A Comprehensive Survey on Graph Neural Networks [Review]

박태준

March 3, 2023

1 Introduction

- We use bold uppercase characters to denote matrices and bold lowercase characters to denote vectors.
- **COMMONLY USED NOTATIONS**
  - ▷ $|\cdot|$: The length of a set.
  - ▷ $\odot$: Element-wise product.
  - ▷ V: The set of nodes in a graph.
  - ▷ v: The node $v \in V$.
  - ▷ E: The set of edges in a graph.
  - ▷ e_{ij}: An edge $e_{ij} \in E$.
  - ▷ $N(v)$: The neighbors of a node v.
  - ▷ $\mathbf{A}$: The graph adjacency matrix.
  - ▷ $\mathbf{A}^\top$: The transpose of the matrix $\mathbf{A}$.
  - ▷ $\mathbf{A}^n, n \in \mathbb{Z}$: THe n^{th} power of $\mathbf{A}$.
  - ▷ $[\mathbf{A}, \mathbf{B}]$: The concatenation of $\mathbf{A}$ and $\mathbf{B}$.
  - ▷ $\mathbf{D}$: The degree matrix of $\mathbf{A}$. $\mathbf{D}_{ii}$Æ $\sum_{j=1}^{n} A_{ij}$.
  - ▷ n: The number of nodes, $n = |V|$.
  - ▷ m: The number of edges, $m = |E|$.

- ▷ d: The dimension of a node feature vector.
- ▷ b: The dimension of a hidden node feature vector.
- ▷ c: The dimension of an edge feature vector.
- ▷ $\mathbf{X} \in \mathbb{R}^{n \times d}$: The feature matrix of a graph.
- ▷ $\mathbf{x} \in \mathbb{R}^{n}$: The feature vector of a graph in teh case of $d = 1$.
- ▷ $\mathbf{x}_v \in \mathbb{R}^{d}$: The feature vector of the node v.
- ▷ $\mathbf{X}^e \in \mathbb{R}^{m \times c}$: The edge feature matrix of a graph.
- ▷ $\mathbf{x}^e_{(v,u)}$: The dege feature vector of the edge (v, u).
- ▷ $\mathbf{X}^{(t)} \in \mathbb{R}^{n \times b}$: The node hidden feature matrix.
- ▷ $\mathbf{H} \in \mathbb{R}^{n \times b}$: The node hidden feature matrix.
- ▷ $\mathbf{h}_v \in \mathbb{R}^{b}$: The hidden feature vector of node v.
- ▷ k: The layer matrix.
- ▷ t: The time step/iteration index.
- ▷ $\sigma(\cdot)$: The sigmoid activation function.
- ▷ $\sigma_h(\cdot)$: The tangent hyperbolic activation function.
- ▷ $\mathbf{W}, \mathbf{\Theta}, w, \theta$: Learnable model parameters.

Definition 1 (Graph)**.** *A* ***graph*** *is represented as*

$$G = (V, E),$$

where V is the set of vertices or nodes, and E is the set of edges. Let $v_i \in V$ to denote a node and $e_{i,j} = (v_i, v_j) \in E$ to denote an edge pointing from v_j to v_i. The neighborhood of a node v is defined as

$$N(v) = \{u \in V : (v, u) \in E\}.$$

The adjacency matrix $\mathbf{A}$ is a $n \times n$ matrix with $A_{ij} = 1$ if $e_{ij} \in E$ and $A_{ij} = 0$ if $e_{ij} \notin E$. A graph may have node attributes $\mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is a node feature matrix with $\mathbf{x}_v \in \mathbb{R}^{d}$ representing the featrue vector of a node v. Meanwile, a graph may have edge attributes $\mathbf{X}^e$, where $\mathbf{X}^e \in \mathbb{R}^{m \times c}$ is an edge feature matrix with $\mathbf{x}^e_{v,u} \in \mathbb{R}^{c}$ representing the feature vector of an dege (v, u).

Definition 2 (Directed Graph)**.** *A directed graph is a graph with all edges directed from one node to another. An undirected graph is considered as a special case of directed graphs where there is a pair of edges with inverse directions if two nodes are connected. A graph is undirected if and only if the adjacency matrix is symmetric.*

Definition 3 (Spatial–Temporal Graph)**.** *A spatial–temporal graph is an attributed graph where the node attributes change dynamically over time. The spatial–temporal graph is defined as*

$$G^{(t)} = (\mathbf{V}, \mathbf{E}, \mathbf{X}^{(t)})$$

with $\mathbf{X}^{(t)} \in \mathbb{R}^{n \times d}$.

1.1 Taxonomy of Graph Neural Networks

1. Recurrent Graph Neural Networks
    - RecGNNs aim to learn node representations with recurrent neural architectures.
    - They assume a node in a graph constantly exchanges information/message with its neighbors until a stable equilibrium is reached.
2. Convolutional Graph Neural Networks
    - The main idea is to generate a node v's representation by aggregating its own features $\mathbf{x}_v$ and neighbors' features $\mathbf{x}_u$, where $u \in N(v)$.
    - ConvGNNs play a central role in building up many other complex GNN models.
3. Graph Autoencoders
    - These are unsupervised learning frameworks that encode nodes/graphs into a latent vector space and reconstruct graph data from the encoded information.
4. Spatial–Temporal Graph Neural Networks

- These aim to learn hidden patterns from spatial–temporal graphs, which becomes increasingly important in a variety of applications, such as
  - ▷ traffic speed forecasting
  - ▷ driver maneuver anticipation
  - ▷ human action recognition
- The key idea of STGNNs is to consider spatial dependence and temporal dependence at the same time.
- Many current approaches integrate graph convolutions to capture spatial dependence with RNNs or CNNs to model temporal dependence.

1.2 Frameworks

- With the graph structure and node content information as inputs, the outputs of GNNs can focus on different graph analytics tasks with one of the following mechanisms.

  1. Node Level
     - ▷ Outputs relate to node regression and node classification tasks.
     - ▷ RecGNNs and ConvGNNs can extract high-level node representations by information propagation/graph convolution.
  2. Edge Level
     - ▷ Outputs relate to the edge classification and link prediction tasks.
     - ▷ With two nodes' hidden representations from GNNs as inputs, a similarity function or a neural network can be utilized to predict the label/connection strength of an edge.
  3. Graph Level
     - ▷ Outputs relate to the graph classification task.
     - ▷ To obtain a compact representation on the graph level, GNNs are often combined with pooling and readout operations.

- **Training Frameworks**: Many GNNs (e.g., ConvGNNs) can be trained in a (semi)supervised or purely unsupervised way within an end-to-end learning framework, depending on the learning tasks and label information available at hand.

1. Semisupervised Learning for Node-Level Classification
    - ▷ Given a single network with partial nodes being labeled and others remaining unlabeled, ConvGNNs can learn a robust model that effectively identifies the class labels for the unlabeled nodes.
2. Supervised Learning for Graph-Level Classification
    - ▷ Graph-level classification aims to predict the class label(s) for an entire graph.
3. Unsupervised Learning for Graph
    - ▷ When no class labels are available in graphs, we can learn the graph embedding in a purely unsupervised way in an end-to-end framework.

2 Recurrent Graph Neural Networks

- RecGNNs are mostly pioneer works of GNNs.
- GNN updates nodes' states by exchanging neighborhood information recurrently until a stable equilibrium is reached.
- A node's hidden state is recurrently updated by

$$\mathbf{h}_v^{(t)} = \sum_{u \in N(v)} f(\mathbf{x}_v, \mathbf{X}^{\mathbf{e}}_{(v,u)}, \mathbf{x}_u, \mathbf{h}_u^{(t-1)})$$

  where $f(\cdot)$ is a parametric function and $\mathbf{h}_v^{(0)}$ is initialized randomly.

  - ▷ The sum operation enables GNN to be applicable to all nodes.
  - ▷ To ensure convergence, the recurrent function $f(\cdot)$ must be a contraction mapping, which shrinks the distance between two points after projecting them into a latent space.

3 Convolutional Graph Neural Networks

- ConvGNNs are closely related to recurrent graph neural networks.
- Instead of iterating node states with contractive constraints, ConvGNNs address the cyclic mutual dependencies architecturally using a fixed number of layers with different weights in each layer.

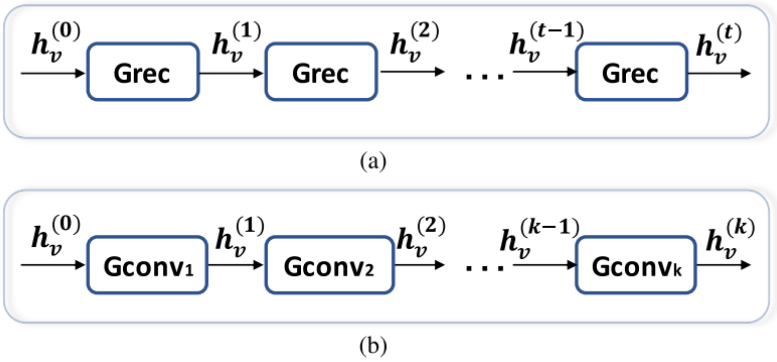

Fig. 3. RecGNNs versus ConvGNNs (a) RecGNNs use the same graph recurrent layer (Grec) in updating node representations. (b) ConvGNNs use a different graph convolutional layer (Gconv) in updating node representations.

- ConvGNNs fall into two categories:
  - ▷ spectral-based
    - Spectral-based approaches define graph convolutions by introducing filters from the perspective of graph signal processing,
    - where the graph convolutional operation is interpreted as removing noises from graph signals.
  - ▷ spatial-based.
    - Spatial-based approaches inherit ideas from RecGNNs to define graph convolutions by information propagation.

3.1 Spectral-Based ConvGNNs

Background:

- Spectral-based methods have a solid mathematical foundation in **graph signal processing.**
- They assume graphs to be undirected.
- The normalized graph Laplacian matrix is a mathematical representation of an undirected graph, defined as

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

  where $\mathbf{D}$ is a diagonal matrix of node degrees, $\mathbf{D}_{ii} = \sum_j(\mathbf{A}_{i,j})$.

  - ▷ The normalized graph Laplacian matrix possesses the property of being real symmetric positive semidefinite.

- With this property, the normalized Laplacian matrix can be factored as

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top,$$

  - ▷ where $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{n-1}] \in \mathbb{R}^{n\times n}$ is the matrix of eigenvectors ordered by eigenvalues
  - ▷ $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues (spectrum), $\mathbf{\Lambda}_{ii} = \lambda_i$.
  - ▷ The eigenvectors of the normalized Laplacian matrix form an orthonormal space, $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$.

- In graph signal processing, a graph signal $\mathbf{x} \in \mathbb{R}^n$ is a feature vector of all nodes of a graph, where x_i is the value of the ith node.
- The graph Fourier transform to a signal $\mathbf{x}$ is defined as $\mathcal{F}(\mathbf{x}) = \mathbf{U}^\top\mathbf{x}$, and the inverse graph Fourier transform is defined as $\mathcal{F}^{-1}(\hat{\mathbf{x}})$,

  - ▷ where $\hat{\mathbf{x}}$ represents the resulted signal from the graph Fourier transform.
  - ▷ The graph Fourier transform projects the input graph signal to the orthonormal space, where the basis is formed by eigenvectors of the normalized graph Laplacian.

- So that the input signal can be represented as $\mathbf{x} = \sum_i x_i \mathbf{u}_i$, which is exactly the inverse graph Fourier transform.

- Now, the graph convolution of the input signal $\mathbf{x}$ with a filter $\mathbf{g} \in \mathbb{R}^n$ is defined as

$$\begin{aligned} \mathbf{x} *_G \mathbf{g} &= \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g})) \\ &= \mathbf{U}(\mathbf{U}^\top \mathbf{x} \odot \mathbf{U}^\top \mathbf{g}) \end{aligned}$$

  where $\odot$ denotes the elementwise product.

- If we denote a filter as $\mathbf{g}_\theta = \mathrm{diag}(\mathbf{U}^\top \mathbf{g})$, then the spectral graph convolution is simplifed as

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta \mathbf{U}^\top \mathbf{x}.$$

- **Spectral-based ConvGNNs all follow this definition.**

- The key difference lies in the choice of the filter $\mathbf{g}_\theta$.

3.1.1 Spectral CNN

- **Spectral CNN** assumes that the filter $\mathbf{g}_\theta = \mathbf{\Theta}_{i,j}^{(k)}$ is a set of learnable parameters and considers graph signals with multiple channels.

- The graph convolutional layer of Spectral CNN is defined as

$$\mathbf{H}_{:,j}^{(k)} = \sigma\left(\sum_{i=1}^{f_{k-1}} \mathbf{U}\mathbf{\Theta}_{i,j}^{(k)} \mathbf{U}^\top \mathbf{H}_{:,i}^{(k-1)}\right), \quad (j = 1, 2, \cdots, f_k)$$

  - ▷ where k is the layer index,
  - ▷ $\mathbf{H}^{(k-1)} \in \mathbb{R}^{n \times f_{k-1}}$ is the input graph signal, $\mathbf{H}^{(0)} = \mathbf{X}$
  - ▷ f_{k-1} is the number of input channels,
  - ▷ f_k is the number of output channels,
  - ▷ $\mathbf{\Theta}_{i,j}^{(k)}$ is a diagonal matrix filled with learnable parameters.

3.1.2 Chebyshev spectral CNN (ChebNet)

- **Chebyshev spectral CNN (ChebNet)** approximates the filter $\mathbf{g}_\theta$ by the Chebyshev polynomials of the diagonal matrix of eigenvalues, i.e., $\mathbf{g}_\theta = \sum_{i=0}^{K} \theta_i T_i(\tilde{\mathbf{\Lambda}})$, where $\tilde{\Lambda} = 2\mathbf{\Lambda}/\lambda_{\max} - \mathbf{I_n}$, and $\tilde{\mathbf{\Lambda}} \in [-1, 1]$
- The Chebyshev polynomials are defined recursively by $T_i(\mathbf{x}) = 2\mathbf{x}T_{i-1}(\mathbf{x}) - T_{i-2}(\mathbf{x})$ with $T_0(\mathbf{x}) = 1$ and $T_1(\mathbf{x}) = \mathbf{x}$.
- As a result, the convolution of a graph signal $\mathbf{x}$ with the defined filter $\mathbf{g}_\theta$ is

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U} \left(\sum_{i=1}^{K} \theta_i T_i(\tilde{\mathbf{\Lambda}}) \right) \mathbf{U}^\top \mathbf{x}$$

  where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I}_n$.
- As $T_i(\tilde{\mathbf{L}}) = \mathbf{U}T_i(\tilde{\mathbf{\Lambda}})\mathbf{U}^\top$, which can be proven by induction on i,
- ChebNet takes the form

$$\mathbf{x} *_G \mathbf{g}_\theta = \sum_{i=1}^{K} \theta_i T_i(\tilde{\mathbf{L}})\mathbf{x} \tag{8}$$

- the filters defined by ChebNet are localized in space -> filters can extract local features independently of the graph size.

3.1.3 Graph convolutional network (GCN)

- **Graph convolutional network (GCN)** introduces a first-order approximation of ChebNet.
- Assuming that $K = 1$ and $\lambda_{\max} = 2$, then (8) is simplified as

$$\mathbf{x} *_G \mathbf{g}_\theta = \theta_0 \mathbf{x} - \theta_1 \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{x}.$$

- To restrain the number of parameters and avoid overfitting, GCN further assume that $\theta = \theta_0 = -\theta_1$, leading to the following **definition of a graph convolution:**

$$\mathbf{x} *_G \mathbf{g}_\theta = \theta(\mathbf{I}_n + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x} \tag{11}$$

- To allow multichannels of inputs and outputs, GCN modifies (11) into a compositional layer, defined as

$$\mathbf{H} = \mathbf{X} *_G \mathbf{g}_\theta = f(\bar{\mathbf{A}}\mathbf{X}\mathbf{\Theta}) \tag{12}$$

  where $\bar{\mathbf{A}} = \mathbf{I}_n + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ and $f(\cdot)$ is an activation function.

- Being a spectral-based method, GCN can be also interpreted as a spatial-based method.

- From a spatial-based perspective, GCN can be considered as aggregating feature information from a node's neighborhood.

- (12) can be expressed as

$$\mathbf{h}_v = f\left(\mathbf{\Theta}^\top \left(\sum_{u \in \{N(v) \cup v\}} \bar{A}_{v,u}\mathbf{x}_u\right)\right), \quad \forall v \in V$$

3.2 Spatial-Based ConvGNNs

- spatial-based methods define graph convolutions based on a node's spatial relations.
- the spatial-based graph convolutions convolve the central node's representation with its neighbors' representations to derive the updated representation for the central node, as shown in

3.2.1 The neural network for graphs (NN4G)

- The neural network for graphs (NN4G), proposed in parallel with GNN*, is the first work toward spatial-based ConvGNNs.
- NN4G performs graph convolutions by summing up a node's neighborhood information directly.
- It also applies residual connections and skip connections to memorize information over each layer.
- As a result, NN4G derives its next-layer node states by

$$\mathbf{h}_v^{(k)} = f\left(\mathbf{W}^{(k)\top}\mathbf{x}_v + \sum_{i=1}^{k-1}\sum_{u\in N(v)} \mathbf{\Theta}^{(k)\top}\mathbf{h}_u^{(k-1)}\right)$$

  where $f(\cdot)$ is an activation function and $\mathbf{h}_v^{(0)} = \mathbf{0}$.

- It can be written in a matrix form

$$\mathbf{H}^{(k)} = f\left(\mathbf{X}\mathbf{W}^{(k)} + \sum_{i=1}^{k-1}\mathbf{A}\mathbf{H}^{(k-1)}\mathbf{\Theta}^{(k)}\right)$$

  which resembles the form of GCN.

- Contextual graph Markov model (CGMM) proposes a probabilistic model inspired by NN4G. While maintaining spatial locality, CGMM has the benefit of probabilistic interpretability.

4 Graph Autoencoders

- GAEs are deep neural architectures that map nodes into a latent feature space and decode graph information from latent representations.

5 Spatial-Temporal Graph Neural Networks

- Graphs in many real-world applications are dynamic both in terms of graph structures and graph inputs.
- STGNNs occupy important positions in **capturing the dynamicity of graphs**.
- Methods under this category aim to model the dynamic node inputs while assuming interdependency between connected nodes.

- STGNNs capture spatial and temporal dependencies of a graph simultaneously.
- STGNNs follow two directions:
  - ▷ RNN-based methods,
  - ▷ CNN-based methods.

- Most RNN-based approaches capture spatial–temporal dependencies by filtering inputs and hidden states passed to a recurrent unit using graph convolutions.
- To illustrate this, suppose that a simple RNN takes the form

$$\mathbf{H}^{(t)} = \sigma\left(\mathbf{W}\mathbf{X}^{(t)} + \mathbf{U}\mathbf{H}^{(t-1)} + \mathbf{b}\right)$$

  where $\mathbf{X}^{(t)} \in \mathbb{R}^{n\times d}$ is the node feature matrix at time step t.
- After inserting graph convolution,

$$\mathbf{H}^{(t)} = \sigma\left(\text{Gconv}(\mathbf{X}^{(t)}, \mathbf{A}; \mathbf{W}) + \text{Gconv}(\mathbf{H}^{(t-1)}, \mathbf{A}; \mathbf{U}) + \mathbf{b}\right)$$

  where $\text{Gconv}(\cdot)$ is a graph convolutional layer.