

2026. 01. 20. UNIST 슈퍼컴퓨팅 청소년캠프

화학 분자를 그래프로 이해하기: 그래프 신경망 실습

우제현

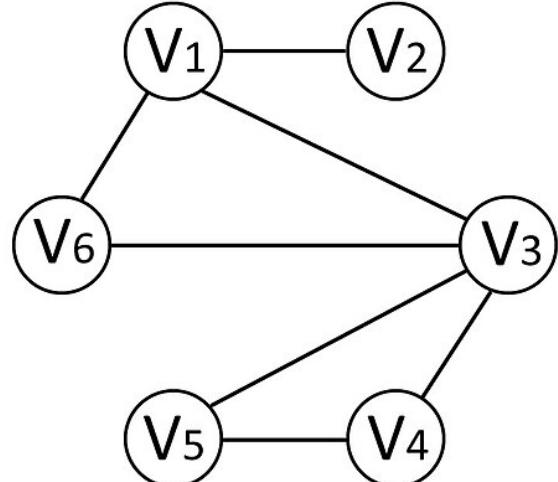
KISTI 슈퍼컴퓨팅가속화연구단



1. What is the graph?
2. Graph Neural Network
 1. Graph Neural Network
 2. Graph Convolution Network (GCN)
 3. Graph Attention Network (GAT)
 4. Graph Isomorphism Network (GIN)
3. Graph Representation of Molecules
 1. SMILES Representation
 2. RDKit
4. Training GNN Models
 1. Implementation of GCN model
 2. Paper Review: MolCLR
 3. Pre-Training / Fine-Tuning (Transfer Learning)
 4. Self-Supervised Learning
 5. MoleculeNet
5. 코드 실습
6. References

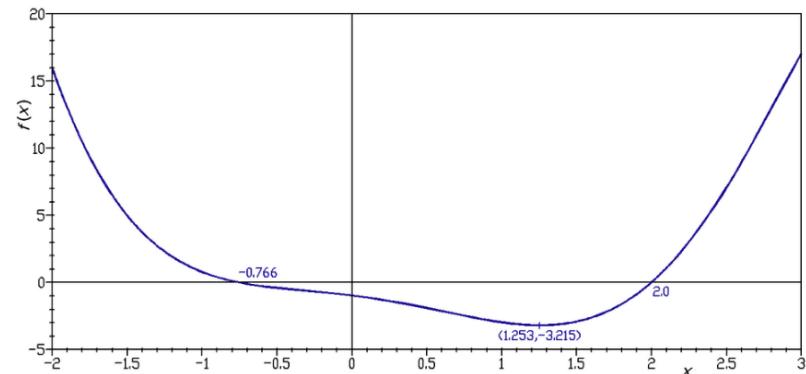
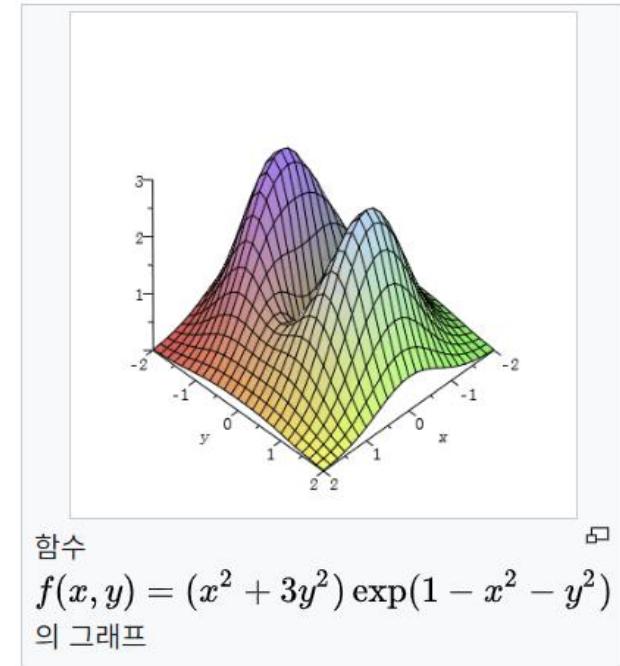
What is the graph?

What is the graph?



6 nodes + 7 edges로 구성된 graph

- Graph: 일련의 꼭짓점(vertex or node)들과 그 사이를 잇는 변(edge)들로 구성된 구조
- “Graph”의 어원:
 - γράφειν (graphein) = 긋다, 새기다, 쓰다, 그리다
 - 함수의 그래프, 그래픽(graphic), etc.
- “Graph”라는 용어는 1878년 수학자 J. J. 실베스터에 의해 처음 사용



함수 $f(x) = x^4 - 4^x$ 의 그래프

Why Graphs?

Graphs are a general language for describing and analyzing entities with relations/interactions.

What is the graph?

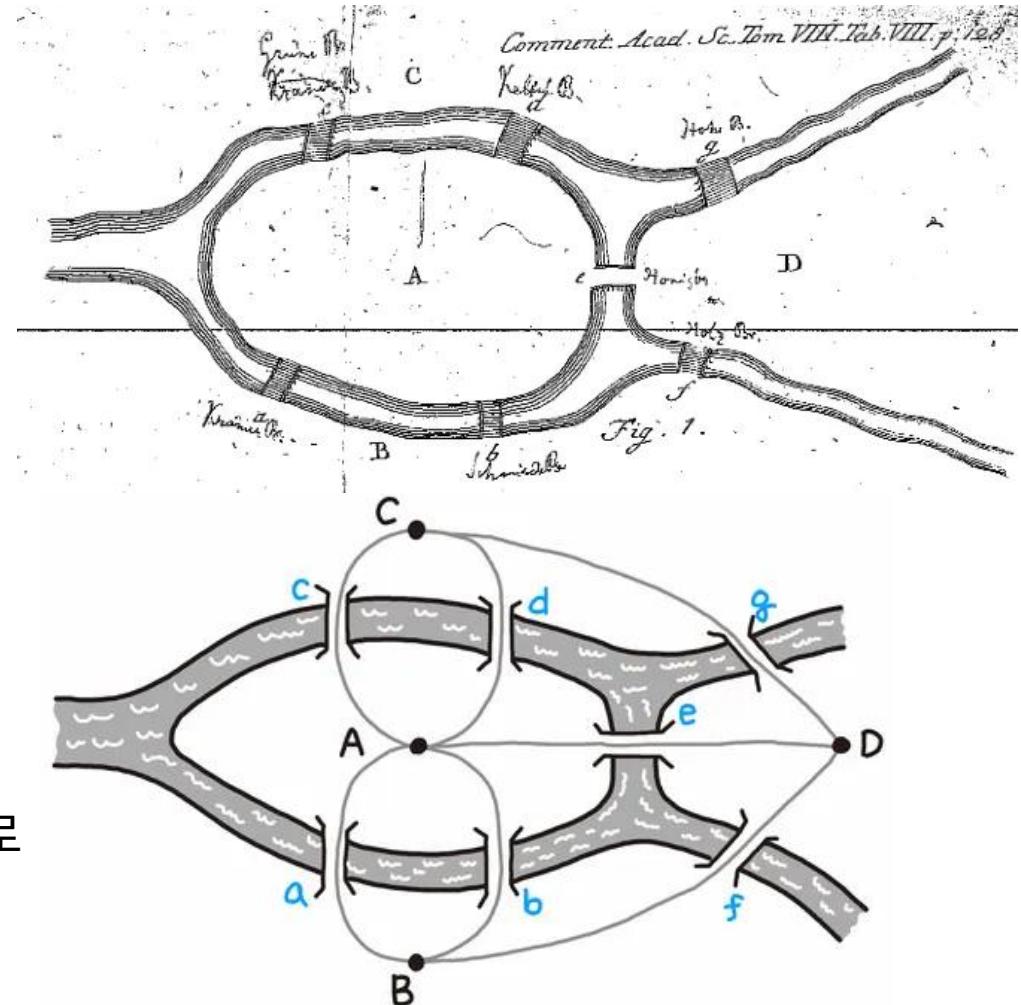


쾨니히스베르크 다리 건너기 문제 (한붓그리기)

쾨니히스베르크시의 한 가운데는 프레겔 강이 흐르고 있고 여기에는 가운데 섬들과 연결되어있는 일곱 개의 다리가 있다. 그 다리를 한 번씩만 차례로 모두 건널 수 있겠는가?

레온하르트 오일러
Leonhard Euler

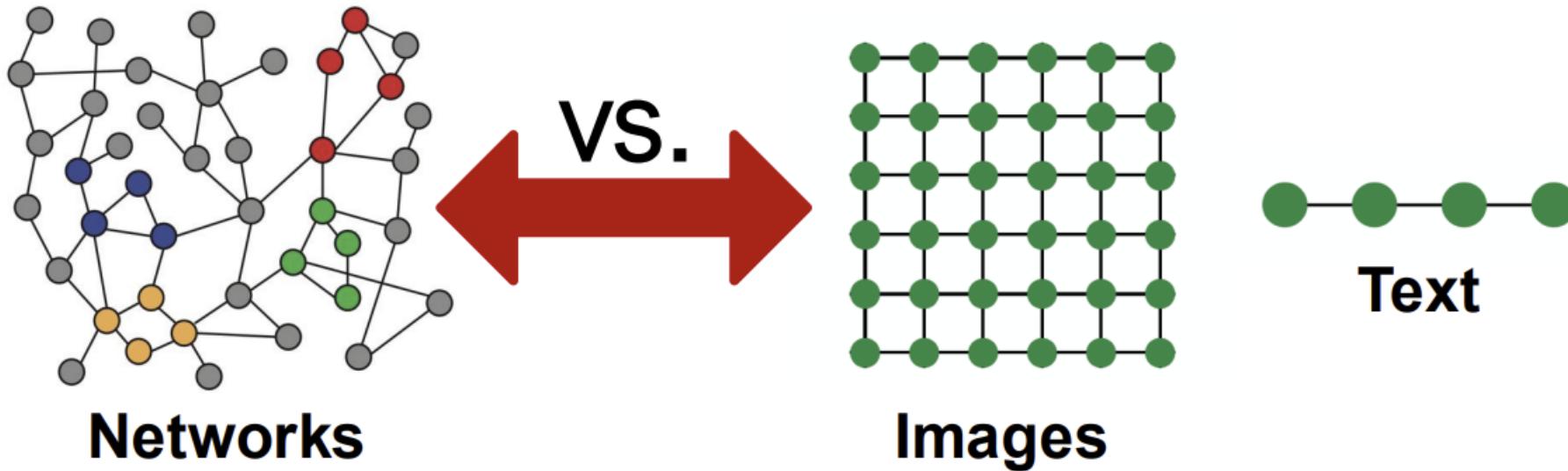
- **오일러 경로:** 그래프에서 모든 간선을 정확히 한 번씩 방문하는 경로
 - G 는 오일러 경로를 가진다.
 - G 의 모든 꼭짓점의 차수는 짝수이다.
- **오일러 회로:** 시작점과 끝 점이 같은 오일러 경로
 - G 는 오일러 회로를 가진다.
 - G 의 홀수 차수 꼭짓점의 수는 0 또는 2이다.



당시 오일러는 이 문제를 다음 그림과 형태로 각각의 다리에 a부터 g까지 이름을 부여하고 도식화해 1735년에 논문을 발표, 해답이 없음을 수학적으로 증명.

What is the graph?

- Arbitrary size and complex topological structure (*i.e.*, no spatial locality like grids)

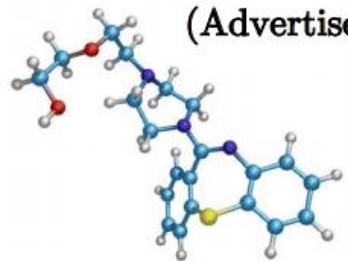


- No fixed node ordering or reference point
- Often dynamic and have multimodal features

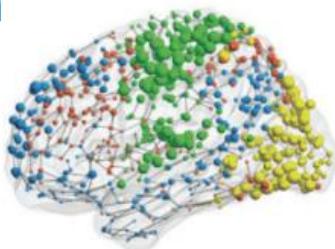
What is the graph?



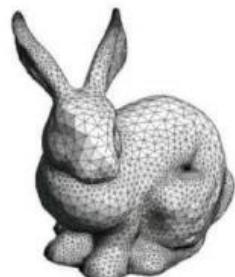
Social networks
(Advertisement)



Drug/Material
molecules
(Chemistry)



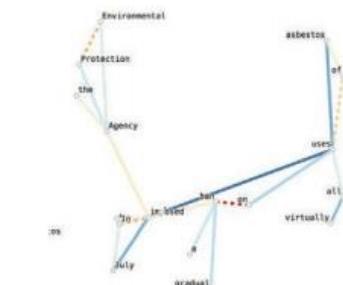
Brain
connectivity
(Neuroscience)



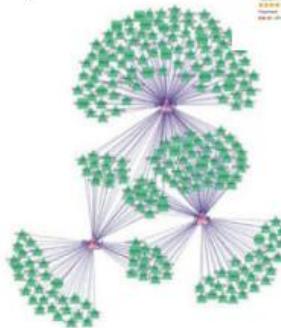
3D Meshes
(Computer Graphics)



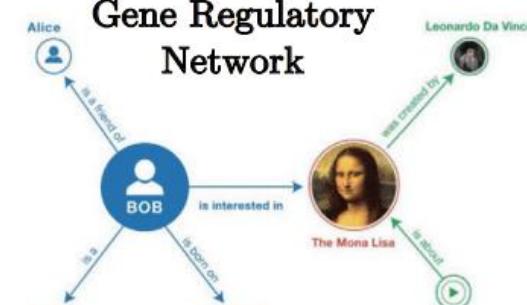
Transportation
networks



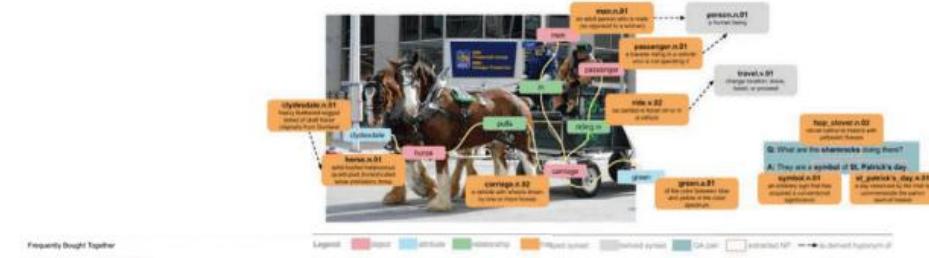
Words relationships
(NLP)



Gene Regulatory
Network



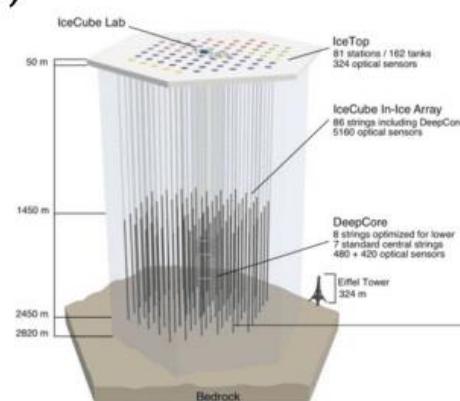
Knowledge graphs



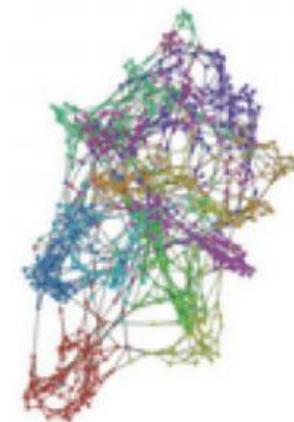
Scene understanding



Recommender
systems (Amazon,
Netflix)



Neutrino
detection (High-
energy Physics)

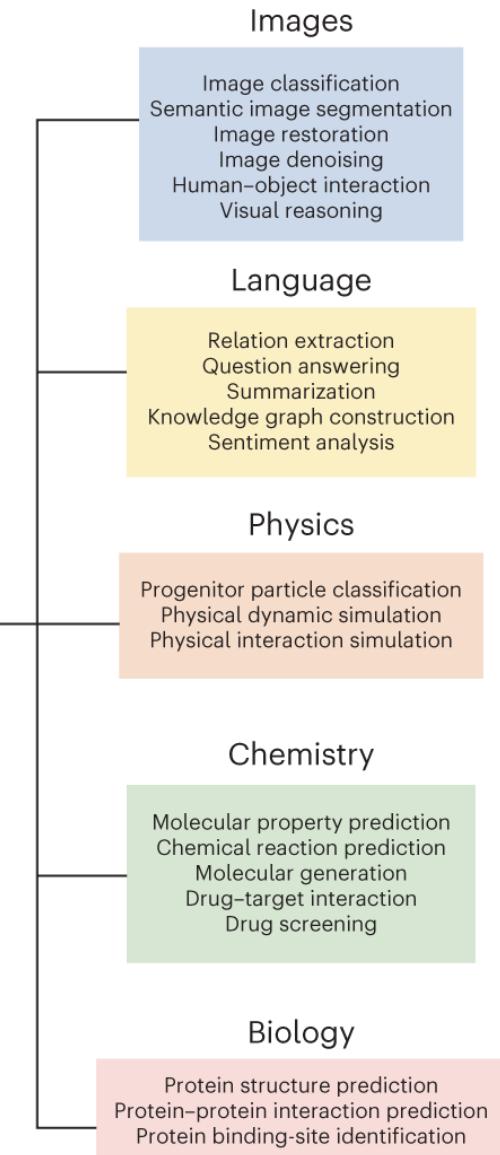
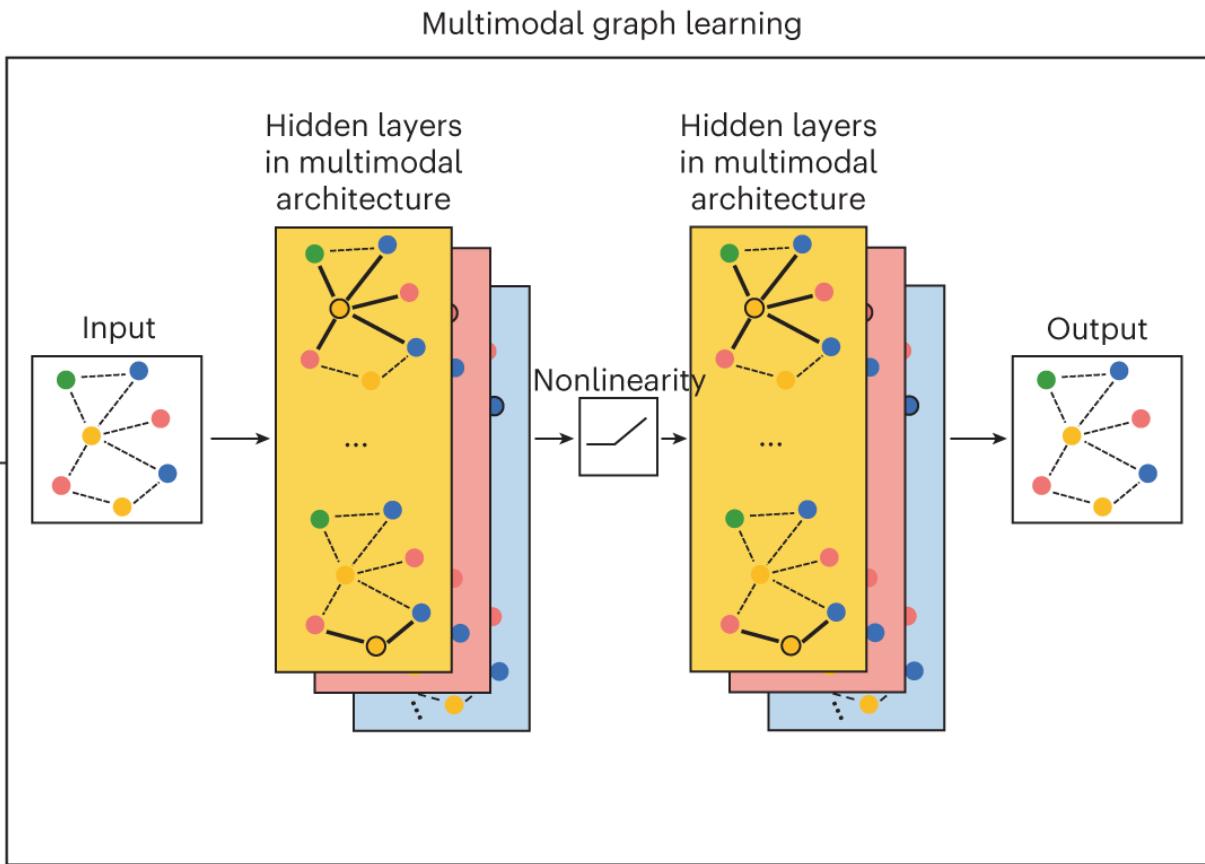
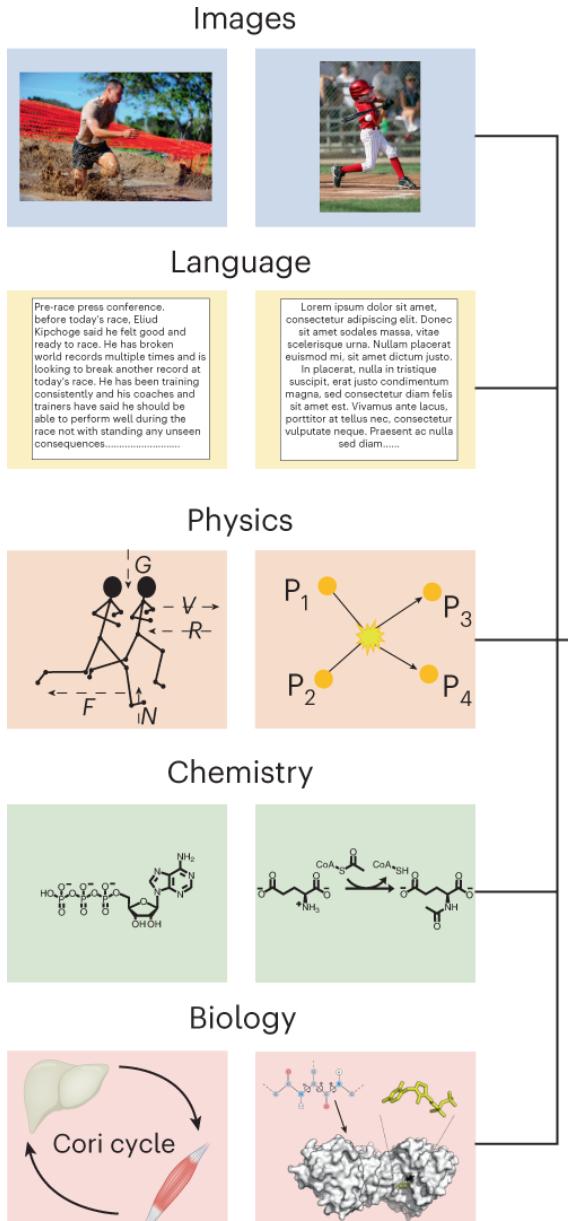


Graph

Graph Neural Network

- Graph Neural Network
- Graph Convolution Network (GCN)
- Graph Isomorphism Network (GIN)
- Graph Attention Network (GAT)

Graph Neural Network



GNN에서의 예측 종류 (Prediction Levels)

GNN은 그래프의 어느 단위를 예측 대상으로 삼느냐에 따라 세 가지 문제로 나뉩니다.

1. Node-level prediction

(노드 하나하나에 대한 예측)

- 각 노드의 속성 / 상태 / 라벨
- “이 노드는 어떤 클래스에 속하는가?”
- “이 노드의 값은 얼마인가?”

2. Link-level (Edge-level) prediction

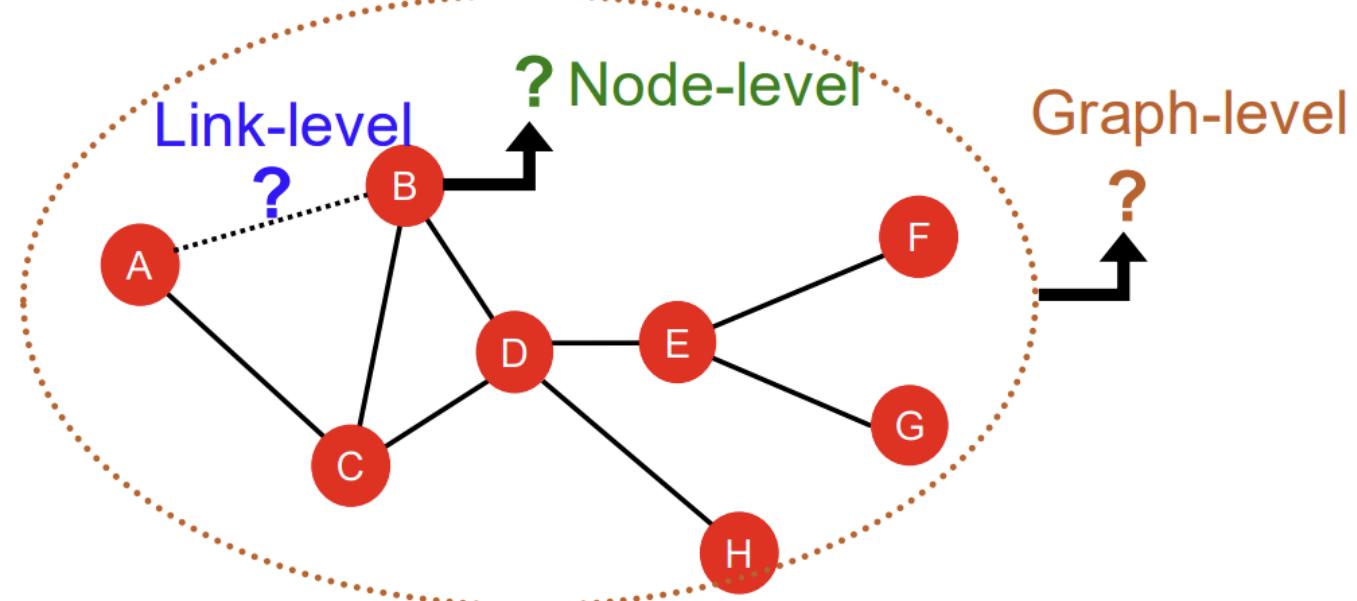
(노드 쌍 사이 관계에 대한 예측)

- 두 노드 사이에 edge가 존재하는지 혹은 edge의 타입 / 강도
- “이 두 노드는 연결될까?”
- “이 연결은 어떤 의미인가?”

3. Graph-level prediction

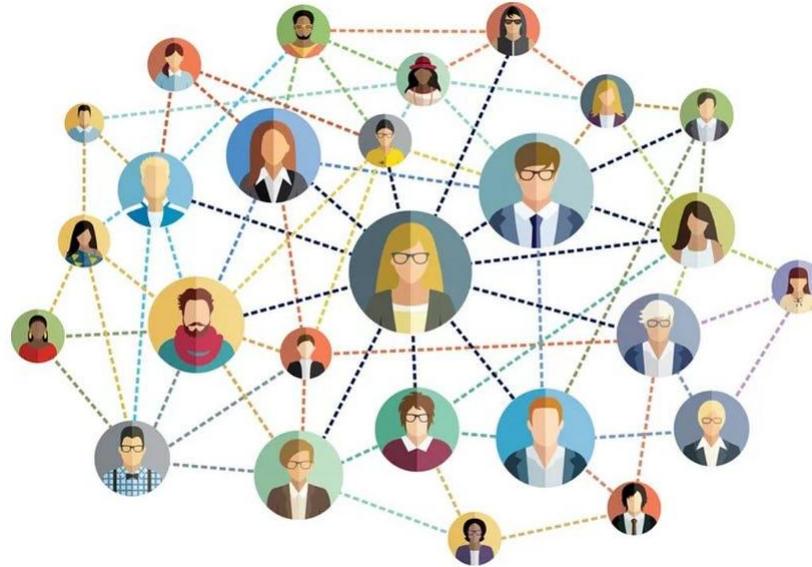
(그래프 전체에 대한 예측)

- 그래프 하나당 하나의 값/라벨
- “이 그래프의 전체 성질은?”
- “이 시스템의 결과는?”



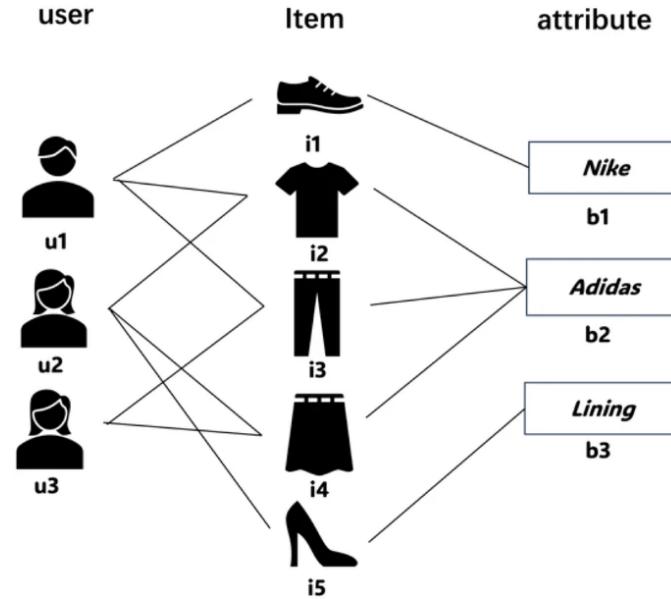
Graph Neural Network

Example: Social network



- Node: 사용자 계정 (Facebook, Instagram 등)
 - 나이, 지역, 관심사, 활동 이력 등
- Edge: 사용자 간의 관계
 - follow, message 등
- Prediction:
 - People You May Know
 - 추천 친구 (link prediction)

Example: Recommender system



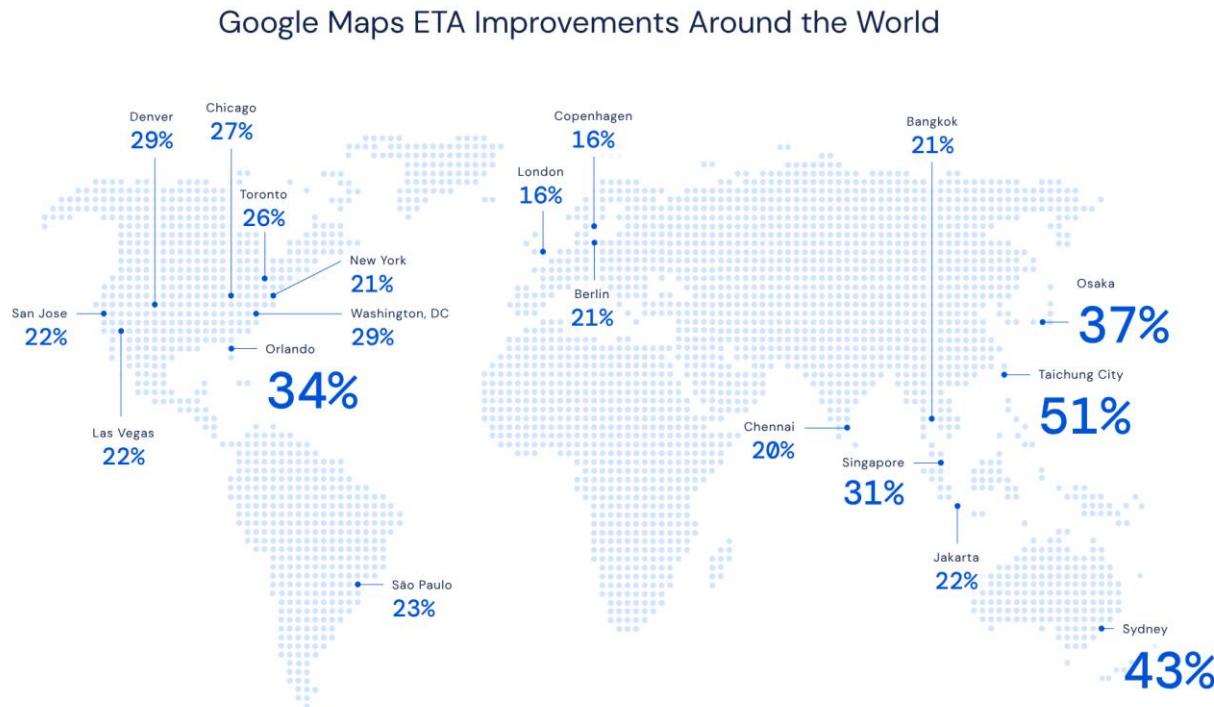
- Node: 사용자, 아이템 (상품, 영화, 음악 등)
 - 나이, 성별, 지역, 관심사, 구매/클릭 이력
 - 카테고리, 브랜드, 가격, 설명, 메타데이터
- Edge: 사용자–아이템 간 상호작용
 - 빈도, 평점, 시간 정보
- Prediction:
 - 추천 상품 / 콘텐츠
 - 평점 예측, 클릭 확률

Graph Neural Network

Example: Estimated Time of Arrival (ETA) prediction

Transportation maps are naturally modelled as graphs

- Nodes: intersections
- Edges: roads
- Features: road length, current speed, historic speed



Run GNN on supersegment graph to estimate time of arrival (graph regression).

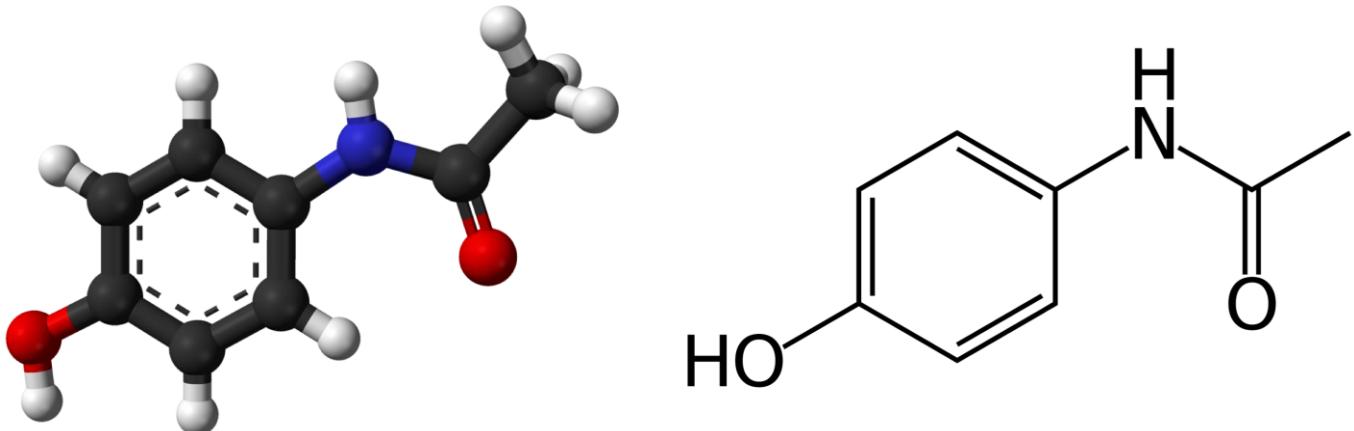
Already deployed in several major cities, significantly reduced negative ETA outcomes.

Graph Neural Network

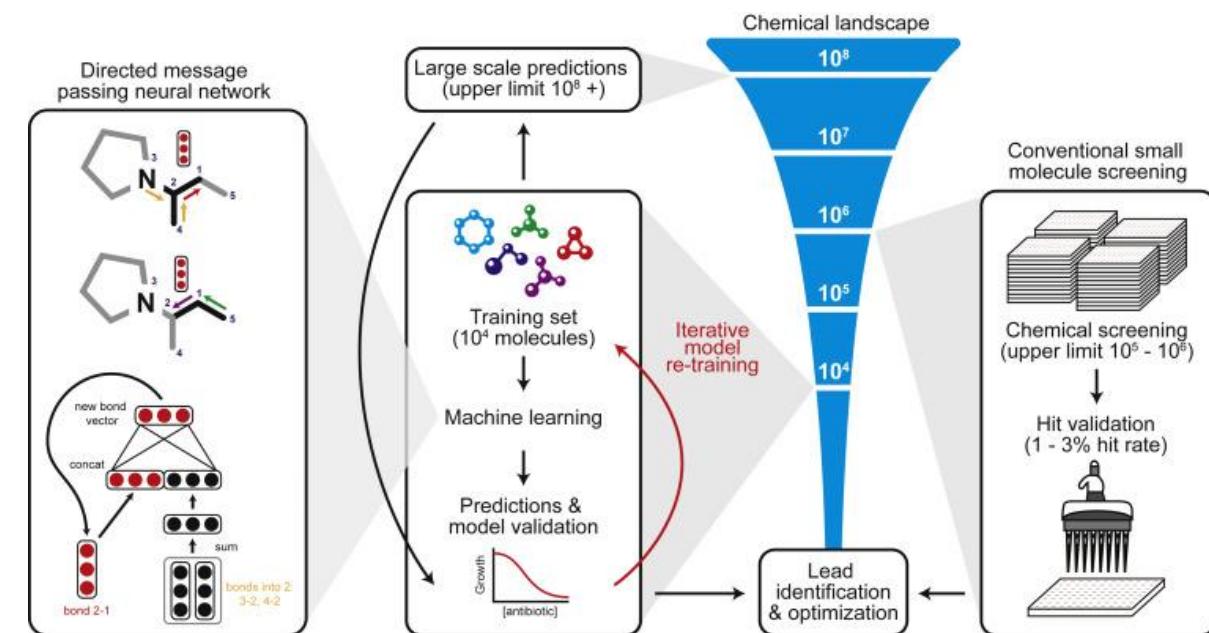
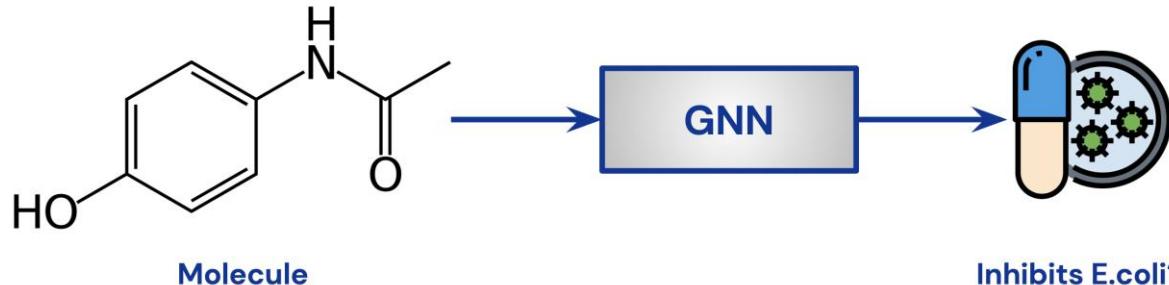
Example: Molecules

Molecules can be represented by a graph

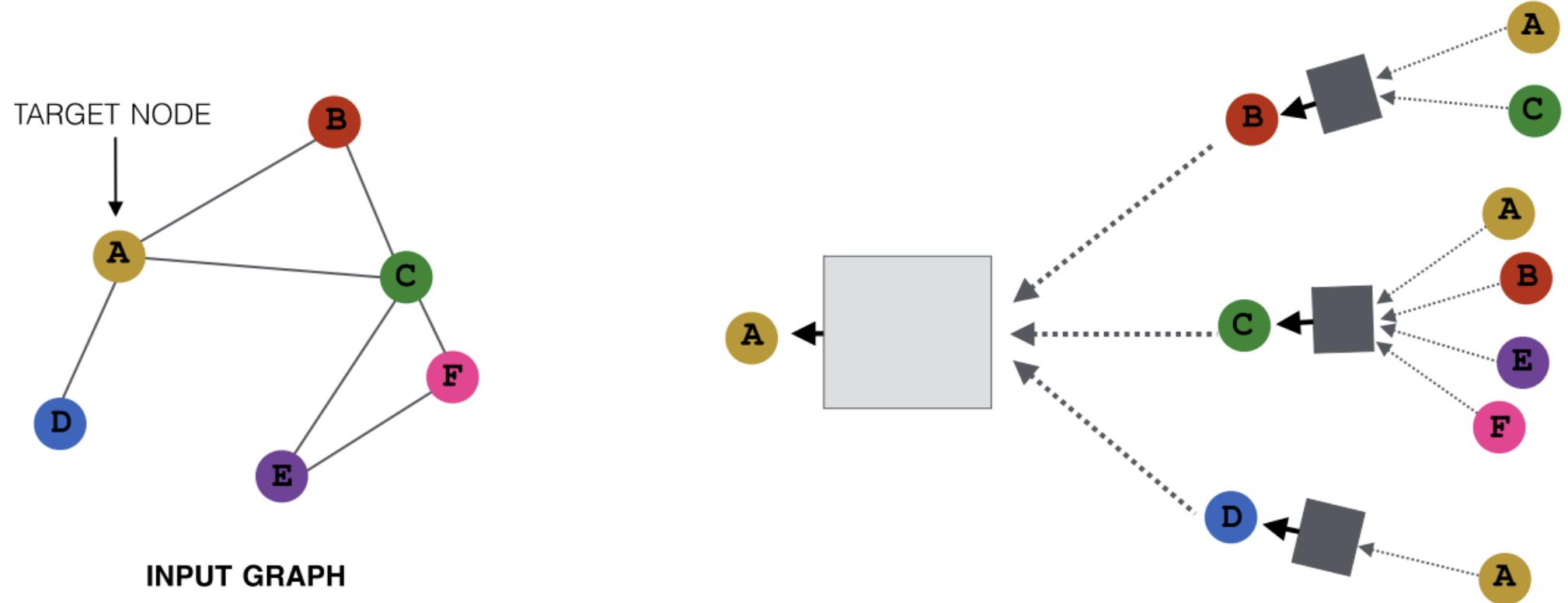
- Nodes: atoms
- Edges: bonds
- Features:
 - atom type (C, O, N, etc.)
 - charge
 - bond type (single, double, aromatic, etc.)



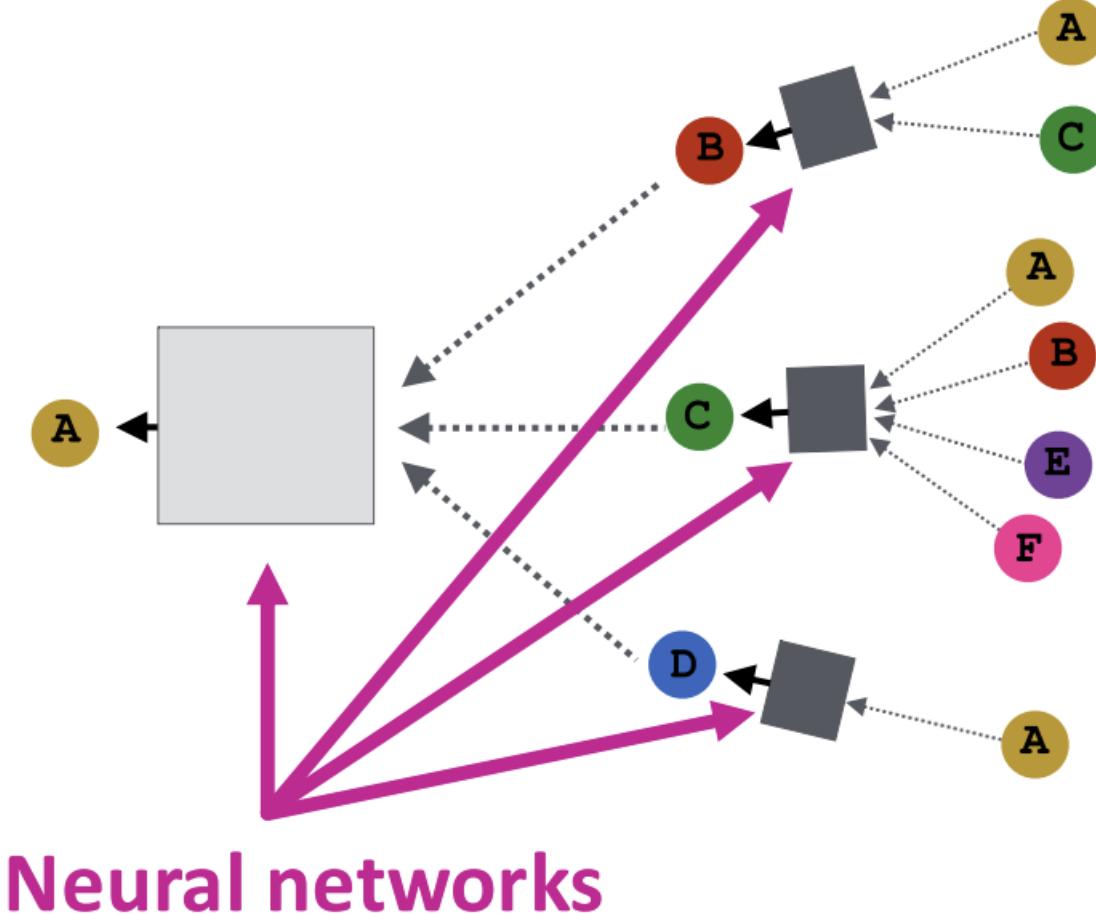
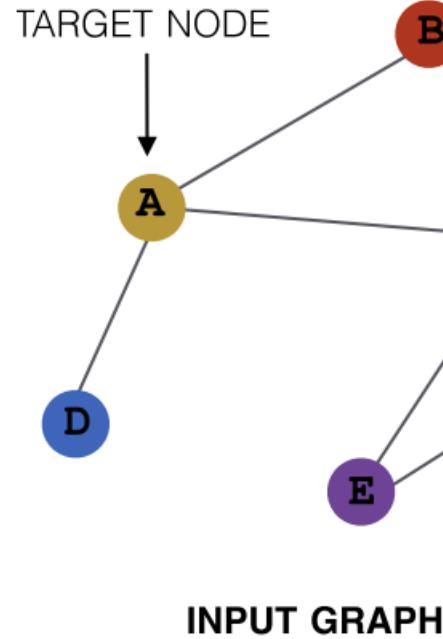
- Task: predict whether a molecule is a potent drug
- binary classification to predict whether a drug will inhibit E.coli (대장균 억제 효과)
 - Train on curated dataset for molecules whose response is known



Graph Neural Network

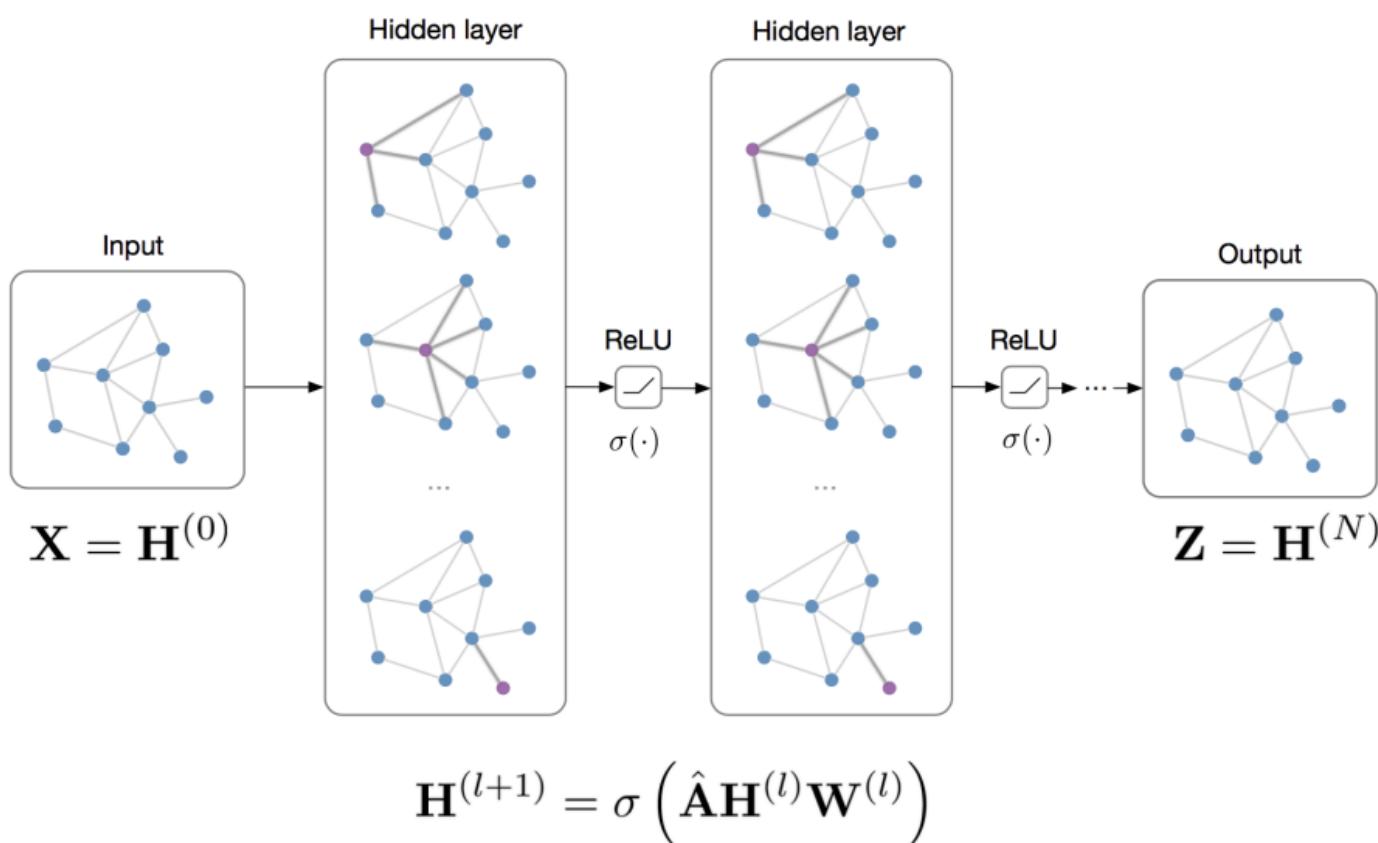


Graph Neural Network



Classification and Link Prediction with GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



Node classification:

$$\text{softmax}(\mathbf{z}_n)$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$\text{softmax}(\sum_n \mathbf{z}_n)$$

e.g. Duvenaud et al. (NIPS 2015)

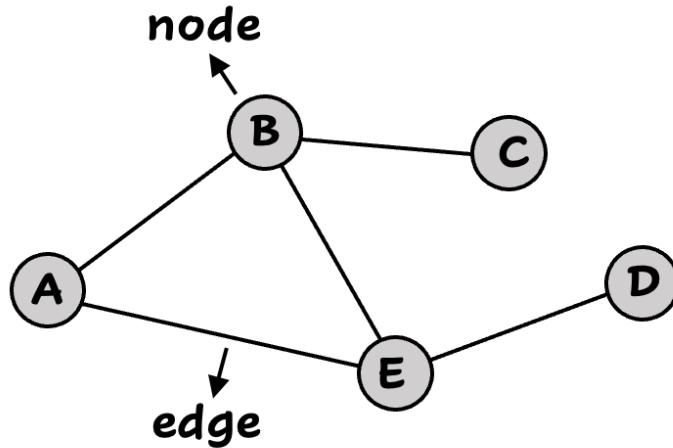
Link prediction:

$$p(A_{ij}) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$$

Kipf & Welling (NIPS BDL 2016)

“Graph Auto-Encoders”

Graph Neural Network



adjacency matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

A — Adjacency Matrix (인접 행렬)

- 그래프의 연결 구조(topology)를 행렬로 표현한 것
- 노드 개수 N 일 때,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix}$$

$$a_{ij} = \begin{cases} 1 & \text{노드 } i \text{와 } j \text{가 연결되어 있으면} \\ 0 & \text{그렇지 않으면} \end{cases}$$

▪ x_i

- i 번째 노드의 입력 feature 벡터
- 예:

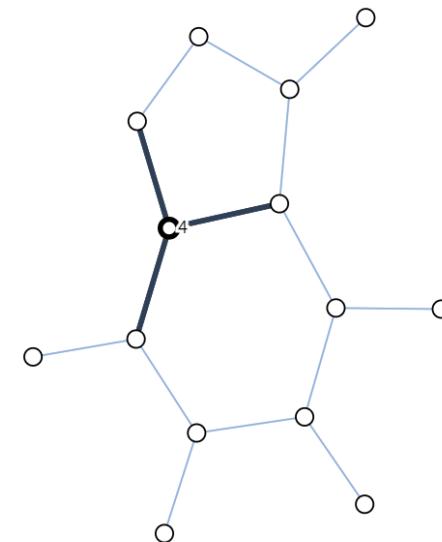
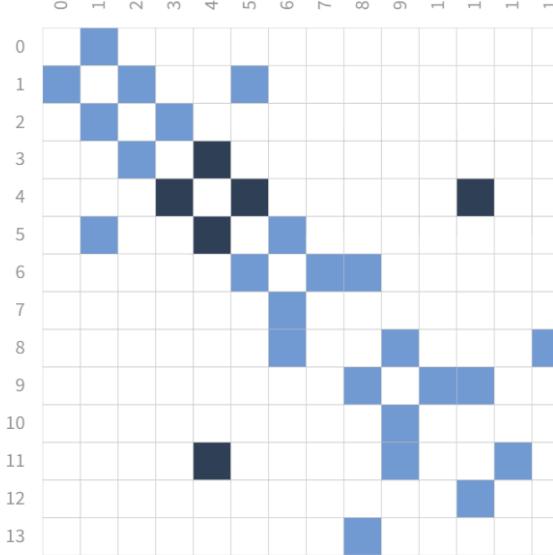
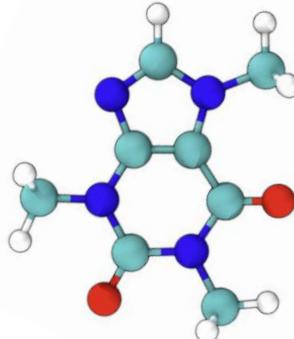
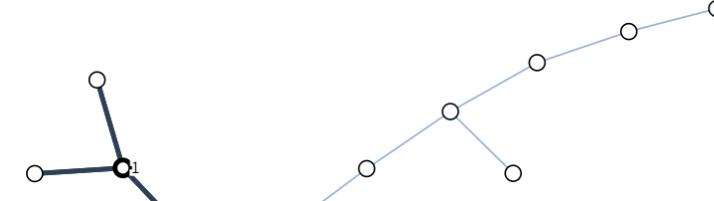
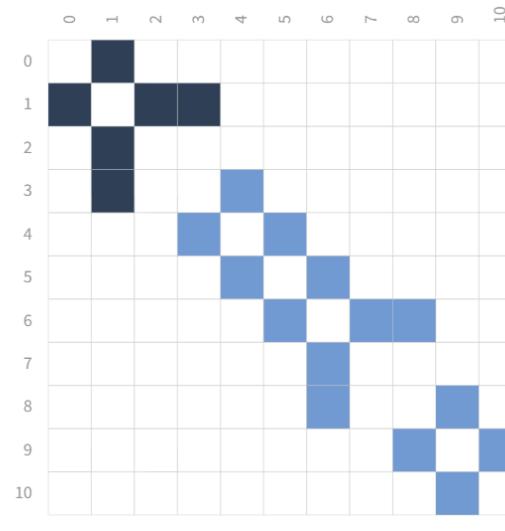
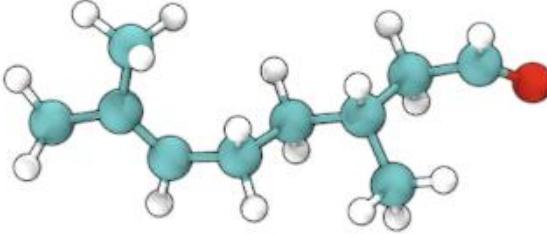
- 분자 그래프: 원자 번호, 전하, hybridization 등
- 차원: $x_i \in \mathbb{R}^F$

▪ X

- 전체 노드의 입력 feature 행렬
- 각 행이 하나의 노드

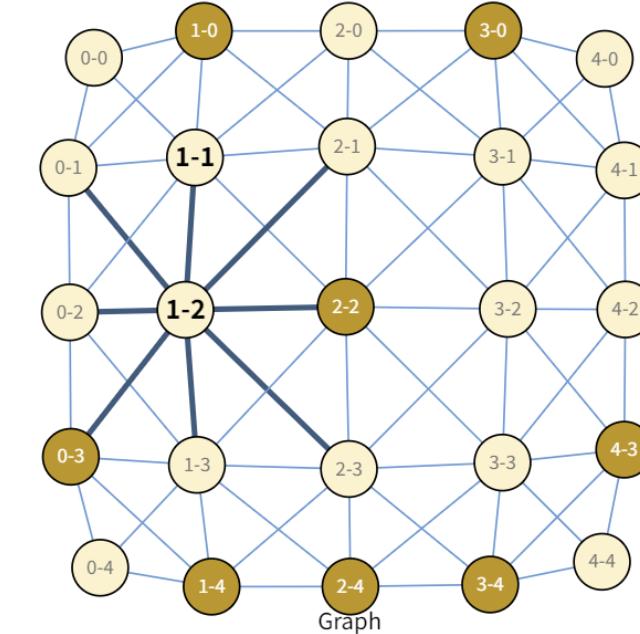
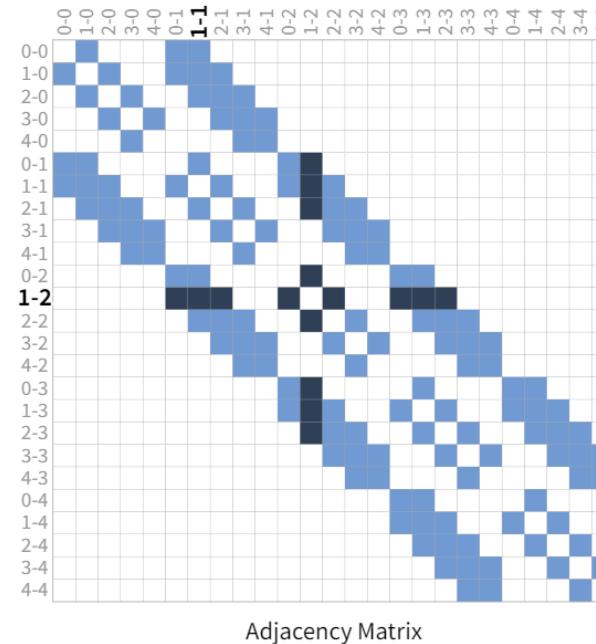
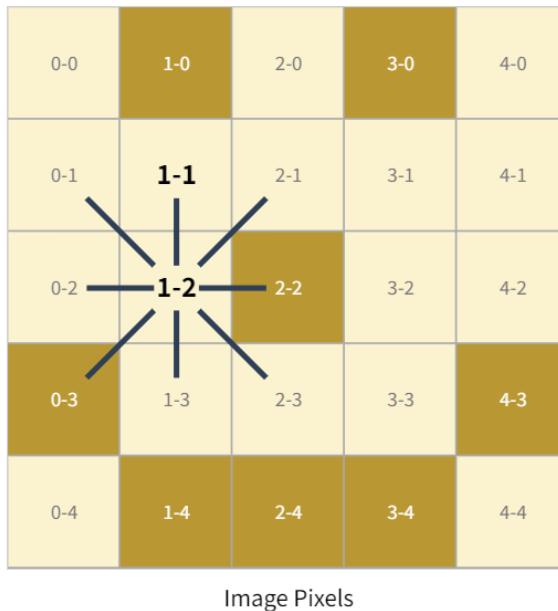
$$X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_N^\top \end{bmatrix} \in \mathbb{R}^{N \times F}$$

Graph Neural Network: Adjacency Matrix

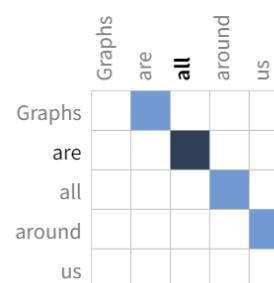


Graph Neural Network: Adjacency Matrix

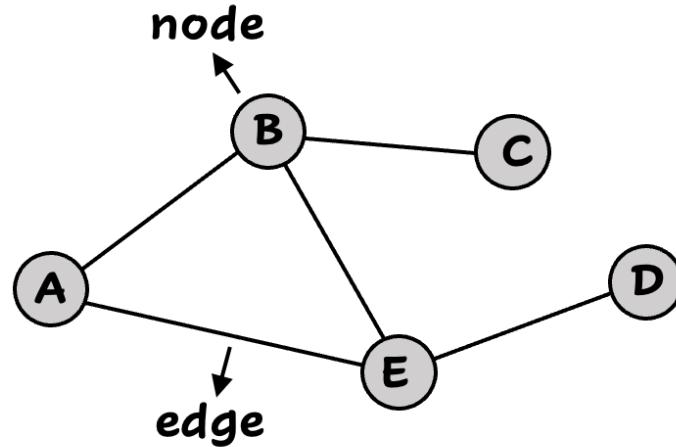
Images as graphs



Text as graphs



Graph Neural Network



- $h_i^{(l)} \in \mathbb{R}^{F_l}$
 - l번째 GCN layer에서 i번째 노드의 hidden representation
 - 즉, 노드 임베딩
- $H^{(l)}$
 - l번째 layer에서의 전체 노드 표현 행렬
 - $H^{(0)} = X$

$$H^{(l)} = \begin{bmatrix} h_1^{(l)\top} \\ h_2^{(l)\top} \\ \vdots \\ h_N^{(l)\top} \end{bmatrix} \in \mathbb{R}^{N \times F_l}$$

l — 레이어 인덱스 (Layer index)

- GCN의 layer 번호
- $l = 0, 1, 2, \dots$
- 의미:
 - $l = 0$: 입력 feature
 - $l = 1$: 1-hop 이웃 정보 반영
 - $l = 2$: 2-hop 이웃 정보 반영

W — 학습 파라미터 (Weights)

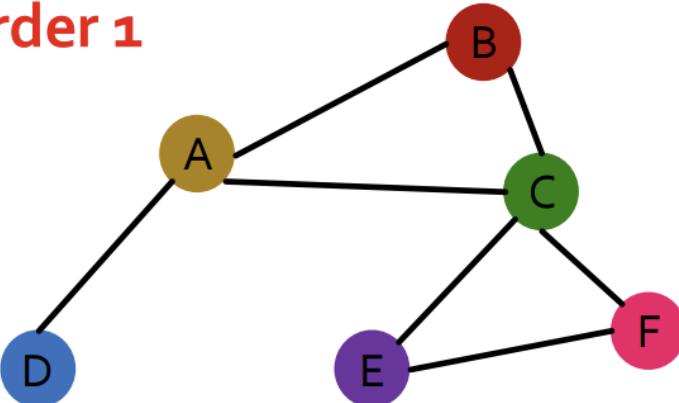
- $W^{(l)} \in \mathbb{R}^{F_l \times F_{l+1}}$
 - l번째 GCN layer의 weight matrix
 - feature 변환 역할

$$\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$$

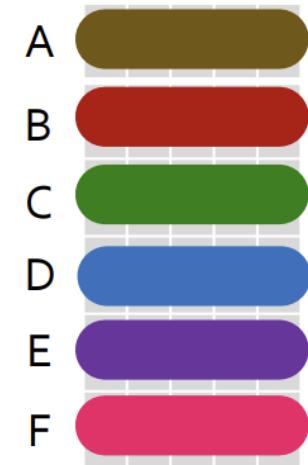
- $\hat{\mathbf{A}}$: normalized A
- σ : non-linear function

Graph Neural Network: Permutation

Order 1



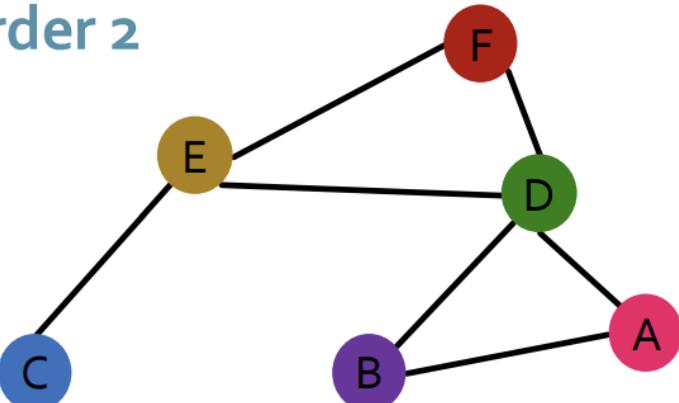
Node features X_1



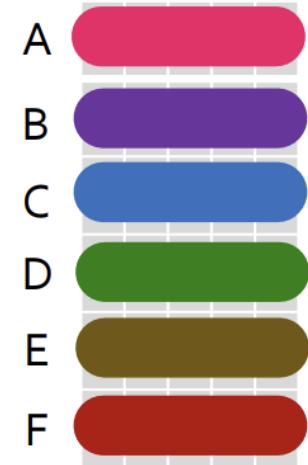
Adjacency matrix A_1

	A	B	C	D	E	F
A	Gray	Blue	Gray	Gray	Gray	Gray
B	Blue	Gray	Blue	Gray	Gray	Gray
C	Gray	Blue	Gray	Blue	Gray	Blue
D	Gray	Gray	Blue	Gray	Gray	Gray
E	Gray	Gray	Gray	Blue	Gray	Blue
F	Gray	Gray	Gray	Gray	Blue	Gray

Order 2



Node features X_2



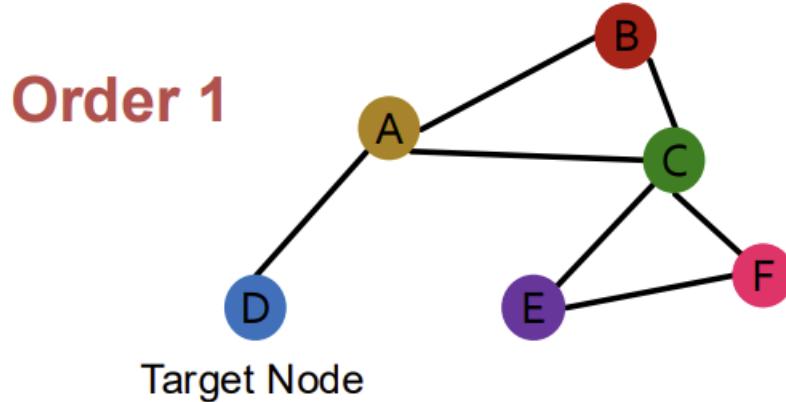
Adjacency matrix A_2

	A	B	C	D	E	F
A	Gray	Blue	Gray	Gray	Gray	Gray
B	Blue	Gray	Gray	Blue	Gray	Gray
C	Gray	Gray	Gray	Blue	Gray	Blue
D	Gray	Blue	Gray	Gray	Gray	Gray
E	Gray	Gray	Gray	Gray	Blue	Blue
F	Gray	Gray	Gray	Gray	Blue	Gray

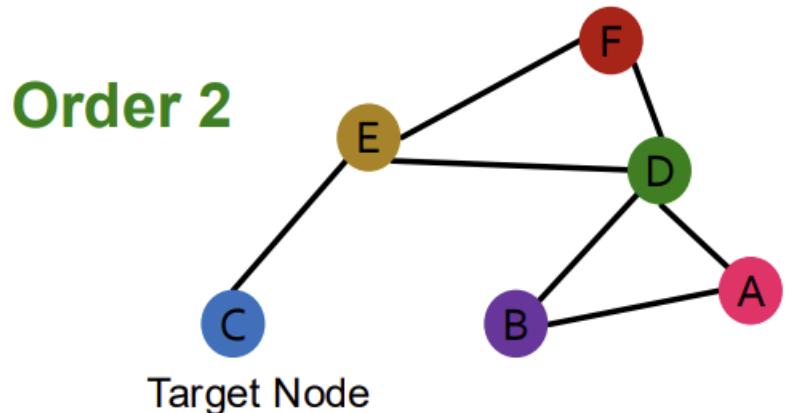
Graph Neural Network: Permutation

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

Order 1



Order 2



Node feature X_1

A	
B	
C	
D	
E	
F	

Adjacency matrix A_1

A	
B	
C	
D	
E	
F	

Embeddings H_1

A	
B	
C	
D	
E	
F	

Permute the input, the output also permutes accordingly - permutation equivariant

Node feature X_2

A	
B	
C	
D	
E	
F	

Adjacency matrix A_2

A	
B	
C	
D	
E	
F	

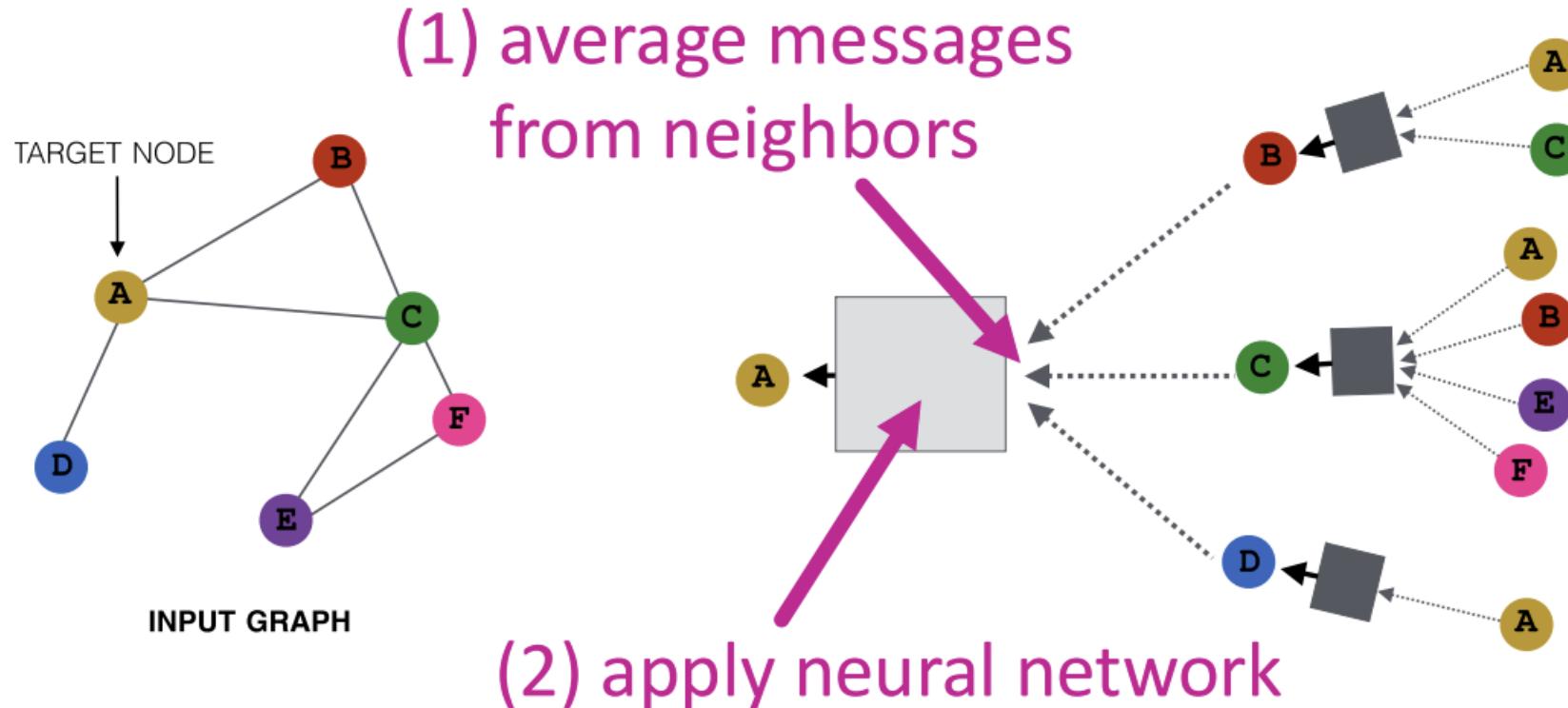
Embeddings H_2

A	
B	
C	
D	
E	
F	

Graph Convolution Network (GCN)

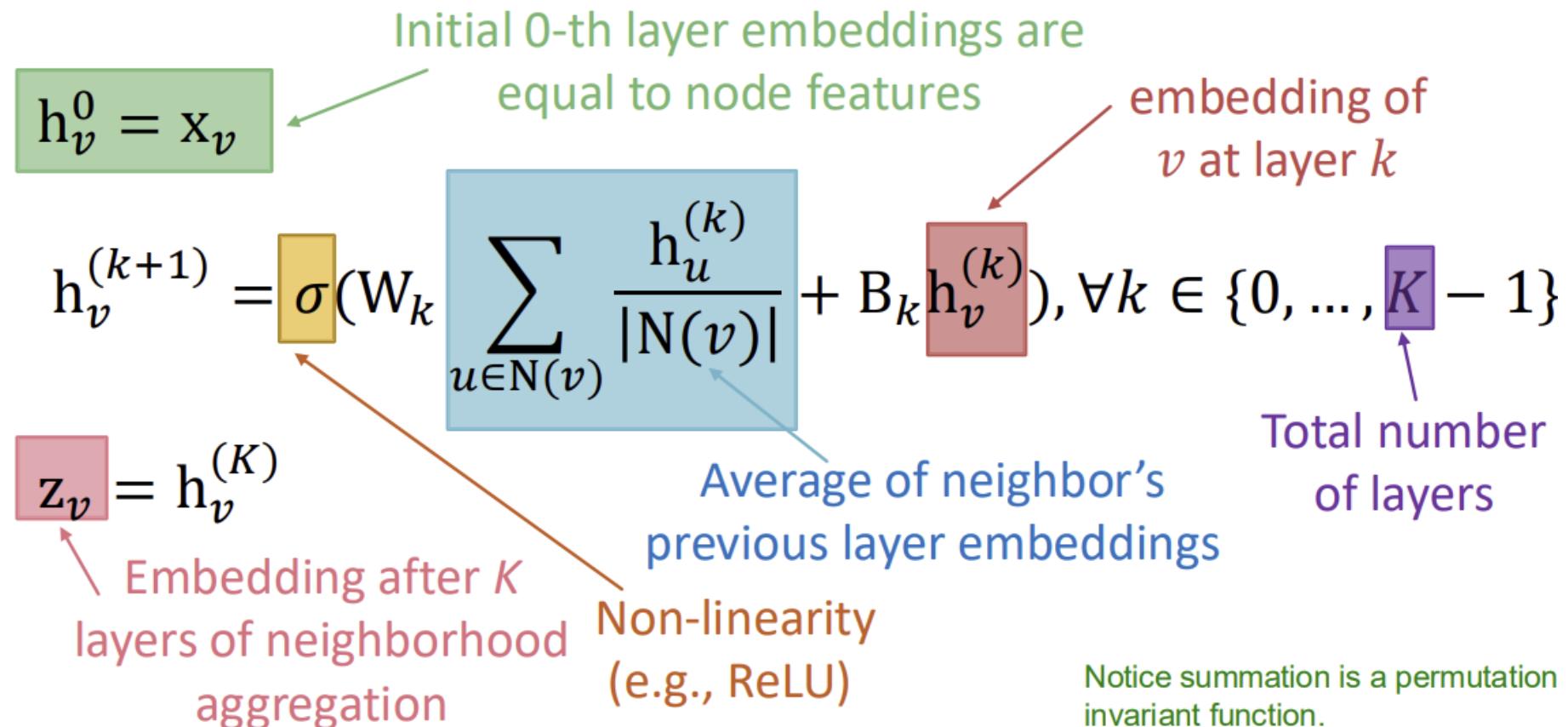
Graph Convolution Network (GCN); ICLR 2017

- **Basic approach:** Average information from neighbors and apply a neural network



Graph Convolution Network (GCN)

Graph Convolution Network (GCN); ICLR 2017



Graph Convolution Network (GCN)

Graph Convolution Network (GCN); ICLR 2017

$$\begin{aligned} h_v^{(0)} &= x_v \\ h_v^{(k+1)} &= \sigma(W_k \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} + B_k h_v^{(k)}), \forall k \in \{0..K-1\} \\ z_v &= h_v^{(K)} \end{aligned}$$

Trainable weight matrices
(i.e., what we learn)

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^k : the 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

Graph Convolution Network (GCN)

Graph Convolution Network (GCN); ICLR 2017

- Many aggregations can be performed efficiently by (sparse) matrix operations

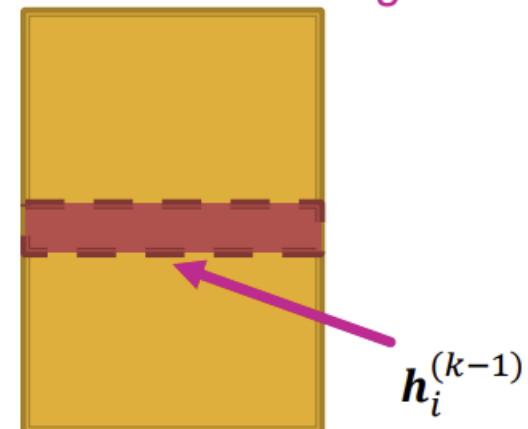
- Let $H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^T$
- Then: $\sum_{u \in N_v} h_u^{(k)} = A_{v,:} H^{(k)}$
- Let D be diagonal matrix where $D_{v,v} = \text{Deg}(v) = |N(v)|$
 - The inverse of D : D^{-1} is also diagonal:
 $D_{v,v}^{-1} = 1/|N(v)|$
- Therefore,

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



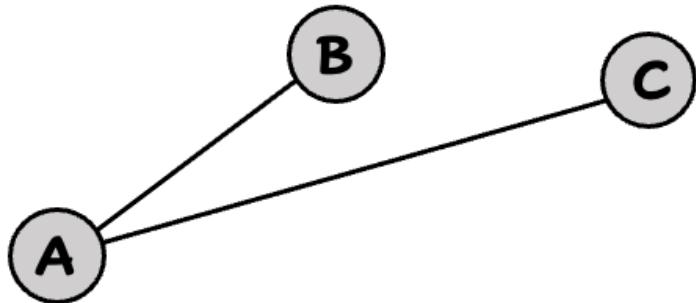
$$H^{(k+1)} = D^{-1} A H^{(k)}$$

Matrix of hidden embeddings $H^{(k-1)}$



Graph Convolution Network (GCN)

Graph Convolution Network (GCN); ICLR 2017



$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$A + I = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \hat{A}$$

$$H = \hat{A}^T X W^T$$

$$D + I = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} = \hat{D}$$

$$\text{normalized } \hat{A} = \hat{D}^{-\frac{1}{2}} \hat{A}^T \hat{D}^{-\frac{1}{2}}$$

$$\hat{D}^{-1} = \begin{pmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$$

$$H = \hat{D}^{-\frac{1}{2}} \hat{A}^T \hat{D}^{-\frac{1}{2}} X W^T$$

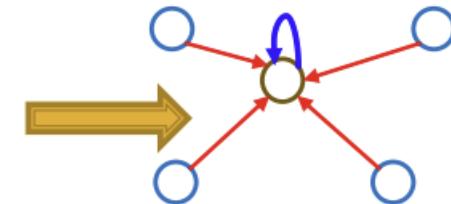
Graph Convolution Network (GCN)

Graph Convolution Network (GCN); ICLR 2017

- Re-writing update function in matrix form:

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

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

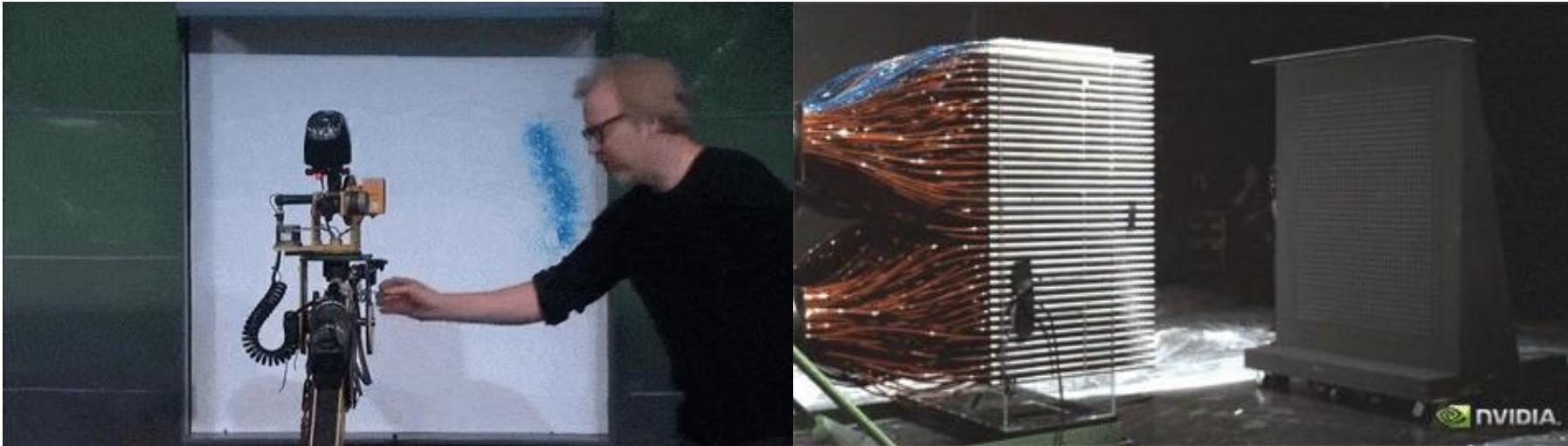


$$H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^T$$

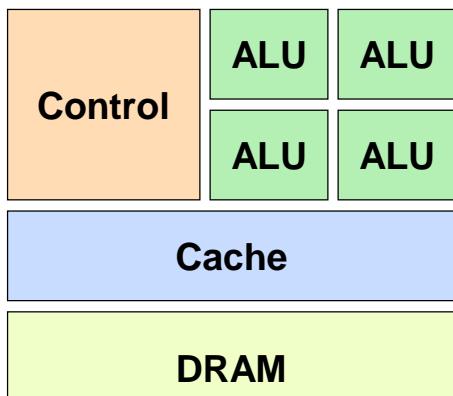
- Red: neighborhood aggregation
- Blue: self transformation
- In practice, this implies that efficient sparse matrix multiplication can be used (\tilde{A} is sparse)
- Note: not all GNNs can be expressed in a simple matrix form, when aggregation function is complex

Graph Convolution Network (GCN)

Matrix Multiplication & GPU

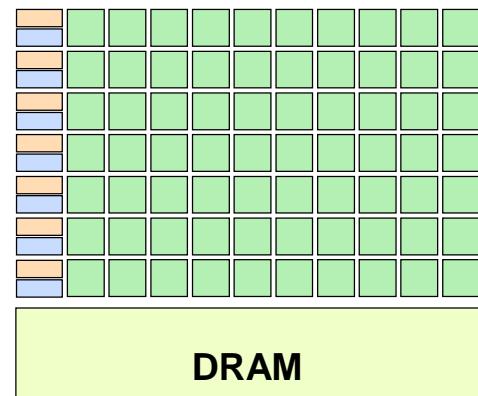


CPU

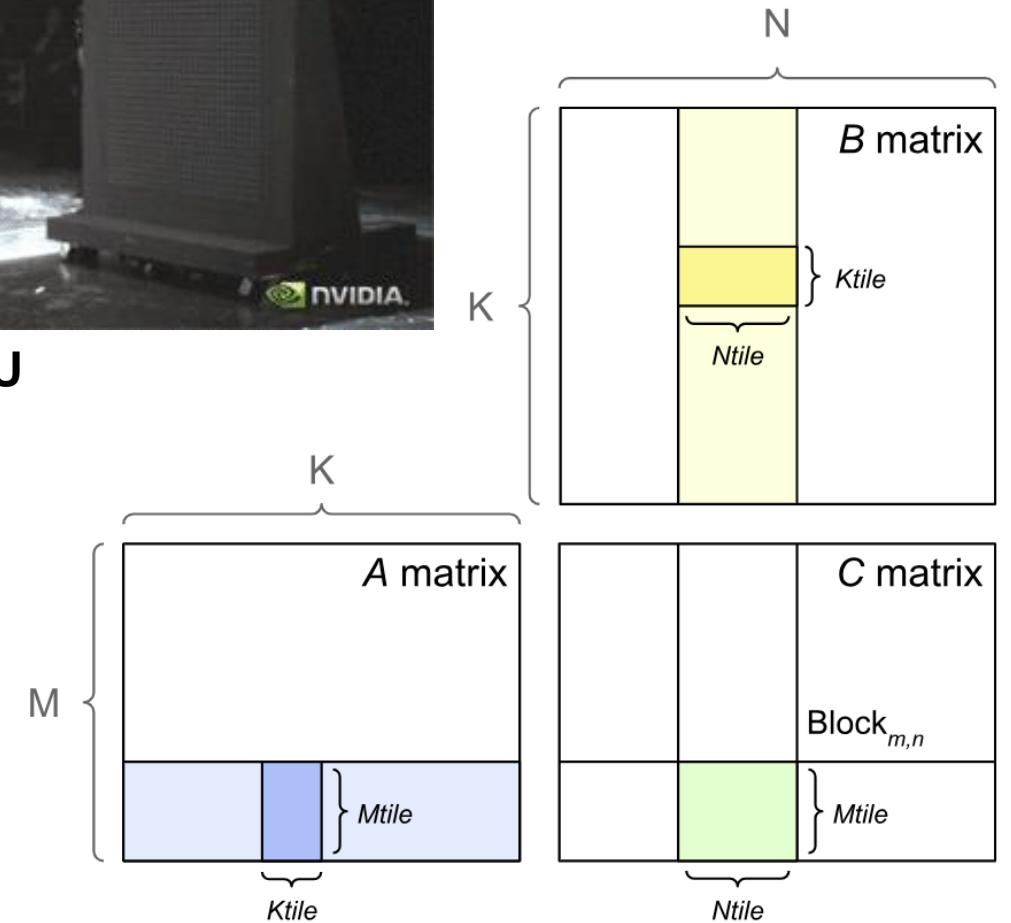


CPU

GPU



GPU



Graph Convolution Network (GCN)

Graph Convolution Network (GCN); ICLR 2017

- Node embedding \mathbf{z}_v is a function of input graph
- **Supervised setting:** We want to minimize loss \mathcal{L} :

$$\min_{\Theta} \mathcal{L}(y, f_{\Theta}(\mathbf{z}_v))$$

- y : node label
- \mathcal{L} could be L2 if y is real number, or cross entropy if y is categorical
- **Unsupervised setting:**
 - No node label available
 - **Use the graph structure as the supervision!**

Standard GNN Pipeline:

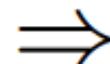
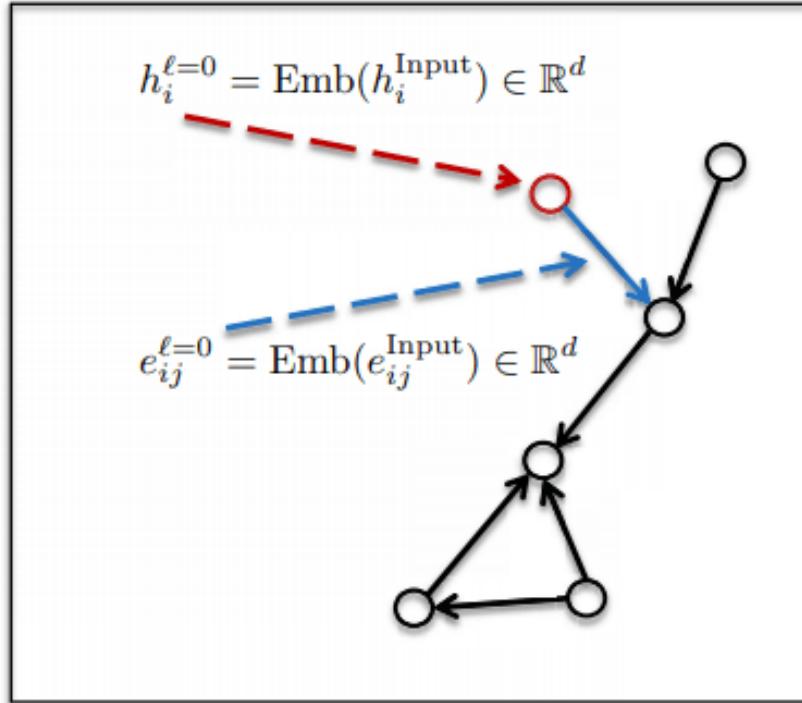


- Input Layer: Linear embedding of input node/edge features.
- GNN Layers: Apply favorite GNN layer L times.
- Task-based layer : Graph/node/edge prediction layer.

Graph Neural Network

- Input layer :

h^{Input}
 e^{Input}



$h^{\ell=0} \in \mathbb{R}^{n \times d}$
 $e^{\ell=0} \in \mathbb{R}^{E \times d}$

Input node/edge
features

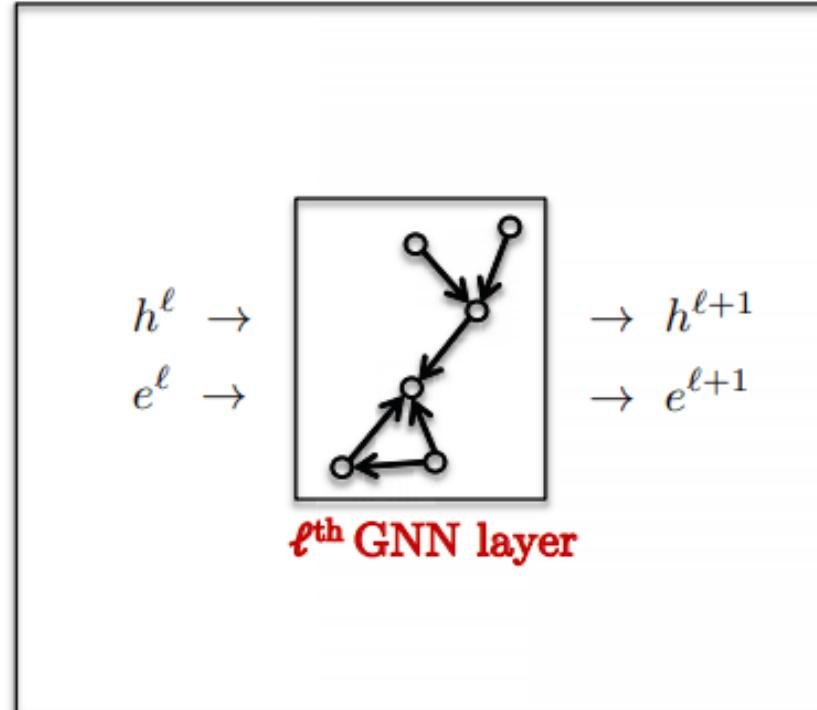
Embedding layer of
input features

Input features
embedded into d -dim
spaces, and passes to
the GNN layers.

Graph Neural Network

- GNN layers :

$$\begin{aligned} h^{\ell=0} &\in \mathbb{R}^{n \times d} \\ e^{\ell=0} &\in \mathbb{R}^{E \times d} \end{aligned} \quad \Rightarrow$$



$$\Rightarrow \begin{aligned} h^{\ell=L} &\in \mathbb{R}^{n \times d} \\ e^{\ell=L} &\in \mathbb{R}^{E \times d} \end{aligned}$$

Input of GNNs
indexed by $\ell=0$.

GNN layers applied
 L times

Output of GNNs
indexed by $\ell=L$.

Graph Neural Network

- Task-based layer

- Graph-level prediction :

$$h^G = \frac{1}{n} \sum_{i=0}^n h_i^{\ell=L} \in \mathbb{R}^d \Rightarrow \boxed{\text{MLP}} \Rightarrow \text{score} \in \begin{cases} \mathbb{R} & \text{regression} \\ \mathbb{R}^K, K > 1 & \text{classification} \end{cases}$$

- Node-level prediction :

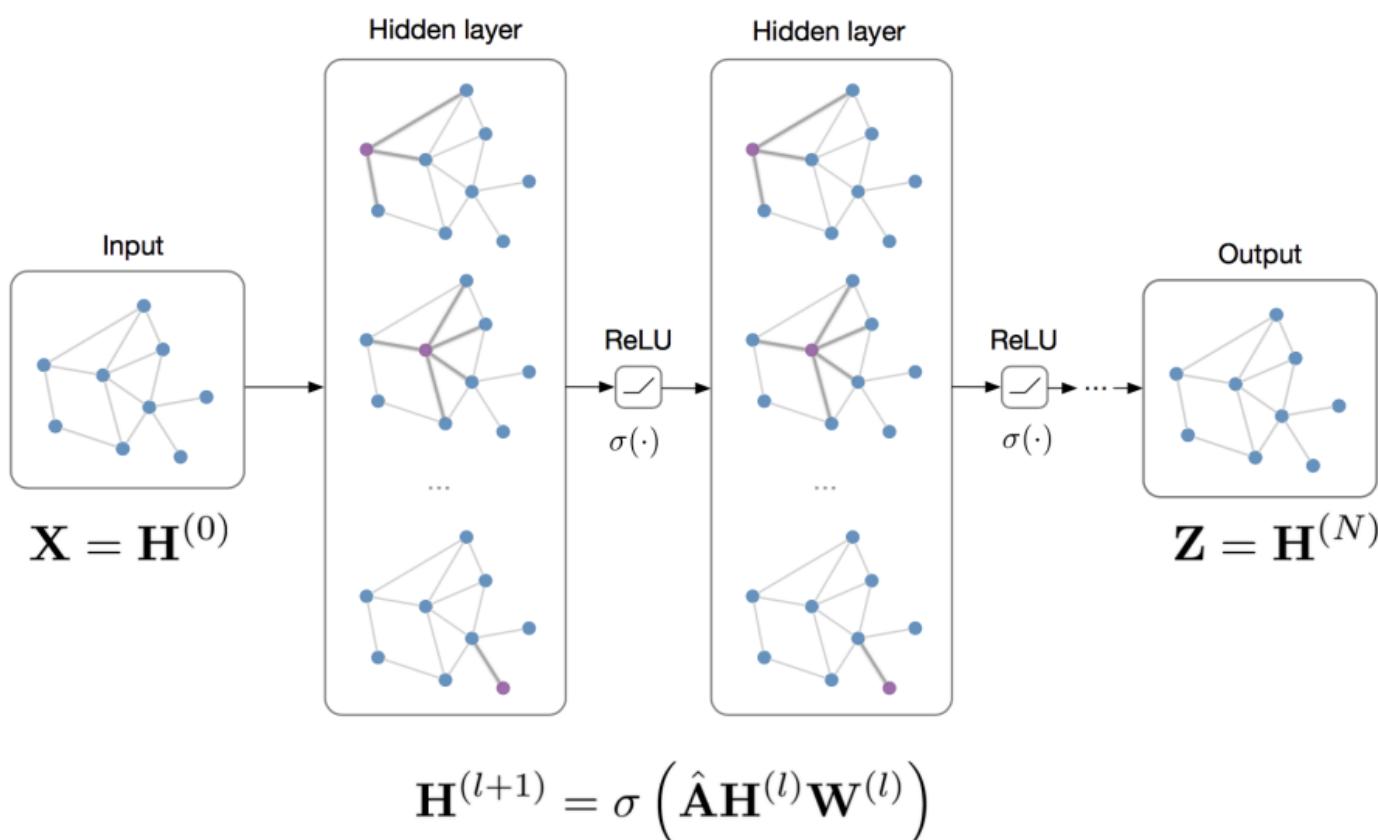
$$h_i^{\ell=L} \in \mathbb{R}^d \Rightarrow \boxed{\text{MLP}} \Rightarrow \text{score}_i \in \begin{cases} \mathbb{R} & \text{regression} \\ \mathbb{R}^K, K > 1 & \text{classification} \end{cases}$$

- Edge-level prediction :

$$e_{ij}^{\text{link}} = \text{Concat}(h_i^{\ell=L}, h_j^{\ell=L}) \in \mathbb{R}^d \Rightarrow \boxed{\text{MLP}} \Rightarrow \text{score}_{ij} \in \begin{cases} \mathbb{R} & \text{regression} \\ \mathbb{R}^K, K > 1 & \text{classification} \end{cases}$$

Classification and Link Prediction with GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



Node classification:

$$\text{softmax}(\mathbf{z}_n)$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$\text{softmax}(\sum_n \mathbf{z}_n)$$

e.g. Duvenaud et al. (NIPS 2015)

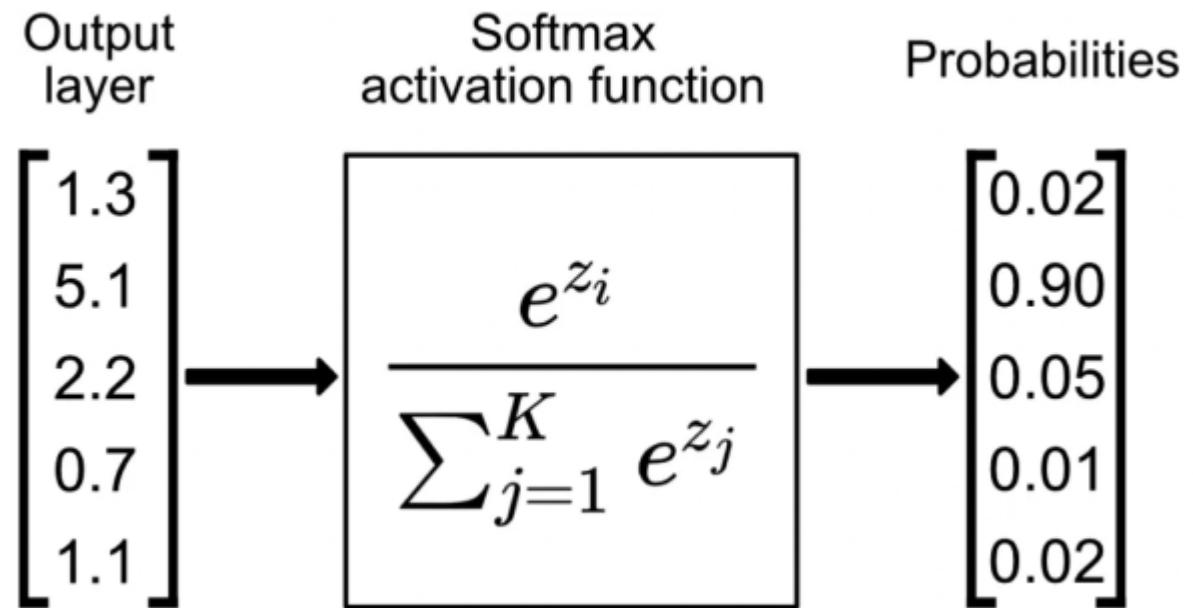
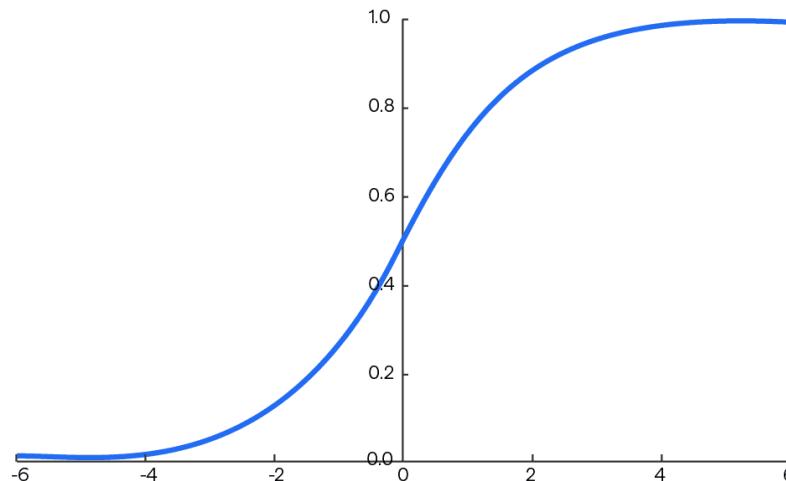
Link prediction:

$$p(A_{ij}) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$$

Kipf & Welling (NIPS BDL 2016)

“Graph Auto-Encoders”

Softmax Function



Graph Attention Network (GAT)

Graph Attention Network (GAT); ICLR 2018

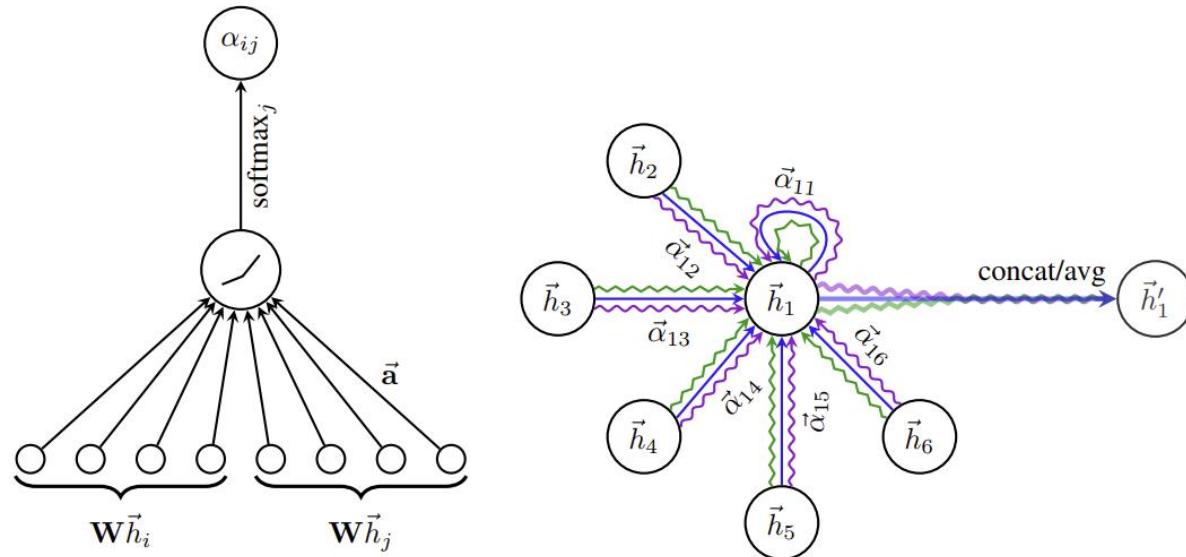


Table 2: Summary of results in terms of classification accuracies, for Cora, Citeseer and Pubmed. GCN-64* corresponds to the best GCN result computing 64 hidden features (using ReLU or ELU).

Method	Transductive		
	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	79.0%
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	79.0 ± 0.3%
GAT (ours)	83.0 ± 0.7%	72.5 ± 0.7%	79.0 ± 0.3%

$$\vec{h}'_i = \sigma \left(\frac{1}{K} \sum_{k=1}^K \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j \right)$$

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

$$\alpha_{ij} = \frac{\exp \left(\text{LeakyReLU} \left(\vec{a}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j] \right) \right)}{\sum_{k \in \mathcal{N}_i} \exp \left(\text{LeakyReLU} \left(\vec{a}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k] \right) \right)}$$

In GCN / GraphSAGE

- $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u 's message to node v
- $\Rightarrow \alpha_{vu}$ is defined **explicitly** based on the **structural properties of the graph (node degree)**
- \Rightarrow All neighbors $u \in N(v)$ are **equally important** to node v

Graph Isomorphism Networks (GIN)

Graph Isomorphism Networks (GIN); ICLR 2019

$$h_i^{(l+1)} = \text{MLP}^{(l)} \left((1 + \epsilon^{(l)}) h_i^{(l)} + \sum_{j \in \mathcal{N}(i)} h_j^{(l)} \right)$$

$$h_i^{(l+1)} = \phi^{(l)} \left(h_i^{(l)}, \ \square_{j \in \mathcal{N}(i)} \psi^{(l)}(h_i^{(l)}, h_j^{(l)}) \right)$$

모델	\square	ψ	ϕ
GCN	sum	고정 계수 $\times Wh_j$	σ
GIN	sum	h_j	MLP
GAT	weighted sum	attention $\times Wh_j$	σ

Graph Representation of Molecules

- SMILES Representation
- RDKit

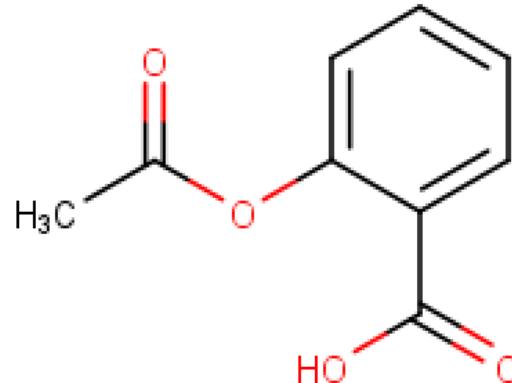
SMILES Representation

SMILES (Simplified Molecular Input Line Entry System)

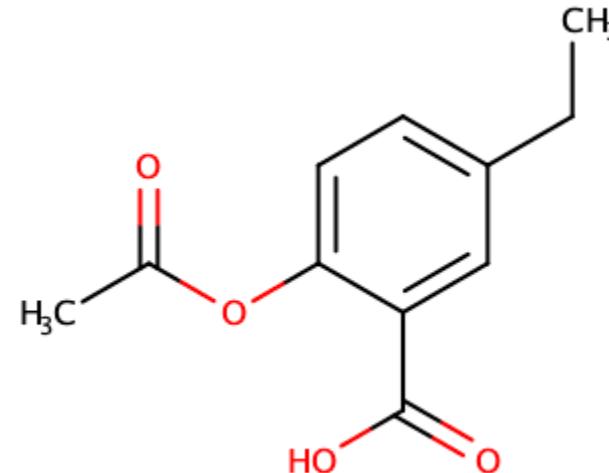
분자 구조를 ‘문자열 형태’로 표현하는 표기법.

Example)

Aspirin (or O-Acetylsalicylic acid)



CC(=O)OC1=CC=CC=C1C(O)=O



CCC1=CC=C(OC(C)=O)C(=C1)C(O)=O

SMILES Representation

SMILES를 구성하는 5가지 요소

1) 원자 (atom)

각 원소는 고유 기호로 표기됩니다.

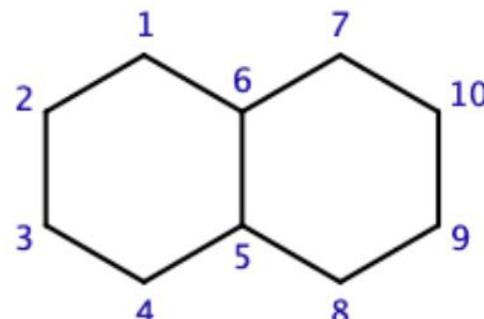
단, 표기를 간단하게 하기 위해 수소는 생략됩니다.

3) 고리 (ring)

고리 구조는 고리가 시작되는 원자를 기준으로 반시계 방향으로 돌아가며 표기합니다.

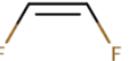
이때, 고리의 시작 원자와 마지막 원자에는 동일한 숫자를 표시하여 고리의 연결을 나타냅니다.

ex) C1CCCC2C1CCCC2



2) 결합 (bond)

.	결합되지 않음	\$	사중결합
-	단일결합	:	방향족결합
=	이중결합	/	입체화학 표시
#	삼중결합	\	입체화학 표시

분자	SMILES
$\text{H}_2\text{C}=\text{CH}_2$	C=C
$\text{H}_3\text{C}-\text{Br}$	CBr
$\text{HC}\equiv\text{N}$	C#N
	F/C=C\F

SMILES Representation

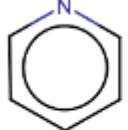
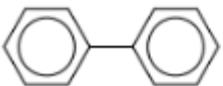
4) 방향족 (aromaticity)

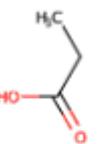
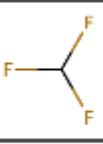
벤젠과 같은 방향족 고리들을 아래와 같이 표현할 수 있습니다.

i) C1=CC=CC=C1 → 단일결합과 이중결합이 교대로 일어나는 Kekule form

ii) C:1:C:C:C:C1 → 방향족결합 기호(:)를 사용

iii) c1ccccc1 → 방향족 고리에 포함된 B, C, N, O, P, S 원자를 소문자로 표시합니다.

분자	SMILES
	n1ccccc1 (pyridine)
	o1ccccc1 (furan)
	c1ccccc1-c2ccccc2 (biphenyl)

분자	SMILES
	CCC(=O)O (propionic acid)
	FC(F)F (fluoroform)
	FC(Br)(Cl)F, BrC(F)(F)Cl, C(F)(Cl)(F)Br, ...

5) 가지 (branch)

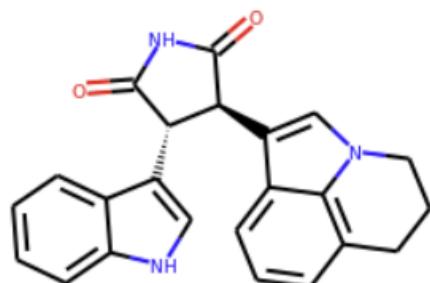
분자의 가지는 괄호()로 표현합니다.

가지는 다양한 순서로 나타낼 수 있으며, 결합 역시 괄호 안에 포함되어야 합니다.

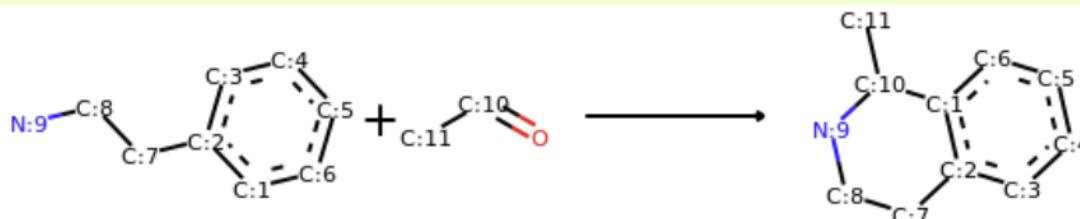
RDKit: 파이썬 라이브러리를 활용하는 분자식 표현 오픈소스 프로그램

```
from rdkit import Chem
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import Draw
IPythonConsole.ipython_useSVG=True #< set this to False if you want PNGs instead of SVGs
```

```
# Test in a kinase inhibitor
mol = Chem.MolFromSmiles("C1CC2=C3C(=CC=C2)C(=CN3C1)[C@H]4[C@@H](C(=O)NC4=O)C5=CNC6=CC=CC=C65")
# Default
mol
```



```
# Pictet-Spengler rxn
rxn = AllChem.ReactionFromSmarts('[cH1:1]1:[c:2](-[CH2:7]-[CH2:8]-[NH2:9]):[c:3]:[c:4]:[c:5]:[c:6]:1.[#6:11]-
[CH1;R0:10]=[OD1]>>[c:1]12:[c:2](-[CH2:7]-[CH2:8]-[NH1:9]-[C:10]-2(-[#6:11])):[c:3]:[c:4]:[c:5]:[c:6]:1')
```



Open-Source Cheminformatics
and Machine Learning

- <https://www.rdkit.org/>
- <https://www.rdkit.org/docs/Cookbook.html>

Training GNN models

- Implementation of GCN model
- Paper Review: MolCLR
- Pre-Training / Fine-Tuning (Transfer Learning)
- Self-Supervised Learning

Implementation of GCN

Train your own GNN model

In the first glimpse of PyG, we implement the training of a GNN for classifying papers in a citation graph. For this, we load the [Cora](#) dataset, and create a simple 2-layer GCN model using the pre-defined [GCNConv](#):

```
import torch
from torch import Tensor
from torch_geometric.nn import GCNConv
from torch_geometric.datasets import Planetoid

dataset = Planetoid(root='.', name='Cora')

class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super().__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, out_channels)

    def forward(self, x: Tensor, edge_index: Tensor) -> Tensor:
        # x: Node feature matrix of shape [num_nodes, in_channels]
        # edge_index: Graph connectivity matrix of shape [2, num_edges]
        x = self.conv1(x, edge_index).relu()
        x = self.conv2(x, edge_index)
        return x

model = GCN(dataset.num_features, 16, dataset.num_classes)
```

Implementation of GCN

▼ We can now optimize the model in a training loop, similar to the [standard PyTorch training procedure](#).

```
import torch.nn.functional as F

data = dataset[0]
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

for epoch in range(200):
    pred = model(data.x, data.edge_index)
    loss = F.cross_entropy(pred[data.train_mask], data.y[data.train_mask])

    # Backpropagation
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

More information about evaluating final model performance can be found in the corresponding [example](#).

Implementation of GCN



The screenshot shows a browser window displaying the PyTorch Geometric documentation for the `GCNConv` module. The URL in the address bar is `pytorch-geometric.readthedocs.io/en/2.5.2/generated/torch_geometric.nn.conv.GCNConv.html`. The page title is `conv.GCNConv`. On the left, there is a sidebar with navigation links for `INSTALL PYG`, `GET STARTED`, `TUTORIALS`, and `ADVANCED CONCEPTS`. The main content area contains the class definition for `GCNConv`, which inherits from `MessagePassing`. It describes the graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper. The formula for the operator is given as $\mathbf{X}' = \hat{\mathbf{D}}^{-1/2} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-1/2} \mathbf{X} \Theta$, where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with inserted self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ its diagonal degree matrix. The adjacency matrix can include other values than 1 representing edge weights via the optional `edge_weight` tensor. Its node-wise formulation is given by:

$$\mathbf{x}'_i = \Theta^\top \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j$$

with $\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$, where $e_{j,i}$ denotes the edge weight from source node `j` to target node `i` (default: `1.0`)

PARAMETERS:

- `in_channels (int)` – Size of each input sample, or `-1` to derive the size from the first input(s) to the forward method.
- `out_channels (int)` – Size of each output sample.

Paper Review: MolCLR

MolCLR; NMI 2022

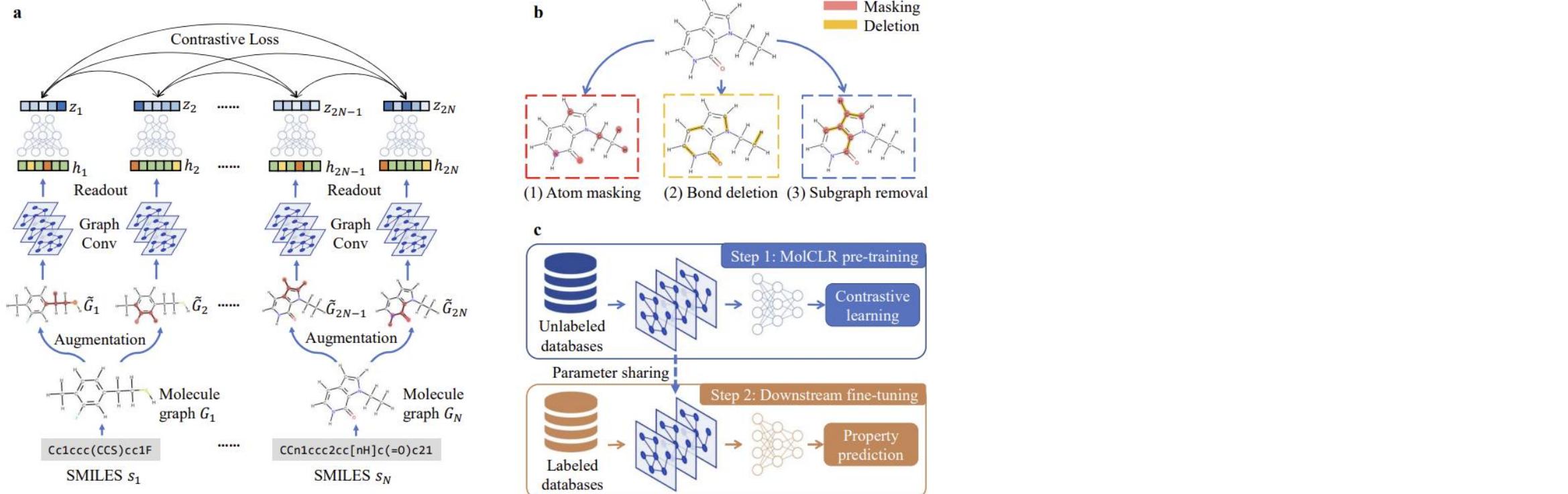


Figure 1. Overview of MolCLR. (a) MolCLR pre-training: A SMILES s_n from a mini-batch of N molecule data is converted to a molecule graph G_n . Two stochastic molecule graph data augmentation operators are applied to each graph, resulting two correlated masked graphs: \tilde{G}_{2n-1} and \tilde{G}_{2n} . A base feature encoder built upon graph convolutions and the readout operation extracts the representation h_{2n-1}, h_{2n} . Contrastive loss is utilized to maximize agreement between the latent vectors z_{2n-1}, z_{2n} from the MLP projection head. (b) Molecule graph augmentation strategies: atom masking, bond deletion, and subgraph removal. (c) The whole MolCLR framework: GNNs are first pre-trained via MolCLR to learn representative features. Fine-tuning for downstream molecular property predictions shares the pre-trained parameters of the GNN encoder and randomly initializes an MLP head. It then follows the supervised learning to train the model.

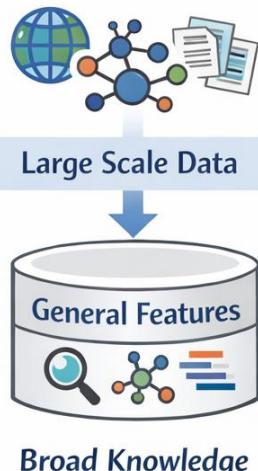
$Cc1ccc(CC)c1F$
SMILES s_1 SMILES s_N

Pre-Training / Fine-Tuning (Transfer Learning)

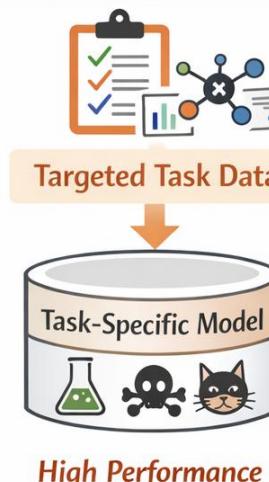
Pre-Training (사전학습)

- 대규모 데이터에서 모델이 일반적인 표현 (Representation)을 학습
- 레이블이 없거나 약한 경우가 많음 (Self-Supervised / Contrastive Learning 등)
- 목적: 문제 일반에 유용한 feature extractor 학습
- 예시:
 - 분자 ML: 대량의 SMILES/그래프로 분자 표현 학습
 - NLP: 대규모 텍스트로 언어 구조 학습

Pre-Training (사전학습)



Fine-Tuning (미세조정)

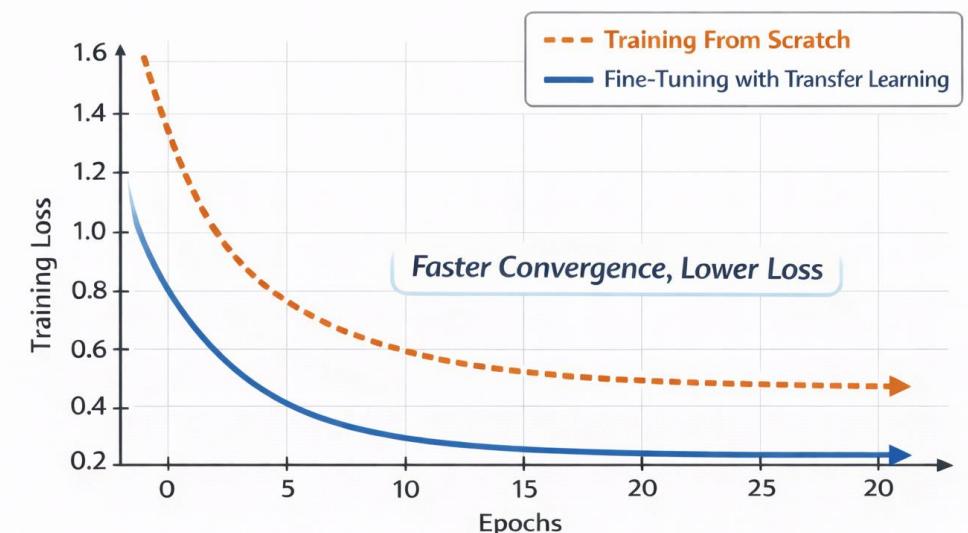


Fine-Tuning (미세조정)

- 작은 규모의 타깃 데이터에 대해 모델을 문제 특화로 조정
- Pre-trained 모델의 파라미터를 초기값으로 사용
- 목적: 특정 태스크 성능 극대화
- 예시:
 - 분자 ML: 독성 예측, HOMO–LUMO gap 예측 등
 - 이미지: 특정 클래스 분류

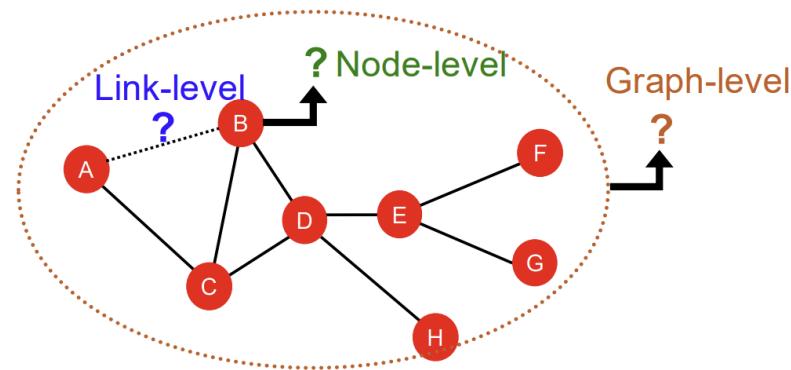
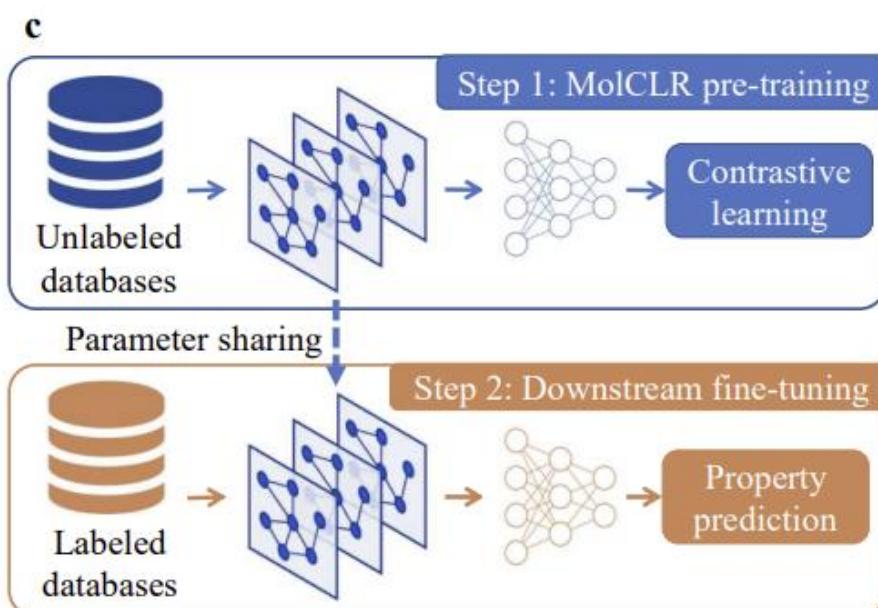
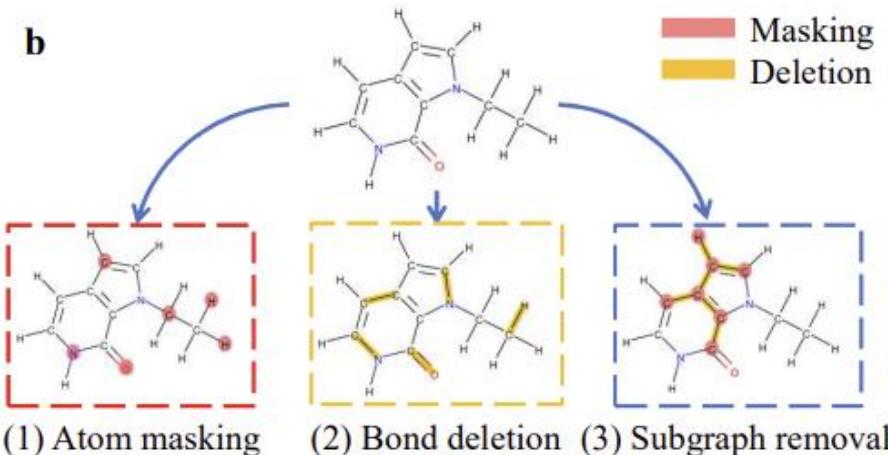
왜 Transfer Learning을 쓰는가?

- 작은 데이터에서도 높은 성능
- 학습 시간 단축
- 일반화 성능 향상



Self-Supervised Learning

MoICLR; NMI 2022



Self-Supervised Learning이란?

- 레이블이 없는 데이터(x only)로부터 학습 목표(pseudo-label)를 만들어 학습하는 방법
- 목적: 일반적이고 재사용 가능한 표현(Representation) 학습
↳ “Label 없이 supervised learning처럼 학습한다”

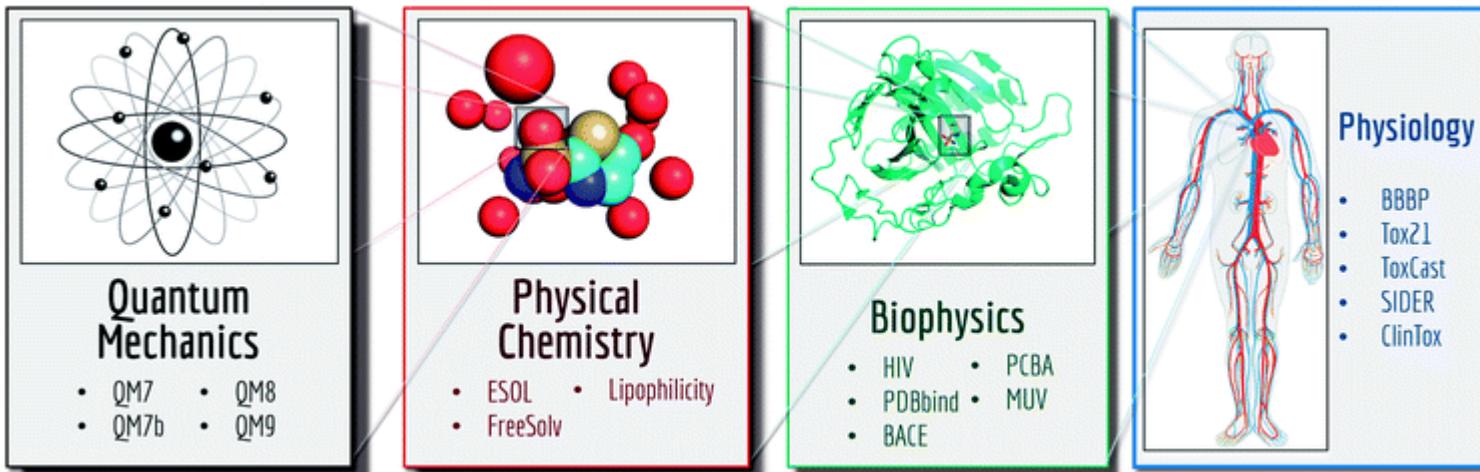
Contrastive Learning

- 같은 데이터의 다른 view는 가깝게
- 다른 데이터는 멀게

$$\mathcal{L}_{\text{contrastive}} = -\log \frac{\exp(\text{sim}(z_i, z_i^+)/\tau)}{\sum_j \exp(\text{sim}(z_i, z_j)/\tau)}$$

MoleculeNet: “분자 머신러닝 모델을 공정하게 비교하기 위한 표준 벤치마크 모음”
(Wu et al., MoleculeNet, Chem. Sci. 2018)

- 입력: **SMILES**
- 출력: **물성(property)**
- Task: regression 또는 classification



1) Classification Datasets (Metric: ROC-AUC)

Dataset	# Molecules	Prediction Target	Chemical / Practical Meaning
BBBP	~2,039	Blood–Brain Barrier permeability	야루시 혈뇌장벽(BBB)을 통과하는 이온의 에너지(IONIC) 약물 서계
BACE	~1,513	β -secretase 1 inhibition	
ClinTox	~1,478	Clinical toxicity / FDA approval	
SIDER	~1,427	Drug side effects (multiple targets)	
HIV	~41,127	Anti-HIV activity	

2) Regression Datasets

Dataset	# Molecules	Prediction Target	Chemical / Physical Meaning
ESOL	~1,128	Aqueous solubility ($\log S$)	분자의 물에 대한 용해도 (약물 흡수/분포에 중요)
FreeSolv	~642	Hydration free energy	물 속에서의 자유에너지 변화 (용매화 특성)
Lipophilicity	~4,200	Octanol/water $\log D$	분자의 소수성/친수성 균형 (막 투과성, ADMET 관련)
QM9	~133,000	Quantum mechanical properties	HOMO/LUMO 에너지, dipole moment 등 양자화학적 성질

What is Hydration Free Energy (수화 자유에너지)?

분자를 진공 → 물로 옮길 때의 자유에너지 변화 (단위: kcal/mol).

물과의 친화도를 정량적으로 나타내는 물리량.

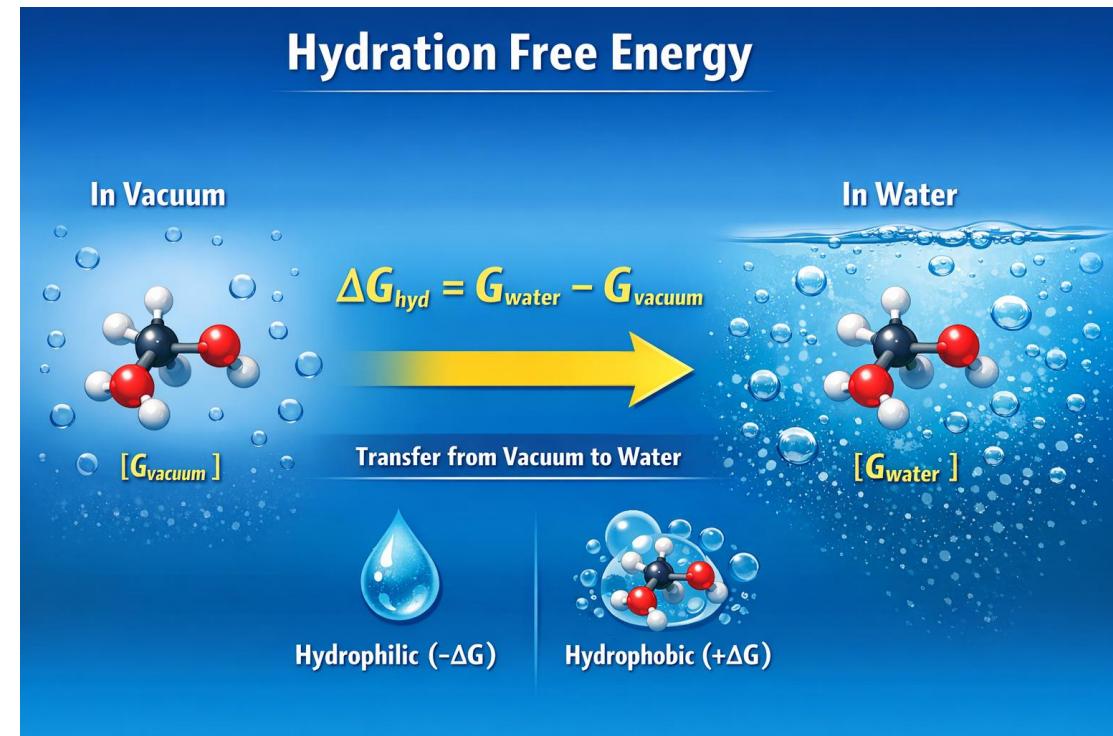
- 더 음수 → 더 친수성 (물에 잘 용해)
- 덜 음수 / 양수 → 소수성 (물에 잘 용해되지 않음)

Why is it important?

대부분의 화학·생체 과정은 물 속에서 일어남.

다음 물성들의 기준(reference) 역할:

- 용해도 (Solubility)
- 분배계수 ($\log P$ / $\log D$)
- 생체 이용성 (Bioavailability)



Why FreeSolv?

- 실험 기반 hydration free energy 데이터
- 소규모 데이터셋 (~640 molecules)
- 단순한 입력 (SMILES) vs 복합적인 물리 효과
 - 극성, 수소결합, 분자 크기

분자 representation의 중요성을 잘 드러내는 benchmark

	iupac	smiles	expt	calc
0	4-methoxy-N,N-dimethyl-benzamide	CN(C)C(=O)c1ccc(cc1)OC	-11.01	-9.625
1	methanesulfonyl chloride	CS(=O)(=O)Cl	-4.87	-6.219
2	3-methylbut-1-ene	CC(C)C=C	1.83	2.452
3	2-ethylpyrazine	CCc1cnccn1	-5.45	-5.809
4	heptan-1-ol	CCCCCCO	-4.21	-2.917

Github link: <https://github.com/jeheon1905/gnn-molecular-ml-tutorial>

```
# 코드 다운로드
git clone https://github.com/jeheon1905/gnn-molecular-ml-tutorial.git
cd gnn-molecular-ml-tutorial

# conda 환경 생성
conda create -y -n gnn-tutorial python=3.10

# 환경 활성화
conda activate gnn-tutorial

# PyTorch & PyG 설치
pip install torch==2.2.2 --index-url https://download.pytorch.org/whl/cpu
pip install "numpy<2.0"
pip install torch_geometric

# Dependencies 설치
conda install -y -c conda-forge rdkit seaborn
pip install scikit-learn==1.4.2 pandas matplotlib

# Jupyter kernel 등록
pip install jupyterlab ipykernel
python -m ipykernel install --user --name gnn-tutorial --display-name "Python (gnn-tutorial)"

# Jupyter lab 실행
jupyter lab notebooks/rdkit_molecular_graph_tutorial.ipynb
```

- CS224W: Machine Learning with Graphs
Stanford / Fall 2025
<https://web.stanford.edu/class/cs224w/>
- CSE 849: Deep Learning
Vishnu Boddeti
<https://hal.cse.msu.edu/teaching/2024-spring-deep-learning/>
- Topics in Artificial Intelligence (CPSC 532S): Multimodal Learning with Vision, Language and Sound
https://www.cs.ubc.ca/~lsigal/532S_2018W2/Lecture18a.pdf
- GNN101: Visual Learning of Graph Neural Networks in Your Web Browser
<https://visual-intelligence-umn.github.io/GNN-101/>
https://www.youtube.com/watch?v=_0jXy4Zoh-o
- RDKit Cookbook
<https://www.rdkit.org/docs/Cookbook.html>
- PyG (PyTorch Geometric) example codes:
MoleculeNet (ESOL) GCN 학습 예제:
https://github.com/pyg-team/pytorch_geometric/blob/master/examples/attentive_fp.py