Graph Neural Networks:

Introduction to Spectral Graph Convolution

Sangmin Kim

2022. 07. 15

Introduction

김상민 (Sangmin Kim)

- Data Mining & Quality Analytics Lab
  - ✓ M.S. Student (2021.03 ~)
- Research Interest
  - ✓ Graph Neural Network
  - ✓ Human Pose Estimation
- E-mail: sanmiz@korea.ac.kr

Contents

Chapter 1

- Graph Neural Networks
- Graph structure
- Graph data

Chapter 2

- Convolution theorem
- Graph Fourier transform
- Spectral Graph Convolution

Chapter 3

- Spectral Graph CNN (Brunaet al. ICLR 2014)
- ChebNet (Defferardet al. NIPS 2016)
- Simplified ChebNet (Kipf & Welling, ICLR 2017)

Graph Neural Networks

Graph Neural Networks

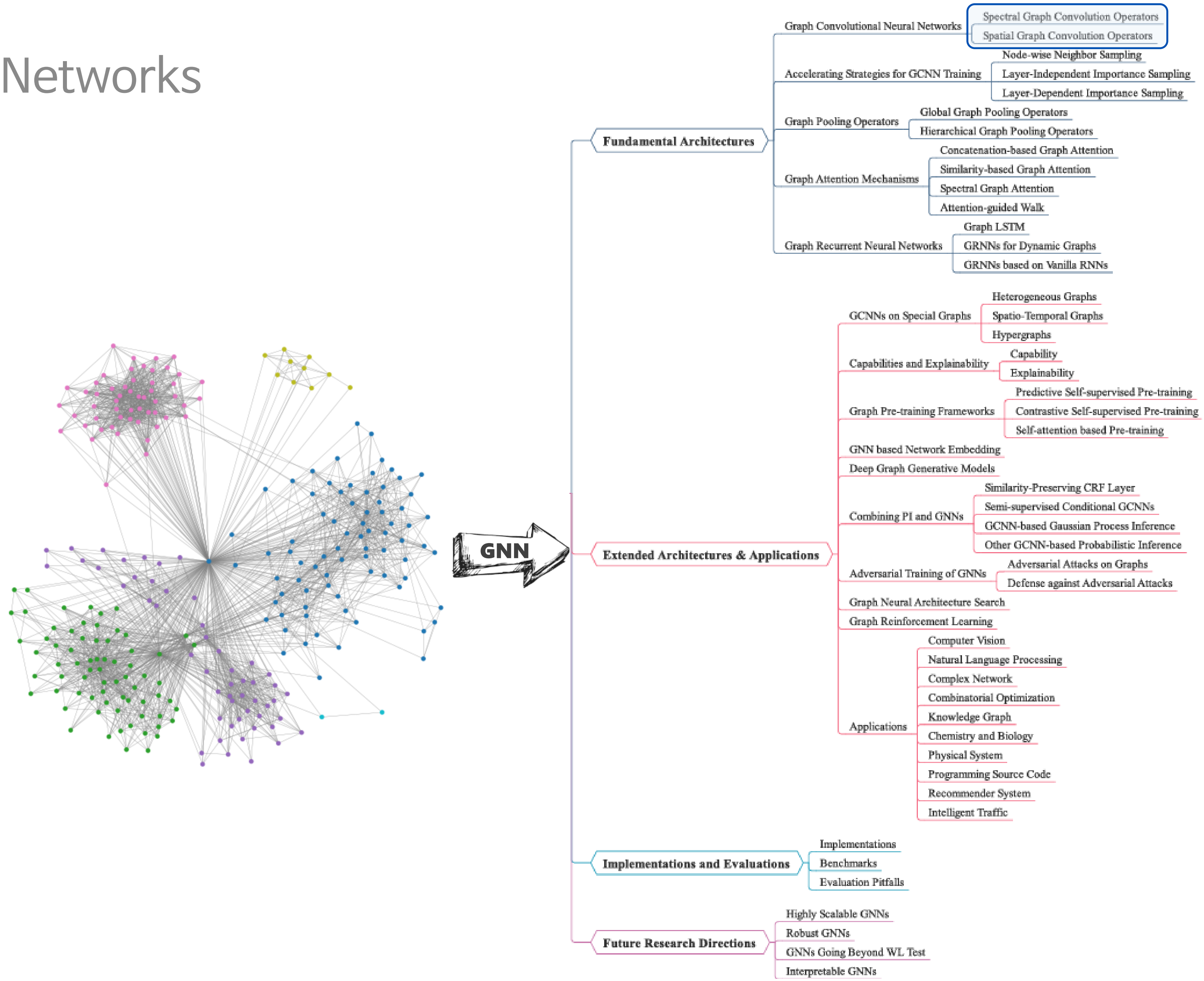
- A type of Neural Network which directly operates on the Graph structure
- In Computer Science, a graph is a data structure consisting of two components, vertices (nodes) and edges

Graph of social network

Image from GDJ, via Pixabay

Graph Neural Networks

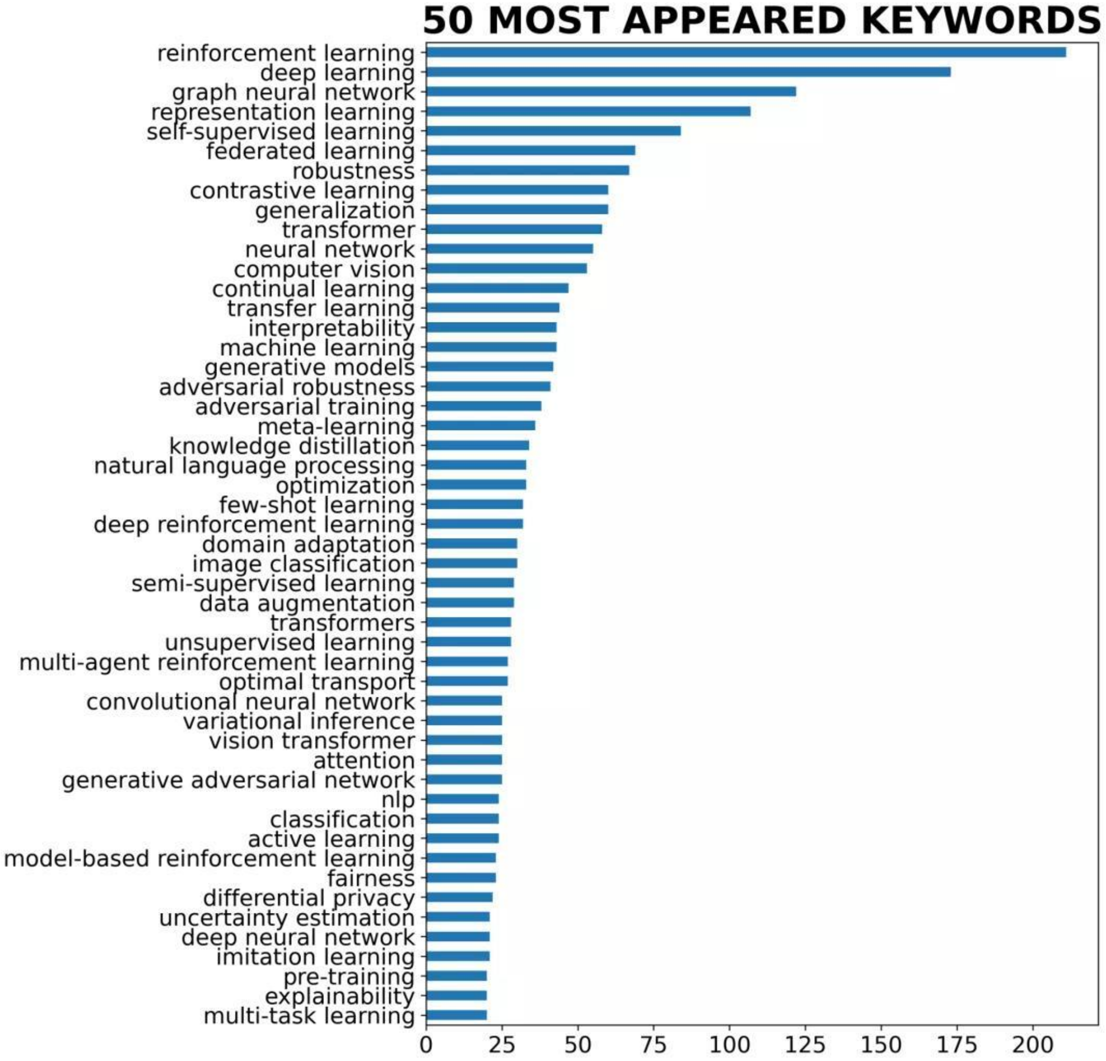
- Taxonomy



Zhou, Yu, et al. "Graph Neural Networks: Taxonomy, Advances, and Trends." *ACM Transactions on Intelligent Systems and Technology (TIST)* 13.1 (2022): 1-54.

Graph Neural Networks

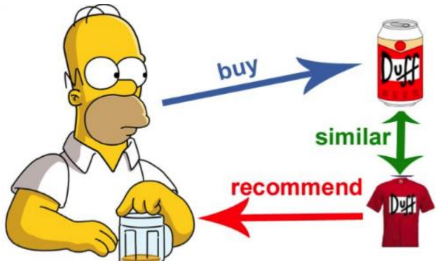
A project ICLR2022-OpenReviewData on GitHub crawled all ICLR 2022 submitted papers (3,407)



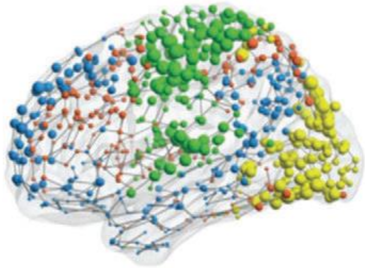
https://inf.news/en/tech/109cef8ab80cc02f9c675b6448bc49df.html

Graph Neural Networks

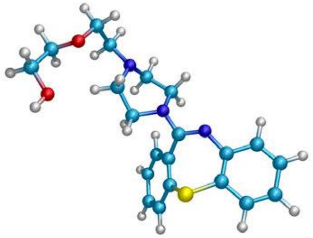
- Application



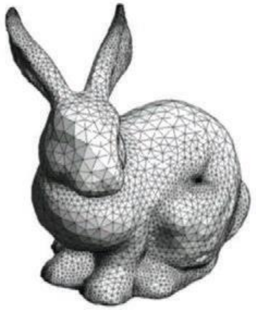
Recommender systems



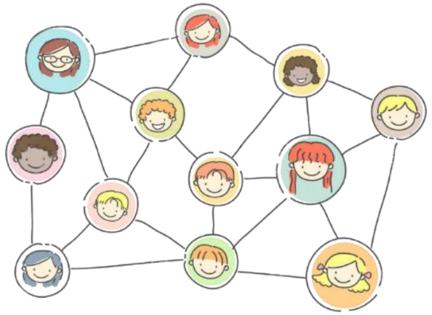
Brain connectivity
(Neuroscience)



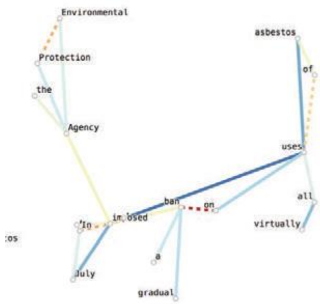
Drug/material molecules
(Chemistry)



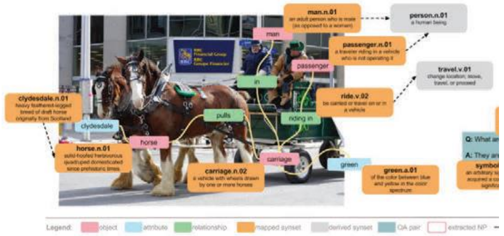
3D meshes
(Computer Graphics)



Social networks
(Advertisement)



World relationships
(NLP)

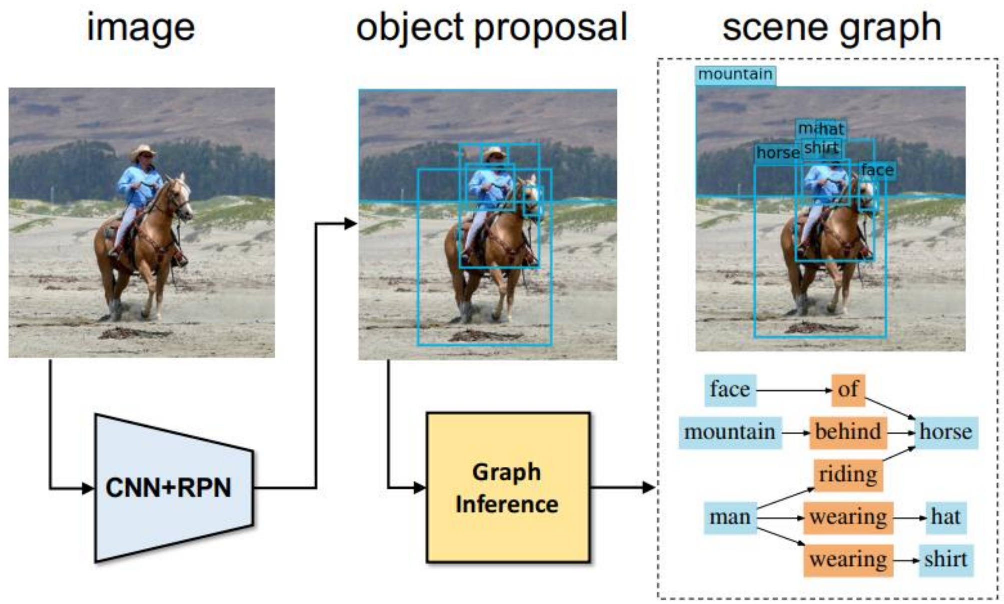


Scene understanding
(Computer Vision)

Transportation networks

Graph Neural Networks

- Application
  - GNN in Computer Vision



Scene graph from relationships between objects in image[1]

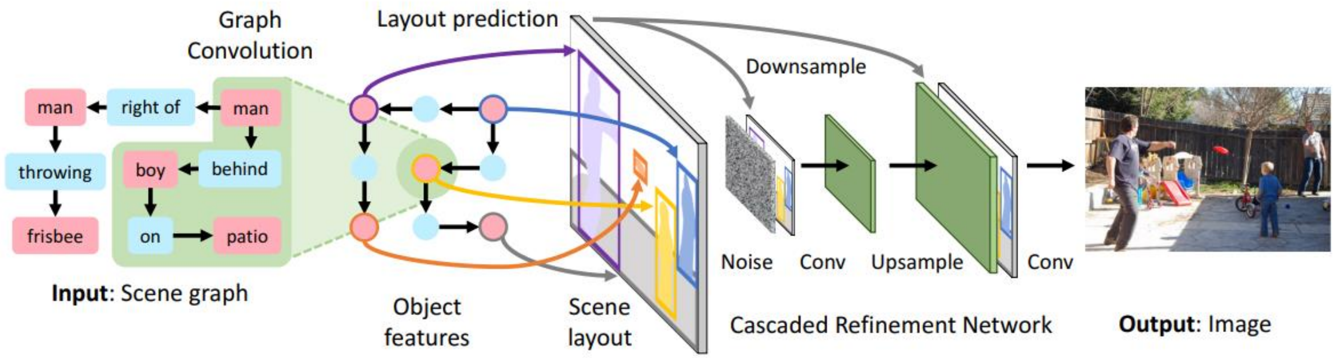


Image generation from a scene graph[2]

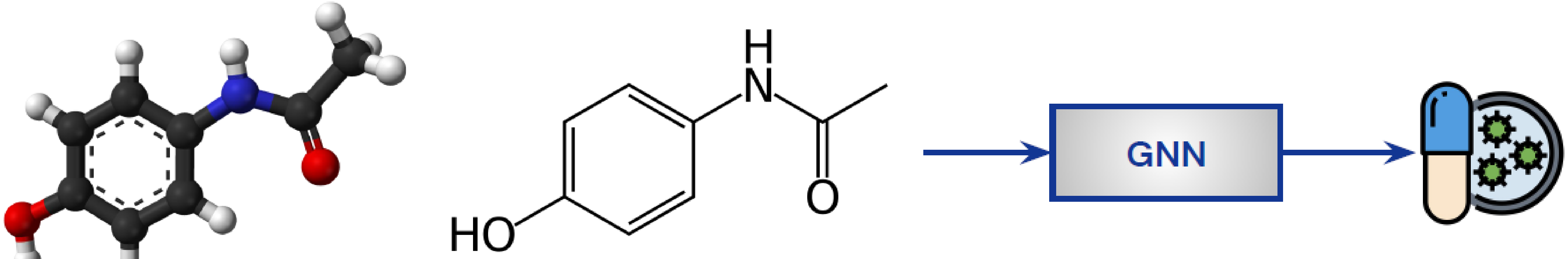
Scene Graph Generation. Figure from D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, “Scene graph generation by iterative message passing,” in *Proc. of CVPR*, 2017

Image generated from scene graphs. Figure from J. Johnson, A. Gupta, and L. Fei-Fei, “Image generation from scene graphs,” in *Proc. of CVPR*, 2018

Graph Neural Networks

- Application
  - GNN in Biology

Molecule이 효능 있는 약(potent drug)이 될 수 있는지를 예측



Molecule(graph)

Inhibist E.coli?

Atoms (nodes), Bonds (edges)

Atom type, charge, bond type (features)

E.coli (대장균) 박테리아가 살 수 있는 약인지를 이진 분류

https://petar-v.com/talks/GNN-Wednesday.pdf , Petar Veličković, DeepMind

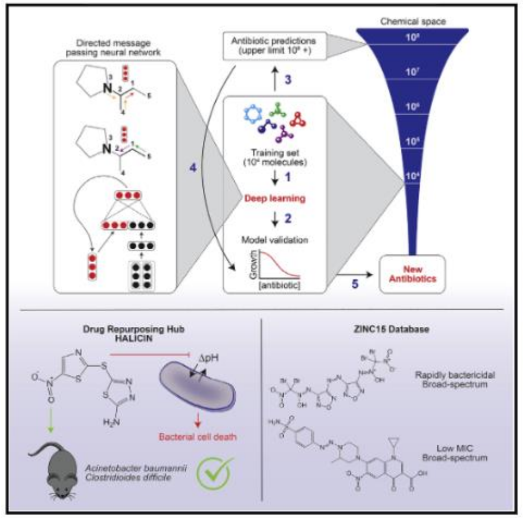
Graph Neural Networks

- Application
  - GNN in Biology

Cell

A Deep Learning Approach to Antibiotic Discovery

Graphical Abstract



Authors

Jonathan M. Stokes, Kevin Yang, Kyle Swanson, ..., Tommi S. Jaakkola, Regina Barzilay, James J. Collins

Correspondence

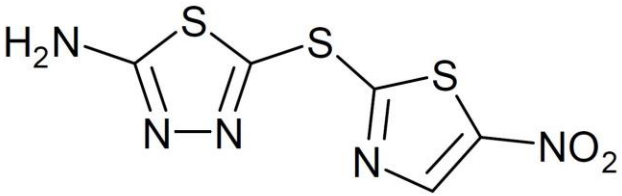
regina@csail.mit.edu (R.B.), jimjc@mit.edu (J.J.C.)

In Brief

A trained deep neural network predicts antibiotic activity in molecules that are structurally different from known antibiotics, among which Halicin exhibits efficacy against broad-spectrum bacterial infections in mice.

Highlights

- A deep learning model is trained to predict antibiotics based on structure
- Halicin is predicted as an antibacterial molecule from the Drug Repurposing Hub
- Halicin shows broad-spectrum antibiotic activities in mice
- More antibiotics with distinct structures are predicted from the ZINC15 database



Halicin

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄ Subscribe

nature > news > article

NEWS | 20 February 2020

Powerful antibiotics discovered using AI

Machine learning spots molecules that work even against 'untreatable' strains of bacteria.

Jo Marchant

AI로 내성균 잡는 강력 항생제 발견

유익한 박테리아는 죽지 않게 최적화 방법 모색

2020.02.21 11:34 | 김병희 객원기자

미국 매사추세츠공대(MIT) 연구팀이 기계-학습 알고리즘을 사용해 항생제 내성균을 죽일 수 있는 강력한 새로운 항생제를 발견했다.

실험실 테스트에서 이 새 항생제는 기존의 모든 항생제에 내성을 보이는 몇몇 세균들을 포함해, 전 세계에서 문제가 되고 있는 병원성 박테리아를 모두 사멸시킨 것으로 나타났다.

이 항생제는 두 종류의 생쥐 모델에서 일으킨 감염도 모두 제거했다.

연구팀은 1억 개 이상의 화합물을 며칠 만에 선별할 수 있는 컴퓨터 모델을 고안해, 기존 항생제와는 전혀 다른 기전으로 박테리아를 죽일 수 있는 잠재적인 항생제들을 선별해 내는 성과를 올렸다.

이 연구는 생명과학저널 '셀(Cell)' 20일 자에 발표됐다.

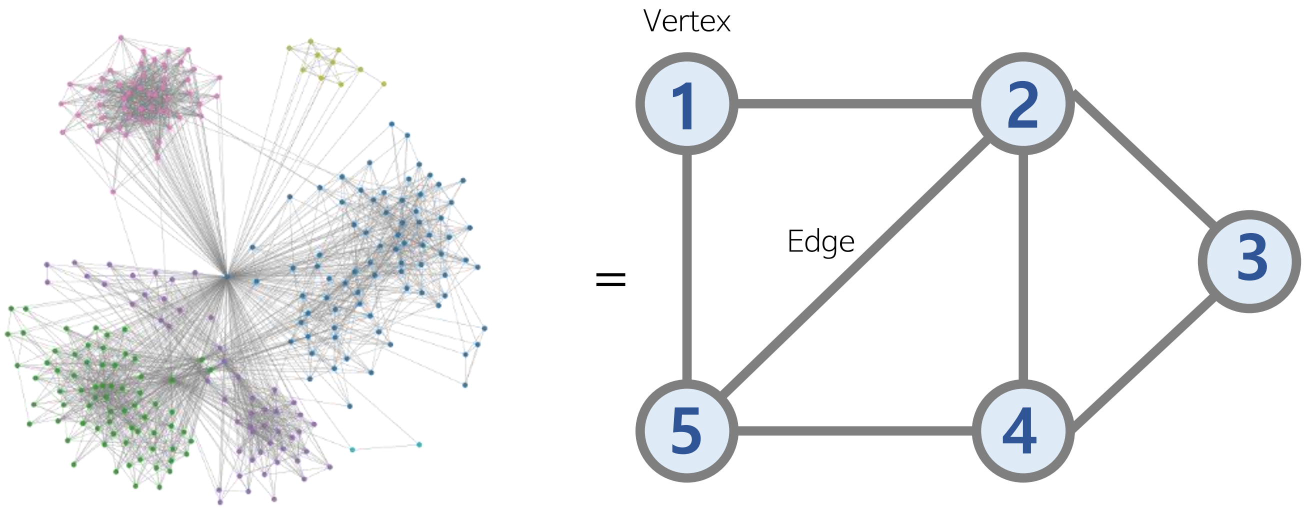
MIT 의공학 및 의과학 연구소(IMES) 제임스 콜린스(James Collins) 석좌교수(의공학, 의과학, 생물공학)는 "인공지능을 이용해 새 시대의 항생제 발견을 선도할 수 있는 플랫폼을 개발하고 싶었다"고 말하고, "우리가 고안한 접근법으로 지금까지 발견된 항생제 중 가장 강력하고 놀라운 분자를 찾아냈다"고 밝혔다.

Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., ... & Collins, J. J. (2020). A deep learning approach to antibiotic discovery. Cell, 180(4), 688-702.
https://www.nature.com/articles/d41586-020-00018-3
https://www.sciencetimes.co.kr/news/ai%EB%A1%9C-%EB%82%B4%EC%84%B1%EA%B7%A0-%EC%9E%A1%EB%8A%94-%EA%B0%95%EB%A0%A5-%ED%95%AD%EC%83%9D%EC%A0%9C-%EB%B0%9C%EA%B2%AC/

Graph structure

Graph structure

- Graph G are defined by:
  - Vertices V (node)
  - Edges E:
    - Directed/ undirected
    - Weighted/ unweighted
  - Adjacency matrix A: $n \times n$으로 표현, {1,0}

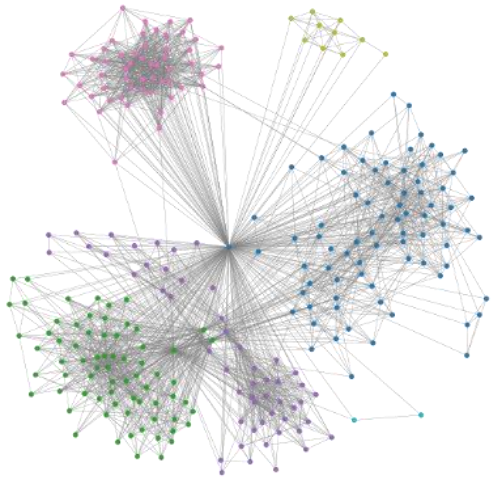


$$G = (V, E)$$

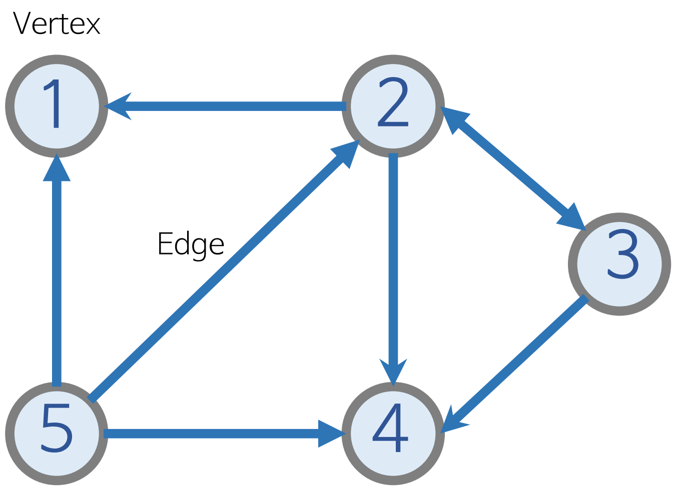
http://dmqm.korea.ac.kr/uploads/seminar/Openseminar_jiyoonlee_210319.pdf

Graph structure

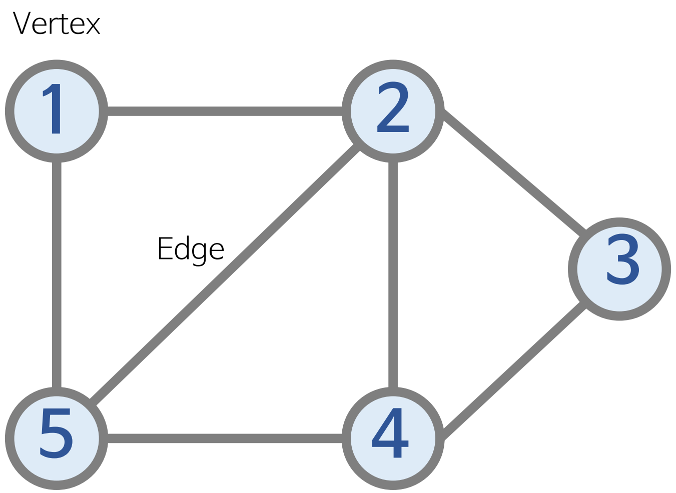
- Graph G are defined by:
  - Vertices V (node)
  - Edges E:
    - Directed/ undirected
    - Weighted/ unweighted
  - Adjacency matrix A: $n \times n$으로 표현, {1,0}



=



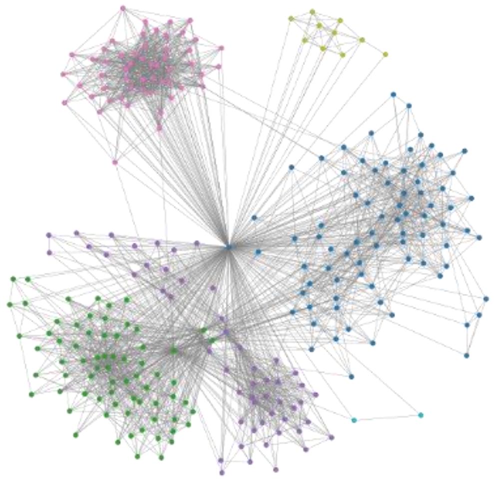
Directed

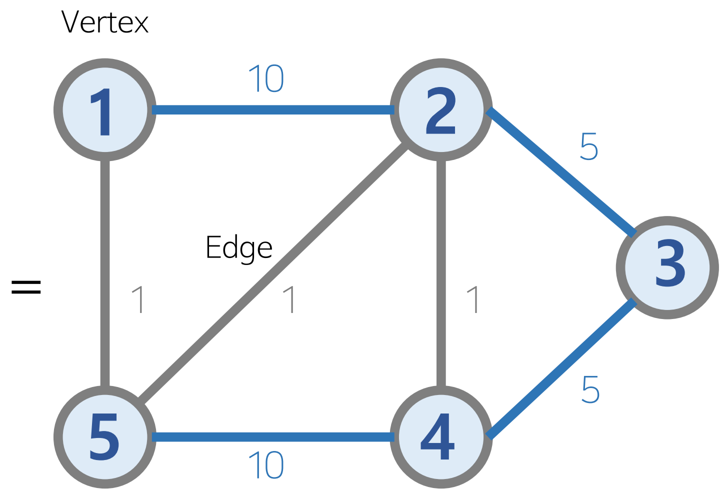


Undirected

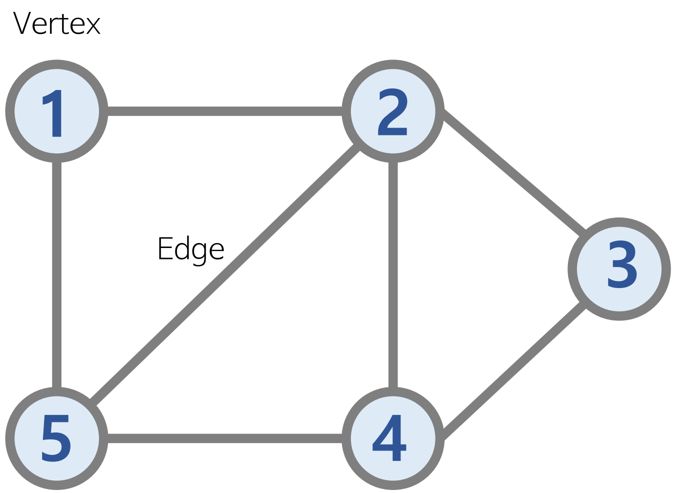
Graph structure

- Graph G are defined by:
  - Vertices V (node)
  - Edges E:
    - Directed/ undirected
    - Weighted/ unweighted
  - Adjacency matrix A: $n \times n$으로 표현, {1,0}





Weighted

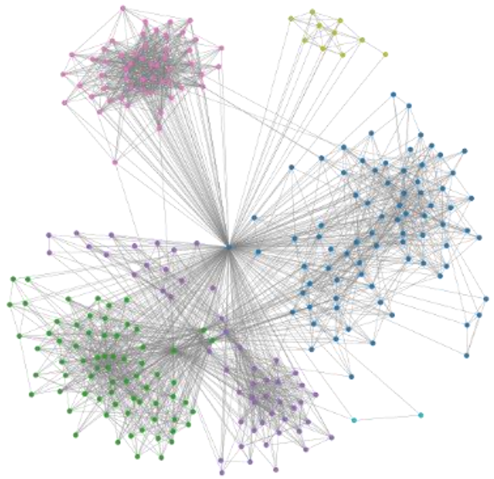


Unweighted

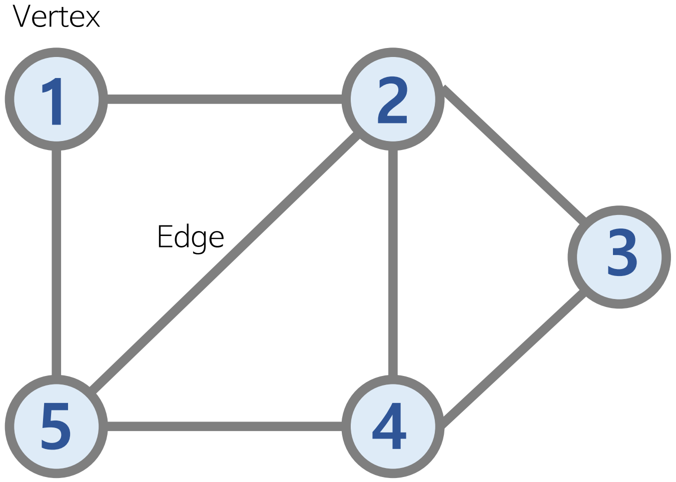
http://dmqm.korea.ac.kr/uploads/seminar/Openseminar_jiyoonlee_210319.pdf

Graph structure

- Graph G are defined by:
  - Vertices V (node)
  - Edges E:
    - Directed/ undirected
    - Weighted/ unweighted
  - Adjacency matrix A: $n \times n$으로 표현, {1,0}



=



$G = (V, E)$

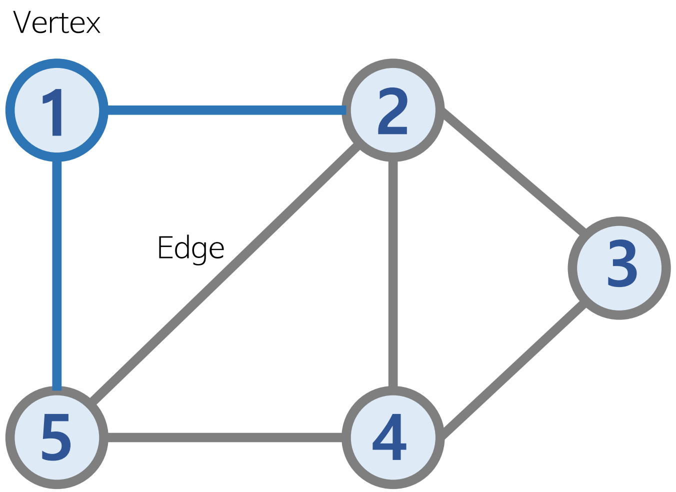
$A \in R^{n \times n}$

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

Adjacency Matrix

Graph structure

- Degree matrix
  - Degree는 각 node에 연결된 edge의 수
    - ➢ 따라서, Adjacency matrix의 행의 합과 동일



$A \in R^{n \times n}$

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

Adjacency Matrix

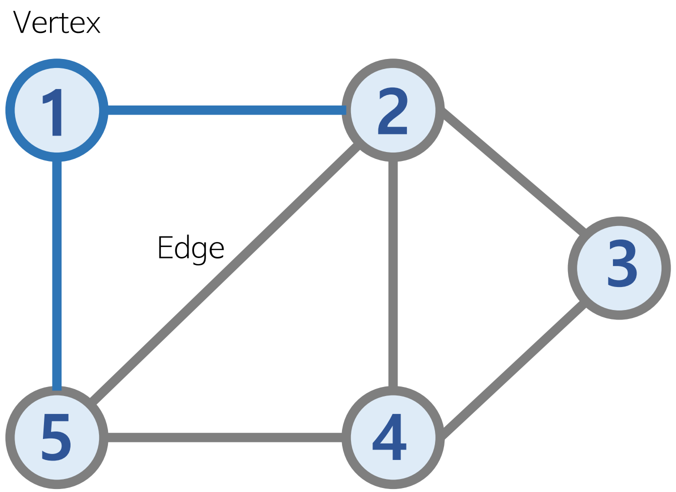
$\sum_j A_{i,j}$

2
4
2
3
3

Degree

Graph structure

- **Degree matrix**
  - Degree는 각 node에 연결된 edge의 수
    - ➢ 따라서, Adjacency matrix의 row sum 값과 동일
  - 대각 행렬에 degree 값으로 구성되며, 나머지 값은 0



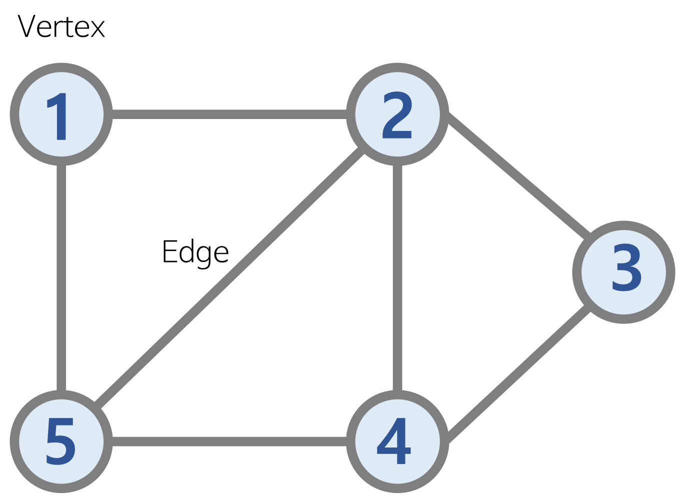
$$D_{i,i} = \sum_{j} A_{i,j} \qquad D \in R^{n \times n}$$

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

Degree Matrix

Graph structure

- **Laplacian matrix**
  - Degree matrix - Adjacency matrix



$D_{i,i} = \sum_j A_{i,j}$ $\quad D \in R^{n \times n}$

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

Degree Matrix

−

$A \in R^{n \times n}$

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

Adjacency Matrix

=

$L = D - A$ $\quad L \in R^{n \times n}$

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

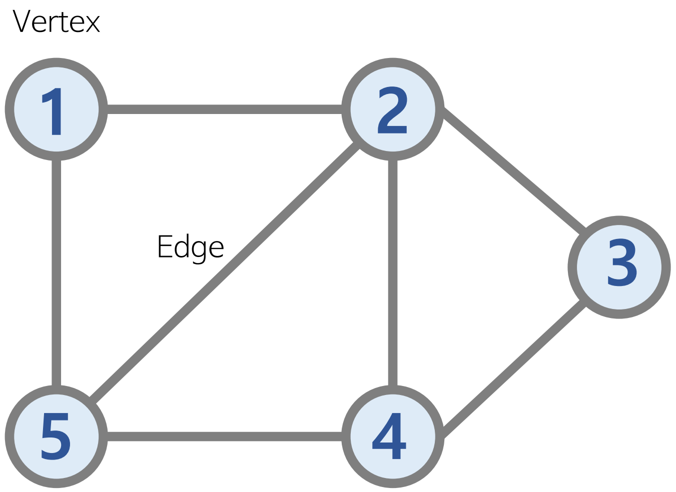
Laplacian Matrix

Degree정보

이웃vertex와의관계정보

Graph structure

- Laplacian matrix
  - Degree matrix - Adjacency matrix
  - 중심 node와 이웃 node사이의 관계 정보



$$L = D - A \qquad L \in R^{n \times n}$$

$[x_1, x_2, x_3, x_4, x_5] \times$

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

Laplacian Matrix

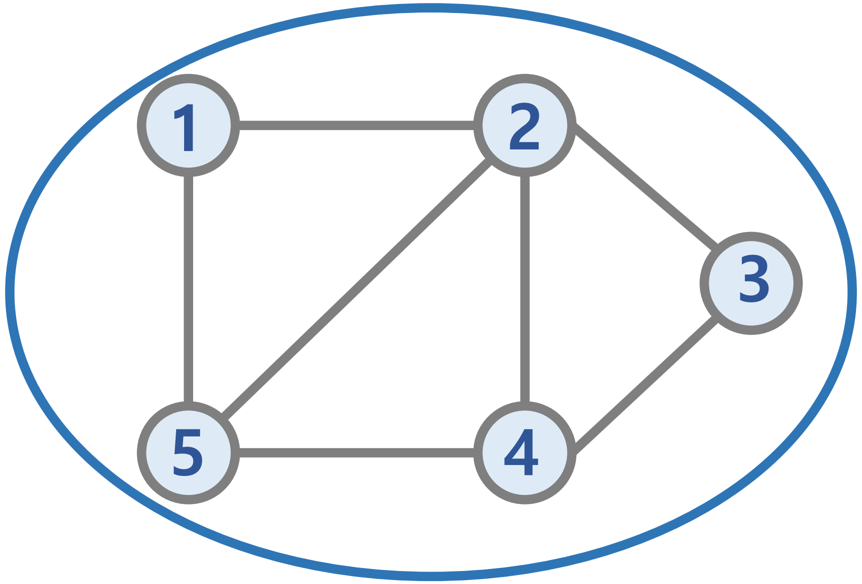
$$= -x_1 + 4x_2 - x_3 - x_4 - x_5$$

$$= (x_2 - x_1) + (x_2 - x_3) + (x_2 - x_3) + (x_2 - x_4) + (x_2 - x_5)$$

중심 node와 이웃 node 사이의 관계 정보 파악

Graph domain

- Task
  - Graph prediction
    - Molecule classification
  - Edge prediction
  - Node prediction



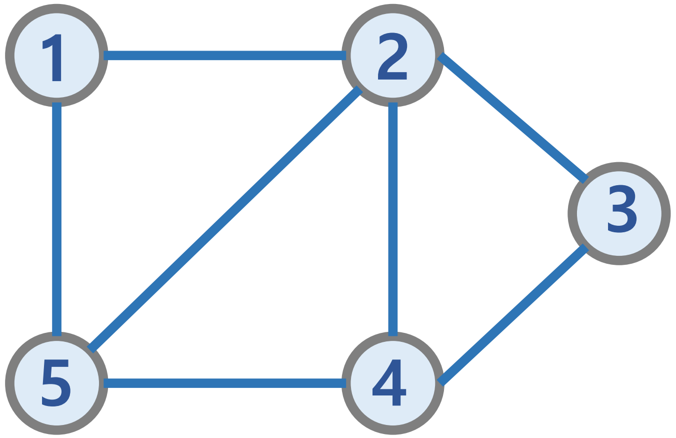
Graph prediction


Manchester United

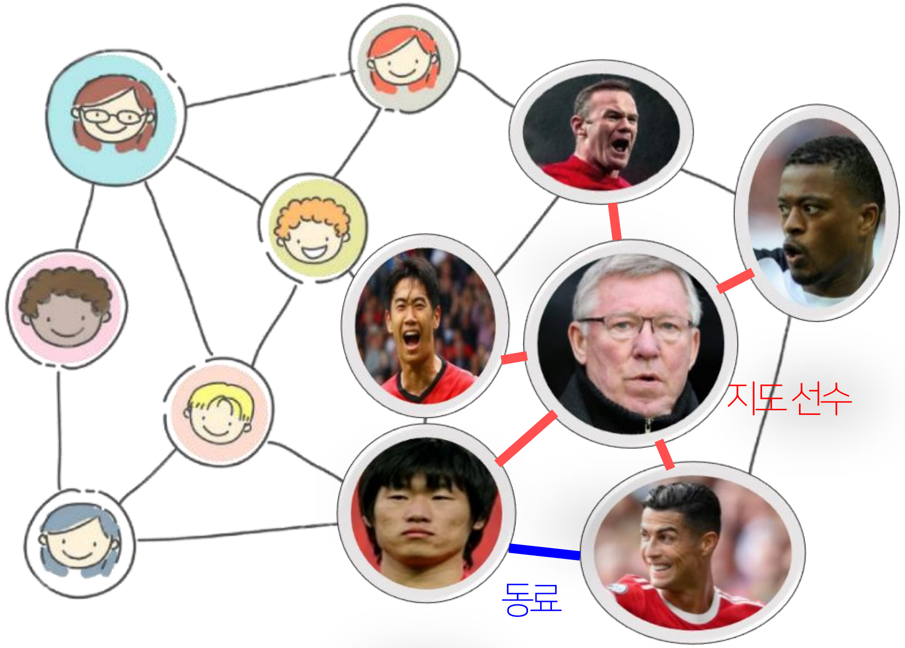
http://dmqm.korea.ac.kr/uploads/seminar/Openseminar_jiyoonlee_210319.pdf

Graph domain

- Task
  - Graph prediction
  - Edge prediction (=link prediction)
    - Recommendation system
  - Node prediction



Edge prediction

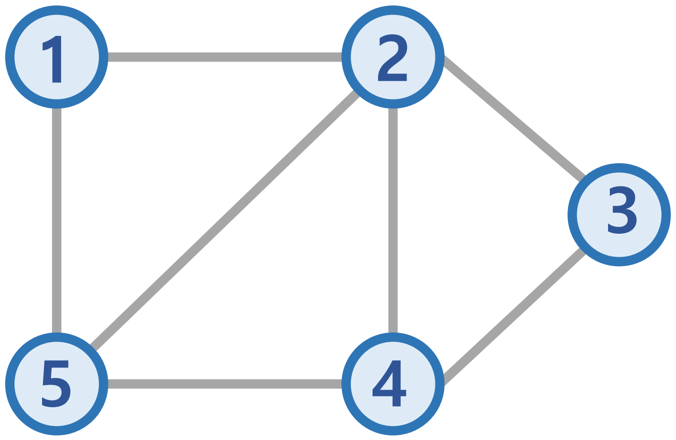


Relationship

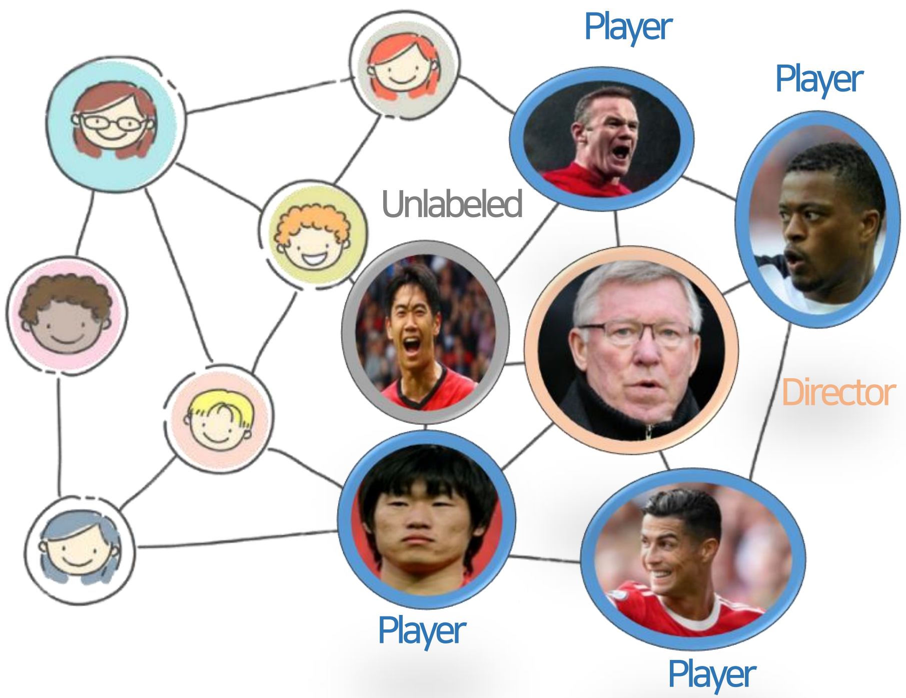
http://dmqm.korea.ac.kr/uploads/seminar/Openseminar_jiyoonlee_210319.pdf

Graph domain

- Task
  - Graph prediction
  - Edge prediction
  - **Node prediction**
    - ➢ A typical application of GNN is node classification



Node prediction



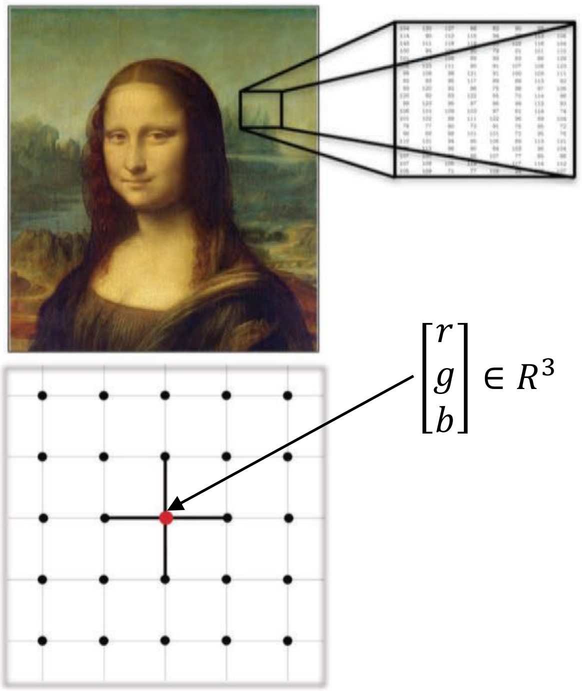
Current status

http://dmqm.korea.ac.kr/uploads/seminar/Openseminar_jiyoonlee_210319.pdf

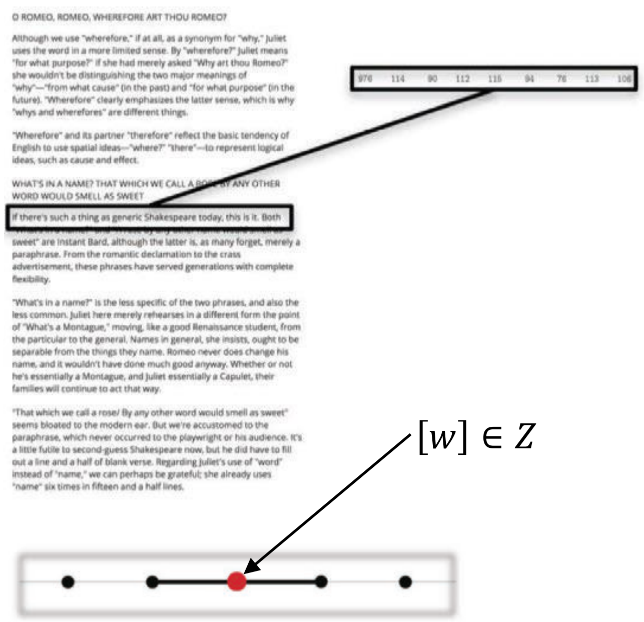
Graph data

Graph data

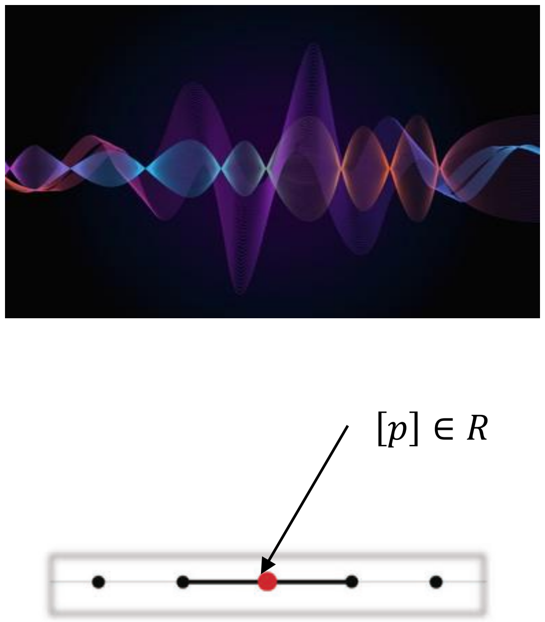
- **Data structure**
  - Image, volume, video 데이터는 2D, 3D Euclidean domains(grids)으로 표현
  - Sentence, word, speech 데이터는 1D Euclidean domains(grids)으로 표현



2D grid



1D grid



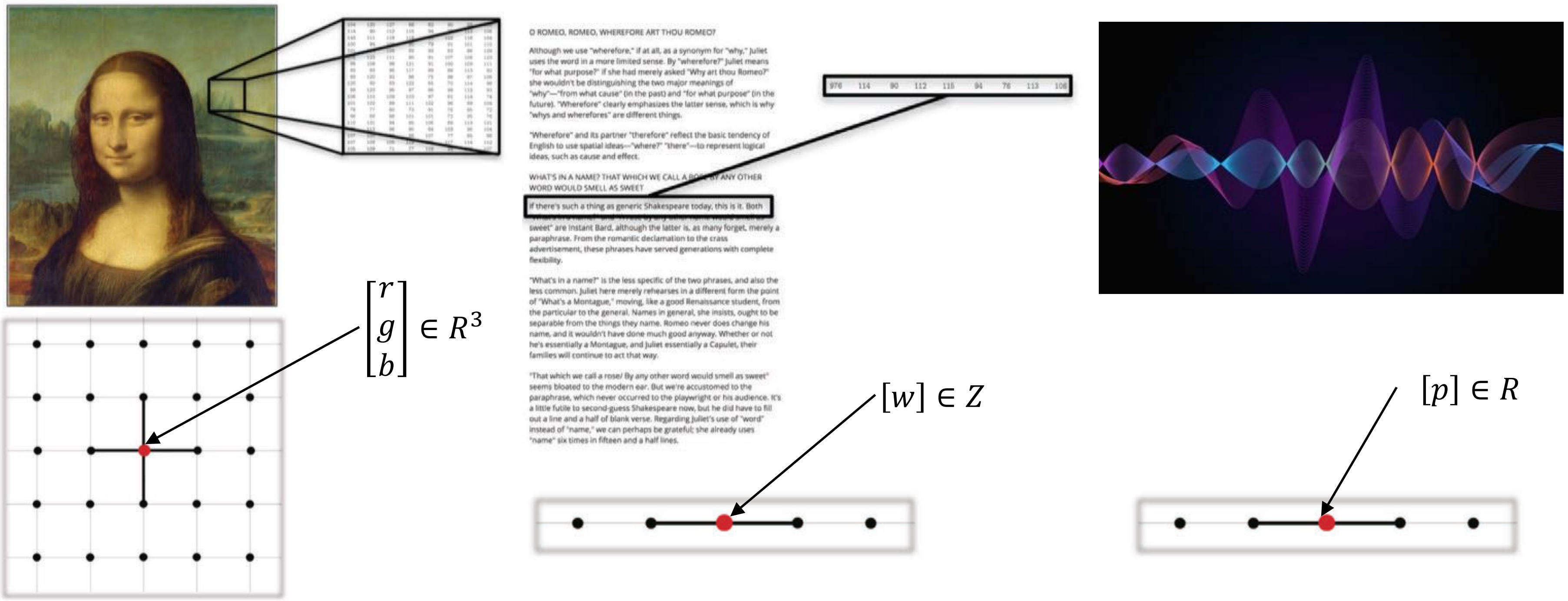
1D grid

https://atcold.github.io/pytorch-Deep-Learning/en/week13/13-1/

Graph data

- **Data structure**
  - Image, volume, video 데이터는 2D, 3D Euclidean domains(grids)으로 표현
  - Sentence, word, speech 데이터는 1D Euclidean domains(grids)으로 표현

규칙적인 공간 구조로 표현 가능하기 때문에 ConvNets 연산 가능(convolution, pooling)



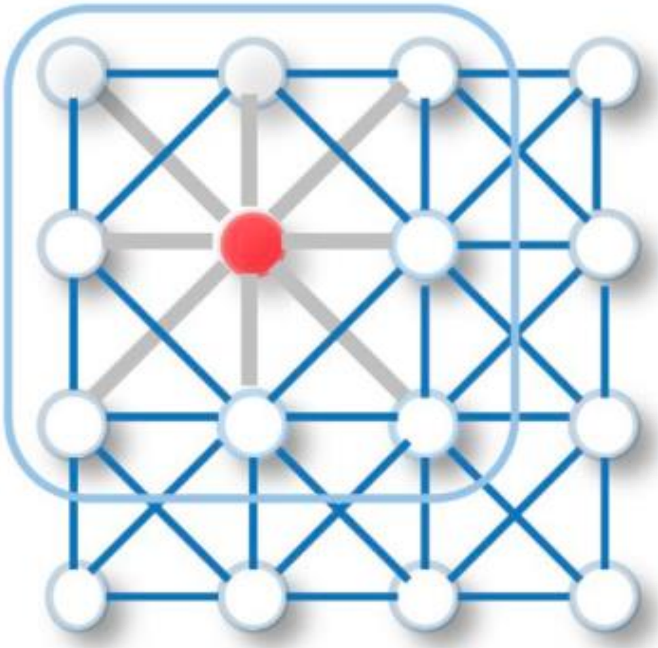
2D grid   1D grid   1D grid

https://atcold.github.io/pytorch-Deep-Learning/en/week13/13-1/

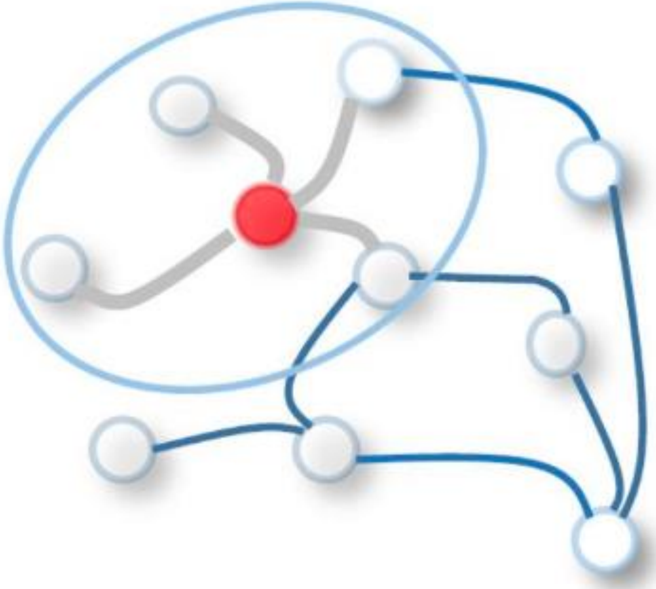
Graph data

- Convolution

CNN ≈ GNN?



2D-Convolution

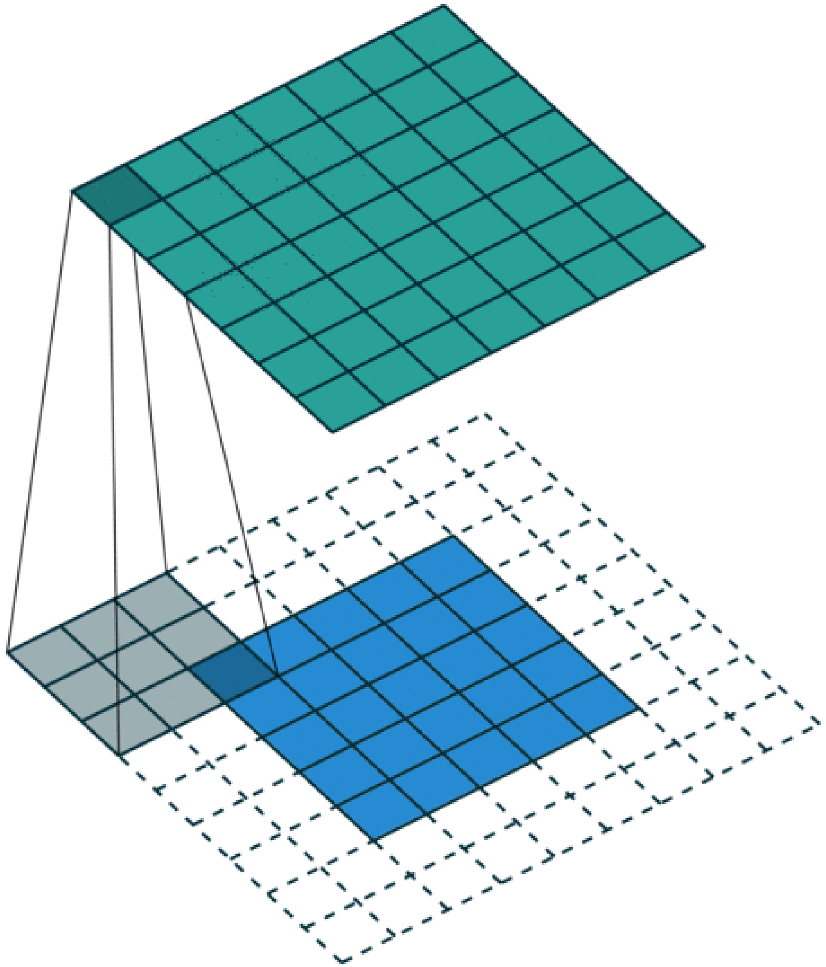


Graph Convolution

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems, 32(1), 4-24.

Convolution

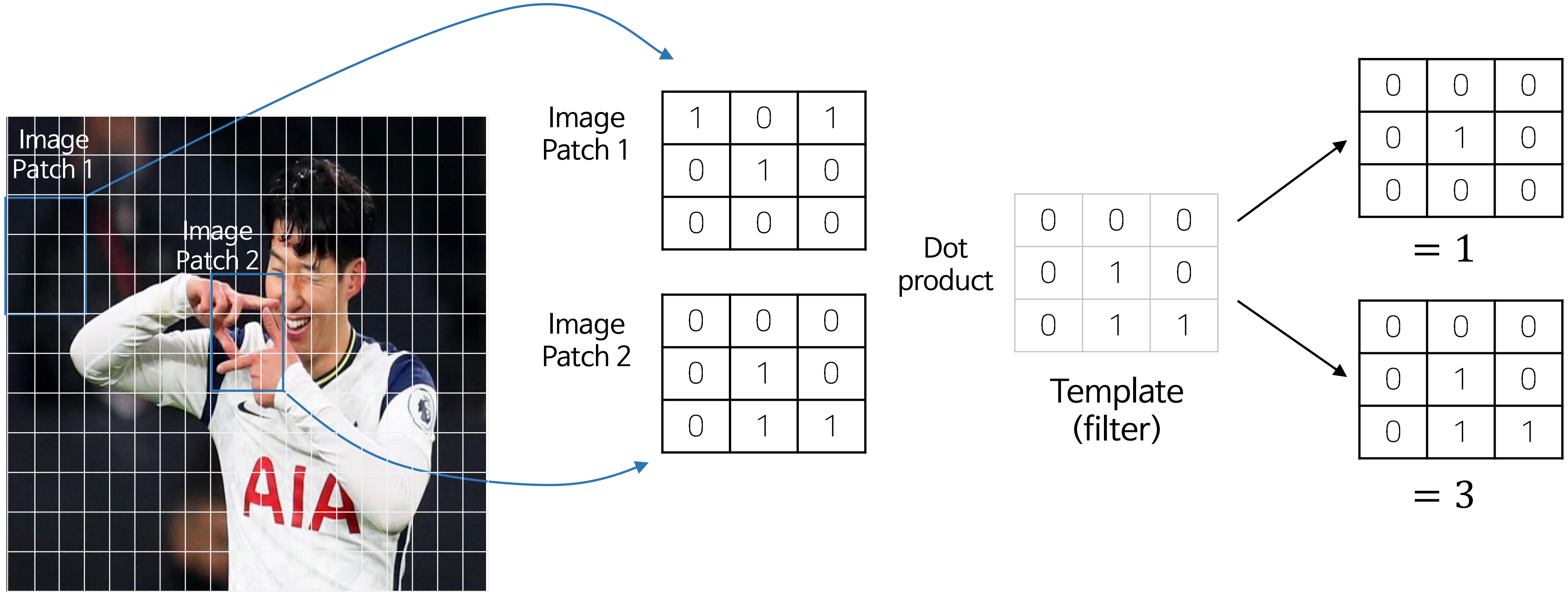
- Template matching



https://arxiv.org/pdf/1603.07285v1.pdf

Convolution

- Template matching
  - Convolution (=correlation) as comparing a template (the filter) with each local image patch
  - Correlation measures similarity between the filter and each local image region

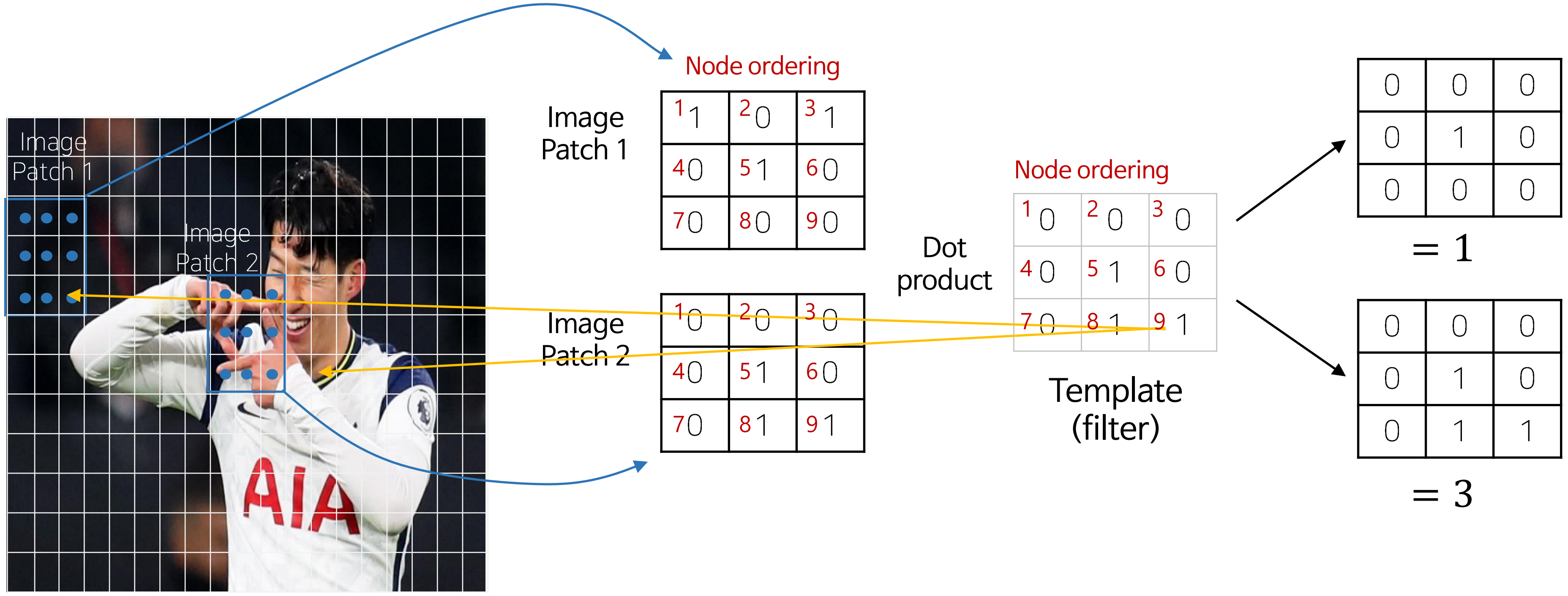


2D grid (h)

Template matching

Convolution

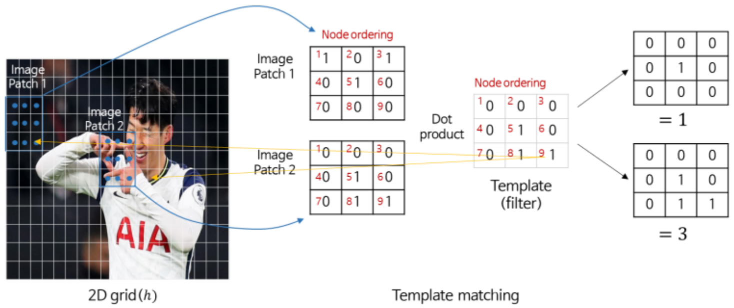
- Template matching
  - Same node ordering: template 내 노드 위치는 변하지 않기 때문에 template 내 노드의 동일한 정보를 image patch에 매칭



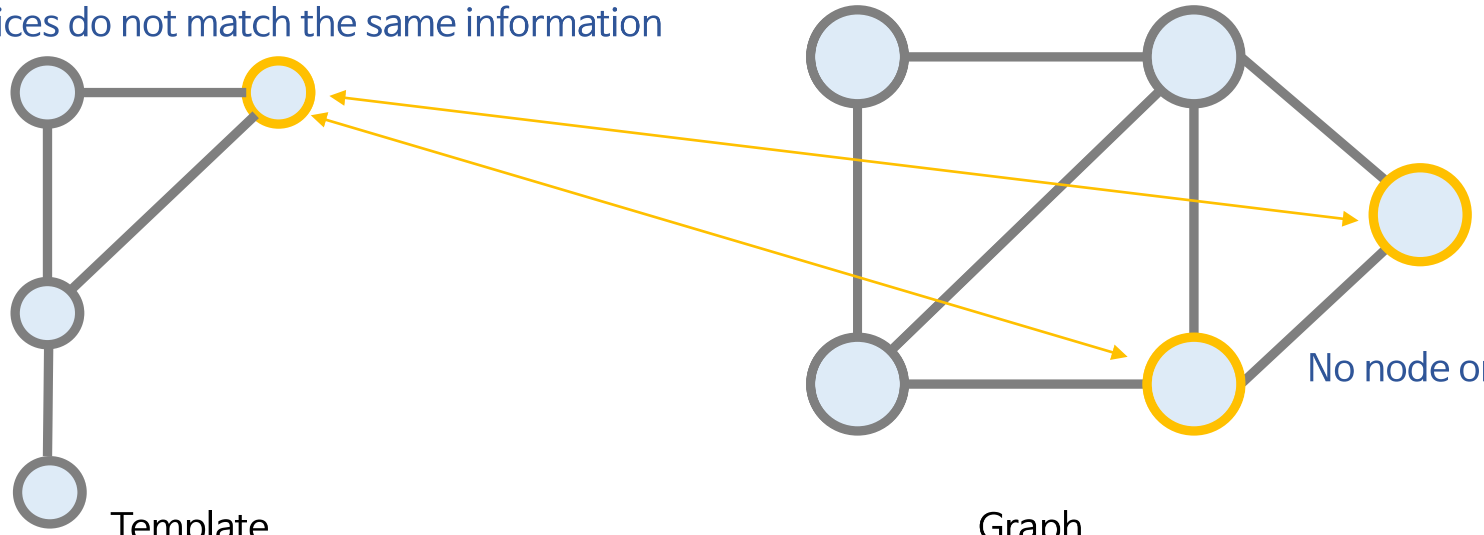
2D grid (h)

Template matching

Graph convolution

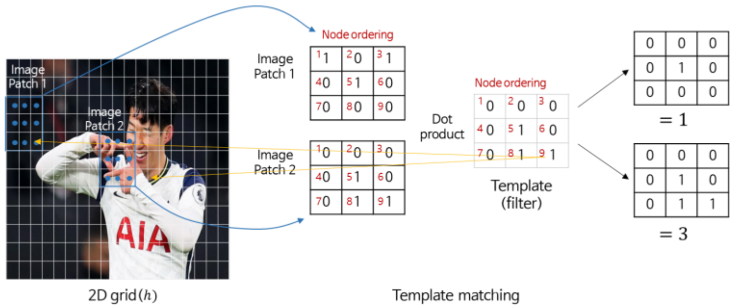


- Template matching for graphs?
  - No node ordering: node에 position(index)가 없다면, graph data에 어떻게 template matching?

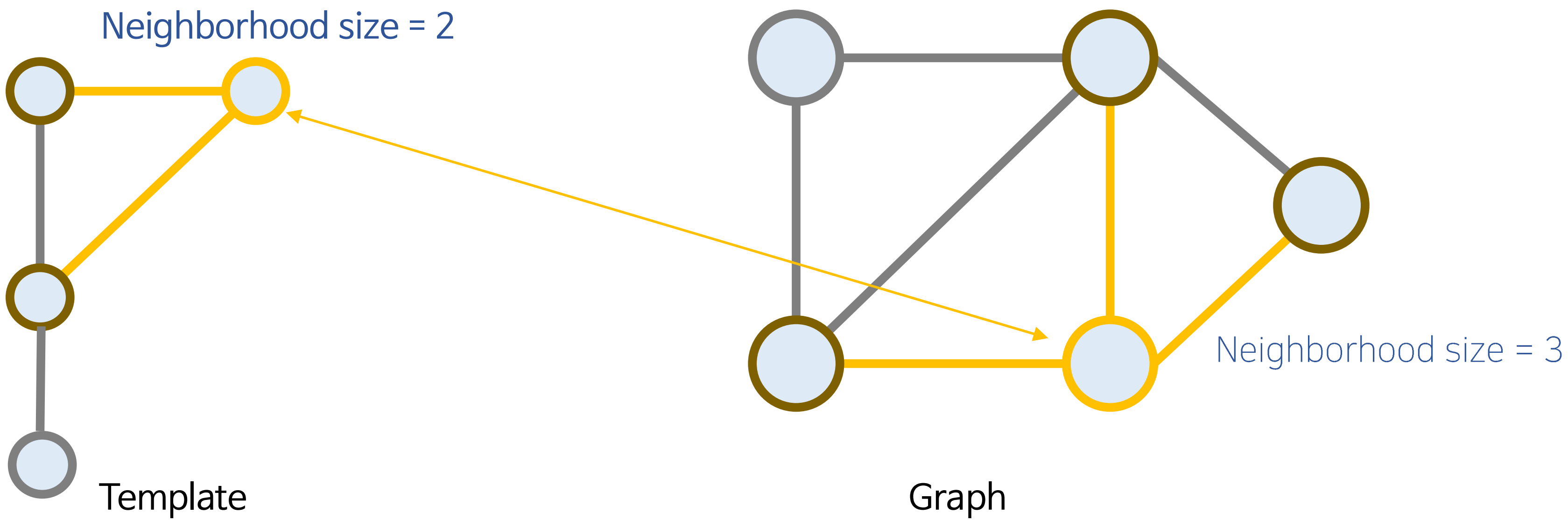


Permutation invariance: independent of the order of neighbor nodes, as there is no specific way to order them.

Graph convolution



- Template matching for graphs?
  - No node ordering: node에 position (index)가 없을 때, graph data에 어떻게 template matching?
  - Heterogeneous neighborhood: node별 이웃의 수가 다를 때, 어떻게 template matching?



How to define graph convolution?

How to define graph convolution?

- Convolution theorem
  - Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms

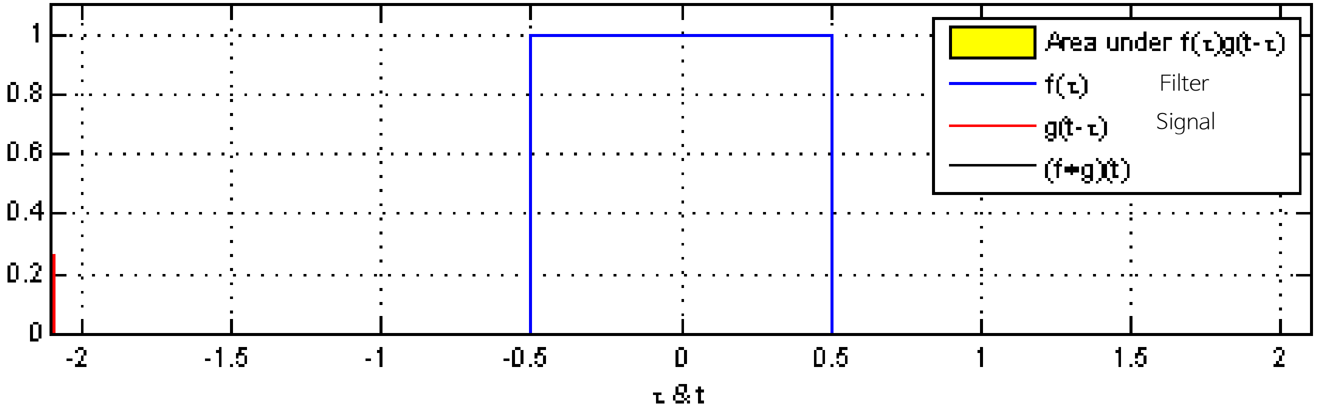
$$\mathcal{F}(w * h) = \mathcal{F}(w) \odot \mathcal{F}(h)$$

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

"Convolution in spatial (time) domain is equivalent to multiplication in Fourier domain"

How to define graph convolution?

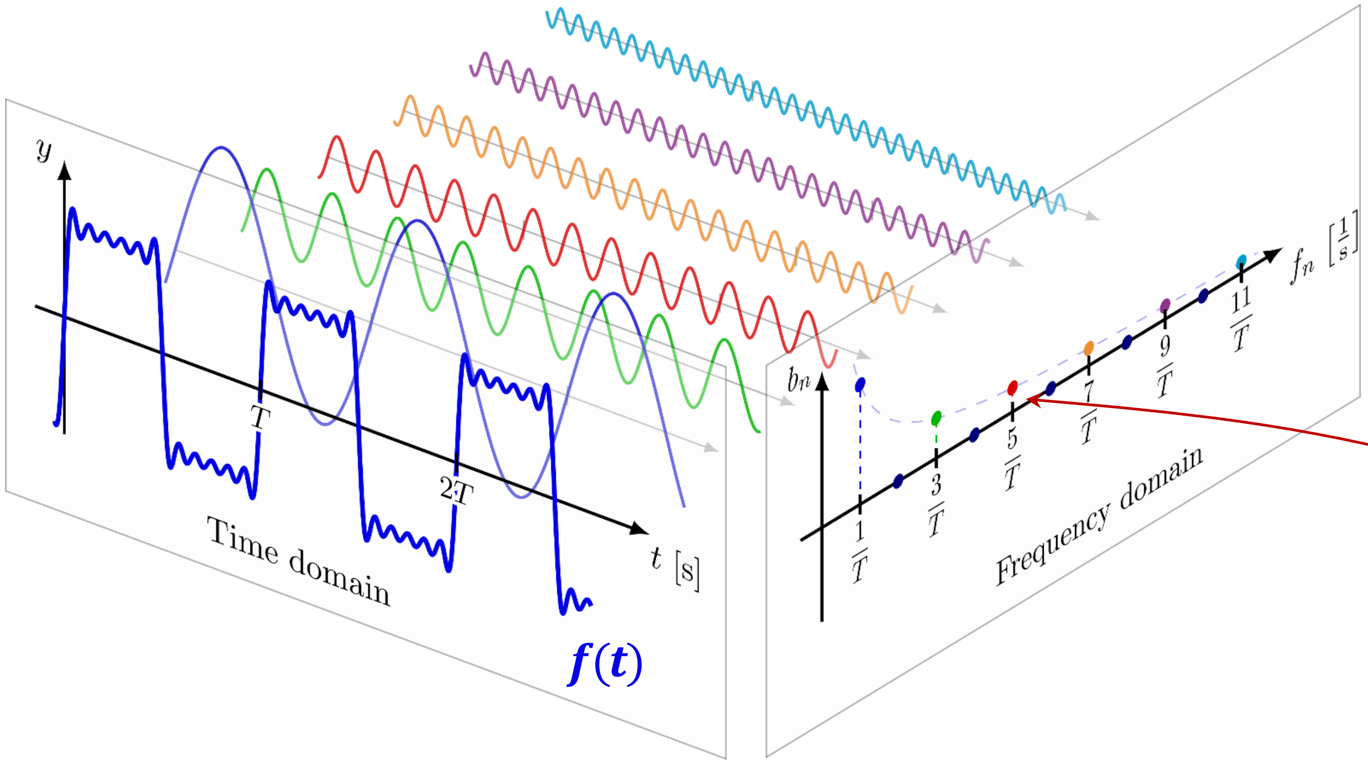
- **Convolution theorem**
  - Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms



https://commons.wikimedia.org/wiki/File:Convolution_of_box_signal_with_itself2.gif

How to define graph convolution?

- Fourier transform
  - 임의의 입력 신호(signal)를 다양한 주파수(frequency) 를 갖는 주기함수들의 합으로 분해하여 표현
  - 푸리에 변환에 사용하는 주기함수는 sin, cos 삼각함수
  - 푸리에 변환은 고주파부터 저주파까지 다양한 주파수까지 다양한 주파수 대역의 sin, cos 삼각 함수들로 원본 신호를 분해하는 것



Fourier transform

$$\hat{f}(\xi) = \int_{\mathbb{R}} f(t)\, e^{-2\pi i \xi t} dt$$

Inverse Fourier transform

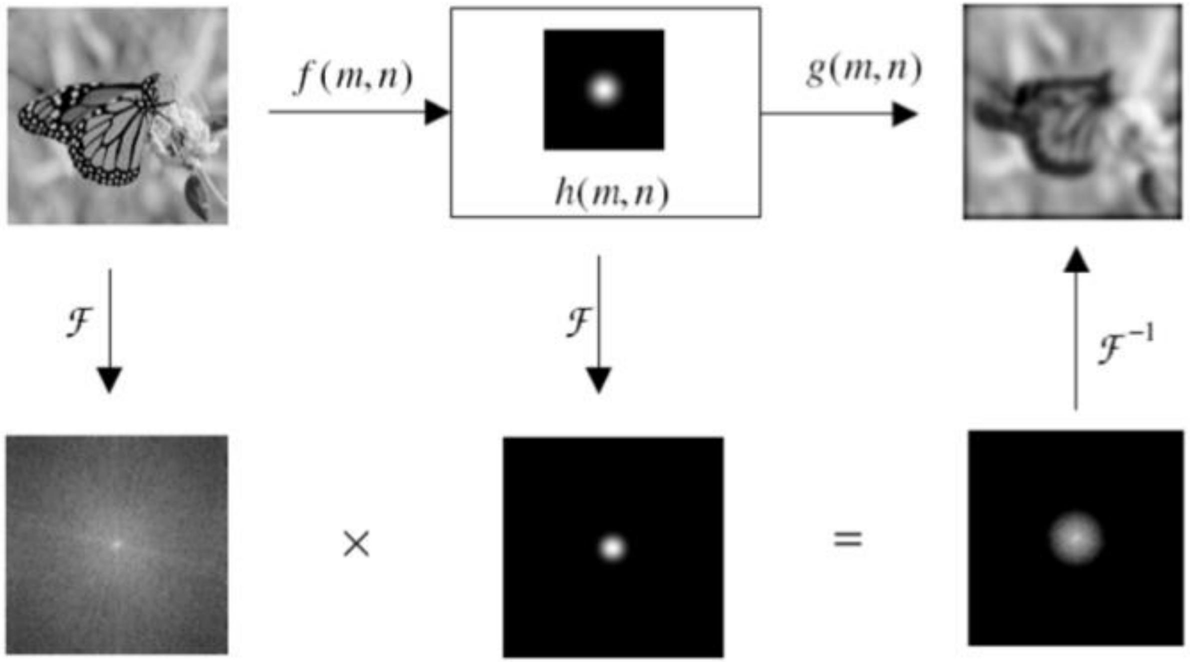
Frequency

$$f(x) = \int_{\mathbb{R}} \hat{f}(\xi)\, \boxed{e^{2\pi i \xi t}} d\xi$$

https://darkpgmr.tistory.com/171

How to define graph convolution?

- **Fourier transform**
  - 입력 신호: 이미지에서는 공간축(spatial)에 대해 정의된 신호이며, 전파, 음성 신호에서는 시간축(time)에 대해 정의
  - 영상처리 분야에서는 spatial domain에서 frequency domain으로의 변환, 통신 분야에서는 time domain에서 frequency domain으로 변환



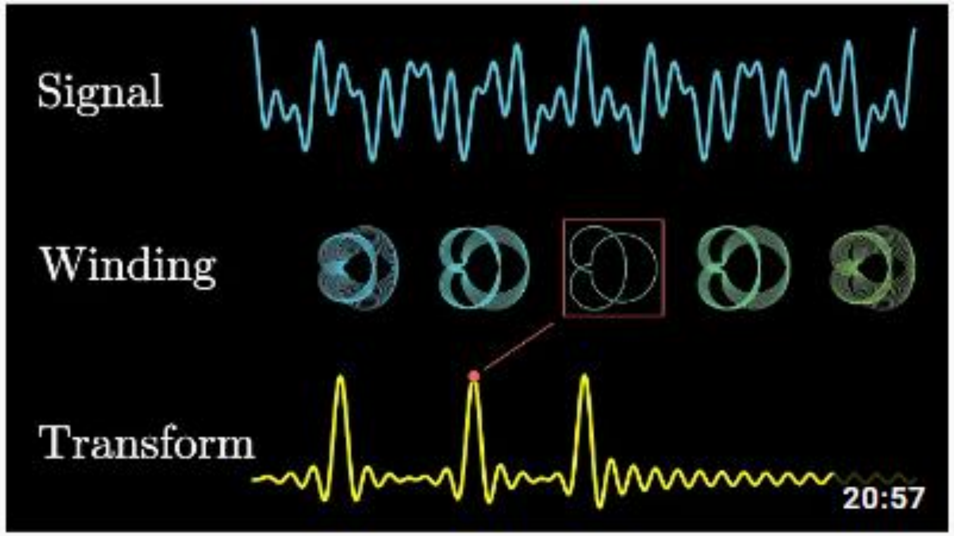
$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

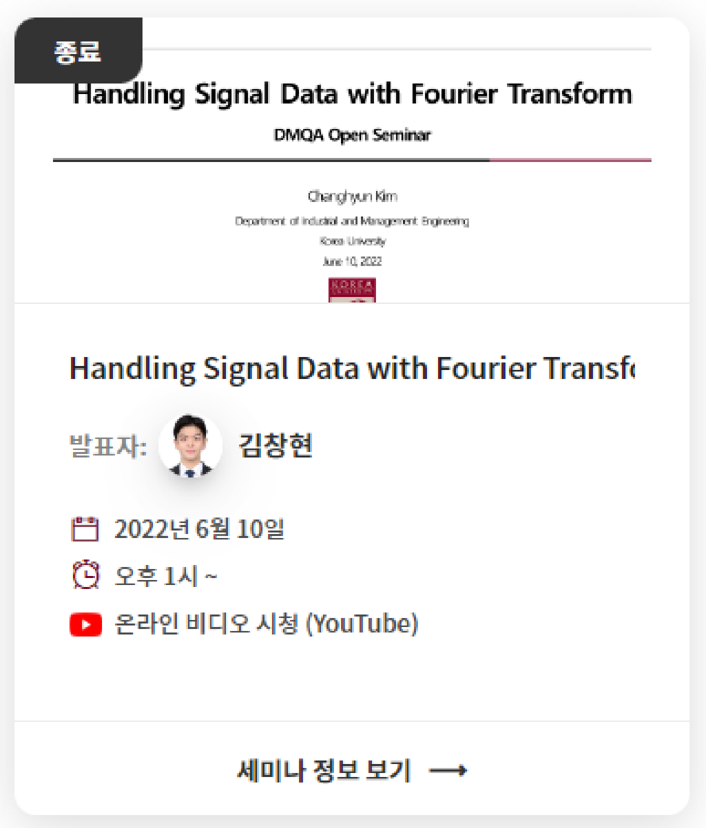
https://sbme-tutorials.github.io/2018/cv/notes/3_week3.html

How to define graph convolution?

- Fourier transform
  - 푸리에 변환에 대한 직관적인 이해 및 수식 및 다양한 종류의 푸리에 변환에 대해서 아래 영상 참고

But what is the Fourier Transform? A visual introduction.



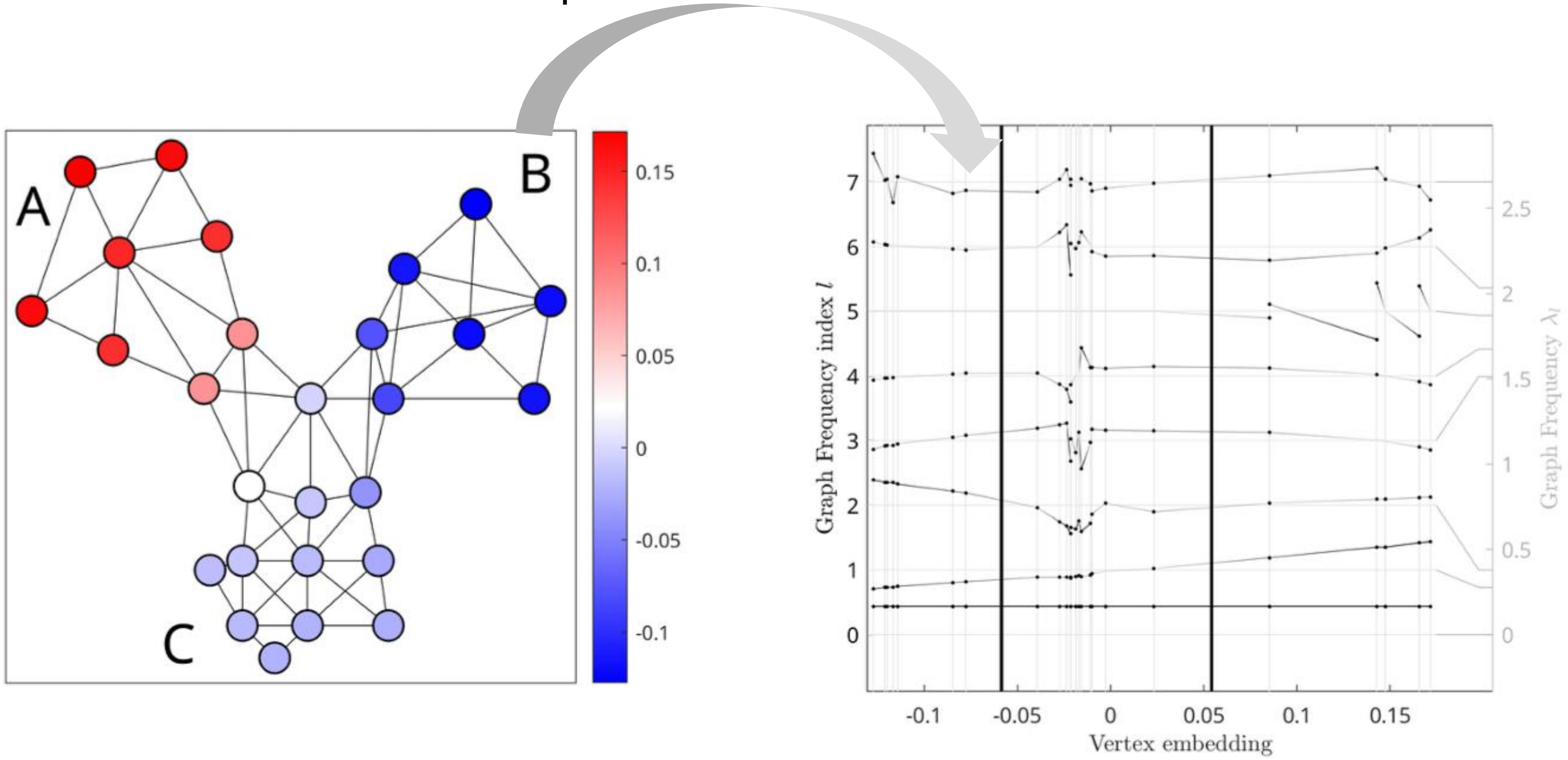


https://www.youtube.com/watch?v=spUNpyF58BY
https://www.youtube.com/watch?v=Q5L_Ky6nKVo

How to define graph convolution?

- Graph Fourier transform

Graph Fourier Transform



Spatial domain

Frequency domain (spectral domain)

Girault, B., & Ortega, A. (2019). What's in a frequency: new tools for graph Fourier Transform visualization. *arXiv preprint arXiv:1903.08827*.

How to define Fourier transforms for graphs?

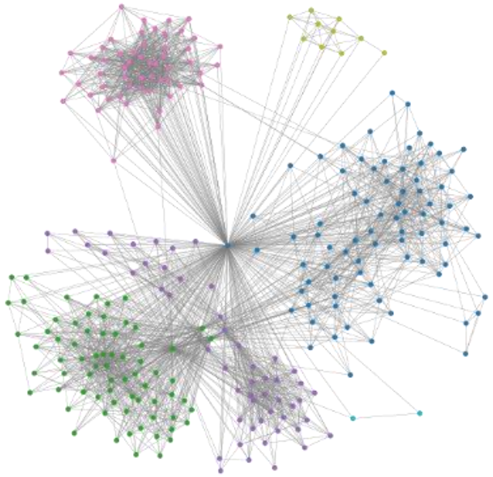
두가지 방향 소개, 나머지 한가지 방향은 appendix 참고하시길 바랍니다.

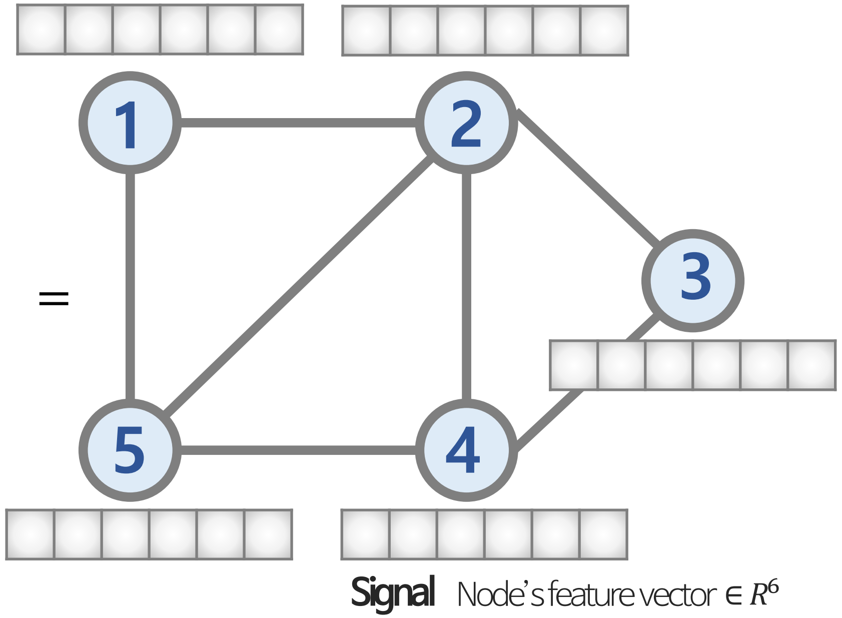
$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

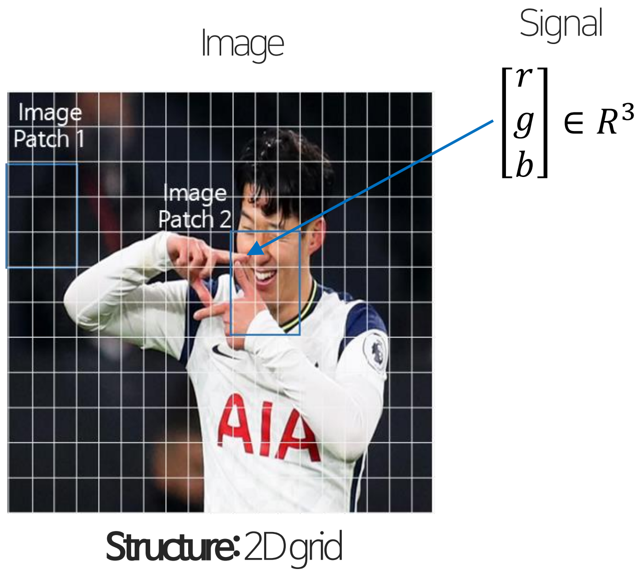
How to define Fourier transforms for graphs?

- Fourier transform
  - 임의의 입력 신호(signal)를 다양한 주파수(frequency) 를 갖는 주기함수들의 합으로 분해하여 표현

입력 신호(signal): Node features





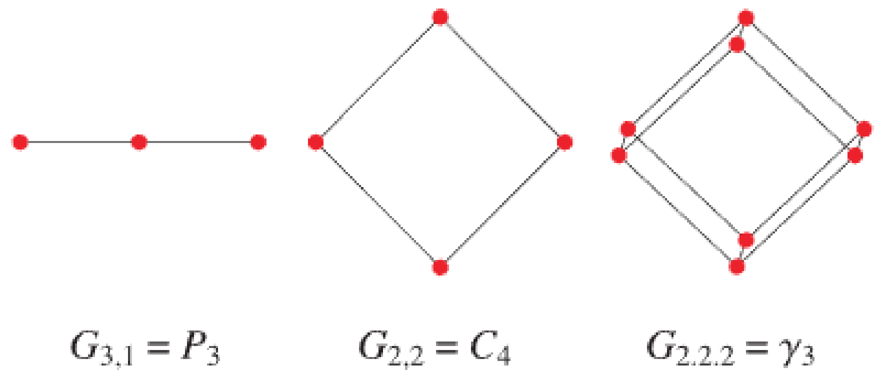


Undirected graph를 가정

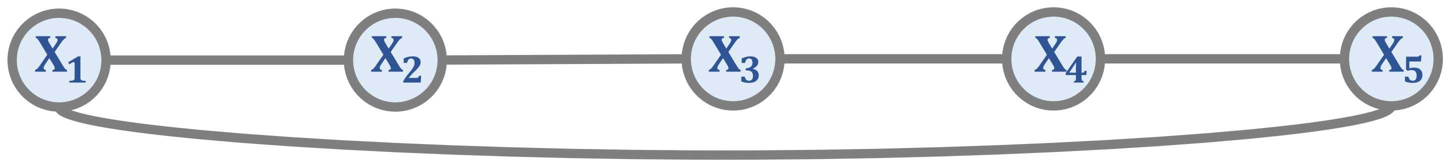
$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

How to define Fourier transforms for graphs?

- Rethinking the convolution on sequences
  - Image *cyclical* ***grid*** *graph*, and ***Convolution*** over it



A generalized grid graph, also known as an –dimensional lattice graph (e.g., Acharya and Gill 1981)



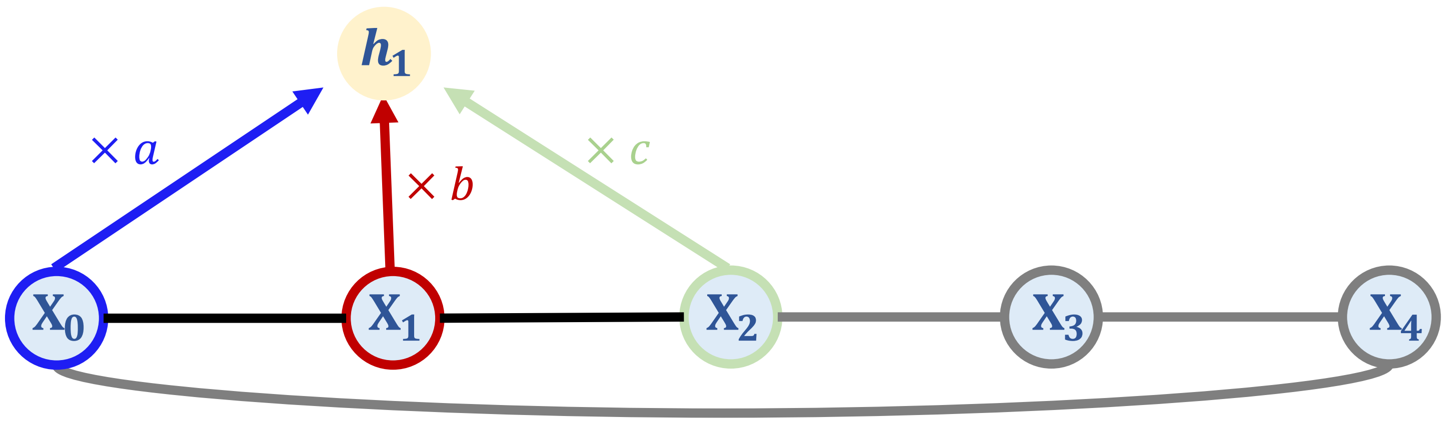
$*$ *for easier handling of boundary conditions*

https://mathworld.wolfram.com/GridGraph.html
https://petar-v.com/talks/GNN-Wednesday.pdf

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

How to define Fourier transforms for graphs?

- Rethinking the convolution on sequences
  - Image $cyclical$ $\boldsymbol{grid}$ $graph$, and $\boldsymbol{Convolution}$ over it

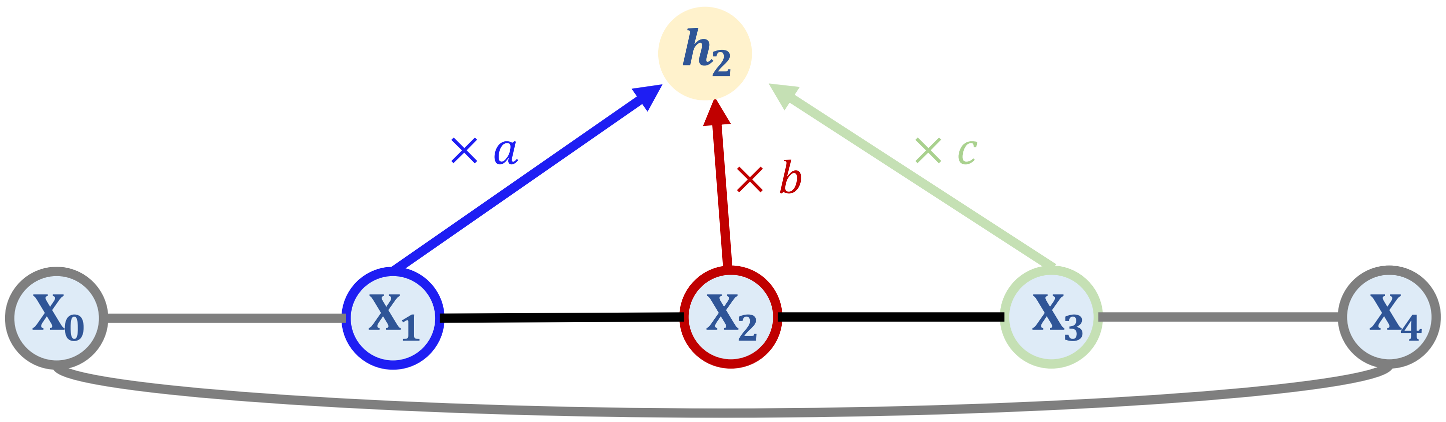


$*$ *for easier handling of boundary conditions*

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

How to define Fourier transforms for graphs?

- Rethinking the convolution on sequences
  - Image $cyclical\ \boldsymbol{grid}\ graph$, and $\boldsymbol{Convolution}$ over it



$*\ for\ easier\ handling\ of\ boundary\ conditions$

https://petar-v.com/talks/GNN-Wednesday.pdf

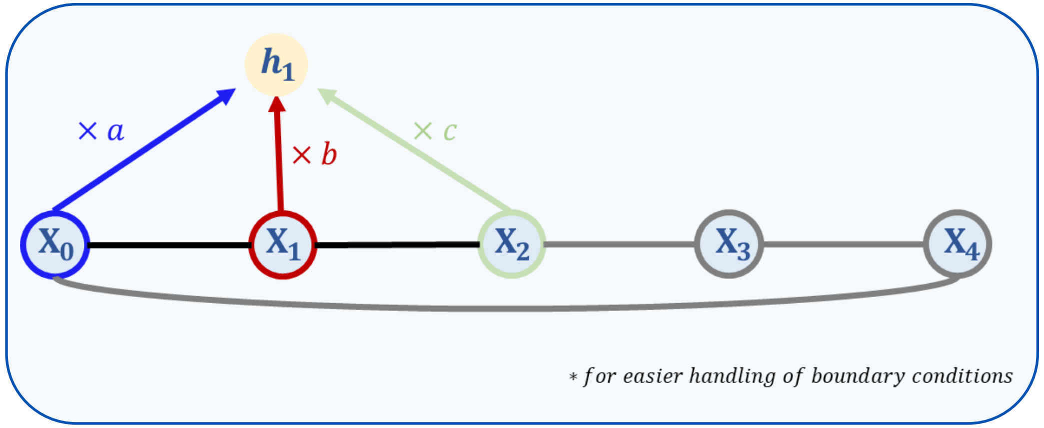
$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

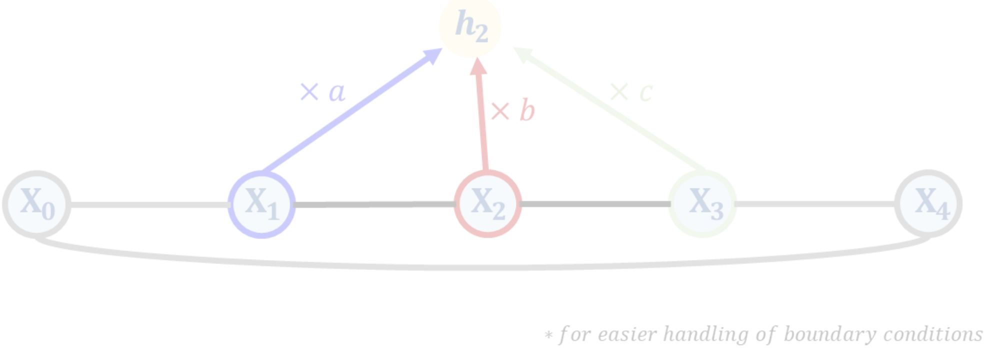
How to define Fourier transforms for graphs?

- Rethinking the convolution on sequences
  - Image $cyclical\ \boldsymbol{grid}\ graph$, and $\boldsymbol{Convolution}$ over it

Filter　　　　Signal

$$\begin{bmatrix} b & c & 0 & 0 & a \\ a & b & c & 0 & 0 \\ 0 & a & b & c & 0 \\ 0 & 0 & a & b & c \\ c & 0 & 0 & a & b \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

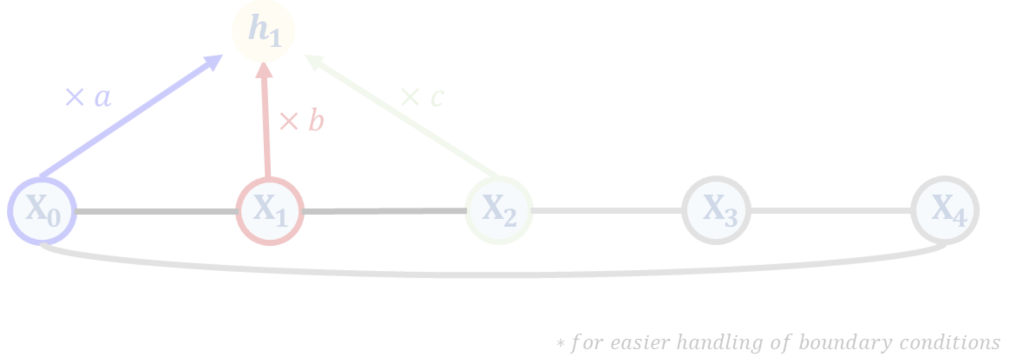




$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

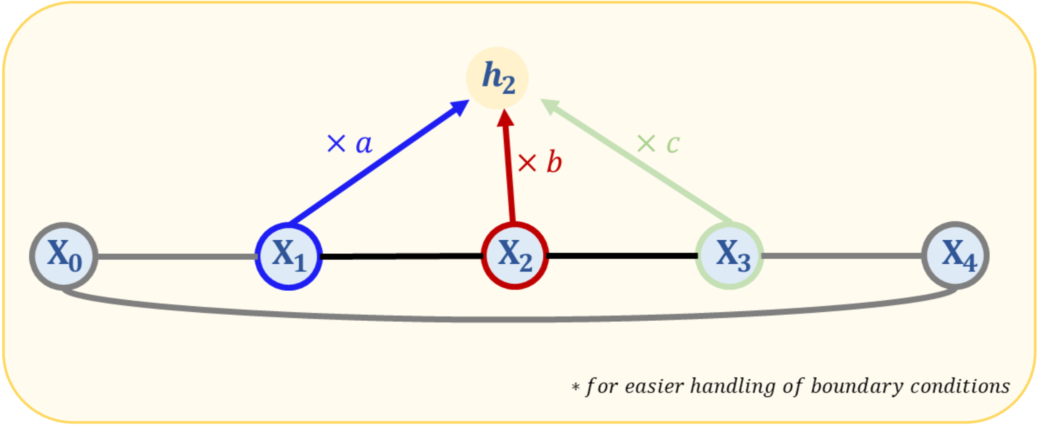
How to define Fourier transforms for graphs?

- Rethinking the convolution on sequences
  - Image *cyclical **grid** graph*, and ***Convolution*** over it



Filter | Signal

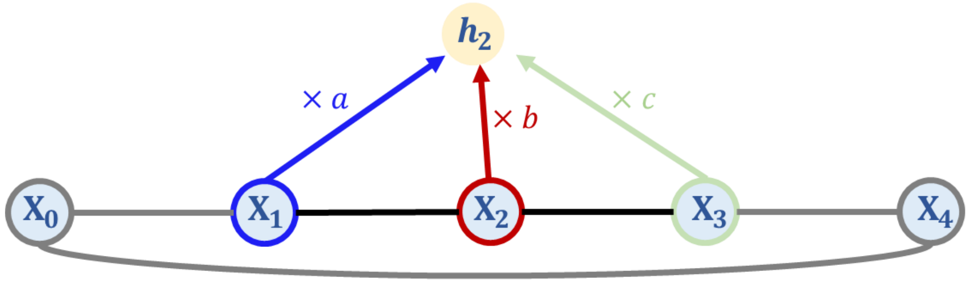
$$\begin{bmatrix} b & c & 0 & 0 & a \\ a & b & c & 0 & 0 \\ 0 & a & b & c & 0 \\ 0 & 0 & a & b & c \\ c & 0 & 0 & a & b \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$



$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

How to define Fourier transforms for graphs?

- Rethinking the convolution on sequences
  - Image *cyclical* ***grid*** *graph*, and ***Convolution*** over it
  - Define a *circulant* matrix $\boldsymbol{C}([\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{0}, \boldsymbol{0}, \cdots, \boldsymbol{0}, \boldsymbol{a}]), \boldsymbol{H} = \boldsymbol{f}(\boldsymbol{X}) = \boldsymbol{C}\boldsymbol{X}$



∗ *for easier handling of boundary conditions*

Circulant matrix(회전행렬):
$\boldsymbol{C}([\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{0}, \boldsymbol{0}, \cdots, \boldsymbol{0}, \boldsymbol{a}])$

Signal

$$\boldsymbol{f}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{b} & \boldsymbol{c} & & & & \boldsymbol{a} \\ \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} & & & \\ & & \ddots & \ddots & \ddots & \\ & & & \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} \\ \boldsymbol{c} & & & & \boldsymbol{a} & \boldsymbol{b} \end{bmatrix} \begin{bmatrix} \mathbf{X}_0 \\ \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{n-2} \\ \mathbf{X}_{n-1} \end{bmatrix}$$

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

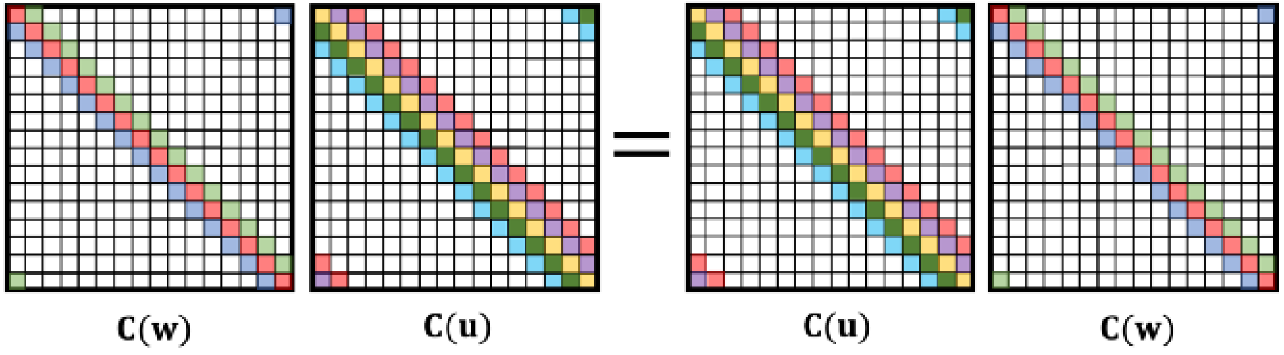
How to define Fourier transforms for graphs?

- Rethinking the convolution on sequences
  - Circulant matrices commute! (교환법칙 성립)

Circulant matrix: $C([b, c, 0, 0, \cdots, 0, a])$

$$\begin{bmatrix} b & c & & & a \\ a & b & c & & \\ & \ddots & \ddots & \ddots & \\ & & a & b & c \\ c & & & a & b \end{bmatrix}$$

(for any parameter vectors u, w)



“Convolution 연산은 commutative 연산이다 $x * w = w * x$”

https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

How to define Fourier transforms for graphs?

- Rethinking the convolution on sequences
  - Circulant matrices commute! (교환법칙 성립)
  - Commuting matrices are jointly diagonalizable(대각화 가능)

$$AX = \lambda X$$

Eigenvectors (고유벡터) — X

Eigenvalues (고유값) — λ

(eigenvalues는 다를 수 있다)

"AB = BA 일 때, A와 B는 동일한 eigenvectors를 가진다"

"모든 circulant matrices는 교환 법칙을 만족하기 때문에 동일한 eigenvectors를 가진다"

https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

How to define Fourier transforms for graphs?

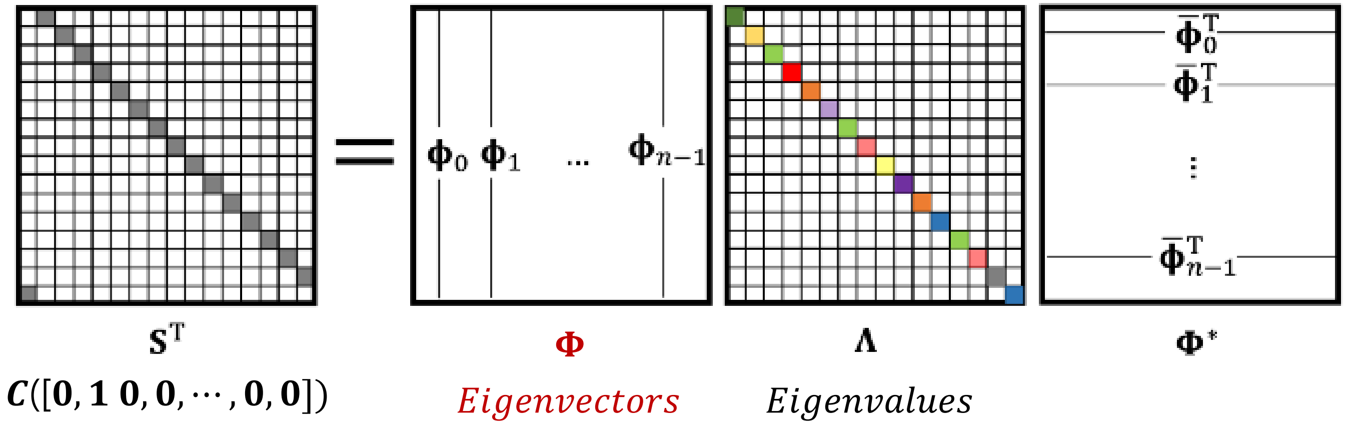
$$AX = \lambda X$$

Eigenvectors (고유벡터) — Eigenvalues (고유값)

(eigenvalues는 다를 수 있다)

- Rethinking the convolution on sequences
  - Circulant matrices commute! (교환법칙 성립)
  - Commuting matrices are jointly diagonalizable (대각화 가능)

> "모든 circulant matrices는 교환 법칙을 만족하기 때문에 **동일한 eigenvectors**를 가진다"



- ✓ 오른쪽으로 1만큼 이동 (translation)
- ✓ Circulant matrix & orthogonal matrix ($S \cdot S^T = I$)

https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

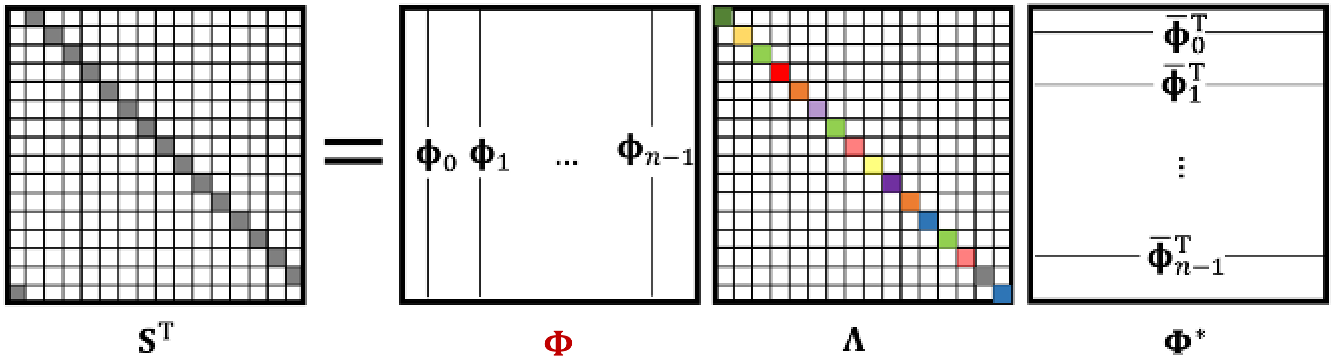
How to define Fourier transforms for graphs?

$$AX = \lambda X$$

Eigenvectors (고유벡터) — Eigenvalues (고유값)

- Rethinking the convolution on sequences
  - Circulant matrices commute! (교환법칙 성립)
  - Commuting matrices are jointly diagonalizable(대각화 가능)

"모든 circulant matrices의 eigenvectors는 discrete fourier basis와 동일하다"



$Shift\ matirx\colon\ C([0,1\ 0,0,\cdots,0,0])$ *Eigenvectors* *Eigenvalues*

- ✓ 오른쪽으로 1만큼 이동(translation)
- ✓ Circulant matrix & orthogonal matrix ($S \cdot S^T = I$)

$$\boldsymbol{\Phi} = [e^{(0)}\ e^{(1)}\ e^{(2)}\ \cdots e^{(n-1)}]$$

$$e^{(k)} = \begin{bmatrix} w_n^{0\cdot k} \\ w_n^{1\cdot k} \\ w_n^{2\cdot k} \\ \vdots \\ w_n^{(n-1)\cdot k} \end{bmatrix},\ w_n = e^{\left(\frac{2\pi i}{n}\right)}$$

"Discrete Fourier basis"

Shift= Translation

Fourier functions form an orthonormal basis

https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

$$\mathcal{F}(h) = \boldsymbol{\Phi}^* \boldsymbol{h}$$

What we have covered so far

- Discrete Fourier Transform (DFT) & Convolution theorem
  1. Convolution 연산은 circulant matrix로 표현 가능
  2. Circulant matrix는 교환 법칙 성립
  3. 모든 circulant matrices는 교환 법칙을 만족하기 때문에 동일한 eigenvectors를 가진다
  4. 모든 circulant matrices의 eigenvectors는 discrete fourier basis와 동일하다 ($C(\theta) = \boldsymbol{\Phi\Lambda\Phi}^*$) $\boldsymbol{\Phi}^* = \boldsymbol{\Phi}^{\mathrm{T}}$

$$\boldsymbol{f}(\boldsymbol{X}) = \underbrace{\begin{bmatrix} \boldsymbol{b} & \boldsymbol{c} & & & \boldsymbol{a} \\ \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} & & \\ & \ddots & \ddots & \ddots & \\ & & \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} \\ \boldsymbol{c} & & & \boldsymbol{a} & \boldsymbol{b} \end{bmatrix}}_{C(\theta)\;(w)} * \underbrace{\begin{bmatrix} \mathbf{X}_0 \\ \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{n-1} \\ \mathbf{X}_{n-2} \end{bmatrix}}_{\mathbf{X}\;(h)} = \underbrace{\boldsymbol{\Phi}}_{\mathcal{F}^{-1}} \underbrace{\begin{bmatrix} \hat{\theta}_0 & & & \\ & \hat{\theta}_1 & & \\ & & \ddots & \\ & & & \hat{\theta}_{n-1} \end{bmatrix}}_{\boldsymbol{\Lambda}\;(\mathcal{F}(w))} \odot \underbrace{\boldsymbol{\Phi}^* \mathbf{X}}_{\mathcal{F}(h)} = \boldsymbol{\Phi}(\hat{\theta} \circ \hat{\mathbf{X}})$$

https://petar-v.com/talks/GNN-Wednesday.pdf

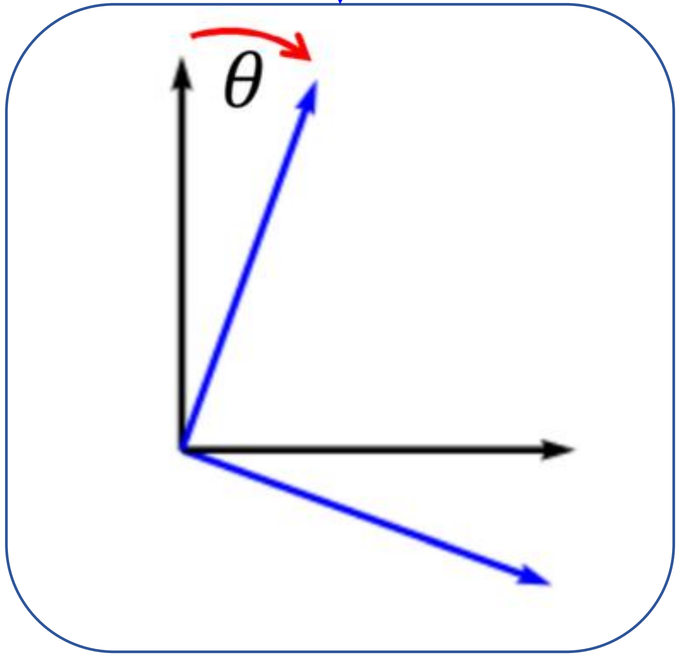
C(θ) =C([b, c, 0, 0, ⋯ , 0, a]): a, b, c 를 weigh로 하는 circulant matrix

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

Fourier Transformation == Change of Basis

$$\mathcal{F}(h) = \boldsymbol{\Phi}^* \boldsymbol{h}$$

$$f(X) = \underbrace{\begin{bmatrix} b & c & & & a \\ a & b & c & & \\ & \ddots & \ddots & \ddots & \\ & & a & b & c \\ c & & & a & b \end{bmatrix}}_{C(\theta)} \underbrace{\begin{bmatrix} \mathbf{X}_0 \\ \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{n-1} \\ \mathbf{X}_{n-2} \end{bmatrix}}_{\mathbf{X}} = \boldsymbol{\Phi} \underbrace{\begin{bmatrix} \hat{\theta}_0 & & & \\ & \hat{\theta}_1 & & \\ & & \ddots & \\ & & & \hat{\theta}_{n-1} \end{bmatrix}}_{\Lambda} \boldsymbol{\Phi}^* \mathbf{X} = \boldsymbol{\Phi}(\hat{\theta} \circ \hat{\mathbf{X}})$$



http://yunshengb.com/wp-content/uploads/2017/10/10202017_GCN_Presentation-1.pdf

C(θ) =C([b, c, 0, 0, ⋯ , 0, a]): a, b, c 를 weigh로 하는 circulant matrix

Spectral Graph Convolution

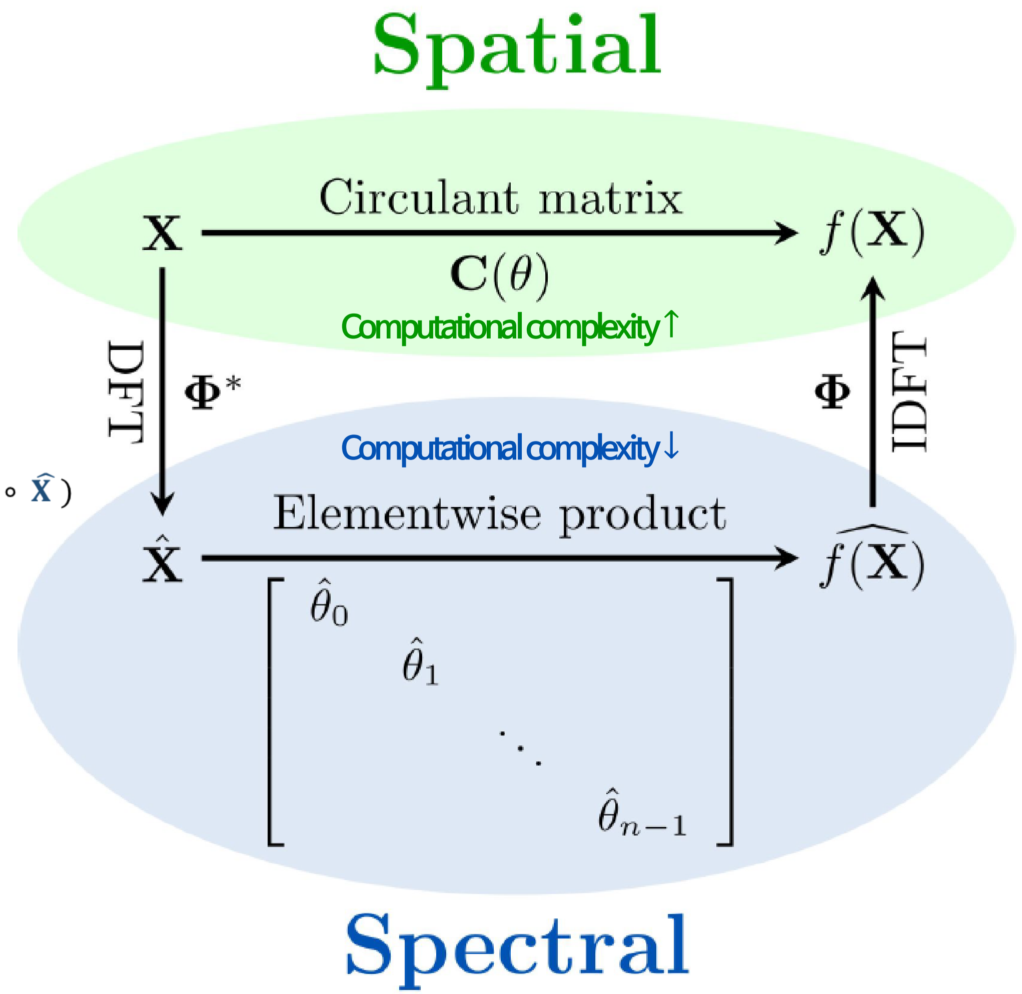
$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

What we have covered so far

- Discrete Fourier Transform (DFT) & Convolution theorem
  - $\mathbf{\Phi}$: Discrete Fourier basis
  - $\hat{\theta}$: Learning parameters

$$C(\theta) \qquad \mathbf{x} \qquad \text{Learning parameters}$$

$$f(X) = \begin{bmatrix} b & c & & & a \\ a & b & c & & \\ & \ddots & \ddots & \ddots & \\ & & a & b & c \\ c & & & a & b \end{bmatrix} \begin{bmatrix} \mathbf{X}_0 \\ \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{n-1} \\ \mathbf{X}_{n-2} \end{bmatrix} = \mathbf{\Phi} \begin{bmatrix} \hat{\theta}_0 & & & \\ & \hat{\theta}_1 & & \\ & & \ddots & \\ & & & \hat{\theta}_{n-1} \end{bmatrix} \mathbf{\Phi}^* \mathbf{x} = \mathbf{\Phi}(\hat{\theta} \circ \hat{\mathbf{X}})$$



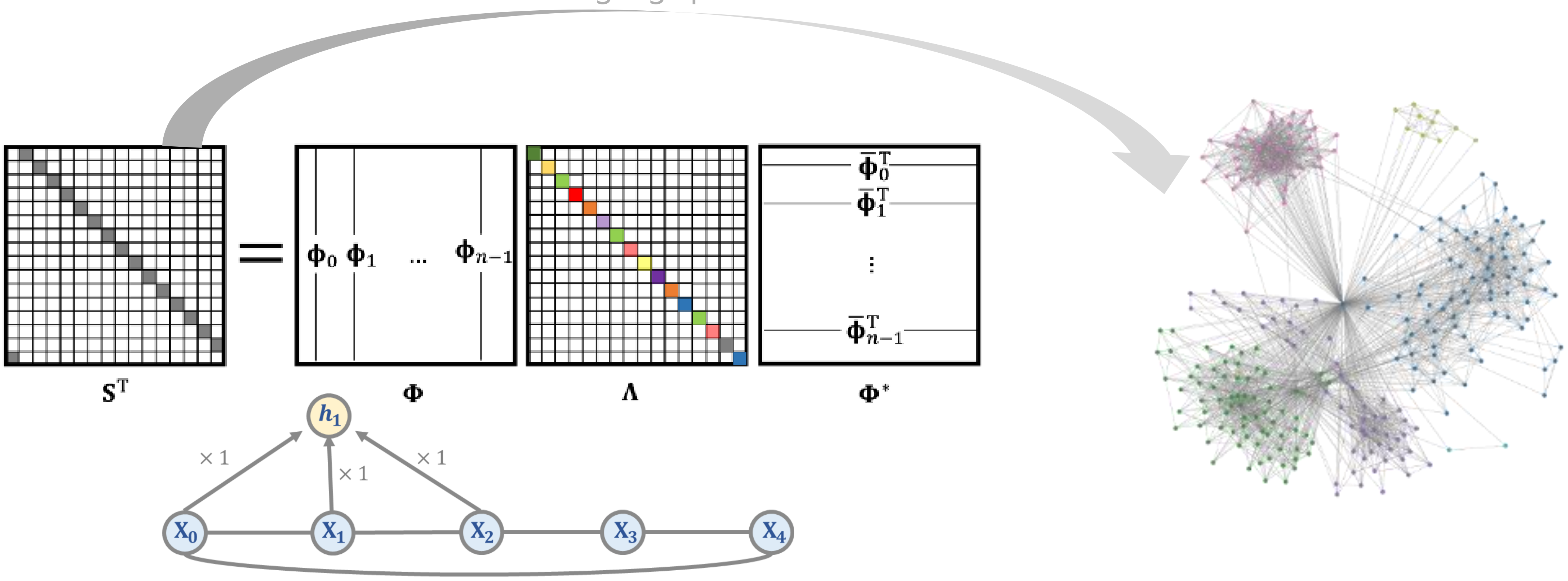
Key idea: 만약 circulant eigenvalues 만 알 수 있다면, circulant matrix는 알 필요가 없다!

https://petar-v.com/talks/GNN-Wednesday.pdf

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

What about graphs?

- Beyond circulants

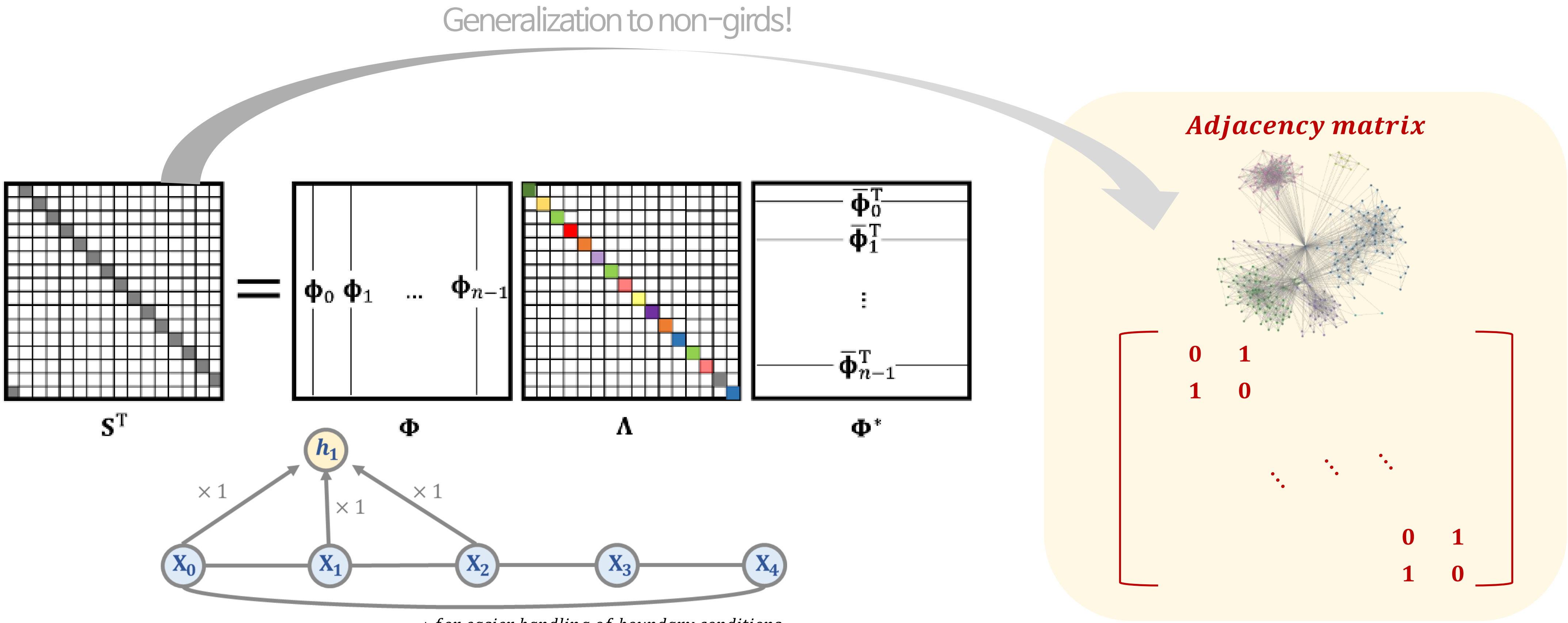


$*$ *for easier handling of boundary conditions*

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

What about graphs?

- Beyond circulants
  - 마찬가지로, **Φ**, “graph Fourier basis” 만 알 수 있다면, eigenvalues를 학습 가능!
  - graph별 adjacency matrix 다르기 때문에 각각의 graph에 대한 **Φ**가 존재



∗ *for easier handling of boundary conditions*

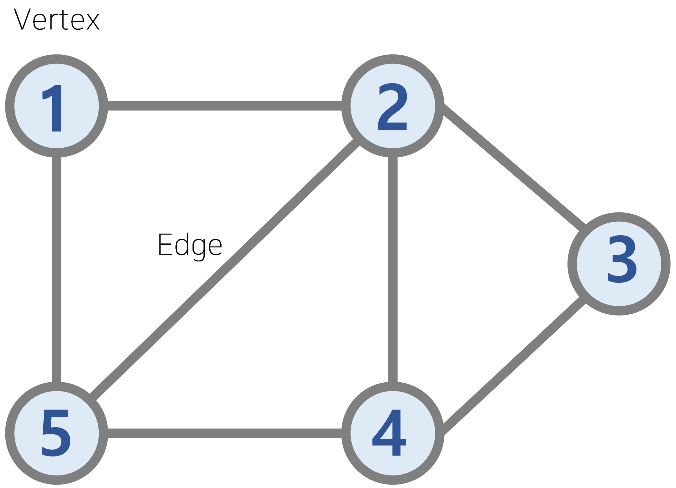
Adjacency matrix가 graph 에서 shift matrix 이며, eigenvector가 graph fourier basis인 이유 증명: Gavili, A., & Zhang, X. P. (2017). On the shift operator, graph frequency, and optimal filtering in graph signal processing

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

What about graphs?

- **Graph Laplacian matrix**
  - Undirected graphs, L is:
    - Symmetric$(L^T = L)$
    - Positive semi−definite$(x^T Lx \geq 0 \; for\, all\, x \in \mathbb{R}^{|V|})$

$$\Longrightarrow eigendecomposable! \; \mathbf{L} = \mathbf{\Phi \Lambda \Phi^*} \quad , \mathbf{\Phi\Phi^*} = \mathbf{I}$$



Adjacency matrix

$A \in R^{n \times n}$

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

Adjacency가 eigendecompose가 안될 수도 있다!

Adjacency의 모든 특징을 담으면서 수학적으로 더 편리함!

Laplacian matrix

$L = D - A \qquad L \in R^{n \times n}$

$[x_1, x_2, x_3, x_4, x_5] \times$

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

$= -x_1 + 4x_2 - x_3 - x_4 - x_5$

$= (x_2 - x_1) + (x_2 - x_3) + (x_2 - x_3) + (x_2 - x_4) + (x_2 - x_5)$

중심 node와 이웃 node 사이의 관계 정보 파악

Adjacency matrix가 graph shift matrix & eigenvector가 graph fourier basis인 이유 증명: Gavili, A., & Zhang, X. P. (2017). On the shift operator, graph frequency, and optimal filtering in graph signal processing

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

What about graphs?

- **Graph Laplacian matrix**
  - Undirected graphs, L is:
    - Symmetric($L^T = L$)
    - Positive semi-definite($x^T L x \geq 0 \; for \; all \; x \in \mathbb{R}^{|V|}$)

$$\Longrightarrow eigendecomposable! \; \mathbf{L} = \mathbf{\Phi \Lambda \Phi}^* \quad , \mathbf{\Phi \Phi}^* = \mathbf{I} \quad , \mathbf{\Phi} = [\emptyset_0, \emptyset_1, \cdots, \emptyset_{n-1}]$$

Eigen – decomposition of graph Laplacian

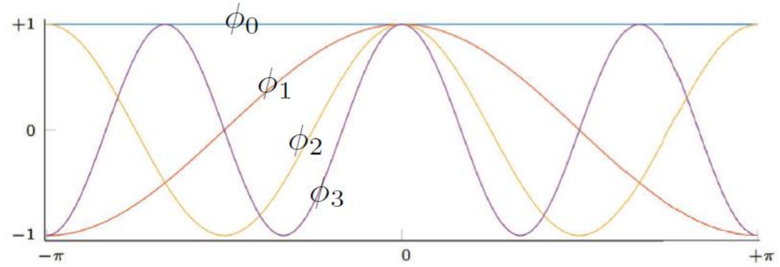
Lap eigenvalues/Spectrum:

$$\mathbf{\Lambda}(\boldsymbol{n \times n})$$

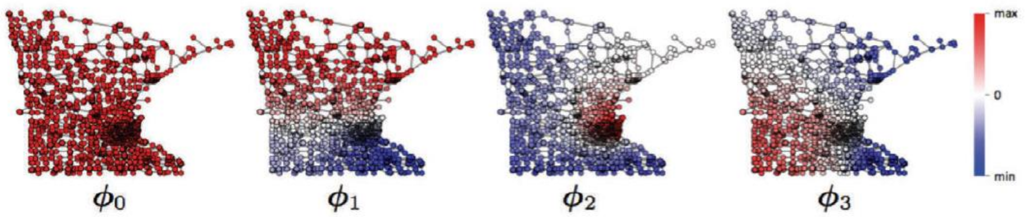
$$\mathbf{L} = \underset{(\boldsymbol{n \times n})}{\mathbf{\Phi}} \begin{bmatrix} \hat{\theta}_\mathbf{0} & & & \\ & \hat{\theta}_\mathbf{1} & & \\ & & \ddots & \\ & & & \hat{\theta}_{\boldsymbol{n-1}} \end{bmatrix} \mathbf{\Phi}^*$$

Lap eigenvectors/
Fourier functions (orthonormal basis)

Fourier functions



Grid/Euclidean domain



Graph domain

Spectral graph clustering [1]

https://atcold.github.io/pytorch-Deep-Learning/en/week13/13-1/

[1] Von Luxburg, A tutorial on spectral clustering, 2007

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

What about graphs?

- Graph Laplacian matrix
  - Undirected graphs, L is:
    - Symmetric$(L^T = L)$
    - Positive semi-definite$(x^T L x \geq 0 \; for\; all\; x \in \mathbb{R}^{|V|})$

$$\Longrightarrow eigendecomposable! \; \mathbf{L} = \mathbf{\Phi\Lambda\Phi}^* \quad , \mathbf{\Phi\Phi}^* = \mathbf{I} \quad , \mathbf{\Phi} = [\emptyset_0, \emptyset_1, \cdots, \emptyset_{n-1}]$$

$\boldsymbol{Eigen - decomposition\ of\ graph\ Laplacian}$

Lap eigenvalues/Spectrum:

$$\mathbf{L} = \underset{(\boldsymbol{n}\times\boldsymbol{n})}{\mathbf{\Phi}} \underset{\boldsymbol{\Lambda}(\boldsymbol{n}\times\boldsymbol{n})}{\begin{bmatrix} \hat{\theta}_0 & & & \\ & \hat{\theta}_1 & & \\ & & \ddots & \\ & & & \hat{\theta}_{n-1} \end{bmatrix}} \mathbf{\Phi}^*$$

Lap eigenvectors/
Fourier functions (orthonormal basis)

Unnomalized Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$

Nomalized Laplacian

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

Graph Convolution

- Spectral convolution

$\mathcal{F}(\mathrm{X}) = \mathbf{\Phi}^{*}\mathrm{X}$: fourier transform

$\odot$: *Pointwise product*

$$\underset{(\boldsymbol{n}\times\mathbf{1})}{\hat{w}} = \begin{bmatrix} \hat{w}(\lambda_0) \\ \vdots \\ \hat{w}(\lambda_{n-1}) \end{bmatrix}$$

$$\underset{(\boldsymbol{n}\times\boldsymbol{n})}{\widehat{w(\boldsymbol{\Lambda})}} = diag(\hat{w}) = \begin{bmatrix} \hat{w}(\lambda_0) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \hat{w}(\lambda_{n-1}) \end{bmatrix}$$

signal h on graph *Graph Fourier transform*

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

filter/kernel

$\mathcal{F}^{-1}$: $\underset{(\boldsymbol{n}\times\boldsymbol{n})}{\mathbf{\Phi}}$, $\mathcal{F}(w)$: $\underset{(\boldsymbol{n}\times\boldsymbol{n})(\boldsymbol{n}\times\mathbf{1})}{\mathbf{\Phi}^{*}\boldsymbol{w} = \hat{\boldsymbol{w}}}$, $\mathcal{F}(h)$: $\underset{(\boldsymbol{n}\times\boldsymbol{n})\ (\boldsymbol{n}\times\mathbf{1})}{\mathbf{\Phi}^{*}\boldsymbol{h}}$

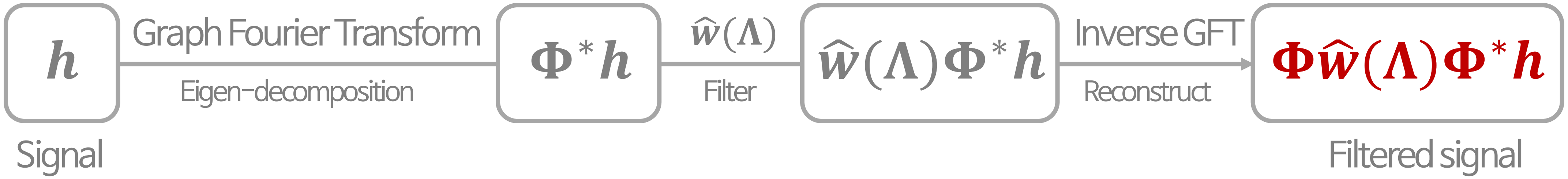
learning parameterss/filter

$$= \underset{(\boldsymbol{n}\times\boldsymbol{n})}{\mathbf{\Phi}}(\underset{(\boldsymbol{n}\times\mathbf{1})}{\hat{\boldsymbol{w}}} \odot \underset{(\boldsymbol{n}\times\mathbf{1})}{\mathbf{\Phi}^{*}\boldsymbol{h}})$$

$$= \underset{(\boldsymbol{n}\times\boldsymbol{n})}{\mathbf{\Phi}}\ \underset{(\boldsymbol{n}\times\boldsymbol{n})}{\hat{\boldsymbol{w}}(\mathbf{\Lambda})}\ \underset{(\boldsymbol{n}\times\mathbf{1})}{\mathbf{\Phi}^{*}\boldsymbol{h}}$$

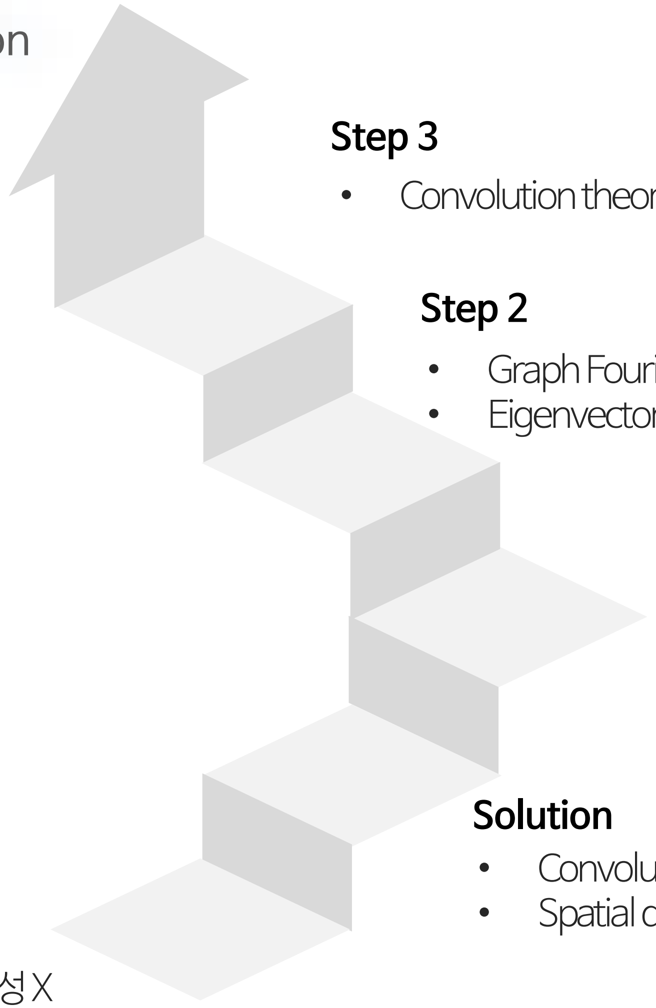
Graph Convolution

- Spectral convolution



Summary

Graph Convolution



Step 3

- Convolution theorem on graph signal (spectral graph convolution)

Step 2

- Graph Fourier transform
- Eigenvector of Laplacian matrix = Graph Fourier basis

Step 1

- Fourier transform
- Eigenvector of circulant matrix = Fourier basis

Solution

- Convolution theorem
- Spatial domain이 아닌 spectral domain에서 convolution 수행

Problem

- Graph data는 grid 형태로 구성 X
- 따라서, Convolution (template matching) 연산 X

Spectral Graph Convolution Networks

Spectral Graph Convolution Networks

Spectral filtering

Spectral/Spatial filtering

Spatial filtering

Spectral Networks and Deep Locally Connected Networks on Graphs

Joan Bruna
New York University
bruna@cims.nyu.edu

Wojciech Zaremba
New York University
woj.zaremba@gmail.com

Arthur Szlam
The City College of New York
aszlam@ccny.cuny.edu

Yann LeCun
New York University
yann@cs.nyu.edu

Abstract

Convolutional Neural Networks are extremely efficient architectures in image and audio recognition tasks, thanks to their ability to exploit the local translational invariance of signal classes over their domain. In this paper we consider possible generalizations of CNNs to signals defined on more general domains without the action of a translation group. In particular, we propose two constructions, one based upon a hierarchical clustering of the domain, and another based on the spectrum of the graph Laplacian. We show through experiments that for low-dimensional graphs it is possible to learn convolutional layers with a number of parameters independent of the input size, resulting in efficient deep architectures.

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

Michaël Defferrard **Xavier Bresson** **Pierre Vandergheynst**

EPFL, Lausanne, Switzerland
{michael.defferrard,xavier.bresson,pierre.vandergheynst}@epfl.ch

Abstract

In this work, we are interested in generalizing convolutional neural networks (CNNs) from low-dimensional regular grids, where image, video and speech are represented, to high-dimensional irregular domains, such as social networks, brain connectomes or words' embedding, represented by graphs. We present a formulation of CNNs in the context of spectral graph theory, which provides the necessary mathematical background and efficient numerical schemes to design fast localized convolutional filters on graphs. Importantly, the proposed technique offers the same linear computational complexity and constant learning complexity as classical CNNs, while being universal to any graph structure. Experiments on MNIST and 20NEWS demonstrate the ability of this novel deep learning system to learn local, stationary, and compositional features on graphs.

Published as a conference paper at ICLR 2017

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

ABSTRACT

We present a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. We motivate the choice of our convolutional architecture via a localized first-order approximation of spectral graph convolutions. Our model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. In a number of experiments on citation networks and on a knowledge graph dataset we demonstrate that our approach outperforms related methods by a significant margin.

$$w * h = \mathcal{F}^{-1}(\mathcal{F}(w) \odot \mathcal{F}(h))$$

Spectral GCN [1]

- Implementation

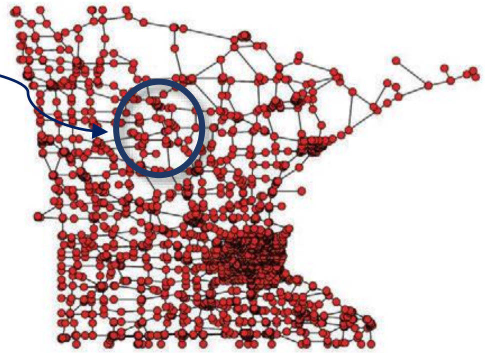
Spatial filter

$$h^{l+1} = \eta(w^l * h^l)$$

$$= \eta(\boldsymbol{\Phi}\, \widehat{\boldsymbol{w}}^l(\boldsymbol{\Lambda})\, \boldsymbol{\Phi}^* \boldsymbol{h}^l)$$

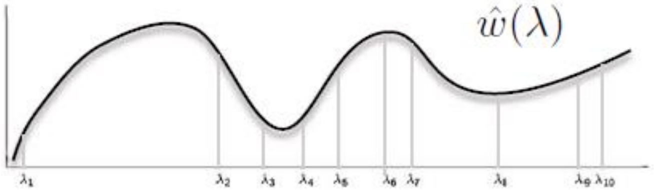
$(\boldsymbol{n} \times \boldsymbol{n})$ $(\boldsymbol{n} \times \boldsymbol{n})$ $(\boldsymbol{n} \times \boldsymbol{1})$

Spectral filter



η: *nonlinear actvation function*

$$\widehat{w(\boldsymbol{\Lambda})} = diag(\widehat{w}) = \begin{bmatrix} \widehat{w}(\lambda_0) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \widehat{w}(\lambda_{n-1}) \end{bmatrix}$$

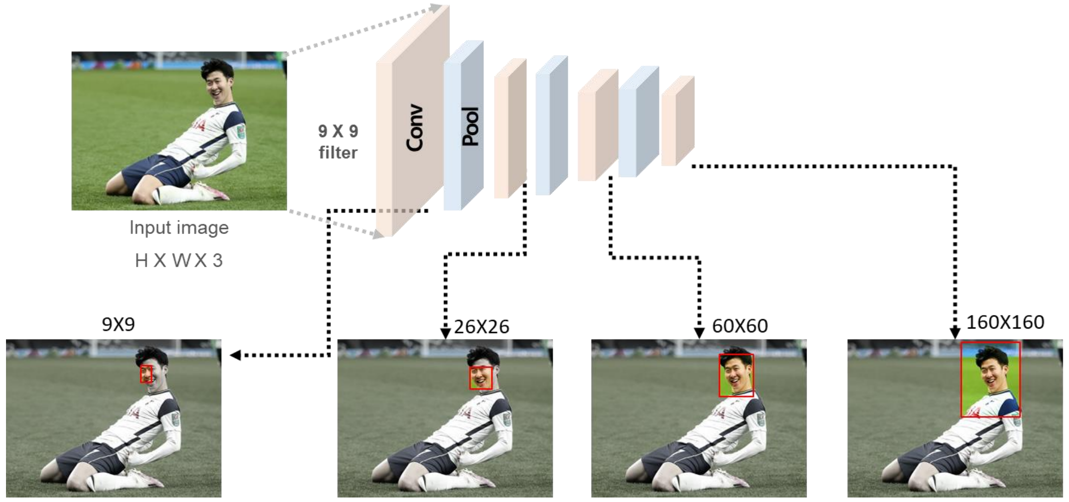


https://www.youtube.com/watch?v=Iiv9R6BjxHM

[1] Bruna, Zaremba, Szlam, LeCun, Spectral Networks and Locally Connected Networks on Graphs, 2014

Spectral GCN [1]

- **Limitation:**
  1) No guarantee of spatial localization of filters



Multi-scale feature

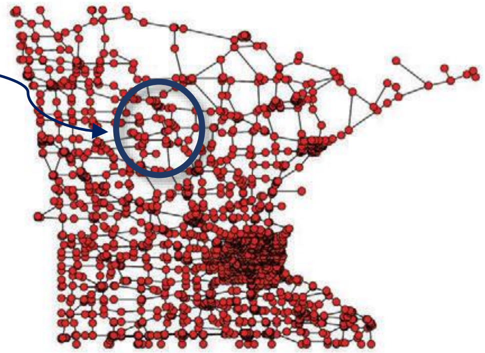
Spatial filter

$$h^{l+1} = \eta(w^l * h^l)$$

$$= \eta(\boldsymbol{\Phi}\, \widehat{\boldsymbol{w}}^l(\boldsymbol{\Lambda})\, \boldsymbol{\Phi}^* \boldsymbol{h}^l)$$

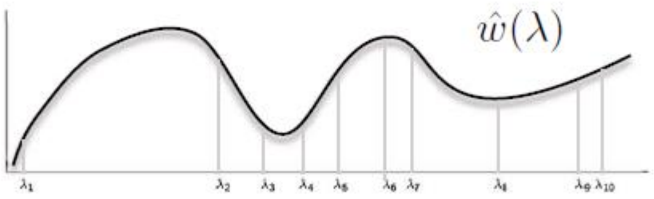
$(\boldsymbol{n} \times \boldsymbol{n})$ $(\boldsymbol{n} \times \boldsymbol{n})$ $(\boldsymbol{n} \times \boldsymbol{1})$

Spectral filter



η: *nonactvation function*

$$\widehat{w(\boldsymbol{\Lambda})} = diag(\widehat{w}) = \begin{bmatrix} \widehat{w}(\lambda_0) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \widehat{w}(\lambda_{n-1}) \end{bmatrix}$$



[1] Bruna, Zaremba, Szlam, LeCun, Spectral Networks and Locally Connected Networks on Graphs, 2014

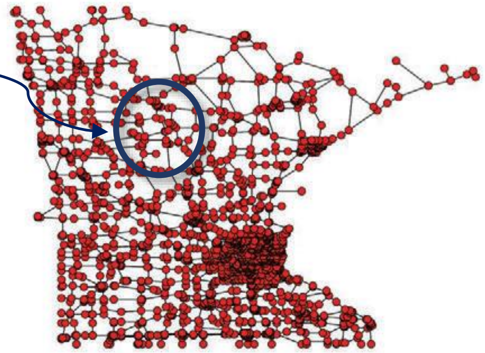
https://www.youtube.com/watch?v=Iiv9R6BjxHM

Spectral GCN [1]

- **Limitation:**
  1) No guarantee of spatial localization of filters
  2) 하나의 layer에서 $\mathcal{O}(n)$ *parameters* 학습($\hat{w}(\lambda_0), \cdots, \hat{w}(\lambda_{n-1})$)
  3) $\mathcal{O}(n^2)$ learning complexity (Fourier transform with full matrix Φ)

Spatial filter

$$h^{l+1} = \eta(w^l * h^l)$$



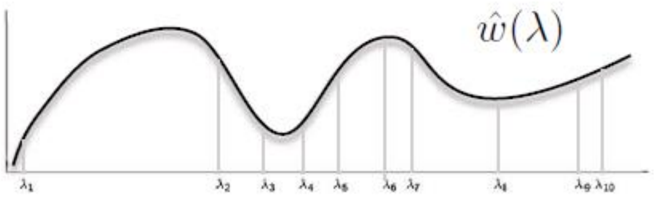
η: *nonactvation function*

$$\widehat{w(\boldsymbol{\Lambda})} = diag(\hat{w}) = \begin{bmatrix} \hat{w}(\lambda_0) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \hat{w}(\lambda_{n-1}) \end{bmatrix}$$

$$= \eta(\boldsymbol{\Phi}\ \hat{\boldsymbol{w}}^{\boldsymbol{l}}(\boldsymbol{\Lambda})\ \boldsymbol{\Phi}^* \boldsymbol{h}^{\boldsymbol{l}})$$

$(\boldsymbol{n} \times \boldsymbol{n})$ $(\boldsymbol{n} \times \boldsymbol{n})$ $(\boldsymbol{n} \times \boldsymbol{1})$

Spectral filter



https://www.youtube.com/watch?v=Iiv9R6BjxHM

[1] Bruna, Zaremba, Szlam, LeCun, Spectral Networks and Locally Connected Networks on Graphs, 2014

ChebNet[1]

- Less Computation!
  - $\mathcal{O}(n^2)$ complexity: direct use of Laplacian eigenvectors, $\Phi(n \times n)$

$$\mathcal{O}(w * h) = \mathcal{O}\left(\underset{\substack{(n \times n) \\ Full\ matrix}}{\Phi}\ \hat{w}^l(\Lambda)\ \Phi^* h^l\right) = \mathcal{O}(n^2)$$

https://www.youtube.com/watch?v=Iiv9R6BjxHM

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016

ChebNet[1]

- Less Computation!
  - $\mathcal{O}(\boldsymbol{n}^2)$ complexity: direct use of Laplacian eigenvectors, $\boldsymbol{\Phi}(\boldsymbol{n} \times \boldsymbol{n})$

    → Eigen-decomposition을 하지 않고, 학습할 수 없을까?

$$w \ * h = \boldsymbol{\Phi} \ \widehat{\boldsymbol{w}}(\boldsymbol{\Lambda}) \ \boldsymbol{\Phi}^* \boldsymbol{h}$$

$$\widehat{\boldsymbol{w}}(\boldsymbol{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \ \boldsymbol{\Lambda}^k$$

Polynomial parameterization!

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016

ChebNet[1]

- Less Computation!
  - $\mathcal{O}(\boldsymbol{n}^2)$ complexity: direct use of Laplacian eigenvectors, $\boldsymbol{\Phi}(\boldsymbol{n} \times \boldsymbol{n})$

    → Eigen-decomposition을 하지 않고, 학습할 수 없을까?

$$w \ast h = \boldsymbol{\Phi}\, \widehat{\boldsymbol{w}}(\boldsymbol{\Lambda})\, \boldsymbol{\Phi}^* \boldsymbol{h}$$

$$= \boldsymbol{\Phi}\left(\sum_{k=0}^{K-1} \theta_k \boldsymbol{\Lambda}^k\right) \boldsymbol{\Phi}^* \boldsymbol{h}$$

$$= \sum_{k=0}^{K-1} \theta_k \boldsymbol{\Phi}\, \boldsymbol{\Lambda}^k\, \boldsymbol{\Phi}^* \boldsymbol{h}$$

$$= \sum_{k=0}^{K-1} \theta_k \boldsymbol{L}^k \boldsymbol{h}$$

No eigen-decomposition!

$$\boldsymbol{L} = \boldsymbol{\Phi\Lambda\Phi}^*$$

$$\boldsymbol{L}^2 = \boldsymbol{\Phi\Lambda\Phi}^*\boldsymbol{\Phi\Lambda\Phi}^* = \boldsymbol{\Phi\Lambda I \Lambda\Phi}^* = \boldsymbol{\Phi\Lambda}^2\boldsymbol{\Phi}^*$$

$$\boldsymbol{L}^k = \boldsymbol{\Phi\Lambda}^k\boldsymbol{\Phi}^*$$

$\boldsymbol{\Phi}$: Eigenvectors of L
(orthonormal b.c. L is symmertric PSD)

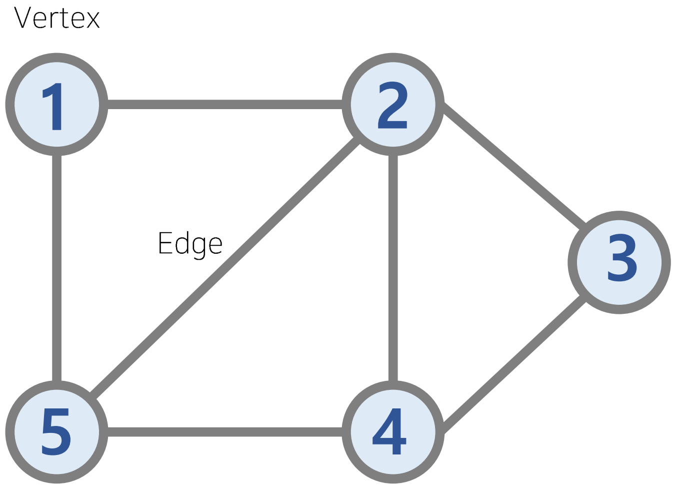
θ_k : $\mathcal{O}(1)$ trainable parameters per layer

$O(Edge. K\ times) = O(n)\ for\ sparse\ (real - world)\ graphs$

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016

ChebNet[1]

- **Localization**
  - Kth-order polynomials filter는 정확히 k-hop만큼 localize 한다!
  - Spectral domain이 아닌 Spatial domain! (no eigen-decomposition of Laplacian)

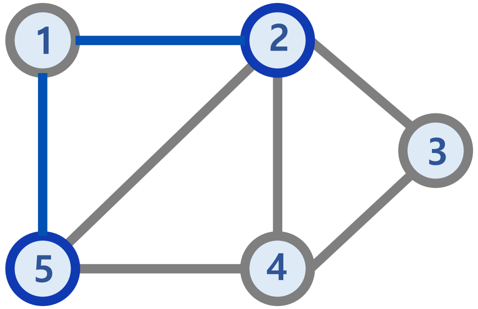


$K=3$

$$\sum_{k=0}^{K-1} \theta_k \mathbf{L}^k \boldsymbol{h} = \theta_0 * \begin{bmatrix} h^{(1)} \\ \vdots \\ h^{(5)} \end{bmatrix} + \theta_1 * \left[\text{1-hop matrix} \right] \begin{bmatrix} h^{(1)} \\ \vdots \\ h^{(5)} \end{bmatrix} + \theta_2 * \left[\text{2-hop matrix} \right] \begin{bmatrix} h^{(1)} \\ \vdots \\ h^{(5)} \end{bmatrix}$$

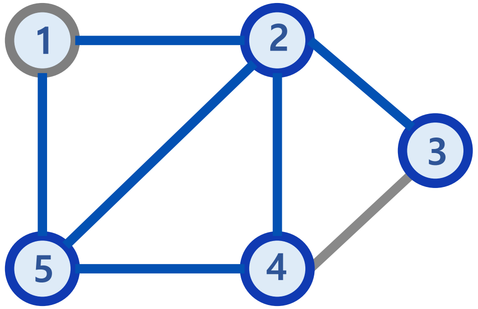
$1-hop$ neighbors

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



$2-hop$ neighbors

6	-5	1	2	-4
-5	20	-5	-5	-5
1	-5	6	-4	2
2	-5	-4	12	-5
-4	-5	2	-5	12



$$w * h = \mathbf{\Phi}\, \hat{\boldsymbol{w}}(\boldsymbol{\Lambda})\, \mathbf{\Phi}^* \boldsymbol{h}$$

$$\widehat{w}(\boldsymbol{\Lambda}) = diag(\widehat{w}) = \begin{bmatrix} \widehat{w}(\lambda_0) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \widehat{w}(\lambda_{n-1}) \end{bmatrix}$$

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016

ChebNet[1]

- Chebyshev expansion
  - Monomials basis are unstable under coefficients perturbation(최적화하기 어려움)

    → Chebyshev polynomials for recursive(재귀) formulation!

Chebyshev polynomials
for recursive formulation

$T_0(x) = 1 \quad T_1(x) = x \quad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$

$-1 \le T_k(x) \le 1 \; for \; x \in [-1,1]$

$$\sum_{k=0}^{K-1} \theta_k \, \boldsymbol{x}^k = \theta_0 + \theta_1 x + \theta_2 \boldsymbol{x}^2 + \theta_3 \boldsymbol{x}^3 + \cdots$$

High-order polynomial has non-orthogonal basis $1, x, x^2, x^3, \ldots$

Unstable under perturbation of coefficients

$$\sum_{k=0}^{K-1} \theta_k \, T_k(x) = \theta_0 + \theta_1 x + \theta_2 T_2(x) + \theta_3 T_3(x) + \cdots$$

Chebyshev polynomials $\{T_k(x)\}$:orthogonal basis

Stable under coefficients perturbation

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016

ChebNet[1]

- Chebyshev expansion
  - Monomials basis are unstable under coefficients perturbation (최적화하기 어려움)

    → Chebyshev polynomials for recursive(재귀) formulation!

Chebyshev polynomials for recursive formulation:

$T_0(x) = 1 \quad T_1(x) = x \quad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$

$-1 \le T_k(x) \le 1 \; for \; x \in [-1,1]$

$$\sum_{k=0}^{K-1} \theta_k T_k(x) = \theta_0 + \theta_1 x + \theta_2 T_2(x) + \theta_3 T_3(x) + \cdots$$

Chebyshev polynomials $\{T_k(x)\}$: orthogonal basis

$$w * h = \boldsymbol{\Phi} \, \hat{\boldsymbol{w}}(\boldsymbol{\Lambda}) \, \boldsymbol{\Phi}^* \boldsymbol{h} = \sum_{k=0}^{K-1} \theta_k \boldsymbol{L}^k \boldsymbol{h}$$

$L^0, L^1, L^2, L^3, \ldots$ High-order polynomial has non-orthogonal basis

$$\sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}) \boldsymbol{h}$$

$-1 \le T_k(\tilde{L}) \le 1 \; for \tilde{L} \in [-1,1]$

Eigenvalues of $L \in [0, \lambda_{max}]$

$\tilde{L} = \frac{2L}{\lambda_{max}} - I$ (rescaled L)

ChebNet[1]

- Implementation

Spectral GCN

$$h^{l+1} = \eta(w^l * h^l)$$

$$= \eta(\boldsymbol{\Phi}\, \widehat{\boldsymbol{w}}^{\boldsymbol{l}}(\boldsymbol{\Lambda})\, \boldsymbol{\Phi}^* \boldsymbol{h}^{\boldsymbol{l}})$$

$$h^{l+1} = \eta(w^l * h^l)$$

$$= \eta(\widehat{w}^l(\tilde{L})\, h^l)$$

$$= \eta(\sum_{k=0}^{K-1} w_k^l\, T_k(\tilde{L}) h^l)$$

$-1 \leq T_k(\tilde{L}) \leq 1 \; for \tilde{L} \in [-1,1]$

Eigenvalues of $L \in [0, \lambda_{max}]$

$\tilde{L} = \frac{2L}{\lambda_{max}} - I$ (rescaled L)

[1] Defferrard, Bresson, Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, 2016

Simplified ChebNet[1] (Vanilla GCNs)

- Vanilla GCNs is a simplification of ChebNets.

$-1 \le T_k(\tilde{L}) \le 1 \; for \tilde{L} \in [-1,1]$

Eigenvalues of $L \in [0, \lambda_{max}]$

$\tilde{L} = \frac{2L}{\lambda_{max}} - I$ (rescaled L)

$T_0(\tilde{L}) = I$

$T_1(\tilde{L}) = \tilde{L}$

$T_k(\tilde{L}) = 2\tilde{L}T_{k-1}(\tilde{L}) - T_{k-2}(\tilde{L})$

Suppose $K=2$

$$h^{l+1} = \eta(w^l * h^l) = \eta(\sum_{k=0}^{K-1} w_k^l T_k(\tilde{L}) h^l)$$

$$= \eta((w_0^l T_0(\tilde{L}) + w_1^l T_1(\tilde{L}))h^l)$$

$$= \eta((w_0^l + w_1^l \tilde{L})h^l)$$

Constrain $w^l = w_0^l = -w_1^l \qquad \lambda_{max} = 2$

$$= \eta((w^l - w^l(L - I))h^l) \qquad \tilde{L} = L - I$$

$$= \eta(w^l(2I - L)h^l)$$

$$= \eta(w^l(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})h^l)$$

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

Renormalization trick

$\tilde{A} = A + I, \qquad \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

(Add self-loop to graphs)

$$= \eta(w^l(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})h^l)$$

$$= \eta(w^l(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})h^l)$$

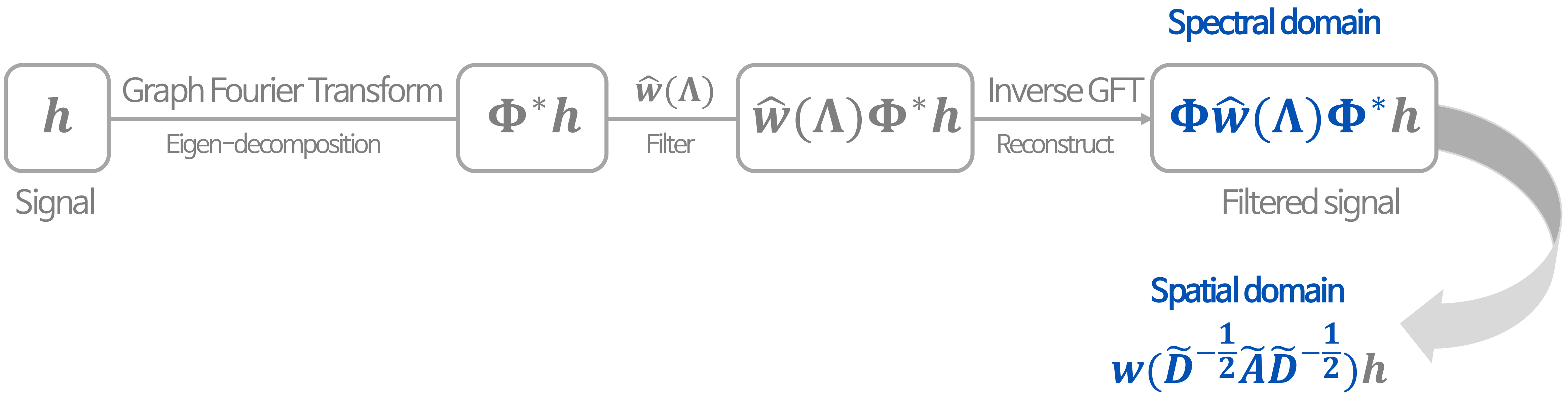
$I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$
↓
$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

Problem: Operator with largest eigenvalue in [0,2] may cause divergence

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

Graph Convolution

- From Spectral domain to Spatial domain

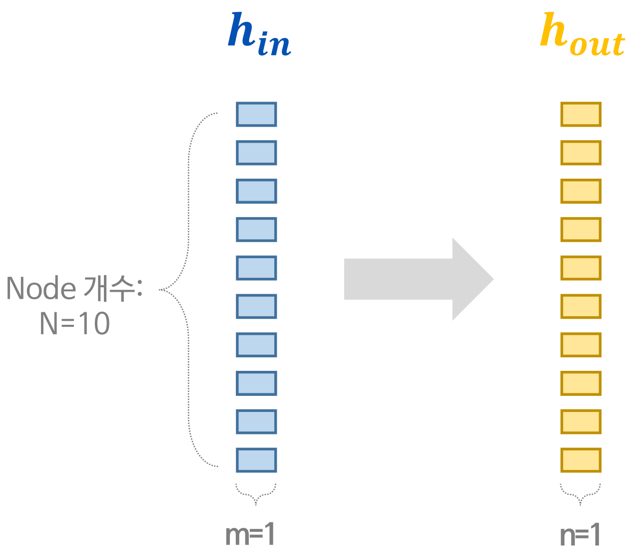


Simplified ChebNet[1] (Vanilla GCNs)

Single channel

$$h_{out} = \eta(w\underbrace{\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\right)}_{\text{고정된값}}h_{in})$$

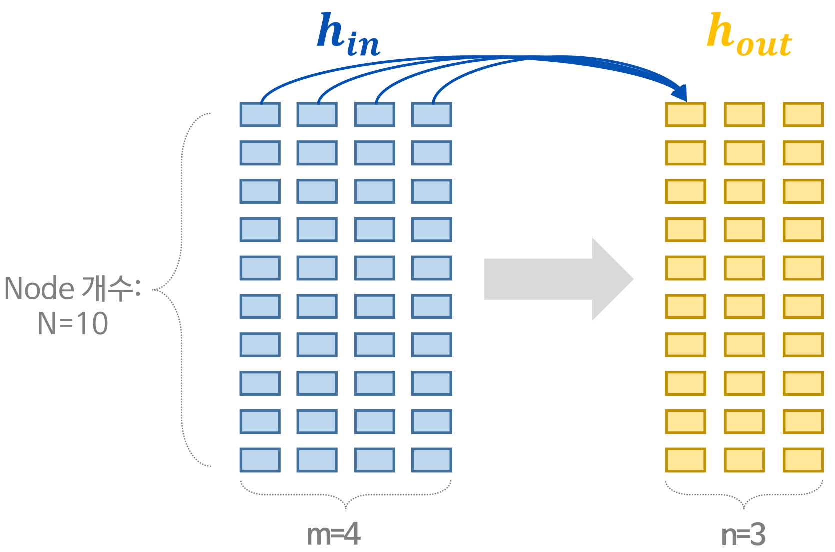
(10×1) (10×10) (10×1)



Multiple channel

$$h_{out} = \eta(w^{(mn)}\underbrace{\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\right)}_{\text{고정된값}}h_{in}^{m})$$

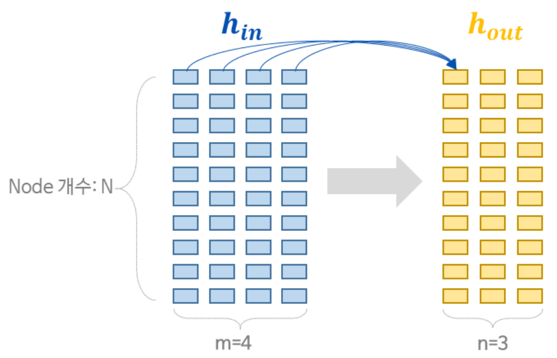
(10×3) (4×3) (10×10) (10×4)



$w^{(mn)} \in \mathbb{R}^{4\times3}$

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

Simplified ChebNet[1] (Vanilla GCNs)



$w^{(mn)} \in \mathbb{R}^{4\times 3}$

Multiple channel

$$\boldsymbol{h_{out}} = \boldsymbol{\eta}\left(\boldsymbol{w^{(mn)}}\left(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\right)\boldsymbol{h_{in}^{m}}\right) \Rightarrow \boldsymbol{h_{out}} = \boldsymbol{\eta}\left(\left(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\right)\boldsymbol{h_{in}^{m}}\boldsymbol{w^{(mn)}}\right)$$

$(N\times n)$ $(m\times n)$ $(N\times N)$ $(N\times m)$ $(N\times n)$ $(N\times N)$ $(N\times m)$ $(m\times n)$

2-layered neural network structure

$Renormalization\ trick:\ \hat{A} = \widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}},\ \tilde{A} = (A + I_N)$

$$\boldsymbol{Z = f(X,A) = softmax\left(\hat{A}Relu\left(\hat{A}XW^{(0)}\right)W^{(1)}\right)}$$

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

Summary of Spectral GCN

Spectral GCN(2014) → ChebNet(2016) → Simplified ChebNet(2017) → …

Spectral GCN(2014)

NN기반 Spectral graph-based 최초의 방법론

Filter를 학습 가능한 파라미터로 대체

ChebNet(2016)

Filter에 Chebyshev expansion을 적용해 학습해야 할 파라미터와 계산 복잡도 감소

Eigen-decomposition 생략

Simplified ChebNet(2017)

ChebyNet을 간소화하여 2-hop neighbors 고려

안정적인 학습을 위해 Self-loop 추가한 Renormalization trick을 제안

Spectral filtering → Spatial filtering

Spectral filtering

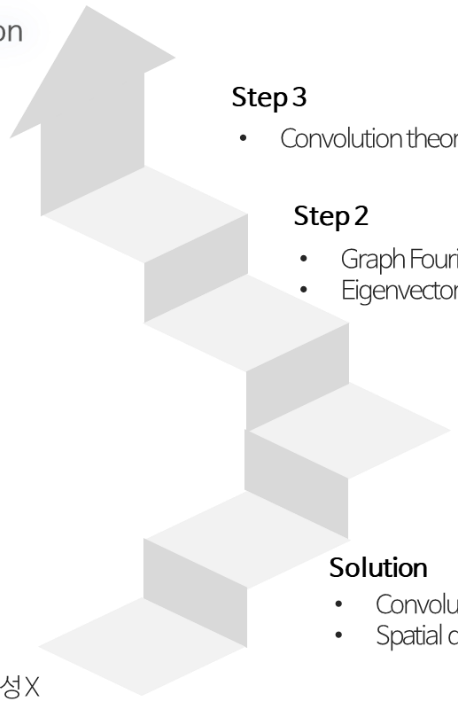
1) Spectral Graph CNN (Brunaet et al. ICLR 2014)
2) Spline GCN (Henaff et al. arXiv 2015)
3) ChebNet(Defferardet et al. NIPS 2016)
4) Simplified ChebNet(Kipf& Welling, ICLR 2017)

Spatial filtering

1) Simplified ChebNet (Kipf& Welling, ICLR 2017)
2) GraphSage (Hamilton et al. NIPS 2017)
3) MPNN (Glimer et al. ICML 2017)
4) GAT (Veličković et al. ICLR 2018)

Conclusion

Graph Convolution



Step 3
- Convolution theorem on graph signal (spectral graph convolution)

Step 2
- Graph Fourier transform
- Eigenvector of Laplacian matrix = Graph Fourier basis

Step 1
- Fourier transform
- Eigenvector of circulant matrix = Fourier basis

Solution
- Convolution theorem
- Spatial domain이 아닌 spectral domain에서 convolution 수행

Problem
- Graph data는 grid 형태로 구성 X
- 따라서, Convolution (template matching) 연산 X

Chapter 1

- Graph Neural Networks
- Graph structure
- Graph data

Chapter 2

- Convolution theorem
- Graph Fourier transform
- Spectral Graph Convolution

Chapter 3

- Spectral Graph CNN (Brunaet al. ICLR 2014)
- ChebNet (Defferardet al. NIPS 2016)
- Simplified ChebNet (Kipf & Welling, ICLR 2017)

Spectral GCN (2014)

NN기반 Spectral graph-based 최초의 방법론

Filter를 학습 가능한 파라미터로 대체

ChebNet (2016)

Filter에 Chebyshev expansion을 적용해 학습해야 할 파라미터와 계산 복잡도 감소

Eigen-decomposition 생략

Simplified ChebNet (2017)

ChebyNet을 간소화하여 2-hop neighbors 고려

안정적인 학습을 위해 Self-loop 추가한 Renormalization trick을 제안

…

"Graphs are the most important discrete models in the world!"

G. Strang (MIT)

https://math.mit.edu/~gs/

Reference

https://www.youtube.com/watch?v=Iiv9R6BjxHM (NYU Deep Learning Spring 2020, Xavier Bresson)

https://ocw.snu.ac.kr/node/31254 (SNU Graph Convolutional Network Spring 2020, Jin Young Choi)

http://yunshengb.com/wp-content/uploads/2017/10/10202017_GCN_Presentation-1.pdf (UCLA ML Seminar Fall 2017, Yunsheng Bai)

https://www.youtube.com/watch?v=uF53xsT7mjc&t=1055s (Theoretical Foundations of Graph Neural Networks, DeepMind, Petar Veličković)

https://www.youtube.com/watch?v=0d2jh3sCdM4 ([DMQA Open Seminar] Graph-Based Semi-supervised Learning, Jiyoon Lee)

https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028 (Deriving convolution from first principles, DeepMind, Michael Bronstein)

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

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

Welling, M., & Kipf, T. N. (2016). Semi-supervised classification with graph convolutional networks. In J. International Conference on Learning Representations (ICLR 2017).

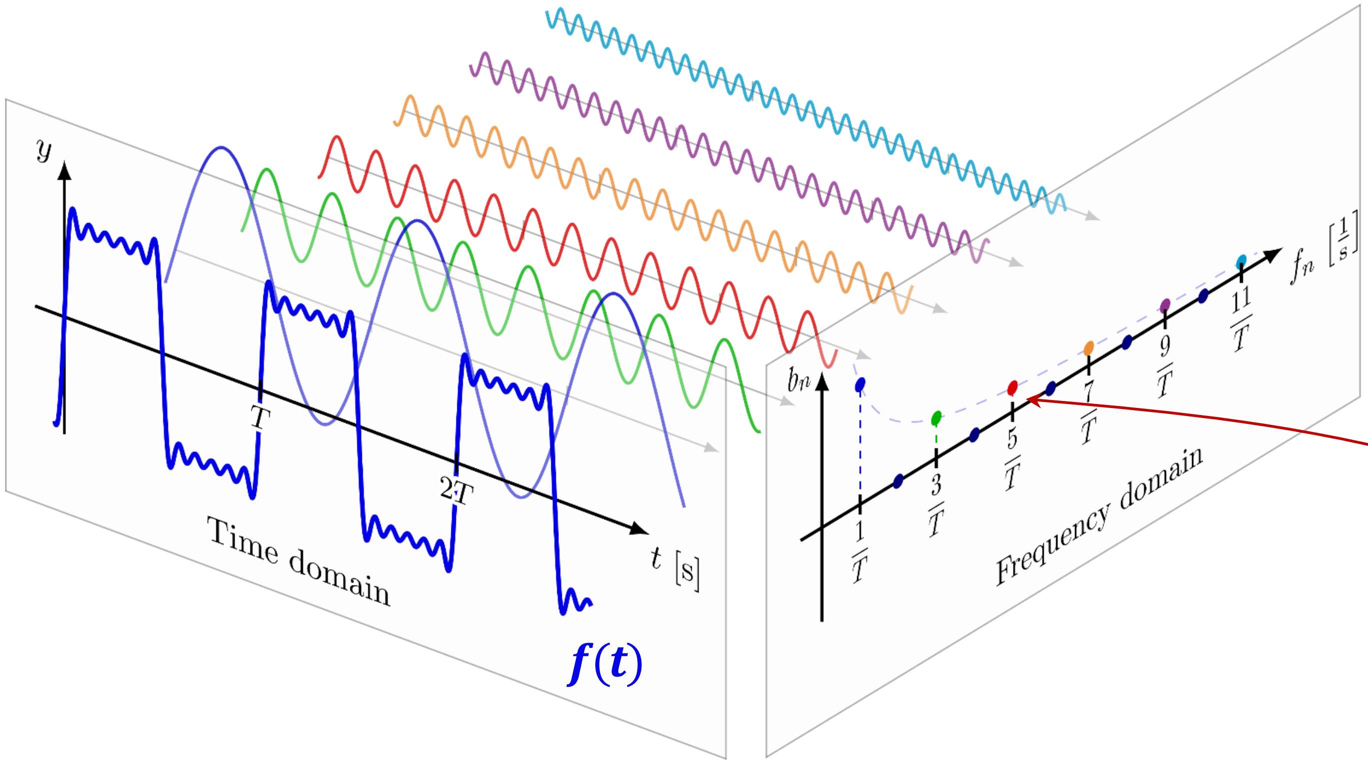
상기 참조 문헌 이외 출처는 각 슬라이드 좌측 하단 표기

Thank you

How to define Fourier transforms for graphs?

How to define Fourier transforms for graphs?

- Fourier transform
  - 임의의 입력 신호(signal)를 다양한 주파수(frequency) 를 갖는 주기함수들의 합으로 분해하여 표현
  - 푸리에 변환에 사용하는 주기함수는 sin, cos 삼각함수
  - 푸리에 변환은 고주파부터 저주파까지 다양한 주파수까지 다양한 주파수 대역의 sin, cos 삼각 함수들로 원본 신호를 분해하는 것



Fourier transform

$$\hat{f}(\xi) = \int_{\mathbb{R}} f(t)\, e^{-2\pi i \xi t} dt$$

Inverse Fourier transform

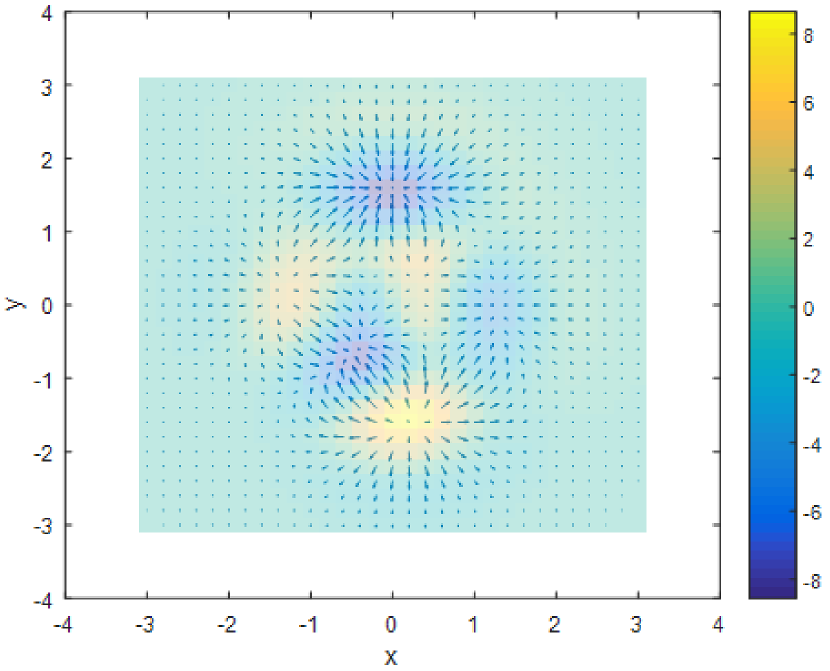
Frequency

$$f(x) = \int_{\mathbb{R}} \hat{f}(\xi) \boxed{e^{2\pi i \xi t}} d\xi$$

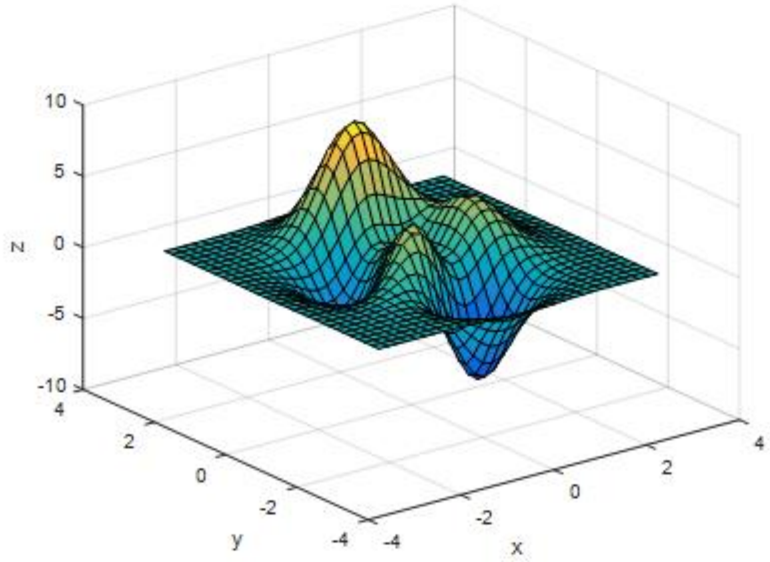
How to define Fourier transforms for graphs?

- Laplace operator
  - 벡터 기울기의 발산(divergence)을 뜻하며, 함수의 높고 낮음을 표시

$$\Delta f = \nabla \cdot \nabla f = \nabla^2 f$$



Divergence



Scalar 함수

https://ralasun.github.io/deep%20learning/2021/02/15/gcn/

How to define Fourier transforms for graphs?

Frequency

$$f(x) = \int_{\mathbb{R}} \hat{f}(\xi)\, e^{2\pi i \xi t} d\xi$$

Complex Exponentials form an Orthonormal basis

$$e^{i\frac{2\pi k}{N}n} = \cos\left(\frac{2\pi k}{N}n\right) + i * \sin\left(\frac{2\pi k}{N}n\right)$$

$$N = 4, k = 0, e_0 = 1, \mathrm{n} = 4: [1,1,1,1]$$

$$N = 4, k = 1, e_1 = \cos\left(\frac{\pi}{2}n\right) + i * \sin\left(\frac{\pi}{2}n\right), n = 4: [1, i, -1, -i]$$

$$N = 4, k = 2, e_2 = \cos(\pi n) + i * \sin(\pi n), n = 4: [1, -1, 1, -1]$$

$$N = 4, k = 3, e_3 = \cos\left(\frac{3\pi}{2}n\right) + i * \sin\left(\frac{3\pi}{2}n\right), n = 4: [1, -i, -1, i]$$

N=4 means any signal x=$[x_0, x_1, x_2, x_3]$ can be expanded as a sum of these 4 bases scaled,
i.e x = $\boldsymbol{x_0 e_0 + x_1 e_1 + x_2 e_2 + x_3 e_3}$

http://yunshengb.com/wp-content/uploads/2017/10/08222017_GCN.pdf

How to define Fourier transforms for graphs?

Frequency

$$f(x) = \int_{\mathbb{R}} \hat{f}(\xi)\, e^{2\pi i \xi t} d\xi$$

Complex Exponentials form an Orthonormal basis

$$Check: < e_0, e_1 >= [1,1,1,1] * [1, i, -1, -i]^T = 1 + i - 1 - i = 0$$

$$Check: < e_2, e_3 >= [1,-1,1,-1] * [1, -i, -1, i]^T = 1 + i - 1 - i = 0$$

$$< e^{i\frac{2\pi k_1}{N}n}, e^{i\frac{2\pi k_2}{N}n} >= 0 \; when \; k_1 \neq k_2 \; and \; \left| e^{i\frac{2\pi k}{N}n} \right| = 1 \; \forall k \in Z$$

$$\rightarrow \left\{ e^{i\frac{2\pi k}{N}n} \right\} \; forms \; an \; orthonoamal \; basis$$

http://yunshengb.com/wp-content/uploads/2017/10/08222017_GCN.pdf

How to define Fourier transforms for graphs?

1) Essentially, find a set of Fourier bases so that a graph signal can be decomposed
2) Do not want to use complex exponentials
   - Complex exponentials involve complex numbers and do NOT involve the link structure of the graph
3) How to find a set of vectors which are orthonormal to each other and related to the graph structure?

http://yunshengb.com/wp-content/uploads/2017/10/08222017_GCN.pdf

How to define Fourier transforms for graphs?

Complex Exponentials are eigenfunctions

$$\Delta\left(e^{2\pi i\xi t}\right) = \frac{\partial^2}{\partial t^2} e^{2\pi i\xi t} = -(2\pi\xi)^2 e^{2\pi i\xi t}$$

Eigenfunction ($e^{2\pi i\xi t}$) Eigenvalue ($-(2\pi\xi)^2$)

(연산자) X (함수) = (상수) X (함수)

$$Af = \lambda f$$

Eigenfunction (고유함수) (f) Eigenvalue (고유값) (λ)

$e^{2\pi i\xi t}(complex\ exponentials)$: 1차원 $Laplacian$ 연산의 $eigenfunction$

"Fourier tranform은 Laplacian operator Δ의 eigenfunction들의 합을 분해하는 변환"

Frequency 성분 (Laplacian operator Δ의 eigenfunction들)

The Emerging Field of Signal Processing on Graphs (IEEE 2013)

How to define Fourier transforms for graphs?

The classical Fourier transform is the expansion of a function f in terms of the complex exponentials, which are the eigenfunctions of the **one-dimensional Laplace operator**

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \, e^{i\frac{2\pi k}{N}n}$$

Analogously, we can define the graph Fourier transform of a function f on the vertices of G as the expansion of f in terms of the eigenvectors of the **graph Laplacian**

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \, e^{i\frac{2\pi k}{N}n}$$

Eigenfunctions of the graph Laplacian matrix

The Emerging Field of Signal Processing on Graphs (IEEE 2013)

How to define Fourier transforms for graphs?

$$\Delta\left(e^{2\pi i\xi t}\right) = \frac{\partial^2}{\partial t^2} e^{2\pi i\xi t} = -(2\pi\xi)^2 e^{2\pi i\xi t}$$

Eigenfunction ($e^{2\pi i\xi t}$), Eigenvalue ($-(2\pi\xi)^2$)

“Fourier transform은 Laplacian operator Δ의 eigenfunctions의 합을 분해하는 변환”

“Graph Fourier transform은 graph Laplacian 의 eigenvectors의 합을 분해하는 변환”

Are Laplacian Eigenvectors Orthonormal?

Check:

$$\text{eig}\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix} =$$

-0.4082	-0.4149	-0.5053	-0.2887	-0.5670	0.0323
-0.4082	-0.3094	0.0403	-0.2887	0.6581	0.4685
-0.4082	-0.0692	0.7590	-0.2887	-0.2051	-0.3564
-0.4082	0.2209	0.2007	0.5774	-0.3084	0.5620
-0.4082	-0.2209	-0.2007	0.5774	0.3084	-0.5620
-0.4082	0.7935	-0.2940	-0.2887	0.1140	-0.1444

$[-0.4082 \quad -0.4082 \quad -0.4082 \quad -0.4082 \quad -0.4082 \quad -0.4082]^*$

$[-0.4149 \quad -0.3094 \quad -0.0692 \quad 0.2209 \quad -0.2209 \quad 0.7935]^T =$

9.7145e-17 = 0 :)

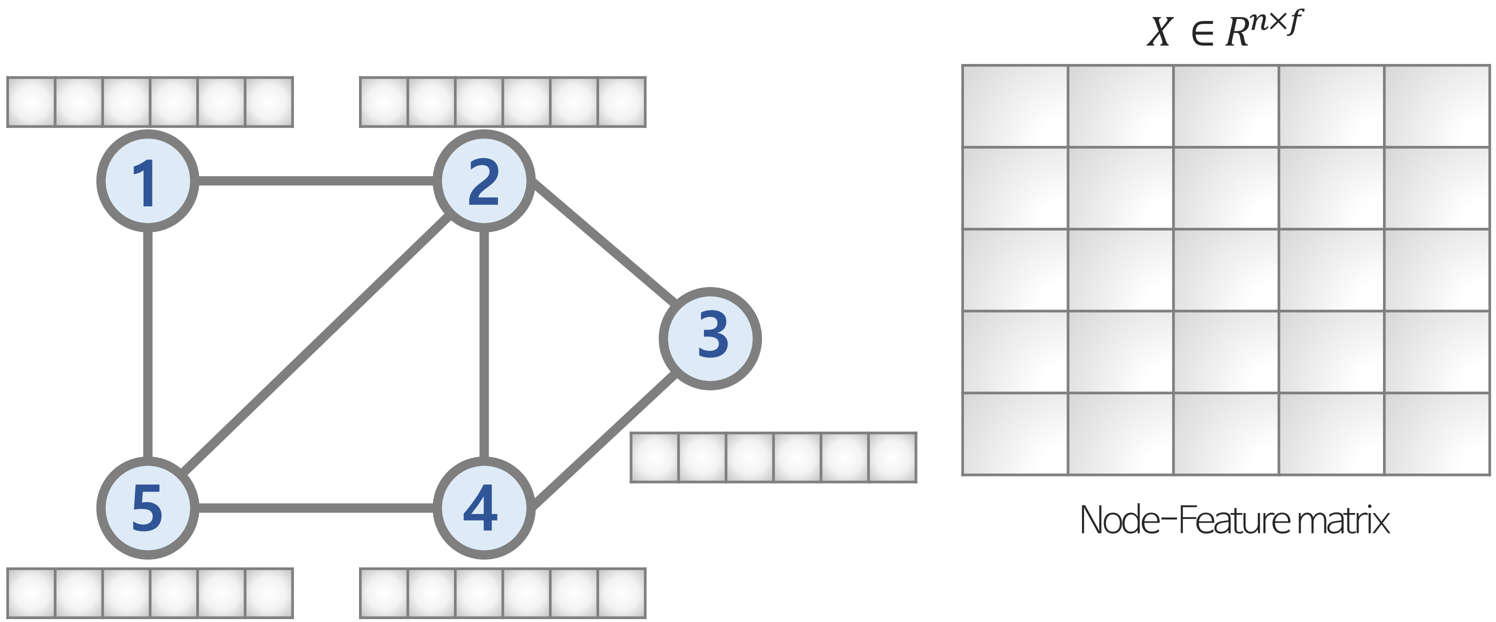
http://yunshengb.com/wp-content/uploads/2017/10/08222017_GCN.pdf

Simplified ChebNet (Vanilla GCNs)

Simplified ChebNet[1] (Vanilla GCNs)

Input & Output

Feature extraction 대상은 node에 대한 정보→ “Node-feature matrix”



$Z \in R^{n \times c}$

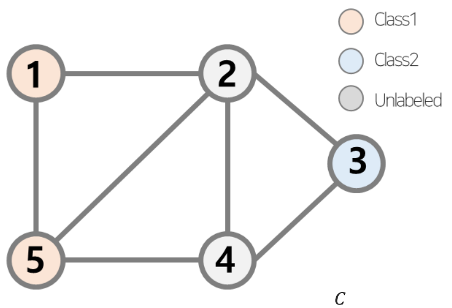
1	0
0	1
0	1
1	0
1	0

Node-Class matrix

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

$$Renormalization\ trick:\ \hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}},\ \tilde{A} = (A + I_N)$$

Simplified ChebNet[1] (Vanilla GCNs)



$$Cross\ entropy\ Loss:\ L = -\sum_{l \in Y_L}\sum_{c=1}^{C} Y_{lc}\ln(Z_{lc})$$

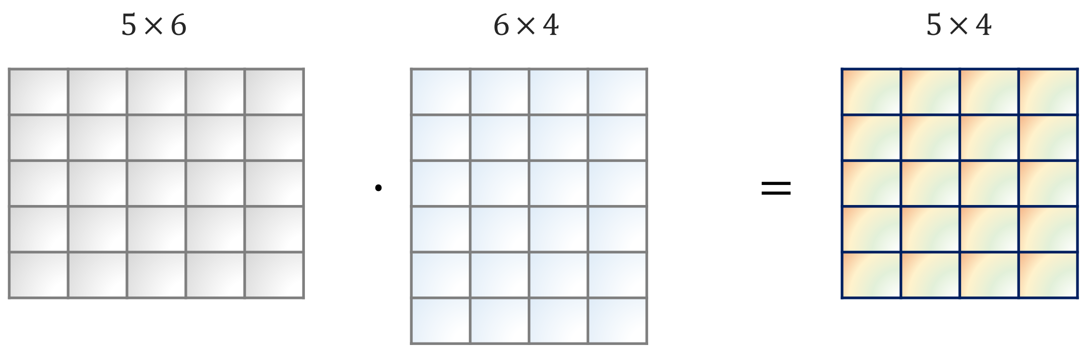
Layer 1

Activation map

$$\boldsymbol{Z = f(X, A) = softmax(\hat{A}Relu(\hat{A}XW^{(0)})W^{(1)})}$$

2-layered neural network structure

- X : Node-feature matrix
- A : Adjacency matrix
- Z : Output
- I : Identity matrix
- D : Degree matrix

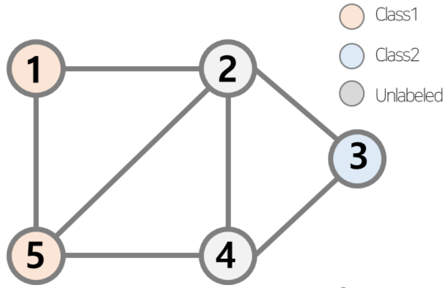


X: Node-Feature matrix W_0: Weight matrix *Activation map*

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

$Renormalization\ trick: \hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}},\ \tilde{A} = (A + I_N)$

Simplified ChebNet[1] (Vanilla GCNs)



Cross entropy Loss: $L = -\sum_{l \in Y_L}\sum_{c=1}^{C} Y_{lc}\ln(Z_{lc})$

Layer 1

Activation map

$$Z = f(X, A) = softmax\left(\hat{A} Relu\left(\hat{A} X W^{(0)}\right) W^{(1)}\right)$$

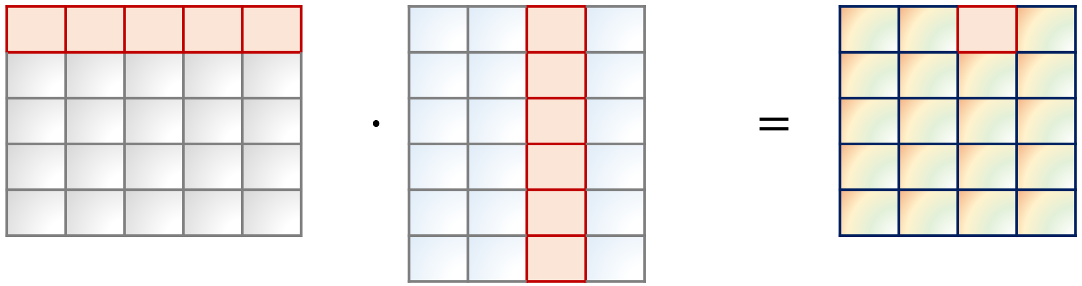
2-layered neural network structure

- X : Node-feature matrix
- A : Adjacency matrix
- Z : Output
- I : Identity matrix
- D : Degree matrix

5×6 $\quad$ 6×4 $\quad$ 5×4

첫번째 node-feature

세번째 filter



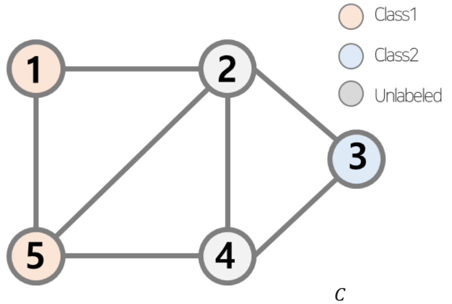
첫번째 node-feature에 대해
세번째 filter를 적용시킨
activation map value

X: Node-Feature matrix $\quad$ W_0: Weight matrix $\quad$ *Activation map*

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

$$Renormalization\ trick:\ \hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}},\ \tilde{A} = (A + I_N)$$

Simplified ChebNet[1] (Vanilla GCNs)



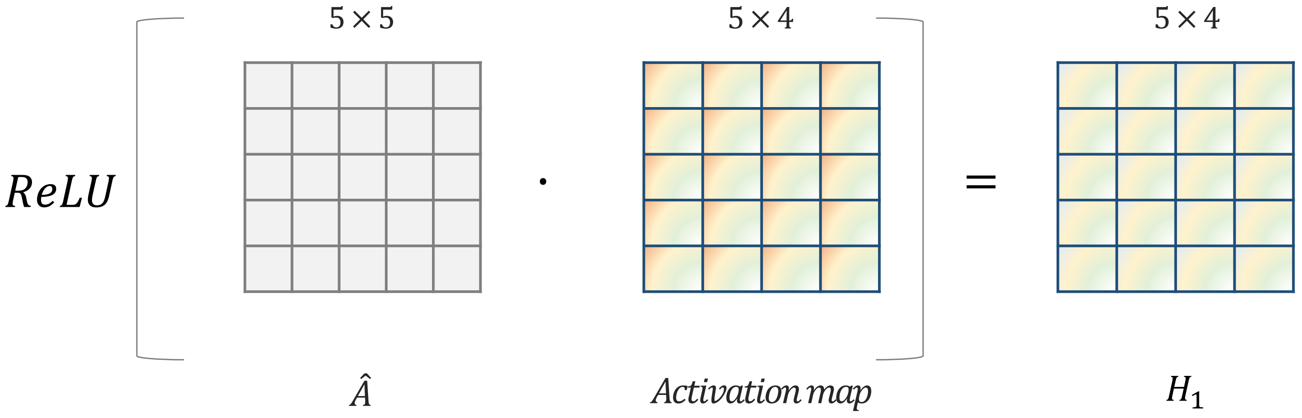
$$Cross\ entropy\ Loss: L = -\sum_{l \in Y_L} \sum_{c=1}^{C} Y_{lc} \ln(Z_{lc})$$

Layer 1

$$\boldsymbol{Z = f(X, A) = softmax(\hat{A}\underbrace{Relu(\hat{A}XW^{(0)})}_{H_1}W^{(1)})}$$

2-layered neural network structure

X : Node-feature matrix
A : Adjacency matrix
Z : Output
I : Identity matrix
D : Degree matrix

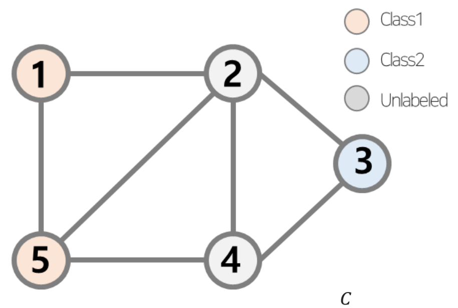


[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

$$Renormalization\ trick:\ \hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}},\ \tilde{A} = (A + I_N)$$

Simplified ChebNet[1] (Vanilla GCNs)

Layer 2



$$Cross\ entropy\ Loss:\ L = -\sum_{l \in Y_L}\sum_{c=1}^{C} Y_{lc}\ln(Z_{lc})$$

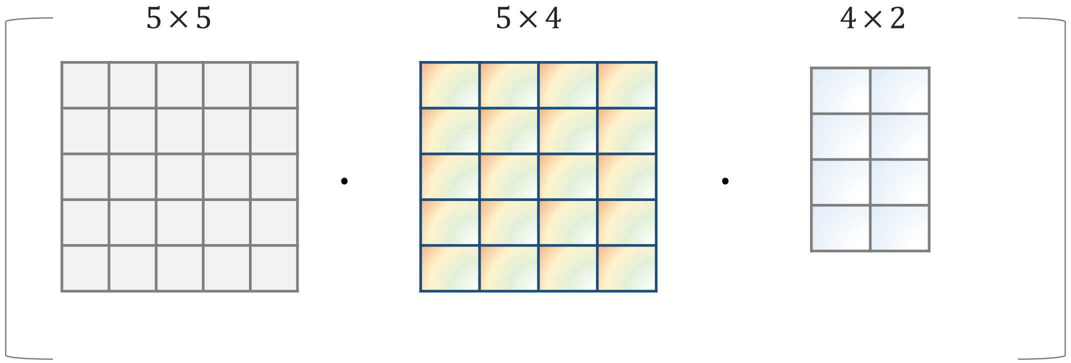
H_2

$$\boldsymbol{Z = f(X, A) = softmax\left(\hat{A}Relu\left(\hat{A}XW^{(0)}\right)W^{(1)}\right)}$$

2-layered neural network structure

X : Node-feature matrix
A : Adjacency matrix
Z : Output
I : Identity matrix
D : Degree matrix

$softmax$
(for classification)



$\hat{A}$ H_1 W_1: Weight matrix

=

5×2

1	0
0	1
0	1
1	0
1	0

Z : Output

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

$Renormalization\ trick: \hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}},\ \tilde{A} = (A + I_N)$

Simplified ChebNet[1] (Vanilla GCNs)

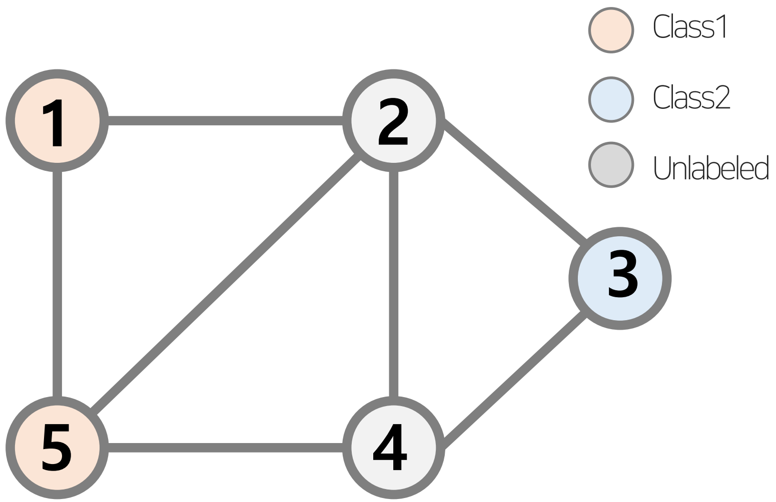
Loss function

$L = -\sum_{l \in Y_L} \sum_{c=1}^{C} Y_{lc} \ln(Z_{lc})$ → "Cross-entropy error over all labeled data"

$$\boldsymbol{Z = f(X, A) = softmax\left(\hat{A}\,Relu\left(\hat{A}XW^{(0)}\right)W^{(1)}\right)}$$

2-layered neural network structure

X : Node-feature matrix
A : Adjacency matrix
Z : Output



$Y_L \in R^{n_L \times c}$

1	0
0	1
1	0

Ground truth

$Z \in R^{n \times c}$

1	0
0	1
0	1
1	0
1	0

Output

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

Simplified ChebNet[1] (Vanilla GCNs)

- Results
  - Dataset:
    - Citation network: 각 node는 문서, edge는 citation link, labeled rate는 총 node 중 training set에 있는 label이 있는 node의 비율
    - NELL: Knowledge graph로 부터 추출된 bipartite graph (이분 그래프), 특정 entity과 그 사이의 relation을 node로 표현

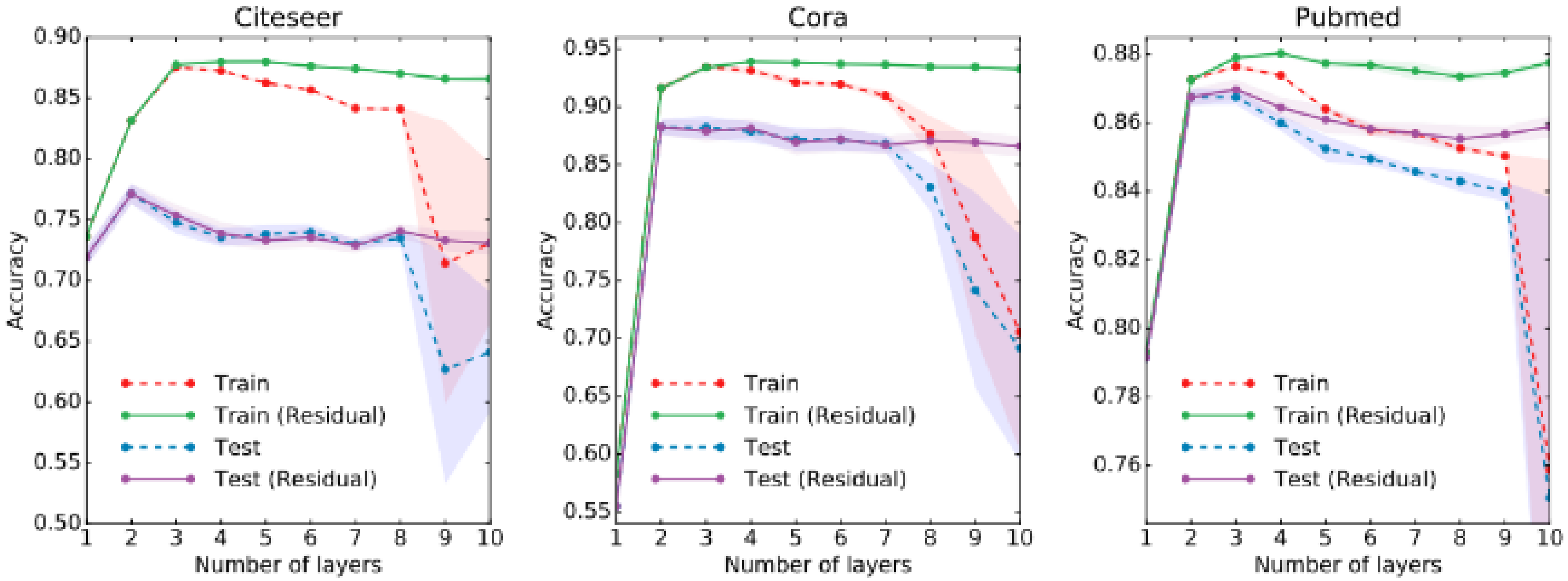
Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

Simplified ChebNet[1] (Vanilla GCNs)

- Results
  - Optimal number of layers:
    - ➢ Convolution layer 수를 1 – 10까지 늘려가며 실험한 결과, 2–3개 layer 사용했을 때 가장 우수한 결과



[1] TN Kipf, M Welling, Semi-supervised classification with graph convolutional networks, 2016

Simplified ChebNet[1] (Vanilla GCNs)

- Limitation
  - Memory requirement:
    - Full-batch gradient descent 방식을 이용하여, dataset 크기가 커질수록 memory 요구량 증가
    - 따라서, mini-batch gradient descent 방법이 필요, K개 layer인 경우 mini-batch 에 포함된 노드들의 K 번째 이웃 노드들 또한 메모리 저장되어야 하는 점을 고려
  - Node feature만 사용, Edge feature (node relation)을 고려하지 않음
    - Undirected graph에만 적용 가능