



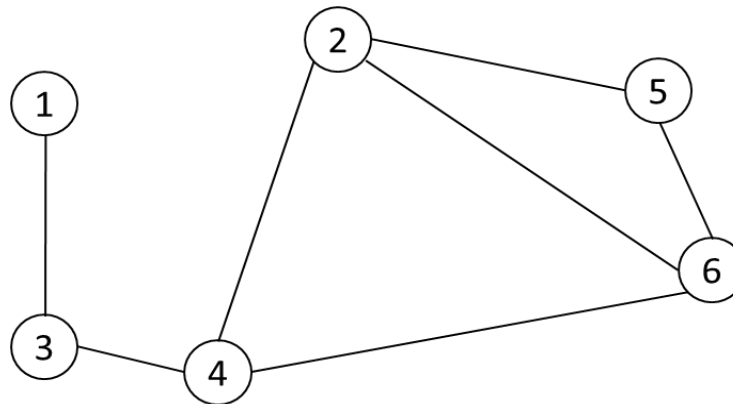
# Graph Neural Network

---

Sang Yup Lee

# GNN

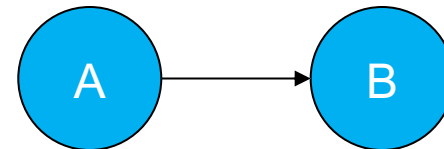
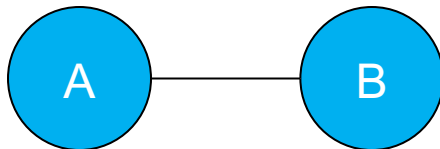
- What is GNN?
  - It is a type of neural networks that can be applied to graph data.
- Then what is a graph (a.k.a., network)?
  - Something that is composed of nodes (or vertices) and ties (edges) between nodes.
  - A graph can be denoted as  $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ , where  $\mathcal{V} = \{v_1, \dots, v_N\}$  is a set of  $N = |\mathcal{V}|$  nodes and  $\mathcal{E} = \{e_1, \dots, e_M\}$  is a set of  $M$  edges.
  - Example graph



# Graph

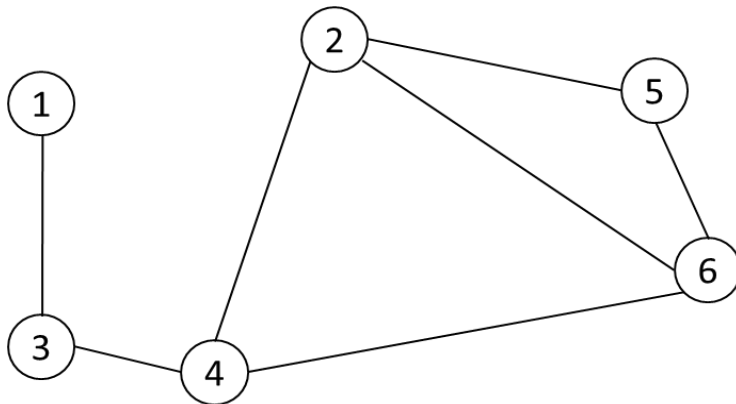
## ■ 그래프의 예

- 소셜 네트워크, 인터넷, 분자, 텍스트, 지역, 인용 네트워크 등
- 타이의 종류 두 가지
  - 방향성이 없는 타이 (undirected tie)
    - 대칭적 타이 (symmetric tie) 라고도 함
    - 친구 관계, 연인 관계, 페이스북에서의 친구 관계 등
  - 방향성이 있는 타이 (directed tie)
    - 비대칭 (asymmetric) 타이라고도 함
    - 의사-환자, follower-followee 등

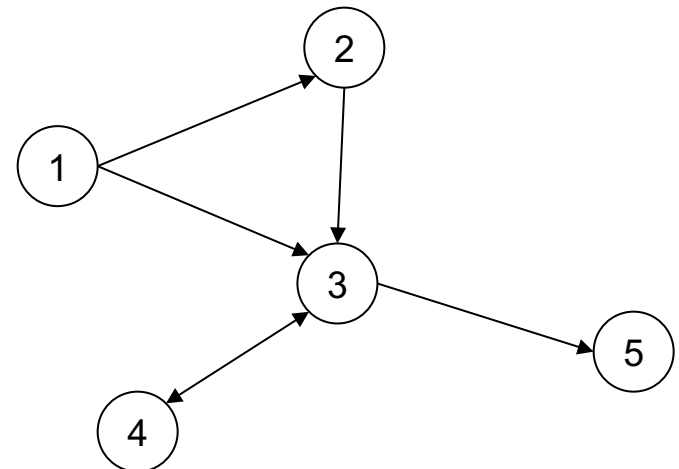


# Graph

- 그래프의 종류
  - Undirected graph
    - Undirected ties로 구성된 그래프
    - 예) 친구 관계 네트워크
  - Directed graph
    - Directed ties로 구성된 그래프
    - 예) 인용 네트워크



12/11/23



GNN



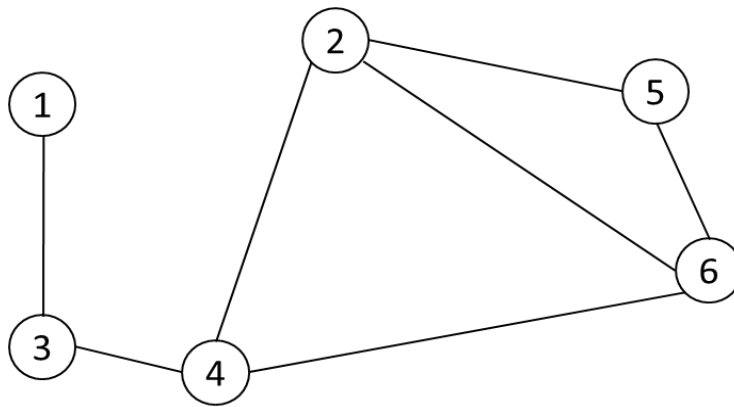
# GNN

---

- GNN 관련 주요 작업의 종류
  - Node level tasks
    - Node classification
      - 예) 소셜 네트워크에서 사람들의 정치 성향
    - Nodes clustering
    - Identifying influential nodes
  - Graph level task
    - Graph classification
      - 예) 문서 분류, 분자 분류 등
  - Edge level task
    - Edge prediction (link prediction)

# Graph

- 파이썬을 이용한 그래프 분석 (network analysis라고도 함)
  - NetworkX 모듈 사용
    - <https://networkx.org/>
  - 예제 그래프

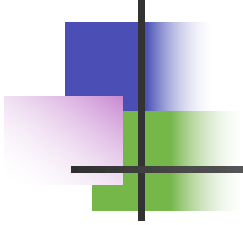




# Graph

---

- 파이썬을 이용한 그래프 분석
  - 예제 코드: `graph_example.ipynb`
- Main procedure of graph analysis using NetworkX
  - 1) Prepare graph data
    - that is information about nodes and their ties
  - 2) Construct a graph representing the graph data using NetworkX
    - create an empty graph
    - add nodes to the graph
    - add edges to the graph
  - 3) Analysis
    - Do graph analysis using NetworkX
      - We usually do EDA with NetworkX
    - Or export network data to other programs and do some analysis



# TEXT NETWORK ANALYSIS





# Text network analysis

---

- You need to construct a network composed of words and ties between words
  - nodes: words
  - tie? how can we define a tie between two words?
    - Commonly we define a tie between two words if they use in the same sentence or paragraph or a document
    - We use a sentence
  - How to choose the words consisting the network
    - 1) Choose certain words based on theoretical backgrounds
    - 2) Choose most frequently used words (e.g., Top 10)



# Text network analysis

---

- Process
  - Prepare text data that you want to analyze
  - Select some words that are used most frequently e.g., top 10
  - In order to find ties between words, you need to split the entire text into sentences
    - For English, you can use `sent_tokenize()`
    - For Korean, you can use `split_into_sents()` from Kiwi
  - Do preprocessing
  - Identify ties between words
    - For this, you need to create your own function



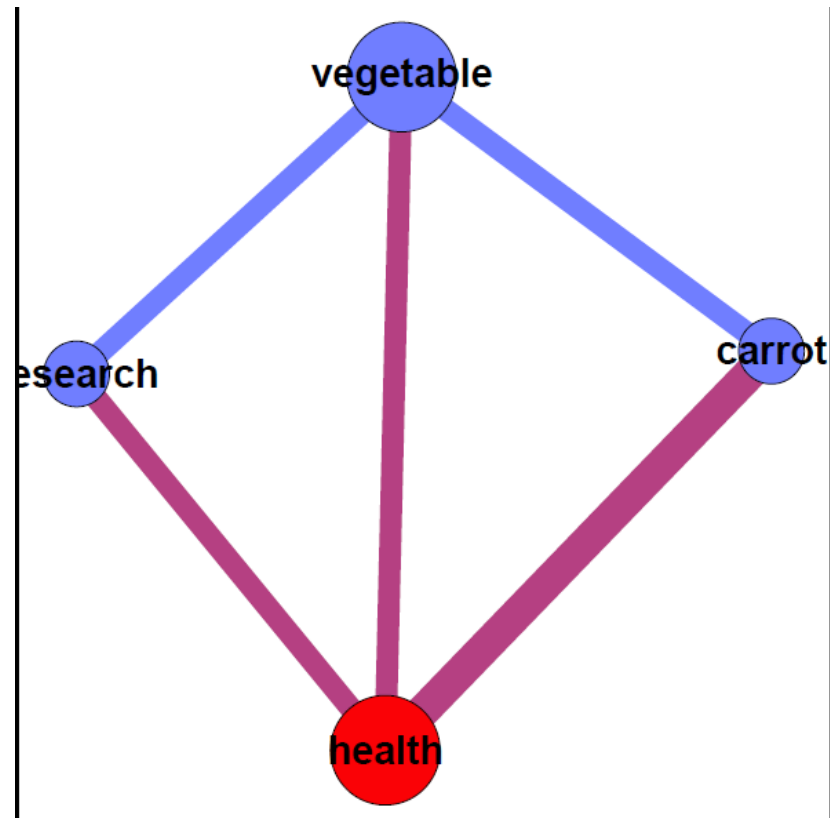
# Text network analysis

---

- Example

- `text1 = 'The carrot is one of vegetables. Research shows vegetables are good for health. Thus, carrots are also good for health. Your health can be improved with carrots.'`
- `[['carrot', 'vegetable'], ['research', 'vegetable', 'health'], ['carrot', 'health'], ['health', 'carrot']]`

# Text network analysis





# Text network analysis

---

- Example

- English

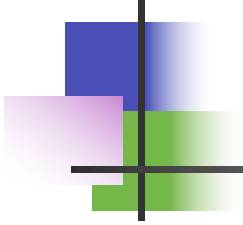
- NY Times

- <https://www.nytimes.com/2017/06/12/well/live/having-friends-is-good-for-you.html>

- Refer to "En\_network\_analysis.ipynb"

- 순서

- 전처리과정을 통해 명사의 단어들만 추출
      - 상위 K개의 명사 선택
      - 해당 단어들을 대상으로 네트워크 생성



# Graph Neural Networks



# GNN

---

- GNN 작업의 주요 과정
  - 분석하고자 하는 그래프 데이터 준비
    - 무엇을 노드로 하고, 노드들 간의 연결을 어떻게 정의할 것인지에 대해서 생각해 보는 것이 필요
  - 분석 목적에 적합한 GNN 모형 적용, 학습
  - 예측 / Inference



# GNN

---

- GNN 작업의 주요 과정의 예: 문서 분류
  - 주요 과정은 아래와 같음
    - 텍스트 데이터 수집 및 전처리
    - 분석 목적 (즉, 문서 분류)을 위해 텍스트 데이터를 어떠한 형태의 그래프로 / 어떻게 표현할 것인지를 결정하는 것 필요 (선행연구들을 참고할 수 있음)
      - 예) 하나의 문서를 하나의 그래프로 표현 => 단어가 노드가 되고 단어들 간의 연결 관계가 타이 정보가 됨 => 그래프 분류 문제
      - 전체 코퍼스를 하나의 그래프로 표현 => 단어와 문서가 각각의 노드가 됨 => 노드 분류 문제
    - 데이터를 학습 데이터와 평가 데이터로 구분
    - 분석 목적과 데이터 특성에 적합한 GNN 알고리즘 선택 및 적용
      - GCN, GraphSAGE, GAT 등 많은 알고리즘 존재
    - 예측





# GNN

---

- GNN 학습의 주요 목적
  - 학습을 통해 노드 또는 그래프의 특성을 잘 나타내는 embedding 벡터를 생성하는 것  $\Rightarrow$  representation learning으로 간주 가능
  - Node level의 작업을 위해서는 작업과 관련된 node의 특성을 잘 반영하는 embedding vector를 생성하는 것이 중요
  - Graph level의 작업을 위해서는 그래프의 특성을 잘 반영하는 embedding vector를 생성하는 것이 중요
    - 그래프의 임베딩 벡터는 일반적으로 노드의 특성을 나타내는 임베딩 벡터 정보를 활용해서 만들어 짐 (예, 평균, LSTM 등)
  - 이를 위해서 일반적으로 GNN 모형은 크게 filtering 부분과 pooling 부분으로 구성



# GNN

---

## ■ Filtering

- 노드의 임베딩 정보를 계산하기 위해서는 일반적으로 두 가지 정보를 사용
  - 노드의 특성 정보 (i.e., features)
    - 노드의 속성 정보라고도 함
  - 그래프의 구조 정보
    - 이는 일반적으로 노드들 간의 연결 정보를 의미
- 노드의 특성 정보와 그래프의 구조 정보를 이용해서 노드의 임베딩 벡터를 계산하는 과정은 아래와 같이 표현

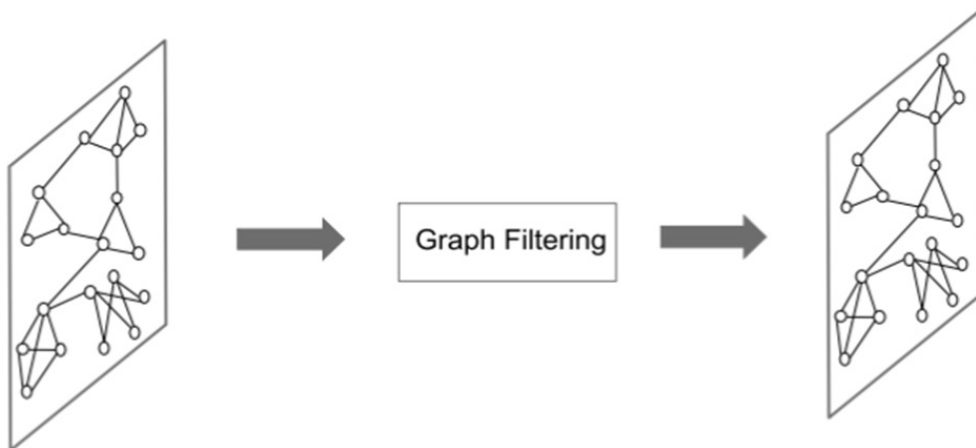
$$\mathbf{F}^{(of)} = h(\mathbf{A}, \mathbf{F}^{(if)})$$

- $\mathbf{A} \in \mathbb{R}^{N \times N}$  denotes the adjacency matrix of the graph with  $N$  nodes  $\Rightarrow$  graph structure 의미
- $\mathbf{F}^{(if)} \in \mathbb{R}^{N \times d_{if}}$  and  $\mathbf{F}^{(of)} \in \mathbb{R}^{N \times d_{of}}$  denote the input and output feature matrices where  $d_{if}$  and  $d_{of}$  are their dimensions, respectively

# GNN

## ■ Filtering (cont'd)

- 이렇게 노드의 특성 정보와 구조 정보 (즉, 인접행렬)를 이용해서 특성 정보를 계산 또는 업데이트하는 과정을 graph filtering이라고 함
- $h()$  가 그래프 필터가 되며, 그래프 필터에는 여러 가지 종류가 존재
- 이러한 필터링 과정을 여러번 반복 (여러 개의 신경망 층을 적용한다고 생각할 수 있음)



$$\mathbf{A} \in \{0, 1\}^{N \times N}, \mathbf{F}^{(if)} \in \mathbb{R}^{N \times d_{if}}$$

$$\mathbf{A} \in \{0, 1\}^{N \times N}, \mathbf{F}^{(of)} \in \mathbb{R}^{N \times d_{of}}$$

- Node level 작업의 경우, 이러한 filtering 과정을 거쳐, 노드의 특성을 잘 나타내는 임베딩 벡터를 얻는 것으로 충분
- 하지만, graph level 작업의 경우는 추가적인 과정이 더 필요  $\Rightarrow$  풀링 과정



# GNN

---

## ■ Pooling

- Graph level 작업에서는 filtering을 거쳐 업데이트된 노드의 임베딩 정보를 통합해서 그래프의 특성을 나타내는 임베딩 정보를 계산하는 것이 필요  $\Rightarrow$  pooling 과정이 필요
- Pooling 과정에서는 하나의 그래프에 대한 정보 (즉, 인접행렬과 노드의 피쳐 혹은 임베딩 정보)를 입력 받아, 노드의 수가 적은 그래프를 결과물로 출력 (이러한 그래프를 coarsened graph라고 표현)
- Pooling 과정의 결과물  $\Rightarrow$  coarsened graph의 인접행렬과 노드 임베딩 벡터. 이는 아래와 같이 표현

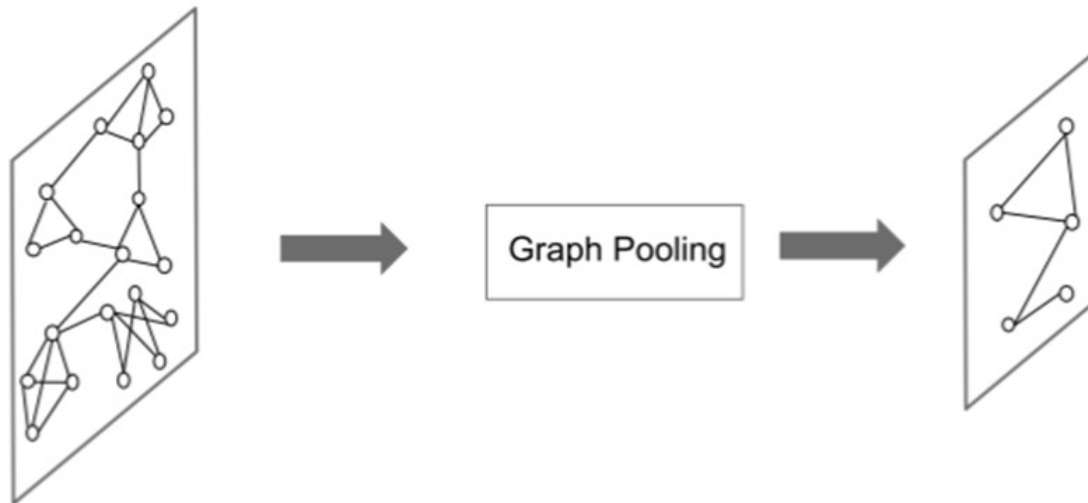
$$\mathbf{A}^{(\text{op})}, \mathbf{F}^{(\text{op})} = \text{pool}(\mathbf{A}^{(\text{ip})}, \mathbf{F}^{(\text{ip})})$$

where  $\mathbf{A}^{(\text{ip})}, \mathbf{F}^{(\text{ip})}, \mathbf{A}^{(\text{op})}, \mathbf{F}^{(\text{op})}$  are the adjacency matrices and feature matrices before and after the pooling operation, respectively.

# GNN

- Pooling (cont'd)

- 이를 그림으로 표현하면 아래와 같음



$$\mathbf{A}^{(\text{ip})} \in \{0, 1\}^{N_{\text{ip}} \times N_{\text{ip}}}, \mathbf{F}^{(\text{ip})} \in \mathbb{R}^{N_{\text{ip}} \times d_{\text{ip}}}$$

$$\mathbf{A}^{(\text{op})} \in \{0, 1\}^{N_{\text{op}} \times N_{\text{op}}}, \mathbf{F}^{(\text{op})} \in \mathbb{R}^{N_{\text{op}} \times d_{\text{op}}}$$

$N_{\text{op}}$  denotes the number of nodes in the coarsened graph and  $N_{\text{op}} < N_{\text{ip}}$ .



# GNN

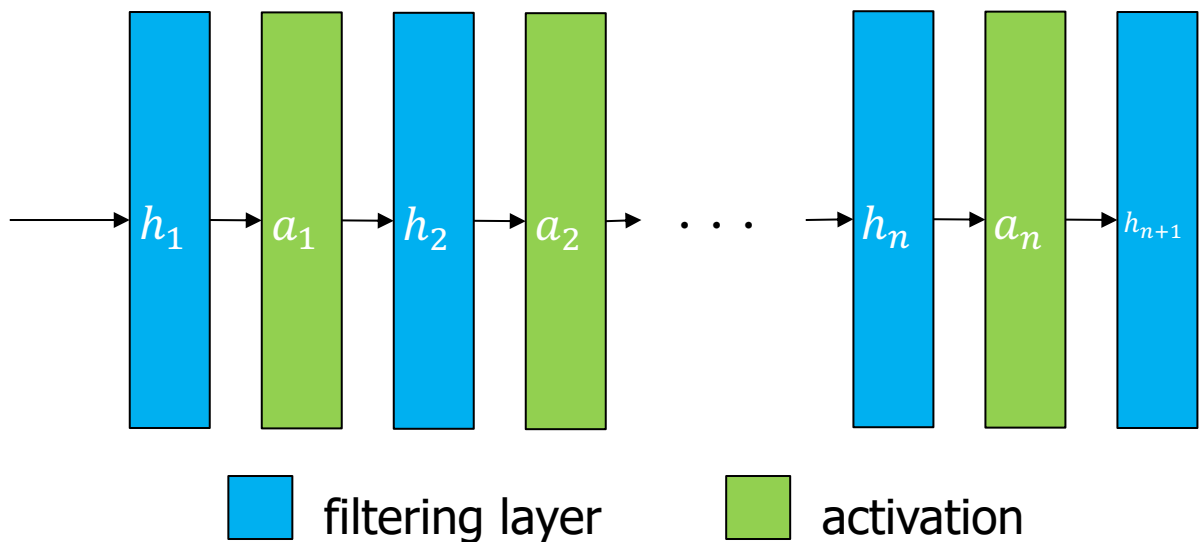
---

- 일반적인 GNN 모형의 구조
  - 여러 개의 filtering 부분과 pooling 부분으로 구성
  - Node level task
    - 여러 개의 filtering 사용 (No pooling required)
      - 하나의 filtering이 일반적으로 하나의 신경망 층을 의미하기 때문에 여러 개의 신경망 층을 사용한다는 것을 의미
  - Graph level task
    - Uses both graph filtering and graph pooling operations

# GNN

## ■ Node level 작업의 일반적 구조

- 일반적으로 그래프 필터링 + 비선형 활성화 함수로 구성
- downstream task에 적합한 노드 특성 정보 추출 (임베딩 생성)
- downstream task를 위한 추가적인 층 사용



$$\mathbf{F}^{(i)} = h_i \left( \mathbf{A}, \alpha_{i-1}(\mathbf{F}^{(i-1)}) \right)$$

- $\mathbf{F}^{(i)}$  denotes the output of the  $i$ th graph filtering layer.  $\mathbf{F}^{(i)} \in \mathbb{R}^{N \times d_i}$
- $\alpha_{i-1}()$  is the element-wise activation function following the  $(i-1)$ st graph filtering layer
- The final output  $\mathbf{F}^{(L)}$  is leveraged as the input to some specific layers according to the downstream node-focused tasks.



# GNN

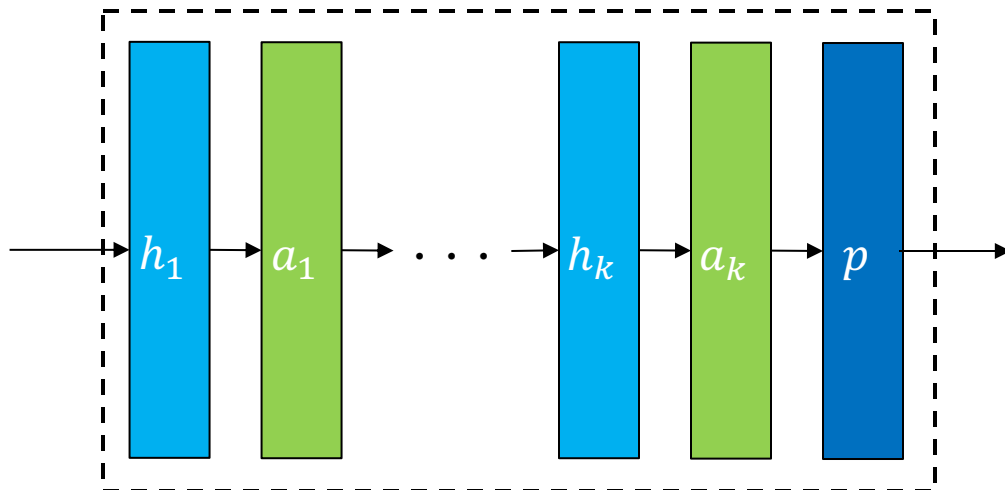
---

- Graph level 작업의 일반적 구조
  - 일반적으로 filtering layer + activation + (graph) pooling layer로 구성
  - The graph pooling layer is utilized to summarize the node features and generate higher-level features that can capture the information of the entire graph.
  - Typically, a graph pooling layer follows a series of graph filtering and activation layers.
  - A coarsened graph with more abstract and higher-level node features is generated after the graph pooling layer.



# GNN

- Graph level 작업의 일반적 구조 (cont'd)
  - 아래와 같은 block 여러 개 사용

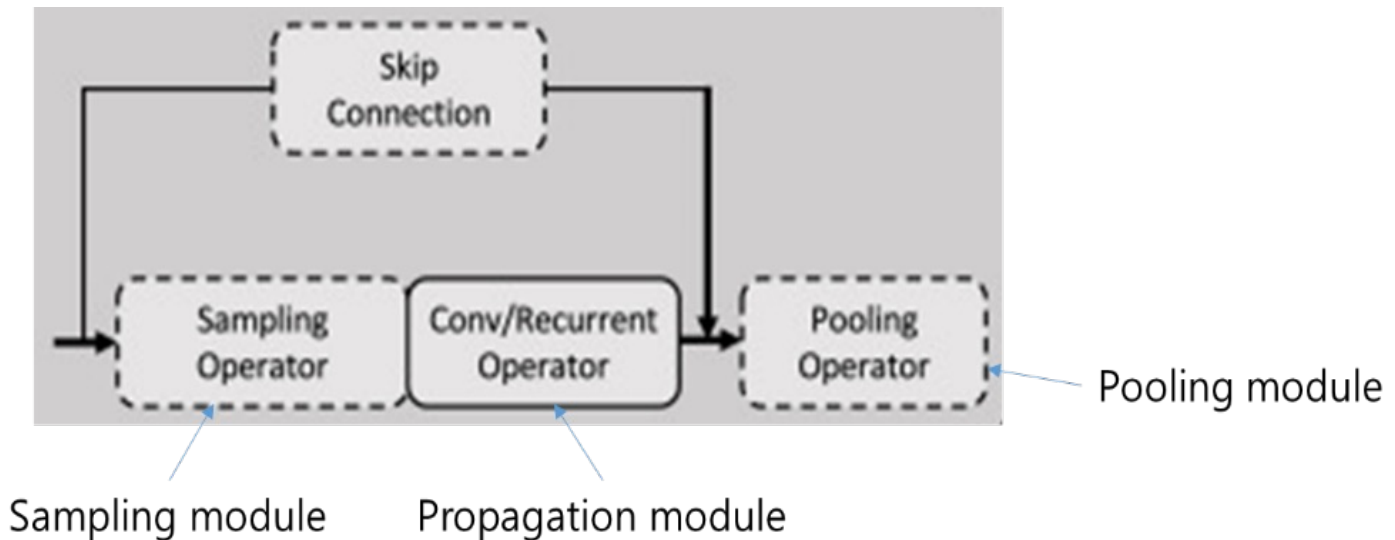


<A block in GNN for graph level tasks>

The input of the block is the adjacency matrix  $A^{(ib)}$  and the features  $F^{(ib)}$  of a graph  $G^{ib} = \{V^{ib}, E^{ib}\}$  and the output are the newly generated adjacency matrix  $A^{(ob)}$  and the features  $F^{(ob)}$  for the coarsened graph  $G^{ob} = \{V^{ob}, E^{ob}\}$

# GNN

- GNN block의 또 다른 예시
  - sampling operation과 skip connection이 포함된 경우





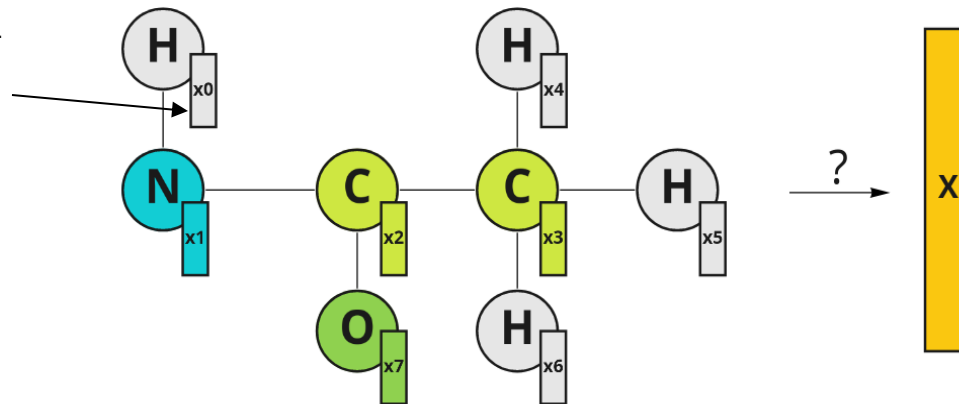
---

# Filtering operation의 예시

# Example

- Example1: 분자 (molecule) 분류
  - 하나의 분자를 하나의 그래프로 간주
    - 분자는 여러 개의 원자들과 원자들 간의 타이로 구성
    - 각 원자는 features를 지님 (예, mass, number of electrons, etc.)

각 노드가 하나의 특성 정보만을 갖는다고 가정, 이는 스칼라로 표현 (여러 개인 경우는 벡터로 표현)



- 그래프를 분류를 하기 위해서는 그래프가 갖는 공간적 구조 (spatial structure) 정보와 각 노드의 특성 정보 (features)를 이용해서 각 그래프를 저차원의 벡터(meaningful representation)로 표현하는 것이 필요



# Example

---

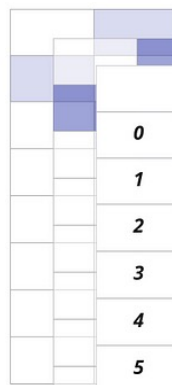
- Example1 (cont'd)
  - How can we generate a representative vector (i.e. embedding vector) of a graph?
  - 가장 간단한 방법은 노드들의 특성 정보를 단순히 aggregate하는 것 (즉, 평균 또는 합)
    - 예:  $X = \frac{1}{8} \sum_{i=0}^7 x_i$
    - 하지만, 이러한 방법의 문제는?
      - 그래프가 갖는 구조적 특성 정보를 반영하지 못한다

# Example

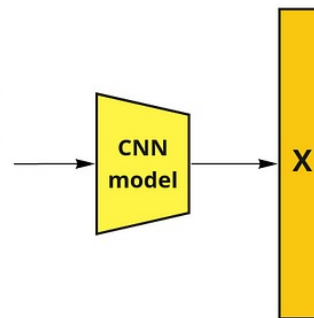
## ■ Example1 (cont'd)

### ■ Another idea

- 그래프를 인접행렬과 노드들의 특성 정보를 이용해서 이미지 처럼 표현해 보자!
- As a result, we obtain a pseudo-image  $[8, 8, N]$ , where  $N$  is the dimension of the node feature vector  $x$ .  $\Rightarrow$  이렇게 이미지 형태로 그래프를 표현하게 되면 CNN에서의 convolutional filter를 적용하여 feature map을 추출할 수 있다.



	0	1	2	3	4	5	6	7
0	0	$[x_0, x_1]$	0	0	0	0	0	0
1	$[x_0, x_1]$	0	$[x_1, x_2]$	0	0	0	0	0
2	0	$[x_1, x_2]$	0	$[x_2, x_3]$	0	0	0	$[x_2, x_7]$
3	0	0	$[x_2, x_3]$	0	$[x_3, x_4]$	$[x_3, x_5]$	$[x_3, x_6]$	0
4	0	0	0	$[x_3, x_4]$	0	0	0	0
5	0	0	0	$[x_3, x_5]$	0	0	0	0
6	0	0	0	$[x_3, x_6]$	0	0	0	0
7	0	0	$[x_2, x_7]$	0	0	0	0	0



- feature 정보는 파란색 부분에만 들어가는 것. 이웃 노드 피쳐 벡터를 활용: concat, mean, sum 등 사용 가능
- 만약 이어붙이기를 하고, 각 노드의 피쳐 벡터가  $N$  차원이라면 채널의 크기는  $N+N=2N$ 이 됨



# Example

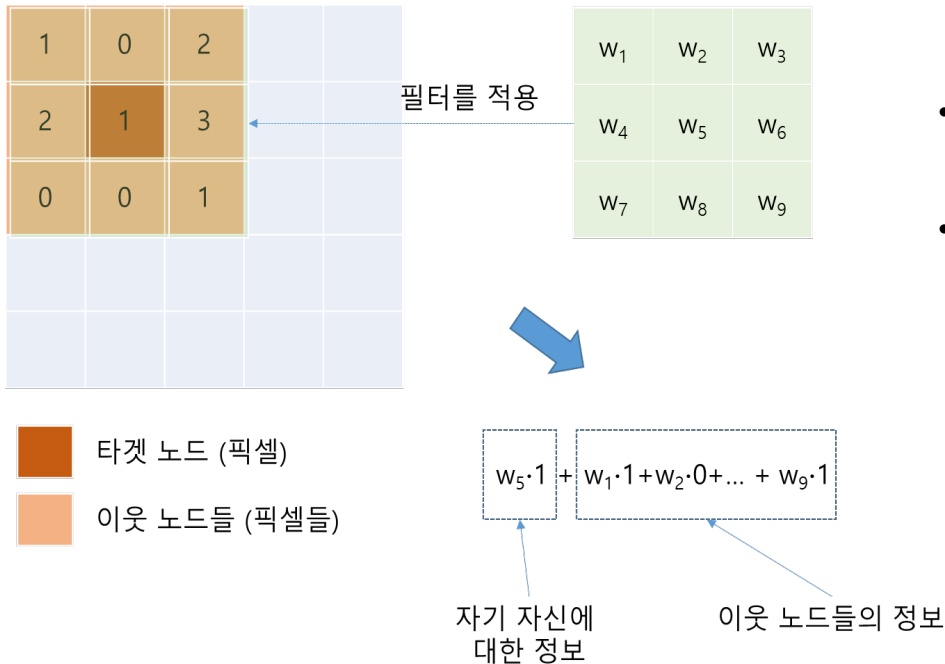
---

- Example1 (cont'd)
  - 하지만, 앞의 방법도 큰 문제를 지님  $\Rightarrow$  노드의 순서가 바뀌면 추출되는 representation도 달라진다 (if we change the node's order we obtain a different representation.)  $\Rightarrow$  즉, such a representation is not a permutation invariant.
  - Meaning of permutation invariant: 그래프의 노드 순서에 상관없이 결과가 동일하다는 것을 의미
  - 그렇다면 어떻게 해야 효율적으로 공간 정보를 추출할 수 있는가?  
 $\Rightarrow$  convolution 방법을 사용해야 하지만, 위와 같은 방법으로 하면 안된다. 그래프에 적용되어야 한다.

# Example

## ■ Example1 (cont'd)

- 먼저, 이미지에 적용되는 합성곱 필터를 살펴보자.
- What happens when we apply regular convolution on images? Values of neighboring pixels multiply on filter weights and sum up (아래 그림 참고).



- 이미지 (channel = 1)의 각 픽셀은 하나의 노드로 간주
- 이미지의 각 셀이 갖는 값은 해당 노드의 피쳐 정보로 간주 (옆 그림의 경우, 하나의 feature만 존재 => 스칼라 값으로 표현됨, 하지만, 이는 벡터로 표현될 수 있음)



# Example

## ■ Example1 (cont'd)

- 그래프에 대해서도 비슷한 작업을 할 수 있는가?  $\Rightarrow$  Yes!!
- How?
  - feature vector와 이웃 행렬을 곱함
  - We can stack node feature vectors in a matrix  $X$  and multiply them by adjacency matrix  $A$ , then we obtain updated features  $X'$  that combine information about node closest neighbors

0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
0	1	0	1	0	0	0	1
0	0	1	0	1	1	1	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0

 $\times$ 

x0
x1
x2
x3
x4
x5
x6
x7

 $=$ 

x1
x0 + x2
x1 + x3 + x7
x2 + x4 + x5 + x6
x3
x3
x3
x2

**A**
**X**
**X'**

12/11/23

- 1-hop 거리에 있는 (즉, 직접적으로 연결되어 있는) 노드의 정보만을 이용함 (Multiplication on adjacency matrix propagates features from node to node)
- 이미지의 경우와 비교하였을 때, 자기 자신에 대한 정보는 반영되지 않았고, 각 특성 정보가 가중치가 곱해지지 않았다는 차이가 있음  $\Rightarrow$  만약 자기 자신에 대한 정보를 더하고 싶다면  $A$ 를 사용하는 것이 아니라  $A+I$ 를 사용하면 됨.
- 관련 파이썬 코드는 graph\_basics.ipynb 참고

GNN

# GNN

## ■ Example1 (cont'd)

- 이미지의 경우, 필터의 크기를 크게하면 더 많은 이웃한 픽셀들의 정보를 사용 가능
- In graphs, we also can take more distant neighbors into account.
  - 예를 들어,  $A^2$ 을 특성 정보 행렬인  $X$ 에 곱하면 2-hop 거리에 있는 노드들의 특성 정보 활용

1	0	1	0	0	0	0	0
0	2	0	1	0	0	0	1
1	0	3	0	1	1	1	0
0	1	0	4	0	0	0	1
0	0	1	0	1	1	1	0
0	0	1	0	1	1	1	0
0	0	1	0	1	1	1	0
0	1	0	1	0	0	0	1

$A^2$

$\times$

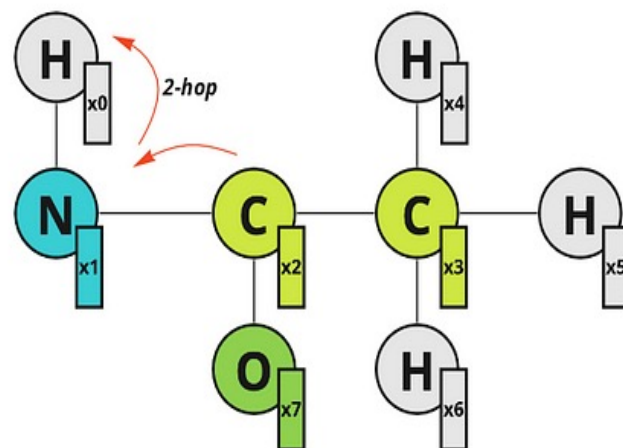
$x_0$
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
$x_6$
$x_7$

$X$

$=$

$x_0 + x_2$
$2x_1 + x_3 + x_7$
$x_0 + 3x_2 + x_4 + x_5 + x_6$
$x_1 + 4x_3 + x_7$
$x_2 + x_4 + x_5 + x_6$
$x_2 + x_4 + x_5 + x_6$
$x_2 + x_4 + x_5 + x_6$
$x_1 + x_3 + x_7$

$X'$





# GNN

---

- Example1 (cont'd)

- Higher powers of matrix  $A$  behaves in the same way: multiplication by  $A^n$  leads to propagation of features from  $n$ -hop distance nodes.

- Generalization

- To generalize this operation, one can define the **function of node updates** as the sum of such multiplications with some weights  $\mathbf{w}$ .

$$P = w_0 I + w_1 A + w_2 A^2 + \cdots + w_n A^n$$

$$x' = Px$$

- $P$  is called a polynomial graph convolution filter, which is parametrized by weights  $\mathbf{w}$ .
- 새로 업데이트된 특성 정보 벡터  $x'$  에는  $n$ -hop 거리 안에 있는 노드들의 특성 정보가 반영되어 있고, 각 노드들의 특성 정보가 반영되는 정도는  $\mathbf{w}$ 의 값에 따라 달라진다.
- Such polynomials satisfy permutation invariance as general convolutions.

# GNN

## ■ Example1 (cont'd)

### ■ 이미지의 경우와 비교

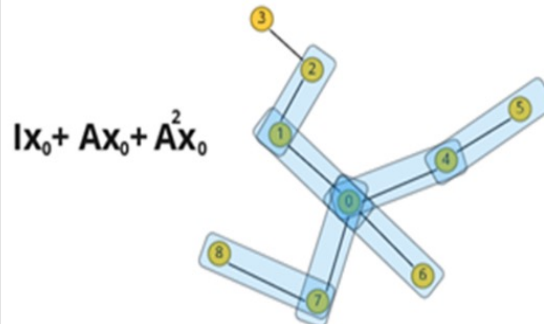
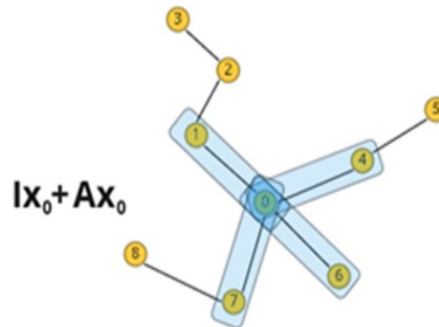
0	0	1	0
0	0	1	0
0	0	1	0
0	1	0	1

kernel 2x2

0	0	1	0
0	0	1	0
0	0	1	0
0	1	0	1

kernel 3x3

vs.



In analogy with convolutions on images, graph convolution filters can have different sizes and aggregate information about node neighbors, but the structure of the neighbors shouldn't be regular like the convolution kernel in the images

이미지의 경우에는 픽셀들의 순서가 고정되어 있지만, 그래프의 경우에는 노드들의 순서가 어떠한 식으로 되든 상관없음 (연결 정보만 유지된다면)



# GNN

---

- Example1 (cont'd)

- 인접행렬이 아닌 Laplacian 행렬을 이용하는 것도 가능
  - Laplacian 행렬을 사용하면 노드 특성 정보의 차이를 반영할 수 있음
  - Laplacian 행렬
    - $L = D - A$
    - D는 각 노드의 디그리 값을 원소로 하는 대각행렬
- Convolutional filter
  - 앞에서 설명한 filter는 convolutional filter의 예 (그 중에서 spatial filter)
  - 이는 spectral filter로도 설명될 수 있음
    - Poly-Filter



# GNN

---

- Message passing framework
  - Message passing
    - 노드들의 피쳐 정보를 업데이트 하는 과정
      - 즉, 그래프의 구조적 정보와 노드의 특성 정보를 이용해서 노드의 임베딩 벡터를 계산하는 과정
    - 앞에서 살펴본 filtering 과정이라고 생각할 수 있음
    - What happens in a message passing iteration?
      - A hidden embedding corresponding to each node, which is denoted as  $u \in \mathcal{V}$ , is **updated** according to information **aggregated** from  $u$ 's graph neighborhood  $N(u)$

# Message Passing

## ■ A message passing iteration

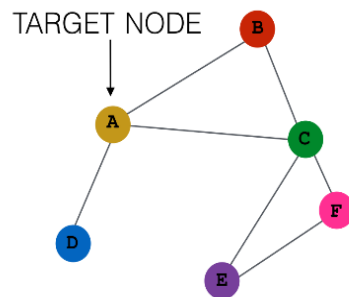
- 아래와 같이 표현될 수 있음

$$h_u^{(k+1)} = \text{UPDATE}^{(k)} \left( h_u^{(k)}, \text{AGGREGATE}^{(k)} \left( \{h_v^{(k)}, \forall v \in N(u)\} \right) \right) = \text{UPDATE}^{(k)} \left( h_u^{(k)}, m_{N(u)}^{(k)} \right)$$

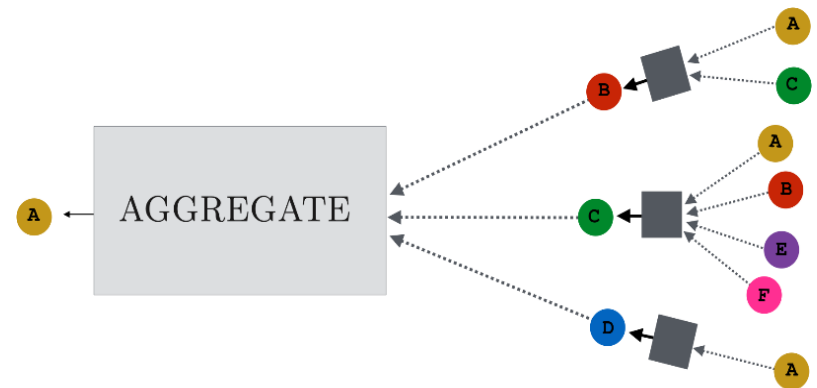
$$\text{즉, } m_{N(u)}^{(k)} = \text{AGGREGATE}^{(k)} \left( \{h_v^{(k)}, \forall v \in N(u)\} \right)$$

The initial embeddings at  $k = 0$  are set to the input features for all the nodes, i.e.,  $h_u^{(0)} = x_u, \forall u \in V$ .

- $u$ 의 이웃 노드들의 정보를 aggregate하고 자기 자신의 정보를 이용해서 update 한다고 생각할 수 있음
- 이러한 과정을 message passing이라고 함



INPUT GRAPH



GNN



# Message passing

---

- Message passing 결과물
  - After running  $K$  iterations of the GNN message passing, we can use the output of the final layer to define the embeddings for each node, i.e.,  $z_u = h_u^{(K)}, \forall u \in V$ .
  - 즉,  $z_u$ 를 이용해서 Node level 작업 (예, 노드 분류 등)을 수행할 수 있음
  - 뿐만 아니라,  $z_u$ 를 활용하여 그래프 단위의 작업도 수행 가능





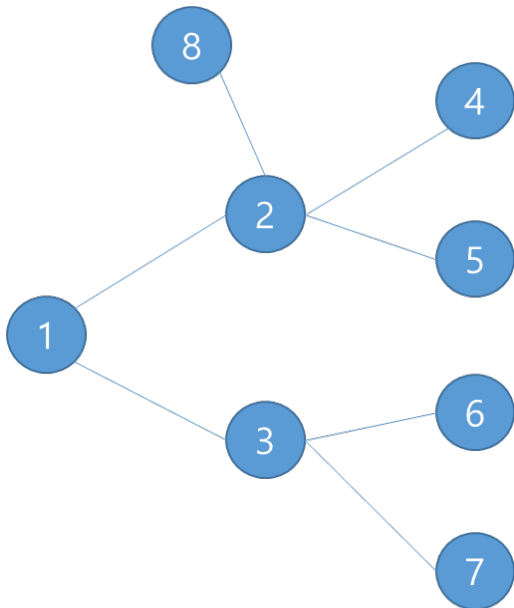
# Message passing

---

- Motivations and intuitions
  - At each iteration, every node aggregates information from its local neighborhood, and as these iterations progress each node embedding contains more and more information from further reaches of the graph.
  - 처음에는 (즉,  $k=1$ ) 1-hop 이웃 노드들의 정보 사용, 두 번째( $k=2$ )에는 2-hop 떨어진 이웃 노드들의 정보도 사용
  - 일반적으로 이러한 과정을  $k$ 번 수행하면  $k$ -hop 떨어진 노드들의 정보 까지 사용하게 됨

# Message passing

## ■ Example



- 첫 iteration
  - 1번 노드의 특성 (혹은 embedding)은 1-hop 떨어진 이웃 노드인 노드2와 3의 특성 정보를 이용해서 업데이트
  - 노드 2의 1-hop 이웃 노드인 노드 4, 5, 8의 특성 정보를 이용해서 임베딩 정보가 업데이트
- 두번째 iteration
  - 노드 1은 (역시나 마찬가지로) 1-hop 이웃 노드인 노드2와 3의 임베딩 정보를 이용해서 업데이트 됨
  - 그런데, 두 번째 iteration에서 사용되는 노드2와 3의 임베딩 정보는 이전 단계를 통해서 각각의 1-hop 이웃 노드들의 정보를 통해서 업데이트된 것임  $\Rightarrow$  따라서 노드 1의 임베딩 정보는 결국 노드 2, 3의 1 hop 이웃 노드의 정보를 반영해서 업데이트 된다는 것을 의미



# Message passing

---

- 이러한 임베딩 벡터는 어떠한 정보를 담고 있는가?
  - 두 가지 종류의 정보
    - 그래프의 구조적 정보
      - 노드들 간의 연결 정보
    - 이웃 노드의 특성 정보
      - The local feature-aggregation behavior of GNNs is analogous to the behavior of the convolutional kernels in CNNs. However, whereas CNNs aggregate feature information from spatially-defined patches in an image, GNNs aggregate information based on local graph neighborhoods.

- ### <첫 번째 iteration 의 경우>

**X**

12/11/23

**X**

$$=$$

**X'**



44

# Message passing

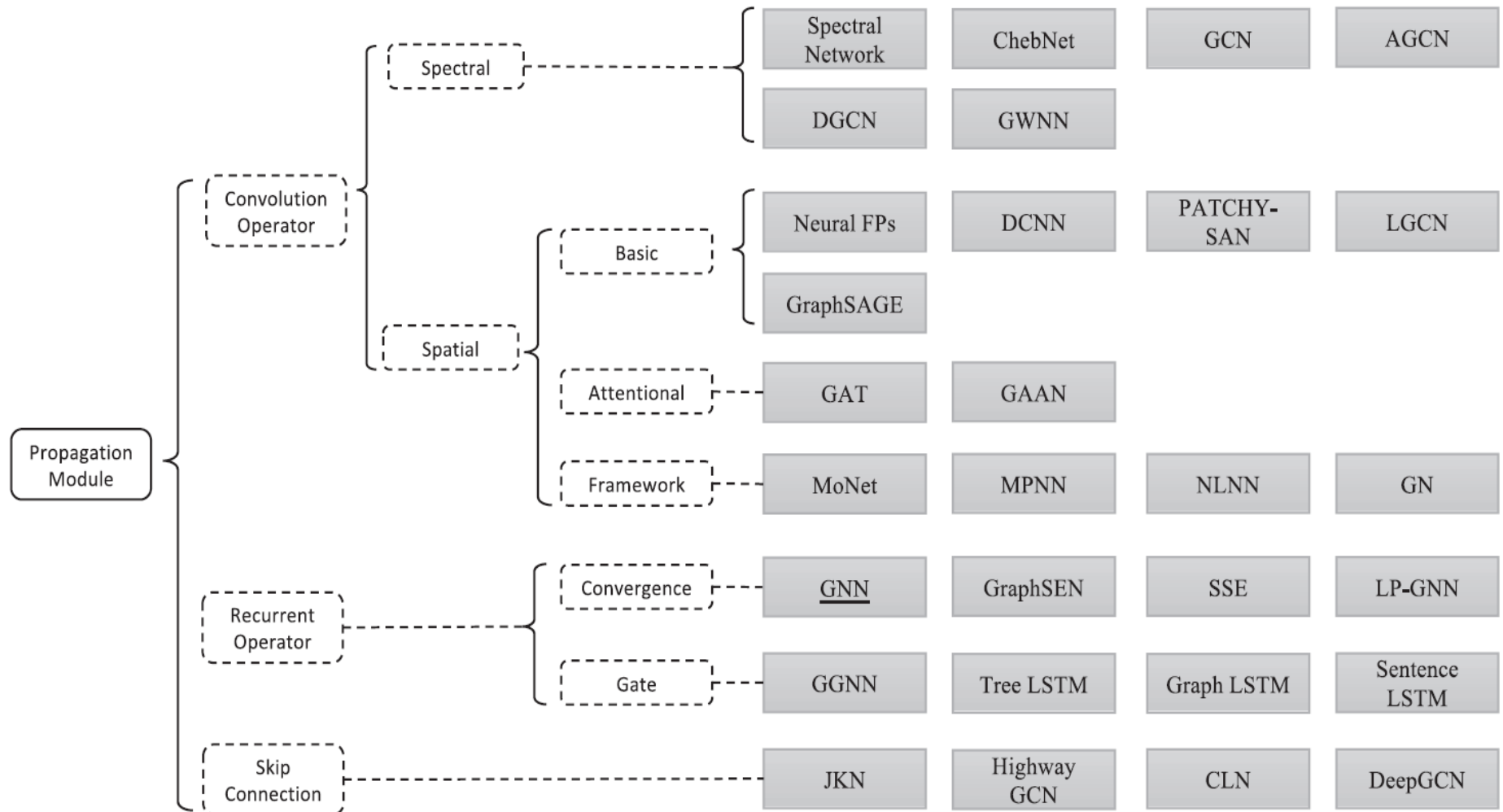
- 앞에서 살펴본 convolution filtering과의 비교 (cont'd)
  - 앞의 과정은, 특정 노드 (say 노드  $u$ )에 대해서 아래와 같이 표현 가능

$$h_u^{(k+1)} = \text{UPDATE} \left( h_u^{(k)}, \text{AGGREGATE} \left( \left[ h_v^{(k)}, \forall v \in N(u) \right] \right) \right)$$

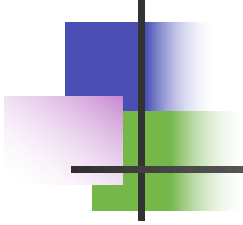
- 앞의 경우,  $\text{update}()$ 와  $\text{aggregate}()$  함수는 단순히 더하기를 하는 함수
  - 이는 convolution filter도 message passing 방법으로 표현할 수 있다는 것을 의미
  - 참고: 앞의 필터는 GCN의 필터와 유사
- 실제 분석에서의 message passing은 아래와 같이 가중치를 사용

$$h_u^{(k+1)} = \text{UPDATE} \left( W_1 h_u^{(k)}, \text{AGGREGATE} \left( \left[ W_2 h_v^{(k)}, \forall v \in N(u) \right] \right) \right)$$

Filtering 과정에 사용되는 방법들은 크게 Convolution 기반 방법과 Recurrent 기반 방법으로 구분 (참고: Filtering 과정을 propagation 과정이라고도 함)



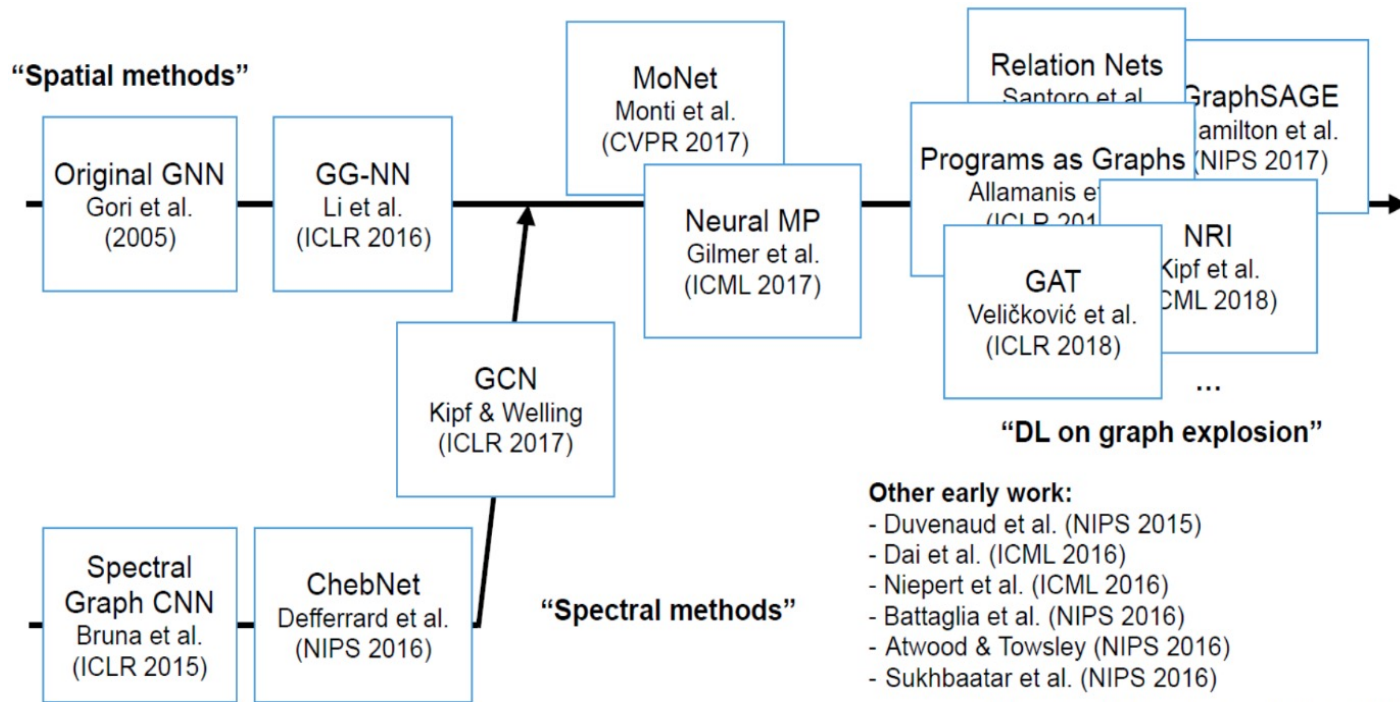
Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, 1, 57-81.



# Convolutional filters

# Convolutional filters

- Convolutional filter의 구분
  - spectral filter and spatial filter



(slide inspired by Alexander Gaunt's talk on GNNs)





# Convolutional filters

---

- Spectral vs. spatial filters
  - Spectral filters
    - The spectral-based graph filters utilize spectral graph theory to design the filtering operation in **the spectral domain**.
  - Spatial filters
    - The spatial-based graph filters **explicitly** (직접적으로) leverage the graph structure (i.e., the connections between the nodes) to perform the feature refining process in the graph domain.
  - 둘의 관계
    - These two categories of graph filters are closely related. In particular, some of the spectral-based graph filters can be regarded as spatial-based filters.
    - Example) GCN의 경우, spectral filter이기는 하지만, spatial filter로 간주 가능
    - 최근에는 spectral filter를 사용하는 빈도 감소 (하지만, 여전히 주요한 방법으로 사용되고 있음)



# Spectral filters

---

- Motivations
  - 그래프 데이터에 그대로 필터를 적용할 수 없다.  
Because it is permutation variant.
  - Spectral filter 적용 과정
    - Spatial domain에서의 feature 정보를 spectral domain에서 표현  
이를 위해 푸리에 변환 (Fourier transform) 수행
    - Spectral domain 에서 filter 적용
    - 그 결과를 다시 spatial domain 에서의 정보로 변환 (역 푸리에  
변환 사용)  $\Rightarrow$  이것이 업데이트된 특성 정보 (즉, 임베딩 벡터)
- Spectral filter는 Spectral graph theory와 Graph signal  
processing을 기반으로 작동



# Spectral graph theory

---

- What is it?
  - 그래프의 Laplacian 행렬의 고유값과 고유벡터를 분석해서 그래프의 특성을 파악하는 이론 또는 학문 분야
- 참고: 그래프를 행렬로 표현하는 방법 두 가지
  - 인접행렬 (adjacency matrix), Laplacian 행렬
- Laplacian 행렬의 정의

For a given graph  $G = \{V, E\}$ , its Laplacian matrix is defined as

$$L = D - A$$

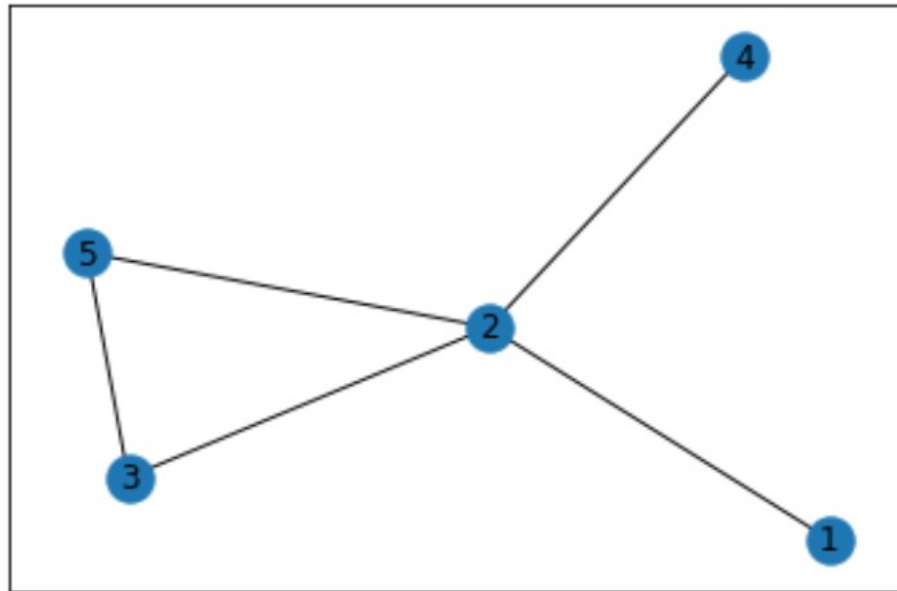
where  $D$  is a diagonal degree matrix  $D = \text{diag}(d(v_1), \dots, d(v_{|V|}))$

- Normalized Laplacian 행렬의 정의

$$L = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$$

# Spectral graph theory

- Laplacian 행렬 (cont'd)
  - 파이썬 코딩해 보기
    - graph\_basics.ipynb 참고
    - 예제 그래프





# Spectral graph theory

---

- Laplacian 행렬의 주요 특성 두 가지
  - 대칭행렬 (symmetric matrix)
    - 왜냐하면,  $D$ 하고  $A$  모두 대칭행렬이기 때문
    - 고유벡터 서로 수직
  - Positive semi-definite 행렬
    - 이를 위해서는 quadratic form에 대해서 알아야 함



# Quadratic form (이차형식)

---

- Quadratic form
  - 각 항에 존재하는 변수들의 지수 합이 2인 form
    - 예)  $4x^2 - xy + 3y^2$
  - Why use?
    - A quadratic form for which there exist established criteria for determining whether its sign is always positive, negative, nonpositive, or nonnegative.



# Positive and negative definiteness

---

- For  $q = au^2 + 2huv + bv^2$ ,
  - A quadratic form  $q$  is said to be
    - positive definite, if  $q$  is invariably positive,
    - positive semidefinite, if  $q$  is invariably nonnegative,
    - negative definite, if  $q$  is invariably negative,
    - negative semidefinite, if  $q$  is invariably nonpositive
  - regardless of the values of the variables i.e.,  $u$  and  $v$  in the quadratic form, not all zero.
  - indefinite  $\Rightarrow$  if  $q$  changes signs



# 행렬로 표현하기

---

- A quadratic form can be expressed using matrices
  - $q = au^2 + 2huv + bv^2 = \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} a & h \\ h & b \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$ 
    - $q = x'Ax$ , where  $A$  is a symmetric matrix
    - When  $q$  is positive (negative) (semi)definite, then  $A$  is also called positive (negative) (semi)definite.



# Spectral graph theory

- Laplacian 행렬의 주요 특성 두 가지
  - Positive semi-definite 행렬 (cont'd)
    - 노드들의 특성 정보를 나타내는 피쳐 벡터  $\mathbf{f}$  가정
    - $i$ 번째 원소 즉,  $\mathbf{f}[i]$  는 노드  $v_i$ 의 특성 정보
    - $\mathbf{f}^T \mathbf{L} \mathbf{f}$  계산하기 (This is a quadratic form)

$$\begin{aligned}\mathbf{f}^T \mathbf{L} \mathbf{f} &= \sum_{v_i \in \mathcal{V}} \mathbf{f}[i] \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}[i] - \mathbf{f}[j]) \\&= \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}[i] \cdot \mathbf{f}[i] - \mathbf{f}[i] \cdot \mathbf{f}[j]) \\&= \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} \left( \frac{1}{2} \mathbf{f}[i] \cdot \mathbf{f}[i] - \mathbf{f}[i] \cdot \mathbf{f}[j] + \frac{1}{2} \mathbf{f}[j] \cdot \mathbf{f}[j] \right) \\&= \frac{1}{2} \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}[i] - \mathbf{f}[j])^2.\end{aligned}$$

$\mathcal{N}(v_i)$ 는 노드  $v_i$ 는  $v_i$ 의 이웃 노드들

- $\mathbf{f}^T \mathbf{L} \mathbf{f}$  is the sum of the squares of the differences between adjacent nodes. In other words, it measures how different the values of adjacent nodes are.
- $\mathbf{f}^T \mathbf{L} \mathbf{f}$  는 항상 nonnegative => L is positive semi-definite



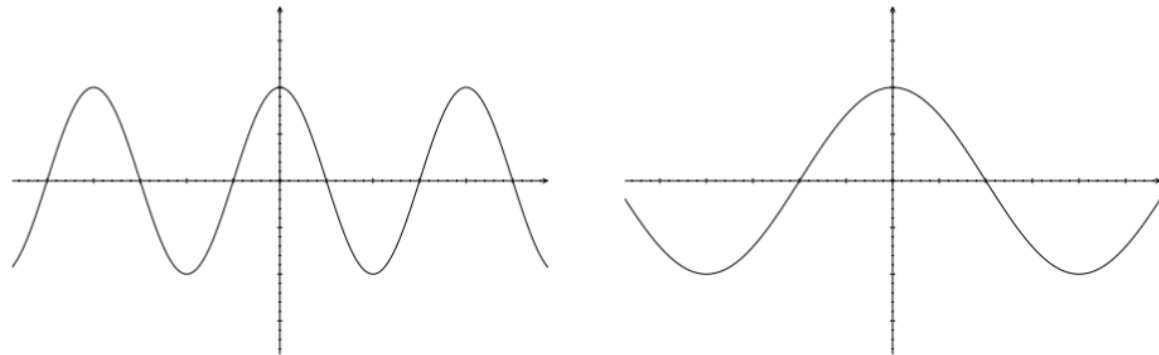
# Spectral graph theory

---

- Laplacian 행렬의 주요 특성 두 가지
  - Positive semi-definite 행렬 (cont'd)
    - 따라서
$$\mathbf{u}_l^T \mathbf{L} \mathbf{u}_l \geq 0, \text{ where } \mathbf{u}_l \text{는 } \mathbf{L} \text{의 } l \text{ 번째 고유벡터}$$
    - 그런데
$$\mathbf{u}_l^T \mathbf{L} \mathbf{u}_l = \lambda_l \text{ 이기 때문에 } \lambda_l \geq 0$$
  - 노드의 수 =  $N$  인 경우, 고유값/고유벡터의 수 =  $N$  (중복 가능)
  - 고유값 중 하나는 반드시 그 값이 0 (증명 생략)

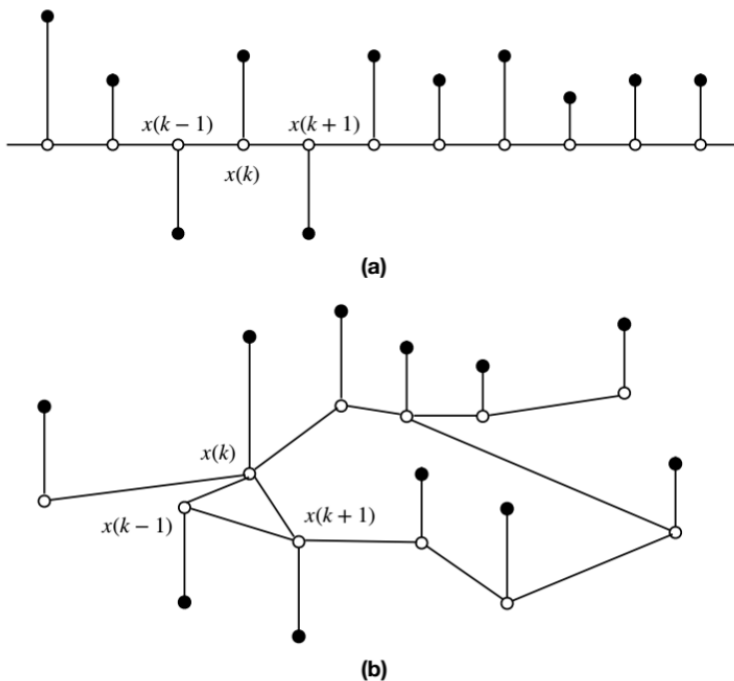
# Graph Signal Processing

- What is graph signal?
  - 그래프 시그널은 그래프를 구성하는 각 노드들이 갖는 특성 정보 혹은 속성 정보를 의미
  - 설명을 위해 각 노드가 갖는 특성 정보는 하나라고 가정 (여러 개인 경우도 동일하게 적용)
  - Feature vector,  $\mathbf{f}$  with  $\mathbf{f}[\mathbf{i}]$  corresponding to node  $v_i$ .
- Frequency (진동수/주파수)
  - Definition: the number of cycles per second
  - Higher frequency signals change faster, i.e., have greater variation, than signals with lower frequencies



# Graph Signal Processing

## ■ Frequency in the time and node domains

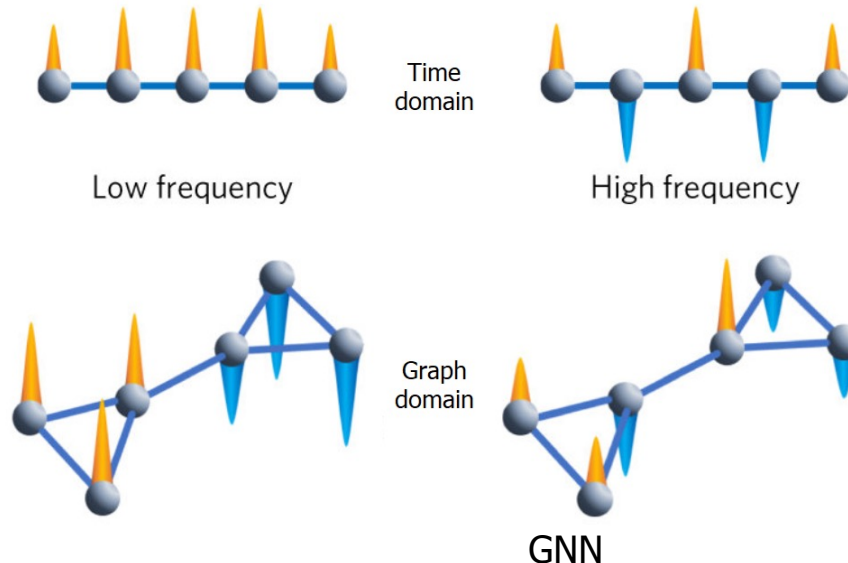


- The same signal seen as (a) observations in time and (b) observations placed on a graph. The signal in (a) can be viewed as being positioned on a line graph, while in (b) we have an arbitrary graph.
- Frequency in the time domain  $\Rightarrow$  이웃한 시간 간의 시그널 값의 차이의 절대값의 합
- Frequency in the graph domain  $\Rightarrow$  이웃한 노드 간 특성 정보 값의 차이의 절대값의 합

# Graph Signal Processing

## ■ Smoothness

- 연결된 노드들의 (feature) 값이 유사한 경우, 해당 graph는 smooth하다고 표현
- A smooth graph signal is low frequency, because the values change slowly across the graph via the edges.





# Graph Signal Processing

---

- Laplacian matrix quadratic form (i.e.,  $\mathbf{f}^T \mathbf{L} \mathbf{f}$ )
  - It can be utilized to measure the smoothness (or the frequency) of a graph signal  $\mathbf{f}$  because it is the summation of the square of the difference between all pairs of connected nodes.
  - When a graph signal  $\mathbf{f}$  is smooth,  $\mathbf{f}^T \mathbf{L} \mathbf{f}$  is small. The value  $\mathbf{f}^T \mathbf{L} \mathbf{f}$  is called the smoothness (or the frequency) of the signal  $\mathbf{f}$ .
- Signal in spatial and spectral domain
  - 전통적인 signal processing 에서 하나의 signal은 두개의 영역으로 표현, 즉, time domain과 frequency domain.
  - 유사하게, graph signal도 두개의 영역으로 표현 가능, 즉, spatial domain과 spectral domain (or frequency domain)
  - The spectral domain of graph signals is based on the graph Fourier transform. It is built upon the spectral graph theory.



# Graph Signal Processing

## ■ Graph Fourier transform

### ■ 정의

$$\hat{\mathbf{f}}[l] = \langle \mathbf{f}, \mathbf{u}_l \rangle = \sum_{i=1}^N \mathbf{f}[i] \mathbf{u}_l[i]$$

- $\mathbf{u}_l$ 은 해당 그래프의 Laplacian 행렬이 갖는  $l$  번째 고유벡터
- $N$ 은 노드의 수
- $\langle \mathbf{f}, \mathbf{u}_l \rangle$ 는  $\mathbf{f}$ 과  $\mathbf{u}_l$ 의 내적

### ■ 고유벡터와 고유값의 관계

- The corresponding eigenvalue  $\lambda_l$  represents the frequency or the smoothness of the eigenvector.

$$\mathbf{u}_l^T \mathbf{L} \mathbf{u}_l = \lambda_l \mathbf{u}_l^T \mathbf{u}_l = \lambda_l$$

$$\mathbf{u}_l^T \mathbf{L} \mathbf{u}_l = \frac{1}{2} \sum_{v_i \in V} \sum_{v_j \in N(v_i)} (\mathbf{u}[i] - \mathbf{u}[j])^2$$



# Graph Signal Processing

---

- Graph Fourier transform (cont'd)
  - 행렬 형태:  $\hat{\mathbf{f}} = \mathbf{U}^T \mathbf{f}$ 
    - $\hat{\mathbf{f}}$ 의 각 원소를 graph Fourier coefficient라고 함
    - The graph Fourier coefficients are the representation of the signal  $\mathbf{f}$  in the spectral domain.
- Inverse graph Fourier transform
  - $\mathbf{f} = \mathbf{U} \hat{\mathbf{f}}$
  - In summary, a graph signal can be denoted in two domains; i.e., the spatial domain and the spectral domain. The representations in the two domains can be transformed to each other via the graph Fourier transform and the inverse graph Fourier transform, respectively.



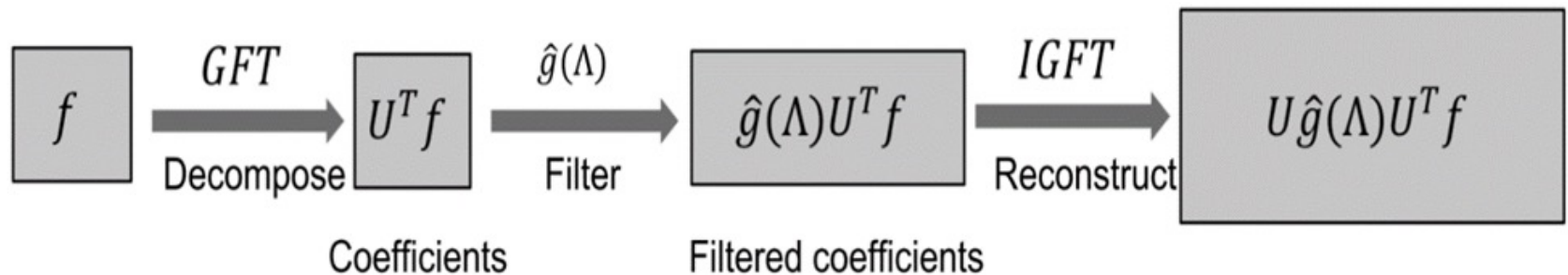
# Spectral-based graph filters

- Graph spectral filtering

- 목적

- 그래프가 갖고 있는 정보중에서 정답을 맞히는데 있어 중요한 역할을 하는 정보를 추출

- 과정





# Spectral-based graph filters

---

- $f'$ 에 필터를 적용하는 것의 의미
  - These graph Fourier coefficients describe how each graph Fourier component contributes to the graph signal  $f$ .
  - 예를 들어, 고유값이 0인 고유벡터에 해당하는 coefficient 값이 크다는 것은 원래 signal 이 smooth 하다는 것을 의미한다. 즉, 노드들 간의 값 차이가 크지 않다는 것을 의미.
  - 반대로 고유값이 큰 고유벡터에 해당하는 coefficient 값이 크다는 것은 원래 signal 이 smooth 하지 않다는 것 (즉, high frequency)을 의미, 즉, 노드들 간의 값 차이가 크다는 것을 의미
  - 관련 예제는 `graph_basics.ipynb` 참고
  - graph Fourier coefficients에 필터를 적용한다는 것은 coefficients 조절하겠다는 것을 의미  $\Rightarrow$  어떠한 값이 얼마만큼 조절 되는지는 학습을 통해 결정 (즉, 정답을 잘 맞추는 방향으로 조절)



# Spectral-based graph filters

- $f'$ 에 필터를 적용하는 것의 의미 (cont'd)
  - 예를 들어, smooth한 고유벡터의 역할을 증가시키면 (즉, 해당 coefficient 값을 증가시키면), 원래의 signal은 더 smooth하게 되고, less smooth한 고유벡터의 역할을 증가시키면, signal은 덜 smooth하게 변환 된다.
  - 이렇게 하면, 정답을 예측하는데 있어서 중요한 역할을 하는 정보 (노드 정보 혹은 연결 정보)를 더 잘 표현할 수 있다.
- $f'$ 에 필터 적용
  - $\hat{f}'[i] = \hat{f}[i]\gamma(\lambda_i)$ , for  $i = 1, \dots, N$
  - where  $\gamma(\lambda_i)$  is a function with the frequency  $\lambda_i$  as input, which determines how the corresponding frequency component should be modulated.  $\Rightarrow$  이는 해당 고유벡터의 역할 정도를 조절한다는 것을 의미한다. smooth한 고유벡터의 역할을 증가시키는 경우, 복원되는 signal도 더 smooth 해 진다.

# Spectral-based graph filters

- $f'$ 에 필터 적용 (cont'd)

- 행렬 형태:

$$\hat{\mathbf{f}}' = \gamma(\Lambda) \cdot \hat{\mathbf{f}} = \gamma(\Lambda) \cdot \mathbf{U}^T \mathbf{f}, \quad \Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{pmatrix}; \quad \gamma(\Lambda) = \begin{pmatrix} \gamma(\lambda_1) & & 0 \\ & \ddots & \\ 0 & & \gamma(\lambda_N) \end{pmatrix}.$$

- $\hat{\mathbf{f}}'$ 을 이용해서 다시 signal을 spatial domain으로 복원

- $\mathbf{f}' = \mathbf{U}\hat{\mathbf{f}}' = \mathbf{U} \cdot \gamma(\Lambda) \cdot \mathbf{U}^T \mathbf{f}$ , where  $\mathbf{f}'$  is the obtained filtered graph signal.
    - The filtering process can be regarded as applying the operator  $\mathbf{U} \cdot \gamma(\Lambda) \cdot \mathbf{U}^T$  to the input graph signal. For convenience, we sometimes refer to the function  $\gamma(\Lambda)$  as the filter because it controls how each frequency component of the graph signal  $\mathbf{f}$  is filtered.
    - $\gamma(\Lambda)$ 으로 무엇을 사용하느냐에 따라 모형의 종류가 달라짐
    - For example, in the extreme case, if  $(\lambda_i)$  equals 0, then  $\hat{\mathbf{f}}'[i] = 0$  and the frequency component  $u_i$  is removed from the graph signal  $\mathbf{f}$ . => 복원될 때 아무런 역할을 하지 못한다.

# Spectral-based graph filters

- 필터,  $\gamma(\Lambda)$ , 의 형태
  - 필터의 역할
    - 시그널이 갖고 있는 frequency를 조절
  - 시그널이 갖고 있는 어떠한 frequency를 얼마나 조절할지는 학습을 통해서 결정
  - 따라서,  $\gamma(\Lambda)$ 는 파라미터(들)의 함수로 표현  $\Rightarrow$  파라미터의 값들은 학습을 통해서 결정
  - 가장 간단하게 생각할 수 있는 형태 (Bruna et al., 2013)
    - $\gamma(\lambda_i) = \theta_i$
    - 따라서,  $\hat{\mathbf{f}}'[i] = \hat{\mathbf{f}}[i]\theta_i, \text{ for } i = 1, \dots, N$

$$\gamma(\Lambda) = \begin{pmatrix} \theta_1 & & 0 \\ & \ddots & \\ 0 & & \theta_N \end{pmatrix}.$$

Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.



# Spectral-based graph filters

- Poly-Filter

- 앞의 필터의 단점은?
- To address these issues, a polynomial filter operator, which we denote as Poly-Filter, was proposed in Defferrard et al. (2016).
- Poly-Filter

$$\gamma(\lambda_l) = \sum_{k=0}^K \theta_k \lambda_l^k = \theta_0 + \theta_1 \lambda_l + \theta_2 \lambda_l^2 + \dots + \theta_K \lambda_l^K$$

- 행렬 형태

$$\gamma(\mathbf{\Lambda}) = \sum_{k=0}^K \theta_k \mathbf{\Lambda}^k$$

- 파라미터수 = K+1

- Furthermore, we can show that  $\mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T$  can be simplified to be a **polynomial of the Laplacian matrix**.

This means that (1) no eigendecomposition is needed and (2) the polynomial parameterized filtering operator is spatially localized; i.e., the calculation of each element of the output  $f'$  only involves a small number of nodes in the graph.

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29.



# Spectral-based graph filters

---

- Poly-Filter (cont'd)

- By applying this Poly-Filter operator on  $\mathbf{f}$ , we can get the output  $\mathbf{f}'$  as follows:

$$\mathbf{f}' = \mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T \mathbf{f} = \mathbf{U} \cdot \sum_{k=0}^K \theta_k \mathbf{\Lambda}^k \cdot \mathbf{U}^T \mathbf{f} = \sum_{k=0}^K \theta_k \mathbf{U} \cdot \mathbf{\Lambda}^k \cdot \mathbf{U}^T \mathbf{f}$$

- 여기에서  $\mathbf{U} \cdot \mathbf{\Lambda}^k \cdot \mathbf{U}^T = \mathbf{L}^k$



# Spectral-based graph filters

$L$ 이 symmetric이기 때문에  $U^T = U^{-1}$

$$\Lambda = \Lambda I = \Lambda U^{-1} U = \Lambda U^T U$$

따라서

$$\begin{aligned} U \cdot \Lambda^k \cdot U^T &= U \cdot (\Lambda U^T U)^k \cdot U^T \\ &= (U \cdot \Lambda \cdot U^T) \cdots (U \cdot \Lambda \cdot U^T) = L^k \end{aligned}$$

따라서

$$\begin{aligned} \mathbf{f}' &= \sum_{k=0}^K \theta_k U \cdot \Lambda^k \cdot U^T \mathbf{f} = \sum_{k=0}^K \theta_k L^k \mathbf{f} \\ &= \theta_0 + \theta_1 L \mathbf{f} + \theta_2 L^2 \mathbf{f} + \cdots + \theta_K L^K \mathbf{f} \end{aligned}$$

여기서  $L^k \mathbf{f}$ 는 k-hop distant 이웃 노드들의 특성 정보까지 사용한다는 것 => 이것 관련해서

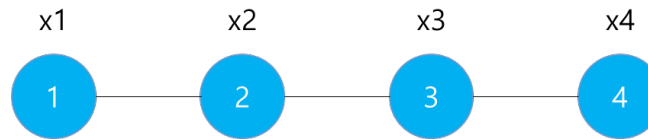
<https://towardsdatascience.com/the-intuition-behind-graph-convolutions-and-message-passing-6dcd0ebf0063> 의 내용을 참고할 것



# Spectral-based graph filters

- Poly-Filter (cont'd)

- Example



$$L = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

$$L^2 = L \cdot L = \begin{bmatrix} 2 & -3 & 1 & 0 \\ -3 & 6 & -4 & 1 \\ 1 & -4 & 6 & -3 \\ 0 & 1 & -3 & 2 \end{bmatrix}$$

$$Lf = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1 - x_2 \\ 2x_2 - (x_1 + x_3) \\ 2x_3 - (x_2 + x_4) \\ x_4 - x_3 \end{bmatrix}$$



# Spectral-based graph filters

---

- Poly-Filter (cont'd)
  - 주요 단점
    - The basis of the polynomial is not an orthogonal basis.  
Hence, the coefficients are dependent on each other. =>  
학습 불안정



# Spectral-based graph filters

---

- Cheby-Filter

- Based on Chebyshev polynomials,  $T_k(y)$
- Chebyshev polynomials

$$T_k(y) = 2yT_{k-1}(y) - T_{k-2}(y)$$

$$\text{with } T_0(y) = 1, T_1(y) = y$$

For  $y \in [-1, 1]$ ,  $T_k(y)$  is bounded in  $[-1, 1]$

- 주요 특징
  - Chebyshev polynomials are orthogonal to each other



# Spectral-based graph filters

---

- Cheby-Filter (cont'd)

- GNN의 경우

- $T_k(\lambda_l)$

- 따라서,  $\tilde{\lambda}_l = \frac{2\lambda_l}{\lambda_{max}} - 1$ 을 사용  $\Rightarrow \tilde{\lambda}_l \in [-1, 1]$

- 행렬의 행태

- $\tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - \mathbf{I}$

- Cheby-Filter

$$\gamma(\Lambda) = \sum_{k=0}^K \theta_k T_k(\tilde{\Lambda})$$

# Spectral-based graph filters

- Cheby-Filter (cont'd)

- The process of applying the Cheby-Filter on a graph signal  $\mathbf{f}$  can be defined as:

$$\mathbf{f}' = \mathbf{U} \sum_{k=0}^K \theta_k T_k(\tilde{\Lambda}) \mathbf{U}^T \mathbf{f} = \sum_{k=0}^K \theta_k \mathbf{U} T_k(\tilde{\Lambda}) \mathbf{U}^T \mathbf{f}$$

- 위 식에서  $\mathbf{U} T_k(\tilde{\Lambda}) \mathbf{U}^T = T_k(\tilde{\mathbf{L}})$ , where  $\tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I}$
- 따라서

$$\mathbf{f}' = \sum_{k=0}^K \theta_k \mathbf{U} T_k(\tilde{\Lambda}) \mathbf{U}^T \mathbf{f} = \sum_{k=0}^K \theta_k T_k(\tilde{\mathbf{L}}) \mathbf{f} = \theta_0 \mathbf{I} + \theta_1 T_1(\tilde{\mathbf{L}}) \mathbf{f} + \theta_2 T_2(\tilde{\mathbf{L}}) \mathbf{f} + \dots + \theta_K T_K(\tilde{\mathbf{L}}) \mathbf{f}$$

where  $T_k(\tilde{\mathbf{L}}) = 2\tilde{\mathbf{L}}T_{k-1}(\tilde{\mathbf{L}}) - T_{k-2}(\tilde{\mathbf{L}})$ , with  $T_0(\tilde{\mathbf{L}}) = \mathbf{I}, T_1(\tilde{\mathbf{L}}) = \tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I}$

- Hence, the Cheby-Filter still enjoys the advantages of Poly-Filter, 하지만 학습 과정에서 더욱 안정적

# Spectral-based graph filters

- Cheby-Filter (from the ordinary NN perspective)

Start with the original features.

$$\mathbf{h}^{(0)} = \mathbf{x}$$

This process is similar to that of message passing.

Color Codes:

- Computed node embeddings.
- Learnable parameters.

Then iterate, for  $k = 1, 2, \dots$  upto  $K$ :

$$p_w(L) = \sum_{i=1}^d w_i T_i(\tilde{L})$$
$$p^{(k)} = p_{w^{(k)}}(L)$$
$$g^{(k)} = p^{(k)} \times \mathbf{h}^{(k-1)}$$
$$\mathbf{h}^{(k)} = \sigma \left( g^{(k)} \right)$$

Compute the matrix  $p^{(k)}$  as the polynomial defined by the filter weights  $w^{(k)}$  evaluated at  $L$ .

Multiply  $p^{(k)}$  with  $\mathbf{h}^{(k-1)}$ : a standard matrix-vector multiply operation.

Apply a non-linearity  $\sigma$  to  $g^{(k)}$  to get  $\mathbf{h}^{(k)}$ .

<https://distill.pub/2021/understanding-gnns/>



# Spectral-based graph filters

- GCN-Filter: Simplified Cheby-Filter Involving 1-Hop Neighbors
  - The Cheby-Filter involves a K-hop neighborhood of a node when calculating the new features for the node.
  - In Kipf and Welling (2016), a simplified ChebyFilter named GCN-Filter is proposed.
  - GCN simplifies Cheby-Filter by setting the order of Chebyshev polynomials to  $K = 1$  and  $\lambda_{max} = 2$ .

$$\gamma(\Lambda) = \theta_0 T_0(\tilde{\Lambda}) + \theta_1 T_1(\tilde{\Lambda}) = \theta_0 \mathbf{I} + \theta_1 \tilde{\Lambda}, \text{ where } \tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - \mathbf{I} = \frac{2\Lambda}{2} - \mathbf{I} = \Lambda - \mathbf{I}$$

$$\text{따라서 } \gamma(\Lambda) = \theta_0 \mathbf{I} + \theta_1 (\Lambda - \mathbf{I})$$

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

# Spectral-based graph filters

- GCN-Filter (cont'd)

- 따라서 입력 시그널  $\mathbf{f}$ 에 GCN 필터를 적용하게 되면 결과로 출력 시그널  $\mathbf{f}'$ 를 다음과 같이 얻는다.

$$\begin{aligned}\mathbf{f}' &= \mathbf{U}\gamma(\boldsymbol{\Lambda})\mathbf{U}^T\mathbf{f} = \mathbf{U}\{\theta_0\mathbf{I} + \theta_1(\boldsymbol{\Lambda} - \mathbf{I})\}\mathbf{U}^T\mathbf{f} \\ &= \mathbf{U}\theta_0\mathbf{I}\mathbf{U}^T\mathbf{f} + \mathbf{U}\theta_1(\boldsymbol{\Lambda} - \mathbf{I})\mathbf{U}^T\mathbf{f} \\ &= \theta_0\mathbf{U}\mathbf{I}\mathbf{U}^T\mathbf{f} + \theta_1\mathbf{U}(\boldsymbol{\Lambda} - \mathbf{I})\mathbf{U}^T\mathbf{f} \\ &= \theta_0\mathbf{f} - \theta_1(\mathbf{L} - \mathbf{I})\mathbf{f} = \theta_0\mathbf{f} - \theta_1(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{f}\end{aligned}$$

- 옆의 식에서는 normalized Laplacian 행렬이 사용 (따라서 마지막 부분이 성립).
- normalized Laplacian 행렬:

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$$

- 추가적으로  $\theta = \theta_0 = -\theta_1$ 라고 하면, 해당 식은 아래와 같음.

$$\mathbf{f}' = \theta_0\mathbf{f} - \theta_1(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{f} = \theta(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{f}$$





# Spectral-based graph filters

- GCN-Filter (cont'd)

- 그런데 앞의 식에서  $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ 의 고유값은  $[0,2] \Rightarrow$  이렇게 되면 학습이 불안정해질 수 있음
- 이러한 문제를 해소하기 위해, 해당 논문에서는 다시 한번 normalization trick을 적용  $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$  대신  $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$  을 사용, where  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$  and  $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{i,j}$
- 따라서, 최종 GCN-Filter의 형태는 다음과 같음

$$\mathbf{f}' = \theta \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{f}$$

참고: The  $i, j$ th element of  $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$  is nonzero only when nodes  $v_i$  and  $v_j$  are connected.

- For a single node, this process can be viewed as aggregating information from its 1-hop neighbors where the node itself is also regarded as its 1-hop neighbor. Thus, the GCN-Filter can also be viewed as a spatial-based filter, which only involves directly connected neighbors when updating node features.

# Spectral-based graph filters

- GCN (from a message passing perspective)

$$h_v^{(0)} = x_v \quad \text{for all } v \in V.$$

Node  $v$ 's  
initial  
embedding.

... is just node  $v$ 's  
original features.

(원 논문에서의 normalization)

$$f \left( W \cdot \sum_{u \in \mathcal{N}(v)} \frac{h_u}{\sqrt{|\mathcal{N}(u)| |\mathcal{N}(v)|}} + B \cdot h_v \right)$$

원 버전에서는 이 둘을 공유

and for  $k = 1, 2, \dots$  upto  $K$ :

$$h_v^{(k)} = f^{(k)} \left( W^{(k)} \cdot \frac{\sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}}{|\mathcal{N}(v)|} + B^{(k)} \cdot h_v^{(k-1)} \right) \quad \text{for all } v \in V.$$

Node  $v$ 's  
embedding at  
step  $k$ .

Mean of  $v$ 's  
neighbour's  
embeddings at  
step  $k - 1$ .

Node  $v$ 's  
embedding at  
step  $k - 1$ .



# Spectral-based graph filters

## ■ General framework of spectral filters

Start with the original features.

$$\textcolor{brown}{h}^{(0)} = x$$

Color Codes:

■ Computed node embeddings.

■ Learnable parameters.

Then iterate, for  $k = 1, 2, \dots$  upto  $K$ :

$$\hat{h}^{(k-1)} = U_m^T \textcolor{brown}{h}^{(k-1)}$$

Convert previous feature  $\textcolor{brown}{h}^{(k-1)}$  to its spectral representation  $\hat{h}^{(k-1)}$ .

$$\hat{g}^{(k)} = \hat{w}^{(k)} \odot \hat{h}^{(k-1)}$$

Convolve with filter weights  $\hat{w}^{(k)}$  in the spectral domain to get  $\hat{g}^{(k)}$ .  
 $\odot$  represents element-wise multiplication.

$$g^{(k)} = U_m \hat{g}^{(k)}$$

Convert  $\hat{g}^{(k)}$  back to its natural representation  $g^{(k)}$ .

$$\textcolor{brown}{h}^{(k)} = \sigma \left( g^{(k)} \right)$$

Apply a non-linearity  $\sigma$  to  $g^{(k)}$  to get  $\textcolor{brown}{h}^{(k)}$ .