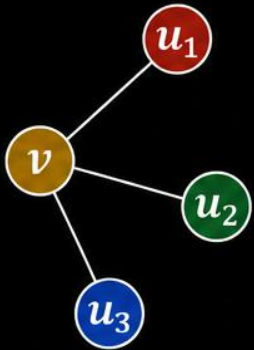
$$A_{v,:} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$A_{v,:} H^k$ sums all neighbor embeddings of node v in k -th hidden layer.

AH^k simultaneously sums the neighboring features for all nodes in k -th hidden layer.

However,
GCN uses mean aggregation
instead of sum aggregation.

Mean Aggregation with Degree Normalization



Degree Matrix

$$D = \begin{bmatrix} \deg(u_1) & 0 & \dots & 0 \\ 0 & \deg(u_2) & \dots & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & \deg(|V|) \end{bmatrix}$$

$$D^{-1} = \begin{bmatrix} \frac{1}{\deg(u_1)} & 0 & \dots & 0 \\ 0 & \frac{1}{\deg(u_2)} & \dots & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\deg(|V|)} \end{bmatrix}$$

Average aggregation in the $(k + 1)$ -th hidden layer

$$H^{k+1} = D^{-1}AH^k$$

모든 노드의 Aggregation을 하나씩 계산하지 말고,
하나의 행렬곱으로 동시에 계산하자 (병렬 연산!)

Deriving the Matrix Formulation

Aggregations can be performed efficiently by matrix operations!

Let $H^k = [h_1^k, h_2^k, \dots, h_{|V|}^k]^T$

Then $\sum_{u \in \mathcal{N}(v)} h_u^k = A_{v,:} H^k$

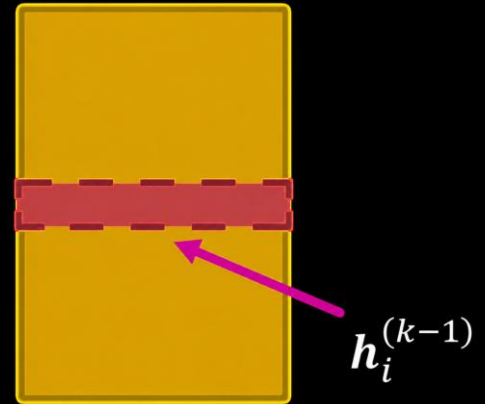
Let D be diagonal matrix, where

$$D_{v,v} = \text{Deg}(v) = |N(v)|$$

The inverse of D (D^{-1}) is also diagonal, $D_{v,v}^{-1} = \frac{1}{|N(v)|}$

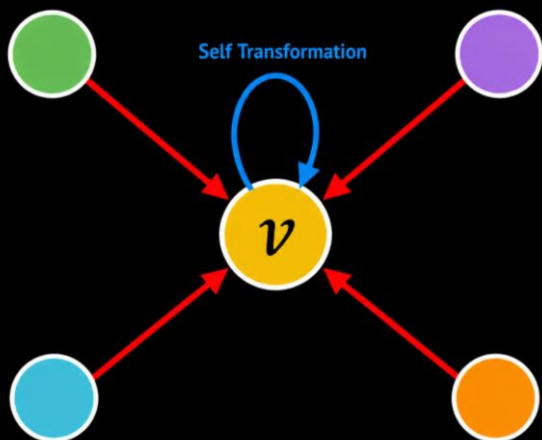
Therefore, $\sum_{u \in \mathcal{N}(v)} \frac{h_u^{k-1}}{|N(v)|} = D^{-1} A H^k$

Matrix of
hidden
embeddings
in H^{k-1}



Final GCN Update Rule

$$H^k = [h_1^k, h_2^k, \dots, h_{|V|}^k]^T$$



Red arrow: neighborhood aggregation

Blue arrow: self transformation

Update function in parallel processing
(Matrix form)

$$H^{k+1} = \sigma(\tilde{A}H^k W_k^T + H^k B_k^T)$$

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

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

Note:

Not all GNNs can be expressed in a simple matrix form, when aggregation function is complex

Training GCN

Training Approaches

Compute embedding z_v from Graph Input

$$z_v = f_{GCN}(G, X)$$

G : Graph structure (e.g., adjacency matrix)

X : Node features

→ **Unsupervised Learning: Use graph structure as supervision**

When no node label available

→ **Supervised Learning: Find parameter θ Minimizing loss $\mathcal{L}$**

$$\arg \min_{\theta} \mathcal{L}(y, f_{\theta}(z_v))$$

y : Actual value (label)

f_{θ} : Classifier

$\mathcal{L}(\cdot)$: Could be one of MSE, Cross Entropy, ...

Unsupervised Learning

Possible idea: "Similar" nodes have similar embedding

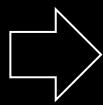
$$\arg \min_{\theta} \mathcal{L} = \sum_{z_u, z_v} CE(y_u \cdot y_v, z_u \cdot z_v)$$

$$z_u = f_{\theta}(u)$$

$y_{u,v} = 1$ when node u and v are similar

$$CE(y, \hat{y}) = - \sum_{i=1}^c y_i \log \hat{y}_i, \text{ where } \hat{y} = f_{\theta}(x)$$

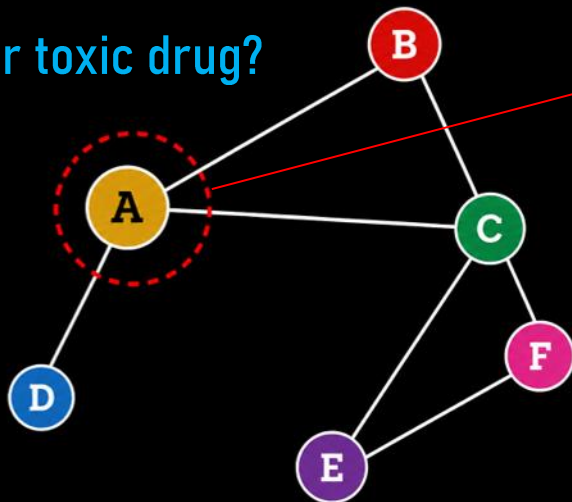
y_i and $f_{\theta}(x)_i$ are actual & predicted values of the i -th class, respectively.



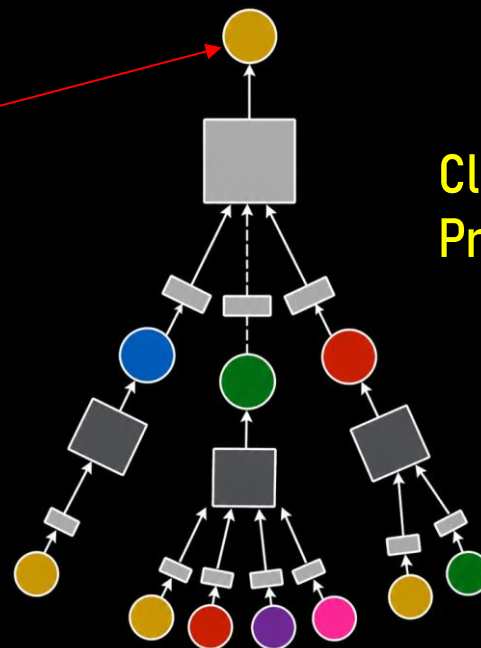
Intuition: the lower the loss,
the closer the prediction is to one-hot

Supervised Learning

Safe or toxic drug?



A drug-to-drug interaction Network



Classification Problem

→ Use BCE or CE!

$$\mathcal{L}_{BCE} = - \sum_{v \in V} (y_v \log \sigma(z_v^T) + (1 - y_v) \log(1 - \sigma(z_v^T)))$$

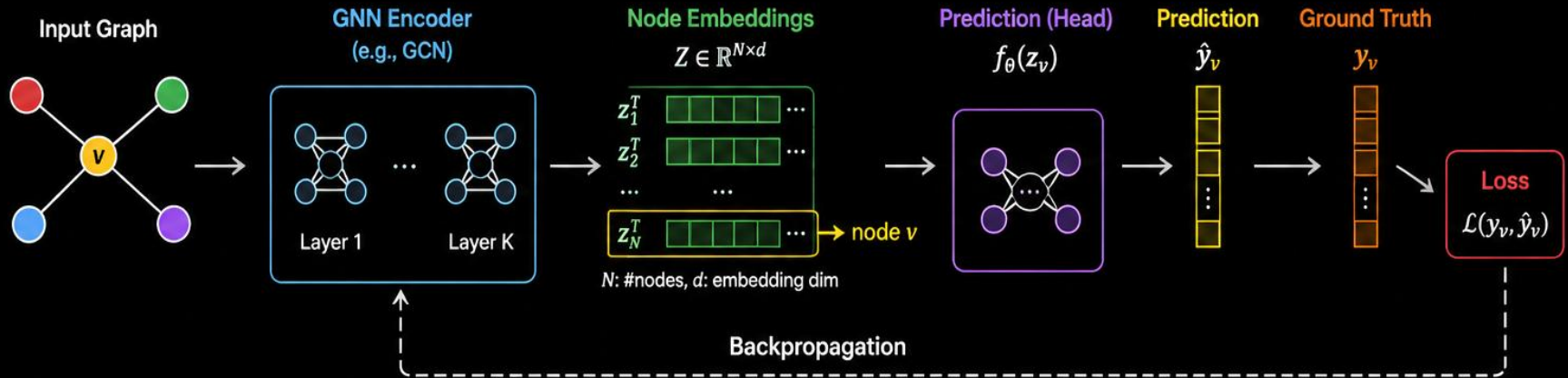
y_v : Node class label

z_v^T : Encoder output (node embedding)

Training: Supervised vs. Unsupervised

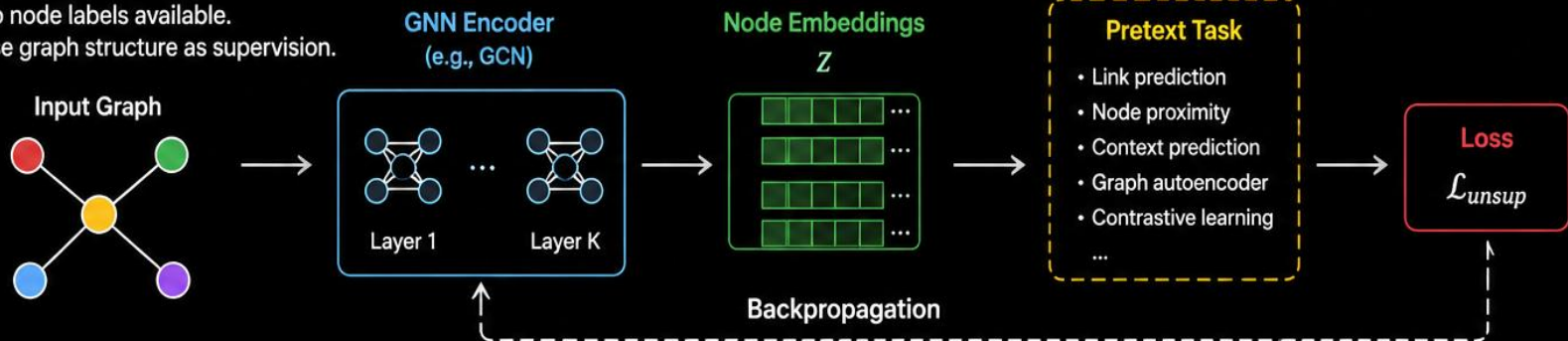
1) Supervised Setting

Minimize loss between prediction and ground truth



2) Unsupervised Setting

No node labels available.
Use graph structure as supervision.



● Node (class 1) ● Node (class 2) ● Node (class k) ● Node (class k+1) ● Node (class k+2)

■ Embedding vector ■ Prediction vector ■ Ground truth vector

Inductive Capabilities

Transductive vs. Inductive Learning

Transductive Learning

직관적 이해: 시험 범위가 동일
(시험 문제는 모두 봤지만 일부 정답을 모르는 상황)

- Training and test nodes belong to the same graph.
- Only labels are hidden for test nodes.
- The entire graph structure is available during training.

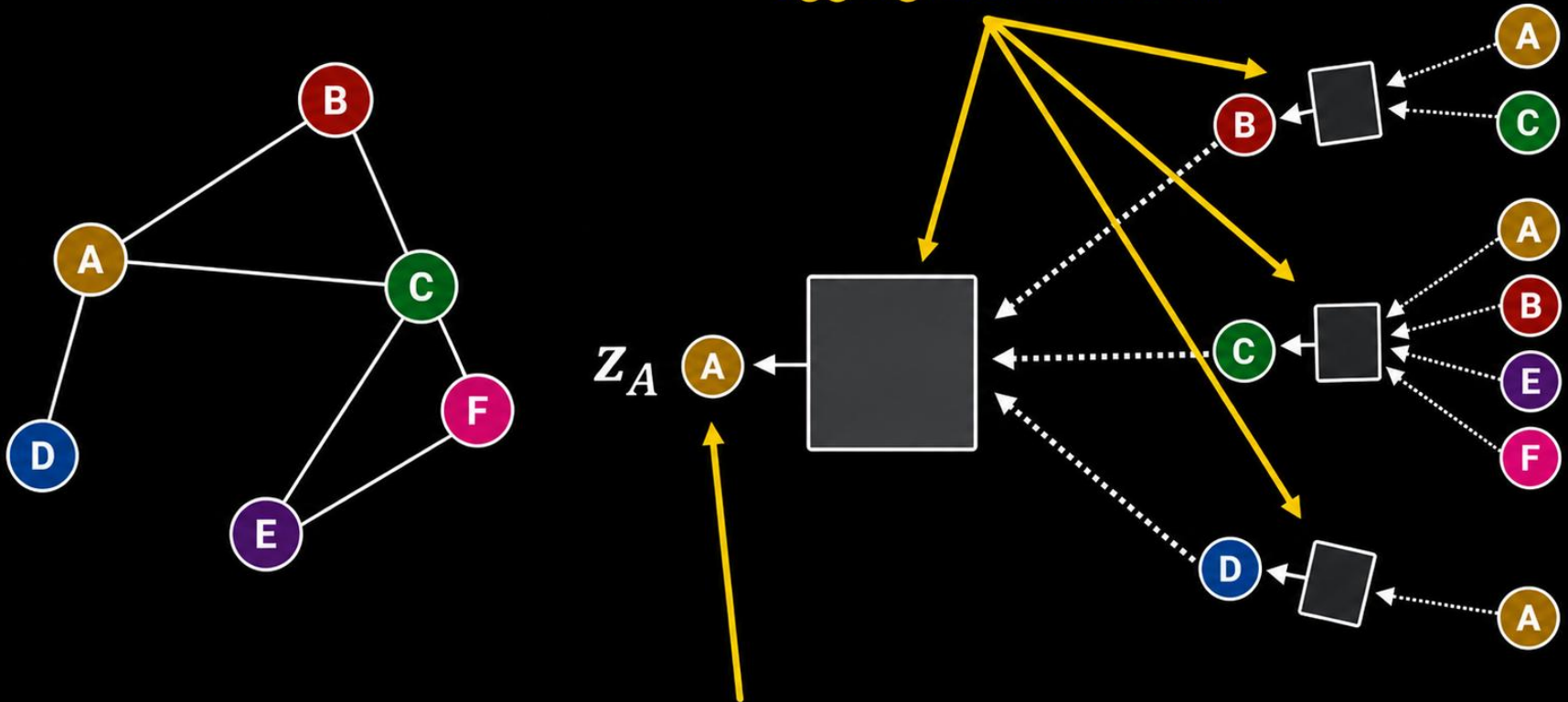
Inductive Learning

직관적 이해: 시험 범위가 계속 변함
(과거 문제로 공부하고, 시험은 당일 새로운 문제가 등장)
→ 문제 푸는 원리를 배웠기 때문에 새로운 문제도 풀 수 있음.

- Training and test nodes (or graphs) can be different.
- The model generalizes to previously unseen nodes or graphs.
- Embeddings are generated without retraining the entire model.

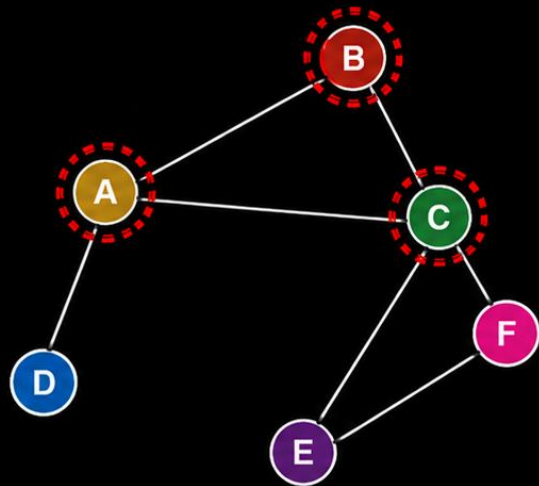
Train Step 1/3

(1) Define a neighborhood aggregation function



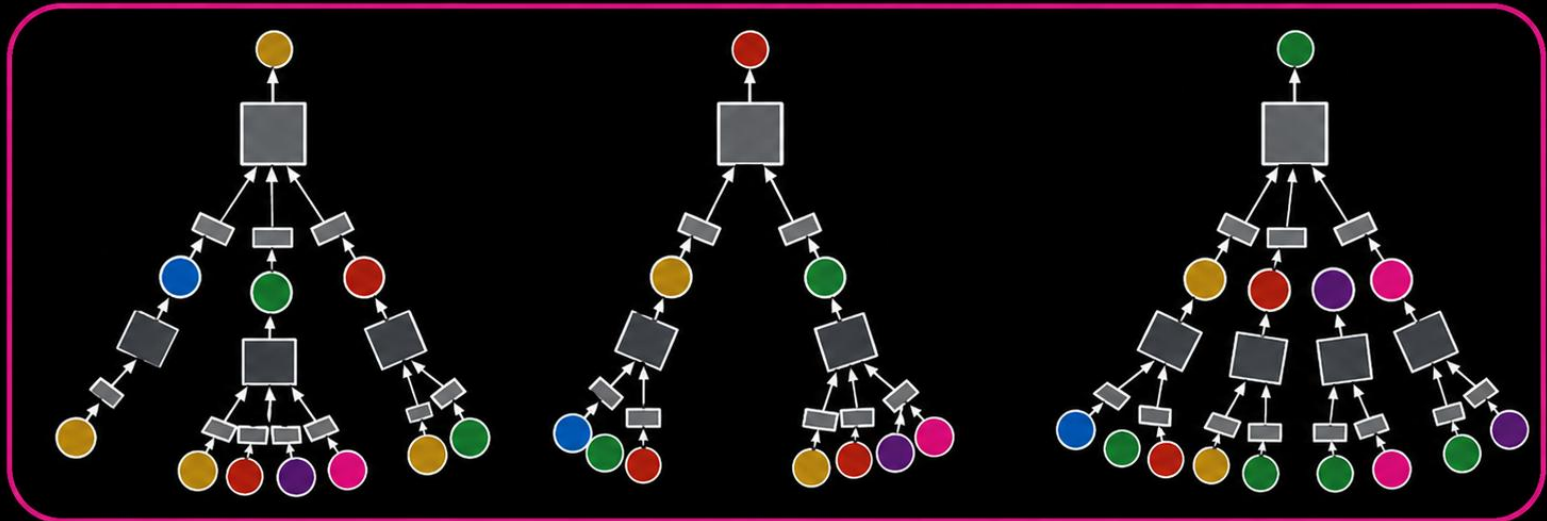
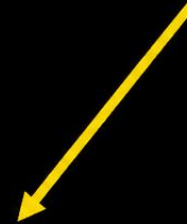
(2) Define a loss function on the embeddings

Train Step 2/3

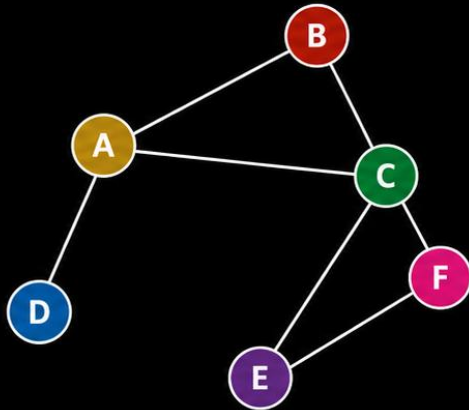


INPUT GRAPH

**(3) Train on a set of nodes, i.e.,
a batch of compute graphs**



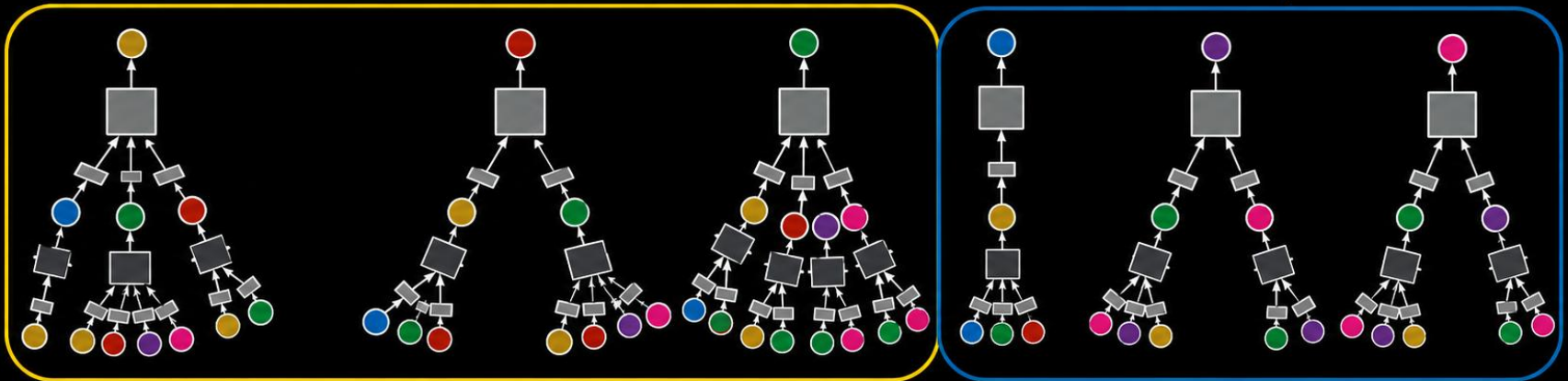
Train Step 3/3



INPUT GRAPH

**(4) Generate embeddings
for nodes as needed**

Even for nodes we never
trained on!



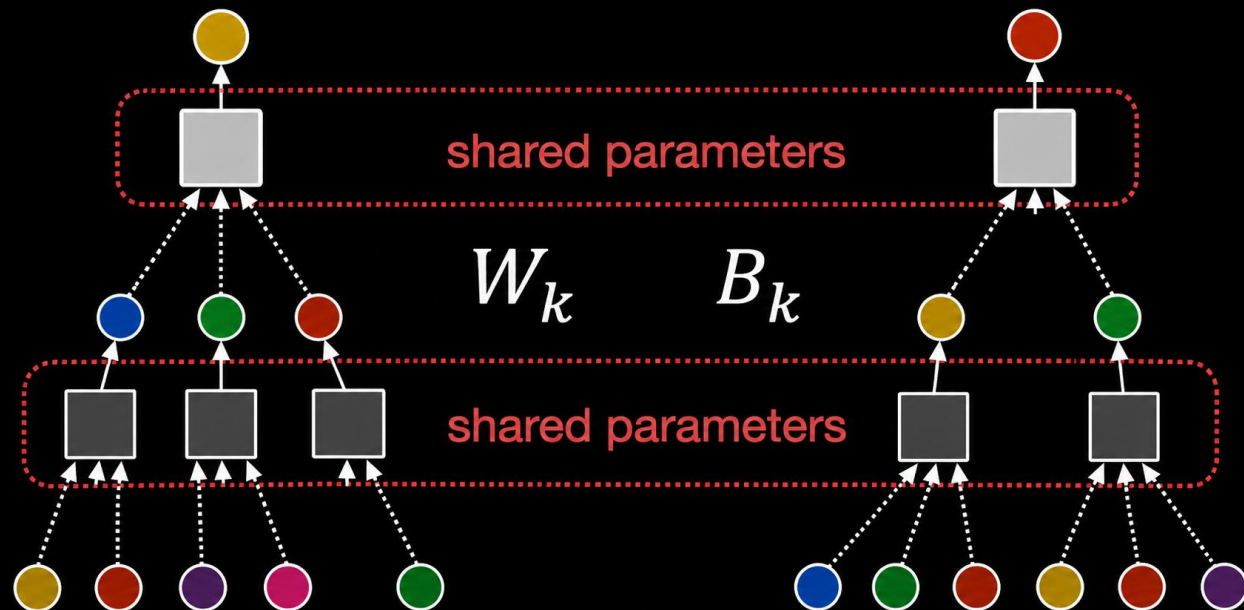
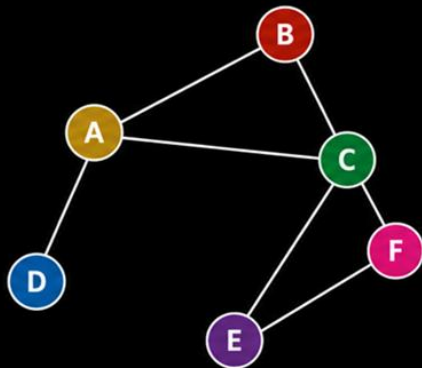
Why Can GNN Generalize to Unseen Nodes?

The same aggregation parameters are shared for all nodes.

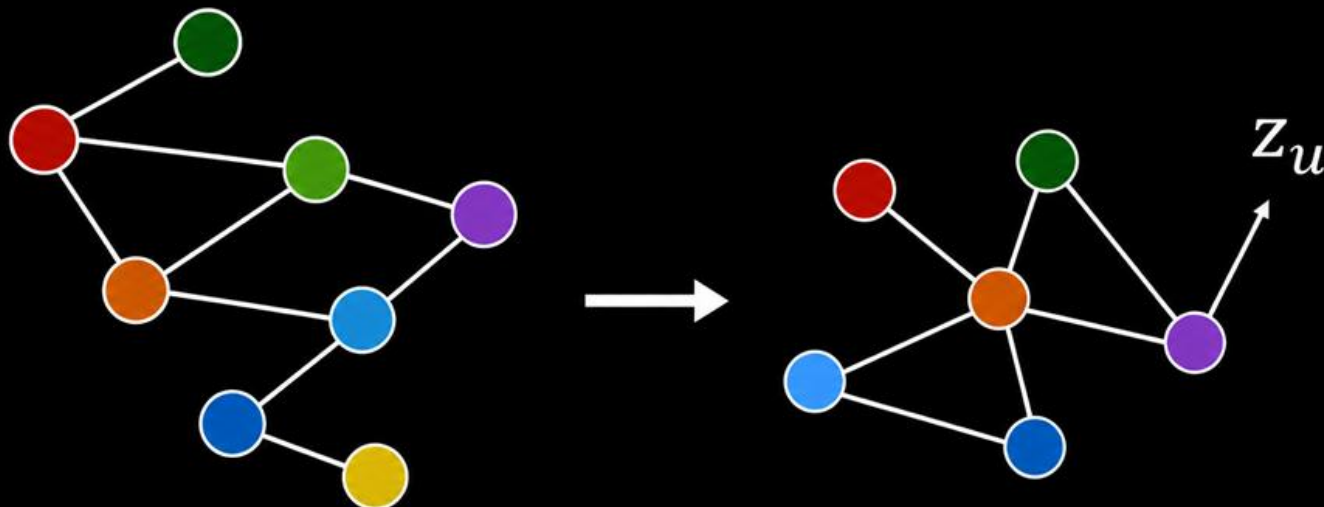
Parameter complexity is independent of the number of nodes $|V|$ because all nodes share the same aggregation function.



This parameter sharing enables generalization to previously unseen nodes.



Inductive Learning: Generalizing to New Graphs



Train on one graph
(GNN parameter W_k, B_k)

Generalize to new graph

- Using **same aggregation function**
- No need to additional training

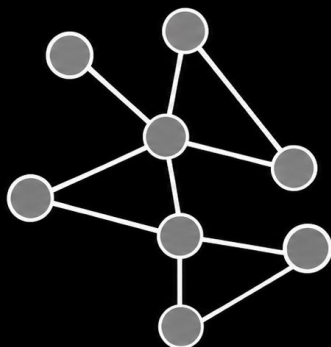
Train on protein interaction graph from organism A and generate embeddings on newly collected data about organism B

- 사람(Human)의 단백질 상호작용 그래프로 학습
- 생쥐(Mouse)나 다른 생물의 새로운 PPI (Protein-Protein Interaction) 그래프에 대해 추가 학습 없이도 노드 임베딩을 생성 가능

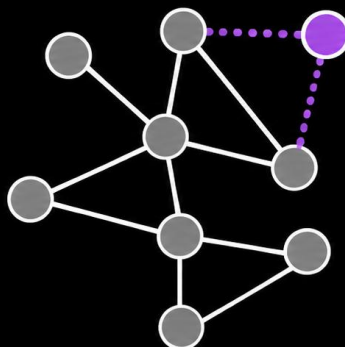
Inductive Learning: Generalizing to New Nodes

Many application settings
constantly encounter
previously unseen nodes!

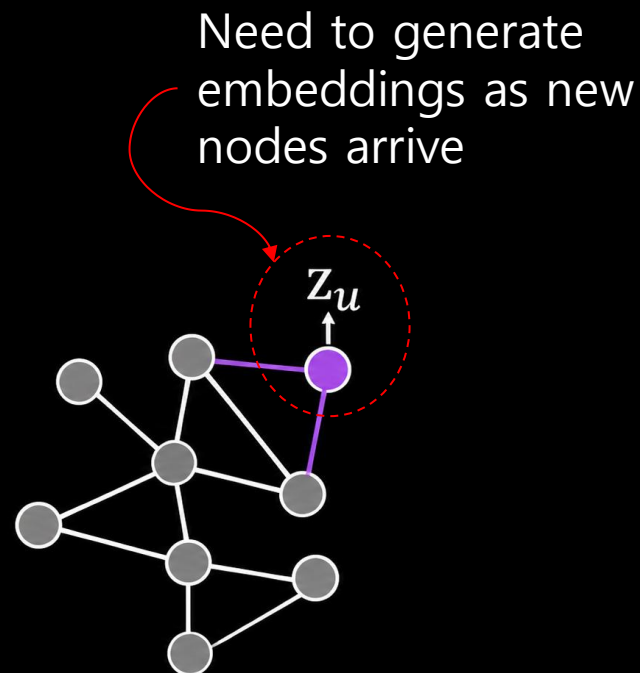
- Reddit
- YouTube
- Google Scholar
- $\vdots$



Train with snapshot



New node arrives



Generate embedding
for new node

Inductive GNNs learn an embedding function rather than storing embeddings for existing nodes.

1. Many real-world graphs continuously receive new nodes.
2. Generate embeddings as new nodes arrive — without retraining.

Play with GNN 101 Website!

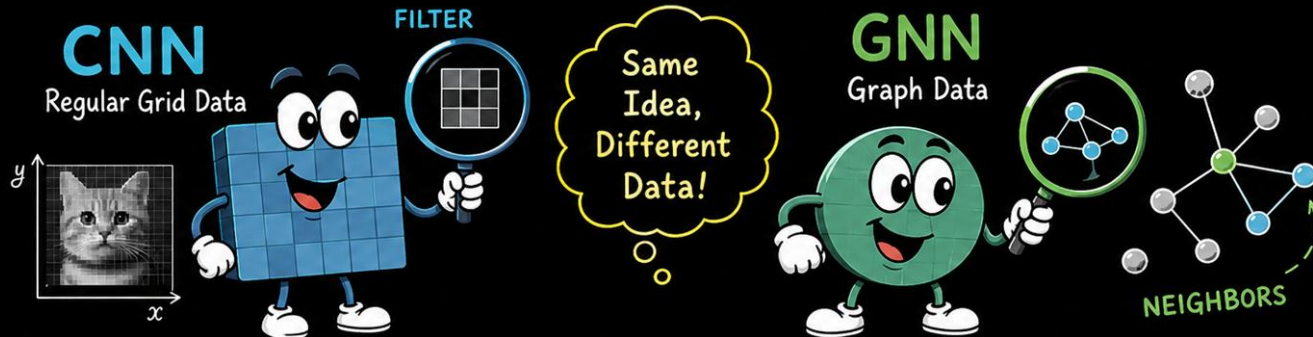
<https://visual-intelligence-umn.github.io/GNN-101/>

- GNN 101은 GNN 내부 동작을 시각적으로 탐색할 수 있는 웹 기반 학습 도구
- 레이어별 출력, Weight Matrix, Aggregation 과정, Node Embedding 변화를 단계적으로 확인
- 수식과 시각화를 연동하여 GNN의 계산 과정을 직관적으로 이해

GCN vs. CNN

What is difference, and what is similar?

What is difference, and what is similar?



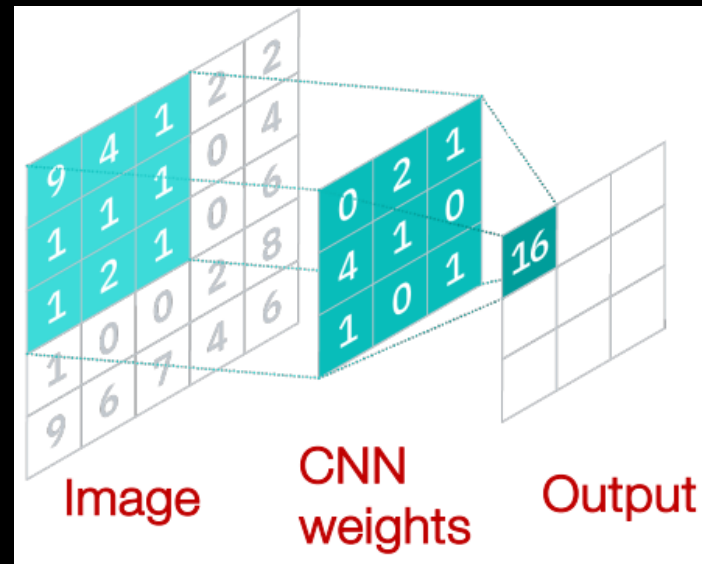
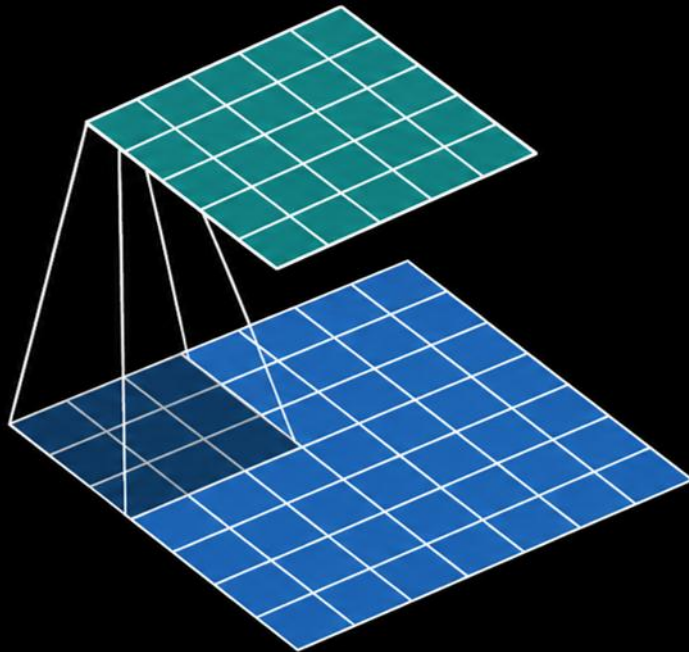
GNNs follow the convolution idea,
but work on **graphs** instead of **grids**!



Why is a GNN called a graph convolution?

Let's explore the answer step by step.

CNN: Convolution over Local Neighborhoods



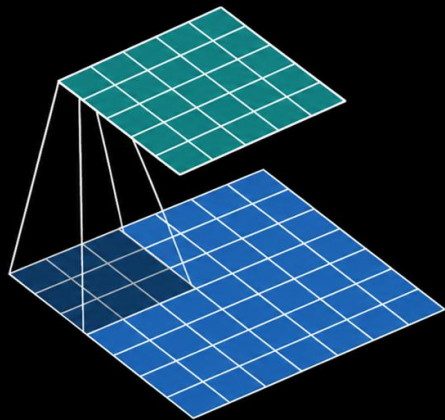
주변 이웃(neighbor)의 정보를
모아 새로운 특징을 만드는 과정

Convolution Operation

$$h_v^{l+1} = \sigma \left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} W_l^l h_u^l \right)$$

From CNN Convolution to GNN Message Passing

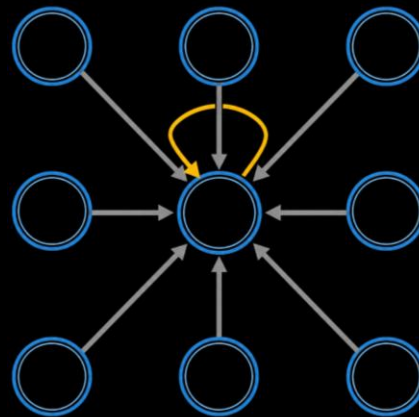
Image



$$h_v^{l+1} = \sigma \left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} (W_l^u h_u^l + B_l h_v^l) \right)$$

CNN can assign **different weights** to **neighboring pixels** because their relative positions are **fixed**.

Graph



$$h_v^{l+1} = \sigma \left(W_l \sum_{u \in \mathcal{N}(v) \cup \{v\}} (h_u^l + B_l h_v^l) \right)$$

GNN shares weights across all neighbors because graphs lack fixed positional information, and neighborhood aggregation **must preserve permutation invariance**.

GNN vs. CNN

Aspect	GNN	CNN	Key Takeaway
 Neighborhood structure	Arbitrary graph (uneven degrees)	Fixed grid structure (regular neighbors)	GNN handles general graphs; CNN has fixed local neighborhoods.
 Neighbor size	Variable (depends on node)	Fixed (e.g., $3 \times 3 = 9$)	CNN filter size is pre-defined.
 Weight sharing	All neighbors share the same weight W	Each neighbor has different weight W^u (by relative position)	GNN : shared weights CNN : position-specific weights
 Positional information	No fixed positions or ordering	Relative positions are fixed	CNN can distinguish neighbors by position; GNN cannot.
 Permutation property	Permutation invariant (equivariant)	Not permutation invariant/equivariant	Switching neighbor order changes CNN output, not in GNN .
 Update rule (aggregation)	$h_v^{(l+1)} = \sigma \left(\mathbf{W}_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{ N(v) } + B_l h_v^{(l)} \right)$	$h_v^{(l+1)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}_l^u h_u^{(l)} + B_l h_v^{(l)} \right)$	CNN is a special GNN with fixed size, ordering, and position-aware weights.



수고하셨습니다 ..^^..