Introduction of Graph Neural Network

Personal Reading Notes

Jianglin Lu

Department of ECE
College of Engineering
Northeastern University
360 Huntington Avenue, Boston, MA 02115, USA
https://jianglin954.github.io/
jianglinlu@outlook.com

Outline

Outline

Preliminaries—Why Graph Neural Network

- While deep learning effectively captures hidden patterns of Euclidean data, there is an increasing number of applications where data are represented in the form of graphs.
- As graphs can be irregular, a graph may have a variable size of unordered nodes, and nodes from a graph may have a different number of neighbors, resulting in some important operations (e.g., convolutions) being easy to compute in the image domain, but difficult to apply to the graph domain.
- A core assumption of existing machine learning algorithms is that instances are independent of each other. This assumption no longer holds for graph data because each instance (node) is related to others by links of various types.

Preliminaries—Spectral Graph Theory

Graph Laplacian

Given a undirected and connected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathbf{A}\}$, where $\mathcal{V}$ is a finite set of $|\mathcal{V}| = N$, $\mathcal{E}$ is a set of edges, and $\mathbf{A} \in \mathbb{R}^{N \times N}$ is a weighted adjacency matrix encoding the connection weight between two vertices. An essential operator in spectral graph analysis is the *graph Laplacian*, which combinatorial definition is:

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{1}$$

where $\mathbf{D}$ is a diagonal degree matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$, and normalized definition is:

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

where $\mathbf{I}_N$ is the identity matrix.

Preliminaries—Spectral Graph Theory

As $\mathbf{L}$ is a real symmetric positive semidefinite matrix, it has complete set of orthonormal eigenvectors $\{\mathbf{u}_l\}_{l=0}^{N-1}$, known as the graph Fourier modes, and their associated ordered real nonnegative eigenvalues $\{\lambda_l\}_{l=0}^{N-1}$. The Laplacian is indeed diagonalized by the Fourier basis $\mathbf{U} = [\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}] \in \mathbb{R}^{N \times N}$ such that:

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \tag{3}$$

where $\mathbf{\Lambda} = diag([\lambda_0, \ldots, \lambda_{N-1}]) \in \mathbb{R}^{N \times N}$.

Preliminaries—Spectral Graph Theory

Graph Fourier Transform

For a signal $\mathbf{f} \in \mathbb{R}^N$ defined on a graph $\mathcal{G}$, its *Graph Fourier Transform* is defined as follows:

$$\hat{\mathbf{f}} = \mathbf{U}^T\mathbf{f} \tag{4}$$

where $\mathbf{U}$ consists of eigenvectors of the Laplacian matrix of $\mathcal{G}$ and $\hat{\mathbf{f}}$ is the obtained graph Fourier coefficients for the signal $\mathbf{f}$.

There graph Fourier coefficients describe how each graph Fourier component contributes to the graph signal $\mathbf{f}$. Specifically, the i-th element of $\hat{\mathbf{f}}$ corresponds to the i-th graph Fourier component $\mathbf{u}_i$ with the frequency λ_i, where λ_i is the eigenvalue corresponding to $\mathbf{u}_i$.

Preliminaries—Spectral Graph Theory

To modulate the frequencies of the signal **f**, we filter the graph Fourier coefficients as follows:

$$\hat{\mathbf{f}}'[i] = \hat{\mathbf{f}}[i] \cdot \gamma(\lambda_i), \quad \textit{for} \quad i = 1, \dots, N. \tag{5}$$

where $\gamma(\lambda_i)$ is a function with the frequency λ_i as input which determines how the corresponding frequency component should be modulated. In matrix form, we have:

$$\hat{\mathbf{f}}' = \gamma(\Lambda) \cdot \hat{\mathbf{f}} = \gamma(\boldsymbol{\Lambda}) \cdot \mathbf{U}^T \mathbf{f}, \tag{6}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix consisting of the frequencies (eigenvalues of the Laplacian matrix).

Preliminaries—Spectral Graph Theory

Inverse Graph Fourier Transform

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

$$\mathbf{f}' = \mathbf{U}\hat{\mathbf{f}'} = \mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T \mathbf{f}, \tag{7}$$

where $\mathbf{f}'$ is the obtained filtered graph signal.

The filtering process can be regarded as applying the operator $\mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T$ to the input graph signal.

Outline

Spectral-based Graph Filters

If we know how we want to modulate the frequencies in the input signal, we can design the function $\gamma(\lambda)$ in a corresponding way. However, when utilizing the spectral-based filter as a graph filter in graph neural networks, we often do not know which frequencies are more important.

To solve this problem, we can model $\gamma(\mathbf{\Lambda})$ with certain functions and then learn the parameters with the supervision from data, by which the graph filters can be learned in a data-driven way.

Spectral-based Graph Filters

Bruna et al., Spectral Networks and Deep Locally Connected Networks on Graphs, arXiv 2013

A natural attempt is to give full freedom when designing $\gamma()$ (or a non-parametric model). Specifically, the function $\gamma()$ is defined as follows:

$$\gamma(\lambda_l) = \theta_l \tag{8}$$

where θ_l is a parameter to be learned from data.

Spectral-based Graph Filters

Drawbacks:

- The number of parameters to be learned is equal to the number of nodes N, which can be extremely large in real-world graphs. Hence, it requires lots of memory to store these parameters and also abundant data to fit them.
- The filter $\mathbf{U} \cdot \gamma(\boldsymbol{\Lambda}) \cdot \mathbf{U}^T$ is likely to be a dense matrix. Therefore, the calculation of the i-th element of the output signal $\mathbf{f}'$ could relate to all the nodes in the graph. In other words, the operator is not spatially localized.
- The computational cost is quite expensive due to the eigendecomposition of the Laplacian matrix and the matrix multiplication between dense matrices when calculating $\mathbf{U} \cdot \gamma(\boldsymbol{\Lambda}) \cdot \mathbf{U}^T$.

Spectral-based Graph Filters

Defferrard et al., Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, NIPS 2016

To address the above-mentioned issues, a polynomial filter operator is proposed, in which the function $\gamma()$ can be modeled with a K-order truncated polynomial as follows:

$$\gamma(\lambda_l) = \sum_{k=0}^{K-1} \theta_k \lambda_l^k \quad or \quad \gamma(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k \tag{9}$$

where θ_l is a parameter to be learned from data.

Spectral-based Graph Filters

With this polynomial filter, we can get the output $\mathbf{f}'$ as follows:

$$\begin{aligned}\mathbf{f}' &= \mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T\mathbf{f} = \mathbf{U} \cdot \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k \cdot \mathbf{U}^T\mathbf{f} \\ &= \sum_{k=0}^{K-1} \theta_k \mathbf{U} \cdot \mathbf{\Lambda}^k \cdot \mathbf{U}^T\mathbf{f} = \sum_{k=0}^{K-1} \theta_k \mathbf{U} \cdot (\mathbf{\Lambda}\mathbf{U}^T\mathbf{U})^k \cdot \mathbf{U}^T\mathbf{f} \\ &= \sum_{k=0}^{K-1} \theta_k \underbrace{(\mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^T) \cdots (\mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^T)}_{k} \cdot \mathbf{f} \\ &= \sum_{k=0}^{K-1} \theta_k \mathbf{L}^k \mathbf{f}\end{aligned} \tag{10}$$

Spectral-based Graph Filters

Lemma

Let $\mathcal{G}$ be a graph and $\mathbf{L}$ be its Laplacian matrix. Then, the i,j-th element of the k-th power of the Laplacian matrix $\mathbf{L}_{ij}^{k} = 0$ if $dis(v_i, v_j) > k$, where $dis()$ is the shortest path distance.

- The polynomials of the Laplacian matrix are all sparse.
- Spectral filters represented by K^{th}-order polynomials of the Laplacian are exactly K-localized (only involves K-hop neighborhoods).
- The learning complexity is $\mathcal{O}(K)$, the support size of the filter, and thus the same complexity as classical CNNS.

Spectral-based Graph Filters

Merits:

- $\mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^T$ can be simplified to be a polynomial of the Laplacian matrix, meaning that no eigendecomposition is needed and the polynomial parametrized filtering operator is spatially localized, i.e., the calculation of each element of the output $\mathbf{f}'$ only involves a small number of nodes in the graph.

Drawbacks:

- The basis of the polynomial $(1, x, x^2, \ldots)$ is not orthogonal to each other. Hence, the coefficients are dependent on each other, making them unstable under perturbation during the learning process. In other words, an update in one coefficient may lead to changes in other coefficients.

Spectral-based Graph Filters

To address the above-mentioned issues, we first give the definition of Chebyshev polynomial.

Chebyshev Polynomial

Chebyshev polynomial $T_k(x)$ of order k can be computed by the stable recurrence relation:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \tag{11}$$

with $T_0 = x$ and $T_1 = x$. These polynomials form an orthogonal basis for $L^2([-1,1], dy/\sqrt{1-y^2})$, the Hilbert space of square integrable functions with respect to the measure $dy/\sqrt{1-y^2}$.

Spectral-based Graph Filters

With the Chebyshev polyomial, the filter can be parametrized as the truncated expansion:

$$\gamma(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\mathbf{\Lambda}}) \tag{12}$$

where $\theta \in \mathbb{R}^K$ is a vector of Chebyshev coefficients, and $T_k(\widetilde{\mathbf{\Lambda}}) \in \mathbb{R}^{N \times N}$ is the Chebyshev polynomial of order k evaluated at $\widetilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{max} - \mathbf{I}_N$, a diagonal matrix of scaled eigenvalues that lie in $[-1, 1]$.

Spectral-based Graph Filters

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

$$\begin{aligned}\mathbf{f}' &= \mathbf{U} \cdot \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\boldsymbol{\Lambda}}) \cdot \mathbf{U}^T \mathbf{f} \\ &= \sum_{k=0}^{K-1} \theta_k \mathbf{U} \cdot T_k(\widetilde{\boldsymbol{\Lambda}}) \cdot \mathbf{U}^T \mathbf{f} \\ &= \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\mathbf{L}}) \mathbf{f}\end{aligned} \tag{13}$$

where $T_k(\widetilde{\mathbf{L}}) \in \mathbf{R}^{N \times N}$ is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $\widetilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I}_N$.

Spectral-based Graph Filters

N. Kipf et al., Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

A simplified version of Chebyshev filter is named GCN filter is proposed, which sets the order of Chebyshev polynomials to $K = 1$ and approximates $\lambda_{max} = 2$ (i.e., $\widetilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{max} - \mathbf{I}_N = \mathbf{\Lambda} - \mathbf{I}_N$):

$$\gamma(\mathbf{\Lambda}) = \theta_0 T_0(\widetilde{\mathbf{\Lambda}}) + \theta_1 T_1(\widetilde{\mathbf{\Lambda}}) = \theta_0 \mathbf{I}_N + \theta_1 \widetilde{\mathbf{\Lambda}} = \theta_0 \mathbf{I}_N + \theta_1(\mathbf{\Lambda} - \mathbf{I}_N) \tag{14}$$

Applying the GCN filter on a graph signal $\mathbf{f}$, we have:

$$\begin{aligned} \mathbf{f}' = \mathbf{U}\gamma(\mathbf{\Lambda})\mathbf{U}^T\mathbf{f} \qquad &= \theta_0 \mathbf{U}\mathbf{U}^T\mathbf{f} + \theta_1 \mathbf{U}(\mathbf{\Lambda} - \mathbf{I}_N)\mathbf{U}^T\mathbf{f} \\ = \theta_0\mathbf{f} + \theta_1(\mathbf{L} - \mathbf{I}_N)\mathbf{f} \quad &= \theta_0\mathbf{f} + \theta_1(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{f} \end{aligned} \tag{15}$$

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

Spectral-based Graph Filters

By setting $\theta = \theta_0 = -\theta_1$, we have:

$$\mathbf{f}' = \theta_0 \mathbf{f} + \theta_1 (\mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}) \mathbf{f} = \theta (\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}) \mathbf{f} \tag{16}$$

Note that, $\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ 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 renormalization trick:

$$\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \rightarrow \widetilde{\mathbf{D}}^{-\frac{1}{2}} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-\frac{1}{2}} \tag{17}$$

where $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ and $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$.

Spectral-based Graph Filters

The final GCN filter after these simplification is defined as:

$$\mathbf{f}' = \theta \widetilde{\mathbf{D}}^{-\frac{1}{2}} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{f} \tag{18}$$

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

Spectral-based Graph Filters

Generalization to Multi-channel Graph Signals:

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

$$\mathbf{F}' = \widetilde{\mathbf{D}}^{-\frac{1}{2}} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{F} \boldsymbol{\Theta} \tag{19}$$

where $\Theta \in \mathbb{R}^{C \times F}$ is a matrix of filter parameters and $\mathbf{F}'$ is the convolved signal matrix.

Outline

Spatial-based Graph Filters

Hamilton et al., Inductive Representation Learning on Large Graphs, NIPS 2017

The GraphSAGE model introduces a spatial based filter, which is based on aggregating information from neighboring nodes. For a single node v_i, the process to generate its new features can be formulated as:

$$\begin{aligned} \mathcal{N}_S(v_i) &= SAMPLE(\mathcal{N}(v_i), S) \\ \mathbf{f}'_{\mathcal{N}_S(v_i)} &= AGGREGATE(\{\mathbf{F}_j, \forall v_j \in \mathcal{N}_S(v_i)\}) \\ \mathbf{F}'_i &= \sigma\left(\left[\mathbf{F}_i, \mathbf{f}'_{\mathcal{N}_S(v_i)}\right]\boldsymbol{\Theta}\right) \end{aligned} \tag{20}$$

where $SAMPLE()$ is a function that takes a set as input and randomly samples S elements from the input as out, $AGGREGATE()$ is a function to combine the information from the neighboring nodes where $\mathbf{f}'_{\mathcal{N}_S(v_i)}$ denotes the output of the $AGGREGATE()$ function, and $[\cdot,\cdot]$ is the concatenation operation.

Spatial-based Graph Filters

There are several $AGGREGATE()$ functions as below:

- Mean aggregator, which is to simply take element-wise mean of the vectors in $\{\mathbf{F}_j, \forall v_j \in \mathcal{N}_S(v_i)\}$.
- LSTM aggregator, which is to treat the set of the sampled neighboring nodes $\mathcal{N}_S(v_i)$ of node v_i as a sequence and utilize the LSTM architecture to process the sequence. Since there is no natural order among the neighbors, a random ordering is adopted.
- Pooling aggregator, which adopts the max pooling operation to summarize the information from the neighboring nodes.

Spatial-based Graph Filters

Velickovic et al., Graph Attention Networks, ICLR 2018

Motivation:

- In all of the aforementioned spectral approaches, the learned filters depend on the Laplacian eigenbasis, which depends on the graph structure. Thus, a model trained on a specific structure cannot be directly applied to graph with a different structure.
- In the spatial approach, GraphSAGE needs to sample a fixed-size neighborhood of each node in order to keep its computational footprint consistent. This does not allow it access to the entirety of the neighborhood while performing inference.
- One of the benefits of attention mechanisms is that they allow for dealing with variable sized inputs, focusing on the most relevant parts of the input to make decisions.

Spatial-based Graph Filters

When generating the new features for a node v_i, GAT attends to all its neighbors to generate an importance score for each neighbor. Specifically, the importance score of node $v_j \in \mathcal{N}(v_i) \cup \{v_i\}$ to the node v_i can be calculated as follows:

$$e_{ij} = a\left(\mathbf{F}_i\boldsymbol{\Theta}, \mathbf{F}_j\boldsymbol{\Theta}\right) \tag{21}$$

where $\boldsymbol{\Theta}$ is a shared parameter matrix and $a()$ is a shared attention function.

To make the importance scores easily comparable across different nodes, we need to normalize them across all choices of j using the softmax function:

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k\in\mathcal{N}_i} exp(e_{ik})} \tag{22}$$

Spatial-based Graph Filters

The attention function $a()$ is a single-layer feedforward network, parametrized by a weight vector $\mathbf{a}$ and applying the *LeakyReLU* nonlinearity, as follows:

$$\alpha\left(\mathbf{F}_i\boldsymbol{\Theta}, \mathbf{F}_j\boldsymbol{\Theta}\right) = LeakyReLU(\mathbf{a}^T[\mathbf{F}_i\boldsymbol{\Theta}, \mathbf{F}_j\boldsymbol{\Theta}]) \tag{23}$$

where $[\cdot, \cdot]$ denotes the concatenation operation, $\mathbf{a}$ is a parametrized vector and *LeakyReLU* is the nonlinear activation function.

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k\in\mathcal{N}_i} exp(e_{ik})} = \frac{exp\left(LeakyReLU\left(\mathbf{a}^T[\mathbf{F}_i\boldsymbol{\Theta}||\mathbf{F}_j\boldsymbol{\Theta}]\right)\right)}{\sum_{k\in\mathcal{N}_i} exp\left(LeakyReLU\left(\mathbf{a}^T[\mathbf{F}_i\boldsymbol{\Theta}||\mathbf{F}_k\boldsymbol{\Theta}]\right)\right)} \tag{24}$$

where $||$ is the concatenation operation.

Spatial-based Graph Filters

With the normalized importance scores, the new representation $\mathbf{F}'_i$ of node v_i can be computed as:

$$\mathbf{F}'_i = \sigma\left(\sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} \alpha_{ij} \mathbf{F}_j \mathbf{\Theta}\right) \tag{25}$$

To stabilize the learning process of self-attention, we employ multi-head attention. Specifically, K independent attention mechanisms execute the transformation of (25) and then their features are concatenated, resulting in the following output feature representation:

$$\mathbf{F}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} \alpha_{ij}^{k} \mathbf{F}_j \mathbf{\Theta}^{k}\right) \tag{26}$$

where $||$ represents concatenation.

Spatial-based Graph Filters

If we perform multi-head attention on the final layer of the network, concatenation is no longer sensible—instead, we employ averaging, and delay applying the final nonlinearity:

$$\mathbf{F}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{v_j\in\mathcal{N}(v_i)\cup\{v_i\}}\alpha_{ij}^{k}\mathbf{F}_j\mathbf{\Theta}^{k}\right) \tag{27}$$

Spatial-based Graph Filters

Simonovsky et al., Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs, CVPR 2017

Motivation:

- There may be edge information available in the graph. However, the current formulations of graph convolution do not exploit edge labels.
- We formulate a convolution-like operation on graph signals performed in the spatial domain where filter weights are conditioned on edge labels and dynamically generated for each specific input sample.

Spatial-based Graph Filters

For a given edge (v_i, v_j), we use $tp(v_i, v_j)$ to denote its type. Then the edge-conditioned convolution (ECC filter) is defined as:

$$\mathbf{F}'_i = \frac{1}{|\mathcal{N}(v_i)|} \sum_{v_j \in \mathcal{N}(v_i)} \mathbf{F}_j \mathbf{\Theta}_{tp(v_i, v_j)} \tag{28}$$

where $\mathbf{\Theta}_{tp(v_i,v_j)} = f(\mathbf{L}_{ji}; \omega)$, $f()$ is parameterized by a learnable network with weights ω, and $\mathbf{L}$ is the edge label matrix.

Spatial-based Graph Filters

Monti et al., Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs, CVPR 2017

The mixture model networks (MoNet) is a general framework allowing to design convolutional deep architectures on non-Euclidean domains such as graphs and manifolds. For each neighbor $v_j \in \mathcal{N}(v_i)$ of a center node v_i, a vector of pseudo-coordinate is introduced to denote the relevant relation between nodes v_j and v_i with their degrees as:

$$\mathbf{u}(v_i, v_j) = \left(\frac{1}{\sqrt{d_i}}, \frac{1}{\sqrt{d_j}} \right) \tag{29}$$

where d_i and d_j denote the degree of nodes v_i and v_j, respectively.

Spatial-based Graph Filters

Then, a Gaussian kernel is applied on the pseudo-coordinate to measure the relation between the two nodes as

$$\alpha_{ij} = exp\left(-\frac{1}{2}(\mathbf{u}(v_i, v_j) - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{u}(v_i, v_j) - \boldsymbol{\mu})\right) \tag{30}$$

where $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ are learnable covariance matrix and mean vector of a Gaussian kernel, respectively. The aggregation process is as:

$$\mathbf{F}_i' = \sum_{v_j \in \mathcal{N}(v_i)} \alpha_{ij} \mathbf{F}_j \tag{31}$$

Typically, a set of K kernels with different means and covariances are adopted, resulting in:

$$\mathbf{F}_i' = \sum_{k=1}^{K} \sum_{v_j \in \mathcal{N}(v_i)} \alpha_{ij}^k \mathbf{F}_j \tag{32}$$

where α_{ij}^k is from the k-th Gaussian kernel.

Spatial-based Graph Filters

Gilmer et al., Neural Message Passing for Quantum Chemistry, ICML 2017

Many spatial-based graph filters can be formulated in the Message Passing Neural Networks (MPNN). For a node v_i, the MPNN filter updates its features as follows:

$$\begin{aligned} \mathbf{m}_i &= \sum_{v_j \in \mathcal{N}(v_i)} M(\mathbf{F}_i, \mathbf{F}_j, \mathbf{e}_{(v_i, v_j)}) \\ \mathbf{F}'_j &= U(\mathbf{F}_i, \mathbf{m}_i) \end{aligned} \tag{33}$$

where $M()$ is the message function that generates the messages to pass to node v_i from its neighbors, $U()$ is the update function that updates the features of node v_i by combing the original features and the aggregated message from its neighbors, and $\mathbf{e}_{(v_i, v_j)}$ is edge features if available.

Comparison between Spectral and Spatial Graph Filters

Wu et al., A Comprehensive Survey on Graph Neural Networks, TNNLS 2021

Spectral models have a theoretical foundation in graph signal processing. However, spatial models are preferred over spectral models because of:

- Efficiency: spectral models either need to perform eigenvector computation or handle the whole graph at the same time. Spatial models are more scalable to large graphs as they directly perform convolutions in the graph domain via information propagation. The computation can be performed in a batch of nodes instead of the whole graph.

Comparison between Spectral and Spatial Graph Filters

- Generality: spectral models which rely on a graph Fourier basis generalize poorly to new graphs. They assume a fixed graph. Any perturbations to a graph would result in a change of eigenbasis. Spatial-based models perform graph convolutions locally on each node where weights can be easily shared across different locations and structures.
- Flexibility: spectral models are limited to operate on undirected graphs. Spatial models are more flexible to handle multi-source graph inputs such as edge inputs, directed graphs, signed graphs, and heterogeneous graphs, because these graph inputs can be incorporated into the aggregation function easily.

Outline

Graph Pooling

- The graph filters refine the node features without changing the graph structure. Typically, the graph filter operations are sufficient for node-focused tasks. However, for graph-based tasks, a representation of the entire graph is desired.
- There are two main kinds of information that are important for generating the graph representation, including node features and graph structure. The graph representation is expected to preserve both the node feature information and the graph structure information.
- Similar to the classical convolutional neural network, graph pooling layers are proposed to generate graph level representations.

Graph Pooling

In general, there are two kinds of graph pooling layers:

- Flat graph pooling: which generates the graph-level representation directly from the node representations in a single step. For example, the mean/max/sum pooling layers can be adapted to GNN by applying them to each feature channel.
- Hierarchical graph pooling: which summarizes the graph information by coarsening the original graph step by step. In this design, there are often several graph pooling layers, each of which follows a stack of several filters.

Graph Pooling—Flat Graph Pooling

Wu et al., A Comprehensive Survey on Graph Neural Networks, TNNLS 2021

The mean/max/sum pooling is the most primitive and effective way to implement down-sampling since calculating the mean/max/sum value in the pooling window is fast:

$$\mathbf{f}_G = mean/max/sum(\mathbf{f}_1^K, \mathbf{f}_2^K, , \cdots, \mathbf{f}_N^K) \tag{34}$$

where K is the index of the last graph convolutional layer.

Graph Pooling—Flat Graph Pooling

Li et al., Gated Graph Sequence Neural Networks, ICLR 2016

An attention-based flat pooling operation utilizes an attention score to summarize the node representations for generating the graph representation. Specifically, the attention score for node v_i is computed as:

$$s_i = \frac{exp\left(h(\mathbf{F}_i)\right)}{\sum_{v_j \in \mathcal{N}(v_i)} exp\left(h(\mathbf{F}_j)\right)} \tag{35}$$

where h is a feedforward network on map $\mathbf{F}_i$ to a scalar. With the learned attention scores, the graph representation can be summarized from the node representations as:

$$\mathbf{f}_G = \sum_{v_j \in \mathcal{N}(v_i)} s_i \cdot tanh\left(\mathbf{F}_i \mathbf{\Theta}_i\right) \tag{36}$$

where $\mathbf{\Theta}_i$ are parameters to be learned.

Graph Pooling—Hierarchical Graph Pooling

The aforementioned pooling methods mainly consider graph features and usually ignore the hierarchical graph structural information. Hierarchical graph pooling layers aim to preserve such information by coarsening the graph step by step until the graph representation is achieved. There are roughly two ways to coarsen the graph:

- Downsampling-based graph pooling: selects the most important nodes as the nodes for the coarsened graph.
- Supernode-based graph pooling: combines nodes in the input graph to form supernodes that serve as the nodes for the coarsened graph.

The main difference between the two ways is that the downsampling based methods keep nodes from the original graph while the supernode-based methods generate new nodes for the coarsened graph.

Graph Pooling—Hierarchical Graph Pooling

Downsampling-based graph pooling:

To coarsen the input graph, a set of N_{op} nodes are selected according to some importance measures, and then graph structure and node features for the coarsened graph are formed upon these nodes.

There are three key parts in a downsampling-based graph pooling layers:

- Develop the measure for downsampling.
- Generate graph structure for the coarsened graph.
- Generate node features for the coarsened graph.

Graph Pooling—Hierarchical Graph Pooling

Gao et al., Graph U-Nets, ICML 2019

In gPool layer, the importance measure for nodes is learned from the input node features $\mathbf{F}^{ip}$ as:

$$\mathbf{y} = \frac{\mathbf{F}^{ip}\mathbf{p}}{||\mathbf{p}||} \tag{37}$$

where $\mathbf{F}^{ip} \in \mathbb{R}^{N_{ip} \times d_{ip}}$ is the matrix of the input node features and $\mathbf{p} \in \mathbb{R}^{d_{ip}}$ is a vector to be learned to project the input features into importance scores. With the importance scores, we can rank all the nodes and select the N_{op} most important ones as:

$$idx = rank(\mathbf{y}, N_{op}) \tag{38}$$

where N_{op} is the number of nodes in the coarsened graph and idx denotes the indices of the selected top N_{op} nodes.

Graph Pooling—Hierarchical Graph Pooling

Then, the graph structure for the coarsened graph can be induced from the graph structure of the input graph as:

$$\mathbf{A}^{op} = \mathbf{A}^{ip}(idx, idx) \tag{39}$$

Similarly, the node features can also be extracted from the input node features. In addition, it adopts a gating system to control the information flow from the input features to the new features. Specifically, the selected nodes with a higher importance score can have more information flow to the coarsened graph, by:

$$\widetilde{\mathbf{y}} = \sigma(\mathbf{y}(idx)); \quad \widetilde{\mathbf{F}} = \mathbf{F}^{ip}(idx, :); \quad \mathbf{F}_p = \widetilde{\mathbf{F}} \odot \left(\widetilde{\mathbf{y}}\mathbf{1}_{d_{ip}}^T\right) \tag{40}$$

where $\sigma()$ is the sigmoid function mapping the importance score to $(0, 1)$.

Graph Pooling—Hierarchical Graph Pooling

Lee et al., Self-Attention Graph Pooling, ICML 2019

The gPool layer learns the importance measure for nodes solely based on the input features, and ignores the graph structure information. To solve this problem, SAGPool utilizes the GCN filter to learn the importance score:

$$\mathbf{y} = \sigma\left(GCNFilter(\mathbf{A}, \mathbf{F})\right) = \sigma\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{F}\boldsymbol{\Theta}_{att}\right) \tag{41}$$

where σ is an activation function, $\boldsymbol{\Theta}_{att} \in \mathbb{R}^{C\times 1}$ is the only parameter of the SAGPool, and $\mathbf{y}$ is a vector of self-attention score.

Graph Pooling—Hierarchical Graph Pooling

Supernode-based graph pooling:

In downsampling-based graph pooling, the information about the unselected nodes is lost as these nodes are discarded. Supernode-based pooling methods aim to coarsen the input graph by generating supernodes. Specifically, they try to learn to assign the nodes in the input graph into different clusters, where these clusters are treated as supernodes.

There are three key parts in a supernode-based graph pooling layers:

- Generate supernodes as the nodes for the coarsened graph.
- Generate graph structure for the coarsened graph.
- Generate node features for the coarsened graph.

Graph Pooling—Hierarchical Graph Pooling

Ying et al., Hierarchical Graph Representation Learning with Differentiable Pooling, NIPS 2018

DiffPool learns a soft assignment matrix from the nodes in the input graph to the supernodes by using a GCN filter as follows:

$$\mathbf{S}^{(l)} = softmax(GCNFilter_{(l),pool}(\mathbf{A}^{(l)}, \mathbf{F}^{(l)})) \tag{42}$$

where $\mathbf{S}^{(l)} \in \mathbb{R}^{N_l \times N_{l+1}}$ is a cluster assignment matrix at layer l, each row of $\mathbf{S}^{(l)}$ corresponds to one of the N_l nodes at layer l and each column of $\mathbf{S}^{(l)}$ corresponds to one of the N_{l+1} clusters (supernode) at the next layer $l+1$. The *softmax* function is applied row-wisely; hence, each row is normalized to have a summation of 1. The j-th element in the i-th row indicates the probability of assigning the i-th node to the j-th supernode.

Graph Pooling—Hierarchical Graph Pooling

With the assignment matrix **S**, the graph structure for the coarsened graph and the node features for the supernodes can be generated by:

$$\mathbf{A}^{(l+1)} = {\mathbf{S}^{(l)}}^T \mathbf{A}^{(l)} \mathbf{S}^{(l)}; \quad \mathbf{F}^{(l+1)} = {\mathbf{S}^{(l)}}^T \mathbf{Z}^{(l)} \tag{43}$$

where

$$\mathbf{Z}^{(l)} = GCNFilter_{(l),embed}(\mathbf{A}^{(l)}, \mathbf{F}^{(l)})$$

is the node embedding at layer l.

Note that, the two GCNFilters consume the same input data but have distinct parameterizations and play separate roles: $GCNFilter_{(l),embed}$ generates new embeddings for the input nodes at layer l, while the $GCNFilter_{(l),pool}$ generates a probabilistic assignment of the input nodes to clusters N_{l+1}.

Graph Pooling—Hierarchical Graph Pooling

Ma et al., Graph Convolutional Networks with EigenPooling, KDD 2019

EigenPooling generates the supernodes using spectral clustering methods, which first constructs a set of non-overlapping clusters (N_{op}) being regarded as the supernodes for the coarsened graph. Given the list of nodes in the k-th cluster Γ^k, a sampling operator $\mathbf{C}^k \in \{0,1\}^{N_{ip} \times N^k}$ is defined as:

$$\mathbf{C}_{ij}^k = 1 \quad \textit{if and only if} \quad \Gamma_j^k = v_i \tag{44}$$

where $\Gamma_j^k = v_i$ means node v_i corresponds to the j-th node in k-th cluster. Then, the adjacency matrix for the k-th cluster can be defined as:

$$\mathbf{A}^k = \mathbf{C}^{k^T} \mathbf{A}^{ip} \mathbf{C}^k \tag{45}$$

Graph Pooling—Hierarchical Graph Pooling

To form the graph structure between the supernodes, only the connections across the clusters in the original graph are considered. To this end, we first generate the intra-cluster adjacency matrix, which only consists of the edges within each cluster as:

$$\mathbf{A}_{intra} = \sum_{k=1}^{N_{op}} \mathbf{C}^k \mathbf{A}^k \mathbf{C}^{k^T} \tag{46}$$

Then, the inter-cluster adjacency matrix, which consists of the edges across the clusters, can be presented as $\mathbf{A}_{inter} = \mathbf{A} - \mathbf{A}_{intra}$, and the adjacency matrix for the coarsened graph can be obtain by:

$$\mathbf{A}^{op} = \mathbf{S}^T \mathbf{A}_{inter} \mathbf{S} \tag{47}$$

where $\mathbf{S} \in \mathbb{R}^{N_{ip} \times N_{op}}$ is an assignment matrix, which indicates whether a node belongs to a specific subgraph as:

$$\mathbf{S}_{ij} = 1 \quad \textit{if and only if} \quad v_i \in \Gamma^j \tag{48}$$

Outline

Graph Autoencoders

- Graph Autoencoders.
- Spatial-Temporal GNN.
- GNN on Complex Graphs.

The End!