

# Vision AI

# Understanding Convolutions on Graphs

---

2022-04-13 | JiHyun Lee

Distill

ABOUT PRIZE SUBMIT

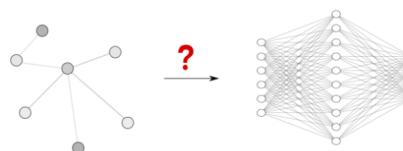
Sept. 2, 2021

PEER-REVIEWED

## Understanding Convolutions on Graphs

Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal

Understanding the building blocks and design choices of graph neural networks.



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

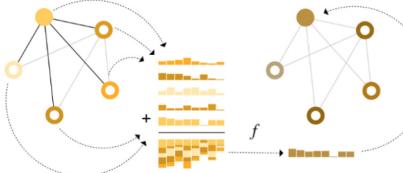
Sept. 2, 2021

PEER-REVIEWED

## A Gentle Introduction to Graph Neural Networks

Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko

What components are needed for building learning algorithms that leverage the structure and properties of graphs?



22/01/19 저널 클럽 발표

This article is one of two Distill publications about graph neural networks.

Take a look at [Understanding Convolutions on Graphs](#) to understand how convolutions over images generalize naturally to convolutions over graphs.

1. 그래프 개념
2. The Challenges of Computation on Graphs
3. Problem Setting and Notation
4. Convolution on Graph
  - Extending Convolutions to Graphs

## Graph란

- A graph represents the relations (**edges**) between a collection of entities (**nodes**).

## Vertex (or node) attributes



- V** Vertex (or node) attributes  
e.g., node identity, number of neighbors
- E** Edge (or link) attributes and directions  
e.g., edge identity, edge weight
- U** Global (or master node) attributes  
e.g., number of nodes, longest path

Three types of attributes we might find in a graph. Hover over to highlight each attribute. Other types of graphs and attributes are explored in the [Other types of graphs](#) section.

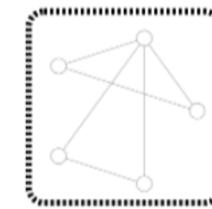
## Edge (or link) attributes and directions



- V** Vertex (or node) attributes  
e.g., node identity, number of neighbors
- E** Edge (or link) attributes and directions  
e.g., edge identity, edge weight
- U** Global (or master node) attributes  
e.g., number of nodes, longest path

Three types of attributes we might find in a graph. Hover over to highlight each attribute. Other types of graphs and attributes are explored in the [Other types of graphs](#) section.

## Global (or master node) attributes



- V** Vertex (or node) attributes  
e.g., node identity, number of neighbors
- E** Edge (or link) attributes and directions  
e.g., edge identity, edge weight
- U** Global (or master node) attributes  
e.g., number of nodes, longest path

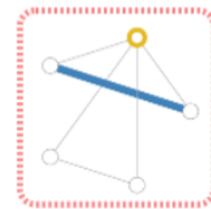
Three types of attributes we might find in a graph. Hover over to highlight each attribute. Other types of graphs and attributes are explored in the [Other types of graphs](#) section.

- Each node, edge, entire graph can store information in each of these pieces of the graph



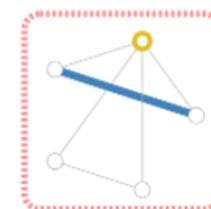
- Vertex (or node) embedding
- Edge (or link) attributes and embedding
- Global (or master node) embedding

Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).



- Vertex (or node) embedding
- Edge (or link) attributes and embedding
- Global (or master node) embedding

Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).



- Vertex (or node) embedding
- Edge (or link) attributes and embedding
- Global (or master node) embedding

Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).

## Passing messages between parts of the graph

Message passing works in three steps :

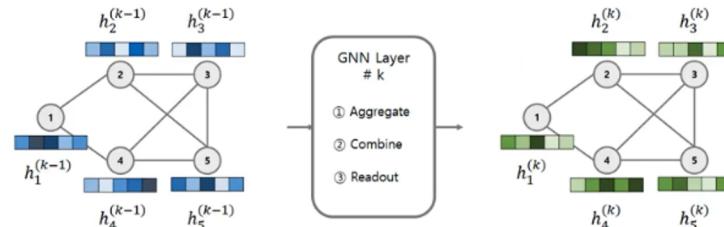
1. [gather messages] 그래프의 각각의 노드는 g function 을 통하여, 이웃된 노드 embeddings 을 gather
2. [aggregate them] Aggregate function (e.g. sum) 을 통하여, 모든 messages Aggregate
3. [update them] All pooled messages 는 update function 을 통과 (일반적으로 학습된 neural network).

- GNN Notation

- $h_v^{(k)}$ : hidden embedding node v at kth GNN layer
- $v = \text{target node}$
- $N(v) = \text{neighbor nodes of } v$
- $u = \text{neighbor node} \in N(v)$

$$a_v^{(k-1)} = \text{aggregate}^k(\{h_u^{(k-1)}, \forall u \in N(v)\})$$

$$h_v^{(k)} = \text{combine}^k(a_u^{(k-1)}, h_v^{(k-1)})$$



$$\mathbf{h}_v^0 = \mathbf{x}_v$$

$$\mathbf{h}_v^k = \sigma \left( \mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1} \right), \quad \forall k \in \{1, \dots, K\}$$

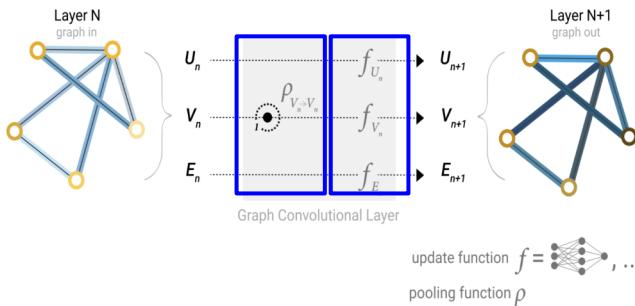
$$\mathbf{z}_v = \mathbf{h}_v^K$$

trainable matrices  
(i.e., what we learn)

After K-layers of neighborhood aggregation, we get output embeddings for each node.

Message passing works in three steps :

1. [gather messages]
2. [aggregate them]
3. [update them]



Schematic for a GCN architecture, which updates node representations of a graph by pooling neighboring nodes at a distance of one degree.

aggregation function used in pooling: max, mean or sum

- 

## GNN

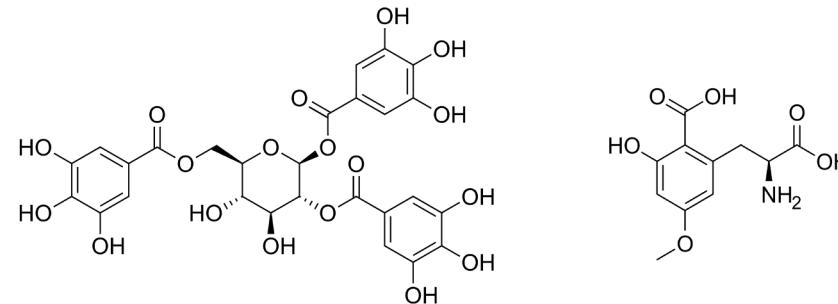
- Aggregate, Combine function의 정의에 따라 다양한 방식의 모델이 존재함.

TABLE 2  
Different variants of graph neural networks.

Name	Variant	Aggregator	Updater
Spectral Methods	ChebNet	$\mathbf{N}_k = \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{X}$	$\mathbf{H} = \sum_{k=0}^K \mathbf{N}_k \Theta_k$
	1 <sup>st</sup> -order model	$\mathbf{N}_0 = \mathbf{X}$ $\mathbf{N}_1 = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{X}$	$\mathbf{H} = \mathbf{N}_0 \Theta_0 + \mathbf{N}_1 \Theta_1$
	Single parameter	$\mathbf{N} = (\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}) \mathbf{X}$	$\mathbf{H} = \mathbf{N} \Theta$
	GCN	$\mathbf{N} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}$	$\mathbf{H} = \mathbf{N} \Theta$
Non-spectral Methods	Neural FPs	$\mathbf{h}_{\mathcal{N}_v}^t = \mathbf{h}_v^{t-1} + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{h}_v^t = \sigma(\mathbf{h}_{\mathcal{N}_v}^t, \mathbf{W}_L^{\mathcal{N}_v})$
	DCNN	Node classification: $\mathbf{N} = \mathbf{P}^* \mathbf{X}$ Graph classification: $\mathbf{N} = \mathbf{l}_N^1 \mathbf{P}^* \mathbf{X} / N$	$\mathbf{H} = f(\mathbf{W}^c \odot \mathbf{N})$
	GraphSAGE	$\mathbf{h}_{\mathcal{N}_v}^t = \text{AGGREGATE}_t(\{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\})$	$\mathbf{h}_v^t = \sigma(\mathbf{W}^t \cdot [\mathbf{h}_v^{t-1} \  \mathbf{h}_{\mathcal{N}_v}^t])$
Graph Attention Networks	GAT	$\alpha_{vk} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T [\mathbf{W}\mathbf{h}_v \  \mathbf{W}\mathbf{h}_k]))}{\sum_{j \in \mathcal{N}_v} \exp(\text{LeakyReLU}(\mathbf{a}^T [\mathbf{W}\mathbf{h}_v \  \mathbf{W}\mathbf{h}_j]))}$ $\mathbf{h}_{\mathcal{N}_v}^t = \sigma(\sum_{k \in \mathcal{N}_v} \alpha_{vk} \mathbf{W}\mathbf{h}_k)$ Multi-head concatenation: $\mathbf{h}_{\mathcal{N}_v}^t = \sum_{m=1}^M \sigma(\sum_{k \in \mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k)$ Multi-head average: $\mathbf{h}_{\mathcal{N}_v}^t = \sigma\left(\frac{1}{M} \sum_{m=1}^M \sum_{k \in \mathcal{N}_v} \alpha_{vk}^m \mathbf{W}^m \mathbf{h}_k\right)$	$\mathbf{h}_v^t = \mathbf{h}_{\mathcal{N}_v}^t$
	GGNN	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{h}_k^{t-1} + \mathbf{b}$	$\mathbf{z}_v^t = \sigma(\mathbf{W}^z \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^z \mathbf{h}_v^{t-1})$ $\mathbf{r}_v^t = \sigma(\mathbf{W}^r \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^r \mathbf{h}_v^{t-1})$ $\mathbf{h}_v^t = \tanh(\mathbf{W}^h \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{U}^h (\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}))$ $\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \mathbf{h}_v^t$
	Tree LSTM (Child sum)	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i \mathbf{x}_v^t + \mathbf{U}^i \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f \mathbf{x}_v^t + \mathbf{U}^f \mathbf{h}_{vk}^{t-1} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o \mathbf{x}_v^t + \mathbf{U}^o \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u \mathbf{x}_v^t + \mathbf{U}^u \mathbf{h}_{\mathcal{N}_v}^t + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_{k-1}^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
	Tree LSTM (N-ary)	$\mathbf{h}_{\mathcal{N}_v}^{t_1} = \sum_{k=1}^K \mathbf{U}_{1k}^t \mathbf{h}_{vk}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{t_f} = \sum_{k=1}^K \mathbf{U}_{fk}^t \mathbf{h}_{vk}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{t_0} = \sum_{k=1}^K \mathbf{U}_{0k}^t \mathbf{h}_{vk}^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{t_u} = \sum_{l=1}^K \mathbf{U}_l^t \mathbf{h}_{vl}^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{t_1} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{t_f} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{t_0} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{t_u} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{l=1}^K \mathbf{f}_{vl}^t \odot \mathbf{c}_{el}^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$
Graph LSTM in [44]	Graph LSTM in [44]	$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^t \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^t = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^u \mathbf{h}_k^{t-1}$ $\mathbf{h}_{\mathcal{N}_v}^{tu} = \sum_{k \in \mathcal{N}_v} \mathbf{U}_{m(v,k)}^u \mathbf{h}_k^{t-1}$	$\mathbf{i}_v^t = \sigma(\mathbf{W}^i \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{t_1} + \mathbf{b}^i)$ $\mathbf{f}_{vk}^t = \sigma(\mathbf{W}^f \mathbf{x}_v^t + \mathbf{U}_{m(v,k)}^f \mathbf{h}_k^{t-1} + \mathbf{b}^f)$ $\mathbf{o}_v^t = \sigma(\mathbf{W}^o \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{t_0} + \mathbf{b}^o)$ $\mathbf{u}_v^t = \tanh(\mathbf{W}^u \mathbf{x}_v^t + \mathbf{h}_{\mathcal{N}_v}^{tu} + \mathbf{b}^u)$ $\mathbf{c}_v^t = \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}$ $\mathbf{h}_v^t = \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)$

## 1. Lack of Consistent Structure

- 그라프는 굉장히 유연한 구조를 가지고 있고, 이는 곧 인스턴스 간에 일관된 구조 결여로 이어짐.



Left: A **non-toxic** 1,2,6-trigalloyl-glucose molecule.

Right: A **toxic** caramboxin molecule.

주어진 두 개의 분자 구조가 독성인지 아닌지 예측!

위 예시에서 다음과 같은 문제 존재

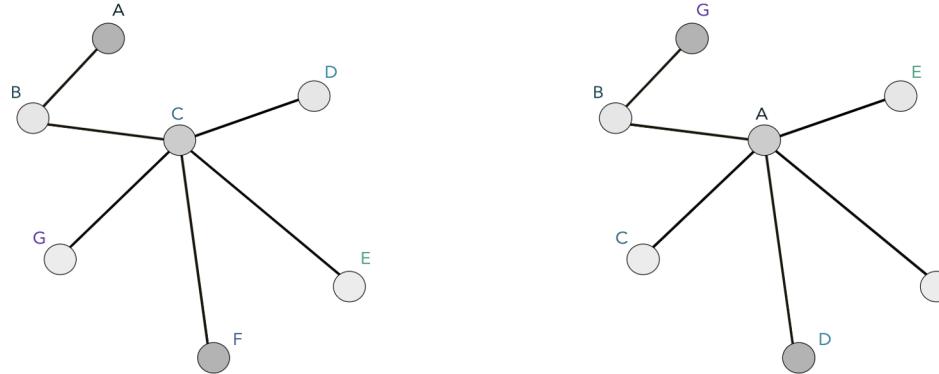
- 분자는 서로 다른 수의 원자를 가질 수 있다.
- 분자의 원자는 다른 종류일 수 있다.
- 각각의 원자들은 서로 다른 connection 을 가질 수 있다.

그래프를 계산할 수 있는 format 으로 변경하는 것은 중요한 문제이며, final representation 은 actual problem 에 큰 영향을 미친다.

## 2.

### Node-Order Equivariance

- 그래프는 노드 간 내재된 순서 (inherent ordering) 가 없다.
- 그래프를 1-d vector 로 표현하기 위해서는 (graph representing) 노드의 순서를 고정 할 필요가 있는데, 노드의 내재된 순서가 없다면? 아래 그림과 같이 동일한 그래프라도 다르게 표현될 수 있음.



Representing the graph as one vector requires us to fix an order on the nodes. But what do we do when the nodes have no inherent order?

Above: The same graph labelled in two different ways. The alphabets indicate the ordering of the nodes.

---

- 우리의 그래프는 **node-order equivariant** 하다. 즉, 그래프 노드의 순서에 의존해서는 안된다. 만약에 그래프 노드 순서를 바꾼다고 해도, 그 래프 representation 은 동일하게 나와야 한다.

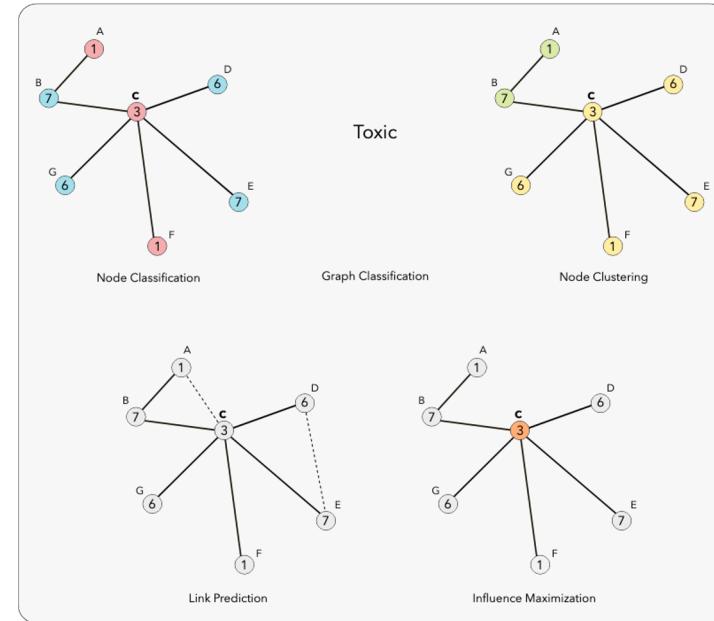
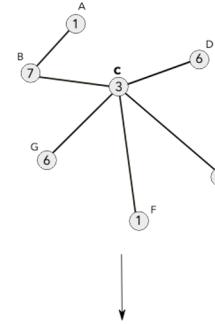
## 3. Scalability

- 확장성, 사용자 수의 증대에 유연하게 대응할 수 있는 정도
- 대부분의 natural 그래프들은 sparse 함 : 그래프 내의 노드 표현을 효율적으로 할 수 있어야 한다.
- 추가적으로, 그래프 자체의 크기에 비해 적은 파라미터를 필요로 함.

# Problem Setting and Notation

그래프를 통해 해결할 수 있는 task

- Node Classification : 각각의 노드 분류
- Graph Classification : 전체 그래프 분류
- **Node Clustering** : connectivity에 의거하여 비슷한 노드 간 그룹핑
- Link Prediction : missing 링크 예측
- Influence Maximization : 영향력 있는 노드 특정



# Convolution on Graph

## Convolution on Graph

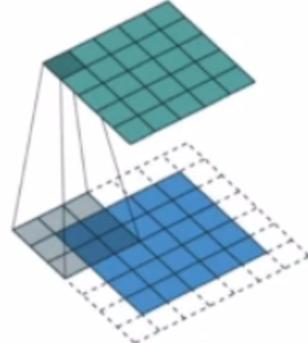
### Convolution 이란?

- 두 함수의 합성곱
- “Convolution Neural Network (CNN)”에서는, 이미지에 대하여 Filter 를 통해 정보들을 Aggregate 해 나가는 과정을 의미.

$$f * g = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau$$

- “ $f, g$  가운데 하나의 함수를 반전, 전이 시킨 다음, 다른 하나의 함수와 곱한 결과를 적분.”
- 전자 공학에서 convolution 은 linear time invariant system 상에서 이전값과 현재값을 연산하기 위해 주로 사용했던 연산.
- 현재의 합성곱의 값은 이전 시간의 결과를 포함한다.

### Convolution

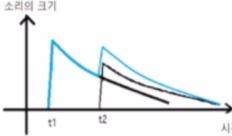
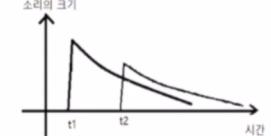
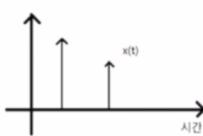
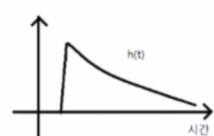


두 함수의 합성곱이라는 뜻은, ‘이미지’에서만 사용되는 개념이 아니다!



종소리 예시

종을 연속적으로 치는 경우,  $t$ 시점의 소리는 이전 소리에 중첩되어 나타난다.



## Convolution on Graph

### Convolution 이란?

- 두 함수의 합성곱

두 함수를 어떻게 곱해서 합하나요?

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau$$

두 연속함수  $f, g$ 를 convolution 하는 식입니다.

이해하셨나요?

먼저, 합성곱을 위해서는 두 함수중 하나를 반전 (reverse)시켜야 합니다.

위의 식을 보면 연속함수  $g$ 의 변수 타우( $\tau$ )앞쪽에 마이너스가 붙어있는게 보입니다.  
즉 함수  $g$ 를 반전(reverse) 시켰단 것을 알 수 있네요.

다음으로, 반전시킨 함수를 전이 (shift) 시켜야 합니다.

마찬가지로 함수  $g$  를  $t$  만큼 이동 시켰단 것을 알 수 있네요.

마지막으로 이동시킨 함수  $g$ 를 함수  $f$ 와 곱한결과를 하나씩 기록합니다.

이때 변수 타우( $\tau$ )를 변화시키며 결과를 쭉 기록하는 것을 convolution 이라고 합니다!

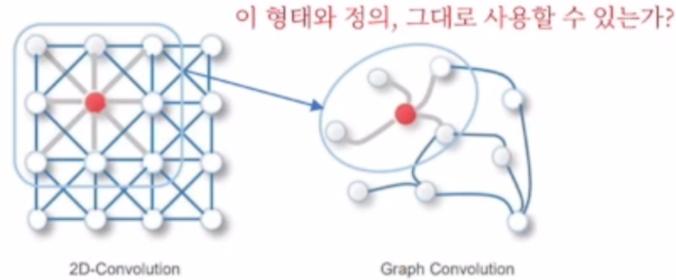
어떨때는 두 함수가 겹치게 되고, 또 어떨때는 겹치는게 없어서 합쳐도 0이 되기도 합니다.

# Convolution on Graph

## Convolution on Graph

### Graph Convolution

- Graph 에 Convolution Filter 를 적용하여, 하나의 노드의 인접 노드와의 관계를 계산하고 싶다.
- Convolution 은 전체 데이터에서 Local Feature 를 뽑아내는데 유용하다.
- Filter 들이 Spatial location 에 따라서 변하지 않는다.



이미지도 격자(Grid)형태 Graph로 나타낼 수 있다

그럼 Graph에도 직접적으로 Convolution을 사용할 수 있는가?

### 일반적 Convolution 사용 불가 이유

- Convolution 은 Regular Grid 에 정의된다.
- Filter 가 모든 데이터에 대하여 같은 크기가 아니다.
- Graph 는 **spatial** domain 이 아니다.

### Convolution Theorem

$$f * g(x) = \mathcal{F}^{-1}\{\mathcal{F}(f)\mathcal{F}(g)\}$$

# Convolution on Graph

## \* Convolution Theorem

$$f * g(x) = \mathcal{F}^{-1}\{\mathcal{F}(f)\mathcal{F}(g)\}$$

$\mathcal{L}\{f(t) * g(t)\}$

$= \int_0^\infty e^{-st} \left( \int_0^\infty f(t-v) g(v) dv \right) dt$

$= \int_0^\infty e^{-st} \left( \int_0^\infty f(t-v) g(v) u(t-v) dv \right) dt$

$= \int_0^\infty \left( \int_0^\infty e^{-st} f(t-v) g(v) u(t-v) dv \right) dt$

$= \int_0^\infty \left( \int_0^\infty e^{-st} f(t-v) g(v) u(t-v) dt \right) dv$

$= \int_0^\infty g(v) \left( \int_0^\infty e^{-st} f(t-v) u(t-v) dt \right) dv$

note:  $\mathcal{L}\{f(t-v)u(t-v)\} = e^{-vs} \mathcal{L}\{f(t)\} = e^{-sv} F(s)$

$= \int_0^\infty g(v) e^{-sv} F(s) dv$

$= F(s) \cdot \int_0^\infty e^{-sv} g(v) dv$

$= F(s) \cdot G(s)$

$\boxed{\mathcal{L}\{f(t) * g(t)\} = \mathcal{L}\{f(t)\} \cdot \mathcal{L}\{g(t)\}}$

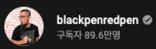
$\boxed{\mathcal{L}^{-1}\{F(s)G(s)\} = \mathcal{L}^{-1}\{F(s)\} * \mathcal{L}^{-1}\{G(s)\}}$

▶ 17:53 / 18:09

Proof of the Convolution Theorem

조회수 86,852회 • 2017. 4. 29.

1.3천 실어요 공유 THANKS 클립 저장



blackpenredpen

구독자 89.6만명

가입

구독

HUMAN TOUCH IN MEDICINE

# Convolution on Graph

## \* Convolution Theorem

$$f * g(x) = \mathcal{F}^{-1}\{\mathcal{F}(f)\mathcal{F}(g)\}$$

- 현 domain 의 convolution 은 다른 domain 의 point-wise multiplication 과 같다.
- Graph domain 의 convolution 은 fourier domain 의 point-wise multiplication 과 같다.
- Convolution 의 laplace 변환은 point-wise multiplication 과 같다.

즉,  $g(x)$  라는 필터를  $f$  란 데이터에 convolution 할 때,  
fourier transform 을 한 다음에 곱하고, 다시 inverse 를 하면 convolution 을 한 것과 동일하다.

$$L = U\Lambda U^T$$

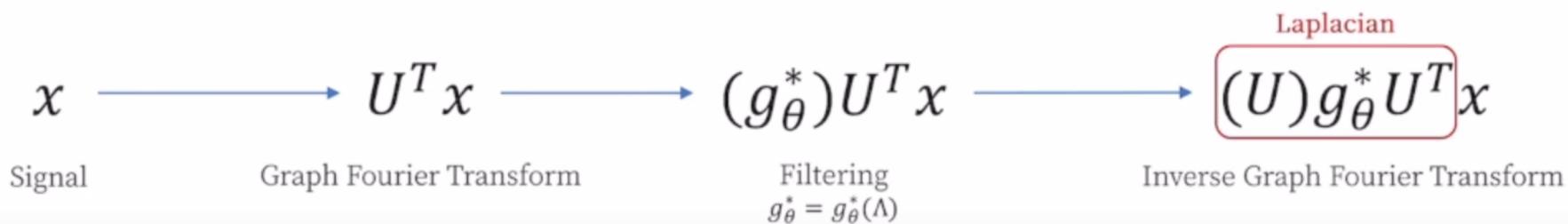
*Eigenvector*      *Eigenvalue*

## Graph Fourier/Inverse Fourier Transform

$$\mathcal{F}(\mathbf{x}) = \mathbf{U}^T \mathbf{x}$$
$$\mathcal{F}^{-1}(\mathbf{x}) = \mathbf{U} \mathbf{x}$$

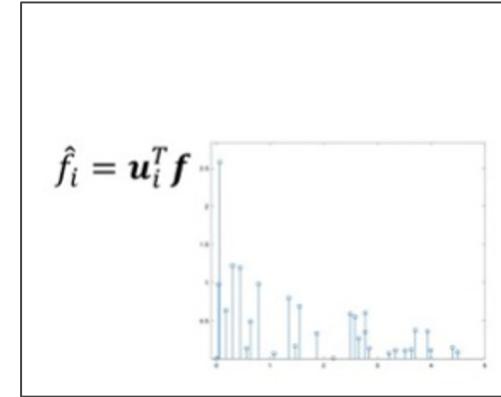
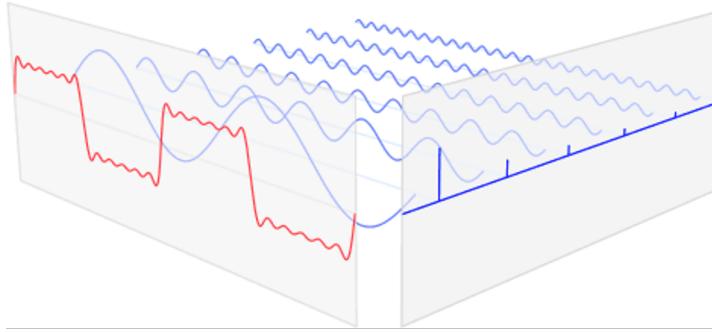
=> Graph domain 에 적용

=> Graph 를 fourier domain 으로 바꿔서 Neural Network 처럼 convolution 을 하자!



# Convolution on Graph

## \* Fourier Transform



- 붉은색 신호 : 입력 신호
- 파란색 신호 : 푸리에 변환을 통해 얻어진 (원본 신호를 구성하는) 주기 함수 성분들.
- 각각의 주기함수 성분들은 고유의 주파수 (frequency) 와 강도 (amplitude) 를 가지고 있으며 이들을 모두 합치면 원본 붉은색 신호가 됨.

통신 분야

- time domain에서 frequency domain으로의 변환이라고 하고,

컴퓨터 비전, 영상 처리

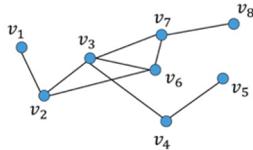
- spatial domain에서 frequency domain으로의 변환이라고 함.

푸리에 변환은 입력 신호가 어떤 신호이든지 관계없이 임의의 입력 신호를  $\sin$ ,  $\cos$  주기함수들의 합으로 분해. 그리고 그 과정을 수식으로 표현한 것이 푸리에 변환식.

# Convolution on Graph

## Laplacian Matrix

- 두 가지 관계를 하나의 Matrix로 규합한 것
  - 1) Degree Matrix: 하나의 노드가 다른 '몇 개의 노드와 연결되어 있는가? (대각행렬)
  - 2) Adjacency Matrix: 하나의 노드가 다른 '어떤 노드와 연결되어 있는가? (비대각행렬)



Adjacency Matrix:  $A[i, j] = 1$  if  $v_i$  is adjacent to  $v_j$   
 $A[i, j] = 0$ , otherwise

Degree Matrix:  $D = \text{diag}(\text{degree}(v_1), \dots, \text{degree}(v_N))$

$$\begin{array}{c} \text{Degree Matrix} \\ \left( \begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right) \end{array} - \begin{array}{c} \text{Adjacency Matrix} \\ \left( \begin{array}{ccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right) \end{array} = \begin{array}{c} \text{Laplacian Matrix} \\ \left( \begin{array}{ccccccc} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{array} \right) \end{array}$$

- 대각 행렬과 비대각 행렬의 '차'이지만, 두 행렬의 차가 값을 뺀다는 의미는 없고, 두 행렬을 하나의 행렬로 합친다는 의미가 있다.

노드의 value 를 나타내는 벡터  $f$  가 Laplacian matrix 와 곱해진다는 뜻 : 하나의 노드와 이웃 노드의 값 차이를 살펴보

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

- 중심 노드와 이웃 노드 간의 값 차이를 한 번에 계산하겠다는 것.

$$L = U \Lambda U^T$$

*Eigenvector*      *Eigenvalue*

# Convolution on Graph

## Laplacian Matrix

노드의 value 를 나타내는 벡터  $f$  가 laplacian matrix 와 곱해진다는 뜻 : 하나의 노드와 이웃 노드의 값 차이를 살펴보겠다는 것

$$h = Lf = (D - A)f = Df - Af$$

$$\begin{aligned} h_i &= L_i f = (D_i - A_i)f = D_i f - A_i f \\ &= D_{i,i}f_i - A_{i,i}f \end{aligned}$$

$$h_i = \sum_{v_j \in \mathcal{N}(v_i)} (f_i - f_j)$$

Laplacian quadratic form:

$$h = Lf, \quad h_i = \sum_{v_j \in \mathcal{N}(v_i)} (f_i - f_j)$$

$$f^T L f = \sum_i f_i h_i = \sum_i \sum_{v_j \in \mathcal{N}(v_i)} f_i (f_i - f_j)$$

$$= \frac{1}{2} \sum_{i,j} A(i,j) f_i (f_i - f_j) + \frac{1}{2} \sum_{i,j} A(j,i) f_i (f_i - f_j)$$

$$= \frac{1}{2} \sum_{i,j} A(i,j) f_i (f_i - f_j) - \frac{1}{2} \sum_{i,j} A(i,j) f_j (f_i - f_j)$$

$$= \frac{1}{2} \sum_{i,j} A(i,j) (f_i - f_j)^2$$

⇒ The smaller, the similar the connected nodes

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

- Laplacian matrix  $L$ 에 graph signal  $f$  를 곱하면, difference operator 로서 의미를 갖게 됨.
- 중심 노드와 이웃 노드 간의 값 차이를 한 번에 계산하는 것.
- (빨간 박스) : 노드  $i$  와 노드  $j$  사이의 연결이 존재할 때,  $f_i - f_j$  의 값이 작으면 연결된 노드의 signal 이 유사하다는 의미



- 그러나 이러한 경우, 단순한 차이를 합한다는 데 한계가 있음. 절댓값 또는 제곱을 취하지 않은 채 difference 값들의 단순 합을 통해 부호가 상쇄되므로,  $h$  값이 0에 가깝다고 중심 노드와 이웃 노드 간의 유사도가 크다고 하기 어렵다.



- 따라서 왼쪽과 같이 laplacian quadratic form 정의
- 단순히  $L$ 과  $f$  를 곱하는 대신,  $f$  를 두 번 곱해 제곱 형태로 만들어, node feature 간 difference 를 제곱합 한 값을 의미함.
- 결과적으로 quadratic 값이 작을수록 neighbor node 와 유사하다는 의미.

# Convolution on Graph

## Laplacian Matrix

Laplacian **quadratic** form:

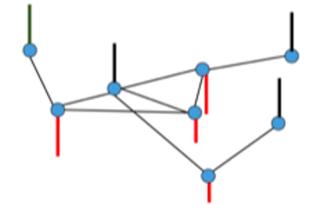
$$f^T L f = \frac{1}{2} \sum_{i,j} A(i,j) (f_i - f_j)^2$$

- The smaller, the similar the connected nodes.  
Can be a cost function for link prediction.
- It can represent  
“Smoothness” or “Frequency” of the signal  $f$
- Spectral Graph Theory

Y. Choi, SNU



Low frequency graph signal



High frequency graph signal

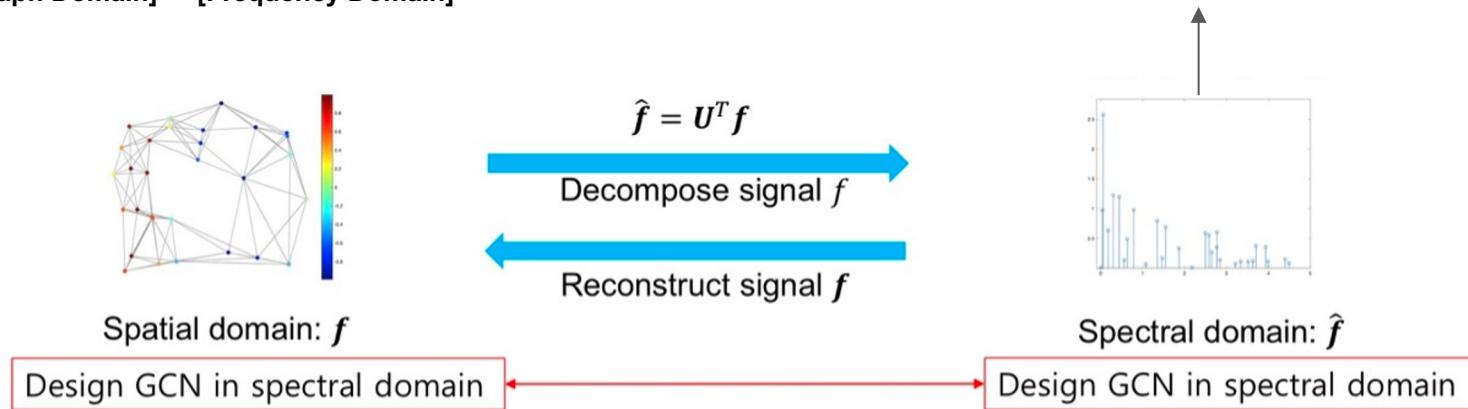
- Laplacian quadratic form 은 결과적으로 signal  $f$  (그래프) 의 smoothness, frequency 를 의미하며,
- signal  $f$  를 이러한 frequency 의 집합으로 나타내어 연산하고자 하는 것이 spectral graph theory 이다.
- 그림의 오른쪽에서 볼 수 있듯이 quadratic form 값이 크면 graph signal 간의 difference 가 큰 것이므로, high frequency 를 가졌다고 볼 수 있고, quadratic 값이 작으면 low frequency 를 가졌다고 볼 수 있다.

# Convolution on Graph

## Graph Fourier Transform

- Graph Signal 을 Frequency 별로 분해하는 과정
  - Graph Signal : Graph 노드의 feature
  - Frequency : feature 들 간의 차이
- [Graph Domain]  $\rightarrow$  [Frequency Domain]

Low frequency : 노드 feature 간 차이가 적음 = 두 feature 가 유사할 가능성이 높다.  
High frequency : 노드 feature 간 차이가 큼 = 두 feature 가 서로 다른 특징의 정보를 많이 담고 있다.



[The emerging field of signal processing on graphs](#): Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 2013 (1867회 인용)

$$L = U \Lambda U^T$$

Eigenvector

Eigenvalue

## Graph Fourier Transform

- 1) laplacian quadratic form 0| graph signal 의 smoothness 를 의미하며,
- 2) laplacian matrix L eigenvalue 가 L eigenvector 의 smoothness 를 의미함.

**Graph Fourier Transform 은 matrix L 을 eigen decomposition 하여 eigenvector 들의 합으로 signal f (그래프) 를 표현하는 것.**

즉, graph signal 은 기초인 spatial 고가에서 frequency 를 나타내는 spectral 고가으로 변환하는 것

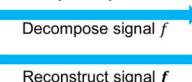
A signal  $f$  can be written as graph Fourier series:

$$f = \sum_i \hat{f}_i u_i \quad f^T u_k = \sum_i \hat{f}_i u_i^T u_k = \hat{f}_k = u_k^T f \quad \hat{f} = \begin{bmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_N \end{bmatrix} = \begin{bmatrix} u_1^T \\ \vdots \\ u_N^T \end{bmatrix} f$$



Spatial domain:  $f$

$$\hat{f} = U^T f$$



Spectral domain:  $\hat{f}$

Design GCN in spectral domain

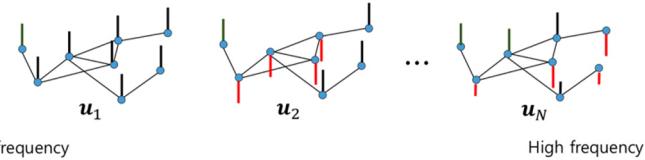
Design GCN in spectral domain

The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE signal processing magazine, 2013 (1867회 인용)

The **frequency** of an eigenvector of Laplacian matrix is its corresponding **eigenvalue**:

$$u_i^T L u_i = u_i^T \lambda_i u_i = \lambda_i \quad \leftarrow \text{Laplacian quadratic form: meaning?}$$

Frequency of the signal  $u_i$



Low frequency

High frequency

$$u_1^T L u_1 = \lambda_1 = 0$$

$$u_1^T L u_1 = \lambda_1$$

...

$$u_N^T L u_N = \lambda_N$$

Laplacian matrix L 을 eigen decomposition 하면 N개의 eigenvector 와 eigenvalue 를 얻을 수 있고,  
각각의 eigenvalue 는 각각 eigenvector signal  $u$  들의 smoothness 를 의미. 오른쪽 그림에서는 eigenvalue 를 작은 것부터  $\lambda_1, \lambda_2, \dots$  로 나열한 것.

작은 eigenvalue 를 갖는 eigenvector 의 경우, graph 를 더욱 smooth하게 나누는 기준이 되며, 노드 간 유사한 특징 정보를 담고 있다.  
반대로, 큰 eigenvalue 를 갖는 eigenvector 의 경우 노드 간 서로 다른 특징 정보를 많이 담고 있다.

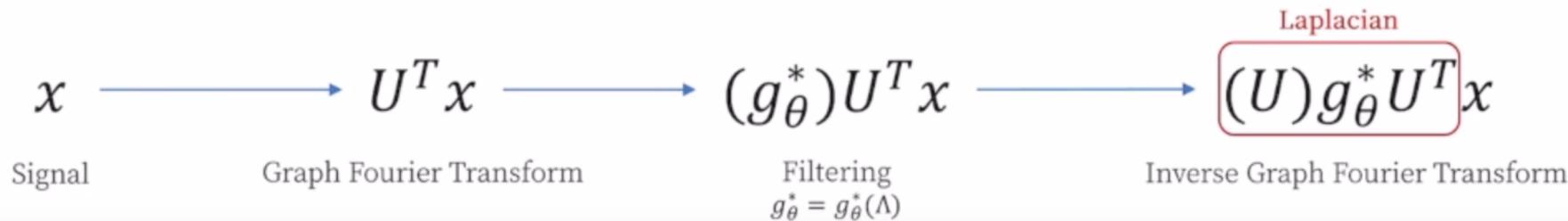
원하는 관계만

GFT → Filtering → IGFT = Laplacian Matrix를 곱하는 것과 같다.

- GFT : 어떤 형태의 그래프 관계가 Signal 에 ‘어느 정도’ 포함되어 있는지 알 수 있다.
- 원하는 관계 (낮은 frequency) 만 필터링 해서,
  - 낮은 Frequency = node feature 간 유사도가 높은 정보를 가진 frequency
  - Frequency 의 spectrum 에서 원하는 frequency 를 filtering
- 다시 복원 (IGFT)
  - Convolution 이 완성되면서, 분석이 용이한 관계를 담을 수 있다.

=&gt; Graph domain 에 적용

=&gt; Graph 를 fourier domain 으로 바꿔서 Neural Network 처럼 convolution 을 하자!



## GCN: Graph Filtering

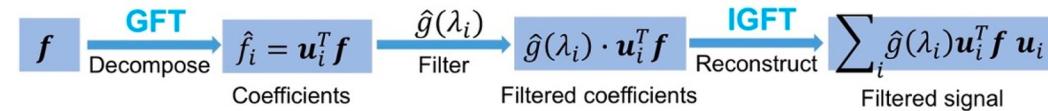
Spectral filtering

**Recall:** a signal  $f$  can be written as graph Fourier series:

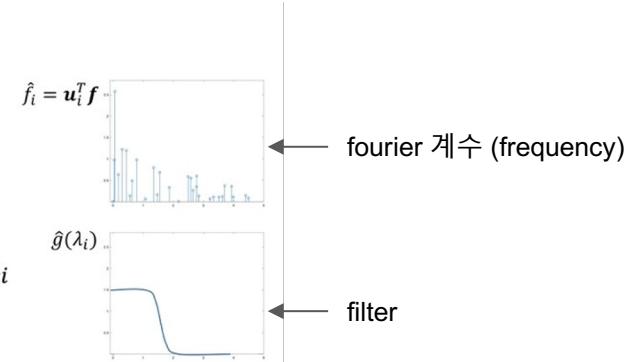
$$\text{GFT: } \hat{f} = U^T f, \hat{f}_i = u_i^T f$$

$$\text{IGFT: } f = U \hat{f} = \sum_i \hat{f}_i u_i$$

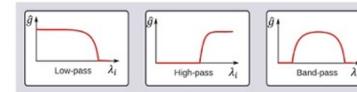
Filter a graph signal  $f$ :



Filter  $\hat{g}(\lambda_i)$ : Modulating the frequency



Example:



Spatial domain에 있던 Signal 정보를 spectral domain으로 보내는 이유?

- Graph signal을 frequency의 조합인 spectral domain에서 표현한 후, 다시 spatial domain으로 복원할 때 사용되는 filter를 학습함으로써, graph signal의 noise를 제거하고 필요한 성분만을 남겨 node를 embedding하기 위함.

# Convolution on Graph (\* review)

## \* Convolution Theorem

$$f * g(x) = \mathcal{F}^{-1}\{\mathcal{F}(f)\mathcal{F}(g)\}$$

- 현 domain 의 convolution 은 다른 domain 의 point-wise multiplication 과 같다.
- Graph domain 의 convolution 은 fourier domain 의 point-wise multiplication 과 같다.
- Convolution 의 laplace 변환은 point-wise multiplication 과 같다.

즉,  $g(x)$  라는 필터를  $f$  란 데이터에 convolution 할 때,  
fourier transform 을 한 다음에 곱하고, 다시 inverse 를 하면 convolution 을 한 것과 동일하다.

$$L = U\Lambda U^T$$

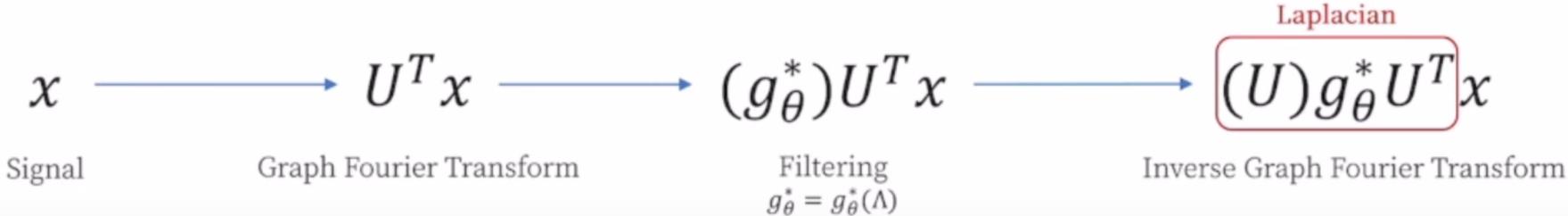
*Eigenvector*      *Eigenvalue*

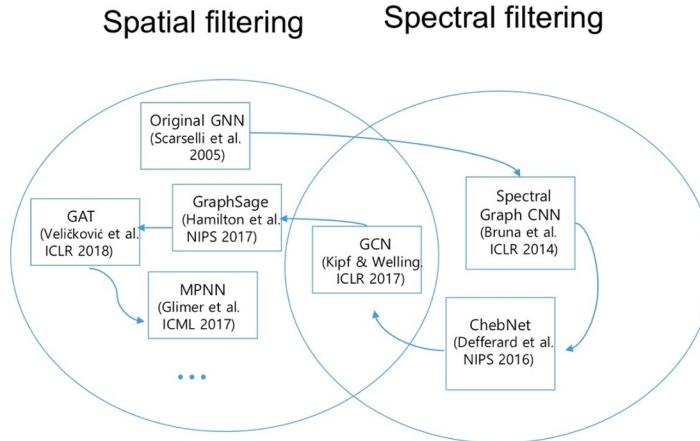
## Graph Fourier/Inverse Fourier Transform

$$\begin{aligned}\mathcal{F}(\mathbf{x}) &= \mathbf{U}^T \mathbf{x} \\ \mathcal{F}^{-1}(\mathbf{x}) &= \mathbf{U} \mathbf{x}\end{aligned}$$

=> Graph domain 에 적용

=> Graph 를 fourier domain 으로 바꿔서 Neural Network 처럼 convolution 을 하자!





- Central Node 를 주위 Neighbor 의 Feature 로 Update
- Spectral 은 계산 비용이 큰데, Spatial 은 간단간단

- 수학적 기반
- Graph Convolution 을 Graph Signal Processing 의 일환으로 보며, Graph Signal 에 대한 Noise 제거 과정으로 간주
- Spectral = Spectrum
- Frequency 의 Spectrum에서 원하는 Frequency 를 필터링

이전의 GNN 모델들은 graph spectral theory 를 바탕으로 graph filtering 해 왔다는 것을 알 수 있음.  
그러나 Kipf & Welling 의 GCN 논문이 ICLR 에 발표되면서, 이후의 모델들은 대부분 spatial domain 에서 편안하게 연산 하기 시작.

GCN 논문은 대략적으로, spectral domain에서 하던 convolution 연산을 spatial domain에서도 매우 간단하게 할 수 있다! 는 내용.

- GCN 의의: spectral convolution 계산으로 spatial convolution Return

# End of the Document