

Seminar Series on Graph Neural Networks 03

A graph signal processing viewpoint of graph neural networks

Yong-Min Shin

School of Mathematics and Computing (Computational Science and Engineering)

Yonsei University

2025.04.14



수학계산학부(계산과학공학)

School of Mathematics and Computing
(Computational Science and Engineering)

Before going in....

Wrap-up: Message passing all the way up
(Up-to-date comprehensive survey on GNN architectures)



Towards application of graph neural networks

Towards efficient graph learning

Explainable graph neural networks



Fundamental topics on graph neural networks

On the representational power of graph
neural networks

A graph signal processing viewpoint of
graph neural networks

On the problem of oversmoothing and
oversquashing



Introduction to graph mining and graph neural networks
(Basic overview to kick things off)

(Some of the topics may change in the future for a better alternative)

* Presentation slides are available at:
jordan7186.github.io/presentations/



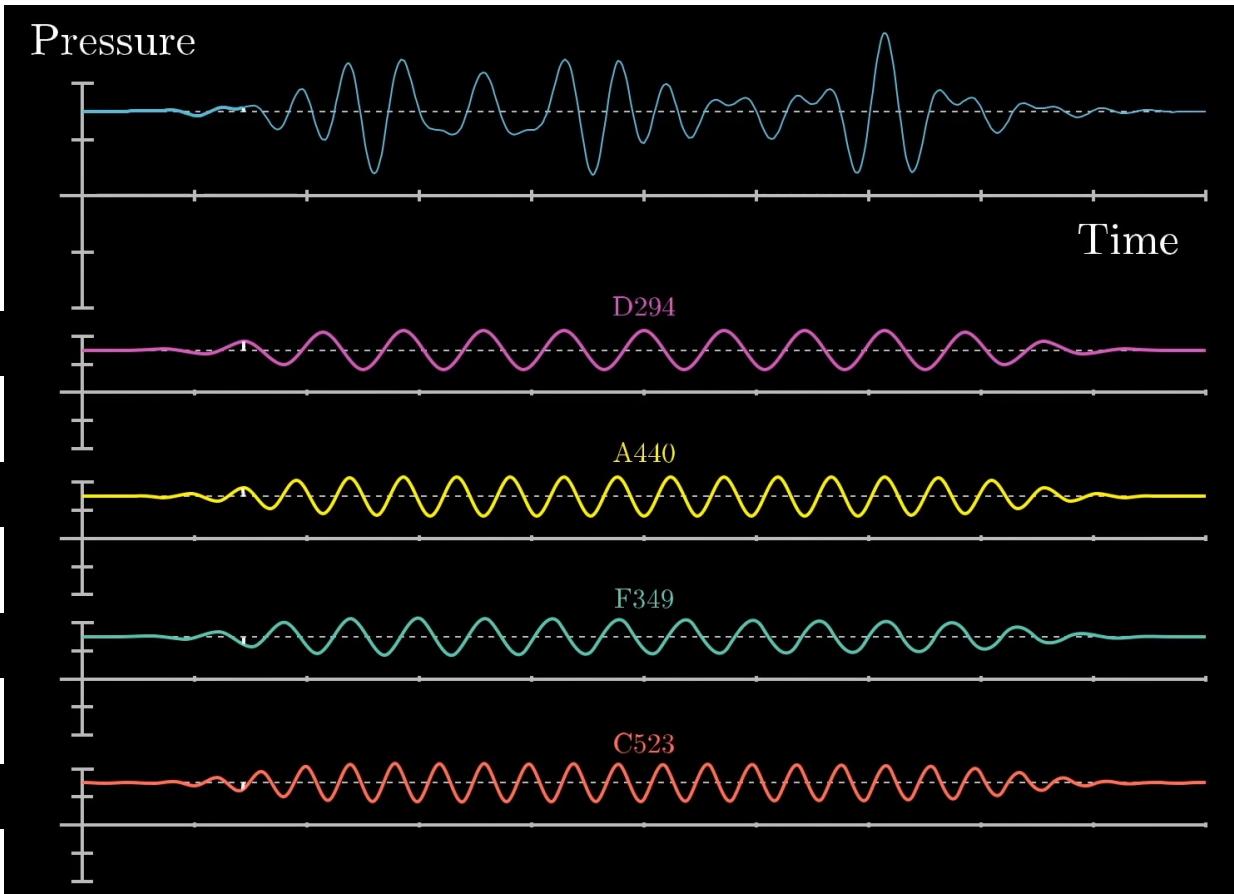
Objectives

1. Preliminary: Singal processing (3blue1brown)
2. Understanding of **graph signals & graph Fourier transform**
3. Understanding the **formulation of ChebNet**
4. **Re-reading GCN, understanding in the original author's way**

Preliminary: Signal processing

Signal processing & filtering

Observed signal



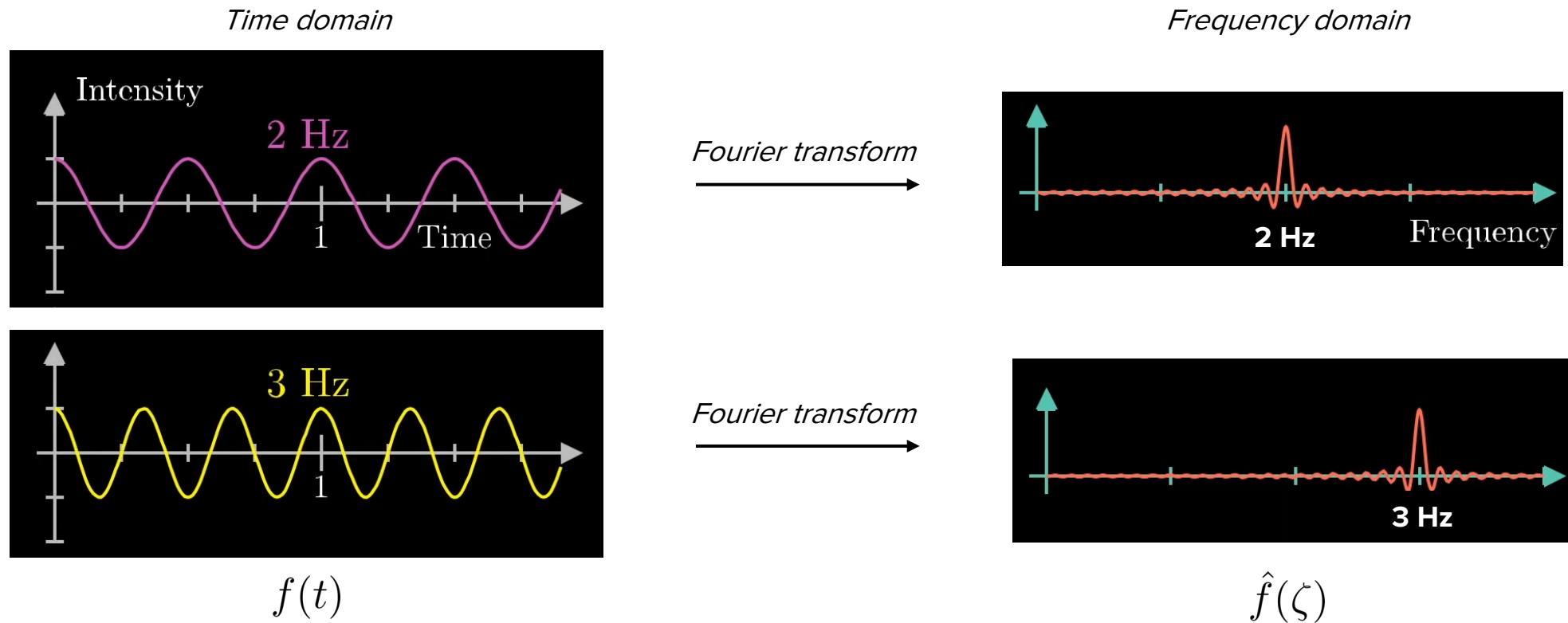
Fourier transforms can be used to analyze signals

Observed signal
with multiple frequencies



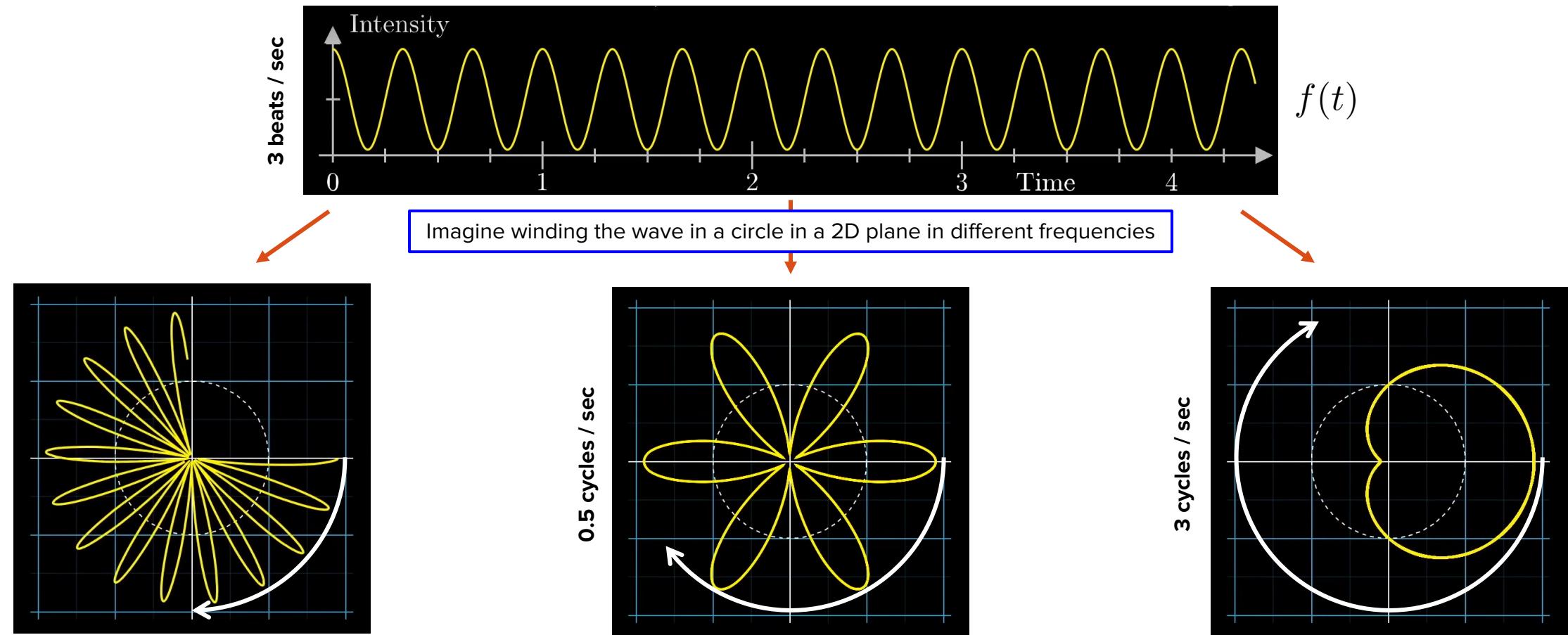
Individual signals
with single frequencies

Signal processing & filtering



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

Signal processing & filtering



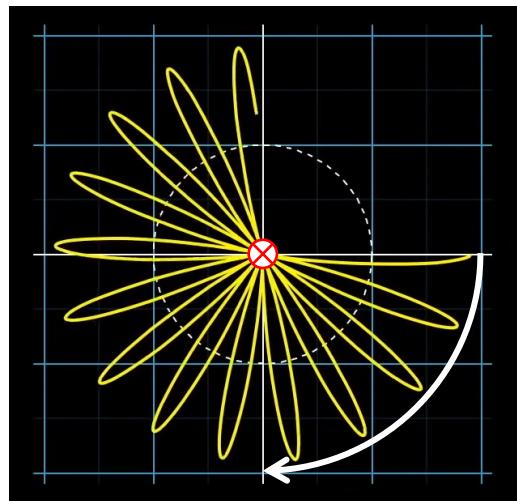
Observation

Something unique happens when the winding frequency *exactly* matches the signal frequency

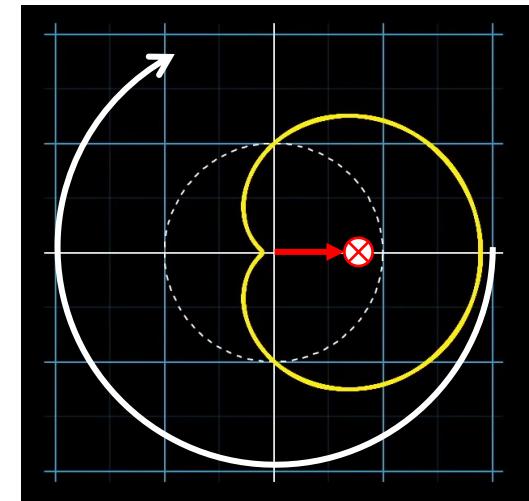
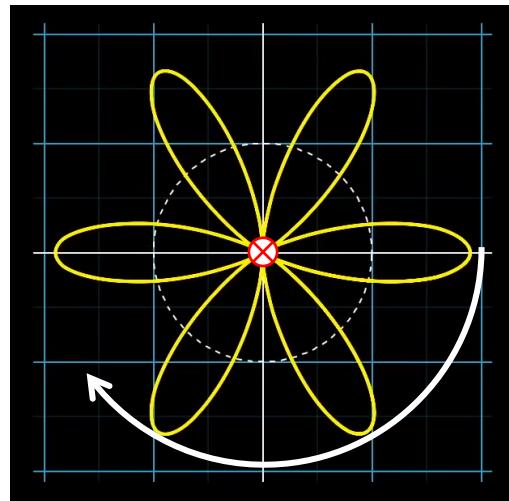
How to measure this?

Signal processing & filtering

 : Center of mass



Measurement of the distance between the origin and **center of mass**



Imagine **winding the wave in a circle** in a 2D plane in different **frequencies**

$$f(t)e^{-2\pi i \zeta t}$$

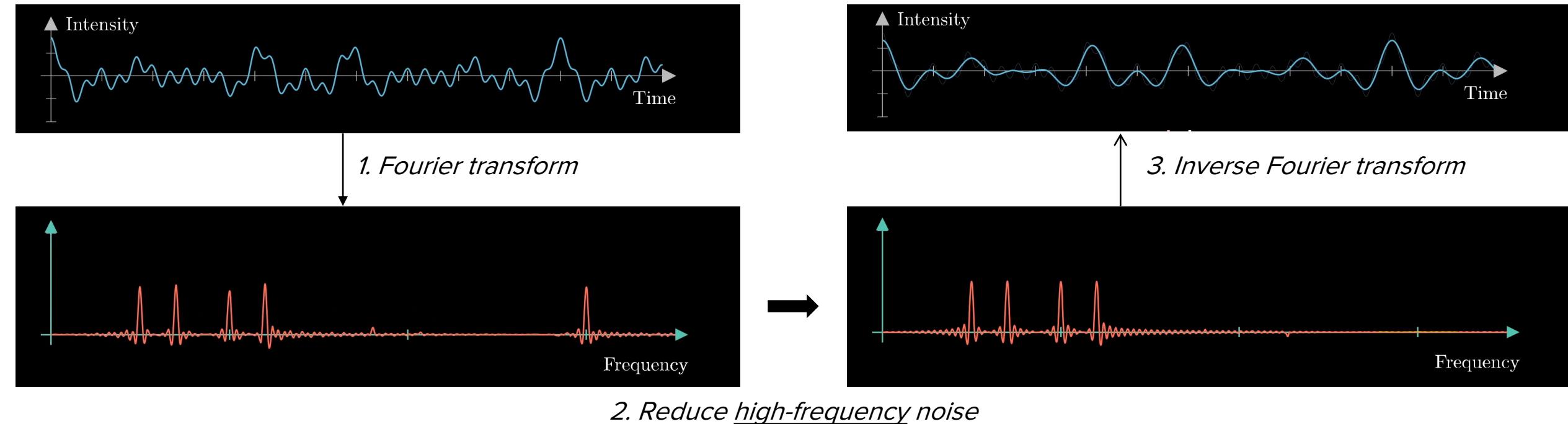
Measurement of the distance between the origin and **center of mass**



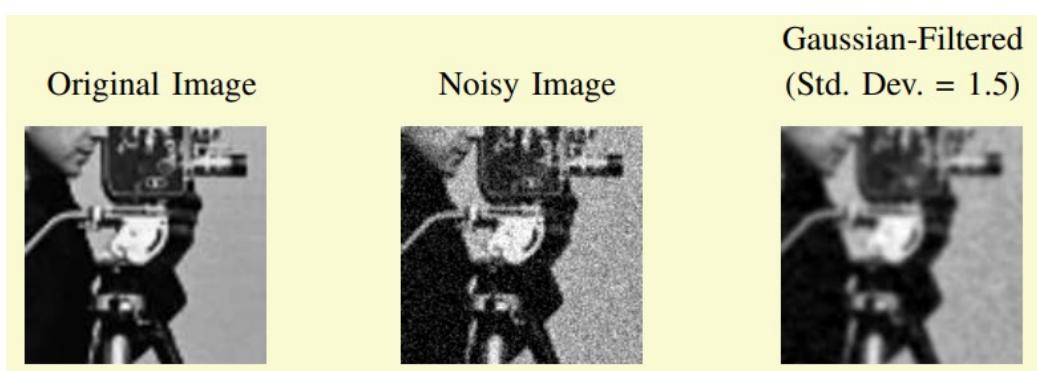
$$\hat{f}(\zeta) = \int_{\mathbb{R}} f(t)e^{-2\pi i \zeta t} dt$$

$$\frac{1}{Z} \int_{\mathbb{R}} f(t)e^{-2\pi i \zeta t} dt$$

Low-pass filtering



*David I Shuman et al., *The Emerging Field of Signal Processing on Graphs*



Additional linear algebra: Spectral decomposition

Spectral Decomposition

- The expression

$$A = PDP^T$$

is called the *spectral decomposition* of A . We can write it as

$$A = [\vec{x}_1 \ \vec{x}_2 \ \cdots \ \vec{x}_n] \begin{bmatrix} \lambda_1 & 0 & \cdots & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & \lambda_n \end{bmatrix} \begin{bmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vdots \\ \vec{x}_n^T \end{bmatrix}$$

- Only works for symmetric matrices.
- When decomposing the adjacency matrix, the eigenvalues are real.
- The eigenvalues are usually ordered.

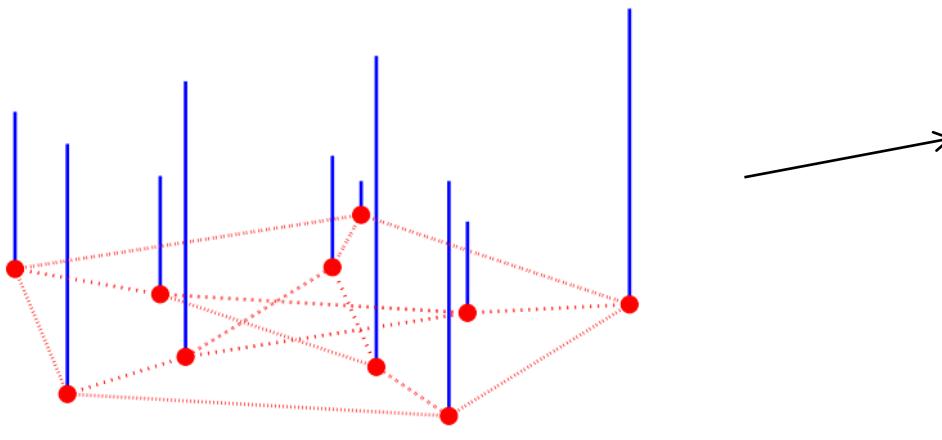
$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

We can also rearrange in an ascending order, just swap the corresponding eigenvectors accordingly.

Understanding of graph signals & graph Fourier transforms

How should we think of signals on graphs?

We will consider a simple 1D node features, which can be easily extended to multi-dimensional case



We can imagine assigning a single value to each vertex
Think of 1-dimensional feature matrix as a function

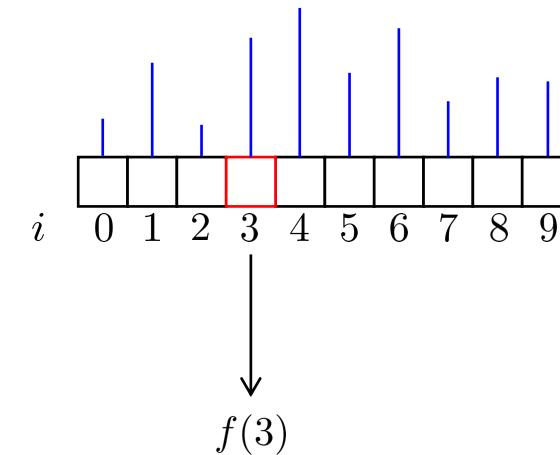
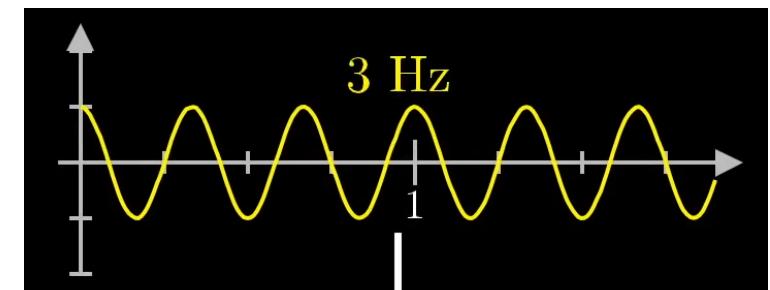


Fig. 1. A random positive graph signal on the vertices of the Petersen graph. The height of each blue bar represents the signal value at the vertex where the bar originates.



Generalization of Fourier transforms to graphs

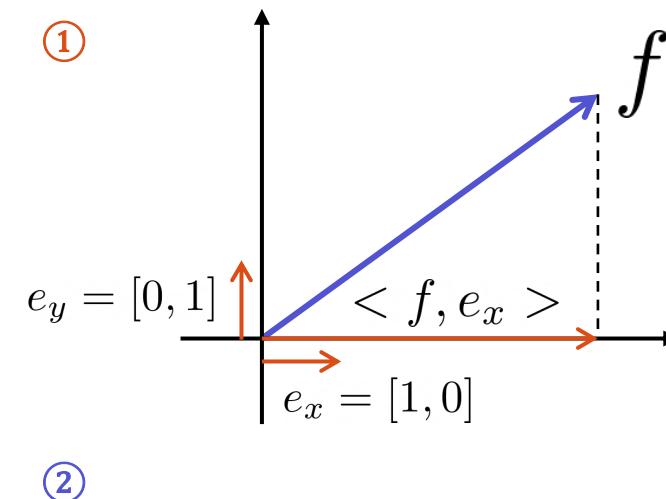
A GRAPH FOURIER TRANSFORM AND NOTION OF FREQUENCY

The classical Fourier transform

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

is the ^①expansion of a function f in terms of the ^②complex exponentials, which are the eigenfunctions of the one-dimensional (1-D) Laplace operator ^③

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



Fourier transform = Inner product with some function
Some function = complex exponentials?

③
Some function = Eigenfunction of the Laplace operator

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

Eigenfunction

Roughly, the Laplace operator measures the local difference between the function and average.

Generalization of Fourier transforms to graphs

Conclusion of the previous slide:

Fourier transform is the inner product between the target function and the eigenfunction of the Laplace operator.

Graph signal!
(# of nodes = N)

Analogously, we can define the graph Fourier transform \hat{f} of any function $f \in \mathbb{R}^N$ on the vertices of \mathcal{G} as the expansion of f in terms of the eigenvectors of the graph Laplacian:

$$\hat{f}(\lambda_\ell) := \langle f, u_\ell \rangle = \sum_{i=1}^N f(i) u_\ell^*(i). \quad (3)$$

This part is the same from before

This part is different because now its discrete

Graph Laplacian

Laplace operator measures the local difference between the function and average.

$$\mathcal{L} = \mathbf{D} - \mathbf{A}$$



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

And we can just get the **eigenvectors** of the graph Laplacian via **spectral decomposition**.

Complementary note 1: On the physical intuition of the graph Laplacian

15

Why is Laplacian related to measuring the difference between the function and average?

Need some further generalizations of mathematical concepts... “Discrete calculus”

Definition 1. Edge derivative at $e=(i, j)$

$$\frac{\partial f}{\partial e} \Big|_i := f(j) - f(i)$$



Definition 2. Graph gradient at vertex i

$$\nabla_i f := \left[\left\{ \frac{\partial f}{\partial e} \Big|_i \right\} \text{ for some edges from } i \right]$$



Definition 3. Local variation at vertex i

$$\begin{aligned} \|\nabla_i f\|_2 &:= \left[\sum_{\text{for some edges from } i} \left(\frac{\partial f}{\partial e} \Big|_i \right)^2 \right]^{1/2} \\ &= \left[\sum_{j \in \mathcal{N}_i} (f(j) - f(i))^2 \right]^{1/2} \end{aligned}$$



We can calculate the **total variation of the whole graph** as the sum of local variation (squared) for all nodes in the graph:

$$\frac{1}{2} \sum_{i \in V} \sum_{j \in \mathcal{N}_i} (f(j) - f(i))^2$$

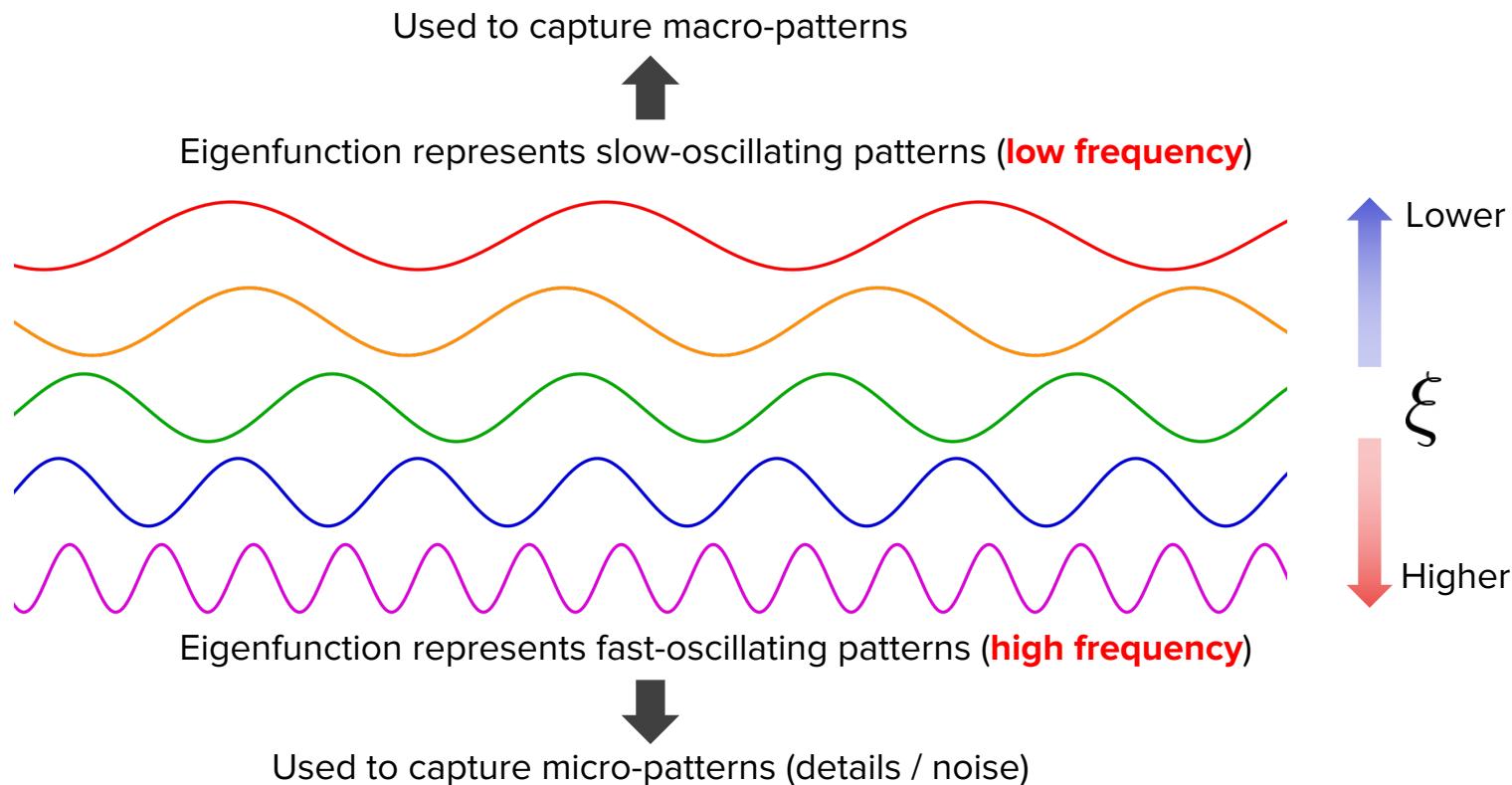
which is

$$f^T \mathcal{L} f$$

....So we can at least understand why the graph Laplacian is useful to capture the patterns of the graph signal. It measures how much the signal differs locally, eventually containing all information on variation of signals.

Complementary note 2: Analogy of eigenfunctions across frequencies

16

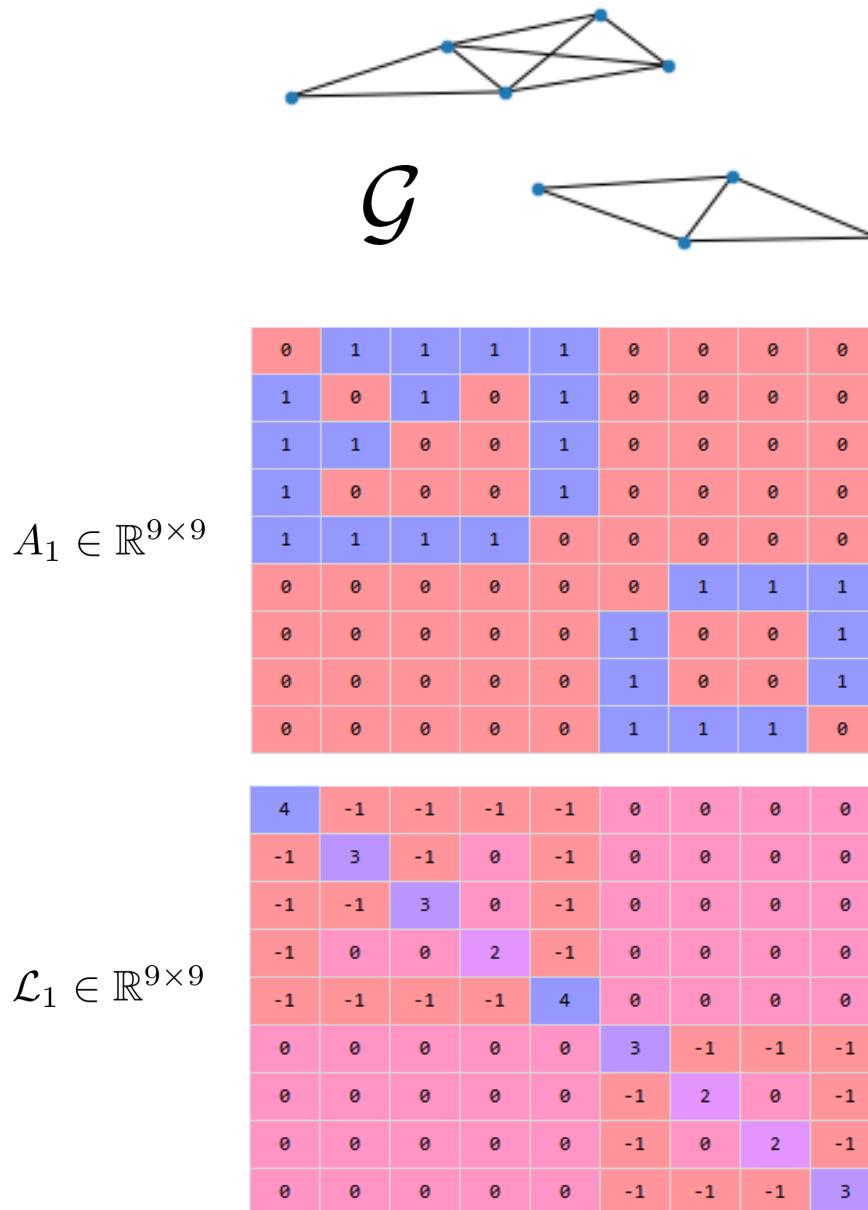


$$e^{2\pi i \xi t}$$

The eigenfunction of the Laplacian

Is this also analogous to graph Fourier basis?

Complementary note 2: Analogy of eigenfunctions across frequencies



Concrete example 1

Spectral Decomposition

The expression

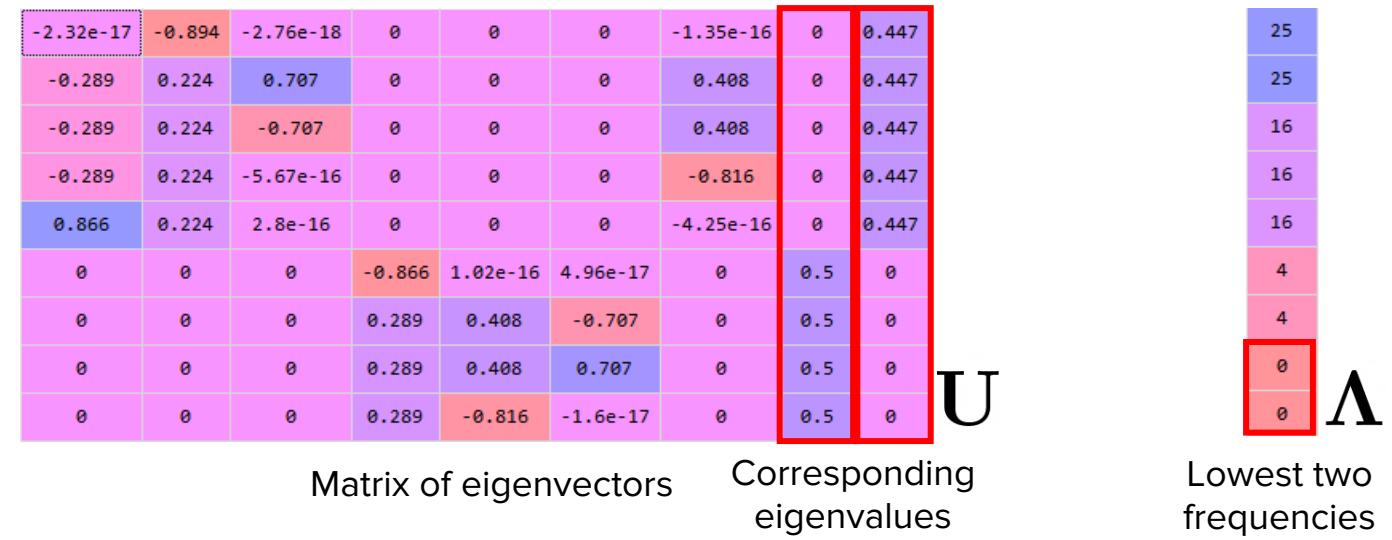
$$A = PDP^T$$

is called the *spectral decomposition* of A . We can write it as

$$\mathcal{L} = U \Lambda U^T$$

$$A = [\vec{x}_1 \ \vec{x}_2 \ \dots \ \vec{x}_n] \begin{bmatrix} \lambda_1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & \ddots & & & & & & \\ & & & \ddots & & & & & \\ & & & & \ddots & & & & \\ & & & & & 0 & \dots & 0 & \lambda_n \end{bmatrix} \begin{bmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vdots \\ \vdots \\ \vec{x}_n^T \end{bmatrix}$$

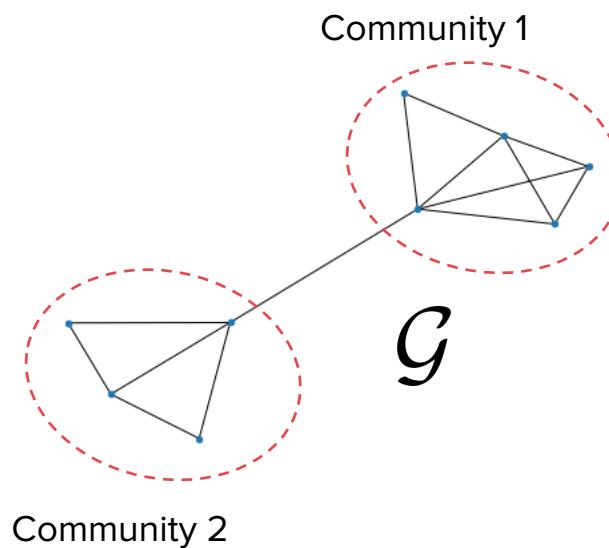
Dr. Cen Babaoglu
Linear Algebra for Machine Learning: Singular Value Decomposition and Principal Component Analysis
cenibabaoglu.com



Notice that the trivial eigenpair reveal the most macro pattern in a graph:
Number of connected components ("blobs")

Complementary note 2: Analogy of eigenfunctions across frequencies

Concrete example 2



1) Notice that there is only one trivial eigenpair since the graph is one giant connected component

-0.153	0.866	-0.125	-1.16e-16	-1.06e-16	-4.25e-17	-9.27e-17	0.316	0.333	40
-0.153	-0.289	-0.125	0.599	0.376	0.116	0.391	0.316	0.333	25
-0.153	-0.289	-0.125	-0.599	-0.376	0.116	0.391	0.316	0.333	18.9
-0.153	-0.289	-0.125	-2.79e-16	3.94e-16	-0.233	-0.783	0.316	0.333	16
0.817	-3.39e-15	0.421	8.45e-16	5.95e-16	-8.86e-18	-9.88e-17	0.212	0.333	16
-0.465	-4.44e-15	0.775	1.33e-15	1.05e-15	0	5.55e-17	-0.27	0.333	4
0.0873	1.35e-15	-0.231	0.217	-0.346	0.678	-0.201	-0.401	0.333	4
0.0873	1.37e-15	-0.231	0.217	-0.346	-0.678	0.201	-0.401	0.333	0.107
0.0873	1.34e-15	-0.231	-0.434	0.692	3.56e-16	5.62e-16	-0.401	0.333	0

U Λ

Lowest two frequencies (eigenvalues)

2) Still, also notice that the first non-trivial eigenvectors returns a ‘soft’ community assignment, which is the the next macro-pattern.
(HIGHLY incourage to read spectral clustering [1])

Complementary note 2: Analogy of eigenfunctions across frequencies

Some further examples... (external slide)

Fourier Bases on Graphs

Fourier Bases on Graphs

- The “complex exponentials” in the graph domain are the eigenvectors of L : [Shuman, 2013]

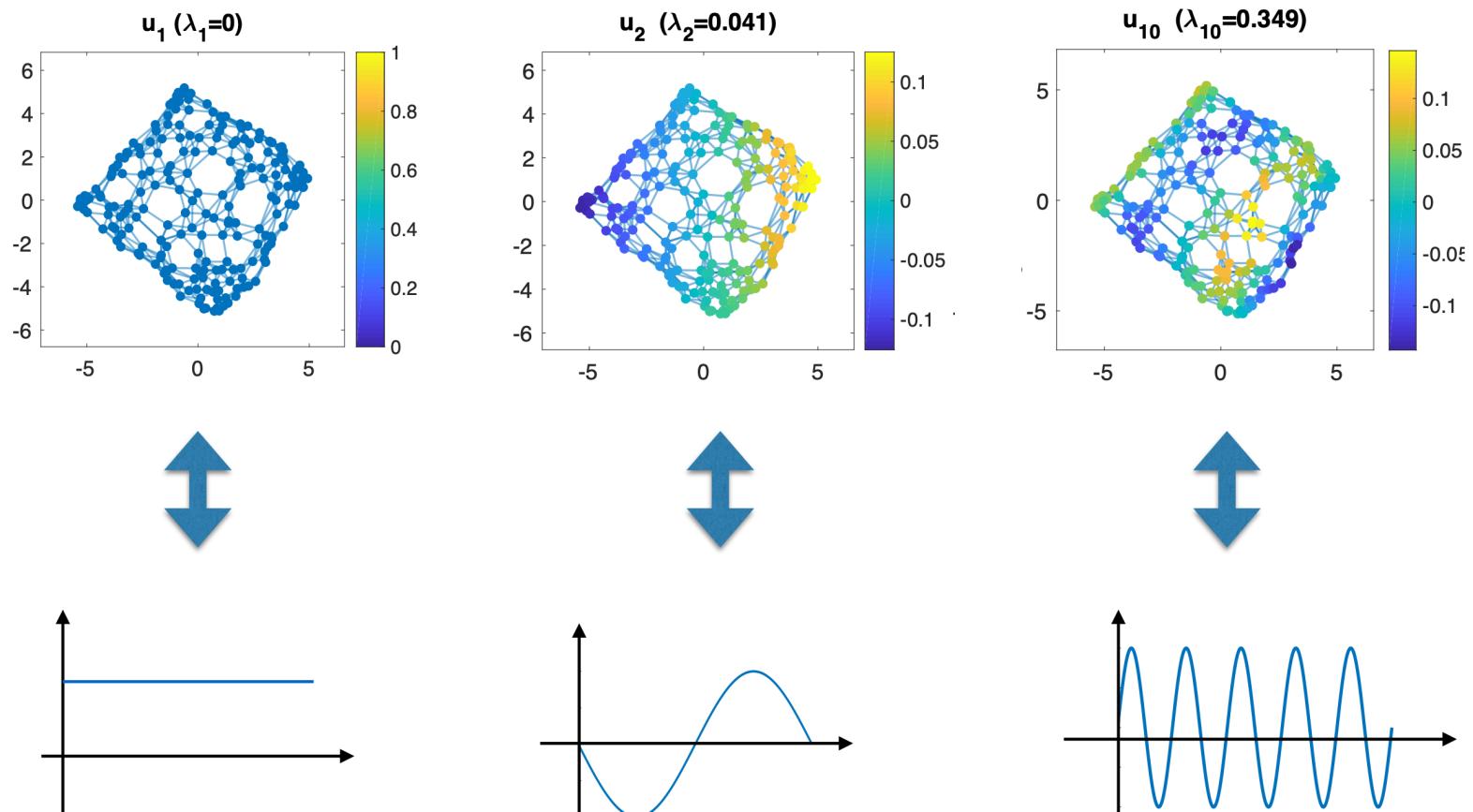
$$Lu_k = \lambda_k u_k \quad \text{for } k = 1, \dots, N$$

- Compare:

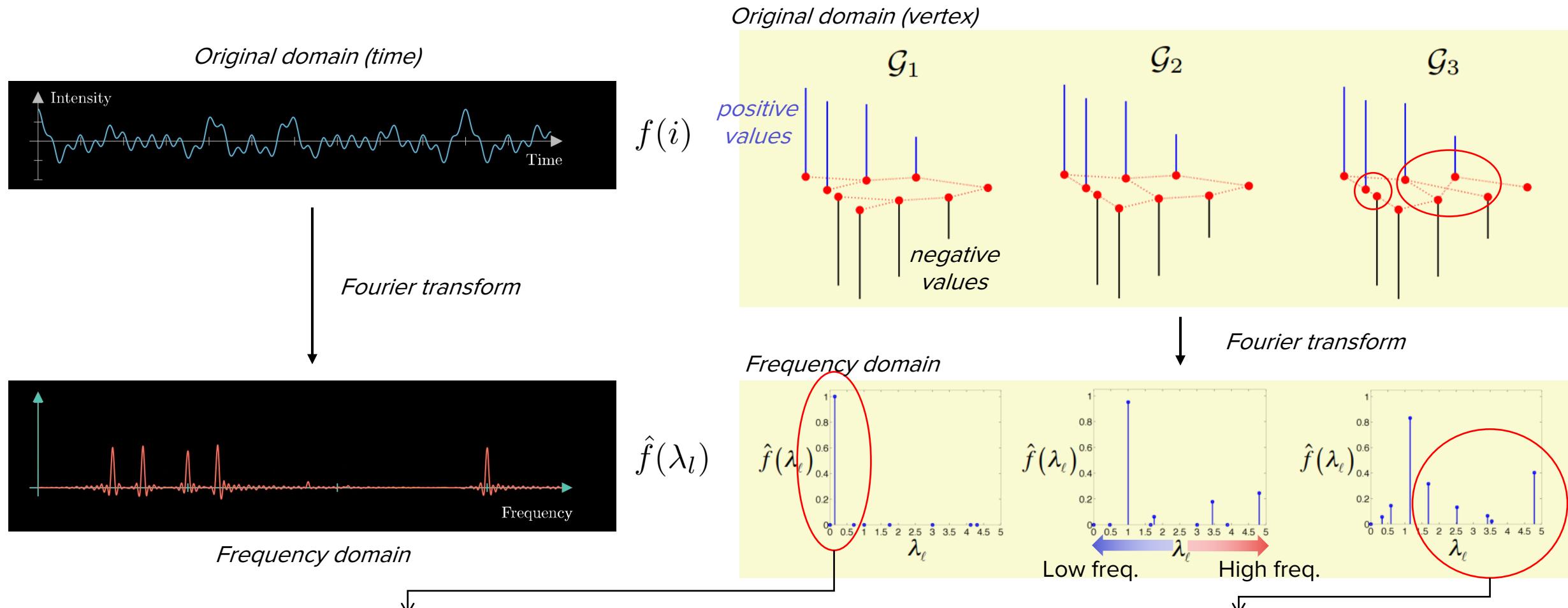
$$-\Delta(e^{j\Omega t}) = \boxed{\Omega^2} e^{j\Omega t} \quad \leftrightarrow \quad Lu_k = \boxed{\lambda_k} u_k$$

frequency

- As λ_k increases, u_k varies more rapidly on the graph.



...Coming back, completing the full graph Fourier transformation



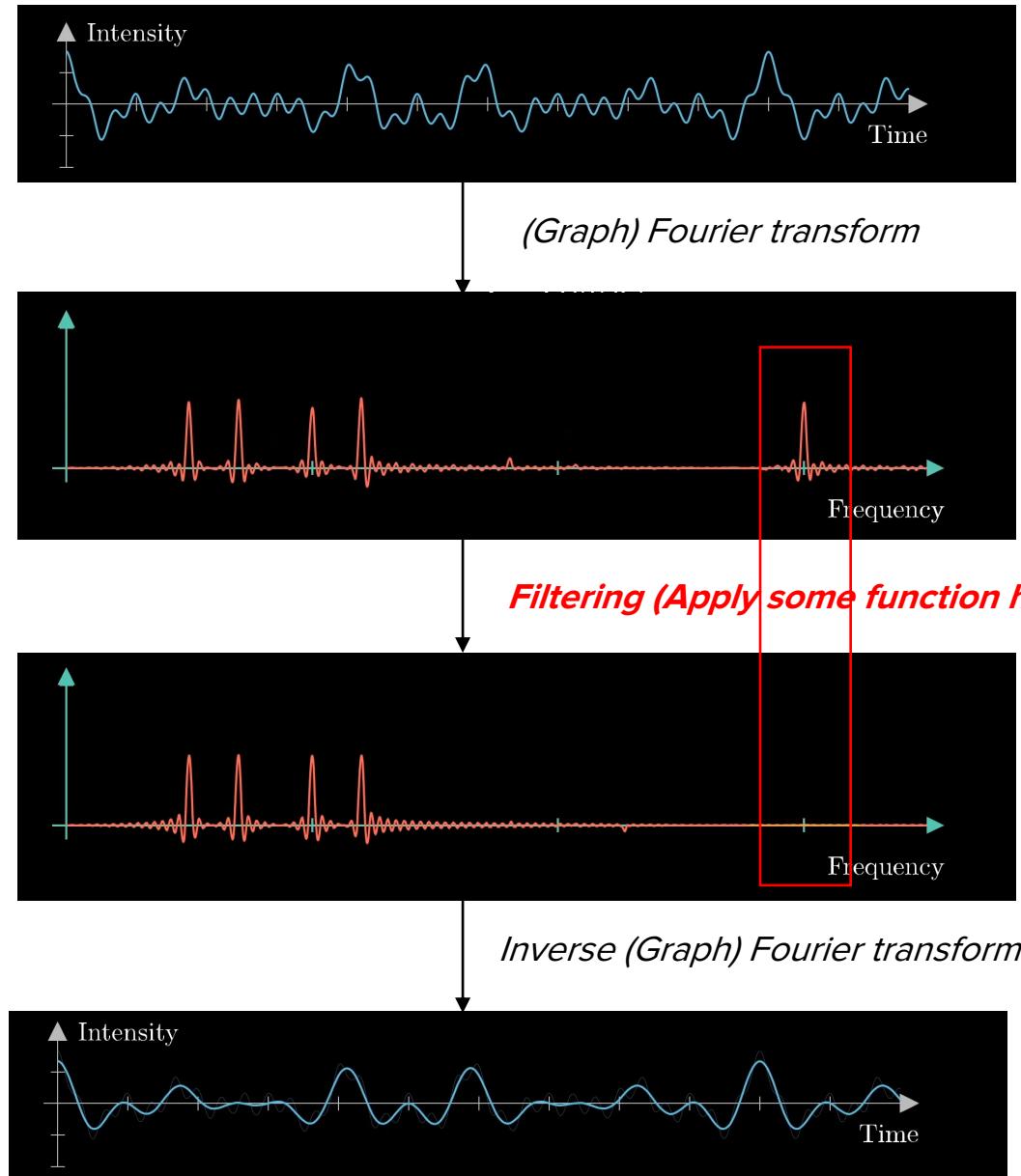
Note that the smoothness of the first graph is reflected in the **high signal of low eigenvalue/frequencies**

* Low frequency = smooth, little variance

In the third graph, **high signals of eigenvalue/frequencies start to appear**

* High frequency = rapid oscillation, high variance

...Coming back, completing the full graph Fourier transformation



$f(i)$

Send f in vertex space to frequency space

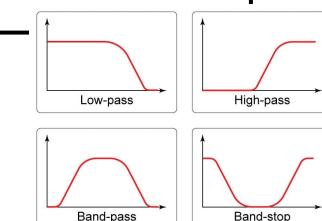
$$\mathcal{L} = \mathbf{U}\Lambda\mathbf{U}^\top \quad \mathcal{L} = \mathbf{D} - \mathbf{A}$$

$$\hat{f}(\lambda_l) = \langle f, u_i \rangle = \mathbf{U}^T f$$

$\hat{f}(\lambda_l)$

Apply a **filtering function** h to eigenvalues

$$\Lambda = \begin{bmatrix} \lambda_0 & & 0 \\ & \ddots & \\ 0 & & \lambda_{N-1} \end{bmatrix} \Rightarrow h(\Lambda) = \begin{bmatrix} h(\lambda_0) & & 0 \\ & \ddots & \\ 0 & & h(\lambda_{N-1}) \end{bmatrix}$$



$$\hat{f}_{out}(\lambda_l) = \hat{f}(\lambda_l) h(\lambda_l)$$

Send back to vertex space

$$f_{new} = \mathbf{U}h(\Lambda)\mathbf{U}^T f$$

Understanding the **formulation of ChebNet & GCN**

Application of convolution (filters) to graphs

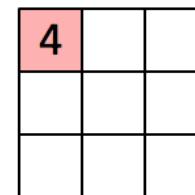
Defferrard et al., Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

1. **Spectral formulation:** Extend the formulation of graph signal processing
2. **Strictly localized filters:** Design a local filter localized in K hop from the central vertex

CNNs were successful because of local parameterized filters

1 <small>\times_1</small>	1 <small>\times_0</small>	1 <small>\times_1</small>	0	0
0	1 <small>\times_1</small>	1 <small>\times_0</small>	1	0
0 <small>\times_1</small>	0 <small>\times_0</small>	1 <small>\times_1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image



Convolved
Feature



Parameterized (Learnable) filter

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

parameterized filter



Convolution theorem: Multiplication in the spectral space = Convolution in the original space

Localized = the kernel is smaller than the original space

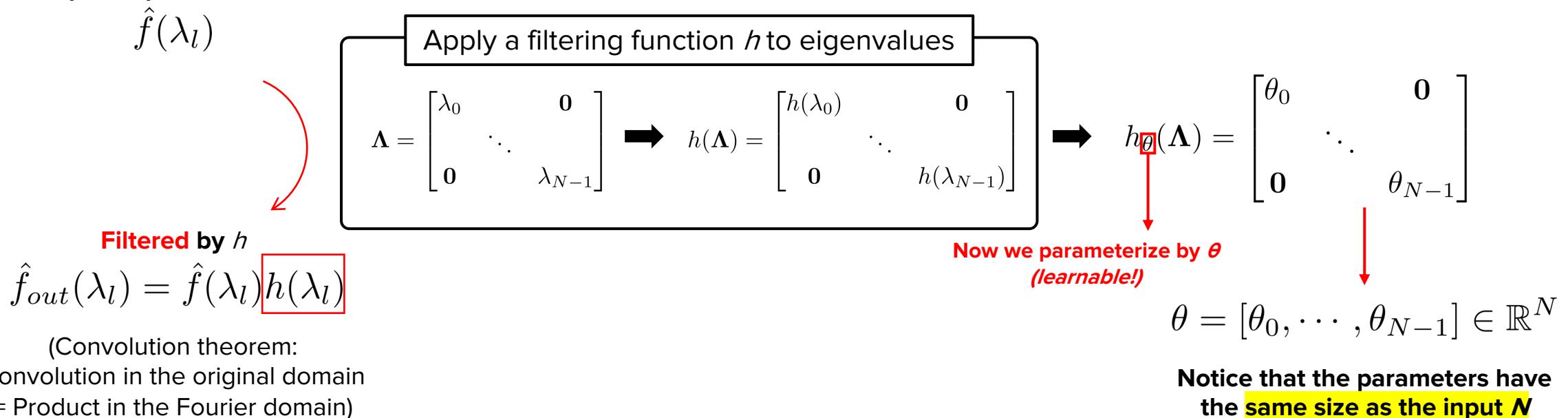
(for example, in the left figure, the kernel is 3 by 3, which is smaller than the image 5 by 5)

3. **Low computational complexity:** Expensive eigenvalue decomposition (spectral decomposition) is not needed

Localizing graph filters

Defferrard et al., Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

Graph in spectral domain



Localizing graph filters

Defferrard et al., Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

$$h_\theta(\Lambda) = \begin{bmatrix} \theta_0 & & 0 \\ & \ddots & \\ 0 & & \theta_{N-1} \end{bmatrix}$$

Global filter

↓

$$\theta = [\theta_0, \dots, \theta_{N-1}] \in \mathbb{R}^N$$

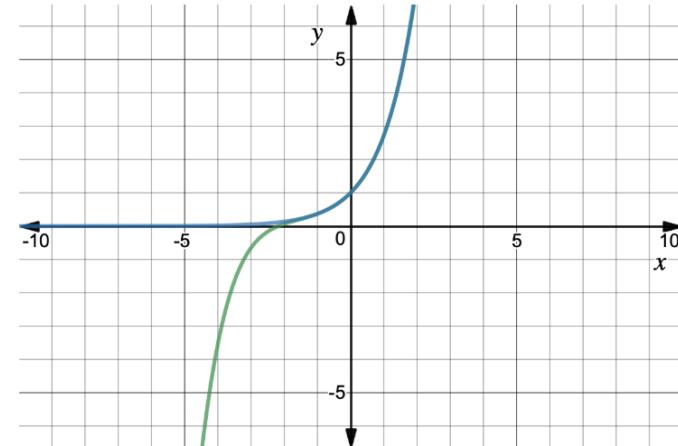
Alternative formulation
→

Polynomial filter

$$h_\theta(\Lambda) = \theta_0 \mathbf{I} + \theta_1 \Lambda + \theta_2 \Lambda^2 + \dots + \theta_{K-1} \Lambda^{K-1}$$

$$= \sum_{k=0}^{K-1} \theta_k \boxed{\Lambda^k}$$

We have the curve $f(x) = e^x$ in blue, and a Polynomial Approximation with equation $g(x) = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5$ in green.



Local filter ($K < N$)

↔

To calculate Λ^k , we calculate up to \mathcal{L}^k .

$$h_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \quad \mathbf{U} h_\theta(\Lambda) \mathbf{U}^\top x = \mathbf{U} \left(\sum_{k=0}^{K-1} \theta_k \Lambda^k \right) \mathbf{U}^\top x = \left(\sum_{k=0}^{K-1} \theta_k \mathcal{L}^k \right) x$$

Local filter ($K < N$)

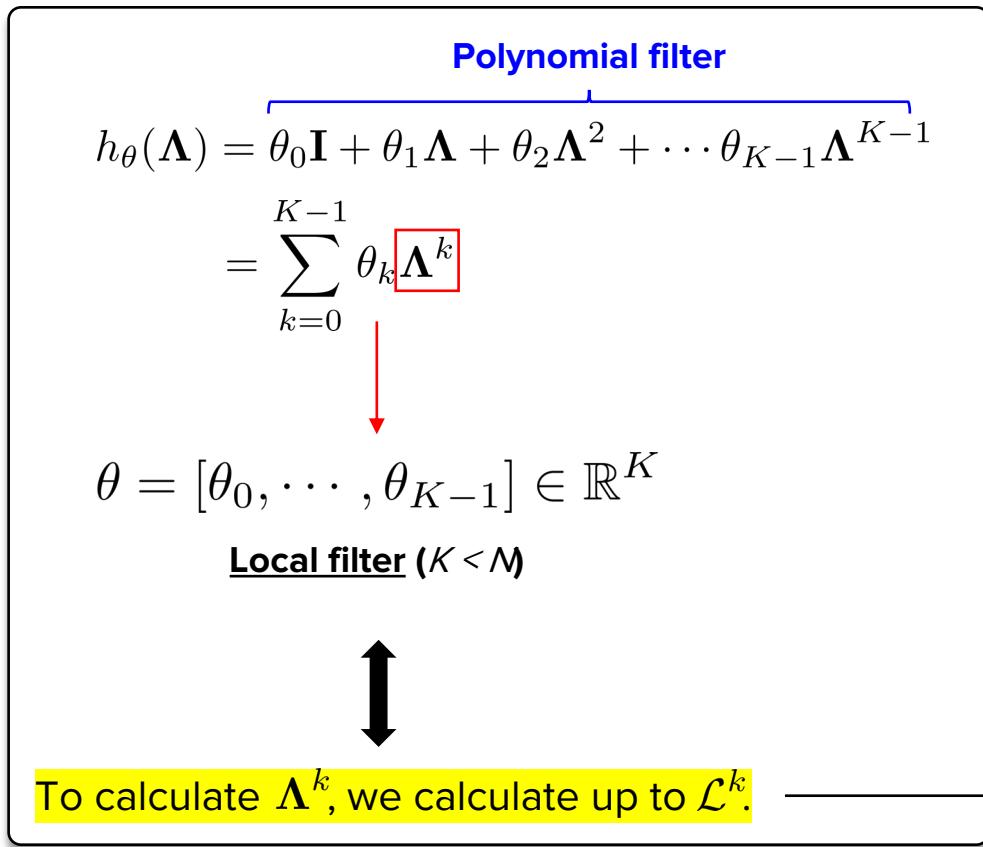
Now think of our original graph convolution,
which becomes...

...this!

$$\mathbf{U} \mathbf{U}^\top = \mathbf{I}$$

Efficient implementation of local graph filters

Defferrard et al., Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering



→ But calculating \mathcal{L}^k still seems heavy...

Idea

1. We can represent polynomials with Chebyshev expansion
2. Chebyshev expansion can be *efficiently* calculated via recursive relation
3. Also, they are superior than polynomial basis since its orthogonal

$$h_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \boxed{\Lambda^k}$$

↓ Replace to...

$$h_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \boxed{T_k(\Lambda)}$$

Chebyshev polynomials

where...

Efficient!

$$\begin{cases} T_0 = 1 \\ T_1 = x \\ T_k = 2xT_{k-1} - T_{k-2} \end{cases} \quad \mathcal{O}(N^2) \rightarrow \mathcal{O}(K|\mathcal{E}|)$$

Efficient implementation of local graph filters

Defferrard et al., Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

conv.ChebConv

```
class ChebConv (in_channels:int, out_channels:int, K:int, normalization:Optional[str] = 'sym',
bias:bool = True, **kwargs) [source]
```

Bases: MessagePassing

The chebyshev spectral graph convolutional operator from the “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering” paper.

$$\mathbf{X}' = \sum_{k=1}^K \mathbf{Z}^{(k)} \cdot \Theta^{(k)}$$

where $\mathbf{Z}^{(k)}$ is computed recursively by

$$\mathbf{Z}^{(1)} = \mathbf{X}$$

$$\mathbf{Z}^{(2)} = \hat{\mathbf{L}} \cdot \mathbf{X}$$

$$\mathbf{Z}^{(k)} = 2 \cdot \hat{\mathbf{L}} \cdot \mathbf{Z}^{(k-1)} - \mathbf{Z}^{(k-2)}$$

and $\hat{\mathbf{L}}$ denotes the scaled and normalized Laplacian $\frac{2\mathbf{L}}{\lambda_{\max}} - \mathbf{I}$.

```
def forward(
    self,
    x: Tensor,
    edge_index: Tensor,
    edge_weight: OptTensor = None,
    batch: OptTensor = None,
    lambda_max: OptTensor = None,
) -> Tensor:

    edge_index, norm = self.__norm__(
        edge_index,
        x.size(self.node_dim),
        edge_weight,
        self.normalization,
        lambda_max,
        dtype=x.dtype,
        batch=batch,
    )

    Tx_0 = x
    Tx_1 = x # Dummy.
    out = self.lins[0](Tx_0)
```

```
# propagate_type: (x: Tensor, norm: Tensor)
if len(self.lins) > 1:
    Tx_1 = self.propagate(edge_index, x=x, norm=norm)
    out = out + self.lins[1](Tx_1)

for lin in self.lins[2:]:
    Tx_2 = self.propagate(edge_index, x=Tx_1, norm=norm)
    Tx_2 = 2. * Tx_2 - Tx_0
    out = out + lin.forward(Tx_2)
    Tx_0, Tx_1 = Tx_1, Tx_2
```

```
if self.bias is not None:
    out = out + self.bias

return out
```

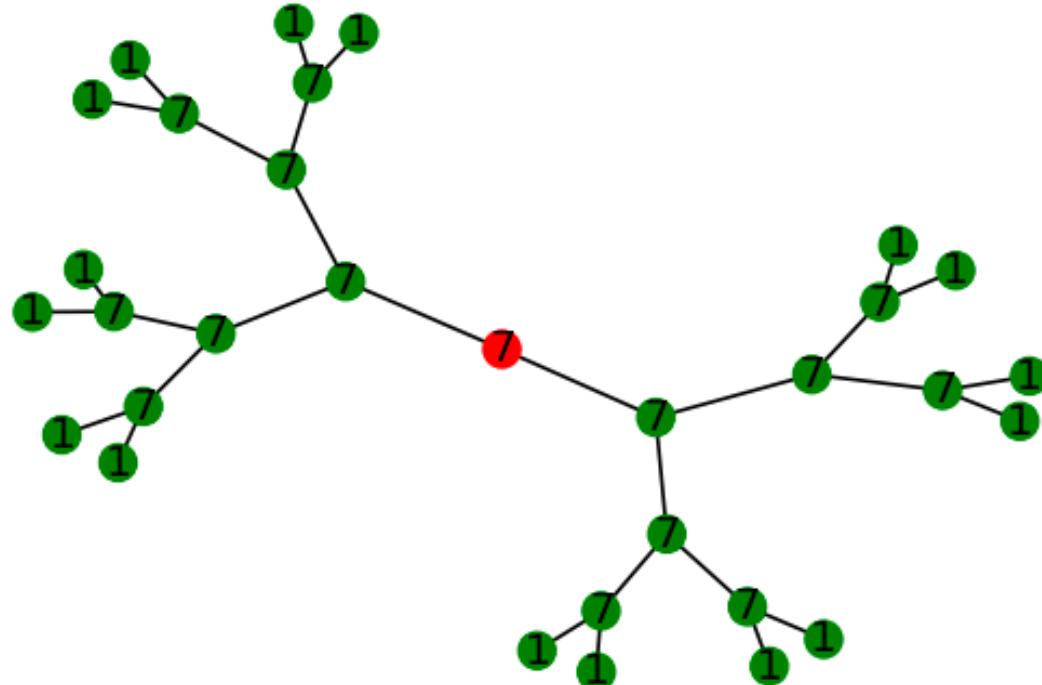
[docs]

A simple example to showcase the locality

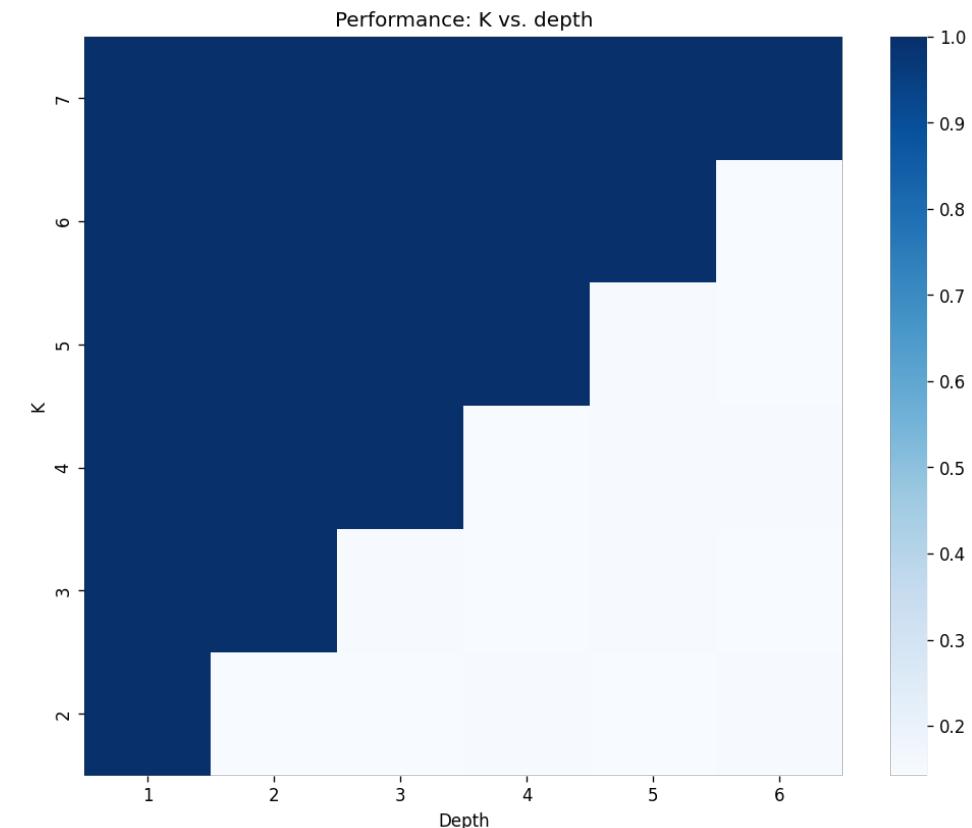
Application of ChebNet to the NEIGHBORSMATCH (Alon & Yahav, 2021) problem

A modified NEIGHBORSMATCH probem

Example (Class label 1)



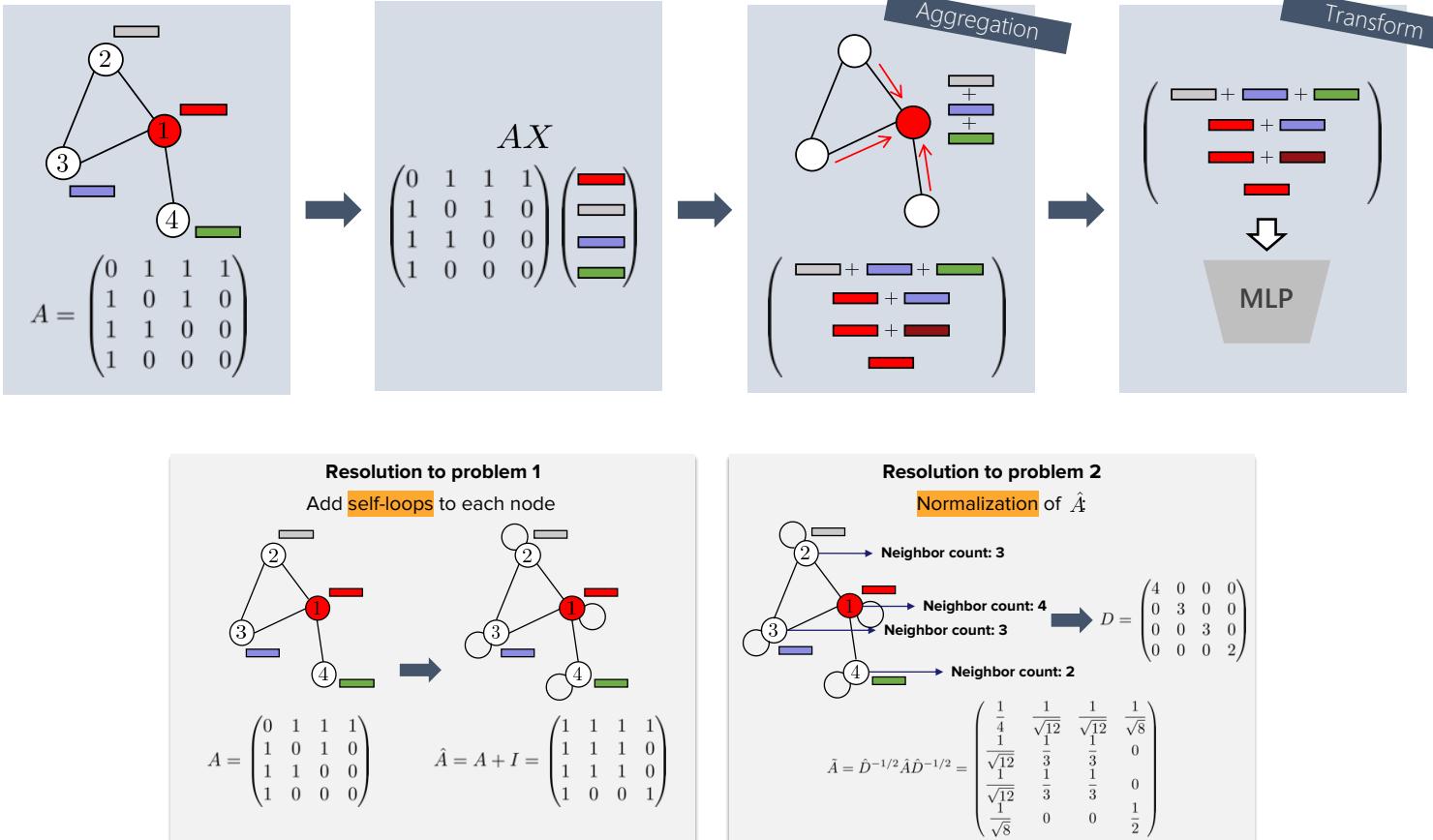
- Target: Root node (red)'s label
- The leaf node's node feature (1 in the example) is the root node's label
- The rest of the node's feature are completely irrelevant
- The model MUST be able to aggregate at least 4-hop local neighbor's information.



- Use a single ChebConv layer
- The model's locality (K) should at least match the minimum depth required by the NEIGHBORSMATCH problem.
- Since the problem is very easy to solve (as long as the information is properly gathered), the performance is 100% or near $1/(\# \text{ of classes})\%$.

Revisiting GCNs, but in the original author's way

Kipf & Welling, Semi-supervised Classification with Graph Convolutional Networks



2 FAST APPROXIMATE CONVOLUTIONS ON GRAPHS

In this section, we provide theoretical motivation for a specific graph-based neural network model $f(X, A)$ that we will use in the rest of this paper. We consider a multi-layer Graph Convolutional Network (GCN) with the following layer-wise propagation rule:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) . \quad (2)$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, such as the ReLU(\cdot) = $\max(0, \cdot)$. $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$. In the following, we show that the form of this propagation rule can be motivated¹ via a first-order approximation of localized spectral filters on graphs (Hammond et al., 2011; Defferrard et al., 2016).

2.1 SPECTRAL GRAPH CONVOLUTIONS

We consider spectral convolutions on graphs defined as the multiplication of a signal $x \in \mathbb{R}^N$ (a scalar for every node) with a filter $g_\theta = \text{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain, i.e.:

$$g_\theta \star x = U g_\theta U^\top x , \quad (3)$$

where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^\top$, with a diagonal matrix of its eigenvalues Λ and $U^\top x$ being the graph Fourier transform of x . We can understand g_θ as a function of the eigenvalues of L , i.e. $g_\theta(\Lambda)$. Evaluating Eq. 3 is computationally expensive, as multiplication with the eigenvector matrix U is $\mathcal{O}(N^2)$. Furthermore,

We are now ready to follow the original author's motivation for GCN.
But keep in mind that all of the previous explanations are still intact. **It is still the same model.**

Revisiting GCNs, but in the original author's way

Kipf & Welling, Semi-supervised Classification with Graph Convolutional Networks

We now understand the first part of Section 2.1.

2.1 SPECTRAL GRAPH CONVOLUTIONS

①: We are assuming a global filter with N parameters.

We consider spectral convolutions on graphs defined as the multiplication of a signal $x \in \mathbb{R}^N$ (a scalar for every node) with a filter $g_\theta = \text{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain, i.e.:

①

$$g_\theta \star x = U g_\theta U^\top x, \quad (3)$$

where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = U\Lambda U^\top$, with a diagonal matrix of its eigenvalues Λ and $U^\top x$ being the graph Fourier transform of x . We can understand g_θ as a function of the eigenvalues of L , i.e. $g_\theta(\Lambda)$.

②

②: Use the original eigenvalues as a basis of some approximation, then we can learn the coefficients.

1. Initial graph signal on every node

2. Send the signal to the frequency domain (Fourier)

3. Apply the (learnable) filter function (Equalizer!)

4. Send the edited signal back to the original domain (inverse Fourier)

Revisiting GCNs, but in the original author's way

Kipf & Welling, Semi-supervised Classification with Graph Convolutional Networks

The next part is now also familiar to us...

As we have seen, this idea has been extensively employed by ChebConv

Evaluating Eq. 3 is computationally expensive, as multiplication with the eigenvector matrix U is $\mathcal{O}(N^2)$. Furthermore, computing the eigendecomposition of L in the first place might be prohibitively expensive for large graphs. To circumvent this problem, it was suggested in Hammond et al. (2011) that $g_\theta(\Lambda)$ can be well-approximated by a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K^{th} order:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^K \theta'_k T_k(\tilde{\Lambda}), \quad (4)$$

with a rescaled $\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_N$. λ_{\max} denotes the largest eigenvalue of L . $\theta' \in \mathbb{R}^K$ is now a vector of Chebyshev coefficients. The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. The reader is referred to Hammond et al. (2011) for an in-depth discussion of this approximation.

Revisiting GCNs, but in the original author's way

Kipf & Welling, Semi-supervised Classification with Graph Convolutional Networks

The next part is now also familiar to us...

Going back to our definition of a convolution of a signal x with a filter $g_{\theta'}$, we now have:

$$g_{\theta'} \star x \approx \sum_{k=0}^K \theta'_k T_k(\tilde{L})x, \quad (5)$$

with $\tilde{L} = \frac{2}{\lambda_{\max}} L - I_N$; as can easily be verified by noticing that $(U\Lambda U^\top)^k = U\Lambda^k U^\top$. Note that this expression is now K -localized since it is a K^{th} -order polynomial in the Laplacian, i.e. it depends only on nodes that are at maximum K steps away from the central node (K^{th} -order neighborhood). The complexity of evaluating Eq. 5 is $\mathcal{O}(|\mathcal{E}|)$, i.e. linear in the number of edges. Defferrard et al. (2016) use this K -localized convolution to define a convolutional neural network on graphs.

In other words, each node requires up to K-hop local neighborhood information to capture up to K-th complex patterns.

Revisiting GCNs, but in the original author's way

Kipf & Welling, Semi-supervised Classification with Graph Convolutional Networks

In Section 2.2, the authors start to introduce the ‘deep learning’ style motivations

2.2 LAYER-WISE LINEAR MODEL

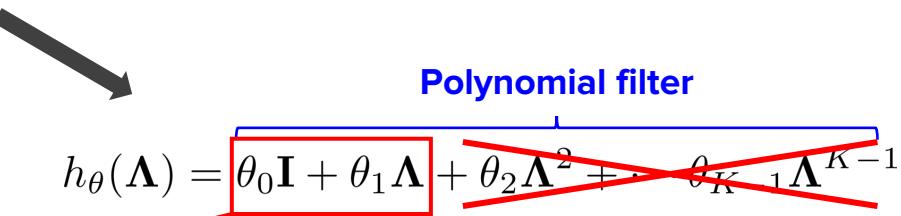
Let’s consider the extremely simplified case as a single layer, and let the designer choose how much layer to stack.

A neural network model based on graph convolutions can therefore be built by stacking multiple convolutional layers of the form of Eq. 5, each layer followed by a point-wise non-linearity. Now, imagine we limited the layer-wise convolution operation to $K = 1$ (see Eq. 5), i.e. a function that is linear w.r.t. L and therefore a linear function on the graph Laplacian spectrum.

⋮

In this linear formulation of a GCN we further approximate $\lambda_{\max} \approx 2$, as we can expect that neural network parameters will adapt to this change in scale during training. Under these approximations Eq. 5 simplifies to:

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 (L - I_N) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x, \quad (6)$$



with two free parameters θ'_0 and θ'_1 . The filter parameters can be shared over the whole graph. Successive application of filters of this form then effectively convolve the k^{th} -order neighborhood of a node, where k is the number of successive filtering operations or convolutional layers in the neural network model.

Revisiting GCNs, but in the original author's way

Kipf & Welling, Semi-supervised Classification with Graph Convolutional Networks

...which leads us to the final (and familiar) GCN layer.

Do we even need to differentiate θ'_0 and θ'_1 ? Let's combine them to a single parameter.

In practice, it can be beneficial to constrain the number of parameters further to address overfitting and to minimize the number of operations (such as matrix multiplications) per layer. This leaves us with the following expression:

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x, \quad (7)$$

with a single parameter $\theta = \theta'_0 = -\theta'_1$. Note that $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ now has eigenvalues in the range $[0, 2]$. Repeated application of this operator can therefore lead to numerical instabilities and exploding/vanishing gradients when used in a deep neural network model. To alleviate this problem, we introduce the following *renormalization trick*: $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, with $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

We can generalize this definition to a signal $X \in \mathbb{R}^{N \times C}$ with C input channels (i.e. a C -dimensional feature vector for every node) and F filters or feature maps as follows:

$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta,$

(8)

GCNs: Let's be more efficient than ChebConv

Kipf & Welling, Semi-supervised Classification with Graph Convolutional Networks

Now the convolution (per layer) is modeled as:

$$\theta(I + \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2})$$

Eigenvalue: [0,1] Eigenvalue: [0,1]
 Eigenvalue: [0,2]

- Stacking multiple layers may cause explosion of eigenvalue (Numerical instability)
- New normalization is needed to keep in [0,1]

*renormalization
trick*

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$$

$$I + \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2} \rightarrow \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2}$$

Final convolution layer

$$\sigma(\tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} \Theta)$$

1. Always set $K=2$ (Up to linear term)

$$h_{\theta}(\Lambda) = \sum_{k=0}^1 \theta_k T_k(\Lambda) = \theta_0 + \theta_1 \Lambda$$

2. Instead, **stack multiple layers**

3. Use normalized Laplacians

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

Takeaways

1. Graph Fourier Transform: Start with the generalized concept of Fourier transform, everything else is the same.
2. ChebNet / ChebConv: A learnable, localized filter for graphs
3. GCN: Push the simplification of graph filters to the extreme, compensate by stacking multiple layers.

*If you are interested in the actual efficiency between ChebNet vs. GCN, check out
https://jordan7186.github.io/blog/2022/Efficiency_Comparison/

*Highly recommended reading (for follow-up work and great summary):
Wang & Zhang, How powerful are spectral graph neural networks, ICML 2022

Thank you!

Please feel free to ask any questions :)

jordan7186.github.io