

Graph Representation

What Does It Mean to Represent a Graph?

그래프 표현은 서로 다른 세 가지 결정으로 나뉨

1. MODELING

모델링

현실의 어떤 객체와 관계를
그래프로 삼을지 정함

2. ENCODING

인코딩

연결 구조를 어떤 자료구조에
저장할지 정함

3. FEATURES

특징 구성

객체와 관계의 속성을 수치로 기록

$$G = (V, E)$$

V : 노드 집합 (객체)

E : 간선 집합 (관계)

속성이 있으면 다음이 추가됨

X : 노드 특징 행렬

F_E : 간선 특징

같은 연결 구조라도 관측 속성이 다를 수 있고, 같은 그래프를 여러 저장 형식으로 정확히 표현할 수 있음

여기서 representation은 반드시 학습된 embedding을 뜻하지 않음

From Real-World Data to a Graph

관측된 상호작용 기록

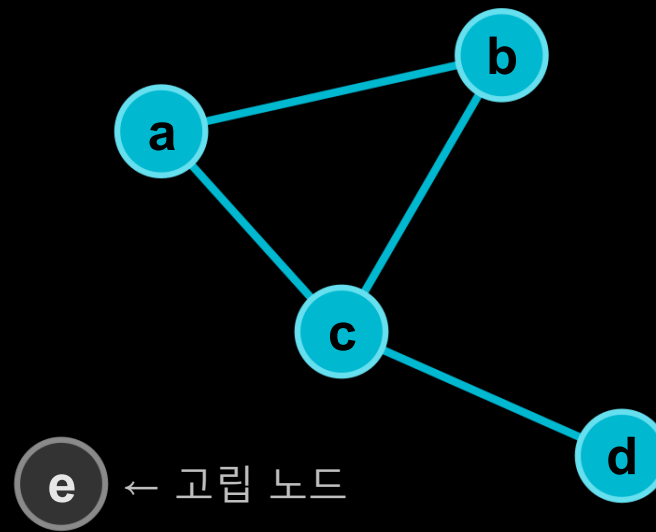
사용자 쌍	관측 횟수
a — b	2
a — c	1
b — c	3
c — d	1

사용자 목록 : a, b, c, d, e

상호작용이 한 번 이상 있으면 무방향 간선 하나를 만들

반복 기록은 간선 특징인 횟수로 집계하고, 활동량과 유형은 사용자 속성으로 분리
기록에 없는 e도 사용자 목록에 있으므로 노드로 남음

그래프 $G = (V, E)$



관측 기간과 관계 정의가 바뀌면 그래프도 바뀜

Choosing Nodes and Edges

같은 논문 데이터라도 무엇을 노드와 간선으로 정의하는지가 분석 질문을 결정

노드 = 논문, 간선 = 인용

지식의 참조 관계를 표현

방향성이 의미를 가짐

노드 = 연구자, 간선 = 공동 저술

협업 관계를 표현

무방향으로 두는 경우가 많음

노드 = 논문, 간선 = 제목의 공통 단어

어휘 공유 관계를 분석하게 됨

인용 관계가 아님

그래프를 만들기 전에 먼저 정해야 하는 것

노드의 단위

간선의 의미

방향

관측 기간

가중치의 의미

양방향 관계를 하나로 합치면 정보가 사라질 수 있음

Different Representations, Different Information

정보를 바꾸는 선택과 정보를 보존하는 변환을 구분

MODELING CHOICE

정보가 사라짐

인용 방향을 제거하면 누가 누구를 인용했는지 사라짐

횃수를 이진 연결로만 저장하면 빈도 차이가 사라짐

STORAGE FORMAT

정보가 보존됨

노드 목록 · 방향 · 가중치를 동일하게 유지

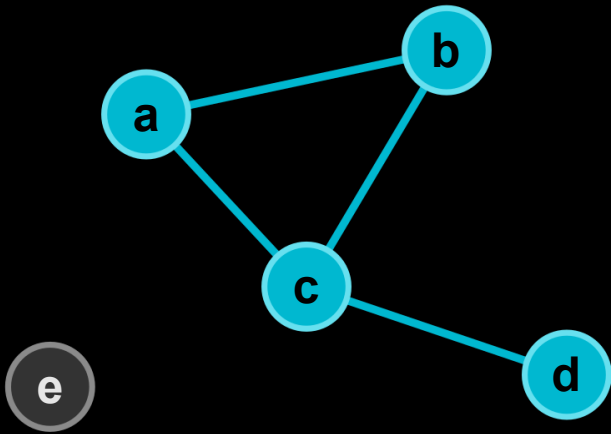
인접 행렬을 간선 목록으로 바꾸는 것은 저장 형식의 변경임

이진 표현으로는 간선에 부여된 값을 복원할 수 없음

어떤 정보를 보존했는가 ≠ 어떻게 저장했는가
두 질문은 별개로 답해야 함

One Graph, Multiple Representations

공통 예제 그래프



$$V = \{a, b, c, d, e\}$$

$$E = \{ab, ac, bc, cd\}$$

$$a=0 \quad b=1 \quad c=2 \quad d=3 \quad e=4$$

동일한 구조, 세 가지 표현

인접 행렬 (Adjacency Matrix)

5×5 행렬로 연결 여부를 기록

간선 목록 (Edge List)

존재하는 간선의 끝점 쌍만 저장

인접 리스트 (Adjacency List)

노드별 이웃 목록을 저장

인접 행렬의 행·열, 특징 행렬의 행, 간선의 정수 인덱스는 모두 같은 노드 ID 표현
고립 노드 e는 간선 목록만으로 확인할 수 없어 노드 집합 또는 노드 수를 따로 보존

이후 모든 변환에서 노드, 간선, 고립 노드 유지

Adjacency Matrix

$A_{ij} = 1$ 노드 i와 j가 연결

$A_{ij} = 0$ 연결되지 않음

행과 열의 노드 순서 : $[a, b, c, d, e]$

	a	b	c	d	e
a	0	1	1	0	0
b	1	0	1	0	0
c	1	1	0	1	0
d	0	0	1	0	0
e	0	0	0	0	0

$$A_{ab} = 1$$

a 행, b 열 → 연결됨

$$A_{ad} = 0$$

a 행, d 열 → 연결 없음

한 원소만 조회하면

연결 여부를 알 수 있음

$$A = A^T$$

무방향이므로 대칭

$$A_{ii} = 0$$

self-loop가 없음

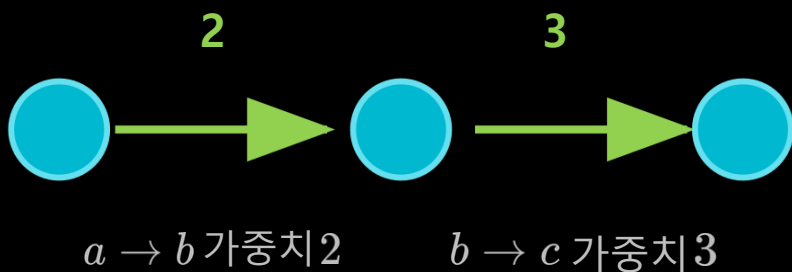
$$5 \times 5$$

e의 존재를 보존

노드 e의 행과 열이 모두 0 이어도 행렬 차원이 5이므로 고립 노드 정보는 유지

Directed and Weighted Adjacency Matrices

별도 예제 : 세 노드 방향 그래프



행·열 순서 $[a, b, c]$

	a	b	c
a	0	2	0
b	0	0	3
c	0	0	0

방향 그래프는 대칭일 필요가 없음

무방향 가중 그래프는
양방향 대응 원소가 같음

가중치가 거리인지 빈도인지
의미를 명시

주의 : 0 가중치를 실제 간선 값으로 허용하는 경우

원소가 0이라는 사실만으로는 간선의 부재를 구별할 수 없음

이때는 연결 여부를 담은 별도 정보가 필요

Reading Structure from the Adjacency Matrix

	a	b	c	d	e
a	0	1	1	0	0
b	1	0	1	0	0
c	1	1	0	1	0
d	0	0	1	0	0
e	0	0	0	0	0

모든 이웃 찾기: c의 행에서 1인 열을
 $N(c) = \{a, b, d\}$

고립 노드 찾기: e의 행과 열이 모두 0
고립 노드 (isolated)

삼각관계 찾기: a, b, c의 세 쌍이 모두 1
삼각형 (triangle)

대칭성 확인

무방향 표현임을 알 수 있음

대각 원소 확인

self-loop 유무를 알 수 있음

행렬의 모양이나 노드 나열 순서 자체에 추가적인 의미를 부여하지 않음

정의된 원소와 연결 관계에만 근거해 그래프 해석

Computing Degree from the Adjacency Matrix

$$d_i = \sum_j A_{ij}$$

$$d = A\mathbf{1}$$

단순 · 무방향 · 비가중 그래프 (self-loop 없음)에서 각 행의 합이 그 노드의 차수임

	a	b	c	d	e	차수
a	0	1	1	0	0	2
b	1	0	1	0	0	2
c	1	1	0	1	0	3
d	0	0	1	0	0	1
e	0	0	0	0	0	0

$$d = [2, 2, 3, 1, 0]$$

$$\sum_i d_i = 8 = 2m$$

방향 그래프에서 source 가 행이면 행 합 = out-degree, 열 합 = in-degree 임
가중치 행렬의 원소 합은 degree 가 아니라 가중 정보(strength information)임

Edge List

존재하는 간선의 끝점 쌍만 저장 (k 는 간선 번호, edge index를 의미)

노드 이름 표현

k	u	v
0	a	b
1	a	c
2	b	c
3	c	d

정수 ID 표현

k	u	v
0	0	1
1	0	2
2	1	2
3	2	3

$$m = 4$$

무방향 간선을 한 번씩 저장

$$s = 8$$

무방향에서 하나의 간선을
두 개의 방향 항목으로 저장할 때

고립 노드 e 는 간선 목록에 나타나지 않음

노드 목록을 별도로 유지해야 e 를 보존할 수 있음

Adjacency List

각 노드마다 이웃 목록을 연결해 저장

a : [*b*, *c*]

이웃 2개

b : [*a*, *c*]

이웃 2개

c : [*a*, *b*, *d*]

이웃 3개

d : [*c*]

이웃 1개

e : []

빈 목록으로 보존

장점

이웃 목록에 직접 접근할 수 있음

이웃 항목의 총수 $2m$

$$2 + 2 + 3 + 1 + 0 = 2m$$

방향 그래프는 나가는 이웃과 들어오는 이웃 중 무엇을 저장할지 정해야 함

Comparing Graph Representations

표현 방식	저장 공간	간선 존재 조회	이웃 전체 조회
dense 인접 행렬	$O(n^2)$	$O(1)$	$O(n)$
간선 목록 정렬되지 않음 · 노드 목록 포함	$O(n + m)$	$O(m)$	$O(m)$
인접 리스트 배열 기반	$O(n + m)$	$O(d_i)$	$O(d_i)$

Dense 행렬의 이웃 목록은 행 전체를 읽으므로 $O(n)$ 임

간선 목록에서 특정 노드의 이웃을 찾으려면 전체 목록을 탐색

전제 조건

무방향 그래프의 상수 배수는 생략한 복잡도임

해시 · 정렬 등 부가 구조를 쓰면 조회 비용이 바뀜

Why Real-World Graphs Are Sparse

희소성 = 가능한 모든 연결 중 실제 연결의 비율이 낮은 상태

가능한 최대 간선 수 (self-loop 없음)

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

밀도 (density)

$$\frac{2m}{n(n-1)}$$

밀도 = 가능한 전체 간선 중 실제 간선이 얼마나 존재하는가

$$\text{Density} = \frac{m}{n(n-1)/2} = \frac{2m}{n(n-1)}$$

density = 1

완전 그래프 (complete graph)

density $\rightarrow 0$

희소 그래프 (sparse graph)

SNS 사용자 10만 명이라도 한 사람이 나머지 99,999명 모두와 친구는 아님
논문 인용에서도 모든 논문이 다른 모든 논문을 인용하지는 않음

다만 모든 현실 그래프가 희소하다고 단정하지는 않음

Density: A Worked Example

노드 10만 개 · 평균 차수 20 인 그래프의 밀도를 단계별로 계산

노드 수 n

100,000

평균 차수 $\bar{d}$

20

간선 수 m

1,000,000

밀도

0.02 %

① Handshake Lemma

$$\sum_i d_i = 2m \Rightarrow n\bar{d} = 2m$$

② 실제 간선 수

$$m = \frac{n\bar{d}}{2} = \frac{100,000 \times 20}{2} = 1,000,000$$

③ 가능한 간선 수

$$\frac{n(n-1)}{2} = \frac{100,000 \times 99,999}{2} \approx 5 \times 10^9$$

④ 밀도

$$\text{Density} = \frac{1,000,000}{4,999,950,000} \approx 0.0002$$

$$0.0002 \times 100 = 0.02\%$$

가능한 연결 10,000개 중 실제 연결은 약 2개뿐
가능한 간선 약 50억 개 가운데 실제 간선은 100만 개

Why Sparsity Grows with Graph Size

가능한 간선은 n^2 로 늘어나지만, 평균 차수가 일정하면 실제 간선은 n 으로만 늘어남

가능한 간선 수

$$O(n^2)$$

노드 수의 제곱으로 증가

실제 간선 수

$$m = \frac{n\bar{d}}{2} = O(n)$$

평균 차수가 일정할 때

두 식을 나누면 밀도가 간단해짐

$$\frac{2m}{n(n-1)} = \frac{\bar{d}}{n-1} \approx \frac{\bar{d}}{n}$$

평균 차수를 20 으로 고정했을 때의 밀도

$$n = 1,000$$

2 %

$$n = 10,000$$

0.2 %

$$n = 100,000$$

0.02 %

$$n = 1,000,000$$

0.002 %

그래프가 커질수록 밀도는 0 에 가까워짐 — 큰 그래프일수록 더 희소해짐

평균 차수 자체가 n 과 함께 커지는 그래프에는 이 근사가 성립하지 않음

Why Sparsity Decides the Storage Format

노드 10만 개 예제를 인접 행렬과 간선 목록으로 저장했을 때의 차이

인접 행렬의 원소 수

$$100,000 \times 100,000 = 10^{10}$$

100억 개의 원소 대부분이 0

실제 간선 수

$$m = 1,000,000$$

존재하는 간선만 기록하면 충분

$$\frac{10^{10}}{10^6} = 10^4$$

행렬 원소가 실제 간선보다 1만 배 많음

그래서 실제로 존재하는 간선만 나열하는 방식이 유리함

$(0, 1), (0, 27), (1, 53), \dots$

희소 그래프에서는 0 을 저장하지 않는 형식이 메모리에서 유리함
다만 저장 효율과 실행 속도는 별개로 따져야 함

희소성은 저장 형식 선택의 출발점

Dense vs. Sparse Representation

DENSE

가능한 모든 위치의 값을 저장

SPARSE

저장할 위치의 인덱스와 값을 저장

공통 예제 A 의 원소 계산

전체 원소
 $5 \times 5 = 25$

비영(None-zero) 원소 s
 8

간선 수 m
 4

무방향 그래프에서 m 개지만 대칭 인접 행렬의 비영 원소 수는 $2m = 8$

인덱스 비용이 추가됨

작거나 밀도가 높은 행렬에서는
항상 더 작지 않음

값 배열의 유무

이진 간선 목록은 값 1을 생략할 수 있음

저장 효율과 실행 속도는 같은 것이 아님

COO Representation

COO (Coordinate) — 좌표와 값으로 기록하는 희소 행렬 형식

모든 칸을 저장하는 대신 "어디에 어떤 값이 있는가"만 저장

세 배열의 같은 위치가 한 묶음 · 인덱스는 0 부터 시작

k	0	1	2	3	4	5	6	7
$row[k]$	0	0	1	1	2	2	2	3
$col[k]$	1	2	0	2	0	1	3	2
$value[k]$	1	1	1	1	1	1	1	1

$$A[row[k], col[k]] = value[k]$$

$$k = 0 : (0, 1, 1)$$

$A[1,0] = 1 \rightarrow a$ 에서 b 로

$$k = 2 : (1, 0, 1)$$

$A[1,0] = 1 \rightarrow$ 무방향 그래프에서 역방향 항목

무방향 간선 ab 하나를 두 좌표에 저장하므로 저장 항목은 $s = 8$ 임

고립 노드 e 를 보존하려면 행렬 크기 $(5,5)$ 도 함께 알아야 함

Isolated Node and Matrix Shape

COO 의 세 배열을 실제 인접 행렬로 펼쳐 보면 고립 노드가 어디에 있는지 드러남

COO 를 행렬로 복원하면

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

노란색이 5번째 행과 열

마지막 5번째 행과 열

고립 노드 e

원소가 모두 0 이지만 행렬에는 자리가 존재함

COO 배열에 없는 정보

row, col, value 에 노드 ID 4가 전혀 없음

4×4 인지 5×5 인지 알 수 없음

별도의 크기 정보가 필요함

$\text{shape} = (5, 5)$

고립 노드 e 를 보존하려면 행렬 크기 $(5, 5)$ 도 함께 알아야 함

좌표 배열은 연결만 기록하므로 노드 수는 별개로 보존해야 함

$A \in \mathbb{R}^{5 \times 5}$ 로 선언해야 노드 4의 존재가 유지됨

COO and PyG edge_index

PyG = PyTorch Geometric, 그래프 학습 라이브러리의 약칭

row 와 col 을 두 행으로 묶은(2,텐서)서 — 각 열이 출발·도착 노드 쌍 하나

source (row)	0	0	1	1	2	2	2	3
target (col)	1	2	0	2	0	1	3	2

첫 번째 열 [0, 1]

$a \rightarrow b$ 연결 항목

일곱 번째 열 [2, 3]

$c \rightarrow d$ 연결 항목

edge_index 자체에는 value 배열이나 간선 특징이 들어 있지 않음

COO 는 저장 형식, **PyG**는 그 형식을 활용하는 라이브러리임

중복 좌표의 처리 규칙은 구현에 따라 달라질 수 있음

CSR Representation

CSR (Compressed Sparse Row) — 반복되는 행 인덱스를 포인터로 압축

row_ptr	0	2	4	7	8	8			
col_idx	1	2	0	2	0	1	3	2	
values	1	1	1	1	1	1	1	1	

row i : $[\text{row_ptr}[i], \text{row_ptr}[i+1])$

시작 포함 · 끝 제외 구간

$c(i=2) : [4, 7)$

col_idx 에서 0, 1, 3 을 얻음

→ 이웃 a, b, d

$e(i=4) : [8, 8)$

구간이 비어 있음

→ 고립 노드

포인터 길이 $n + 1 = 6$

마지막 값 $s = 8$

COO 와 같은 행렬을 표현

행 인덱스만 포인터로 바뀜

Why CSR Is Efficient

$n + 1$ 개의 포인터 + s 개의 col-value — 이 표현이 의미하는 절감 효과

COO — 세 배열 모두 길여

$$s + s + s = 3s$$

row 인덱스가 s 개 그대로 저장됨

CSR — row_ptr 만 $n + 1$ 로 압축

$$(n + 1) + s + s$$

$n + 1$ 개의 row_ptr
+ s 개의 col + s 개의 value

예제 : n (#nodes) = 100,000 · s ($s = 2m$, #entries in sparse representation) = 2,000,000

$$n + 1 \ll s$$

COO 의 row 인덱스

2,000,000

CSR 의 row_ptr

100,001

행 정보 저장량

약 1/20

GNN 관점 — 노드 의 이웃 탐색

$$\text{col}[\text{row_ptr}[i] : \text{row_ptr}[i + 1]]$$

이 구간만 읽으면 되므로 노드별 이웃 탐색과 message aggregation 에 효율적

row 인덱스 200만 개를 저장하는 대신 약 10만 개의 row_ptr 만 저장하면 됨

Memory Complexity of Graph Representations

가상 예제 : $n = 100,000$ · $m = 1,000,000$ · 양방향 저장 $s = 2,000,000$

형식	저장 내용	메모리
dense	n^2 개의 값	40 GB
COO	s 개의 row × col × value	40 MB
CSR	$n + 1$ 포인터 + s 개의 col × value	약 24.8 MB

가정

float32 값 4 B × int64 인덱스 8 B

PyG edge_index 두 행만

32 MB

dense 40 GB → 희소 형식 수십 MB : 약 1,000 배 차이

단위는 십진 GB · MB 기준임

객체 오버헤드, 특징, 모델, 중간 연산 메모리는 제외한 값임
이진 그래프에서 값 배열을 생략하면 비용이 달라짐

Real-World Graphs at Scale

공개된 대규모 그래프에서는 저장 형식 선택이 실제 실행 가능성을 결정함

실제 데이터셋	노드	간선	그래프 의미	규모 문제
Friendster	65,608,366	1,806,067,135	사용자-친구	Dense 표현 불가능
Twitter-2010	41,652,230	1,468,364,884	사용자-팔로워	Edge list 도 수십 GB
ogbn-papers100M	111,059,956	1,615,685,872	논문-인용	GNN 학습 메모리 한계

Friendster 를 float32 dense 로 펼치면

$$(65.6 \text{ M})^2 \times 4 \text{ B} \approx 17.2 \text{ PB}$$

Friendster

Dense adjacency matrix 는
PB 단위가 될 수 있음

Twitter-2010

Sparse edge list 로 바뀌도
수십 GB 가 될 수 있음

ogbn-papers100M

Sparse 로 저장해도 GNN 학습
단계에서 메모리 문제가 남음

Dense → Sparse (COO / CSR) → Neighbor Sampling · Mini-batch GNN

취향의 선택이 아니라 실제 규모를 다루려고 차례로 등장한 단계임

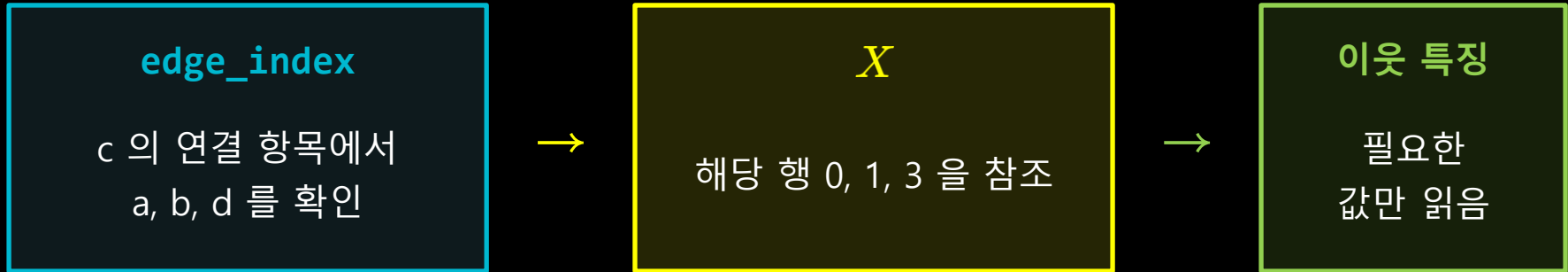
Friendster · Twitter-2010 은 SNAP, ogbn-papers100M 은 Open Graph Benchmark 의 공개 수치임

Twitter-2010 은 directed edge 기준이며, edge-list 인코딩이 25 GB 를 넘었다는 보고가 있음

Why Sparse Representation Matters for GNNs

모든 노드 쌍 n^2 을 저장하기보다 실제 연결 s 개를 참조

데이터 접근 사례 : 노드 c 의 이웃 특징이 필요한 경우



edge_index

어떤 노드의 특징을 참조할지
알려주는 연결 정보

X

참조할 수치 정보

희소 그래프에서 연결 저장량을 줄이면 이웃 참조가 실용적인 규모가 됨

주의

전체 비용은 특징 차원 · 모델 · 중간 결과에 따라 달라짐

Structure Alone Is Not Enough

연결 구조가 같아도 노드와 관계의 속성이 다르면 필요한 정보가 달라짐

A 가 알려주는 것

$$A_{ab} = 1$$

a 와 b 가 연결되어 있음

연결 여부만 기록

A 가 알려주지 못하는 것

a 의 활동량 2, b 의 활동량 1

a 는 유형 A, b 는 유형 B

속성은 별도로 기록해야 함

동일한 연결 패턴을 갖는 두 데이터셋도 관측 속성에 따라 다른 예측 문제가 됨

세 가지를 별개로 정의

연결 정보

$$A \cdot edge_index$$

관측 특징

$$X \cdot F_E$$

예측할 정답

$$y$$

구조만 존재하는 데이터도 있으며, 속성의 유용성은 Domain Task에 달려 있음

Node Feature Representation

$X \in \mathbb{R}^{n \times F}$ – 행은 노드, 열은 특징

노드	활동량	유형 A	유형 B
a	2	1	0
b	1	0	1
c	3	1	0
d	1	0	1
e	0	1	0

$$x_c = [3, 1, 0]$$

c의 특징 벡터

$$X \in \mathbb{R}^{5 \times 3}$$

노드 5개 × 특징 3개

행 순서 [a, b, c, d, e]

열 순서 [활동량, 유형A, 유형B]

X 의 행 번호는 A와 edge_index의 노드 ID와 반드시 일치

수치 특징에는 단위가 있고, 결측값은 실제 0과 구별해야 함
표준화 등의 통계는 학습에 허용된 데이터에서만 산출

여기의 X 는 관측 정보를 인코딩한 입력이며, 학습된 노드 벡터가 아님

Categorical Node Features

순서가 없는 범주형 속성은 one-hot encoding 으로 표현

유형 A
[1, 0]

유형 B
[0, 1]

범주 사전과 열 순서는 모든 노드에서 동일하게 유지

잘못된 방식 : 범주를 0, 1, 2 같은 수치로 그대로 넣는 경우

존재하지 않는 순서나 거리 해석을 유도할 수 있음

유형 A = 0, 유형 B = 0, 유형 C = 0 → "C 가 A 보다 크다" 로 읽힘

범주 수 K

one-hot 차원 = K

다중 범주 소속

multi-hot 으로 표현

미지 · 결측

처리 규칙을 따로 정함

노드 식별자와 의미 있는 범주 속성은 구별해야 함

Edge Feature Representation

간선 특징 = 연결된 두 노드 사이에서 관측한 속성 (횟수 · 거리 · 관계 유형 등)

edge_index 의 k 번째 열과 E k 번째 행이 같은 연결 항목에 대응

	0	1	2	3	4	5	6	7
연결 항목	a,b	a,c	b,a	b,c	c,a	c,b	c,d	d,c
(횟수)	2	1	2	3	1	3	1	1

$$F_E \in \mathbb{R}^{8 \times 1}$$

저장 항목 8개 × 특징 1개

무방향 간선 ab

$k=0, k=2$ 에 같은 값 2를 복사

저장 순서를 바꾸면 간선 특징도 반드시 함께 재정렬해야 함

방향별 속성이 있는 데이터는 그 의미에 따라 방향마다 다른 값을 씀

다차원 간선 특징

여러 열을 갖는 행렬

스칼라 가중치

간선당 값 하나

Graph-Level Features

그래프 전체에 하나씩 부여하는 특징임

구조 요약

노드 수 · 간선 수 · 평균 차수

외부 메타데이터

측정 조건 등 관측 외부 정보

공통 그래프의 구조 요약

노드 수	간선 수	평균 차수
5	4 $\frac{2m}{n} = \frac{8}{5} = 1.6$	1.6

고립 노드 e 도 분모에 포함됨 양방향 저장 항목 8 은 간선 수가 아님

그래프 특징 벡터와 그래프 단위 정답은 역할이 다름

전체를 요약한 값으로는 각 노드의 상세한 연결을 복원할 수 없음

Features from Graph Structure

외부 속성이 없어도 연결 구조에서 결정적으로 계산할 수 있는 특징이 있음

노드 수준

차수
참여 삼각형 수

간선 수준

양 끝점의
공통 이웃 수

그래프 수준

노드 수
간선 수

공통 예제에서의 계산 결과

노드 c 의 차수

3

c 의 참여 삼각형

1

간선 ab 의 공통 이웃 → c 하나

간선 cd 의 공통 이웃 → 없음

어떤 통계량을 쓸지 사람이 정하므로 **수작업 구조 특징**이라고 부름

구조 특징은 요약이므로 서로 다른 구조를 같은 값으로 만들 수 있음

Degree as a Structural Feature

차수 벡터를 기존 x 에 열로 추가하면 특징 차원이 3 에서 4 로 늘어남

노드	활동량	유형 A	유형 B	차수 (추가)
a	2	1	0	2
b	1	0	1	2
c	3	1	0	3
d	1	0	1	1
e	0	1	0	0

$$x_c = [3, 1, 0, 3]$$

차수가 네 번째 열로 붙음

$$\text{정규화} : \frac{d_i}{n-1}$$

$$[0.5, 0.5, 0.75, 0.25, 0]$$

차수가 설명하지 못하는 것

이웃 사이의 연결이나 먼 곳의 구조는 드러내지 못함

차수가 높다고 모든 과제에서 중요한 것은 아님 정규화 방식도 문제에 맞게 정함

Local Structural Features

차수만으로 드러나지 않는 이웃 사이의 연결을 측정

참여 삼각형 수

t_i : 노드 i의 이웃 사이에 존재하는 간선 수

Local clustering coefficient

$$C_i = \frac{2t_i}{d_i(d_i-1)}$$

$d_i \geq 2$ 에 대해 정의하며, $d_i < 2$ 일 때 $C_i = 0$ 으로 설정

노드	a	b	c	d	e
d_i	2	2	3	1	0
t_i	1	1	1	0	0
C_i	1	1	1/3	0	0

노드 c 의 풀이

이웃은 a, b, d 이므로 가능한 이웃 쌍은 ab, ad, bd 의 세 개

이 중 실제로 연결된 쌍은 ab 하나뿐임

$$C_c = \frac{2 \cdot 1}{3 \cdot 2} = \frac{1}{3}$$

C_i 는 이웃 간 연결 비율이며, 범용적인 중요도 점수가 아님

Raw Features vs. Structural Features

RAW FEATURES

원시 관측 특징

활동량 · 유형

외부 기록에서 읽어 인코딩

STRUCTURAL FEATURES

구조 특징

차수 · clustering coefficient

E 에서 계산

노드 c 의 결합 예시

$[3, 1, 0]$

+

$[3, 1/3]$

=

$[3, 1, 0, 3, 1/3]$

연결이 바뀌면 구조 특징을 다시 계산해야 하고, 관측 속성도 측정 시점에 따라 달라짐

정보 누출 (leakage) 사례

숨긴 연결을 예측하는 Task에서 그 연결까지 포함해 구조 특징을 계산하면

정답 정보가 입력으로 새어 들어감

예측 시점에 사용할 수 있는 정보만 입력으로 삼아야 함

A Machine-Learning-Ready Graph

모델에 전달할 구성 요소

구성 요소	역할	예제의 크기
노드 ID 사전	이름 ↔ 정수 대응	$n = 5$
<code>edge_index</code>	연결 인덱스	$(2, 8)$
X	노드 특징	$(5, 3)$
F_E (선택)	간선 특징	$(8, 1)$
그래프 특징 (선택)	그래프 단위 요약	Task에 따라 다름
y	예측할 정답	노드 단위 / 그래프 단위

정렬 조건 체크리스트

모든 인덱스가 0 이상 n 미만인가	X 의 행 수가 n 인가
간선 특징 행 수 = 저장 항목 수인가	고립 노드를 보존했는가

정답 y 는 특징에 섞지 않음

PyTorch Geometric Data Object

COO

데이터 표현 형식

PyG

라이브러리

Data

그래프 정보를 담는 객체

PyTorch 와 PyG 의 관계

PyTorch : 텐서 연산과 자동 미분의 기반 제공

PyG : 그래프 데이터 객체와 GNN 관련 기능 제공

PyG (PyTorch Geometric) 는 PyTorch 를 기반으로 그래프 데이터와 그래프 신경망(GNN) 을 다루는 라이브러리

Data Object Fields

필드	의미	예제 크기
x	노드별 특징 각 행이 노드 하나	(5, 3)
edge_index	연결된 노드의 인덱스 쌍 첫 행 source, 둘째 행 target · torch.long	(2, 8)
edge_attr	연결별 특징 edge_index 의 같은 순서 열에 대응	(8, 1)
y	Task에 따른 예측 정답 노드 단위인지 그래프 단위인지에 따라 다름	Not Given in Our Slide
num_nodes	노드 수 고립 노드를 포함	5

```
from torch_geometric.data import Data
data = Data(x=x, edge_index=edge_index,
            edge_attr=edge_attr, num_nodes=5)
```

Data 에 저장했다고 특징이 자동으로 학습되는 것은 아님
x 가 없고 연결만으로 노드 수를 추론하면 고립 노드를 놓칠 수 있음
그래서 num_nodes 를 명시함 COO 의 value=1 과 edge_attr 는 다른 값임

One Small Graph: End-to-End Representation

```
import torch
from torch_geometric.data import Data

x = torch.tensor([
    [2, 1, 0], [1, 0, 1], [3, 1, 0],
    [1, 0, 1], [0, 1, 0]
], dtype=torch.float)

edge_index = torch.tensor([
    [0, 0, 1, 1, 2, 2, 2, 3],
    [1, 2, 0, 2, 0, 1, 3, 2]
], dtype=torch.long)

edge_attr = torch.tensor([
    [2], [1], [2], [3], [1], [3], [1], [1]
], dtype=torch.float)

data = Data(x=x, edge_index=edge_index, ...)
```

x 의 3번째 행

c 의 특징 [3, 1, 0]

7번째 열 [2, 3]

c 에서 d 로 향하는 항목
edge_attr 값은 1

edge_index 에 4 가 없음

x 의 5번째 행과
num_nodes=5 가 e 를 보존

수학적 간선 수

4

저장 항목 수

8

노드 수

5

From Hand-Crafted to Learned Representation

사람이 ML (Graph Learning)을 위해 결정한 사항

01

노드와 관계의 정
의

02

보존할 관측 속성

03

범주의 수치 인코
딩

04

추가할 구조 통계

입력을 준비하는 일

의미와 정렬을 사람이 관리

표현을 얻는 일

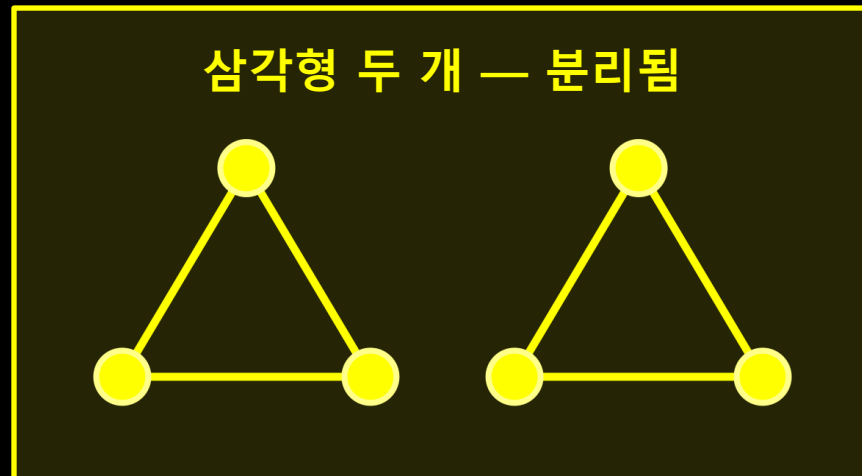
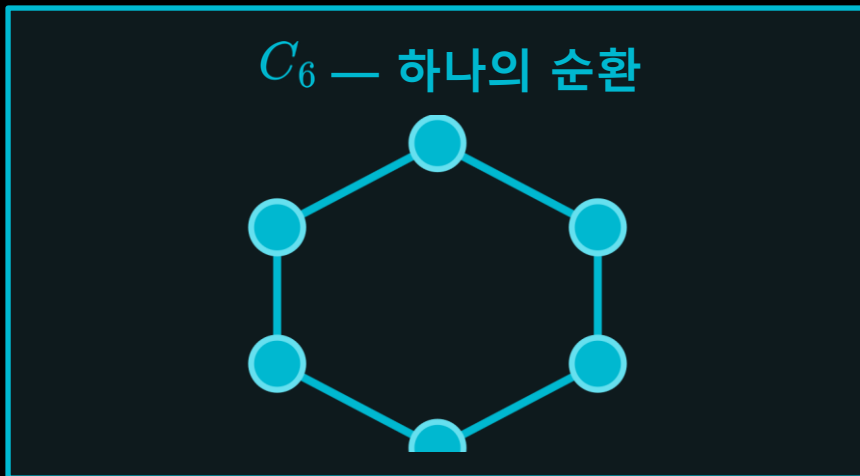
학습 과정에서 다루는 별개의 문제

두 일은 구별되지만, 입력의 의미와 정렬은 이후에도 계속 중요함

사람이 모든 유용한 구조 특징을 미리 정해야 할까?

Limitations of Hand-Crafted Features

별도 반례 : 노드 6개가 하나의 순환을 이루는 C_6 와 분리된 삼각형 두 개



실제 그래프 특성	순환 링 C_6	삼각형x2 (double triangle)
각 노드의 degree만 사용한 특성	모두 2	모두 2
clustering coefficient 추가	0	1

차수만 정보만 그래프 신경망에 입력하면 두 그래프의 연결성 차이를 드러내지 못함
Clustering coefficient 를 추가하면 극복 가능

그러나 추가한 특징도 그 통계가 측정하는 정보만 포함 (내부 정보 파악 불가)

Next: Node Embedding

This Lecture:

연결 구조

인접 행렬 · 희소 인덱스

관측 특징

노드 · 간선에 정렬

구조 특징

차수 등 사람이 선택

Next Step:

노드의 유용한 표현을 데이터로부터 학습할 수 있을까?

Node Embedding

학습된 노드 표현