7

Scalable Graph Neural Networks

7.1 Introduction

Graph Neural Networks suffer from severe scalability issue, which prevents them from being adopted to large-scale graphs. Take the GCN-Filter based model for the node classification task as an example, where we adopt gradient-based methods to minimize the following loss function (the same as Eq. (5.49)):

$$\mathcal{L}_{train} = \sum_{v_i \in \mathcal{V}_l} \ell(f_{GCN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta})_i, y_i), \tag{7.1}$$

where $\ell()$ is a loss function and $f_{GCN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta})$ consists of L GCN-Filter layers as described in Eq. (5.21) as:

$$\mathbf{F}^{(l)} = \hat{\mathbf{A}}\mathbf{F}^{(l-1)}\mathbf{\Theta}^{(l-1)}, \quad l = 1, \ldots, L. \tag{7.2}$$

where $\hat{\mathbf{A}}$ is utilized to denote $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ and $\mathbf{F}^{(0)} = \mathbf{F}$. For the convenience of analysis, the node representations in all layers are assumed to have the same dimension d. Note that, in this formulation, we ignore the activation layer that can be added between the graph filtering layers. The parameters $\mathbf{\Theta}$ in Eq.(7.1) include $\mathbf{\Theta}^{(l)}$, $l = 1, \ldots, L$ and the parameters $\mathbf{\Theta}_2$ to perform the prediction as Eq. (5.47). One step of the gradient descent algorithm to minimize the loss can be described as:

$$\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \eta \cdot \nabla_{\mathbf{\Theta}} \mathcal{L}_{train},$$

where η is the learning rate and the gradient $\nabla_{\mathbf{\Theta}} \mathcal{L}_{train}$ needs to be evaluated over the entire training set $\mathcal{V}_l$. Furthermore, due to the design of the GCN-Filter layers as shown in Eq. (7.2), when evaluating $\mathcal{L}_{train}$ in the forward pass, all nodes in $\mathcal{V}$ are involved in the calculation as all node representations are computed in each layer. Hence, in the forward pass of each training epoch, the representations for all nodes and the parameters in each graph filtering layer need to be

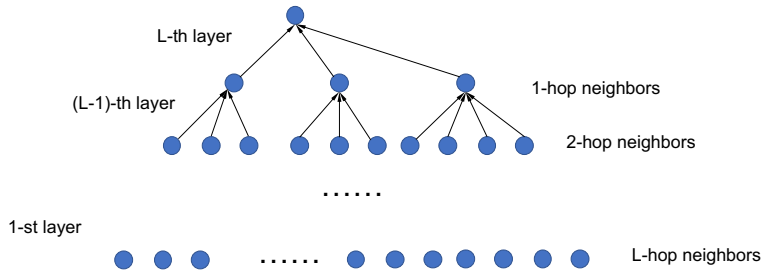

Figure 7.1 The aggregation process

stored in the memory, which becomes prohibitively large when the scale of the graph grows. Specifically, we can calculate the required memory explicitly as follows. During the forward pass, the normalized adjacency matrix $\hat{\mathbf{A}}$, the node representations in all layers $\mathbf{F}^{(l)}$ and the parameters in all layers $\mathbf{\Theta}^{(l)}$ need to be stored in the memory, which requires $O(|\mathcal{E}|)$, $O(L\cdot|\mathcal{V}|\cdot d)$ and $O(L\cdot d^2)$, respectively. Thus, in total, the required memory is $O(|\mathcal{E}|+L\cdot|\mathcal{V}|\cdot d+L\cdot d^2)$. When the size of the graph is large, i.e., $|\mathcal{V}|$ and/or $|\mathcal{E}|$ are large, it becomes impossible to fit them into the memory. Furthermore, the calculation in the form of Eq. (7.2) is not efficient as the final representations (or the representations after the L-th layer) for the unlabeled nodes in $\mathcal{V}_u$ are also calculated, although they are not required for evaluating Eq. (7.1). In detail, $O(L\cdot(|\mathcal{E}|\cdot d+|\mathcal{V}|\cdot d^2)) = O(L\cdot|\mathcal{V}|\cdot d^2)$ operations are required to perform the full epoch of the forward process. As in traditional deep learning scenario, a natural idea to reduce the memory requirement during training is to adopt Stochastic Gradient Descent (SGD). Instead of using all the training samples, it utilizes a single training sample (or a subset of training samples) to estimate the gradient. However, adopting SGD in graph-structured data is not as convenient as that in the traditional scenario since the training samples in Eq. (7.1) are connected to other labeled/unlabeled samples in the graph. To calculate the loss $\ell(f_{GCN}(\mathbf{A},\mathbf{F};\mathbf{\Theta})_i, y_i)$ for node v_i, the node representations of many other nodes (or even the entire graph as indicated by the adjacency matrix $\hat{\mathbf{A}}$) are also involved due to the graph filtering operations as described in Eq. (7.2). To perceive the calculation more clearly, we analyze Eq. (7.2) from a local view for a node v_i as:

$$\mathbf{F}_i^{(l)} = \sum_{v_j\in\tilde{\mathcal{N}}(v_i)} \hat{\mathbf{A}}_{i,j}\mathbf{F}_j^{(l-1)}\mathbf{\Theta}^{(l-1)}, \quad l = 1,\ldots,L, \tag{7.3}$$

which takes the form of aggregating information from neighboring nodes. Note that we use $\mathbf{F}_i^{(l)}$ to denote the node representation for node v_i after l-th graph fil-

tering layer; $\hat{\mathbf{A}}_{i,j}$ to indicate the i,j-th element of $\hat{\mathbf{A}}$; and $\tilde{\mathcal{N}}(v_i) = \mathcal{N}(v_i) \cup \{v_i\}$ to denote the set of neighbors of node v_i including itself. Hence, clearly, as shown in Figure 7.1, from a top-down perspective, i.e., from the L-th layer to the input layer, to calculate the representation of node v_i in the L-th graph filtering layer (the output layer), only the representations of its neighbors (including itself) in the $(L-1)$-th layer are required. To calculate the $(L-1)$-th layer representation for a node $v_j \in \tilde{\mathcal{N}}(v_i)$, all the $(L-2)$-th layer representations of its neighbors are required. The neighbors of all nodes in $\tilde{\mathcal{N}}(v_i)$ are the "neighbors of neighbors" of node v_i, i.e., the 2-hop neighbors of node v_i. In general, computing the loss term for node v_i needs the representation of node v_i after the L-th layer and its l-hop neighbors are required by the $(L-l+1)$-th graph filtering layer. Specifically, its L-hop neighbors are required by the input layer (i.e. the first layer). Hence, for the entire process of calculation, all nodes with the L-hop of node v_i are involved. Based on this analysis, we rewrite the loss for the node v_i as:

$$\ell(f_{GCN}(\mathbf{A}, \mathbf{F}; \boldsymbol{\Theta})_i, y_i) = \ell(f_{GCN}(\mathbf{A}\{\mathcal{N}^L(v_i)\}, \mathbf{F}\{\mathcal{N}^L(v_i)\}; \boldsymbol{\Theta}), y_i), \tag{7.4}$$

where $\mathcal{N}^L(v_i)$ is the set of nodes within L-hop away of node v_i, i.e., all nodes shown in Figure 7.1, $\mathbf{A}\{\mathcal{N}^L(v_i)\}$ denotes the induced structure on $\mathcal{N}^L(v_i)$ (i.e., the rows and columns in the adjacent matrix corresponding to nodes in $\mathcal{N}^L(v_i)$ are retrieved) and $\mathbf{F}\{\mathcal{N}^L(v_i)\}$ indicates the input features for nodes in $\mathcal{N}^L(v_i)$. Typically, the mini-batch SGD algorithm, where a min-batch of training instances are sampled from $\mathcal{V}_l$ to estimate the gradient, is used for parameter updates. The batch-wise loss function can be expressed as:

$$\mathcal{L}_{\mathcal{B}} = \sum_{v_i \in \mathcal{B}} \ell(f_{GCN}(\mathbf{A}\{\mathcal{N}^L(v_i)\}, \mathbf{F}\{\mathcal{N}^L(v_i)\}; \boldsymbol{\Theta}), y_i), \tag{7.5}$$

where $\mathcal{B} \subset \mathcal{V}_l$ is the sampled mini-batch. However, even if SGD is adopted for optimization, the memory requirement can be still high. The major issue is that, as shown in Figure 7.1, the node set $\mathcal{N}^L(v_i)$ expands exponentially as the number of the graph filtering layers L increases. Specifically, the number of nodes in $\mathcal{N}^L(v_i)$ is in the order of deg^L, where deg denotes the average degree of the nodes in the graph. Furthermore, in practice, we need to prepare the memory that is sufficient for the "worst" batch which requires the "most" memory instead of the average one. This could lead to quite a lot of memory when there is a node with a large degree in the batch, as many other nodes are involved due to the inclusion of this large degree node. Thus, to perform SGD optimization, $O(deg^L \cdot d)$ memory is required to store the node representations. This issue of exponentially growing neighborhood is usually referred as "neighborhood expansion" or "neighborhood explosion" (Chen et al., 2018a,b; Huang et al., 2018). When L is larger than the diameter of the graph, we have

$\mathcal{N}^L(v_i) = \mathcal{V}$. It means that the entire node set is required for calculation, which demonstrates an extreme case of neighborhood explosion. Furthermore, the "neighborhood explosion" issue also impacts the time efficiency of the SGD algorithm. Specifically, the time complexity to calculate the final representation $\mathbf{F}_i^{(L)}$ for the node v_i is $O(deg^L \cdot (deg \cdot d + d^2))$, which is $O(deg^L \cdot d^2)$ as deg is usually much smaller than d. Then, the time complexity to run an epoch over the entire training set $\mathcal{V}_l$ is $O(|\mathcal{V}_l| \cdot deg^L \cdot d^2)$ when we assume that each batch only contains a single training sample. When the batch size $|\mathcal{B}| > 1$, the time complexity for an epoch can be lower, as in each batch $\mathcal{B}$, some of the involved nodes may exist in $\mathcal{N}^L(v_i)$ for several samples v_i in the batch $\mathcal{B}$ and their representations can be shared during the calculation. Compared to the full gradient algorithm, which takes $O(L \cdot |\mathcal{V}| \cdot d^2)$ to run a full epoch, the time complexity for SGD can be even higher when L is large although no extra final representations for unlabeled nodes are calculated.

Although we introduce the "neighborhood explosion" issue for the GNN models with GCN-Filters, this issue exists in GNN models with other graph filters as long as they follow a neighborhood aggregation process as Eq. (7.3). In this chapter, without loss of generality, the discussion and analysis are based on the GCN-Filters. To solve the "neighborhood explosion" issue and correspondingly improve the scalability of graph neural network models, various neighborhood sampling methods have been proposed. The main idea of sampling methods is to reduce the number of nodes involved in the calculation of Eq. (7.5) and hence lower the required time and memory to perform the calculation. There are mainly three types of sampling methods:

- **Node-wise sampling methods.** To calculate the representation for a node v_i with Eq. (7.3), in each layer, a set of nodes is sampled from its neighbors. Then, instead of aggregating information from its entire neighborhood, the node representation will only be calculated based on these sampled nodes.
- **Layer-wise sampling methods.** A set of nodes is sampled for the node representation calculation of the entire layer. In other words, to calculate $\mathbf{F}_i^{(l)}$ and $\mathbf{F}_j^{(l)}$ for nodes v_i and v_j, the same set of sampled nodes are utilized to perform the calculation.
- **Subgraph-wise sampling methods.** A subgraph is sampled from the original graph. Then, the node representation learning is based on the sampled subgraph.

In this chapter, we detail and analyze representative algorithms from each group of sampling methods.

7.2 Node-wise Sampling Methods

The node-wise aggregation process in Eq.(7.3) can be rewritten as:

$$\mathbf{F}_i^{(l)} = |\tilde{\mathcal{N}}(v_i)| \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \frac{1}{|\tilde{\mathcal{N}}(v_i)|} \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}, \tag{7.6}$$

which can be regarded as the following expectation form:

$$\mathbf{F}_i^{(l)} = |\tilde{\mathcal{N}}(v_i)| \cdot \mathbb{E}[\mathscr{F}_{v_i}] \tag{7.7}$$

where $\mathscr{F}_{v_i}$ is a discrete random variable as defined below:

$$p\left(\mathscr{F}_{v_i} = \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}\right) = \begin{cases} \frac{1}{\tilde{\mathcal{N}}(v_i)}, & \text{if } v_j \in \tilde{\mathcal{N}}(v_i), \\ 0, & \text{otherwise.} \end{cases}$$

A natural idea to speed up the computation while reducing the memory need for Eq. (7.7) is to approximate the expectation by Monte-Carlo sampling. Specifically, the expectation $\mathbb{E}[\mathscr{F}_{v_i}]$ can be estimated as:

$$\mathbb{E}[\mathscr{F}_{v_i}] \approx \hat{\mathscr{F}}_{v_i} = \frac{1}{|n^l(v_i)|} \sum_{v_j \in n^l(v_i)} \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}, \tag{7.8}$$

where $n^l(v_i) \subset \tilde{\mathcal{N}}(v_i)$ is a set of nodes sampled from $\mathcal{V}$ for the l-th layer calculation for node v_i according to the following probability distribution:

$$p\left(v_j | v_i\right) = \begin{cases} \frac{1}{\tilde{\mathcal{N}}(v_i)}, & \text{if } v_j \in \tilde{\mathcal{N}}(v_i), \\ 0, & \text{otherwise.} \end{cases} \tag{7.9}$$

The estimator in Eq. (7.8) is unbiased as shown below:

$$\begin{aligned}
\mathbb{E}[\hat{\mathscr{F}}_{v_i}] &= \mathbb{E}\left[\frac{1}{|n^l(v_i)|} \sum_{v_j \in n^l(v_i)} \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \mathbb{1}\{v_j \in n^l(v_i)\}\right] \\
&= \mathbb{E}\left[\frac{1}{|n^l(v_i)|} \sum_{v_j \in \mathcal{V}} \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \mathbb{1}\{v_j \in n^l(v_i)\}\right] \\
&= \frac{1}{|n^l(v_i)|} \sum_{v_j \in \mathcal{V}} \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \mathbb{E}\left[\mathbb{1}\{v_j \in n^l(v_i)\}\right] \\
&= \frac{1}{|n^l(v_i)|} \sum_{v_j \in \mathcal{V}} \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \frac{|n^l(v_i)|}{|\tilde{\mathcal{N}}(v_i)|} \\
&= \frac{1}{|\tilde{\mathcal{N}}(v_i)|} \sum_{v_j \in \mathcal{V}} \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \\
&= \mathbb{E}[\mathscr{F}_{v_i}].
\end{aligned}$$

where $\mathbb{1}\{v_j \in n^l(v_i)\}$ is an indicator random variable, which takes value 1 if $v_j \in n^l(v_i)$ and 0 otherwise.

With Eq. (7.8), the node-wise aggregation process can be expressed as:

$$\mathbf{F}_i^{(l)} = \frac{|\tilde{\mathcal{N}}(v_i)|}{|n^l(v_i)|} \sum_{v_j \in n^l(v_i)} \hat{\mathbf{A}}_{i,j} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}. \tag{7.10}$$

The sampling process utilized in Eq. (7.10) is called *node-wise sampling*, as the node set $n^l(v_i)$ is sampled only for the node v_i and not shared with other nodes. Specifically, the GraphSAGE-Filter (see Section 5.3.2 for details on GraphSAGE-Filter) can be viewed as a node-wise sampling method due to the neighbor sampling process. Typically, for a specific graph filtering layer, the sampling size $|n^l(v_i)|$ is set to a fixed value $|n^l(v_i)| = m$ for all nodes. While different graph filtering layers can have different sampling sizes, for convenience, we assume that they all have the same sampling size m in this chapter.

Although the node-wise sampling methods can help control the size of the number of involved nodes in each layer to a fixed size m, it still suffers from the "neighborhood explosion" issue when m is large. In detail, following the same top-down perspective in Figure 7.1, the number of nodes involved to calculate the final representation $\mathbf{F}_i^{(L)}$ for node v_i is in the order of m^L, which increases exponentially as the number of layers L grows. The space and time complexity are $O(m^L \cdot d^2)$ and $O(|\mathcal{V}_l| \cdot m^L \cdot d^2)$, respectively. One way to alleviate this issue is to control the sampling size m to be a small number. However, a small m leads to a large variance in the estimation in Eq. (7.8), which is not desired.

A sampling method, which utilizes an extremely small sampling size m (as small as 2) while maintaining a reasonable variance is proposed in (Chen et al., 2018a). The idea is to keep a historical representation $\bar{\mathbf{F}}_i^{(l-1)}$ for each $\mathbf{F}_i^{(l-1)}$ for $l = 2, \ldots, L$, and then utilize these historical representations during the calculation in Eq.(7.3). Each time when $\mathbf{F}_i^{(l)}$ is calculated, we update its corresponding historical representation $\bar{\mathbf{F}}_i^{(l)}$ with $\mathbf{F}_i^{(l)}$. The historical representations are expected to be similar to the real representations if the model parameters do not change too fast during the training process. We still use Monte-Carlo sampling to estimate Eq. (7.3). However, for those nodes that are not sampled to $n^l(v_i)$, we include their historical representations in the calculation. Formally, Eq. (7.3) can be decomposed into two terms as:

$$\mathbf{F}_i^{(l)} = \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \hat{\mathbf{A}}_{i,j} \Delta\mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \hat{\mathbf{A}}_{i,j} \bar{\mathbf{F}}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \tag{7.11}$$

where

$$\Delta\mathbf{F}_j^{(l-1)} = \mathbf{F}_j^{(l-1)} - \bar{\mathbf{F}}_j^{(l-1)}.$$

The term $\Delta\mathbf{F}_j^{(l-1)}$ denotes the difference between the real up-to-date representation and the historical representation. Instead of using Monte-Carlo sampling to estimate the entire term in Eq. (7.3), only the difference is estimated as:

$$\sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \hat{\mathbf{A}}_{i,j} \Delta\mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \approx \frac{|\tilde{\mathcal{N}}(v_i)|}{|n^l(v_i)|} \sum_{v_j \in n^l(v_i)} \hat{\mathbf{A}}_{i,j} \Delta\mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}. \tag{7.12}$$

With Eq. (7.12), the aggregation process in Eq. (7.11) can be estimated as:

$$\mathbf{F}_i^{(l)} \approx \frac{|\tilde{\mathcal{N}}(v_i)|}{|n^l(v_i)|} \sum_{v_j \in n^l(v_i)} \hat{\mathbf{A}}_{i,j} \Delta\mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \hat{\mathbf{A}}_{i,j} \bar{\mathbf{F}}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}, \tag{7.13}$$

which is named as the control-variate (CV) estimator and utilized to update the node representations. Note that, the second term in the right hand side of Eq. (7.13) is calculated from stored historical node representations, which does not require the recursive calculation process and thus is computationally efficient. The CV-estimator is unbiased as the estimation in Eq. (7.12) is unbiased. The variance of Eq.(7.13) is smaller than that of Eq. (7.10) as $\Delta\mathbf{F}_i^{(l-1)}$ is much smaller than $\mathbf{F}_i^{(l-1)}$. However, the reduced variance does not come for free. While the time complexity of this process remains $O(m^L \cdot d^2)$ (m can be much smaller in the CV estimator) as the aggregation process described in Eq. (7.10), much more memory are required. In fact, to store the historical representations for all nodes involved in the process, $O(deg^L \cdot d)$ memory is required. It is the same as the SGD process without the node-wise sampling. Note that the space complexity is not dependent on the sampling size m; hence, a smaller m cannot ensure the lower space complexity.

7.3 Layer-wise Sampling Methods

In the node-wise sampling methods, to calculate the final representation $\mathbf{F}_i^{(L)}$ for node v_i, the node set $n^L(v_i)$ is sampled from $\tilde{\mathcal{N}}(v_i)$ and $\mathbf{F}_j^{(L-1)}$ for $v_j \in n^L(v_i)$ is utilized during the calculation. Furthermore, to calculate $\mathbf{F}_j^{(L-1)}$ for each $v_j \in n^L(v_i)$, a node set $n^{(L-1)}(v_j)$ needs to be sampled. Specifically, let N^l denote all the nodes sampled for the calculation of the l-th layer, then N^l can be recursively defined from top to down as:

$$N^{l-1} = \cup_{v_j \in N^l} n^l(v_j), \quad \text{with } N^L = n^L(v_i). \quad l = L, \ldots, 2, 1. \tag{7.14}$$

When the mini-batch SGD is adopted and the final representations of a batch $\mathcal{B}$ of nodes need to be calculated, N^L can be defined as $N^L = \cup_{v_i \in \mathcal{B}} n^L(v_i)$. This recursive process in Eq. (7.14) makes N^l grows exponentially; thus, the

node-wise sampling methods still suffer from the "neighborhood explosion" issue. One way to solve the issue is to utilize the same set of sampled nodes to calculate all node representations in a specific layer. In other words, we allow $n^{l-1}(v_j) = n^{l-1}(v_k)$ for $\forall v_j, v_k \in N^l$; thus the size of $N^{(l-1)}$ remains constant as L increases. Then, we only need to sample once for each layer and this strategy is called as *layer-wise sampling*. However, it is impractical to make $n^{l-1}(v_j) = n^{l-1}(v_k)$ as they are sampled according to different node-specific distributions as described in Eq. (7.9). In detail, the set $n^{l-1}(v_j)$ is sampled from the neighborhood of the node v_j; while $n^{l-1}(v_k)$ is sampled from the neighborhood of the node v_k.

Importance sampling is adopted by (Chen et al., 2018b; Huang et al., 2018) to design the layer-wise sampling methods. For the l-th layer, instead of using the node-specific distributions to sample the nodes, a shared distribution, which is defined over the entire node set $\mathcal{V}$, is utilized to sample a shared set of nodes. Then all the output node representations for this layer are calculated only based on these shared sampled nodes. Next, we introduce the details of two representative layer-wise sampling methods (Chen et al., 2018b; Huang et al., 2018). Since these two methods follow the similar design, we focus on the method in (Huang et al., 2018) and then briefly describe the one in (Chen et al., 2018b).

To be consistent with the original paper (Huang et al., 2018), we first reformulate the process from Eq. (7.6) to Eq.(7.9) as follows. The node-wise aggregation process in Eq.(7.3) can be rewritten as:

$$\mathbf{F}_i^{(l)} = D(v_i) \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \frac{\hat{\mathbf{A}}_{i,j}}{D(v_i)} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}, \tag{7.15}$$

where $D(v_i) = \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \hat{\mathbf{A}}_{i,j}$. Eq. (7.15) can be regarded as the following expectation form:

$$\mathbf{F}_i^{(l)} = D(v_i) \cdot \mathbb{E}[\mathscr{F}_{v_i}] \tag{7.16}$$

where $\mathscr{F}_{v_i}$ is a discrete random variable as defined below:

$$p\left(\mathscr{F}_{v_i} = \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}\right) = \begin{cases} \frac{\hat{\mathbf{A}}_{i,j}}{D(v_i)}, & \text{if } v_j \in \tilde{\mathcal{N}}(v_i), \\ 0, & \text{otherwise.} \end{cases}$$

Assume that $q^l(v_j)$ is a known distribution defined on the entire node set $\mathcal{V}$ and $q^l(v_j) > 0, \forall v_j \in \mathcal{V}$. Instead of using Monte-Carlo sampling to estimate

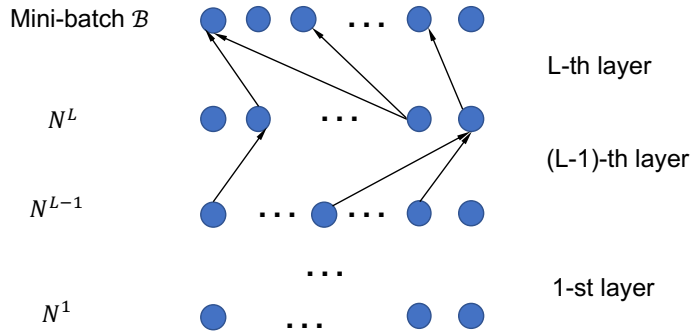

Figure 7.2 Layer-wise sampling

$\mathbb{E}[\mathscr{F}_{v_i}]$, we use importance sampling based on $q^l(v_j)$ as:

$$\mathbb{E}[\mathscr{F}_{v_i}] \approx \hat{\mathscr{F}}_{v_i} = \frac{1}{|N^l|} \sum_{v_j \in N^l} \frac{p(v_j|v_i)}{q^l(v_j)} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}, v_j \sim q^l(v_j) \; \forall v_j \in N^l, \quad (7.17)$$

where N^l denotes a set of nodes sampled according to the distribution $q^l(v_j)$ and $p(v_j|v_i) = \frac{\hat{\mathbf{A}}_{i,j}}{D(v_i)}$ if $v_j \in \tilde{\mathcal{N}}(v_i)$, otherwise $p(v_j|v_i) = 0$. The superscript l in $q^l(v_j)$ indicates that the distribution is utilized in the l-th layer to sample the node set N^l and different layers may use different sampling distributions. The set of nodes N^l is shared by all nodes (e.g., v_i) which need to calculate the representations (e.g., $\mathbf{F}_i^{(l)}$) in the l-th layer. With the importance sampling estimation for $\mathbb{E}[\mathscr{F}_{v_i}]$ in Eq. (7.17), the node-wise aggregation process (as described in Eq. (7.16)) with the layer-wise sampling strategy can then be described as:

$$\begin{aligned} \mathbf{F}_i^{(l)} &= D(v_i) \cdot \frac{1}{|N^l|} \sum_{v_j \in N^l} \frac{p(v_j|v_i)}{q(v_j)} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \\ &= \frac{1}{|N^l|} \sum_{v_j \in N^l} \frac{\hat{\mathbf{A}}_{i,j}}{q^l(v_j)} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)} \end{aligned} \quad (7.18)$$

where nodes in N^l are sampled from $q^l(v_j)$. Note that the distribution $q^l(v_j)$ is not dependent on the center node v_i and it is shared by all nodes. Before describing how to design the sampling distribution $q^l(v_j)$ appropriately, we first introduce the process to sample nodes and build the computational graph to calculate the final representations for all nodes in a sampled batch $\mathcal{B}$. As shown in Figure 7.2, from a top-down perspective, to calculate $\mathbf{F}_i^{(L)}$ for all

nodes $v_i \in \mathcal{B}$, a set of nodes N^L are sampled according to $q^L(v_j)$. The representations $\mathbf{F}_j^{(L-1)}$ of all $v_j \in N^L$ are used to calculate $\mathbf{F}_i^{(L)}$ for all nodes $v_i \in \mathcal{B}$ according to Eq. (7.18). To calculate $\mathbf{F}_j^{(L-1)}$ of $v_j \in N^L$, we need to sample N^{L-1} and aggregate information from them. This process goes to the bottom layer where N^1 is sampled and the input features $\mathbf{F}_j^{(0)}$ for $v_j \in N^1$ are utilized for the calculation. The memory required to compute the final representation $\mathbf{F}_i^{(L)}$ for each node $v_i \in \mathcal{B}$ according to Eq. (7.18), assuming $|N^l| = m$ for all layers, is $O(L \cdot m \cdot d)$. It is much smaller than that required by node-wise sampling based methods. Correspondingly, the time efficiency for each epoch is improved as fewer node representations are required to be computed during this process.

The importance sampling based estimator (IS-estimator) in Eq. (7.17) is unbiased and we want to find a distribution $q(v_j)$ such that the variance of Eq. (7.17) can be minimized. According to the derivations of importance sampling in (Owen, 2013), we conclude that:

Proposition 7.1 *(Huang et al., 2018) The variance of the estimator $\hat{\mathscr{F}}_{v_i}$ in Eq.* (7.17) *is given by:*

$$Var_q(\hat{\mathscr{F}}_{v_i}) = \frac{1}{|N^l|}\left[\frac{(p(v_j|v_i) \cdot |\mathbf{F}_j^{(l-1)}\boldsymbol{\Theta}^{(l-1)}| - \mathbb{E}[\mathscr{F}_{v_i}] \cdot q(u_j))^2}{(q(v_j))^2}\right].$$

The optimal sampling distribution $q(v_j)$, which minimizes the above variance is given by:

$$q(v_j) = \frac{p(v_j|v_i) \cdot |\mathbf{F}_j^{(l-1)}\boldsymbol{\Theta}^{(l-1)}|}{\sum_{v_k \in \mathcal{V}} p(v_k|v_i) \cdot |\mathbf{F}_k^{(l-1)}\boldsymbol{\Theta}^{(l-1)}|}. \tag{7.19}$$

However, the optimal sampling distribution in Eq. (7.19) is not feasible as it is dependent on all node representations in the $(l-1)$-th layer $\mathbf{F}^{(l-1)}$ but we are trying to use the sampling distribution to decide which of them to be calculated. Note that in (Chen et al., 2018b), the variance to be minimized is based on all the nodes in the same layer instead of a single node v_i as Proposition 7.1 and the optimal distribution takes a slightly different form but it is still dependent on $\mathbf{F}^{(l-1)}$.

Hence, two different approaches are proposed in (Chen et al., 2018b) and (Huang et al., 2018), respectively. In (Chen et al., 2018b), the dependence on $\mathbf{F}^{(l-1)}$ is directly discarded and a sampling distribution designed according to the opti-

mal probability distribution is adopted as $q(v_j)$:

$$q(v_j) = \frac{\|\hat{\mathbf{A}}_{:,j}\|^2}{\sum_{v_k \in \mathcal{V}} \|\hat{\mathbf{A}}_{:,k}\|^2}. \tag{7.20}$$

Note that the same $q(v_j)$ as described in Eq. (7.20) is used for all the layers. Hence, the superscript l is removed from $q^l(v_j)$. In (Huang et al., 2018), $\mathbf{F}_j^{(l-1)}\mathbf{\Theta}^{(l-1)}$ is replaced by $\mathbf{F}_j^{(0)}\mathbf{\Theta}_{in}$, where $\mathbf{F}_j^{(0)}$ denotes the input features of node v_j and $\mathbf{\Theta}_{in}$ is a linear projection to be learned. Furthermore, the sampling distribution in Eq. (7.19) is optimal for a specific node v_i, but not ready for layer-wise sampling. To make the distribution applicable to layer-wise sampling, the following distribution, which summarizes computations over all nodes in N^{l+1}, is proposed in (Huang et al., 2018):

$$q^l(v_j) = \frac{\sum_{v_i \in N^{l+1}} p(v_j|v_i) \cdot |\mathbf{F}_j^{(0)}\mathbf{\Theta}_{in}|}{\sum_{v_k \in \mathcal{V}} \sum_{v_i \in N^{l+1}} p(v_k|v_i) \cdot |\mathbf{F}_k^{(0)}\mathbf{\Theta}_{in}|}. \tag{7.21}$$

Note that N^{l+1} denotes the nodes involved in the $(l+1)-th$ layer, which is on the top of the l-th layer. Hence, the distribution $q^l(v_j)$ is dependent on the nodes in its top-layer. Furthermore the distribution is changing in an adaptive way during the training since the parameters $\mathbf{\Theta}_{in}$ are kept updated. With these modifications to the optimal distribution in Eq. (7.19), the distribution in Eq. (7.21) is not guaranteed to lead to minimal variance. Therefore, the variance terms are directly included into the loss function to be explicitly minimized during the training process (Huang et al., 2018).

7.4 Subgraph-wise Sampling Methods

The layer-wise sampling based methods largely reduce the number of nodes involved in calculating final node representations and resolve the neighborhood explosion issue. However, the nature of layer-wise sampling methods is likely to cause another issue in the aggregation process from layer to layer. Specifically, it can be observed from Eq. (7.18) that the aggregation process to generate $\mathbf{F}_i^{(l)}$ is dependent on the term $\hat{\mathbf{A}}_{i,j}$ in each sampled node to be aggregated. This observation indicates that not all nodes in N^l are used to generate $\mathbf{F}_i^{(l)}$, but only those that have connections to node v_i are utilized. Then, if the connections between node v_i and the sampled nodes in N^l are too sparse, the representation $\mathbf{F}_i^{(l)}$ of node v_i may not be learned well. In an extreme case where there are no nodes in N^l connected to the node v_i, the representation of node v_i in

this layer is rendered to 0 according to Eq. (7.18). Hence, to improve the stability of the training process, we need to sample N^l with a reasonable number of connections to node v_i. In other words, we need to ensure that the connectivity between the sampled nodes in N^l and N^{l-1} is dense so that all nodes are likely to have some nodes to aggregate information from. Note that the layer-wise sampling methods described in Section 7.3 do not consider this when designing the layer-wise sampling distribution. To improve the connections between sampled nodes in consecutive layers, the sampling distributions for consecutive layers must be designed in a dependent way, which introduces significant difficulty. One way to ease the design is to use the same set of sampled nodes for all the layers, i.e., $N^l = N^{l-1}$ for $l = L \ldots, 2$. Only one sample distribution needs to be designed so that more connections between the sampled nodes are encouraged. Furthermore, suppose the same set of nodes, denoted as $\mathcal{V}_s$, is adopted for all layers. In that case, the layer-wise aggregation in Eq. (7.18) is running the full neighborhood aggregations on the graph $\mathcal{G}_s = \{\mathcal{V}_s, \mathcal{E}_s\}$ that is induced on the sampled node sets $\mathcal{V}_s$. The induced graph $\mathcal{G}_s$ is a subgraph of the original graph $\mathcal{G}$ as $\mathcal{V}_s \subset \mathcal{V}$ and $\mathcal{E}_s \subset \mathcal{E}$. Instead of sampling $\mathcal{V}_s$, for each batch, we can directly sample a subgraph $\mathcal{G}_s$ from $\mathcal{G}$ and perform model training on the subgraph. The strategy to sample subgraphs for node representation and model training is called *subgraph-wise sampling*. There exist various subgraph-wise sampling based methods (Chiang et al., 2019; Zeng et al., 2019) with different focuses on the sampled graph $\mathcal{G}_s$.

In (Chiang et al., 2019), graph clustering methods such as METIS (Karypis and Kumar, 1998) and Graclus (Dhillon et al., 2007) are adopted to partition the graph $\mathcal{G}$ into a set of subgraphs (clusters) $\{\mathcal{G}_s\}$ such that the number of links within each cluster is much more than that between clusters. To perform SGD, a subgraph is sampled from $\{\mathcal{G}_s\}$ each time and the gradient is estimated based on the following loss function:

$$\mathcal{L}_{\mathcal{G}_s} = \sum_{v_i \in \mathcal{V}_l \cap \mathcal{V}_s} \ell(f_{GNN}(\mathbf{A}_s, \mathbf{F}_s; \mathbf{\Theta})_i, y_i), \tag{7.22}$$

where $\mathbf{A}_s$, $\mathbf{F}_s$ denote the adjacency matrix and features for the sampled subgraph $\mathcal{G}_s$, respectively. The set $\mathcal{V}_l \cap \mathcal{V}_s$ consists of labeled nodes that are in $\mathcal{V}_s$. The memory required to perform one step of SGD based on the sampled subgraph $\mathcal{G}_s$ is $O(|\mathcal{E}_s| + L \cdot |\mathcal{V}_s| \cdot d + L \cdot d^2)$.

In (Zeng et al., 2019), various node samplers are designed to sample a set of nodes $\mathcal{V}_s$ where the subgraph $\mathcal{G}_s$ is induced from. Specifically, an edge-based node sampler is designed to pair-wisely sample nodes that have high influence on each other and random-walk based sampler is designed to improve

the connectivity between the sampled nodes. We briefly describe these two samplers.

- **Edge-based Sampler:** Given a budget m, m edges are randomly sampled according to the following distribution:

$$p((u,v)) = \frac{\left(\frac{1}{d(u)+d(v)}\right)}{\sum_{(u',v')\in\mathcal{E}}\left(\frac{1}{d(u')+d(v')}\right)}, \tag{7.23}$$

  where $d(v)$ denotes the degree of node v. The end nodes of the m sampled edges consist of the sampled nodes $\mathcal{V}_s$, which is used to induce the subgraph $\mathcal{G}_s$.
- **RW-based Sampler:** A set of r root nodes is uniformly sampled (with replacement) from the $\mathcal{V}$. Then starting from each sampled node, a random walk is generated. The nodes in the random walk consist of the final sampled node set $\mathcal{V}_s$, which is used to induce the subgraph $\mathcal{G}_s$.

Some normalization tricks are introduced in the aggregation process to make it less biased as:

$$\mathbf{F}_i^{(l)} = \sum_{v_j\in\mathcal{V}_s} \frac{\hat{\mathbf{A}}_{i,j}}{\alpha_{i,j}} \mathbf{F}_j^{(l-1)} \mathbf{\Theta}^{(l-1)}, \tag{7.24}$$

where $\alpha_{i,j}$ can be estimated from the sampled subgraphs. In detail, a set of M subgraphs is sampled from the samplers and C_i and $C_{i,j}$ count the frequency of the node v_i and edge (v_i, v_j) appearing in the sampled M graphs, respectively. Then, $\alpha_{i,j}$ is estimated by $C_{i,j}/C_i$. Furthermore, the loss function for a mini-batch based on the sampled subgraph $\mathcal{G}_s$ is also normalized as:

$$\mathcal{L}_{\mathcal{G}_s} = \sum_{v_i\in\mathcal{V}_l\cap\mathcal{V}_s} \frac{1}{\lambda_i} \ell(f_{GNN}(\mathbf{A}_s, \mathbf{F}_s; \mathbf{\Theta})_i, y_i), \tag{7.25}$$

where λ_i can be estimated as C_i/M. This normalization also makes the loss function less biased.

7.5 Conclusion

In this chapter, we discuss various sampling-based methods to improve the scalability of graph neural network models. We first introduce the "neighborhood explosion" issue, which makes stochastic gradient descent (SGD) methods impractical in training graph neural network models. Then, we present

three types of sampling methods, including node-wise, layer-wise, and subgraph-wise sampling. They aim to reduce the number of involved nodes during the forward pass of mini-batch SGD and improve scalability. For each group, we discuss their advantages and disadvantages and introduce representative algorithms.

7.6 Further Reading

In this chapter, we mainly discuss sampling-based methods to improve the scalability of the graph neural networks. Some of the introduced sampling techniques have been successfully applied to real-world applications. For example, the node-wise sampling based method GraphSage is adapted and applied to large scale graph based recommendation (Ying et al., 2018a); and the layer-wise sampling based method FastGCN is adopted to anti-money laundering in large scale bitcoin transaction network (Weber et al., 2019). Other efforts have been made to develop distributed frameworks for GNNs (Ma et al., 2018a; Wang et al., 2019e; Zhu et al., 2019c; Ma et al., 2019a). These distributed architectures for graph neural networks can handle large graphs with distributed data storage and parallel computation.