

Permutation Equivariance & Invariance

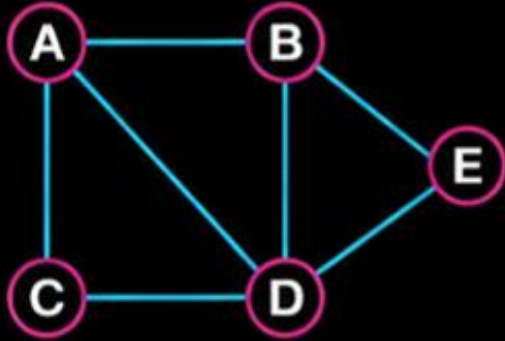
소프트웨어 끈대 강의

노기섭 교수

(kafa46@gmail.com)

Graph Setup

Assume there is a graph G



V : the set of vertex of G

A : the adjacent matrix in G

X : the matrix of node features, $X \in \mathbb{R}^{|V| \times m}$

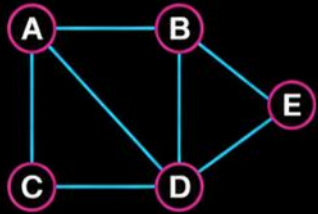
v : a node in V

$N(v)$: the set of neighbors of node v

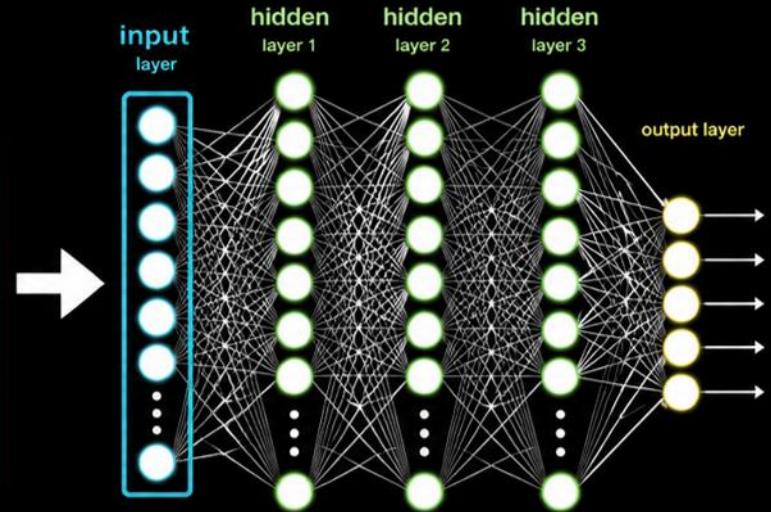
Node features:

- Social networks: User profile, user image, ...
- Biological networks: Gene expression profiles, functional info, ...
- If there is no node features in the graph datasets,
 - Indicator vectors (one-hot encoding of a node)
 - vector of constant 1 ($[1, 1, \dots, 1]$)

Why Not Use a Standard DNN?



	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



Parameter explosion:

- Input size: $n^2 + n \times m$
 $\rightarrow O(|V|) \sim O(|V|^2)$

Not applicable different graph sizes

- In general, DNN uses fixed input size
- Graph size varies
 - $G_1: |V| = 5$
 - $G_2: |V| = 20$
 - $\vdots$

Very sensitive to node ordering

$$A' = PAP^T$$

Reordering by

Permutation Matrix P

- $vec(A) \neq vec(A')$
- Node A is still the same node, connected to the same neighbors, but it may have a different representation!

Recap: Permutation Matrix (1/3) - Math Expression

How to reorder the adjacent matrix A ?

Permutation matrix:

A permutation matrix is created by rearranging the rows (or columns) of an identity matrix.

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Row ordering

(1, 2, 3)



(3, 1, 2)

Example graph: 1 — 2 — 3

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad PA = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

	1	2	3
3	0	1	0
1	0	1	0
2	1	0	1

- The row order has changed,
- but the column order has not.
- Therefore, this is not a valid adjacency matrix.
- The columns must also be reordered.

Recap: Permutation Matrix (2/3) - Visual Interpretation

Original graph



Original adjacency matrix

A

Row/column order: (1, 2, 3)

$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

Node reordering

$(1, 2, 3) \rightarrow (3, 1, 2)$

Permutation matrix

$$P = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 3 \\ 1 \\ 2 \end{matrix} & \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

P reorders rows into the order (3, 1, 2).

After applying P to rows

PA

Row order: (3, 1, 2)
Column order: (1, 2, 3)

$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 3 \\ 1 \\ 2 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \end{matrix}$$

Not a valid adjacency matrix yet.

Rows changed, columns did not.

Reordered graph
(node order 3, 1, 2)



Same graph structure, different representation.

Reordered adjacency matrix

$A' = PAP^T$

Row/column order: (3, 1, 2)

$$\begin{matrix} & \begin{matrix} 3 & 1 & 2 \end{matrix} \\ \begin{matrix} 3 \\ 1 \\ 2 \end{matrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

Valid adjacency matrix.

Rows and columns are reordered consistently.

Recap: Permutation Matrix (3/3)

Left multiplication by P permutes the rows of the adjacent matrix A .

$PA \rightarrow$ Rows are reordered!

	1	2	3
3	0	1	0
1	0	1	0
2	1	0	1

Right multiplication by P^T permutes the columns in the same order.

$AP^T \rightarrow$ Columns are reordered!

	3	1	2
1	0	0	1
2	1	1	0
3	0	0	1

Both the row and column indices must be permuted consistently to preserve the graph structure.

Graph structure is preserved!
 $1 - 2 - 3$

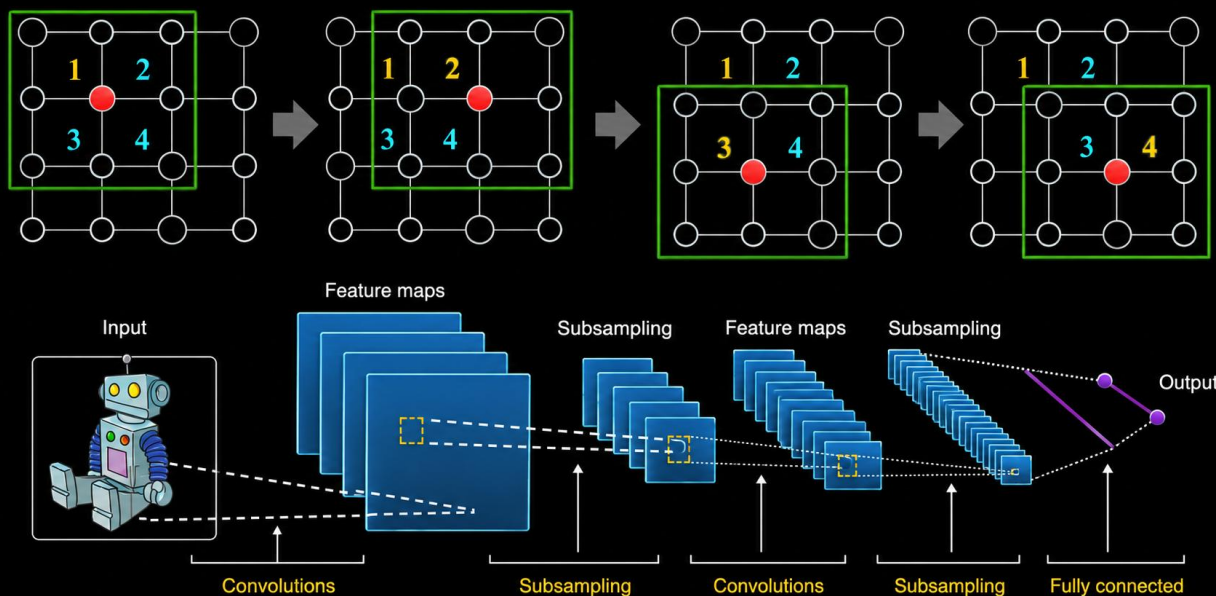
$A' = PAP^T \rightarrow$ Rows & Columns are reordered!

$$PAP^T = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

	3	1	2
3	0	0	1
1	0	0	1
2	1	1	0

Idea: Convolutional Neural Net

Goal: Generalize convolution from regular grids to irregular graphs.

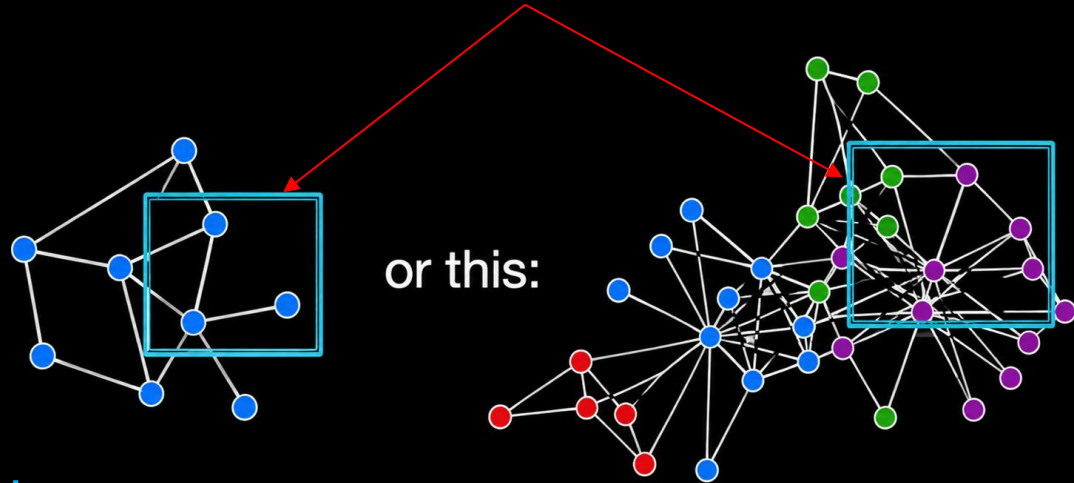


- Images have a regular grid structure.
- Each pixel has a fixed neighborhood (top, bottom, left, right).
- A convolution kernel slides consistently across the image.
- The same kernel weights are shared at every location.

Problem 1. No Sliding Window

Images vs. Graphs

Images use a fixed 3×3 receptive field, whereas graphs have variable-size neighborhoods with different numbers of neighboring nodes.



Images:

- Regular grid
- Fixed local neighborhood
- Consistent spatial ordering

Graphs:

- Irregular topology
- Variable neighborhood size
- Arbitrary neighbor ordering

Difficult to define fixed kernel 🤔

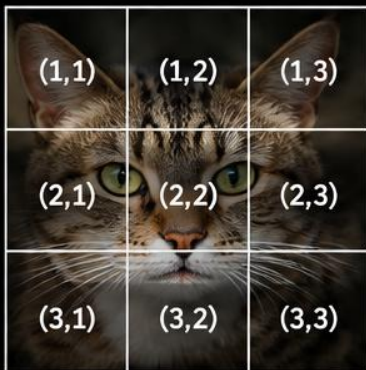
Solution: Node Aggregation 🤗

Canonical Ordering

A graph does not have a canonical ordering of its nodes.

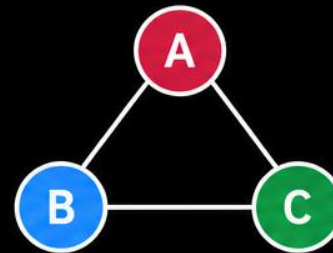
Canonical: uniquely defined standard form/order

Images have a canonical ordering



- Pixels have a natural **2D grid structure**.
- Each pixel has a **fixed position** (e.g., top-left, top-right, ...).
- There is a **canonical ordering of pixels**.

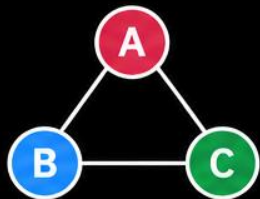
Graphs do not have a canonical ordering



- A graph is just a set of nodes and edges.
- **There is no inherent "first", "second", or "third" node.**
- The same graph can be represented with **many different node orderings**.

The same graph, different node orderings

Ordering 1: (A, B, C)

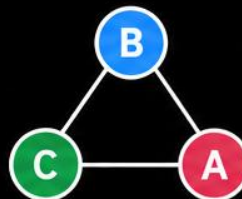


Adjacency matrix **A**

$$\begin{matrix} & \begin{matrix} A & B & C \end{matrix} \\ \begin{matrix} A \\ B \\ C \end{matrix} & \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

Same graph

Ordering 2: (B, C, A)

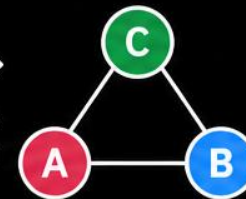


Adjacency matrix **A'**

$$\begin{matrix} & \begin{matrix} B & C & A \end{matrix} \\ \begin{matrix} B \\ C \\ A \end{matrix} & \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

Same graph

Ordering 3: (C, A, B)



Adjacency matrix **A''**

$$\begin{matrix} & \begin{matrix} C & A & B \end{matrix} \\ \begin{matrix} C \\ A \\ B \end{matrix} & \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

Problem 2. Permutation Invariance

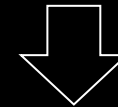
Node-level Outputs

- Permutation Equivariance is required.
 - Node Classification, Node Embedding
-

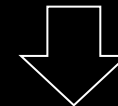
Graph-level Outputs

- Permutation Invariance is required.
- Graph Classification, Graph Regression

A graph does not
have a canonical
ordering of its nodes



Many different node
orderings of the same
graph.



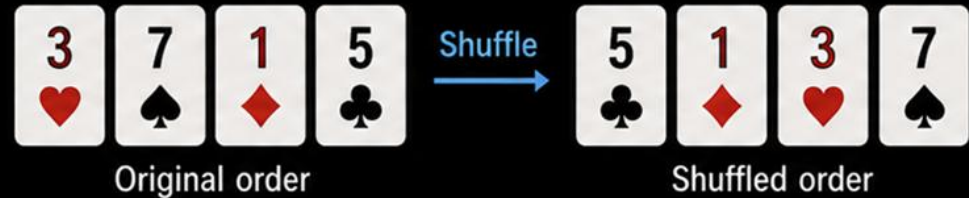
We want get the same
embedding regardless
of how the nodes are
numbered.

Permutation Invariance

The output remains the same
Even if the order of the input is changed.

$$f(\pi \cdot X) = f(X)$$

Intuitive Example (Card Set)



Sum = 16
Average = 4.0

=

Sum = 16
Average = 4.0

The output (sum, average) is identical
regardless of the order.

Main Use

Used for tasks that classify or
predict properties of an entire set
as a whole

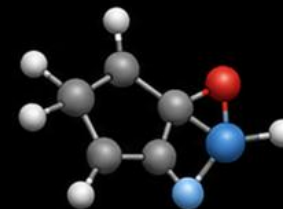
3D Point Cloud Classification



"Chair"
(Class)

Classify whether the 3D point cloud
represents a 'chair' or a 'table'.

Molecular Property Prediction



Energy
Fluctuation
= 0.83

Predict a global property (e.g.,
energy fluctuation) of a molecule.

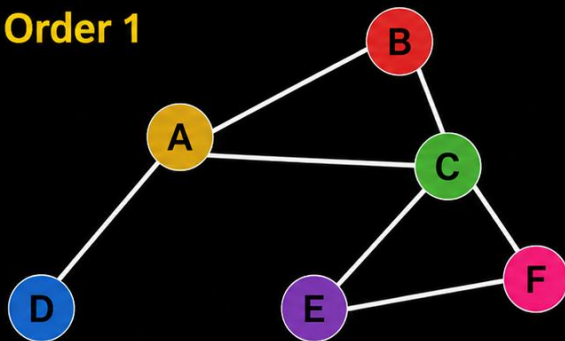
Not Canonical Ordering in Graph

Graph does not have a canonical ordering of the nodes!

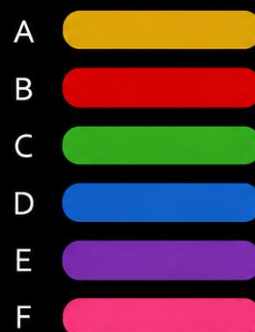
No unique or natural ordering of the nodes in a graph.

(cf. Image pixels (matrix) are always canonical)

Order 1



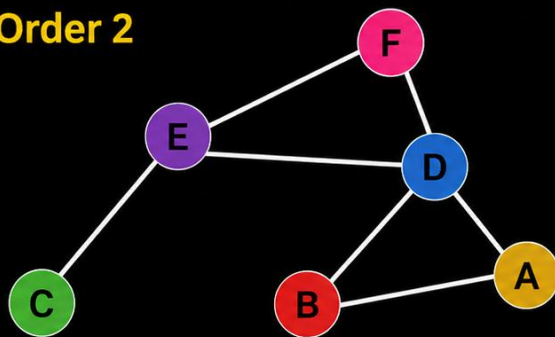
Node features X_1



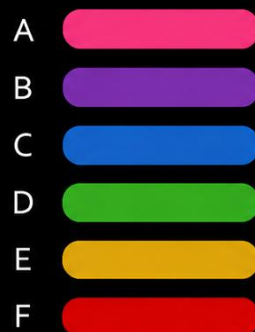
Adjacency matrix A_1

	A	B	C	D	E	F
A		1	1	1		
B			1			
C					1	1
D						
E			1			1
F					1	

Order 2



Node features X_2

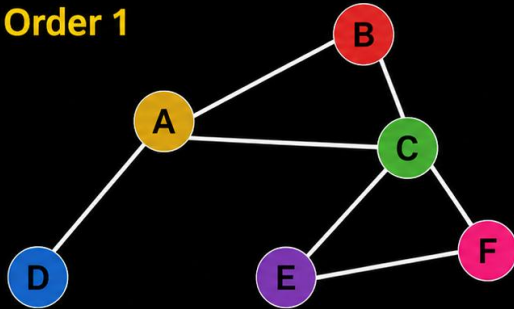


Adjacency matrix A_2

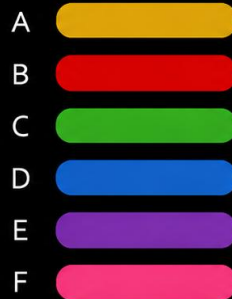
	A	B	C	D	E	F
A				1		1
B			1	1		
C					1	
D	1	1				1
E			1	1		
F	1			1	1	

Different Ordering, Same Structure

Order 1



Node features X_1



Adjacency matrix A_1

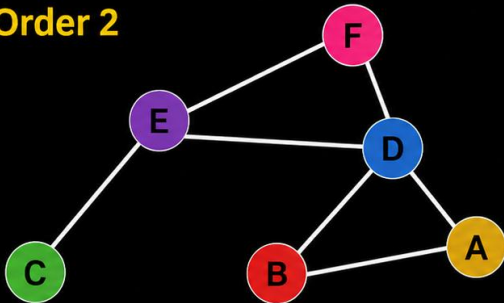
	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

Different Adj. Mat.
However,

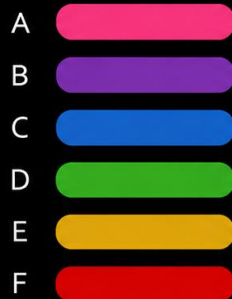
$$A_1 \neq A_2$$

$$A_2 = PA_1P^T$$

Order 2



Node features X_2



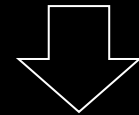
Adjacency matrix A_2

	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

Different Graph Mat.
However,

$$X_1 \neq X_2$$

$$X_2 = PX_1$$



Still the same graph!

Graph embedding should be

$$g(A_1, X_1) = g(A_2, X_2)$$

Graph-level embedding은
permutation에 불변(invariant)해야 한다!

Mathematical Definition of Permutation Equivariance

Function f : maps a graph $G = (A, X)$ to a matrix $\mathbb{R}^{|V| \times d}$

If $f(A, X) \rightarrow z$, where $z \in \mathbb{R}^{|V| \times d}$

If the input nodes are permuted, the output node embeddings must be permuted in exactly the same way.

We say f is permutation equivariant (순열 등변성)

For any graph function $f: \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times m} \rightarrow \mathbb{R}^{|V| \times d}$

f is permutation equivariant

if $Pf(A, X) = f(A', X') = f(PAP^T, PX)$ for any permutation P

d : output embedding dimensionality

m : node feature dimensionality

Permutation Equivariance (순열 불변성)

When the input order changes,
the output order changes in the
same way.

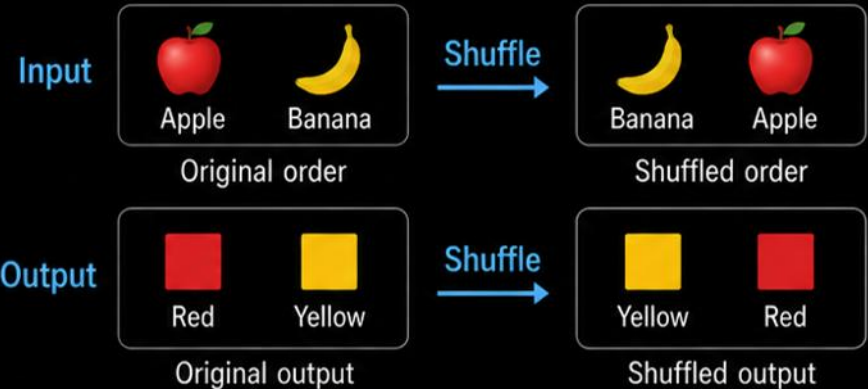
$$f(\pi \cdot X) = \pi f(X)$$

입력의 순서를 π 만큼 바꾸면 출력의 순서도
정확히 같은 π 만큼 바뀐다

Main Use

Used for node-level prediction
in graphs

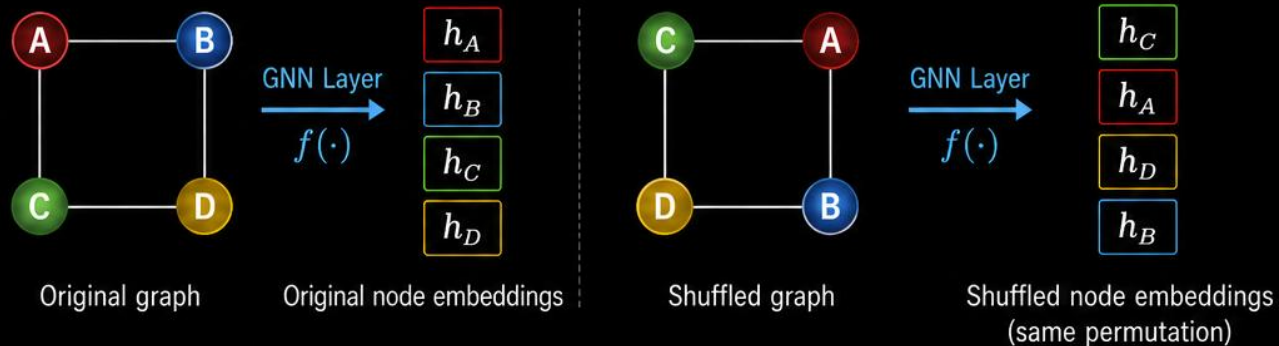
Intuitive Example (Basket of Objects)



Each item's output moves to the new position
according to the **same permutation**.

Key Properties

- Only the order changes, not the contents.
- Individual identity is maintained.
- The relative correspondence between input & output is preserved.

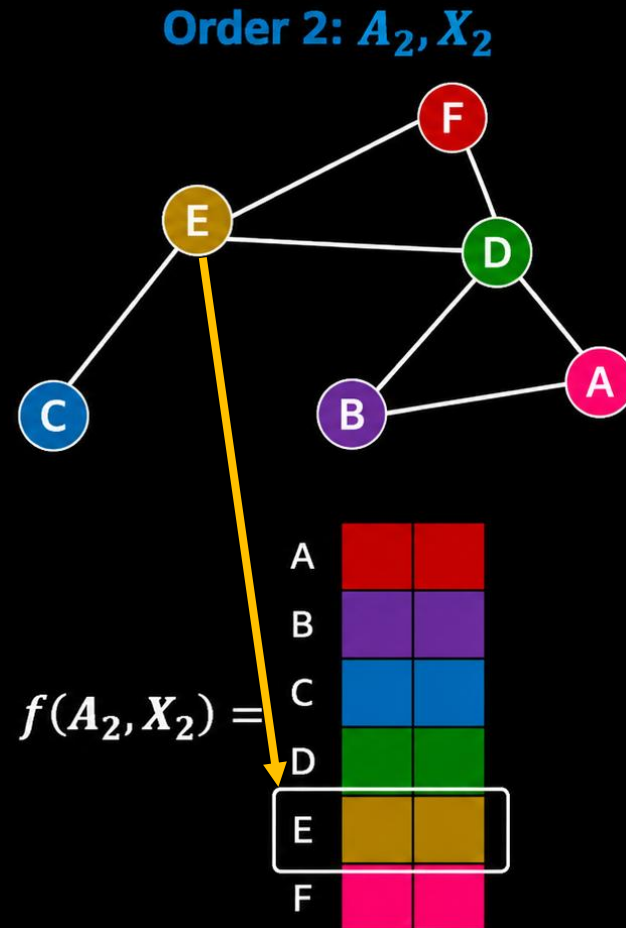
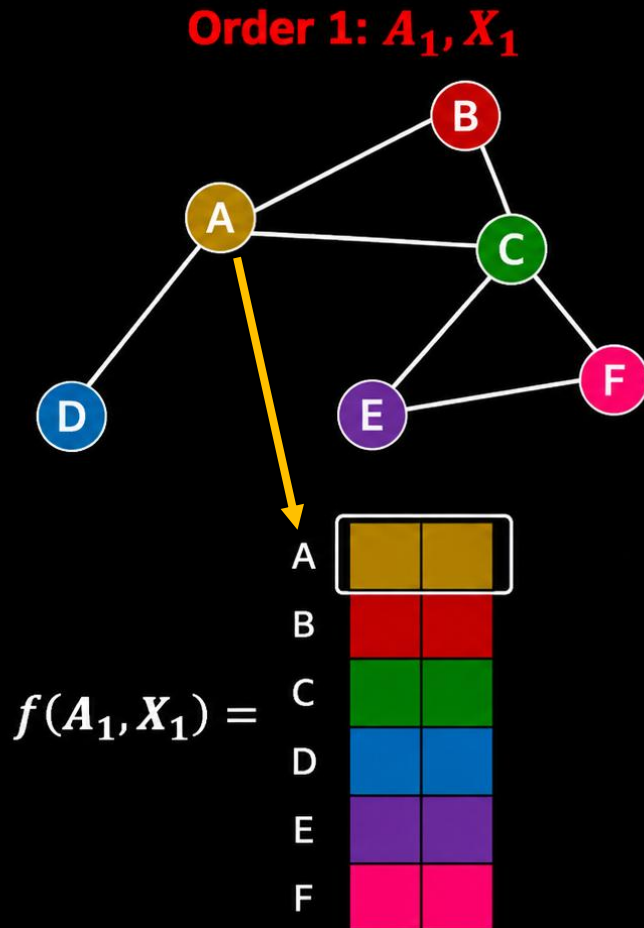


Permutation Equivariance (순열 불변성): Example 1/2

The node label changes, but the node identity does not.

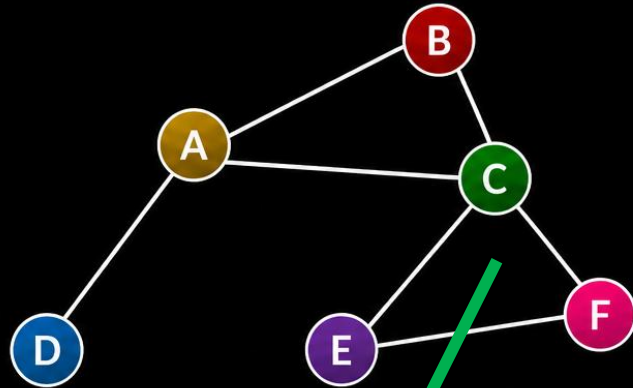
Therefore, its embedding remains unchanged.

실제 구조적 identity는 그대로임



Permutation Equivariance (순열 불변성): Example 2/2

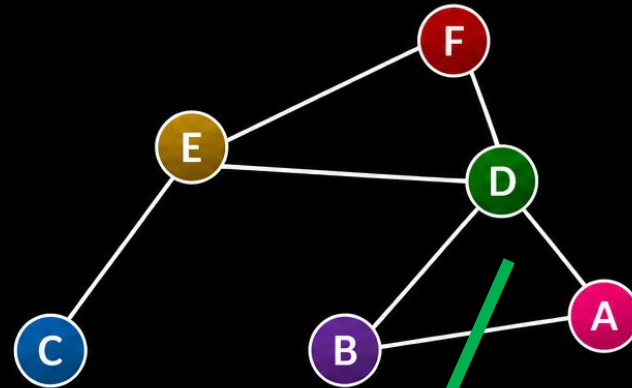
Order 1: A_1, X_1



$f(A_1, X_1) =$

A		
B		
C		
D		
E		
F		

Order 2: A_2, X_2



$f(A_2, X_2) =$

A		
B		
C		
D		
E		
F		

"같은 행 번호의 embedding이 같다"가 아니라,
같은 실제 노드 또는 같은 구조적 identity에 대응하는 embedding이 같다

Mathematical Definition of Permutation Invariance

We learn a function g that maps a graph G to a graph level representation

Function g : maps a graph $G = (A, X)$ to a vector $z \in \mathbb{R}^d$

If $g(A, X) \rightarrow z$, where $z \in \mathbb{R}^d$

If the input nodes are permuted, the graph-level output must remain unchanged.

For a given G with $|V|$ nodes, there exist $|V|!$ orderings

For any graph function $g: \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times m} \rightarrow \mathbb{R}^d$

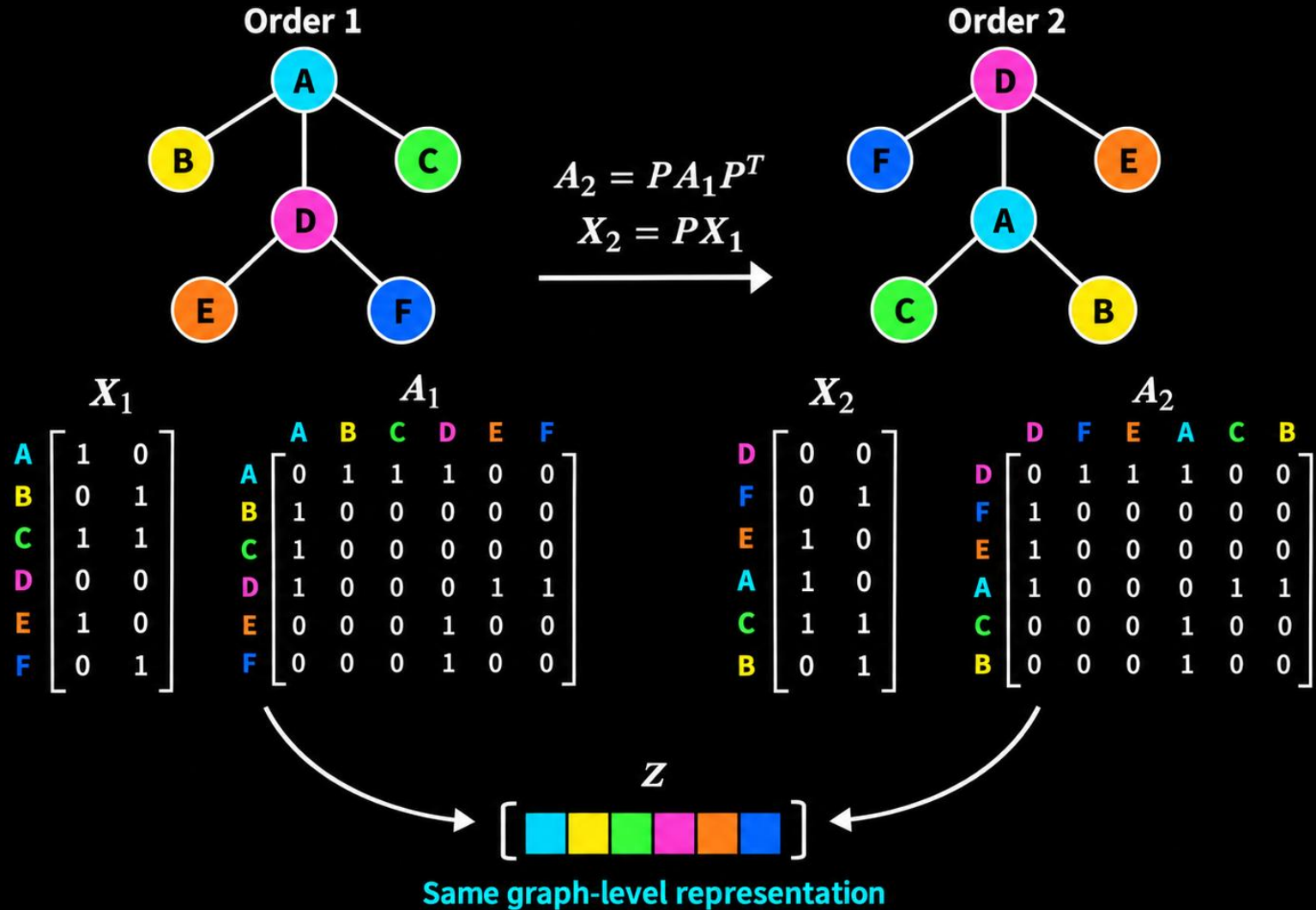
g is permutation invariant (순열 불변성)

if $g(A, X) = g(PAP^T, PX)$ for any permutation P

d : output embedding dimensionality

m : node feature dimensionality

Permutation Invariance: Example



$$g(A_1, X_1) = g(A_2, X_2) \iff g(P A P^T, P X) = g(A, X)$$

Summary: Invariance & Equivariance

Invariance

$$f(A, X) = f(PAP^T, PX)$$

입력을 아무리 재배열해도
출력은 변하지 않는다!

$$f(A, X) = 1^T X$$

$$\because f(PAP^T, PX) = 1^T PX = 1^T X = f(A, X)$$

where $1^T = [1, 1, \dots, 1]$

Add all features.
No matter what orderings are.

Graph Classification, Graph Regression,
Graph-level Embedding etc.

- Molecule $\rightarrow$ toxic / non-toxic
- Protein $\rightarrow$ function prediction
- Entire graph $\rightarrow$ one representation

Equivariance

$$Pf(A, X) = f(PAP^T, PX)$$

노드의 순서를 바꾸면 임베딩
출력의 순서도 같이 바뀐다!

$$f(A, X) = X \quad \because f(PAP^T, PX) = PX = Pf(A, X)$$

$$f(A, X) = AX$$

$$\because f(PAP^T, PX) = PAP^T X = PAX = Pf(A, X)$$

Key Idea of Graph Convolutional Networks (GCN)
& Message Passing Neural Networks (MPNN)

Node Classification, Node Regression, Node Embedding, etc.

- Node $\rightarrow$ class label
- Node $\rightarrow$ embedding vector
- Node $\rightarrow$ predicted property

From AX to Message Passing

Adjacent Matrix

$$A = \begin{bmatrix} 0 & 1 & 1 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Feature Matrix

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

$$\begin{aligned} (AX)_1 &= A_{11}x_1 + A_{12}x_2 + A_{13}x_3 + \cdots + A_{1n}x_n \\ &= 0 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_3 + \cdots + 0 \cdot x_n \end{aligned}$$

$$\begin{aligned} (AX)_2 &= A_{21}x_1 + A_{22}x_2 + A_{23}x_3 + \cdots + A_{2n}x_n \\ &= 1 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 + \cdots + 1 \cdot x_n \\ &\vdots \end{aligned}$$

$$\begin{aligned} (AX)_n &= A_{n1}x_1 + A_{n2}x_2 + A_{n3}x_3 + \cdots + A_{nn}x_n \\ &= 1 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 + \cdots + 0 \cdot x_n \end{aligned}$$

$$a_i = \sum_{j \in \mathcal{N}(i)} m_{ij}$$

A node's new feature is obtained by aggregating the features of its neighbors.



수고하셨습니다 ..^^..