8

Graph Neural Networks on Complex Graphs

8.1 Introduction

In the earlier chapters, we have discussed graph neural network models focusing on simple graphs where the graphs are static and have only one type of nodes and one type of edges. However, graphs in many real-world applications are much more complicated. They typically have multiple types of nodes, edges, unique structures, and often are dynamic. As a consequence, these complex graphs present more intricate patterns that are beyond the capacity of the aforementioned graph neural network models on simple graphs. Thus, dedicated efforts are desired to design graph neural network models for complex graphs. These efforts can significantly impact the successful adoption and use of GNNs in a broader range of applications. In this chapter, using complex graphs introduced in Section 2.6 as examples, we discuss the methods to extend the graph neural network models to capture more sophisticated patterns. More specifically, we describe more advanced graph filters designed for complex graphs to capture their specific properties.

8.2 Heterogeneous Graph Neural Networks

Heterogeneous graphs, which consist of multiple types of nodes and edges as defined in Definition 2.35, are widely observed in real-world applications. For example, the relations between papers, authors, and venues can be described by a heterogeneous graph, as discussed in Section 2.6.1. Graph neural network models have been adapted to heterogeneous graphs (Zhang et al., 2018b; Wang et al., 2019i; Chen et al., 2019b). Meta-paths (see the definition of meta-path schema and meta-paths in Definition 4.5), which capture various relations between nodes with different semantics, are adopted to deal

with the heterogeneity in the heterogeneous graphs. In (Zhang et al., 2018b; Chen et al., 2019b), meta-paths are utilized to split a heterogeneous graph into several homogeneous graphs. Especially, meta-paths are treated as edges between nodes, and those meta-paths following the same meta-path schema are considered as the same type of edges. Each meta-path schema defines a simple homogeneous graph with meta-path instances following this schema as the edges. Graph filtering operations in Chapter 5 are applied to these simple homogeneous graphs to generate node representations capturing different local semantic information, which are then combined to generate the final node representations. Similarly, meta-paths are used to define meta-path based neighbors, which are treated differently during the graph filtering process in (Wang et al., 2019i). Specifically, given a meta-path schema ψ, a node v_j is defined as a ψ-neighbor for the node v_i, if the node v_j can be reached by node v_i through a meta-path following the schema ψ. The information aggregated from different types of meta-path based neighbors is combined through the attention mechanism to generate the updated node representations (Wang et al., 2019i). Next, we first formally define the meta-path based neighbors and then describe the graph filters designed for heterogeneous graphs.

Definition 8.1 (Meta-path based neighbors) Given a node v_i and a meta-path schema ψ in a heterogeneous graph, the ψ-neighbors of node v_i, denoted as $\mathcal{N}_\psi(v_i)$, consist of nodes that connect with node v_i through a meta-path following schema ψ.

The graph filters for heterogeneous graphs are designed in two steps: 1) aggregating information from ψ-neighbors for each $\psi \in \Psi$, where Ψ denotes the set of meta-path schemas adopted in the task; and 2) combining the information aggregated from each type of neighbors to generate the node representations. Specifically, for a node v_i, the graph filtering operation (for the l-th layer) updates its representation as:

$$\mathbf{z}_{\psi,i}^{(l)} = \sum_{v_j \in \mathcal{N}_\psi(v_i)} \alpha_{\psi,ij}^{(l-1)} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}_\psi^{(l-1)}$$
$$\mathbf{F}_i^{(l)} = \sum_{\psi \in \Psi} \beta_\psi^{(l)} \mathbf{z}_{\psi,i}^{(l)},$$

where $\mathbf{z}_{\psi,i}^{(l)}$ is the information aggregated from ψ-neighbors of node v_i, $\mathbf{\Theta}_\psi^{l-1}$ is parameters specific to meta-path ψ based neighbors, and $\alpha_{\psi,ij}^{(l-1)}$ and $\beta_\psi^{(l)}$ are attention scores which can be learned as similar to the GAT-Filter introduced in Section 5.3.2. Specifically, $\alpha_{\psi,ij}^{(l-1)}$ is used to updates node representations of v_i and it indicates the contribution to v_i from its ψ-neighbor $v_j \in \mathcal{N}_\psi(v_i)$ in the

l-th layer. It is formally defined as:

$$\alpha_{\psi,ij}^{(l-1)} = \frac{\exp\left(\sigma\left(\mathbf{a}_{\psi}^{\mathrm{T}} \cdot \left[\mathbf{F}_i^{(l-1)}\boldsymbol{\Theta}_{\psi}^{(l-1)}, \mathbf{F}_j^{(l-1)}\boldsymbol{\Theta}_{\psi}^{(l-1)}\right]\right)\right)}{\sum_{v_k \in \mathcal{N}_{\psi}(v_i)} \exp\left(\sigma\left(\mathbf{a}_{\psi}^{\mathrm{T}} \cdot \left[\mathbf{F}_i^{(l-1)}\boldsymbol{\Theta}_{\psi}^{(l-1)}, \mathbf{F}_k^{(l-1)}\boldsymbol{\Theta}_{\psi}^{(l-1)}\right]\right)\right)},$$

where $\mathbf{a}_{\psi}$ is a vector of parameters to be learned. Meanwhile, the attention score $\beta_{\psi}^{(l)}$ to combine information from different meta-based neighbors is not specific for each node v_i but shared by all nodes in $\mathcal{V}$ in their representation updates. $\beta_{\psi}^{(l)}$ indicates the contribution from the ψ-neighbors of v_i. It is formally defined as:

$$\beta_{\psi}^{(l)} = \frac{\exp\left(\frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} \mathbf{q}^{\top} \cdot \tanh\left(\mathbf{z}_{\psi,i}^{(l)}\boldsymbol{\Theta}_{\beta}^{(l)} + \mathbf{b}\right)\right)}{\sum_{\psi\in\Psi} \exp\left(\frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} \mathbf{q}^{\top} \cdot \tanh\left(\mathbf{z}_{\psi,i}^{(l)}\boldsymbol{\Theta}_{\beta}^{(l)} + \mathbf{b}\right)\right)},$$

where $\mathbf{q}$, $\boldsymbol{\Theta}_{\beta}^{l}$ and $\mathbf{b}$ are the parameters to be learned.

8.3 Bipartite Graph Neural Networks

Bipartite graphs are widely observed in real-world applications such as recommendations, where users and items are the two disjoint sets of nodes, and their interactions are the edges. In this section, we briefly introduce one general graph filter designed for bipartite graphs since we will present the advanced ones in Section 12.2.2, where we discuss the applications of graph neural networks in recommendations.

As introduced in Definition 2.36, there are two disjoint sets of nodes $\mathcal{U}$ and $\mathcal{V}$, which can be of different types. There are only edges across the two sets while no edges exist within each set. To design the spatial based graph filters, the key idea is to aggregate information from neighboring nodes. In bipartite graphs, for any node $u_i \in \mathcal{U}$, its neighbors is a subset of $\mathcal{V}$, i.e., $\mathcal{N}(u_i) \subset \mathcal{V}$. Similarly, for a node $v_j \in \mathcal{V}$, its neighbors are from $\mathcal{U}$. Hence, two graph filtering operations are needed for these two sets of nodes, which can be described as:

$$\mathbf{F}_{u_i}^{(l)} = \frac{1}{|\mathcal{N}(u_i)|}\sum_{v_j\in\mathcal{N}(u_i)} \mathbf{F}_{v_j}^{(l-1)}\boldsymbol{\Theta}_{v}^{(l-1)},$$

$$\mathbf{F}_{v_i}^{(l)} = \frac{1}{|\mathcal{N}(v_i)|}\sum_{u_j\in\mathcal{N}(v_i)} \mathbf{F}_{u_j}^{(l-1)}\boldsymbol{\Theta}_{u}^{(l-1)},$$

where we use $\mathbf{F}_{u_i}^{(l)}$ to denote the node representation of node u_i after the l-th layer, $\boldsymbol{\Theta}_{v}^{(l-1)}$ and $\boldsymbol{\Theta}_{u}^{(l-1)}$ are parameters to transform embedding from the node space $\mathcal{V}$ to $\mathcal{U}$ and $\mathcal{U}$ to $\mathcal{V}$, respectively.

8.4 Multi-dimensional Graph Neural Networks

In many real-world graphs, multiple types of relations can simultaneously exist between a pair of nodes. These graphs with multiple types of relations can be modeled as multi-dimensional graphs, as introduced in Section 2.6.3. In multi-dimensional graphs, the same set of nodes is shared by all the dimensions, while each dimension has its structure. Hence, when designing graph filters for multi-dimensional graphs, it is necessary to consider both within- and across- dimension interactions. Specifically, the within-dimension interactions are through the connections between the nodes in the same dimension, while the across-dimension interactions are between the "copies" of the same node in different dimensions. In (Ma et al., 2019c), a graph filter, which captures both within- and across- information, is proposed. In detail, during the graph filtering process, for each node v_i, a set of representations of node v_i in all dimensions is first learned and then combined to generate an overall representation for node v_i. To update the representation of node v_i in the dimension d, we need to aggregate information from its neighbors in the same dimension and also the information about v_i in the other dimensions. Hence, we define two types of neighbors in multi-dimensional graphs: the within-dimension neighbors and the across-dimension neighbors. For a given node v_i in the dimension d, the within-dimension neighbors are those nodes that directly connect to the node v_i in the dimension d. In contrast, the across-dimension neighbors consist of the "copies" of the node v_i in other dimensions. The set of within-dimension neighbors of node v_i in dimension d is denoted as $\mathcal{N}_d(v_i)$. For example, in the multi-dimensional graph shown in Figure 8.1, for node 4, its within-dimension neighbors in the "red" dimension include the nodes $1, 2$ and 5. Furthermore, the same node 4 is shared by all dimensions, which can be viewed as "copies" of the same node in different dimensions. These copies of node 4 implicitly connect to each other, and we call them as the across-dimension neighbors for node 4. As shown in Figure 8.1, the across-dimension neighbors for node 4 in the "red" dimension are the copies of node 4 in the "blue" and "green" dimensions. With these two types of neighbors, we can now describe the graph filtering operation (for node v_i in the l-th layer) designed for the multi-dimensional

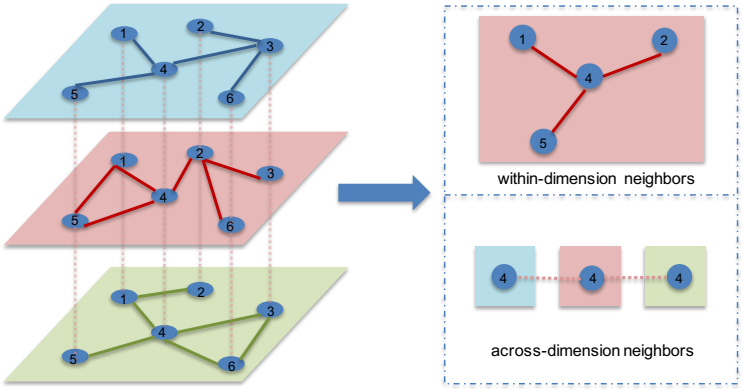

Figure 8.1 An illustrative example of two types of neighbors in the multi-dimensional graph

graph in (Ma et al., 2019c) as:

$$\mathbf{F}_{d,j}^{(l-1)} = \sigma\left(\mathbf{F}_{j}^{(l-1)}\mathbf{\Theta}_{d}^{(l-1)}\right) \quad \text{for } v_j \in \mathcal{N}_d(v_i) \tag{8.1}$$

$$\mathbf{F}_{g,i}^{(l-1)} = \sigma\left(\mathbf{F}_{i}^{(l-1)}\mathbf{\Theta}_{g}^{(l-1)}\right) \quad \text{for } g = 1, \dots D \tag{8.2}$$

$$\mathbf{F}_{w,d,i}^{(l)} = \sum_{v_j \in \mathcal{N}_d(v_i)} \mathbf{F}_{d,j}^{(l-1)} \tag{8.3}$$

$$\mathbf{F}_{a,d,i}^{(l)} = \sum_{g=1}^{D} \beta_{g,d}^{(l-1)} \mathbf{F}_{g,i}^{(l-1)} \tag{8.4}$$

$$\mathbf{F}_{i}^{(l)} = \eta \mathbf{F}_{w,d,i}^{(l)} + (1-\eta)\mathbf{F}_{a,d,i}^{(l)}. \tag{8.5}$$

We next explain the steps of graph filters as described from Eq. (8.1) to Eq. (8.5). In Eq. (8.1), the representations of within-dimension neighbors of node v_i from the previous layer (the $(l-1)$-th layer) are mapped to dimension d by $\mathbf{\Theta}_{d}^{(l-1)}$ and $\sigma()$ is a non-linear activation function. Similarly, the representation of node v_i from the previous layer is projected to different dimensions where D is the total number of dimensions in the multi-dimension graph. The within-dimension aggregation is performed in Eq. (8.3), which generates the within-dimension representation for node v_i in the l-th layer. The across-dimension information aggregation is performed in Eq. (8.4), where $\beta_{g,d}^{(l-1)}$ is the attention score mod-

eling the impact of dimension g on dimension d, which is calculated as:

$$\beta_{g,d}^{(l-1)} = \frac{tr(\mathbf{\Theta}_g^{(l-1)^\top}\mathbf{W}^{(l-1)}\mathbf{\Theta}_d^{(l-1)})}{\sum_{g=1}^{D} tr(\mathbf{\Theta}_g^{(l-1)^\top}\mathbf{W}^{(l-1)}\mathbf{\Theta}_d^{(l-1)})},$$

where $\mathbf{W}^{(l-1)}$ is a parameter matrix to be learned. Finally, the within-dimension representation and the across-dimension representation of node v_i are combined in Eq. (8.5) to generate the updated v_i's representation $\mathbf{F}_i^{(l)}$ after the l-th layer, where η is a hyperparameter balancing these two parts.

8.5 Signed Graph Neural Networks

In many real-world systems, relations can be both positive and negative. For instance, social media users not only have positive edges such as friends (e.g., Facebook and Slashdot), followers (e.g., Twitter), and trust (e.g., Epinions), but also can create negative edges such as foes (e.g., Slashdot), distrust (e.g., Epinions), blocked and unfriended users (e.g., Facebook and Twitter). These relations can be represented as graphs with both positive and negative edges. Signed graphs have become increasingly ubiquitous with the growing popularity of online social networks. A formal definition of signed graphs can be found in Section 2.6.4. The graph filters designed for simple graphs in Chapter 5 cannot be directly applied to signed graphs because of the existence of the negative edges. The negative edges carry very different or even opposite relations compared with the positive edges. Hence, to design graph filters for signed graphs, dedicated efforts are desired to properly handle the negative edges. A naive approach to address the negative edges is to split a signed graph into two separate unsigned graphs, each of which consists of only positive or negative edges. Then the graph filters in Section 5.3 can be separately applied to these two graphs, and the final node representations can be obtained by combining the representations from these two graphs. However, this approach ignores the complex interactions between the positive and negative edges suggested by social balance theories (Heider, 1946; Cartwright and Harary, 1956; Leskovec et al., 2010b), which can provide fruitful results if extracted properly (Kunegis et al., 2009; Leskovec et al., 2010a; Tang et al., 2016b). In (Derr et al., 2018), the balance theory is facilitated to model the relations between the positive and negative edges, based on which a specific graph filter is designed for signed graphs. Specifically, balanced and unbalanced paths are proposed based on the balance theory, which are then adopted to guide the aggregation process when designing the graph filters for signed graphs. Two representations for each node

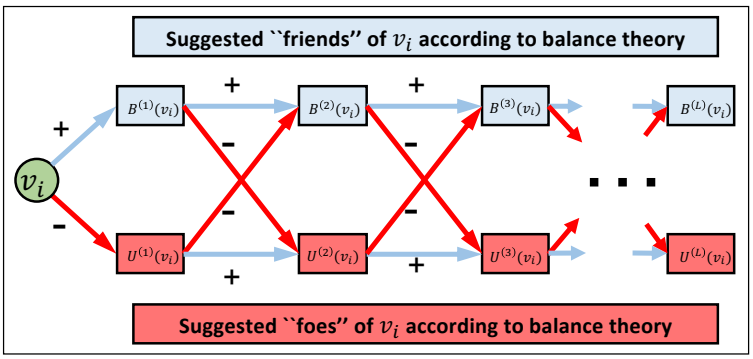

Figure 8.2 Balanced and unbalanced neighbors

are maintained, i.e., one catching the information aggregating from balanced paths and the other capturing information aggregating from unbalanced paths. Next, we first introduce the balanced and unbalanced paths, and then the graph filters designed for signed graphs. In general, balance theory (Heider, 1946; Cartwright and Harary, 1956) suggests that "the friend of my friend is my friend" and "the enemy of my friend is my enemy". Therefore, the cycles in the graphs are classified as *balanced* or *unbalanced*. Specifically, a cycle with an even number of negative edges is considered as *balanced*, otherwise *unbalanced*. It is evident from numerous empirical studies that the majority of circles in real-world signed graphs are balanced (Tang et al., 2016b). Inspired by the definition of balanced cycles, we define a path consisting of an even number of negative edges as a *balanced path*. In contrast, an *unbalanced path* consists of an odd number of negative edges. Given the definition of the balanced path, we can see that a balanced path between node v_i and node v_j indicates a positive relation between them since a balanced cycle is expected according to balance theory and empirical studies. Similarly, an unbalanced path between nodes v_i and v_j indicates a negative relation between them. Given the definition of the balanced and unbalanced paths, we then define the balanced and unbalanced multi-hop neighbors. Nodes that can be reached by a balanced path of length $l-1$ from the node v_i are defined as the $(l-1)$-hop balanced neighbors of the node v_i, denoted as $B^{(l-1)}(v_i)$. Similarly the set of unbalanced $(l-1)$-hop neighbors can be defined and denoted as $U^{l-1}(v_i)$. Given the $(l-1)$-hop balanced and unbalanced neighbors, we can conveniently introduce the $l-$ hop balanced and unbalanced neighbors. As shown in Figure 8.2, adding a positive edge to a balanced path of length $l-1$ or adding a negative edge to an unbalanced path of length $l-1$ lead to balanced paths of length l. Unbalanced paths of length l

can be similarly defined. Formally, we can define the balanced neighbors and unbalanced neighbors of different hops (for $l > 2$) recursively as follows:

$$\begin{aligned} B^l(v_i) &= \{v_j | v_k \in B^{(l-1)}(v_i) \text{ and } v_j \in \mathcal{N}^+(v_k)\} \\ &\cup \{v_j | v_k \in U^{(l-1)}(v_i) \text{ and } v_j \in \mathcal{N}^-(v_k)\}, \\ U^l(v_i) &= \{v_j | v_k \in U^{(l-1)}(v_i) \text{ and } v_j \in \mathcal{N}^+(v_k)\} \\ &\cup \{v_j | v_k \in B^{(l-1)}(v_i) \text{ and } v_j \in \mathcal{N}^-(v_k)\}, \end{aligned}$$

where $\mathcal{N}^+(v_i)$ and $\mathcal{N}^-(v_i)$ denote 1-hop positive and 1-hop negative neighbors of node v_i and we have $B^1(v_i) = \mathcal{N}^+(v_i)$, $U^1(v_i) = \mathcal{N}^-(v_i)$, respectively.

When designing the graph filters for signed graphs, the information from the balanced neighbors and the unbalanced neighbors should be separately maintained, as they could carry very different information. In particular, the balanced neighbors can be regarded as potential "friends", while the unbalanced neighbors can be viewed as potential "foes". Hence, two types of representations are maintained to keep information aggregated from balanced and unbalanced neighbors, respectively. For a node v_i, $\mathbf{F}_i^{(B,l)}$ and $\mathbf{F}_i^{(U,l)}$ are used to denote the representations of node v_i containing information aggregated from balanced and unbalanced neighbors respectively, after l graph filtering layers. Specifically, the process of the graph filters in the l-th layer can be described as follows:

$$\mathbf{F}_i^{(B,l)} = \sigma\left(\left[\sum_{v_j \in \mathcal{N}^+(v_i)} \frac{\mathbf{F}_j^{(B,l-1)}}{|\mathcal{N}^+(v_i)|}, \sum_{v_k \in \mathcal{N}^-(v_i)} \frac{\mathbf{F}_k^{(U,l-1)}}{|\mathcal{N}^-(v_i)|}, \mathbf{F}_i^{(B,l-1)}\right] \mathbf{\Theta}^{(B,l)}\right), \tag{8.6}$$

$$\mathbf{F}_i^{(U,l)} = \sigma\left(\left[\sum_{v_j \in \mathcal{N}^+(v_i)} \frac{\mathbf{F}_j^{(U,l-1)}}{|\mathcal{N}^+(v_i)|}, \sum_{v_k \in \mathcal{N}^-(v_i)} \frac{\mathbf{F}_k^{(B,l-1)}}{|\mathcal{N}^-(v_i)|}, \mathbf{F}_i^{(U,l-1)}\right] \mathbf{\Theta}^{(U,l)}\right), \tag{8.7}$$

where $\mathbf{\Theta}^{(B,l)}$ and $\mathbf{\Theta}^{(U,l)}$ are parameters to learning. In Eq. (8.6), the balanced representations are aggregated from node v_i's positive neighbors, while the unbalanced representations are aggregated from its negative neighbors. As shown in Figure 8.2 and the definitions of balanced and unbalanced neighbors, both sources of information are aggregated through balanced paths for node v_i. In particular, $\mathbf{F}_i^{(B,l)}$ is the concatenation of three types of information – (1) the aggregation of balanced representations (in the $(l-1)$-th layer) from the positive neighbors of node v_i, i.e., $\sum_{v_j \in \mathcal{N}^+(v_i)} \frac{\mathbf{F}_j^{(B,l-1)}}{|\mathcal{N}^+(v_i)|}$; (2) the aggregation of unbalanced representations (in the $(l-1)$-th later) from the negative neighbors of node v_i, i.e., $\sum_{v_k \in \mathcal{N}^-(v_i)} \frac{\mathbf{F}_k^{(U,l-1)}}{|\mathcal{N}^-(v_i)|}$; and (3) the balanced representation of v_i in the $(l-1)$ layer. Similarly, $\mathbf{F}_i^{(U,l)}$ is generated by aggregating information from unbalanced paths in Eq. (8.7). After L graph filtering layers, the balanced and

unbalanced representations for node v_i are combined to form the final representation for v_i as follows:

$$\mathbf{z}_i = [\mathbf{F}_i^{(B,L)}, \mathbf{F}_i^{(U,L)}],$$

where $\mathbf{z}_i$ denotes the generated final representation for node v_i. In (Li et al., 2020b), attention mechanism is adopted to differentiate the importance of nodes when performing the aggregation in Eq. (8.6) and Eq. (8.7). In detail, GAT-Filter is used to perform aggregation from the balanced/unbalanced neighbors in Eq. (8.6) and Eq. (8.7).

8.6 Hypergraph Neural Networks

In many real-world problems, relations go beyond pairwise associations. For example, in a graph describing the relations between papers, a specific author can connect with more than two papers authored by him/her. Here the "author" can be viewed as a "hyperedge" connecting with multiple "papers" (nodes). Compared with edges in simple graphs, hyperedges can encode higher-order relations. The graphs with the hyperedges are named as hypergraphs. A formal definition of hypergraphs can be found in Section 2.6.5. The key to build graph filters for hypergraphs is to facilitate the high-order relations encoded by hyperedges. Specifically, pairwise relations are extracted from these hyperedges, which render the hypergraphs into a simple graph and graph filters designed for simple graphs as introduced in Section 5.3 can thus be applied (Feng et al., 2019b; Yadati et al., 2019). Next, we introduce some representative ways to extract the pairwise relations from the hyperedges. In (Feng et al., 2019b), pairwise relations between node pairs are estimated through the hyperedges. Two nodes are considered to be connected if they appear together in at least one hyperedge. If they appear in several hyperedges, the impact of these hyperedges is combined. An "adjacency matrix" describing the pairwise node relations can be formulated as:

$$\tilde{\mathbf{A}}_{hy} = \mathbf{D}_v^{-1/2}\mathbf{H}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}^\top\mathbf{D}_v^{-1/2}.$$

where the matrices $\mathbf{D}_v, \mathbf{H}, \mathbf{W}, \mathbf{D}_e$ are defined in Definition 2.39. In detail, $\mathbf{H}$ is the indication matrix describing relations between nodes and hyperedges, $\mathbf{W}$ is a diagonal matrix describing the weights on the hyperedges, $\mathbf{D}_v$ and $\mathbf{D}_e$ are the node and hyperedge degree matrices, respectively. Graph filters can then be applied to the simple graph defined by the matrix $\tilde{\mathbf{A}}_{hy}$. In (Feng et al., 2019b),

the GCN-Filter is adopted that can be described as:

$$\mathbf{F}^{(l)} = \sigma(\tilde{\mathbf{A}}_{hy}\mathbf{F}^{(l-1)}\mathbf{\Theta}^{(l-1)}),$$

where σ is a non-linear activation function.

In (Yadati et al., 2019), the method proposed in (Chan et al., 2018) is adopted to convert the hyperedges to pairwise relations. For each hyperedge e, which consists of a set of nodes, two nodes are chosen to be used to generate a simple edge as:

$$(v_i, v_j) := \arg\max_{v_i, v_j \in e} \|\mathbf{h}(v_i) - \mathbf{h}(v_j)\|_2^2,$$

where $\mathbf{h}(v_i)$ can be regarded as some attributes (or some features) that are associated with node v_i. Specifically, in the setting of graph neural networks, for the l-th layer, the hidden representations learned from the previous layer $\mathbf{F}^{(l-1)}$ are the features to measure the relations. A weighted graph can then be constructed by adding all these extracted pairwise relations to the graph, and the weights for these edges are determined by their corresponding hyperedges. We then use $\mathbf{A}^{(l-1)}$ to denote the adjacency matrix describing these relations. The graph filter for the l-th layer can then be expressed as:

$$\mathbf{F}^{(l)} = \sigma(\tilde{\mathbf{A}}^{(l-1)}\mathbf{F}^{(l-1)}\mathbf{\Theta}^{(l-1)}), \tag{8.8}$$

where $\tilde{\mathbf{A}}^{(l-1)}$ is a normalized version of $\mathbf{A}^{l-1}$ with the way introduced in the GCN-Filter in Section 5.3.2. Note that the adjacency matrix $\mathbf{A}^{(l-1)}$ for the graph filter is not fixed but adapted according to the hidden representations from the previous layer.

One major shortcoming for this definition is that only two nodes of each hyperedge are connected. It is likely to cause information loss for other nodes in the hyperedge. Furthermore, this might also lead to a very sparse graph. Hence, one approach to improve the adjacency matrix is proposed in (Chan and Liang, 2019). The chosen nodes are also connected to the remaining nodes in the corresponding hyperedge. Hence, each hyperedge results in $2|e| - 3$ edges, where $|e|$ denotes the number of nodes in the hyperedge e. The weight of each extracted edge is assigned as $1/(2|e| - 3)$. The adjacency matrix $\mathbf{A}^{(l-1)}$ is then built upon these edges, which can be utilized in the graph filtering process in Eq. (8.8).

8.7 Dynamic Graph Neural Networks

Dynamic graphs are constantly evolving; thus the existing graph neural network models are inapplicable as they are not able to capture the temporal in-

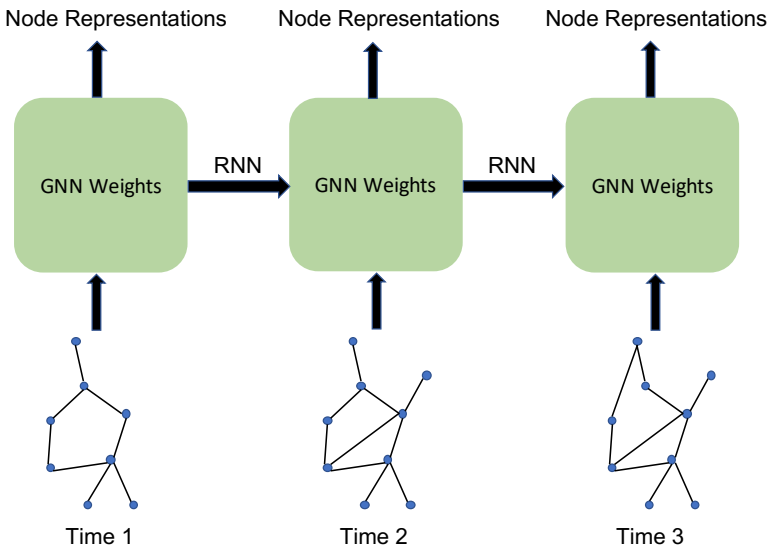

Figure 8.3 An illustration for EvolveGCN

formation. In (Pareja et al., 2019), a graph neural network model, which has evolving weights across graph snapshots over time, named EvolveGCN, is proposed to deal with the discrete dynamic graphs (see the definition of discrete dynamic graphs in Section 2.6.6). For a discrete dynamic graph consisting of T snapshots, T graph neural network models with the same structure (i.e., a stack of several GNN-Filters) are learned. The model parameters for the first graph neural network model are randomly initialized and learned during training, while the model parameters for the t-th GNN model is evolved from the model parameters for the $(t-1)$-th model. As shown in Figure 8.3, the RNN architecture is adopted to update the model parameters. Both the LSTM and GRU variants of RNN as introduced in Section 3.4.2 can be used to update the model parameters. We take GRU as an example to illustrate the l-th graph filtering layer for the t-th graph snapshot as:

$$\mathbf{\Theta}^{(l-1,t)} = \text{GRU}(\mathbf{F}^{(l-1,t)}, \mathbf{\Theta}^{(l-1,t-1)}), \tag{8.9}$$

$$\mathbf{F}^{(l,t)} = \text{GNN-Filter}(\mathbf{A}^{(t)}, \mathbf{F}^{(l-1,t)}, \mathbf{\Theta}^{(l-1,t)}), \tag{8.10}$$

where $\mathbf{\Theta}^{(l-1,t)}$ and $\mathbf{F}^{(l,t)}$ denote the parameters and the output for the l-th graph filtering layer of the t-th GNN model, respectively. The matrix $\mathbf{A}^{(t)}$ is the adjacency matrix of the t-th graph snapshot. Note that the parameters $\mathbf{\Theta}^{(l-1,t)}$ of the l-th layer for the t-th GNN model in Eq. (8.10) are evolved from $\mathbf{\Theta}^{(l-1,t-1)}$ with

GRU as shown in Eq. (8.9). The detailed architecture of GRU can be found in Section 3.4.3. General GNN-Filters can be adopted in Eq. (8.10), while the GCN-Filter is adopted by (Pareja et al., 2019).

8.8 Conclusion

This chapter discusses how the graph neural network models can be extended to complex graphs, including heterogeneous graphs, bipartite graphs, multi-dimensional graphs, signed graphs, hypergraphs, and dynamic graphs. For each type of complex graphs, we introduce representative graph filters that have been specifically designed to capture their properties and patterns.

8.9 Further Reading

While we have introduced representative graph neural networks for these complicated graphs, more works are constantly emerging. In (Zhang et al., 2019a), random walk is utilized to sample heterogeneous neighbors for designing graph neural networks for heterogeneous graphs. In (Sankar et al., 2018), a self-attention mechanism is utilized for discrete dynamic graphs. The attention mechanism is introduced for modeling hypergraph neural networks in (Bai et al., 2019). Graph neural networks have been designed for dynamic graphs in (Jiang et al., 2019; Ma et al., 2020b).