14
Advanced Topics in Graph Neural Networks

14.1 Introduction

In Part TWO, we have discussed the most established methods of deep learning on graphs. On the one hand, with the increasingly deep understandings, numerous limitations have been identified for existing GNNs. Some of these limitations inherit from traditional DNNs. For example, as DNNs, GNNs are often treated as black-boxes and lack human-intelligible explanations; and they might present discrimination behaviors to protected groups that can result in unprecedented ethical, moral, and even legal consequences for human society. Others are unique to GNNs. For instance, increasing the number of layers of GNNs often leads to significant performance drop, and there are limitations on the expressive power of existing GNNs in distinguishing graph structures. On the other hand, recently, more successful experiences from traditional DNNs have been adapted to advance GNNs. For example, strategies have been designed to explore unlabeled data for GNNs, and there attempts to extend GNNs from Euclidean space to hyperbolic space. We package these recent efforts into this chapter about advanced topics in GNNs with two goals. First, we aim to bring our readers near the frontier of current research on GNNs. Second, these topics can serve as promising future research directions. For the aforementioned advanced topics, some are relatively well developed, including deeper graph neural networks, exploring unlabeled data via self-supervised learning for GNNs, and the expressiveness of GNNs. We will detail them in the following sections. In contrast, others are just initialized, and we will provide corresponding references as further reading.

14.2 Deeper Graph Neural Networks

It is observed that increasing the number of graph filtering layers (such as GCN-Filter, GAT-Filter; see Section 5.3.2 for more graph filtering operations) to a large number often results in a significant drop in node classification performance. The performance drop is mainly caused by *oversmoothing*. It describes the phenomenon that the node features become similar and less distinguishable as the number of graph filtering layers increases (Li et al., 2018b). Next, we discuss the "*oversmoothing*" issue based on the GCN-Filter. Intuitively, from a spatial perspective, the GCN-Filter is to update a node's representation by "averaging" its neighbors' representations. This process naturally renders representations for neighboring nodes to be similar. Thus, deeply stacking graph filtering operations tends to make all the nodes (assume that the graph is connected) have similar representations. In (Li et al., 2018b), the *oversmoothing* phenomenon is studied asymptotically as the number of graph filtering layers goes to infinity. Specifically, when the number of filtering layers goes to infinity, the nodes' representations converge to the same regardless of their input features. For the ease of analysis, the non-linear activation layers between the graph filtering layers are ignored in (Li et al., 2018b). Without the non-linear activation layers, repeatedly applying L GCN-Filters to $\mathbf{F}$ can be expressed as:

$$\begin{aligned}&\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\left(\cdots\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{F}\mathbf{\Theta}^{(0)}\right)\mathbf{\Theta}^{(2)}\right)\cdots\right)\mathbf{\Theta}^{(L-1)},\\ &=\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\right)^{L}\mathbf{F}\mathbf{\Theta},\end{aligned} \tag{14.1}$$

where $\mathbf{\Theta}$ denotes the multiplication of $\mathbf{\Theta}^{(0)}, \ldots, \mathbf{\Theta}^{(L-1)}$, $\widetilde{\mathbf{A}} = \mathbf{A}+I$ as introduced in the GCN-Filter in Eq. (5.21) and $\widetilde{\mathbf{D}}$ is the corresponding degree matrix. The filtering process in Eq. (14.1) can be viewed as applying the operation $\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\right)^{L}$ to each column of $\mathbf{F}\mathbf{\Theta}$. The following theorem (Li et al., 2018b) demonstrates the *oversmoothing* phenomenon on single channel graph signals:

Theorem 14.1 *Let $\mathcal{G}$ denote a connected non-bipartite graph with $\mathbf{A}$ as its adjacency matrix. Then, for any input feature $\mathbf{f} \in \mathbb{R}^N$, we have*

$$\lim_{L\to\infty}\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\right)^{L}\mathbf{f} = \theta_1 \cdot \mathbf{u}_1, \tag{14.2}$$

where $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\widetilde{\mathbf{D}}$ denotes its corresponding degree matrix. Here $\widetilde{\mathbf{A}}$ can be regarded as the adjacency matrix of a modified version of graph $\mathcal{G}$ with self-loops. The vector $\mathbf{u}_1$ is the eigenvector of $\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ associated with its largest eigenvalue and $\theta_1 = \mathbf{u}_1^\top\mathbf{f}$. In detail, $\mathbf{u}_1 = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{1}$, which only contains the information of the node degree.

Proof Let $\widetilde{\mathbf{L}}_{nor} = \mathbf{I} - \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ denote the normalized Laplacian matrix corresponding to $\widetilde{\mathbf{A}}$. According to Lemma 1.7 in (Chung and Graham, 1997), $\widetilde{\mathbf{L}}_{nor}$ has a complete set of eigenvalues $0 = \lambda_1 < \lambda_2 \ldots, \lambda_N < 2$ with their corresponding eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_N$. Specifically, in the matrix form, the eigen-decomposition of $\widetilde{\mathbf{L}}_{nor}$ can be represented as $\widetilde{\mathbf{L}}_{nor} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, where $\mathbf{U} = [\mathbf{u_1}, \ldots, \mathbf{u}_N]$ is the matrix that consists of all eigenvectors and $\mathbf{\Lambda} = \text{diag}([\lambda_1, \ldots, \lambda_N])$ is the diagonal eigenvalue matrix. The eigenvalues and eigenvectors of $\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ can be related to those of $\widetilde{\mathbf{L}}$ as:

$$\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}} = \mathbf{I} - \widetilde{\mathbf{L}}_{nor} = \mathbf{U}\mathbf{U}^\top - \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \mathbf{U}\left(I - \mathbf{\Lambda}\right)\mathbf{U}^\top.$$

Hence, $1 = 1 - \lambda_1 > 1 - \lambda_2 \ldots, > 1 - \lambda_N > -1$ are the eigenvalues of $\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ with $\mathbf{u}_1, \ldots, \mathbf{u}_N$ as its corresponding eigenvectors. Then, we have

$$\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\right)^L = \left(\mathbf{U}\left(I - \mathbf{\Lambda}\right)\mathbf{U}^\top\right)^L = \mathbf{U}\left(I - \mathbf{\Lambda}\right)^L\mathbf{U}^\top.$$

As the eigenvalues of $\widetilde{\mathbf{A}}$ are in the range of $[0, 1)$, the limit in Eq. (14.2) can be expressed as:

$$\begin{aligned}\lim_{k\to\infty}\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\right)^L \mathbf{f} &= \lim_{k\to\infty}\mathbf{U}\left(I - \mathbf{\Lambda}\right)^L\mathbf{U}^\top\mathbf{f}\\ &= \mathbf{U}\text{diag}([1, 0, \ldots, 0])\mathbf{U}^\top\mathbf{f}\\ &= \mathbf{u}_1 \cdot (\mathbf{u}_1^\top\mathbf{f})\\ &= \theta_1 \cdot \mathbf{u}_1,\end{aligned}$$

which completes the proof. □

Theorem 14.1 shows that repeatedly applying the GCN-Filters to a graph signal $\mathbf{f}$ results in $\theta_1 \cdot \mathbf{u}_1$, which captures information no more than the node degrees. For the multi-channel case as shown in Eq. (14.1), each column of the matrix $\mathbf{F}\mathbf{\Theta}$ is mapped to $\theta_1 \cdot \mathbf{u}_1$ with different θ_1. Hence, different columns contain the same information with different scales. Furthermore, the degree information contained in $\mathbf{u}_1$ is likely not be useful for most node classification tasks, which also explains why the node classification performance decreases as the number of graph filtering layers increases. Similar observations for the case where the non-linear activation (limited to the ReLU activation function) is included are made in (Oono and Suzuki, 2020). Specifically, it is shown in (Oono and Suzuki, 2020) that the ReLU activation function accelerates the process of *oversmoothing*. The goal of the GCN-Filter is to update node representations with the information of neighboring nodes. Stacking k GCN-Filters allows each node to access information from its k-hop neighborhood. To achieve good performance for node classification, it is necessary to aggregate the information from the local neighborhood for each node. However, as shown in above, stacking too many graph filtering layers leads to the *oversmoothing* issue. Various remedies have been proposed to alleviate the oversmoothing issue (Xu et al., 2018a; Rong et al., 2020; Zhao and Akoglu, 2019).

14.2.1 Jumping Knowledge

It is argued in (Xu et al., 2018a) that different nodes require neighborhoods with different depth and thus different numbers of graph filtering layers are required for different nodes. Hence, a strategy named *Jumping Knowledge* is proposed in (Xu et al., 2018a), which adaptively combines the hidden representations for each node from different layers as the final representations. Specifically, let $\mathbf{F}_i^{(1)}, \ldots, \mathbf{F}_i^{(L)}$ be the hidden representations for node v_i after the $1, \ldots, L$-th layer, respectively. These representations are combined to generate the final representation for node v_i as follows:

$$\mathbf{F}_i^o = \mathrm{JK}\left(\mathbf{F}_i^{(0)}, \mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(L)}\right),$$

where JK() is a function which is adaptive for each node. In particular, it can be implemented as the max-pooling operation or an attention based LSTM.

14.2.2 DropEdge

Dropedge (Rong et al., 2019) is introduced to alleviate the *oversmoothing* issue by randomly dropping some edges in the graph during each training epoch. Specifically, before the training of each epoch, a fraction of edges $\mathcal{E}_p$ is uniformly sampled from $\mathcal{E}$ with a sampling rate p. These sampled edges are removed from the edge set, and the remaining edges are denoted as $\mathcal{E}_r = \mathcal{E}/\mathcal{E}_p$. The graph $\mathcal{G}' = \{\mathcal{V}, \mathcal{E}_r\}$ is then used for training in this epoch.

14.2.3 Pairnorm

As discussed before, we desire some smoothness of the node representations to ensure good classification performance while preventing them from being too similar. An intuitive idea is to ensure that the representations of disconnected nodes are relatively distant. In (Zhao and Akoglu, 2019), a regularization term is introduced to force representations of nodes that are not connected to be different.

14.3 Exploring Unlabeled Data via Self-supervised Learning

To train good deep learning models, it usually requires a huge amount of labeled data. For a specific task, it is usually hard and expensive to collect/annotate massive labeled data. However, unlabeled data is typically rich and easy to obtain. For instance, if we are building a sentiment analysis model, the annotated data might be limited, while unlabeled texts are widely available. Thus, it is appealing to take advantage of unlabeled data. In fact, unlabeled data has been used to advance many areas, such as computer vision and natural language processing. In image recognition, deep convolutional neural networks pre-trained on ImageNet such as Inception (Szegedy et al., 2016) and VGG (Simonyan and Zisserman, 2014) have been widely adopted. Note that images from ImageNet are originally labeled; however, they are considered as unlabeled data for a given specific image recognition task, which could have very different labels from these in the ImageNet dataset. In natural language processing, pre-trained language models such as GPT-2 (Radford et al., 2019) and BERT (Devlin et al., 2018) have been adopted to achieve the state of the art performance for various tasks such as question answering and natural language generation. Therefore, it is promising and has the great potential to explore unlabeled data to enhance deep learning on graphs. This chapter discusses how graph neural networks can use unlabeled data for node classification and graph classification/regression tasks. For node classification, unlabeled data has been incorporated by graph neural networks via the simple information aggregation process. This process could be insufficient to make use of unlabeled data fully. Hence, we discuss strategies to leverage unlabeled data more thoroughly. In graph classification/regression tasks, labeled graphs could be limited, but many unlabeled graphs are available. For example, when performing classification/regression tasks on molecules, labeling molecules is expensive, while unlabeled molecules can be easily collected. Therefore, we present approaches to leverage unlabeled graphs for graph-focused tasks.

14.3.1 Node-focused Tasks

The success of deep learning relies on massive labeled data. Self-supervised learning (SSL) has been developed to alleviate this limitation. It often first designs a domain-specific pretext task and then learns better representations with the pretext task to include unlabeled data. As aforementioned, GNNs simply aggregate features of unlabeled data that cannot thoroughly take advantage of the abundant information. Thus, to fully explore unlabeled data, SSL has been harnessed for providing additional supervision for GNNs. The node-focused self-supervised tasks usually generate additional supervised signals from the graph structure and/or node attributes. Such generated self-supervised information can serve as the supervision of auxiliary tasks to improve the performance of GNNs on the node classification task. There are majorly two ways to utilize these generated self-supervised signals (Jin et al., 2020c): 1) two-stage training, where the self-supervised task is utilized to pre-train the graph neural network model, and then the graph neural network model is fine-tuned for the node classification task; 2) joint training, where the self-supervised task and the main task are optimized together. Specifically, the objective of joint training can be formulated as follows:

$$\mathcal{L} = \mathcal{L}_{label} + \eta \cdot \mathcal{L}_{self},$$

where $\mathcal{L}_{label}$ denotes the loss of the main task, i.e., node classification task and $\mathcal{L}_{self}$ is the loss of the self-supervised task. Next, we briefly introduce some of the self-supervised tasks. In detail, we categorize these self-supervised tasks by the information they leverage to construct the self-supervised signals: 1) constructing self-supervised signals with graph structure information; 2) constructing self-supervised signals with node attribute information; and 3) constructing self-supervised signals with both graph structure and node attribute information.

Graph Structure Information

In this subsection, we introduce self-supervised tasks based on graph structure information.

- **Node Property** (Jin et al., 2020c). In this task, we aim to predict the node property using the learned node representations. These node properties can be node degree, node centrality, and local clustering coefficient.
- **Centrality Ranking** (Hu et al., 2019). In this task, we aim to preserve the centrality ranking of the nodes. Instead of directly predict centrality as that in **Node Property**, the task aims to predict pair-wise ranking given any pair of nodes.
- **Edge Mask** (Jin et al., 2020c; Hu et al., 2020, 2019). In this task, we randomly mask (or remove) some edges from the graph and try to predict their existence using the node representations learned by graph neural networks.
- **Pairwise Distance** (Peng et al., 2020; Jin et al., 2020c). In this task, we aim to utilize the node representations to predict the distance between pairs of nodes in the graph. Specifically, the distance between two nodes is measured by the length of the shortest path between them.

- **Distance2Clusters** (Jin et al., 2020c). Instead of predicting the distance between node pairs, in this task, we aim to predict the distance between a node to the clusters in the graph, which can help learn the global position information of these nodes. Clustering methods based on graph structure information such as METIS graph partitioning algorithm (Karypis and Kumar, 1998) are first utilized to generate a total of K clusters. Then, for each cluster, the node with the largest degree in this cluster is chosen as its center. The task of Distance2Cluster is to predict the distances between a node to the centers of these K clusters. Again, the distance is measured by the length of the shortest path between nodes.

Graph Attribute Information

In this subsection, we introduce self-supervised tasks based on the graph attribute information.

- **Attribute Mask** (Jin et al., 2020c; You et al., 2020; Hu et al., 2020). In this task, we randomly mask (or remove) the associated attribute information of some nodes in the graph and aim to utilize the node representations learned from the graph neural network models to predict these node attributes.
- **PairwiseAttrSim** (Jin et al., 2020c). This task is similar to Pair-wise Distance in the sense that we also aim to predict pair-wise information between nodes. Specifically, we aim to predict the similarity between node attributes, where the similarity can be measured by cosine similarity or Euclidean distance.

Graph Attribute and Structure Information

In this subsection, we introduce self-supervised tasks based on both graph structure and attribute information.

- **Pseudo Label** (Sun et al., 2019c; You et al., 2020). In this task, pseudo labels are generated for unlabeled nodes using the graph neural network model or other models. Then they are utilized as supervised signals to retrain the model together with the labeled nodes. In (You et al., 2020), clusters are generated using the learn node representations from the graph neural network model, and the clusters are used as the pseudo labels. In (Sun et al., 2019c), these clusters are aligned with the real labels and then employed as the pseudo labels.
- **Distance2Labeled** (Jin et al., 2020c). This task is similar to the task of Distance2Cluster. Instead of predicting distance between nodes and precalculated clusters, we aim to predict the distance between unlabeled nodes to the labeled nodes.

- **ContextLabel** (Jin et al., 2020c). The ContextLabel task is to predict the label distribution of the context for the nodes in the graph. The context of a given node is defined as all its k-hop neighbors. The label distribution of the nodes in the context of a given node can then be formulated as a vector. Its dimension is the number of classes where each element indicates the frequency of the corresponding label in the context. Nevertheless, the label information of the unlabeled nodes is unknown. Hence the distribution can not be accurately measured. In (Jin et al., 2020c), methods such as Label propagation (LP) (Zhu et al., 2003) and Iterative Classification Algorithm (ICA) (Neville and Jensen, n.d.) are adopted to predict the pseudo labels, which are then used to estimate the label distribution.
- **CorrectedLabel** (Jin et al., 2020c). This task is to enhance the ContextLabel task by iteratively refining the pseudo labels. Specifically, there are two phases in this task: the training phase and the label correction phase. Given the pseudo labels, the training phase is the same as the task of ContextLabel. The predicted pseudo labels in the training phase are then refined in the label correction phase using the noisy label refining algorithm proposed in (Han et al., 2019). These refined (corrected) pseudo labels are adopted to extract the context label distribution in the training phase.

14.3.2 Graph-focused Tasks

In the graph-focused tasks, we denote the set of labeled graphs as $\mathcal{D}_l = \{(\mathcal{G}_i, y_i)\}$, where y_i is the associated label of the graph $\mathcal{G}_i$. The set of unlabeled graphs is denoted as $\mathcal{D}_u = \{(\mathcal{G}_j)\}$. Typically, the number of the unlabeled graphs is much larger than that of labeled graphs, i.e., $|\mathcal{D}_u| \gg |\mathcal{D}_l|$. Exploring unlabeled data aims to extract knowledge from $\mathcal{D}_u$ to help train models on $\mathcal{D}_l$. To take advantage of unlabeled data, self-supervision signals are extracted. As the node-focused case, there are mainly two ways to leverage knowledge from $\mathcal{D}_u$. One is via two-stage training, where GNNs are pre-trained on the unlabeled data $\mathcal{D}_u$ with the self-supervised objective and then fine-tuned on the labeled data $\mathcal{D}_l$. The other is through joint training, where the self-supervised objective is included as a regularization term to be optimized with the supervised loss. In this section, we introduce graph level self-supervised tasks.

- **Context Prediction** (Hu et al., 2019). In context prediction, the pre-training task is to predict whether a given pair of *K-hop neighborhood* and *context graph* belongs to the same node. Specifically, for every node v in a graph $\mathcal{G}$, its *K-hop neighborhood* consists of all nodes and edges that are at most K-hops away from node v in $\mathcal{G}$, which can be denoted as $\mathcal{N}^K_{\mathcal{G}}(v)$. Meanwhile,

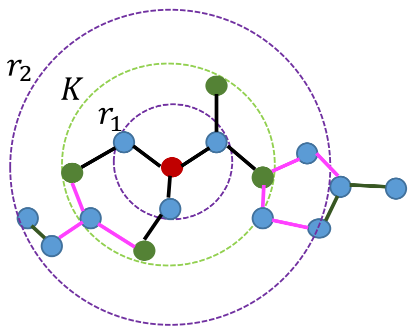

Figure 14.1 Context Prediction

the *context graph* of a node $v \in \mathcal{G}$ is defined by two hyper-parameters r_1, r_2, and it is a subgraph that contains nodes and edges between r_1 and r_2 hops away from the node v in the graph $\mathcal{G}$. In detail, the context graph of node v in a graph $\mathcal{G}$ is a ring of width $r_2 - r_1$ as shown in Figure 14.1, which can be denoted as $C_{v,\mathcal{G}}$. Note that r_1 is required to be smaller than K to ensure that there are some nodes shared between the *neighborhood* and the *context graph* of node v. The task of context prediction is then modeled as a binary classification. It is to determine whether a particular neighborhood $\mathcal{N}^K_{\mathcal{G}}(v)$ and a particular *context graph* $C_{v,\mathcal{G}}$ belong to the same node. A similar task is proposed in (Sun et al., 2019b) where, given a node and a graph, the goal is to predict whether the node belongs to the given graph.

- **Attribute Masking** (Hu et al., 2019). In attribute masking, some node/edge attributes (e.g., atom types in molecular graphs) in a given graph from $\mathcal{D}_u$ is randomly masked. Then graph neural network models are trained to predict these masked node/edge attributes. Note that the attribute masking strategy can only be applied to graphs with node/edge attributes.
- **Graph Property Prediction** (Hu et al., 2019). While there might be no labels for graphs in $\mathcal{D}_u$ for the specific task we want to perform on $\mathcal{D}_l$, there could be other graph attributes available for them. These graph attributes can serve as the supervised signal to pre-train the graph neural network model.

14.4 Expressiveness of Graph Neural Networks

Increasing efforts have been made to analyze the expressiveness of graph neural network models. They aim to analyze the capability of graph neural network models to distinguish graph structures from the graph-level perspective.

Hence, for the ease of discussion, we quickly recap the graph neural network models for graph-focused tasks. We write a general aggregation based spatial graph filter of the l-th layer of graph neural network model as:

$$\mathbf{a}_i^{(l)} = \text{AGG}\left(\left\{\mathbf{F}_j^{(l-1)} | v_j \in \mathcal{N}(v_i)\right\}\right),$$
$$\mathbf{F}_i^{(l)} = \text{COM}\left(\mathbf{F}_i^{(l-1)}, \mathbf{a}_i^{(l)}\right),$$

where $\mathbf{a}_i^{(l)}$ represents the information aggregated from the neighbors of node v_i with the function AGG() and $\mathbf{F}_i^{(l)}$ is the hidden representation of node v_i after the l-th graph filtering layer. The function COM() combines the hidden representation of node v_i from the $(l-1)$-th layer together with the aggregated information to generate the hidden representations in the l-th layer. For graph-focused tasks, a pooling operation is usually operated on the representations $\{\mathbf{F}_i^{(L)} | v_i \in \mathcal{V}\}$ to generate the graph representation, where L is the number of graph filtering layers. Note that, in this chapter, for convenience, we only consider flat pooling operations and the process of pooling is described as:

$$\mathbf{F}_{\mathcal{G}} = \text{POOL}\left(\left\{\mathbf{F}_i^{(L)} | v_i \in \mathcal{V}\right\}\right),$$

where $\mathbf{F}_{\mathcal{G}}$ denotes the graph representation. There are different choices and designs for the AGG(), COM(), and POOL() functions, which results in graph neural network models with different expressiveness. It is shown in (Xu et al., 2019d) that no matter what kinds of functions are adopted, the graph neural network models are at most as powerful as Weisfeiler-Lehman (WL) graph isomorphism test (Weisfeiler and Leman, n.d.) in distinguishing graph structures. The WL test is a powerful test, which can distinguish a broad class of graph structures. Conditions are further established under which the graph neural network models can be as powerful as the WL test in distinguishing graph structures. Next, we first briefly introduce the WL test and how graph neural network models are related. We then present some key results on the expressiveness of graph neural network models.

14.4.1 Weisfeiler-Lehman Test

Two graphs are considered to be topologically identical (or isomorphic) if there is a mapping between the node sets of the graphs such that the adjacency relations are the same. For example, two isomorphic graphs are shown in Figure 14.2, where the color and number indicate the mapping relations between the two sets of nodes. The graph isomorphism task aims to tell whether two given graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are topologically identical. It is computationally expensive to test graph isomorphism, and no polynomial-time algorithm has been

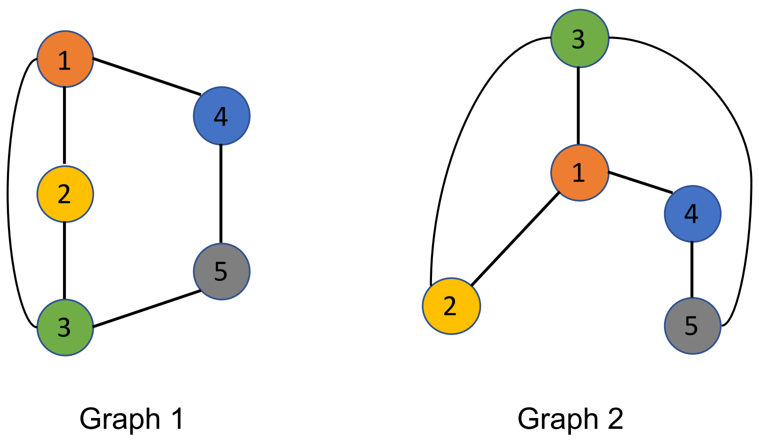

Figure 14.2 Two isomorphic graphs

found yet (Garey and Johnson, n.d.; Babai, 2016). The Weisfeiler-Lehman test is an efficient and effective approach for the graph isomorphism task. It can distinguish a broad class of graphs while failing to distinguish some corner cases (Cai et al., 1992).

For convenience, we assume that each node in the two graphs is associated with labels (attributes). For example, in Figure 14.2, the numbers can be treated as the labels. In practice, the same labels could be associated with different nodes in the graph. A single iteration of the WL test can be described as:

- For each node v_i, we aggregate its neighbors' labels (including itself) into a multi-set $\mathcal{NL}(v_i)$, i.e, a set with repeated elements.
- For each node v_i, we hash the multi-set $\mathcal{NL}(v_i)$ into a unique new label, which is now associated with node v_i as its new label. Note that any nodes with the same multi-set of labels are hashed to the same new label.

The above iteration is repeatedly applied until the sets of labels of two graphs differ from each other. If the sets of labels differ, then the two graphs are non-isomorphic, and the algorithm is terminated. After N (or the number of nodes in the graph) iterations, if the sets of labels of the two graphs are still the same, the two graphs are considered to be isomorphic, or the WL test fails to distinguish them (see (Cai et al., 1992) for the corner cases where the WL test fails). Note that the graph neural network models can be regarded as a generalized WL test. Specifically, the AGG() function in the GNNs corresponds to the ag-

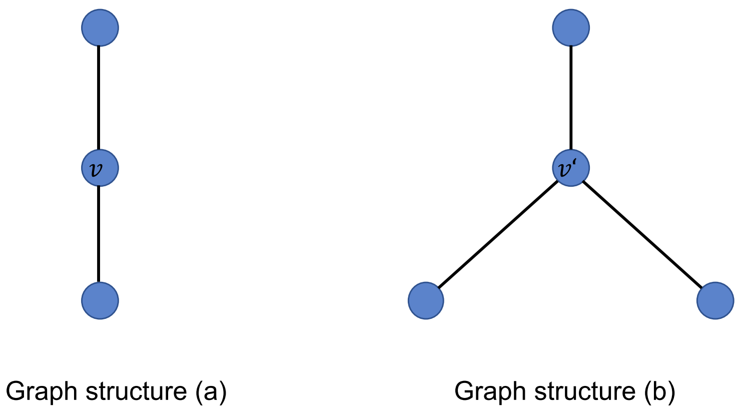

Figure 14.3 Graph structures that mean and max functions fail to distinguish

gregation step in the WL test, and the COM() function corresponds to the hash function in the WL test.

14.4.2 Expressiveness

The expressiveness of the graph neural network models can be related to the graph isomorphism task. Ideally, a graph neural network model with sufficient expressiveness can distinguish graphs with different structures by mapping them into different embeddings. The following lemma shows that the graph neural network models are at most as powerful as the WL test in distinguishing non-isomorphic graphs.

Lemma 14.2 *(Xu et al., 2019d) Given any two non-isomorphic graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, if a graph neural network model maps these two graphs into different embeddings, the WL test also decides that these two graphs are non-isomorphic.*

The power of the WL test is largely attributed to its injective aggregation operation, i.e., the hash function maps nodes with different neighborhoods to different labels. However, many popular aggregation functions in graph neural network models are not injective. Next, we briefly discuss some AGG() functions and provide examples of graph structures where these AGG() functions fail to distinguish. Both the mean function and max function introduced in (Hamilton et al., 2017a) are not injective. As shown in Figure 14.3, assuming that all nodes have the same label (or the same feature), the local structures of nodes v and v' are distinct as they have different numbers of neighbors. However, if mean or max is adopted as the AGG() function, the same representation, i.e. the original label (or feature) of nodes v and v_i, is obtained as the aggregation result for nodes v and v_i. Hence, these two substructures shown in Figure 14.3 cannot be distinguished if mean or max is adopted as the AGG() function. To improve the expressiveness of the GNN models, it is important to design the functions carefully, including AGG(), COM(), and POOL() to be injective. Specifically, the graph neural network models are as powerful as WL test if all these functions are injective as stated in the following theorem:

Theorem 14.3 *(Xu et al., 2019d) A graph neural network model with sufficient graph filtering layers can map two graphs that are tested as non-isomorphic by the WL test to different embeddings, if all AGG(), COM() and POOL() functions in the graph neural network model are injective.*

Theorem 14.3 provides guidance on designing graph neural network models with high expressiveness. While the graph neural network models are at most as powerful as the WL test in distinguishing graph structures, they have their advantages over the WL test. The GNNs models can map graphs into low-dimensional embeddings, which capture the similarity between them. However, the WL test is not able to compare the similarity between graphs except for telling whether they are isomorphic or not. Hence, GNNs are suitable for tasks like graph classification, where graphs can have different sizes, and non-isomorphic graphs with similar structures could belong to the same class.

14.5 Conclusion

In this chapter, we discussed advanced topics in graph neural networks. We describe the oversmoothing issue in graph neural networks and discuss some remedies to mitigate this issue. We introduce various self-supervised learning tasks on graph-structured data for both node- and graph-focused tasks. We demonstrate that the graph neural network models are at most as powerful as the WL test in distinguishing graph structures and provide some guidance in developing graph neural networks with high expressiveness.

14.6 Further Reading

As aforementioned, there are more new directions on graph neural networks that are just initialized. In (Ying et al., 2019; Yuan et al., 2020), explainable graph neural network models have been developed. Specifically, In (Ying et al., 2019), sample-level explanations are generated for graph neural networks, i.e., generating explanations for each sample. While in (Yuan et al., 2020), model-level explanations have been studied for graph neural networks, i.e., understanding how the entire graph neural network model works. In (Tang et al., 2020), the fairness issues of GNNs have been investigated. It is observed that the node classification performance based on GNNs varies in terms of the node degrees. In particular, nodes with low-degrees tend to have a higher error rate than those with high degrees. In (Chami et al., 2019; Liu et al., 2019a), graph neural networks have been extended to hyperbolic space to facilitate both the expressiveness of GNNs and the hyperbolic geometry.