Understanding Graph Neural Networks Intuitively from the Perspective of Signal Processing

Zhaoyang Chu
Huazhong University of Science and Technology
chuzhaoyang@hust.edu.cn

Abstract

Recent years have witnessed the powerful expressivity of Graph Neural Networks (GNNs) for graph/network-structured data. GNNs have become the dominant approach in graph representation learning with superior performance in a variety of prediction tasks on graphs, such as node classification, link prediction, and graph classification. In this paper, we introduce the basic principle of GNNs and provide an idea to understand GNNs intuitively from the perspective of signal processing. We attempt to explain the core and origin of GNNs from two views, spatial and spectral domains, respectively.

1 Graphs and Graph Neural Networks

1.1 Graph/Network-Structured Data

In real-world scenarios, there are various kinds of graph data structures, such as social networks, networks of neurons, molecular graphs, and so on.

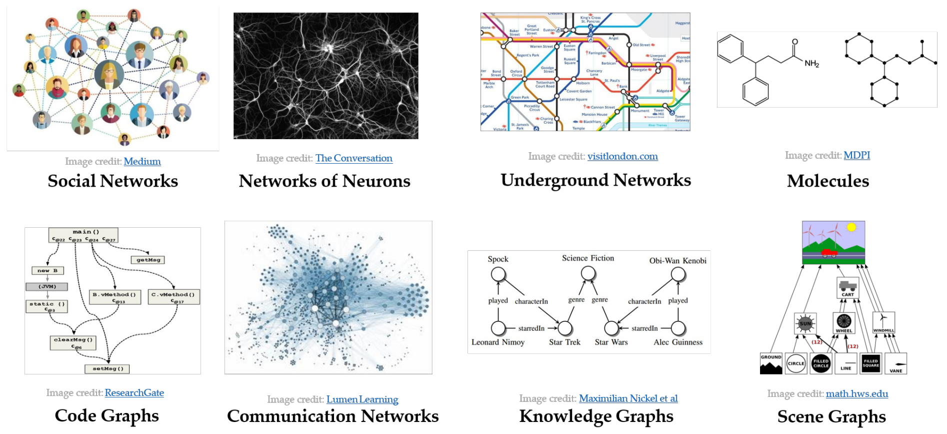


Image credit: Medium
Social Networks
Image credit: The Conversation
Networks of Neurons
Image credit: visitlondon.com
Underground Networks
Image credit: MDPI
Molecules
Image credit: ResearchGate
Code Graphs
Image credit: Lumen Learning
Communication Networks
Image credit: Maximilian Nickel et al
Knowledge Graphs
Image credit: math.hws.edu
Scene Graphs

Figure 1: Graph/Network-structured data.

1.2 Various Tasks on Graphs

Numerous real-world applications can be treated as computational tasks on graphs. For example, air quality forecasting can be regarded as a node classification task, friend recommendation in social networks can be solved as a link prediction task, and protein interface prediction can be regarded as a graph classification task.

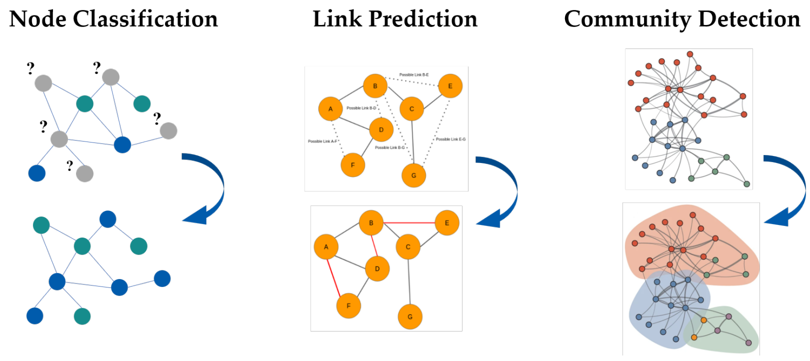


Figure 2: Various tasks on graphs.

1.3 Graph Neural Networks

GNNs map nodes to d-dimensional embeddings such that similar nodes in the network are embedded close together. The key idea of GNNs is to generate node embeddings based on local network neighborhoods, which will be described in detail later.

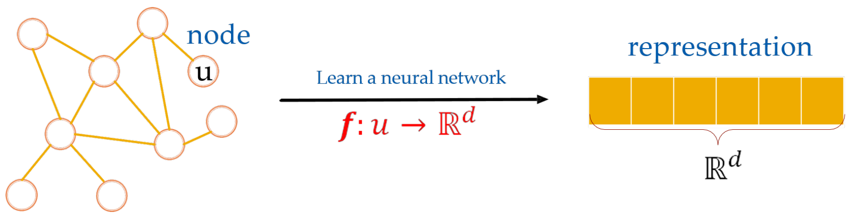


Figure 3: Graph representation learning.

2 Spatial-Based Graph Neural Networks

2.1 Graph Signal

Figure 4 depicts a graph signal consisting of an unweighted undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ and a mapping function $f : \mathcal{V} \to \mathbb{R}^{N \times d}$ from nodes to real values. Without loss of generality, we set $d = 1$ and denote the mapped values for all nodes as $\mathbf{f} \in \mathbb{R}^N$ with $\mathbf{f}[i]$ corresponding to node v_i.

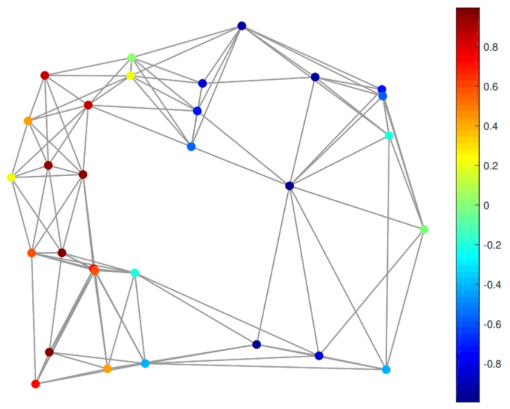


Figure 4: Graph signal.

2.2 Sequence Signal and 1D Laplacian Operator

Given a sequence signal $\mathbf{s} = (1, 1, -1, 1, 1, 1, -1, 1, 1)$ produced by a temporal function $s(t)$, where t is the time step, we aim to detect changes of its frequency information to filter/reject noise signals, where the frequency is analogous to velocity: $\omega = \frac{\mathbf{d}s}{\mathbf{d}t}$.

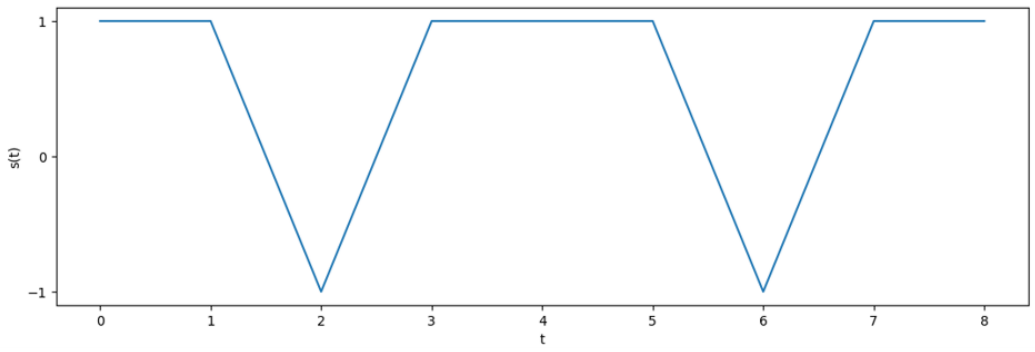


Figure 5: 1D sequence signal.

To this end, we introduce the Laplacian operator in the discrete form to represent the variation of frequencies. We describe it starting from some basic concepts step by step in the following.

A basic operator $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$, named Del or Nabla:

- Given a scalar function g, the gradient operation transforms a scalar field to a vector field via scalar multiplication: $grad(g) = \nabla g = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})g = (\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}, \frac{\partial g}{\partial z})$.
- Given a function vector $\mathbf{g} = (g_1, g_2, g_3)$, the divergence operation transforms a vector field to a scalar field via dot product: $div(\mathbf{g}) = \nabla \cdot \mathbf{g} = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}) \cdot (g_1, g_2, g_3) = \frac{\partial g_1}{\partial x} + \frac{\partial g_2}{\partial y}, + \frac{\partial g_3}{\partial z}$.
- Laplacian operator (Δ or ∇^2) is the divergence of the gradient of g: $\Delta g = div(grad(g)) = \nabla \cdot \nabla g = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}) \cdot (\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}, \frac{\partial g}{\partial z}) = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} + \frac{\partial^2 g}{\partial z^2}$.

The discrete Laplacian operator:

- First derivative: $s'(t) = s(t+1) - s(t)$.
- Laplacian or Second derivative: $\Delta s = s''(t) = s'(t) - s'(t-1) = (s(t+1) - s(t)) - (s(t) - s(t-1)) = s(t+1) + s(t-1) - 2s(t)$.

We can design a filter (or called convolution kernel), i.e., (1, -2, 1), to capture the salient frequency variation.

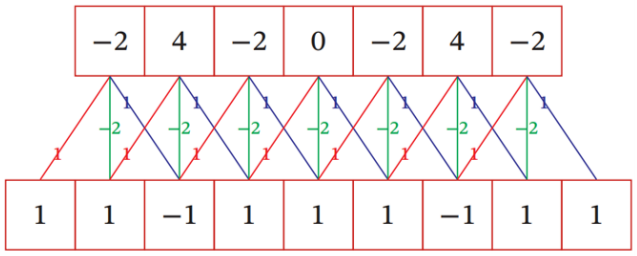


Figure 6: Detecting noise signals by the 1D Laplacian operator.

Relatively, we can design a filter (1/3, 1/3, 1/3) to smooth the temporal signal, i.e., simple moving average.

(1.00, 1.00, -1.00, 1.00, 1.00, 1.00, -1.00, 1.00, 1.00)

$\downarrow$ (1/3, 1/3, 1/3)

(0.66, 0.33, 0.33, 0.33, 1.00, 0.33, 0.33, 0.33, 0.66)

$\downarrow$ (1/3, 1/3, 1/3)

(0.33, 0.44, 0.33, 0.55, 0.55, 0.55, 0.33, 0.44, 0.33)

$\downarrow$ (1/3, 1/3, 1/3)

(0.26, 0.37, 0.44, 0.48, 0.55, 0.48, 0.44, 0.37, 0.26)

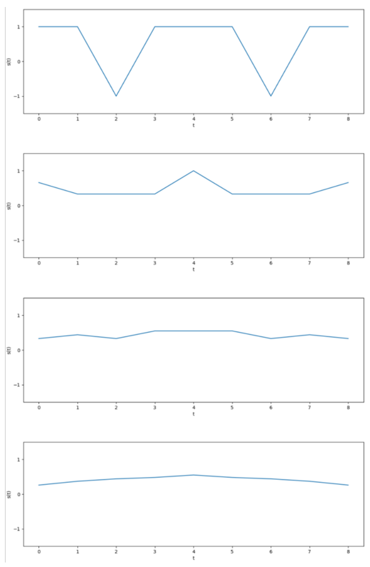


Figure 7: Smoothing the 1D sequence signal by simple moving average.

2.3 Image Signal and 2D Laplacian Operator

Given a 2D image with pixels as signals $\mathbf{P} \in \mathbb{R}^{m \times n}$, we formulate it as a 2D discrete function $p(x, y)$. We generalize the discrete Laplacian operator to the 2D image:

$$\begin{aligned}
\Delta p &= \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} \\
&= p(x+1, y) + p(x-1, y) - 2p(x, y) + p(x, y+1) + p(x, y-1) - 2p(x, y) \\
&= p(x+1, y) + p(x-1, y) + p(x, y+1) + p(x, y-1) - 4p(x, y) \\
&= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} p(x-1, y+1) & p(x, y+1) & p(x+1, y+1) \\ p(x-1, y) & p(x, y) & p(x+1, y) \\ p(x-1, y-1) & p(x, y-1) & p(x+1, y-1) \end{bmatrix}
\end{aligned}$$

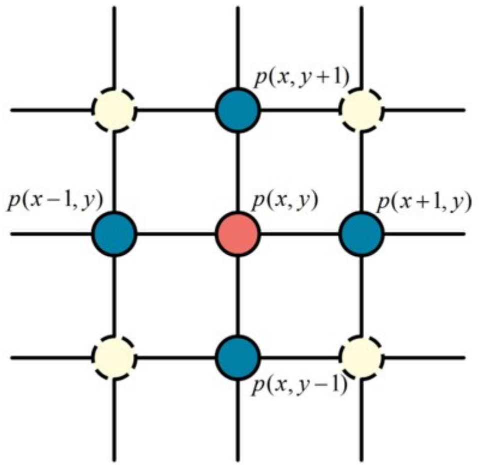


Figure 8: Applying the 2D Laplacian operator on the image grid.

Regarding image processing, the pixel values show "jump" changes (high-frequency information) in the edge areas of the target object. Thus, the 2D discrete Laplacian operator can be used for the edge detection task.

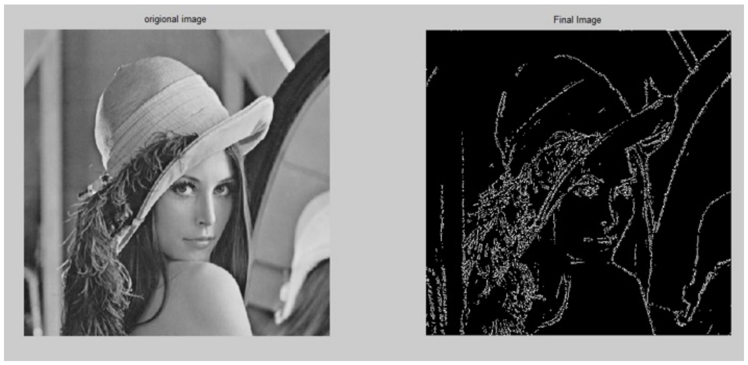


Figure 9: Edge detection.

Relatively, to smooth a noisy image signal (image denoising), we can design a filter as follows:

$$\begin{bmatrix} 0 & 1/5 & 0 \\ 1/5 & 1/5 & 1/5 \\ 0 & 1/5 & 0 \end{bmatrix} \odot \begin{bmatrix} p(x-1,y+1) & p(x,y+1) & p(x+1,y+1) \\ p(x-1,y) & p(x,y) & p(x+1,y) \\ p(x-1,y-1) & p(x,y-1) & p(x+1,y-1) \end{bmatrix}$$

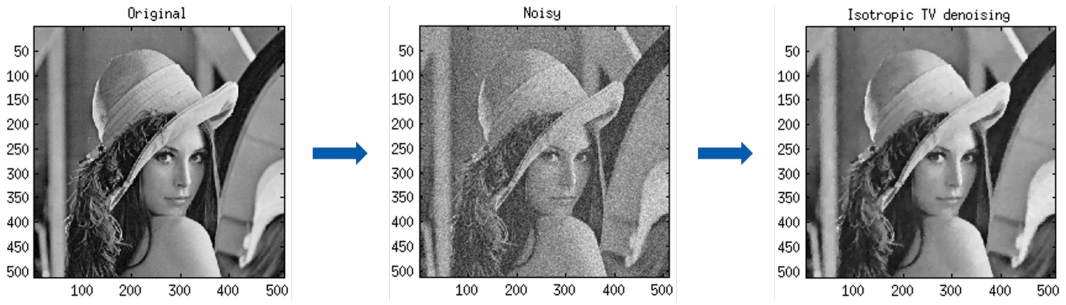


Figure 10: Image denoising.

2.4 Graph Laplacian

We can generalize the 2D discrete Laplacian operator to the graph signal $\mathbf{f}$:

- Gradient: $grad(f) = \nabla f = (\mathbf{f}[i] - \mathbf{f}[j])_{A_{i,j}=1}$.
- Graph Laplacian: $\Delta f = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}[i] - \mathbf{f}[j])$. This equation represents the summarization of the differences between node v_i and its neighbors $\mathcal{N}(v_i)$.

For node v_i, we convert the graph Laplacian operation to:

$$\Delta f = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}[i] - \mathbf{f}[j]) = d(v_i) \cdot \mathbf{f}[i] - \sum_{v_i \in \mathcal{N}(v_i)} \mathbf{f}[j] = d(v_i) \cdot \mathbf{f}[i] - \sum_{j=1}^{N} A_{i,j} \cdot \mathbf{f}[j]$$

On this basis, we rewrite the graph Laplacian operation in the matrix form and obtain a new vector as:

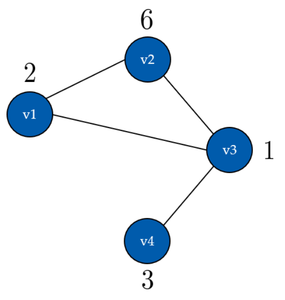
$$\mathbf{h} = \mathbf{D}\mathbf{f} - \mathbf{A}\mathbf{f} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{L}\mathbf{f}$$

where $\mathbf{h}[i]$ is corresponding to node v_i, $\mathbf{L}$ is called Laplacian matrix.

2.5 Spatial-Based Graph Filter

The Laplacian matrix $\mathbf{L}$ can be viewed as a spatial-based filter to find salient or noisy nodes in the graph signal $\mathbf{f}$. The elements of the Laplacian matrix $\mathbf{L}$ are given by:

$$L_{i,j} = \begin{cases} d(v_i) & \text{if } i = j, \\ -1 & \text{if } (v_i, v_j) \in \mathcal{E}, \\ 0 & \text{otherwise.} \end{cases}$$



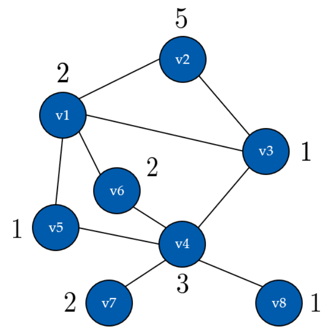
$$\mathbf{D}=\begin{bmatrix}2&0&0&0\\0&2&0&0\\0&0&3&0\\0&0&0&1\end{bmatrix},\mathbf{A}=\begin{bmatrix}0&1&1&0\\1&0&1&0\\1&1&0&1\\0&0&1&0\end{bmatrix},\mathbf{L}=\begin{bmatrix}2&-1&-1&0\\-1&2&-1&0\\-1&-1&3&-1\\0&0&-1&1\end{bmatrix}$$

$$\underbrace{\boxed{\begin{bmatrix}2&-1&-1&0\\-1&2&-1&0\\-1&-1&3&-1\\0&0&-1&1\end{bmatrix}}}_{\text{Convolution kernel}}\odot\begin{bmatrix}2\\6\\1\\3\end{bmatrix}=\begin{bmatrix}-3\\9\\-8\\2\end{bmatrix}$$

Figure 11: Finding noisy nodes via the graph Laplacian operator.

To eliminate the impact of the number of neighbors of the node, we introduce the normalized form of the Laplacian matrix, called random walk normalized Laplacian, as follows:

$$\mathbf{L}^{rw}=\mathbf{D}^{-1}\mathbf{L}, L^{rw}_{i,j}=\begin{cases}1 & \text{if } i=j,\\ -\frac{1}{d(v_i)} & \text{if } (v_i,v_j)\in\mathcal{E},\\ 0 & \text{otherwise.}\end{cases}$$

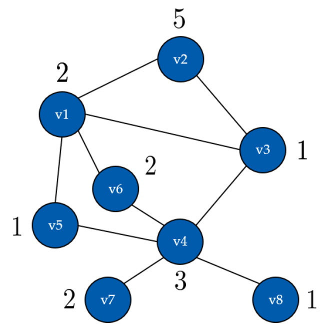


$$\begin{bmatrix}2\times4-9\\5\times2-3\\1\times3-10\\3\times5-7\\1\times2-5\\2\times2-5\\2\times1-3\\1\times1-3\end{bmatrix}=\begin{bmatrix}-1\\7\\-7\\8\\-3\\-1\\-1\\-2\end{bmatrix}\xrightarrow{\mathbf{L}\to\mathbf{L}^{rw}}\begin{bmatrix}-0.25\\3.5\\-2.33\\1.6\\-1.5\\-0.5\\-1\\-2\end{bmatrix}$$

Figure 12: Random walk normalized graph Laplacian.

Further, we consider the impact of the neighbor number of the neighbor of the node and introduce the symmetric normalized Laplacian as:

$$\mathbf{L}^{sym}=\mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}, L^{sym}_{i,j}=\begin{cases}1 & \text{if } i=j,\\ -\frac{1}{\sqrt{d(v_i)d(v_j)}} & \text{if } (v_i,v_j)\in\mathcal{E},\\ 0 & \text{otherwise.}\end{cases}$$



$$\begin{bmatrix}2\times4-9\\5\times2-3\\1\times3-10\\3\times5-7\\1\times2-5\\2\times2-5\\2\times1-3\\1\times1-3\end{bmatrix}=\begin{bmatrix}-1\\7\\-7\\8\\-3\\-1\\-1\\-2\end{bmatrix}\xrightarrow{\mathbf{L}\to\mathbf{L}^{sym}}\begin{bmatrix}-1.117\\3.885\\-2.393\\0.451\\-0.656\\0.344\\0.658\\-0.342\end{bmatrix}$$

Figure 13: Symmetric normalized graph Laplacian.

Relatively, we can design a spatial-based filter $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ to smooth the graph signal $\mathbf{f}$, where $\tilde{\mathbf{A}}=\mathbf{A}+\mathbf{I}$ and $\tilde{\mathbf{D}}=\mathbf{D}+\mathbf{I}$, which is analogous to simple moving average over sequence or image.

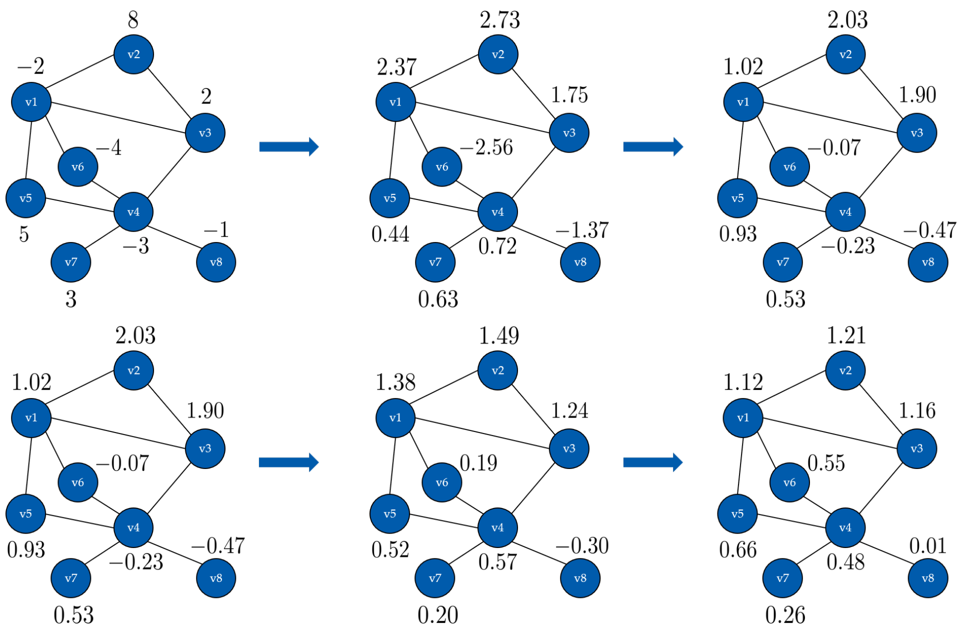


Figure 14: Graph smoothing.

In general, the spatial-based filter is also considered to propagate and aggregate information among 1-hop neighbors locally based on graph structures (i.e., topological relationships between nodes). We can design various information aggregators to propose different spatial-based graph filters.

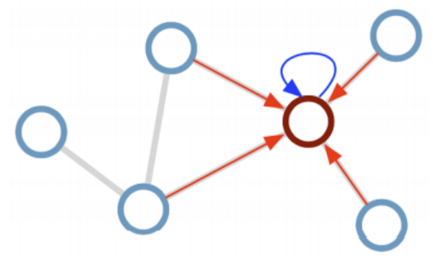


Figure 15: Spatial-based graph filter for local neighborhood aggregation.

With multiple filters stacked, each node can receive more information from higher-order neighbors, e.g., 2-hop, 3-hop, ..., k-hop neighbors.

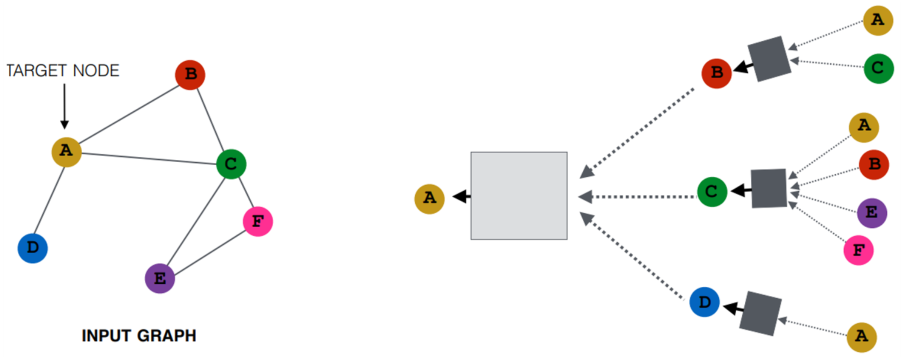


Figure 16: High-order neighborhood aggregation.

We can build various graph neural networks by adding the trainable parameter matrix $\mathbf{W}$ and the activation function σ to different spatial-based graph filters, as follows:

$$\mathbf{F}' = \sigma(\mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}\mathbf{F}\mathbf{W}) \text{ or } \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{F}\mathbf{W}),\ \mathbf{F} = [\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_d]$$

3 Spectral-Based Graph Neural Networks

3.1 Fourier Transform

Given a 1D or 2D signal, the time domain (red) and frequency domain (blue) are its two expression forms with identical information, which can be analogous to general and polar coordinate systems.

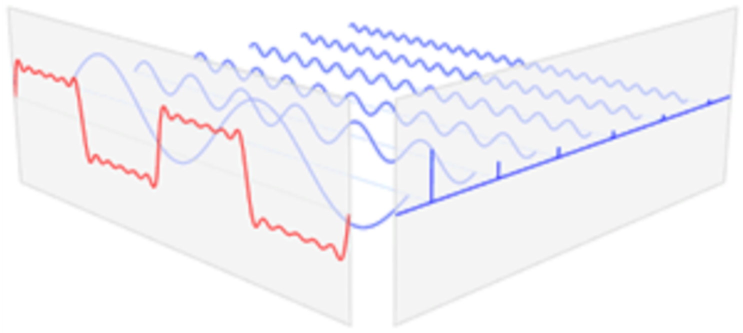


Figure 17: The time and frequency domains of a continuous temporal signal.

For example, for the sinusoidal function g(t) = sin(t), we draw it in the time domain as follows:

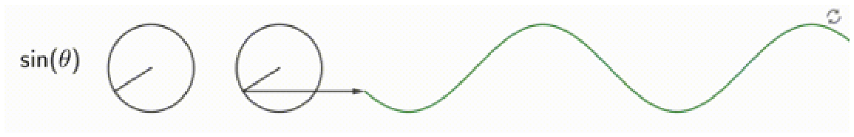


Figure 18: The time domain of the sinusoidal function.

We calculate its frequency by $\omega = \frac{1}{T} = \frac{1}{2\pi}$ and obtain it in the frequency domain:

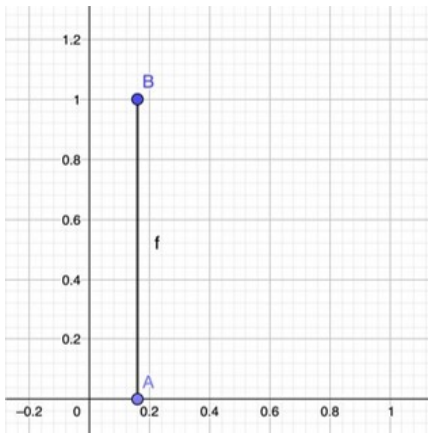


Figure 19: The frequency domain of the sinusoidal function.

As the bridge between the time and frequency domains, the classic Fourier Transform:

$$\hat{f}(\xi) = < f(t), exp(-2\pi it\xi) > = \int_{-\infty}^{\infty} f(t)exp(-2\pi it\xi)\mathrm{d}t$$

decomposes a signal $f(t)$ into a series of complex exponentials $exp(-2\pi it\xi)$ for any real number ξ, where ξ can be viewed as the frequency of the corresponding exponential.

3.2 Spectral Domain of Graph

Given an unweighted undirected graph, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, we aim to encode this graph with minimal information loss. To this end, we need to find a mapping function $z(v_i) = z_i$ to encode each node and ensure that the generated encoding vectors can reflect the structural information of the graph most accurately, i.e., the embeddings of adjacent nodes should be as close as possible in the vector space. We can formulate our goal as an optimization function as follows:

$$\min \frac{1}{2}\sum_{i}^{n}\sum_{j}^{n} A_{i,j}(z_i - z_j)^2 \quad \Rightarrow \quad \min\ \mathbf{z}^T\mathbf{L}\mathbf{z}$$

To avoid the solution being degenerate, e.g., all of the nodes may be mapped to 0, and to fix the scale of the node embedding overall, we impose the following restriction on $\mathbf{z}$:

$$\sum_{i}^{n} z_i^2 = \|\mathbf{z}\|^2 = 1$$

Even with this restriction, another degenerate solution is possible: each node is mapped to $\frac{1}{\sqrt{n}}$. To prevent this from happening, we add an additional restriction that

$$\sum_{i}^{n} z_i = \mathbf{z}^T\mathbf{1} = 0, \text{ i.e., } \mathbf{z} \perp \mathbf{1}$$

Thus, the original optimization problem is equivalent to

$$\min \frac{\mathbf{z}^T\mathbf{L}\mathbf{z}}{\mathbf{z}^T\mathbf{z}} \quad s.t.\ \mathbf{z}^T\mathbf{1} = 0$$

Let $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n$ are the normalized eigenvectors of a matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ and let $\lambda_1, \lambda_2, \cdots, \lambda_n$ are the corresponding eigenvalues. We consider a matrix $\mathbf{U}$ whose columns are $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n$ and obtain that

$$\mathbf{M}\mathbf{U} = \mathbf{M}[\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n] = [\lambda_1\mathbf{u}_1, \lambda_2\mathbf{u}_2, \cdots, \lambda_n\mathbf{u}_n] = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n]\mathbf{\Lambda} = \mathbf{U}\mathbf{\Lambda}$$

where $\mathbf{\Lambda}$ is a diagonal matrix whose diagonal elements are the eigenvalues of $\mathbf{M}$.

If $\mathbf{U}^{-1}$ exists, then we can write:

$$\mathbf{M} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1} \quad \text{[eigenvalue decomposition]}$$

$$\mathbf{U}^{-1}\mathbf{M}\mathbf{U} = \mathbf{\Lambda} \quad \text{[diagonalization of } \mathbf{M}\text{]}$$

Let $\mathbf{M}$ is the Laplacian matrix $\mathbf{L}$. $\mathbf{L}$ is a real symmetric matrix, thus:

$$\mathbf{u}_i^T\mathbf{u}_j = 1 \ \Rightarrow\ \mathbf{U}^T\mathbf{U} = \mathbf{I} \ \Rightarrow\ \mathbf{U}^{-1} = \mathbf{U}^T \ \Rightarrow\ \mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

For convenience, we rearrange the eigenvalues in non-decreasing order as $0 = \lambda_1 \leq \lambda_2 \leq, \cdots, \leq \lambda_n$.

$$\lambda_l = \lambda_l\mathbf{u}_l^T\mathbf{u}_l = \mathbf{u}_l^T\lambda_l\mathbf{u}_l = \mathbf{u}_l^T\mathbf{L}\mathbf{u}_l \geq 0$$

We rewrite the optimization problem as:

$$\min \frac{\mathbf{z}^T\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{z}}{\mathbf{z}^T\mathbf{z}} \quad s.t.\ \mathbf{z}^T\mathbf{1} = 0$$

Through the calculation, the optimized result is the smallest non-zero eigenvalue λ_2 and its corresponding eigenvector $\mathbf{u}_2$ is the optimal one-dimensional encoding vector of the graph $\mathcal{G}$.

We extend the optimization problem to the k-dimensional situation, i.e.,

$$\mathbf{z}(v_i) = \mathbf{z}_i = (z^1(v_i), z^2(v_i), \cdots, z^k(v_i))$$

We can obtain the k-dimensional optimal encoding of the graph $\mathcal{G}$:

$$\mathbf{Z} = [\mathbf{u_1}, \mathbf{u_2}, \cdots, \mathbf{u_k}] \in \mathbb{R}^{n \times k}$$

Thus, the eigenvectors of the Laplacian matrix can be regarded as a group of basis signals that exactly express the topological information of the graph. The vector space consisting of the eigenvectors of the Laplacian matrix is the "frequency domain" (or called "spectral domain") of the graph signal.

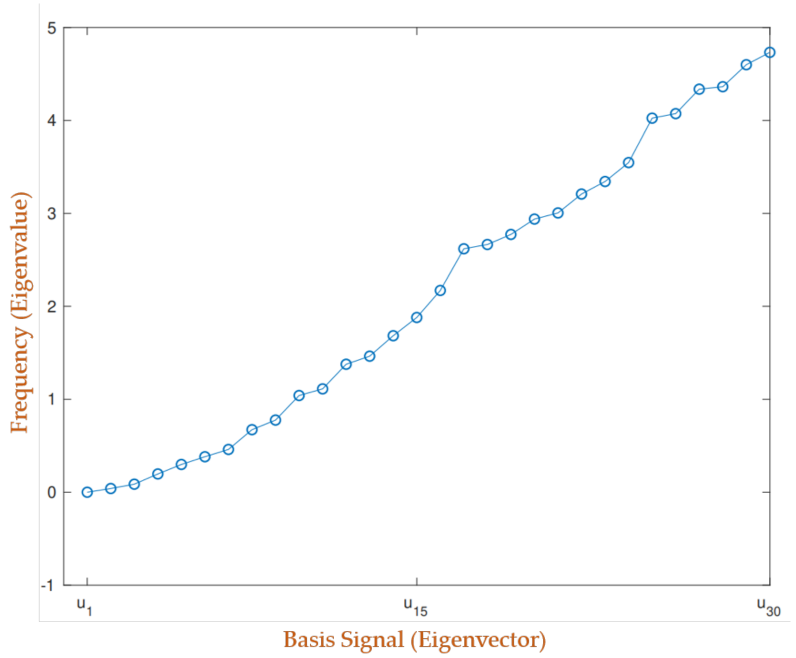


Figure 20: Spectral domain.

3.3 Graph Fourier Transform

In classic signal processing, we transform a signal from the time domain to the frequency domain via Fourier Transform:

$$\hat{f}(\xi) = \int f(t)e^{-2\pi i\xi t}\,\mathrm{d}t = < f(t), e^{-2\pi i\xi t} >$$

Further, we can consider $e^{-2\pi i\xi t}$ as the "eigenvector" of the Laplacian operator Δ, i.e.,

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

Thus, we can transform the graph signal from the spatial domain to the spectral domain by extending Fourier Transform to the graph, i.e., Graph Fourier Transform (GFT):

$$\hat{\mathbf{f}}[l] = < \mathbf{f}, \mathbf{u}_l > = \sum_{i=1}^{n} \mathbf{f}[i]\mathbf{u}_l[i] = \mathbf{u}_l^T\mathbf{f} \quad \Rightarrow \quad \hat{\mathbf{f}} = \mathbf{U}^T\mathbf{f}$$

In this process, the Graph Fourier Transform decomposes the graph signal $\mathbf{f}$ into graph Fourier bases with different frequencies/eigenvalues. The obtained coefficient $\hat{\mathbf{f}}[l]$ denotes how many components the corresponding graph Fourier basis $\mathbf{u}_l$ contributes to the graph signal.

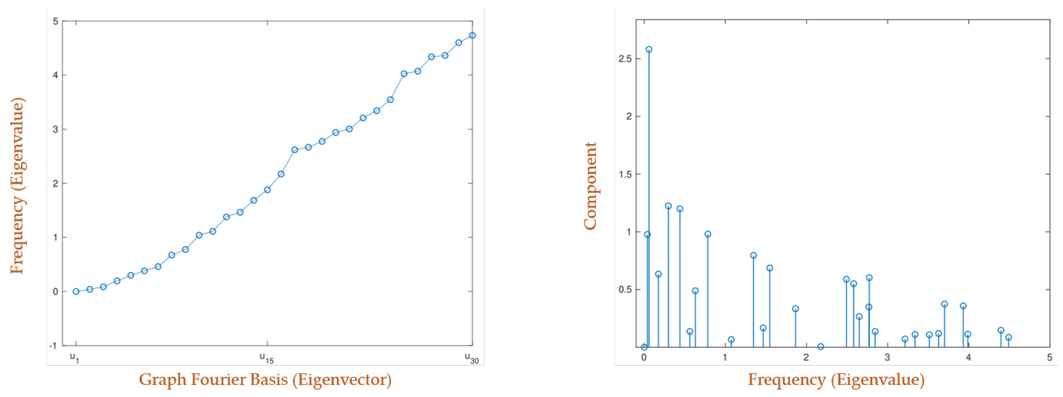
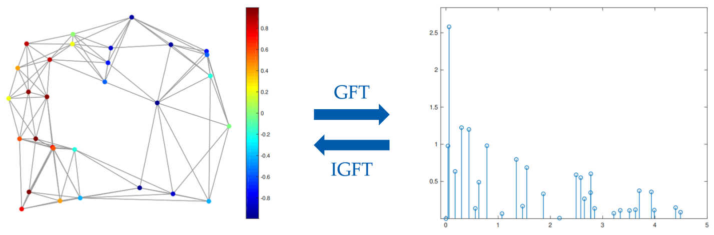


Figure 21: Signal decomposition via Graph Fourier Transform.

There is also Inverse Graph Fourier Transform (IGFT), which can transform the spectral representation $\hat{\mathbf{f}}$ to the spatial representation $\mathbf{f}$ as:

$$\mathbf{f}[i] = \sum_{l=1}^{n} \mathbf{u}_l[i]\hat{\mathbf{f}}[l] \quad \Rightarrow \quad \mathbf{f} = \mathbf{U}\hat{\mathbf{f}}$$



(a) A graph signal in the spatial domain (b) A graph signal in the spectral domain

Figure 22: Graph Fourier Transform (GFT) and Inverse Graph Fourier Transform (IGFT).

Actually, we can understand GFT with the help of the light spectrum:

- **Glass Prism**: Graph Laplacian $\mathbf{L}$.
- **Polychromatic Input Light**: Spatial Representation $\mathbf{f}$ of Graph Signal.
- **Decomposed Monochromatic Lights**: Eigenvectors $\{\mathbf{u}_l\}_{l=1}^{n}$ of Graph Laplacian.
- **Light Spectrum**: Eigenvector Space (Spectral Domain).
- **Strength of Decomposed Monochromatic Lights**: Components $\{\hat{\mathbf{f}}[l]\}_{l=1}^{n}$ of Graph Signal on Different Eigenvectors.

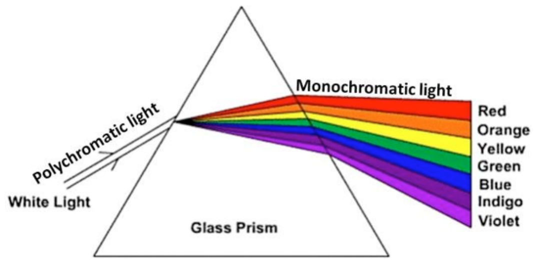


Figure 23: Spectrum.

3.4 Spectral-Based Graph Filter

To modulate the frequencies of the graph signal $\mathbf{f}$, we filter the graph Fourier coefficients as follows:

$$\hat{\mathbf{f}}'[l] = \hat{\mathbf{f}}[l] \cdot \gamma(\lambda_l),\ l = 1, 2, \cdots, n$$

where $\gamma(\lambda_l)$ is a filter function with the frequency λ_l as input which determines how many corresponding frequency components should be modulated/filtered/kept/amplified/removed/diminished.

For example, in the extreme case, if $\gamma(\lambda_l)$ equals to 0, then, $\hat{\mathbf{f}}'[l] = 0$ and the frequency components on the $\mathbf{u}_l$ basis are removed from the graph signal $\mathbf{f}$.

This process can be expressed in a matrix form as follows:

$$\hat{\mathbf{f}}' = \gamma(\mathbf{\Lambda}) \cdot \hat{\mathbf{f}} = \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T\mathbf{f},\ \ \mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix},\ \ \gamma(\mathbf{\Lambda}) = \begin{bmatrix} \gamma(\lambda_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \gamma(\lambda_n) \end{bmatrix}$$

With the filtered coefficients, we can reconstruct the graph signal to the spatial domain using the Inverse Graph Fourier Transform as follows:

$$\mathbf{f}' = \mathbf{U}\hat{\mathbf{f}}' = \mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T\mathbf{f}$$

The filtering process can be regarded as applying the convolution kernel/filter $\mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T$ to the input graph signal $\mathbf{f}$ in the spatial domain, where $\gamma(\mathbf{\Lambda})$ is the convolution kernel/filter in the spectral domain.

If we know how we want to modulate the frequencies in the input graph signal, we can design the filter function $\gamma(\lambda)$ in a corresponding way. However, in real scenarios, we often do not know which frequencies are more important. Hence, just like the classical neural networks, we can model $\gamma(\lambda)$ as the learnable parameter that can be trained under the supervision of data, as follows:

$$\gamma(\lambda_l) = \theta_l \ \ \Rightarrow \ \ \gamma(\mathbf{\Lambda}) = \begin{bmatrix} \theta_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \theta_n \end{bmatrix}$$

4 Conclusion

In this paper, we present an intuitive way to understand the principles of GNNs. Starting from traditional signal processing, we discuss graph filters in the spatial and spectral domains, respectively. In the future, we will explain GNNs from other interesting perspectives and explore advanced approaches and applications of GNNs.