Chapter 18
Graph Neural Networks: Self-supervised Learning

Yu Wang, Wei Jin, and Tyler Derr

Abstract Although deep learning has achieved state-of-the-art performance across numerous domains, these models generally require large annotated datasets to reach their full potential and avoid overfitting. However, obtaining such datasets can have high associated costs or even be impossible to procure. Self-supervised learning (SSL) seeks to create and utilize specific pretext tasks on unlabeled data to aid in alleviating this fundamental limitation of deep learning models. Although initially applied in the image and text domains, recent interest has been in leveraging SSL in the graph domain to improve the performance of graph neural networks (GNNs). For node-level tasks, GNNs can inherently incorporate unlabeled node data through the neighborhood aggregation unlike in the image or text domains; but they can still benefit by applying novel pretext tasks to encode richer information and numerous such methods have recently been developed. For GNNs solving graph-level tasks, applying SSL methods is more aligned with other traditional domains, but still presents unique challenges and has been the focus of a few works. In this chapter, we summarize recent developments in applying SSL to GNNs categorizing them via the different training strategies and types of data used to construct their pretext tasks, and finally discuss open challenges for future directions.

Yu Wang
Department of Electrical Engineering and Computer Science, Vanderbilt University, e-mail: yu.wang.1@vanderbilt.edu

Wei Jin
Department of Computer Science and Engineering, Michigan State University, e-mail: jinwei2@msu.edu

Tyler Derr
Department of Electrical Engineering and Computer Science, Vanderbilt University, e-mail: tyler.derr@vanderbilt.edu

18.1 Introduction

Recent years have witnessed the great success of applying deep learning in numerous fields. However, the superior performance of deep learning heavily depends on the quality of the supervision provided by the labeled data and collecting a large amount of high-quality labeled data tends to be time-intensive and resource-expensive (Hu et al, 2020c; Zitnik and Leskovec, 2017). Therefore, to alleviate the demand for massive labeled data and provide sufficient supervision, self-supervised learning (SSL) has been introduced. Specifically, SSL designs domain-specific pretext tasks that leverage extra supervision from unlabeled data to train deep learning models and learn better representations for downstream tasks. In computer vision, various pretext tasks have been studied, e.g., predicting relative locations of image patches (Noroozi and Favaro, 2016) and identifying augmented images generated from image processing techniques such as cropping, rotating and resizing (Shorten and Khoshgoftaar, 2019). In natural language processing, self-supervised learning has also been heavily utilized, e.g., predicting the masked word in BERT (Devlin et al, 2019).

Simultaneously, graph representation learning has emerged as a powerful strategy for analyzing graph-structured data over the past few years (Hamilton, 2020). As the generalization of deep learning to the graph domain, Graph Neural Networks (GNNs) has become one promising paradigm due to their efficiency and strong performance in real-world applications (You et al, 2021; Zitnik and Leskovec, 2017). However, the vanilla GNN model (i.e., Graph Convolutional Network (Kipf and Welling, 2017b)) and even more advanced existing GNNs (Hamilton et al, 2017b; Xu et al, 2019d, 2018a) are mostly established in a semi-supervised or supervised manner, which still requires high cost label annotation. Additionally, these GNN models may not take full advantage of the abundant information in unlabeled data, such as the graph topology and node attributes. Hence, SSL can be naturally harnessed for GNNs to gain additional supervision and thoroughly exploit the information in the unlabeled data.

Compared with grid-based data such as images or text (Zhang et al, 2020e), graph-structured data is far more complex due to its highly irregular topology, involved intrinsic interactions and abundant domain-specific semantics (Wu et al, 2021d). Different from images and text where the entire structure represents a single entity or expresses a single semantic meaning, each node in the graph is an individual instance with its own features and positioned in its own local context. Furthermore, these individual instances are inherently related with each other, which forms diverse local structures that encode even more complex information to be discovered and analyzed. While such complexity engenders tremendous challenges in analyzing graph-structured data, the substantial and diverse information contained in the node features, node labels, local/global graph structures, and their interactions and combinations provide golden opportunities to design self-supervised pretext tasks.

Embracing the challenges and opportunities to study self-supervised learning in GNNs, the works (Hu et al, 2020c, 2019c; Jin et al, 2020d; You et al, 2020c) have been the first research that systematically design and compare different self-

supervised pretext tasks in GNNs. For example, the works (Hu et al, 2019c; You et al, 2020c) design pretext tasks to encode the topological properties of a node such as centrality, clustering coefficient, and its graph partitioning assignment, or to encode the attributes of a node such as individual features and clustering assignments in embeddings output by GNNs. The work (Jin et al, 2020d) designs pretext tasks to align the pairwise feature similarity or the topological distance between two nodes in the graph with the closeness of two nodes in the embedding space. Apart from the supervision information employed in creating pretext tasks, designing effective training strategies and selecting reasonable loss functions are another crucial components in incorporating SSL into GNNs. Two frequently used training strategies that equip GNNs with SSL are 1) pre-training GNNs through completing pretext task(s) and then fine-tuning the GNNs on downstream task(s), and 2) jointly training GNNs on both pretext and downstream tasks (Jin et al, 2020d; You et al, 2020c). There are also few works (Chen et al, 2020c; Sun et al, 2020c) applying the idea of self-training in incorporating SSL into GNNs. In addition, loss functions are selected to be tailored for purposes of specific pretext tasks, which includes classification-based tasks (cross-entropy loss), regression-based tasks (mean squared error loss) and contrastive-based tasks (contrastive loss).

In view of the substantial progress made in the field of graph neural networks and the significant potential of self-supervised learning, this chapter aims to present a systematic and comprehensive review on applying self-supervised learning into graph neural networks. The rest of the chapter is organized as follows. Section 18.2 first introduces self-supervised learning and pretext tasks, and then summarizes frequently used self-supervised methods from the image and text domains. In Section 18.3, we introduce the training strategies that are used to incorporate SSL into GNNs and categorize the pretext tasks that have been developed for GNNs. Section 18.4 and 18.5 present detailed summaries of numerous representative SSL methods that have been developed for node-level and graph-level pretext tasks. Thereafter, in Section 18.6 we discuss representative SSL methods that are developed using both node-level and graph-level supervision, which we refer to as node-graph-level pretext tasks. Section 18.7 collects and reinforces the major results and the insightful discoveries in prior sections. Concluding remarks and future forecasts on the development of SSL in GNNs are provided in Section 18.8.

18.2 Self-supervised Learning

Supervised learning is the machine learning task of training a model that maps an input to an output based on the ground-truth input-output pairs provided by a labeled dataset. Good performance of supervised learning requires a decent amount of labeled data (especially when using deep learning models), which are expensive to manually collect. Conversely, self-supervised learning generates supervisory signals from unlabeled data and then trains the model based on the generated supervisory signals. The task used for training the model based on the generative signal is

referred to as the pretext task. In comparison, the task whose ultimate performance we care about the most and expect our model to solve is referred to as the downstream task. To guarantee the performance benefits from self-supervised learning, pretext tasks should be carefully designed such that completing them encourages the model to have the similar or complementary understanding as completing downstream tasks. Self-supervised learning initially originated to solve tasks in image and text domains. The following part focuses on introducing self-supervised learning in these two fields with the specific emphasis on different pretext tasks.

In computer vision (CV), many ideas have been proposed for self-supervised representation learning on image data. A common example is that we expect that small distortion on an image does not affect its original semantic meaning or geometric forms. The idea to create surrogate training datasets with unlabeled image patches by first sampling patches from different images at varying positions and then distorting patches by applying a variety of random transformations are proposed in (Dosovitskiy et al, 2014). The pretext task is to discriminate between patches distorted from the same image or from different images. Rotation of an entire image is another effective and inexpensive way to modify an input image without changing semantic content (Gidaris et al, 2018). Each input image is first rotated by a multiple of 90 degrees at random. The model is then trained to predict which rotation has been applied. However, instead of performing pretext tasks on an entire image, the local patches could also be extracted to construct the pretext tasks. Examples of methods using this technique include predicting the relative position between two random patches from one image (Doersch et al, 2015) and designing a jigsaw puzzle game to place nine shuffled patches back to the original locations (Noroozi and Favaro, 2016). More pretext tasks such as colorization, autoencoder, and contrastive predictive coding have also been introduced and effectively utilized (Oord et al, 2018; Vincent et al, 2008; Zhang et al, 2016d).

While computer vision has achieved amazing progress on self-supervised learning in recent years, self-supervised learning has been heavily utilized in natural language processing (NLP) research for quite a while. Word2vec (Mikolov et al, 2013b) is the first work that popularized the SSL ideas in the NLP field. Center word prediction and neighbor word prediction are two pretext tasks in Word2vec where the model is given a small chunk of the text and asked to predict the center word in that text or vice versa. BERT (Devlin et al, 2019) is another famous pre-trained model in NLP where two pretest tasks are to recover randomly masked words in a text or to classify whether two sentences can come one after another or not. Similar works have also been introduced, such as having the pretext task classify whether a pair of sentences are in the correct order (Lan et al, 2020), or a pretext task that first randomly shuffles the ordering of sentences and then seeks to recover the original ordering (Lewis et al, 2020).

Compared with the difficulty of data acquisition encountered in image and text domains, machine learning in the graph domain faces even more challenges in acquiring high-quality labeled data. For example, for molecular graphs it can be extremely expensive to perform the necessary laboratory experiments to label some molecules (Rong et al, 2020a), and in a social network obtaining ground-truth labels

for individual users may require large-scale surveys or be unable to be released due to privacy agreements/concerns (Chen et al, 2020a). Therefore, the success achieved by applying SSL in CV and NLP naturally leads the question as to whether SSL can be effectively applied in the graph domain. Given that graph neural network is among the most powerful paradigms for graph representation learning, in following sections we will mainly focus on introducing self-supervised learning within the framework of graph neural networks and highlighting/summarizing these recent advancements.

18.3 Applying SSL to Graph Neural Networks: Categorizing Training Strategies, Loss Functions and Pretext Tasks

When seeking to apply self-supervised learning to GNNs, the major decisions to be made are how to construct the pretext tasks, which includes what information to leverage from the unlabeled data, what loss function to use, and what training strategy to use for effectively improving the GNN's performance. Hence, in this section we will first mathematically formalize the graph neural network with self-supervised learning and then discuss each of the above. More specifically, we will introduce three training strategies, three loss functions that are frequently employed in the current literature, and categorize current state-of-the-art pretext tasks for GNNs based on the type of information they leverage for constructing the pretext task.

Given an undirected attributed graph $\mathscr{G} = \{\mathscr{V}, \mathscr{E}, X\}$, where $\mathscr{V} = \{v_1, ..., v_{|\mathscr{V}|}\}$ represents the vertex set with $|\mathscr{V}|$ vertices, $\mathscr{E}$ represents the edge set and $e_{ij} = (v_i, v_j)$ is an edge between node v_i and v_j, $X \in \mathbb{R}^{|\mathscr{V}| \times d}$ represents the feature matrix and $\mathbf{x}_i = X[i,:]^\top \in \mathbb{R}^d$ is the d-dimensional feature vector of the node v_i. $A \in \mathbb{R}^{|\mathscr{V}| \times |\mathscr{V}|}$ is the adjacency matrix where $A_{ij} = 1$ if $e_{ij} \in \mathscr{E}$ and $A_{ij} = 0$ if $e_{ij} \notin \mathscr{E}$. We denote any GNN-based feature extractor as $f_\theta : \mathbb{R}^{|\mathscr{V}| \times d} \times \mathbb{R}^{|\mathscr{V}| \times |\mathscr{V}|} \to \mathbb{R}^{|\mathscr{V}| \times d'}$ parametrized by θ, which takes any node feature matrix X and the graph adjacency matrix A and outputs the d'-dimensional representation for each node $Z_{\text{GNN}} = f_\theta(X, A) \in \mathbb{R}^{|\mathscr{V}| \times d'}$, which is further fed into any permutation invariant function READOUT : $\mathbb{R}^{|\mathscr{V}| \times d'} \to \mathbb{R}^{d'}$ to obtain the graph embeddings $\mathbf{z}_{\text{GNN},\mathscr{G}} = \text{READOUT}(f_\theta(X, A)) \in \mathbb{R}^{d'}$. More specifically, we note that here θ represents the parameters encoded in the corresponding network architectures of the GNN (Hamilton et al, 2017b; Kipf and Welling, 2017b; Petar et al, 2018; Xu et al, 2019d, 2018a). Considering the transductive semi-supervised tasks where we are provided with the labeled node set $\mathscr{V}_l \subset \mathscr{V}$, the labeled graph $\mathscr{G}$, the associated node label matrix $Y_{\text{sup}} \in \mathbb{R}^{|\mathscr{V}_l| \times l}$, and the graph label $\mathbf{y}_{\text{sup},\mathscr{G}} \in \mathbb{R}^l$ with label dimension l, we aim to classify nodes and graphs. The node and graph representations output by GNNs are firstly processed by the extra adaptation layer $h_{\theta_{\text{sup}}}$ parametrized by the supervised adaptation parameter θ_{sup} to obtain the predicted l-dimensional node label $Z_{\text{sup}} \in \mathbb{R}^{|\mathscr{V}| \times l}$ and graph label $\mathbf{z}_{\text{sup},\mathscr{G}} \in \mathbb{R}^l$ by Eq. equation 18.1-equation 18.2. Then the model parameters θ in GNN-based extractor f_θ and the parameters θ_{sup} in adaptation layer $h_{\theta_{\text{sup}}}$ are learned

by optimizing the supervised loss calculated between the output/predicted label and the true label for labeled nodes and the labeled graph, which can be formulated as:

$$Z_{\text{sup}} = h_{\theta_{\text{sup}}}(f_\theta(X,A)) \tag{18.1}$$

$$\mathbf{z}_{\text{sup},\mathscr{G}} = h_{\theta_{\text{sup}}}(\text{READOUT}(f_\theta(X,A))) \tag{18.2}$$

$$\theta^*, \theta^*_{\text{sup}} = \arg\min_{\theta,\theta_{\text{sup}}} \mathscr{L}_{\text{sup}}(\theta,\theta_{\text{sup}}) = \begin{cases} \underbrace{\arg\min_{\theta,\theta_{\text{sup}}} \frac{1}{|\mathscr{V}_l|} \sum_{v_i \in \mathscr{V}_l} \ell_{\text{sup}}(\mathbf{z}_{\text{sup},i}, \mathbf{y}_{\text{sup},i})}_{\text{Node supervised task}} \\ \underbrace{\arg\min_{\theta,\theta_{\text{sup}}} \ell_{\text{sup}}(\mathbf{z}_{\text{sup},\mathscr{G}}, \mathbf{y}_{\text{sup},\mathscr{G}})}_{\text{Graph supervised task}} \end{cases}, \tag{18.3}$$

where $\mathscr{L}_{\text{sup}}$ is the total supervised loss function and ℓ_{sup} is the supervised loss function for each example, $\mathbf{y}_{\text{sup},i} = Y_{\text{sup}}[i,:]^\top$ indicates the true label for node v_i in node supervised task and $\mathbf{y}_{\text{sup},\mathscr{G}}$ indicates the true label for graph $\mathscr{G}$ in graph supervised task. Their corresponding predicted label distributions are denoted as $\mathbf{z}_{\text{sup},i} = Z_{\text{sup}}[i,:]^\top$ and $\mathbf{z}_{\text{sup},\mathscr{G}}$. $\theta, \theta_{\text{sup}}$ are parameters to be optimized for any GNN model and the extra adaptation layer for the supervised downstream task, respectively. Note that for ease of notation, we assume the above graph supervised task is operated only on one graph but the above framework can be easily adapted to supervised tasks on multiple graphs.

18.3.1 Training Strategies

In this chapter, we view SSL as the process of designing a specific pretext task and learning the model on the pretext task. In this sense, SSL can either be used as unsupervised pre-training or be integrated with semi-supervised learning.

The model capability of extracting features for completing pretext and downstream tasks is improved through optimizing the model parameters $\theta, \theta_{\text{ssl}}$, and θ_{sup}, where θ_{ssl} denotes the parameters of the adaptation layer for the pretext task. Inspired by relevant discussions (Hu et al, 2019c; Jin et al, 2020d; Sun et al, 2020c; You et al, 2020b,c), we summarize three possible training strategies that are popular in the literature to train GNNs in the self-supervised setting as self-training, pre-training with fine-tuning, and joint training.

18.3.1.1 Self-training

Self-training is a strategy that leverages the supervision information in the training process generated by the model itself (Li et al, 2018b; Riloff, 1996). A typical self-training pipeline begins with first training the model over the labeled data, then generating pseudo labels to unlabeled samples that have highly confident predictions, and including them into the labeled data in the next round of training. In this way, the pretext task is the same as the downstream task by utilizing the pseudo labels for some of the originally unlabeled data. A detailed overview is presented in Fig. 18.1 where the prediction results are re-utilized to augment the training data in the next iteration as done in (Sun et al, 2020c).

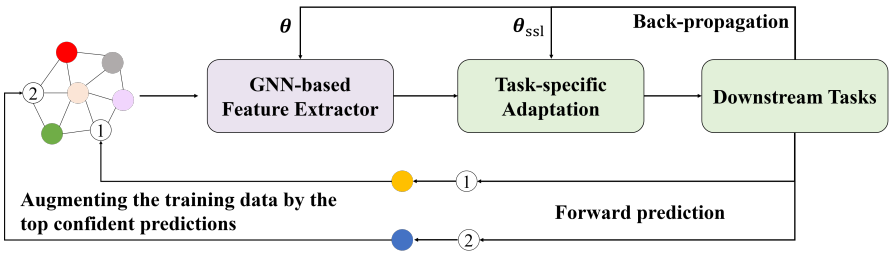

Fig. 18.1: An overview of GNNs with SSL using self-training.

18.3.1.2 Pre-training and Fine-tuning

A common strategy to utilize features learned from completing pretext tasks includes applying the optimized parameters from self-supervision as initialization for fine-tuning in downstream tasks. This strategy consists of two stages: pre-training on the self-supervised pretext tasks and fine-tuning on the downstream tasks. The overview of this two-stage optimization strategy is given in Fig. 18.2.

The whole model consists of one shared GNN-based feature extractor and two adaptation modules, one for the pretext task and one for the downstream task. In the pre-training process, the model is trained with the self-supervised pretext task(s) as:

$$Z_{\text{ssl}} = h_{\theta_{\text{ssl}}}(f_\theta(X,A)), \tag{18.4}$$

$$\mathbf{z}_{\text{ssl},\mathcal{G}} = h_{\theta_{\text{ssl}}}(\text{READOUT}(f_\theta(X,A))), \tag{18.5}$$

$$\theta^*, {\theta^*}_{\text{ssl}} = \arg\min_{\theta,\theta_{\text{ssl}}} \mathscr{L}_{\text{ssl}}(\theta,\theta_{\text{ssl}}) = \begin{cases} \underbrace{\arg\min_{\theta,\theta_{\text{ssl}}} \frac{1}{|\mathscr{V}|} \sum_{v_i \in \mathscr{V}} \ell_{\text{ssl}}(\mathbf{z}_{\text{ssl},i}, \mathbf{y}_{\text{ssl},i})}_{\text{Node pretext tasks}} \\ \underbrace{\arg\min_{\theta,\theta_{\text{ssl}}} \ell_{\text{ssl}}(\mathbf{z}_{\text{ssl},\mathscr{G}}, \mathbf{y}_{\text{ssl},\mathscr{G}})}_{\text{Graph pretext tasks}} \end{cases}, \tag{18.6}$$

where θ_{ssl} denotes the parameters of the adaptation layer $h_{\theta_{\text{ssl}}}$ for the pretext tasks, ℓ_{ssl} is the self-supervised loss function for each example, and $\mathscr{L}_{\text{ssl}}$ is the total loss function of completing the self-supervised task. In node pretext tasks, $\mathbf{z}_{\text{ssl},i} = Z_{\text{ssl}}[i,:]^\top$ and $\mathbf{y}_{\text{ssl},i} = Y_{\text{ssl}}[i,:]^\top$, which are the self-supervised predicted and true label(s) for the node v_i, respectively. In graph pretext tasks, $\mathbf{z}_{\text{ssl},\mathscr{G}}$ and $\mathbf{y}_{\text{ssl},\mathscr{G}}$ are the self-supervised predicted and true label(s) for the graph $\mathscr{G}$, respectively. Then, in the fine-tuning process, the feature extractor f_θ is trained by completing downstream tasks in Eq. equation 18.1-equation 18.3 with the pre-trained θ^* as the initialization. Note that to utilize the pre-trained node/graph representations the fine-tuning process can also be replaced by training a linear classifier (e.g., Logistic Regression (Peng et al, 2020; Veličković et al, 2019; You et al, 2020b; Zhu et al, 2020c)).

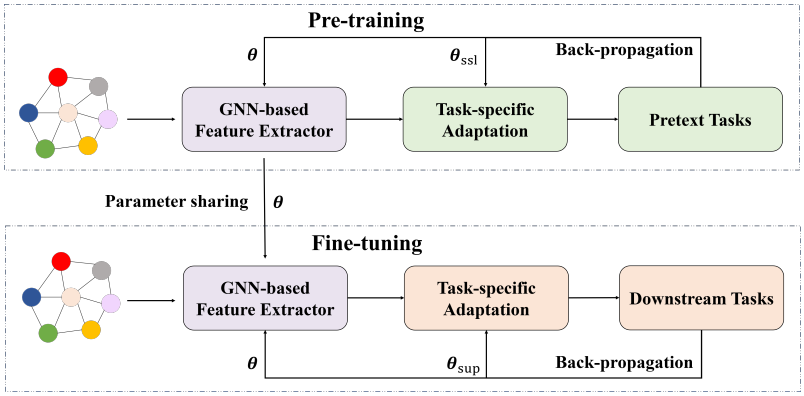

Fig. 18.2: An overview of GNNs with SSL using pre-training and fine-tuning.

18.3.1.3 Joint Training

Another natural idea to harness self-supervised learning for graph neural networks is to combine losses of completing pretext task(s) and downstream task(s) and jointly train the model. The overview of the joint training is shown in Fig. 18.3.

The joint training consists of two components: feature extraction by a GNN and adaption processes for both the pretext tasks and downstream tasks. In the feature extraction process, a GNN takes the graph adjacency matrix A and the feature ma-

trix X as input and outputs the node embeddings Z_{GNN} and/or graph embeddings $\mathbf{z}_{\text{GNN},\mathcal{G}}$. In the adaptation procedure, the extracted node and graph embeddings are further transformed to complete pretext and downstream tasks via $h_{\theta_{\text{ssl}}}$ and $h_{\theta_{\text{sup}}}$, respectively. We then jointly optimize the pretext and downstream task losses as:

$$Z_{\text{sup}} = h_{\theta_{\text{sup}}}(f_\theta(X,A)), \quad Z_{\text{ssl}} = h_{\theta_{\text{ssl}}}(f_\theta(X,A)), \tag{18.7}$$

$$\mathbf{z}_{\text{sup},\mathcal{G}} = h_{\theta_{\text{sup}}}(\text{READOUT}(f_\theta(X,A))), \quad \mathbf{z}_{\text{ssl},\mathcal{G}} = h_{\theta_{\text{ssl}}}(\text{READOUT}(f_\theta(X,A))), \tag{18.8}$$

$$\theta^*, \theta^*_{\text{sup}}, \theta^*_{\text{ssl}} = \begin{cases} \underbrace{\arg\min_{\theta,\theta_{\text{sup}},\theta_{\text{ssl}}} \frac{1}{|\mathcal{V}|} \sum_{v_i \in \mathcal{V}} (\alpha_1 \ell_{\text{sup}}(\mathbf{z}_{\text{sup},i}, \mathbf{y}_{\text{sup},i}) + \alpha_2 \ell_{\text{ssl}}(\mathbf{z}_{\text{ssl},i}, \mathbf{y}_{\text{ssl},i}))}_{\text{Node pretext tasks}} \\ \underbrace{\arg\min_{\theta,\theta_{\text{sup}},\theta_{\text{ssl}}} \alpha_1 \ell_{\text{sup}}(\mathbf{z}_{\text{sup},\mathcal{G}}, \mathbf{y}_{\text{sup},\mathcal{G}}) + \alpha_2 \ell_{\text{ssl}}(\mathbf{z}_{\text{ssl},\mathcal{G}}, \mathbf{y}_{\text{ssl},\mathcal{G}})}_{\text{Graph pretext tasks}} \end{cases}, \tag{18.9}$$

where $\alpha_1, \alpha_2 \in \mathbb{R} > 0$ are the weights for combining the supervised loss ℓ_{sup} and the self-supervised loss ℓ_{ssl}.

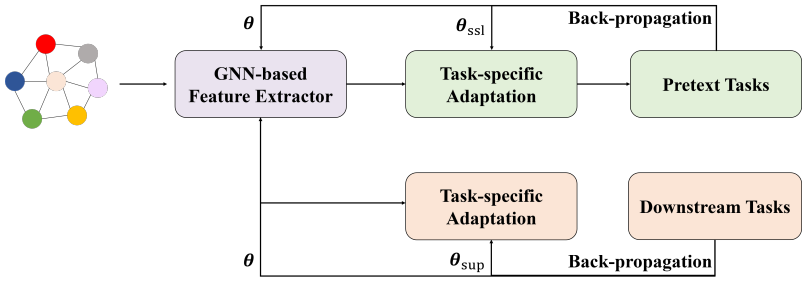

Fig. 18.3: An overview of GNNs with SSL using joint training.

18.3.2 Loss Functions

A loss function is used to evaluate the performance of how well the algorithm models the data. Generally in GNNs with self-supervised learning, the loss function for the pretext task has three forms, which are classification loss, regression loss and contrastive learning loss. Note that the loss functions we discuss here are only for the pretext tasks rather than downstream tasks.

18.3.2.1 Classification and Regression Loss

In completing classification-based pretext tasks such as node clustering where node embeddings are expected to encode the assignment information of the clusters, the objective for the pretext is to minimize the following loss function:

$$\mathscr{L}_{\text{ssl}} = \begin{cases} \underbrace{\frac{1}{|\mathscr{V}|}\sum_{v_i \in \mathscr{V}} \ell_{\text{CE}}(\mathbf{z}_{\text{ssl},i}, \mathbf{y}_{\text{ssl},i}) = -\frac{1}{|\mathscr{V}|}\sum_{v_i \in \mathscr{V}}\sum_{j=1}^{L} 1(\mathbf{y}_{\text{ssl},ij}=1)\log(\tilde{\mathbf{z}}_{\text{ssl},ij})}_{\text{Node pretext tasks}} \\ \underbrace{\ell_{\text{CE}}(\mathbf{z}_{\text{ssl},\mathscr{G}}, \mathbf{y}_{\text{ssl},\mathscr{G}}) = -\sum_{j=1}^{L} 1(\mathbf{y}_{\text{ssl},\mathscr{G}j}=1)\log(\tilde{\mathbf{z}}_{\text{ssl},\mathscr{G}j})}_{\text{Graph pretext tasks}} \end{cases}, \tag{18.10}$$

where ℓ_{CE} indicates the cross entropy function, $\mathbf{z}_{\text{ssl},i}$ and $\mathbf{z}_{\text{ssl},\mathscr{G}}$ represents the predicted label distribution of node v_i and graph $\mathscr{G}$ for the pretext task, and their corresponding class probability distribution $\tilde{\mathbf{z}}_{\text{ssl},i}$ and $\tilde{\mathbf{z}}_{\text{ssl},\mathscr{G}}$ are calculated by softmax normalization, respectively. For example, $\tilde{\mathbf{z}}_{\text{ssl},ij}$ is the probability of node v_i belonging to class j. Since every node v_i has its own pseudo label (i.e., $\mathbf{y}_{\text{ssl},i}$) in completing pretext tasks, we can consider all the nodes $\mathscr{V}$ in the graph compared to only the labeled set of nodes $\mathscr{V}_l$ as before in downstream tasks.

In completing regression-based pretext tasks, such as feature completion, the mean squared error loss is typically used as the loss function:

$$\mathscr{L}_{\text{ssl}} = \begin{cases} \underbrace{\frac{1}{|\mathscr{V}|}\sum_{v_i \in \mathscr{V}} \ell_{\text{MSE}}(\mathbf{z}_{\text{ssl},i}, \mathbf{y}_{\text{ssl},i}) = \frac{1}{|\mathscr{V}|}\sum_{v_i \in \mathscr{V}} ||\mathbf{z}_{\text{ssl},i} - \mathbf{y}_{\text{ssl},i}||^2}_{\text{Node pretext tasks}} \\ \underbrace{\ell_{\text{MSE}}(\mathbf{z}_{\text{ssl},\mathscr{G}}, \mathbf{y}_{\text{ssl},\mathscr{G}}) = ||\mathbf{z}_{\text{ssl},\mathscr{G}} - \mathbf{y}_{\text{ssl},\mathscr{G}}||^2}_{\text{Graph pretext tasks}} \end{cases}, \tag{18.11}$$

where the objective is minimizing the distance from our learned embedding to $\mathbf{y}_{\text{ssl},i}$ which represents any ground-truth value of node v_i, such as the original attribute in the feature completion or other values of node v_i.

18.3.2.2 Contrastive Learning Loss

Inspired by the significant progress achieved by employing the contrastive learning in natural language processing and computer vision (Le-Khac et al, 2020), recent studies (Hassani and Khasahmadi, 2020; Veličković et al, 2019; You et al, 2020b; Zhu et al, 2020c, 2021) propose similar contrastive frameworks to enable SSL in GNNs. The general goal of contrastive learning in GNNs is to train GNN-based encoders such that the agreement of representations between similar graph instances (e.g., multiple views generated from the same instance) is maximized while the agreement between dissimilar graph instances (e.g., multiple views generated from different instances) is minimized. Such maximization and minimization of agree-

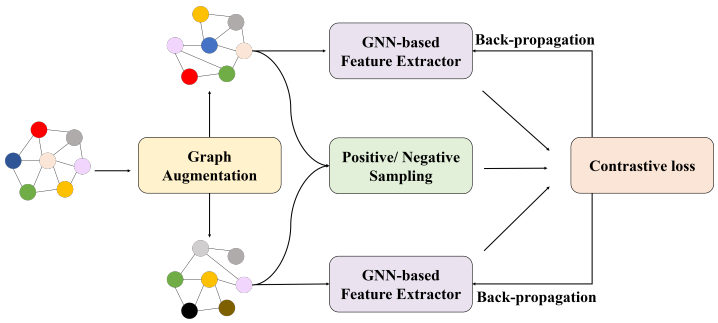

Fig. 18.4: An overview of GNNs with SSL using contrastive learning.

ments between different views of instances is typically formalized as maximizing the mutual information $\mathscr{I}(Z^1_{\text{ssl}}, Z^2_{\text{ssl}})$ between representations Z^1_{ssl} and Z^2_{ssl} under two different views as:

$$\max_{\theta,\theta_{\text{ssl}}} \mathscr{I}(Z^1_{\text{ssl}}, Z^2_{\text{ssl}}), \tag{18.12}$$

where $Z^1_{\text{ssl}}, Z^2_{\text{ssl}}$ correspond to representations output from any GNN-based encoder followed by an adaptation layer $h_{\theta_{\text{ssl}}}$ under two different graph views $\mathscr{G}^1, \mathscr{G}^2$.

In order to computationally estimate and maximize the mutual information that is originally intractable to be exactly computed in most cases (Belghazi et al, 2018; Gabrié et al, 2019; Paninski, 2003; Xie et al, 2021), multiple estimators to evaluate the lower bounds to the mutual information are derived, including normalized temperature-scaled cross-entropy (NT-Xent) (Chen et al, 2020l), Donsker-Varadhan representation of the KL-divergence (Donsker and Varadhan, 1976), noise-contrastive estimation (InfoNCE) gutmann2010noise, Jensen-Shannon estimator (Nowozin et al, 2016). For simplicity, here we only present one frequently used mutual information estimator NT-Xent, which is formalized as:

$$\begin{aligned}\mathscr{L}_{\text{ssl}} &= \frac{1}{|\mathscr{P}^+|} \sum_{(i,j)\in\mathscr{P}^+} \ell_{\text{NT-Xent}}(Z^1_{\text{ssl}}, Z^2_{\text{ssl}}, \mathscr{P}^-) \\ &= -\frac{1}{|\mathscr{P}^+|} \sum_{(i,j)\in\mathscr{P}^+} \log \frac{\exp(\mathscr{D}(\mathbf{z}^1_{\text{ssl},i}, \mathbf{z}^2_{\text{ssl},j}))}{\sum_{k\in\{j\cup\mathscr{P}^-_i\}} \exp(\mathscr{D}(\mathbf{z}^1_{\text{ssl},i}, \mathbf{z}^2_{\text{ssl},k}))}\end{aligned} \tag{18.13}$$

where $\mathscr{D}(\mathbf{z}^1_{\text{ssl},i}, \mathbf{z}^2_{\text{ssl},j})) = \frac{\text{sim}(\mathbf{z}^1_{\text{ssl},i}, \mathbf{z}^2_{\text{ssl},j})}{\tau}$ is a learnable discriminator parametrized with the similarity function (i.e., cosine similarity) and the temperature factor τ, $\mathscr{P}^+$ represents the set of all pairs of positive samples while $\mathscr{P}^- = \bigcup_{(i,j)\in\mathscr{P}^+} \mathscr{P}^-_i$ represents all sets of negative samples. Especially $\mathscr{P}^-_i$ contains all negative samples of the sample i. Note that we can contrast both node representations, graph representations and node-graph representations under different views. Therefore, $\mathbf{z}^1_{\text{ssl}}$ is not limited to the node embeddings, but could refer to the embeddings of both node and

graph under the first graph view $\mathcal{G}^1$. Thus, i, j, k could refer to both node and graph samples.

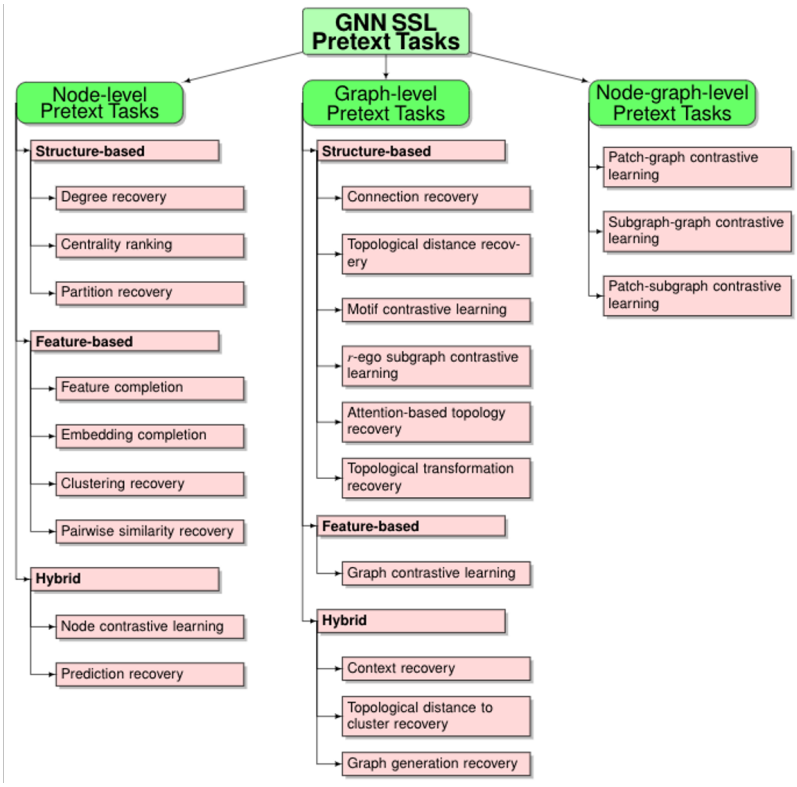

Fig. 18.5: A categorization of pretext tasks in self-supervised learning.[1]

18.3.3 Pretext Tasks

Pretext tasks are constructed by leveraging different types of supervision information coming from different components of graphs. Based on the components that generate the supervision information, pretext tasks that are prevalent in the literature are categorized into node-level, graph-level and node-graph level. In completing node-level and graph-level pretext tasks, three types of information can be leveraged: graph structure, node features, or hybrid, where the latter combines the infor-

[1] Additional summary details and the corresponding code links for these methods can be found at https://github.com/NDS-VU/GNN-SSL-Chapter.

mation from node features, graph structure, and even information from the known training labels (as presented in (Jin et al, 2020d)). We summarize the categorization of pretext tasks as a tree where each leaf node represents a specific type of pretext tasks in Fig. 18.5 while also including the corresponding references. In next three sections, we give detailed explanations about each of these pretext tasks and summarize the majority of existing methods.

18.4 Node-level SSL Pretext Tasks

For node-level pretext tasks, methods have been developed to use easily-accessible data to generate pseudo labels for each node or relationships for each pair of nodes. In this way, the GNNs are then trained to be predictive of the pseudo labels or to keep the equivalence between the node embeddings and the original node relationships.

18.4.1 Structure-based Pretext Tasks

Different nodes have different structure properties in graph topology, which can be measured by the node degree, centrality, node partition, etc. Thus, for structure-based pretext tasks at the node-level, we expect to align node embeddings extracted from the GNNs with their structure properties, in an attempt to ensure this information is preserved while GNNs learn the node embeddings.

Since degree is the most fundamental topological property, Jin et al (2020d) designs the pretext task to recover the node degree from the node embeddings as follows:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{V}|} \sum_{v_i \in \mathscr{V}} \ell_{\text{MSE}}(\mathbf{z}_{\text{ssl},i},\, d_i) \tag{18.14}$$

where d_i represents the degree of node i and $\mathbf{z}_{\text{ssl},i} = Z_{\text{ssl}}[i,:]^\top$ denotes the self-supervised GNN embeddings of node i. It should be noted that this pretext task can be generalized to harness any structure property in the node level.

Node centrality measures the importance of nodes based on their structure roles in the whole graph (Newman, 2018). Hu et al (2019c) designs a pretext task to have GNNs estimate the rank scores of node centrality. The specific centrality measures considered are eigencentrality, betweenness, closeness, and subgraph centrality. For a node pair (u,v) and a centrality score s, with relative order $R^s_{u,v} = 1(s_u > s_v)$ where $R^s_{u,v} = 1$ if $s_u > s_v$ and $R_{u,v} = 0$ if $s_u \leq s_v$, a decoder D^{rank}_s for centrality score s estimates its rank score by $S_v = D^{rank}_s(\mathbf{z}_{\text{GNN},v})$. The probability of estimated rank order is defined by the sigmoid function $\tilde{R}^s_{u,v} = \frac{\exp(S_u - S_v)}{1+\exp(S_u - S_v)}$. Then predicting the relative order between pairs of nodes could be formalized as a binary classification problem with the loss:

$$\mathscr{L}_{\text{ssl}} = -\sum_{s}\sum_{u,v\in\mathscr{V}}(R^{s}_{u,v}\log\tilde{R}^{s}_{u,v} + (1-R^{s}_{u,v})\log(1-\tilde{R}^{s}_{u,v})). \tag{18.15}$$

Different from peer works, Hu et al (2019c) does not consider any node feature but instead extract the node features directly from the graph topology, which includes: (1) degree that defines the local importance of a node; (2) core-number that defines the connectivity of the subgraph around a node; (3) collective influence that defines the neighborhood importance of a node; and (4) local clustering coefficient, which defines the connectivity of 1-hop neighborhood of a node. Then, the four features (after min-max normalization) are concatenated with a nonlinear transformation and fed into the GNN where (Hu et al, 2019c) uses the pretext tasks: centrality ranking, clustering recovery and edge prediction. Another innovative idea in (Hu et al, 2019c) is to choose a fix-tune boundary in the middle layer of GNNs. The GNN blocks below this boundary are fixed, while the ones above the boundary are fine-tuned. For downstream tasks that are closely related to the pre-trained tasks, a higher boundary is used.

Another important node-level structure property is the partition each node belongs after performing a graph partitioning method. In (You et al, 2020c), the pretext task is to train the GNNs to encode the node partition information. Graph partitioning is to partition the nodes of a graph into different groups such that the number of edges between each group is minimized. Given the node set $\mathscr{V}$, the edge set $\mathscr{E}$, and a preset number of partitions $p \in [1, |\mathscr{V}|]$, a graph partitioning algorithm (e.g., (Karypis and Kumar, 1995) as used in (You et al, 2020c)) will output a set of nodes $\{\mathscr{V}_{\text{par}_1}, ..., \mathscr{V}_{\text{par}_p} | \mathscr{V}_{\text{par}_i} \subset \mathscr{V}, i = 1, ..., p\}$. Then the classification loss is set exactly the same as:

$$\mathscr{L}_{\text{ssl}} = -\frac{1}{|\mathscr{V}|}\sum_{v_i\in\mathscr{V}}\ell_{\text{CE}}(\mathbf{z}_{\text{ssl},i}, \mathbf{y}_{\text{ssl},i}) \tag{18.16}$$

where $\mathbf{z}_{\text{ssl},i}$ denotes the embedding of node v_i and assuming that the partitioning label is a one-hot encoding $\mathbf{y}_{\text{ssl},i} \in \mathbb{R}^p$ with k-th entry as 1 and others as 0 if $v_i \in \mathscr{V}_{\text{par}_k}, i = 1, ..., |\mathscr{V}|, \exists k \in [1, p]$.

18.4.2 Feature-based Pretext Tasks

Node features are another important information that can be leveraged to provide extra supervision. Since the state-of-the-art GNNs suffer from over-smoothing (Chen et al, 2020c), the original feature information is partially lost after fed into the GNNs. In order to reduce the information loss in node embeddings, the pretext task in (Hu et al, 2020c; Jin et al, 2020d; Manessi and Rozza, 2020; Wang et al, 2017a; You et al, 2020c) is to first mask node features and let the GNN predict those features. More specifically, they randomly mask input node features by replacing them with special mask indicators and then apply GNNs to obtain the corresponding node embeddings. Finally a linear model is applied on top of embeddings to predict the corresponding masked node features. Assuming the set of nodes that are masked is

$\mathcal{V}_{\text{m}}$, then the self-supervised regression loss to reconstruct these masked features is:

$$\mathcal{L}_{\text{ssl}} = \frac{1}{|\mathcal{V}_{\text{m}}|} \sum_{v_i \in \mathcal{V}_{\text{m}}} \ell_{\text{MSE}}(\mathbf{z}_{\text{ssl},i}, \mathbf{x}_i) \tag{18.17}$$

To handle the high sparsity of the node features, it is beneficial to first perform feature dimensionality reduction on X (such as principle component analysis (PCA) used in (Jin et al, 2020d)). Additionally, instead of reconstructing node features, node embeddings could also be reconstructed from their corrupted version, such as in (Manessi and Rozza, 2020).

Contrary to the graph partitioning where nodes are grouped by the graph topology, in graph clustering the clusters of nodes are discovered based on their features (You et al, 2020c). In this way the pretext task can be designed to recover the node clustering assignment. Given the node set $\mathcal{V}$, the feature matrix X, and a preset number of clusters $p \in [1, |\mathcal{V}|]$ (or without if the clustering algorithm automatically learns the number of clusters) as input, the clustering algorithm will output a set of node clusters $\{\mathcal{V}_{\text{clu}_1}, \ldots, \mathcal{V}_{\text{clu}_p} | \mathcal{V}_{\text{clu}_i} \subset \mathcal{V}, i = 1, ..., p\}$ and assuming for node v_i, the partitioning label is a one-hot encoding $\mathbf{y}_{\text{ssl},i} \in \mathbb{R}^p$ with k-th entry as 1 and others as 0 if $v_i \in \mathcal{V}_{\text{clu}_k}, i = 1, ..., |\mathcal{V}|, \exists k \in [1, p]$. Then the loss is the same as Eq. equation 18.16.

Instead of focusing on individual nodes, pretext tasks have also been developed based on the relationship between pairs of nodes (Jin et al, 2021, 2020d). The basic idea is to retain the node pairwise feature similarity in the node embeddings from GNNs. Suppose $\mathcal{T}_s, \mathcal{T}_d$ denote the sets of node pairs having the highest and the lowest similarity:

$$\mathcal{T}_s = \{(v_i, v_j) | \text{ sim}(\mathbf{x}_i, \mathbf{x}_j) \text{ in top-B of } \{\text{sim}(\mathbf{x}_i, \mathbf{x}_b)\}_{b=1}^{B} \backslash \text{sim}(\mathbf{x}_i, \mathbf{x}_i), \forall v_i \in \mathcal{V}\}, \tag{18.18}$$

$$\mathcal{T}_d = \{(v_i, v_j) | \text{ sim}(\mathbf{x}_i, \mathbf{x}_j) \text{ in bottom-B of } \{\text{sim}(\mathbf{x}_i, \mathbf{x}_b)\}_{b=1}^{B} \backslash \text{sim}(\mathbf{x}_i, \mathbf{x}_i), \forall v_i \in \mathcal{V}\}, \tag{18.19}$$

where $\text{sim}(\mathbf{x}_i, \mathbf{x}_j)$ measures the cosine similarity of features between two nodes v_i, v_j and B is the number of top/bottom pairs selected for each node. Then the pretext task is to optimize the following regression loss:

$$\mathcal{L}_{\text{ssl}} = \frac{1}{|\mathcal{T}_s \cup \mathcal{T}_d|} \sum_{(v_i, v_j) \in \mathcal{T}_s \cup \mathcal{T}_d} \ell_{\text{MSE}}\big(f_w(|\mathbf{z}_{\text{GNN},i} - \mathbf{z}_{\text{GNN},j}|), \text{sim}(\mathbf{x}_i, \mathbf{x}_j)\big), \tag{18.20}$$

where f_w is a function mapping the difference between two node embeddings from GNNs to a scalar representing the similarity between them.

18.4.3 Hybrid Pretext Tasks

Instead of employing only the topology or only the feature information as the extra supervision, some pretext tasks combine them together as a hybrid supervision, or even utilize information from the known training labels.

A contrastive framework for unsupervised graph representation learning, GRACE, where two correlated graph views are generated by randomly performing corruption on attributes (masking node features) and topology (removing or adding graph edges) is proposed in (Zhu et al, 2020c). Then the GNNs are trained using a contrastive loss to maximize the agreement between node embeddings in these two views. In each iteration two graph views $\mathscr{G}^1 = \{A^1, X^1\}$ and $\mathscr{G}^2 = \{A^2, X^2\}$ are generated randomly according to the possible augmentation functions from an input graph $\mathscr{G} = \{A, X\}$.

The objective is to maximize the similarity of the same nodes in different views of the graph while minimizing the similarity of different nodes in the same or different views of the graph. Thus, if we denote the node embeddings in the two views as $Z^1_{\text{GNN}} = f_\theta(X^1, A^1), Z^2_{\text{GNN}} = f_\theta(X^2, A^2)$, then the contrastive NT-Xent loss is:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{P}^+|} \sum_{(v_i^1, v_i^2) \in \mathscr{P}^+} \ell_{\text{NT-Xent}}(Z^1_{\text{GNN}}, Z^2_{\text{GNN}}, \mathscr{P}^-), \tag{18.21}$$

where $\mathscr{P}^+$ includes positive pairs of (v_i^1, v_i^2) where v_i^1, v_i^2 correspond to the same node in different views, while $\mathscr{P}^- = \bigcup_{(v_i^1, v_i^2) \in \mathscr{P}^+} \mathscr{P}^-_{v_i^1}$ represents all sets of negative samples with $\mathscr{P}^-_{v_i^1}$ containing nodes different from v_i in the same view (intra-view negative pairs) or the other view (inter-view negative pairs).

More specifically, in the above, the two graph corruptions are removing edges and masking node features. In removing edges, a random masking matrix $M \in \{0,1\}^{|\mathscr{V}| \times |\mathscr{V}|}$ is randomly sampled whose entry is drawn from a Bernoulli distribution $M_{ij} \sim \mathscr{B}(1 - p_r)$ if $A_{ij} = 1$ for the original graph. p_r is the probability of each edge being removed. The resulting matrix can be computed as $A' = A \odot M$ creating the adjacency matrix of graph view $\mathscr{G}'$ from $\mathscr{G}$.

In masking node features, a random vector $\mathbf{m} \in \{0,1\}^d$ is utilized, where each dimension of m is independently drawn from a Bernoulli distribution with probability $1 - p_m$ and d is the dimension of the node features X. Then, the generated node features X' for graph view $\mathscr{G}'$ from $\mathscr{G}$ is computed by:

$$X' = [\mathbf{x}_1 \odot \mathbf{m}; \mathbf{x}_2 \odot \mathbf{m}; \cdots; \mathbf{x}_{|\mathscr{V}|} \odot \mathbf{m}], \tag{18.22}$$

where $[;]$ is the concatenation operator. Moreover, a modified version of the GRACE is proposed in (Zhu et al, 2021) where the whole contrastive procedure is the same as GRACE except that the graph augmentation is adaptively performed based on the importance of nodes and edges. Specifically, the probability of removing an edge between nodes v_i, v_j should reflect the importance of the edge (v_i, v_j) such that the augmentation function is more likely to corrupt unimportant edges while keeping

important connective structures intact in augmented views. Similarly the feature dimensions frequently appearing in influential nodes are seen as important and so are masked with lower probability.

The observation made in (Chen et al, 2020b) that nodes with further topological distance to the labeled nodes are more likely to be misclassified indicates the uneven distribution of the ability of GNNs to embed node features in the whole graph. However, existing graph contrastive learning methods ignore this uneven distribution, which motivates Chen et al (2020b) to propose the distance-wise graph contrastive learning (DwGCL) method that can adaptively augment the graph topology, sample the positive and negative pairs, and maximize the mutual information. The topology information gain (TIG) is calculated based on Group PageRank and node features to describe the task information effectiveness that the node obtains from labeled nodes along the graph topology. By ranking the performance of GNNs on nodes according to their TIG values with/without contrastive learning, it is found that contrastive learning mainly improves the performance on nodes that are topologically far away from the labeled nodes. Based on the above finding, Chen et al (2020b) proposes to: 1) perturb the graph topology by augmenting nodes according to their TIG value; 2) sampling the positive and negative pairs considering local/-global topology distance and node embedding distance; and 3) assigning different weights to nodes in the self-supervised loss based on their TIG rankings. Results demonstrate the performance improvement of this distance-wise graph contrastive learning over the typical contrastive learning approach.

Another special supervision information to exploit is the prediction results of the model itself. Sun et al (2020c) leverages the multi-stage training framework to utilize the information of the pseudo labels generated by predictions in the next rounds of training. The multi-stage training algorithm repeatedly adds the most confident predictions of each class to the label set and re-utilizes these pseudo labeled data to train the GNNs. Furthermore, a self-checking mechanism based on DeepCluster (Caron et al, 2018) is proposed to guarantee the precision of labeled data. Assuming that the cluster assignment for node v_i is $\mathbf{c}_i \in \{0,1\}^p$ (here the number of clusters is assumed to equal to the number of predefined classes p in the downstream classification task) and the centroid matrix $C \in R^{d' \times p}$ represents the feature of each cluster, then we obtain the cluster assignment $\mathbf{c}_i$ for each node v_i by optimizing:

$$\min_{C} \frac{1}{\mathscr{V}} \sum_{v_i \in \mathscr{V}} \min_{\mathbf{c}_i \in \{0,1\}^p} ||\mathbf{z}_{\mathrm{GNN},i} - C\mathbf{c}_i||_2^2, \quad s.t. \quad \mathbf{c}_i^{\mathrm{T}} \mathbf{1}_p = 1. \tag{18.23}$$

After applying DeepCluster to group nodes into multiple clusters, an aligning mechanism is used to assign nodes in each cluster to their corresponding class defined by downstream tasks. For each cluster $k \in [1, p]$ in unlabeled data, the computation of aligning mechanism is:

$$c^k = \arg\min_{m} ||\kappa_k - \mu_m||^2, \tag{18.24}$$

where μ_m denotes the centroid of class m in labeled data, κ_k denotes the centroid of cluster k in unlabeled data and c^k represents the aligned class that has the closest distance to the centroid κ_k of the cluster k among all centroids of classes in the original labeled data. Note that the self-checking can be directly performed by comparing the distance of each unlabeled node to centroids of classes in labeled data. However, directly checking in this naïve way is very time consuming.

18.5 Graph-level SSL Pretext Tasks

After having just presented the node-level SSL pretext tasks, in this section we focus on the graph-level SSL pretext tasks where we desire the node embeddings coming from the GNNs to encode information of graph-level properties.

18.5.1 Structure-based Pretext Tasks

As the counterpart of the nodes in the graph, the edges encode abundant information of the graph, which can also be leveraged as an extra supervision to design pretext tasks. The pretext task in (Zhu et al, 2020a) is to recover the graph topology, i.e., predict edges, after randomly removing edges in the graph. After node embeddings $\mathbf{z}_{\text{GNN},i}$ is obtained for each node v_i, the probability of the edge between any pair of nodes v_i, v_j is calculated by their feature similarity as follows:

$$A'_{ij} = \text{sigmoid}(\mathbf{z}_{\text{GNN},i}(\mathbf{z}_{\text{GNN},j})^\top), \tag{18.25}$$

and the weighted cross-entropy loss is used during training, which is defined as:

$$\mathscr{L}_{\text{ssl}} = -\sum_{v_i,v_j \in \mathscr{V}} W(A_{ij}\log A'_{ij}) + (1-A_{ij})\log(1-A'_{ij}), \tag{18.26}$$

where W is the weight hyperparameter used for balancing two classes; which are node pairs having an edge and node pairs without an edge between them.

As it is known that unclean graph structure usually impedes the applicability of GNNs (Cosmo et al, 2020; Jang et al, 2019). A method that trains the GNNs by downstream supervised tasks based on the cleaned graph structure reconstructed from completing a self-supervised pretext task is introduced in (Fatemi et al, 2021). The self-supervised pretext task aims to train a separate GNN to denoise the corrupted node feature $\hat{X}$ generated by either randomly zeroing some dimensions of the original node feature X when having binary features or by adding independent Gaussian noise when X is continuous. Two methods are used to generate the initial graph adjacency matrix $\tilde{A}$. The first method Full Parametrization (FP) treats every entry in $\tilde{A}$ as a parameter and directly optimizes its $|\mathscr{V}|^2$ parameters by denoising the corrupted feature $\hat{X}$. The second method MLP-kNN considers a mapping function

kNN(MLP(X)), where a multilayer perceptron (i.e., MLP($\cdot$)) updates the original node features and kNN($\cdot$) produces a sparse matrix by selecting top-k similar nodes to each node and adds edges between them. Then, the generated initial adjacency matrix $\tilde{A}$ is normalized and symmetrized into a new adjacency matrix A as follows:

$$A = D^{-\frac{1}{2}} \frac{\tilde{P}(\tilde{A}) + \tilde{P}(\tilde{A})^\top}{2} D^{-\frac{1}{2}}, \tag{18.27}$$

where $\tilde{P}$ is a function with a non-negative range to ensure the positivity of every entry in A. In MLP-kNN method, $\tilde{P}$ is the element-wise ReLU function. However, the ReLU function could result in the gradient flow problem in the FP method, thus the element-wise ELU function followed by an addition of 1 to avoid the problem of gradient flow is used instead. Next, a separate GNN-based encoder takes noisy node features $\hat{X}$ and the new normalized adjacency matrix A as input and output the updated node features $\hat{Z} = \text{GNN}(\hat{X}, A)$. The parameters in FP and MLP-kNN used for generating the initial adjacency matrix $\tilde{A}$ is optimized by:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{V}_{\text{m}}|} \sum_{v_i \in \mathscr{V}_{\text{m}}} \ell_{\text{MSE}}(\mathbf{x}_i, \hat{\mathbf{z}}_i), \tag{18.28}$$

where $\hat{\mathbf{z}}_i = \hat{Z}[i,:]^\top$ is the noisy embedding vector of the node v_i obtained by the separate GNN-based encoder. The optimized parameters in FP and MLP-kNN leads to the generation of more cleaned graph adjacency matrix, which in turn results in the better performance in the downstream tasks.

In addition to the graph edges and the adjacency matrix, topological distance between nodes is another important global structure property in graph. The pretext task in (Peng et al, 2020) is to recover the topological distance between nodes. More specifically, they leverage the shortest path length between nodes denoted as p_{ij} between nodes v_i and v_j, but this could be replaced with any other distance measure. Then, they define the set $\mathscr{C}_i^k$ as all the nodes having the shortest path distance of length k from node v_i. More formally, this is defined as:

$$\mathscr{C}_i = \mathscr{C}_i^1 \cup \mathscr{C}_i^2 \cup \cdots \cup \mathscr{C}_i^{\delta_i}, \quad \mathscr{C}_i^k = \{v_j | d_{ij} = k\}, \quad k = 1, 2, \cdots, \delta_i, \tag{18.29}$$

where δ_i is the upper bound of the hop count from other nodes to v_i, d_{ij} is the length of the path p_{ij}, and $\mathscr{C}_i$ is the union of all the k-hop shortest path neighbor sets C_i^k. Based on these sets, one-hot encodings $\mathbf{d}_{ij} \in \mathbb{R}^{\delta_i}$ are created for pairs of nodes v_i, v_j, where $v_j \in \mathscr{C}_i$, according to their distance d_{ij}. Then, the GNN model is guided to extract node embeddings that encode node topological distance as follows:

$$\mathscr{L}_{\text{ssl}} = \sum_{v_i \in \mathscr{V}} \sum_{v_j \in \mathscr{C}_i} \ell_{\text{CE}}(f_w(|\mathbf{z}_{\text{GNN},i} - \mathbf{z}_{\text{GNN},j}|), \mathbf{d}_{ij}), \tag{18.30}$$

where f_w is a function mapping the difference between two node embeddings to the probabilities of pairs of nodes belonging to the corresponding category of the topological distance. Since the number of the categories depends on the upper bound

of the hop count (topological distance) but precisely determining this upper bound is time-consuming for a big graph, it is assumed that the number of hops (distance) is under control based on small-world phenomenon (Newman, 2018) and is further divided into several major categories that clearly discriminates the dissimilarity and partly tolerates the similarity. Experiments demonstrate that dividing the topological distance into four categories: $\mathscr{C}_i^1, \mathscr{C}_i^2, \mathscr{C}_i^3, \mathscr{C}_i^k (k \geq 4)$ achieves the best performance (i.e., δ_i=4). Another problem is that the number of nodes that are close to the focal node v_i is much less than the nodes that are further away (i.e., the magnitude of $\mathscr{C}_i^{\delta_i}$ will be significantly larger than other sets). To circumvent this imbalance problem, node pairs are sampled with adaptive ratio.

Network motifs are recurrent and statistically significant subgraphs of a larger graph and (Zhang et al, 2020f) designs a pretext task to train a GNN encoder that can automatically extract graph motifs. The learned motifs are further leveraged to generate informative subgraphs used in graph-subgraph contrastive learning. Firstly, a GNN-based encoder f_θ and a m-slot embedding table $\{\mathbf{m}_1, ..., \mathbf{m}_m\}$ denoting m cluster centers of m motifs are initialized. Then, a node affinity matrix $U \in \mathbb{R}^{|\mathscr{V}|\times|\mathscr{V}|}$ is calculated by softmax normalization on the embedding similarity $\mathscr{D}(\mathbf{z}_{\text{GNN},i}, \mathbf{z}_{\text{GNN},j})$ between nodes i, j as in Eq. equation 18.13. Afterwards, spectral clustering (VON-LUXBURG, 2007) is performed on U to generate different groups, within which $n_{\mathscr{G}}$ connected components that have more than three nodes are collected as the sampled subgraphs from the graph $\mathscr{G}$ and their embeddings are calculated by applying READOUT function. For each subgraph, its cosine similarity to each of the m motifs is calculated to obtain a similarity metric $S \in \mathbb{R}^{m\times n_{\mathscr{G}}}$. To produce semantic-meaningful subgraphs that are close to motifs, the top 10% most similar subgraphs to each motif are selected based on the similarity metric S and are collected into a set $\mathscr{G}^{\text{top}}$. The affinity values in U between pairs of nodes in each of these subgraphs are increased by optimizing the loss:

$$\mathscr{L}_1 = -\frac{1}{|\mathscr{G}^{\text{top}}|} \sum_{i=1}^{|\mathscr{G}^{\text{top}}|} \sum_{(v_j, v_k)\in\mathscr{G}_i^{\text{top}}} U[j,k]. \tag{18.31}$$

The optimization of the above loss forces nodes in motif-like subgraphs to be more likely to be grouped together in spectral clustering, which leads to more subgraph samples aligned with the motifs. Next, the embedding table of motifs is optimized based on the sampled subgraphs. The assignment matrix $Q \in \mathbb{R}^{m\times n_{\mathscr{G}}}$ is found by maximizing similarities between embeddings and its assigned motif:

$$\max_Q \; Tr(Q^{\mathrm{T}} S) - \frac{1}{\lambda} \sum_{i,j} Q[i,j] \log Q[i,j], \tag{18.32}$$

where the second term controlled by hyperparameter λ is to avoid all representations collapsing into a single cluster center. After the cluster assignment matrix Q is obtained, the GNN-based encoder and the motif embedding table are trained, which is equivalent to a supervised m-class classification problem with labels Q and the prediction distribution $\widetilde{S}$ obtained by applying a column-wise softmax normaliza-

tion with temperature τ:

$$\mathscr{L}_2 = -\frac{1}{n_{\mathscr{G}}} \sum_{i=1}^{n_{\mathscr{G}}} \ell_{\text{CE}}(\mathbf{q}_i, \tilde{\mathbf{s}}_i), \tag{18.33}$$

where $\mathbf{q}_i = Q[:,i]$ and $\tilde{\mathbf{s}}_i = \widetilde{S}[:,i]$ denote the assignment distribution and predicted distribution for the subgraph i, respectively. Optimizing Eq. equation 18.33 jointly enhances the ability of GNN encoder to extract subgraphs that are similar to motifs and improves the embeddings of motifs. The last step is to train the GNN-based encoder by a classification task where subgraphs are reassigned back to their corresponding graphs. Note that the subgraphs are generated by the Motif-guided extractor, which are more likely to capture higher-level semantic information compared with randomly sampled subgraphs. The whole framework is trained jointly by weighted combining $\mathscr{L}_1, \mathscr{L}_2$ and the contrastive loss.

Aside from the network motifs, other subgraph structures can be leveraged to provide extra supervision in designing pretext tasks. In (Qiu et al, 2020a), an r-ego network for a certain vertex is defined as the subgraph induced by nodes that have shortest path with length shorter than r. Then a random walk with restart is initiated at ego vertex v_i and the subgraph induced by nodes that are visited during the random walk starting at v_i are used as the augmented version of the r-ego network. First, two augmented r-ego networks centered around vertex v_i are obtained by performing the random walk twice (i.e., $\mathscr{G}_i$ and $\mathscr{G}_i^+$), which are defined as a positive pair since they come from the same r-ego network. In comparison, a negative pair corresponds to two subgraphs augmented from different r-ego networks (e.g., one coming from v_i and another coming from v_j resulting in random walk induced subgraphs $\mathscr{G}_i$ and $\mathscr{G}_j$, respectively). Based on the above defined positive and negative subgraph pairs, a contrastive loss is set up to optimize the GNNs as follows:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{P}^+|} \sum_{(\mathscr{G}_i, \mathscr{G}_i^+) \in \mathscr{P}^+} \ell_{\text{NT-Xent}}(Z^1_{\text{ssl}}, Z^2_{\text{ssl}}, \mathscr{P}^-), \tag{18.34}$$

where $Z^1_{\text{ssl}}, Z^2_{\text{ssl}}$ denotes the GNN-based graph embeddings and specifically here the two different views are the same $Z^1_{\text{ssl}} = Z^2_{\text{ssl}}$. $\mathscr{P}^+$ contains positive pairs of subgraphs $(\mathscr{G}_i, \mathscr{G}_i^+)$ sampled by random walk starting at the same ego vertex v_i in the same graph while $P^- = \bigcup_{(\mathscr{G}_i, \mathscr{G}_i^+) \in \mathscr{P}^+} \mathscr{P}^-_{\mathscr{G}_i}$ represents all sets of negative samples. Specifically $\mathscr{P}^-_{\mathscr{G}_i}$ represents subgraphs sampled by random walk starting at either different ego vertex from v_i in $\mathscr{G}$ or directly sampled by random walk in different graphs from $\mathscr{G}$.

Although Graph Attention Network (GAT) (Petar et al, 2018) achieves performance improvements over the original GCN (Kipf and Welling, 2017b), there is little understanding of what graph attention learns. To this end, Kim and Oh (2021) proposes a specific pretext task to leverage the edge information to supervise what graph attention learns:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{E} \cup \mathscr{E}^-|} \sum_{(j,i) \in \mathscr{E} \cup \mathscr{E}^-} 1\big((j,i) \in \mathscr{E}\big) \cdot \log \chi_{ij} + 1\big((j,i) \in \mathscr{E}^-\big) \log(1 - \chi_{ij}), \tag{18.35}$$

where $\mathscr{E}$ is the set of edges, $\mathscr{E}^-$ is the sampled set of node pairs without edges, and χ_{ij} is the edge probability between node i, j calculated from their embeddings. Based on two primary edge attentions, the GAT attention (shortly as GO) (Petar et al, 2018) and the dot-product attention (shortly as DP) (Luong et al, 2015), two advanced attention mechansims, SuperGAT$_{\text{SD}}$ (Scaled Dot-product, shortly as SD) and SuperGAT$_{\text{MX}}$ (Mixed GO and DP, shortly as MX) are proposed:

$$e_{ij,\text{SD}} = e_{ij,\text{DP}} / \sqrt{F}, \qquad \chi_{ij,\text{SD}} = \sigma(e_{ij,\text{SD}}), \tag{18.36}$$

$$e_{ij,\text{MX}} = e_{ij,\text{GO}} \cdot \sigma(e_{ij,\text{DP}}), \quad \chi_{ij,\text{MX}} = \sigma(e_{ij,\text{DP}}), \tag{18.37}$$

where σ denotes the sigmoid function taking the edge weight e_{ij} and calculating the edge probability χ_{ij}. SuperGAT$_{\text{SD}}$ divides the dot-product of edge $e_{ij,\text{DP}}$ by a square root of dimension as Transformer (Vaswani et al, 2017) to prevent some large values from dominating the entire attention after softmax. SuperGAT$_{\text{MX}}$ multiplies GO and DP attention with sigmoid, which is motivated by the gating mechanism of Gated Recurrent Units (GRUs) (Cho et al, 2014a). Since DP attention with the sigmoid denotes the edge probability, multiplying $\sigma(e_{ij,\text{DP}})$ in calculating $e_{ij,\text{MX}}$ can softly drop neighbors that are not likely linked while implicitly assigning importance to the remaining nodes. $e_{ij,\text{DP}}, e_{ij,\text{GO}}$ are the weight of edge (i, j) used to calculate the GO and DP attention. Results disclose several insightful discovers including the GO attention learns label-agreement better than DP, whereas DP predicts edge presence better than GO, and the performance of the attention mechanism is not fixed but depends on homophily and average degree of the specific graph.

The topological information can also be generated manually for designing pretext tasks. Gao et al (2021) proposes to encode the transformation information between two different graph topologies in the representations of nodes obtained by GNNs. First, they transform the original graph adjacency matrix A into $\hat{A}$ by randomly adding or removing edges from the original edge set. Then, by feeding the original and transformed graph topology and the node feature matrix into any GNN-based encoder, the feature representation $Z_{\text{GNN}}, \hat{Z}_{\text{GNN}}$ before and after topology transformation are calculated and their difference $\Delta Z \in \mathbb{R}^{N \times F'}$ is defined as:

$$\Delta Z = \hat{Z}_{\text{GNN}} - Z_{\text{GNN}} = [\Delta \mathbf{z}_{\text{GNN},1}, ..., \Delta \mathbf{z}_{\text{GNN},N}]^\top = [\hat{\mathbf{z}}_{\text{GNN},1} - \mathbf{z}_{\text{GNN},1}, ..., \hat{\mathbf{z}}_{\text{GNN},N} - \mathbf{z}_{\text{GNN},N}]^\top. \tag{18.38}$$

Next they predict the topology transformation between node v_i and v_j through the node-wise feature difference ΔZ by constructing the edge representation as:

$$\mathbf{e}_{ij} = \frac{\exp(-(\Delta \mathbf{z}_i - \Delta \mathbf{z}_j) \odot (\Delta \mathbf{z}_i - \Delta \mathbf{z}_j))}{||\exp(-(\Delta \mathbf{z}_i - \Delta \mathbf{z}_j) \odot (\Delta \mathbf{z}_i - \Delta \mathbf{z}_j))||}, \tag{18.39}$$

where $\odot$ denotes the Hardamard product. This edge representation $\mathbf{e}_{ij}$ is then fed into an MLP for the prediction of the topological transformation, which includes

four classes: edge addition, edge deletion, keeping disconnection and keeping connection between each pair of nodes. Thus, the GNN-based encoder is trained by:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{V}|^2} \sum_{v_i, v_j \in \mathscr{V}} \ell_{\text{CE}}(\text{MLP}(\mathbf{e}_{ij}), \mathbf{t}_{ij}) \tag{18.40}$$

where we denote the topological transformation category between nodes v_i and v_j as one-hot encoding $\mathbf{t}_{ij} \in \mathbb{R}^4$.

18.5.2 Feature-based Pretext Tasks

Typically, graphs does not come with any feature information and here the graph-level features refer to the graph embeddings obtained after applying a pooling layer on all node embeddings from GNNs.

GraphCL (You et al, 2020b) designs the pretext task to first augment graphs by four different augmentations including node dropping, edge perturbation, attribute masking and subgraph extraction and then maximize the mutual information of the graph embeddings between different augmented views generated from the same original graph while also minimizing the mutual information of the graph embeddings between different augmented views generated from different graphs. The graph embeddings Z_{ssl} are obtained through any permutational-invariant READOUT function on node embeddings followed by applying an adaptation layer. Then the mutual information is maximized by optimizing the following NT-Xent contrastive loss:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{P}^+|} \sum_{(\mathscr{G}_i, \mathscr{G}_j) \in \mathscr{P}^+} \ell_{\text{NT-Xent}}(Z^1_{\text{ssl}}, Z^2_{\text{ssl}}, \mathscr{P}^-), \tag{18.41}$$

where $Z^1_{\text{ssl}}, Z^2_{\text{ssl}}$ represent graph embeddings under two different views. The view could be the original view without any augmentation or the one generated from applying four different augmentations. $\mathscr{P}^+$ contains positive pairs of graphs $(\mathscr{G}_i, \mathscr{G}_j)$ augmented from the same original graph while $\mathscr{P}^- = \bigcup_{(\mathscr{G}_i, \mathscr{G}_j) \in \mathscr{P}^+} \mathscr{P}^-_{\mathscr{G}_i}$ represents all sets of negative samples. Specifically $\mathscr{P}^-_{\mathscr{G}_i}$ contains graphs augmented from the graph different from $\mathscr{G}_i$. Numerical results demonstrate that the augmentation of edge perturbations benefits social networks but hurts biochemical molecules. Applying attribute masking achieves better performance in denser graphs. Node dropping and subgraph extraction are generally beneficial across all datasets.

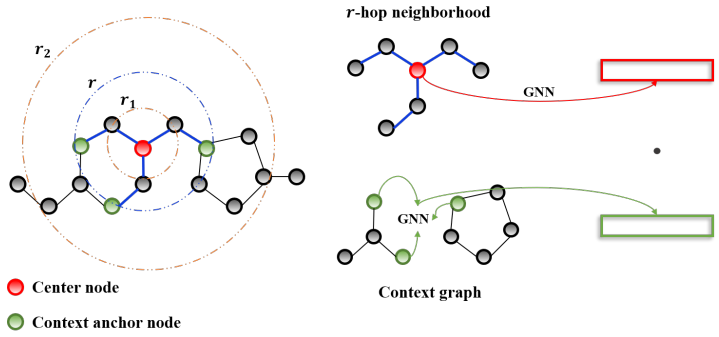

Fig. 18.6: An example of a context and r-neighborhood graph.

18.5.3 Hybrid Pretext Tasks

One way to use the information of the training nodes in designing pretext tasks is developed in (Hu et al, 2020c) where the context concept is raised. The goal of this work is to pre-train a GNN so that it maps nodes appearing in similar graph structure contexts to nearby embeddings. For every node v_i, the r-hop neighborhood of v_i contains all nodes and edges that are at most r-hops away from v_i in the graph. The context graph of v_i is a subgraph between r_1-hops and r_2-hops away from node v_i. It is required that $r_1 < r$ so that some nodes are shared between the neighborhood and the context graph, which is referred to as context anchor nodes. Examples of neighborhood and context graphs are shown in Fig. 18.6. Two GNN encoders are set up: the main GNN encoder is to get the node embedding $\mathbf{z}^r_{\text{GNN},i}$ based on their r-hop neighborhood node features and the context GNN is to get the node embeddings of every other node in the context anchor node set, which are then averaged to get the node context embedding $\mathbf{c}_i$. Then Hu et al (2020c) used negative sampling to jointly learn the main GNN and the context GNN. In the optimization process, positive samples refer to the situation when the center node of the context and the neighborhood graphs is the same while the negative samples refer to the situation when the center nodes of the context and the neighborhood graphs are different. The learning objective is a binary classification of whether a particular neighborhood and a particular context graph have the same center node and the negative likelihood loss is used as follows:

$$\mathscr{L}_{\text{ssl}} = -\Big(\frac{1}{|\mathscr{K}|}\sum_{(v_i,v_j)\in\mathscr{K}} (y_i \log(\sigma((\mathbf{z}^r_{\text{GNN},i})^\top \mathbf{c}_j)) + (1-y_i)\log(1-\sigma((\mathbf{z}^r_{\text{GNN},i})^\top \mathbf{c}_j))\Big) \tag{18.42}$$

where $y_i = 1$ for the positive sample where $i = j$ while $y_i = 0$ for the negative sample where $i \neq j$, with $\mathscr{K}$ denoting the set of positive and negative pairs, and σ is the sigmoid function computing the probability.

Similar idea to employ the context concept in completing pretext tasks is also proposed in (Jin et al, 2020d). Specifically, the context here is defined as:

$$\mathbf{y}_{ic} = \frac{|\Gamma_{\mathscr{V}_l}(v_i,c)| + |\Gamma_{\mathscr{V}_u}(v_i,c)|}{|\Gamma_{\mathscr{V}_l}(v_i)| + |\Gamma_{\mathscr{V}_u}(v_i)|}, c = 1,...,l, \tag{18.43}$$

where $\mathscr{V}_u$ and $\mathscr{V}_l$ denote the unlabeled and labeled node set, $\Gamma_{\mathscr{V}_u}(v_i)$ denotes the unlabeled nodes that are adjacency to node v_i, $\Gamma_{\mathscr{V}_u}(v_i,c)$ denotes the unlabeled nodes that have been assigned class c and are adjacency to node v_i, $\mathscr{N}_{\mathscr{V}_l}(v_i)$ denotes the labeled nodes that are adjacency to node v_i, $\Gamma_{\mathscr{V}_l}(v_i,c)$ denotes the labeled nodes that are adjacency to node v_i and of class c. To generate labels for the unlabeled nodes so as to calculate the context vector $\mathbf{y}_i$ for each node v_i, label propagation (LP) (ZHU, 2002) or the iterative classification algorithm (ICA) (Neville and Jensen, 2000) is used to construct pseudo labels for unlabeled nodes in $\mathscr{V}_u$. Then the pretext task is approached by optimizing the following loss function:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{V}|} \sum_{v_i \in \mathscr{V}} \ell_{\text{CE}}(\mathbf{z}_{\text{ssl},i}, \mathbf{y}_i), \tag{18.44}$$

The main issue of the above pretext task is the error caused by generating labels from LP or ICA. The paper (Jin et al, 2020d) further proposed two methods to improve the above pretext task. The first method is to replace the procedure of assigning labels of unlabeled nodes based on only one method such as LP or ICA with assigning labels by ensembling results from multiple different methods. Their second method treats the initial labeling from LP or ICA as noisy labels, and then leverages an iterative approach (Han et al, 2019) to improve the context vectors, which leads to significant improvements based on this correction phase.

One previous pretext task is to recover the topological distance between nodes. However, calculating the distance of the shortest path for all pairs of nodes even after the sampling is time-consuming. Therefore, Jin (Jin et al, 2020d) replaces the pairwise distance between nodes with the distance between nodes and their corresponding clusters. For each cluster, a fixed set of anchor/center nodes is established. For each node, its distance to this set of anchor nodes is calculated. The pretext task is to extract node features that encode the information of this node2cluster distance. Suppose k clusters are obtained by applying the METIS graph partitioning algorithm (Karypis and Kumar, 1998) and the node with the highest degree is assumed to be the center of the corresponding cluster, then each node v_i will have a cluster distance vector $\mathbf{d}_i \in \mathbb{R}^k$ and the distance-to-cluster pretext task is completed by optimizing:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{V}|} \sum_{v_i \in \mathscr{V}} \ell_{\text{MSE}}(\mathbf{z}_{\text{ssl},i}, \mathbf{d}_i), \tag{18.45}$$

Aside from the graph topology and the node features, the distribution of the training nodes and their training labels are another valuable source of information for designing pretext tasks. One of the pretext tasks in (Jin et al, 2020d) is to require the node embeddings output by GNNs to encode the information of the topological

distance from any node to the training nodes. Assuming that the total number of classes is p and for class $c \in \{1,...,p\}$ and the node $v_i \in \mathcal{V}$, the average, minimum and maximum shortest path length from v_i to all labeled nodes in class c is calculated and denoted as $\mathbf{d}_i \in \mathbb{R}^{3p}$, then the objective is to optimize the same regression loss as defined in Eq. equation 18.45

The generating process of networks encodes abundant information for designing pretext tasks. Hu et al (2020d) proposes the GPT-GNN framework for generative pre-training of GNNs. This framework performs attribute and edge generation to enable the pre-trained model to capture the inherent dependency between node attributes and graph structure. Assuming that the likelihood over this graph by this GNN model is $p(\mathcal{G};\theta)$ which represents how the nodes in $\mathcal{G}$ are attributed and connected, GPT-GNN aims to pre-train the GNN model by maximizing the graph likelihood, i.e., $\theta^* = \max_\theta p(\mathcal{G};\theta)$. Given a permutated order, the log likelihood is factorized autoregressively - generating one node per iteration as:

$$\log p_\theta(X,\mathcal{E}) = \sum_{i=1}^{|\mathcal{V}|} \log p_\theta(\mathbf{x}_i, \mathcal{E}_i | X_{<i}, \mathcal{E}_{<i}) \tag{18.46}$$

For all nodes that are generated before the node i, their attributes $X_{<i}$, and the edges between these nodes $\mathcal{E}_{<i}$ are used to generate a new node v_i, including both its attribute $\mathbf{x}_i$ and its connections with existing nodes $\mathcal{E}_i$. Instead of directly assuming that $\mathbf{x}_i, \mathcal{E}_i$ are independent, they devise a dependency-aware factorization mechanism to maintain the dependency between node attributes and edge existence. The generation process can be decomposed into two coupled parts: (1) generating node attributes given the observed edges, and (2) generating the remaining edges given the observed edges and the generated node attributes. For computing the loss of attribute generation, the generated node feature matrix X is corrupted by masking some dimensions to obtain the corrupted version $\hat{X}^{\text{Attr}}$ and further fed together with the generated edges into GNNs to get the embeddings $\hat{Z}^{\text{Attr}}_{\text{GNN}}$. Then, the decoder $\text{Dec}^{\text{Attr}}(\cdot)$ is specified, which takes $\hat{Z}^{\text{Attr}}_{\text{GNN}}$ as input and outputs the predicted attributes $\text{Dec}^{\text{Attr}}(\hat{Z}^{\text{Attr}}_{\text{GNN}})$. The attribute generation loss is:

$$\mathcal{L}^{\text{Attr}}_{\text{ssl}} = \frac{1}{|\mathcal{V}|} \sum_{v_i \in \mathcal{V}} \ell_{\text{MSE}}(\text{Dec}^{\text{Attr}}(\hat{\mathbf{z}}^{\text{Attr}}_{\text{GNN},i}), \mathbf{x}_i), \tag{18.47}$$

where $\hat{\mathbf{z}}^{\text{Attr}}_{\text{GNN},i} = \hat{Z}^{\text{Attr}}_{\text{GNN}}[i,:]^\top$ denotes the decoded embedding of node v_i. For computing the loss of edge reconstruction, the original generated node feature matrix X is directly fed together with the generated edges into GNNs to get the embeddings $Z^{\text{Edge}}_{\text{GNN}}$. Then the contrastive NT-Xent loss is calculated:

$$\mathcal{L}^{\text{Edge}}_{\text{ssl}} = \frac{1}{|\mathcal{P}^+|} \sum_{(v_i,v_j)\in\mathcal{P}^+} \ell_{\text{NT-Xent}}(Z^{\text{Edge}}_{\text{GNN}}, Z^{\text{Edge}}_{\text{GNN}}, \mathcal{P}^-), \tag{18.48}$$

where $\mathscr{P}^+$ contains positive pairs of connected nodes (v_i, v_j) while $\mathscr{P}^- = \bigcup_{(v_i,v_j)\in\mathscr{P}^+} \mathscr{P}^-_{v_i}$ represents all sets of negative samples and $P^-_{v_i}$ contains all nodes that are not directly linked with node v_i. Note here two views are set equal, i.e., $Z^1 = Z^2 = Z^{\text{Edge}}_{\text{GNN}}$.

18.6 Node-graph-level SSL Pretext Tasks

All the above pretext tasks are designed based on either the node or the graph level supervision. However, there is another final line of research combining these two sources of supervision to design pretext tasks, which we summarize in this section.

Veličković et al (2019) proposed to maximize the mutual information between representations of high-level graphs and low-level patches. In each iteration, a negative sample $\hat{X}, \hat{A}$ is generated by corrupting the graph through shuffling node features and removing edges. Then a GNN-based encoder is applied to extract node representations Z_{GNN} and $\hat{Z}_{\text{GNN}}$, which are also named as the local patch representations. The local patch representations are further fed into an injective readout function to get the global graph representations $\mathbf{z}_{\text{GNN},\mathscr{G}} = \text{READOUT}(Z_{\text{GNN}})$. Then the mutual information between Z_{GNN} and $\mathbf{z}_{\text{GNN},\mathscr{G}}$ is maximized by minimizing the following loss function:

$$\mathscr{L}_{\text{ssl}} = \frac{1}{|\mathscr{P}^+| + |\mathscr{P}^-|}\Big(\sum_{i=1}^{|\mathscr{P}^+|} \mathbb{E}_{(X,A)}[\log \sigma(\mathbf{z}^\top_{\text{GNN},i} W \mathbf{z}_{\text{GNN},\mathscr{G}})] + \sum_{j=1}^{|\mathscr{P}^-|} \mathbb{E}_{(\hat{X},\hat{A})}[\log(1-\sigma(\tilde{\mathbf{z}}^\top_{\text{GNN},i} W \mathbf{z}_{\text{GNN},\mathscr{G}}))]\Big), \tag{18.49}$$

where $|\mathscr{P}^+|$ and $|\mathscr{P}^-|$ are the number of the positive and negative pairs, σ stands for any nonlinear activation function and PReLU is used in (Veličković et al, 2019), $\mathbf{z}^\top_{\text{GNN},i} W \mathbf{z}_{\text{GNN},\mathscr{G}}$ calculates the weighted similarity between the patch representation centered at node v_i and the graph representation. A linear classifier is followed up to classify nodes after the above contrastive pretext task.

Similar to (Veličković et al, 2019) where the mutual information between the patch representations and the graph representations is maximized, Hassani and Khasahmadi (2020) proposed another framework of contrasting the node representations of one view and the graph representations of another view. The first view is the original graph and the second view is generated by a graph diffusion matrix. The heat and personalized PageRank (PPR) diffusion matrix are considered, which are:

$$S^{\text{heat}} = \exp(tAD^{-1} - t), \tag{18.50}$$

$$S^{\text{PPR}} = \alpha(\mathbf{I}_n - (1-\beta)D^{-1/2}AD^{-1/2})^{-1}, \tag{18.51}$$

where β denotes teleport probability, t is the diffusion time, and D is the diagonal degree matrix. After D is obtained, two different GNN encoders followed by a

shared projection head are applied on nodes in the original graph adjacency matrix and the generated diffusion matrix to get two different node embeddings Z^1_{GNN} and Z^2_{GNN}. Two different graph embeddings $\mathbf{z}^1_{\text{GNN},\mathcal{G}}$ and $\mathbf{z}^2_{\text{GNN},\mathcal{G}}$ are further obtained by applying a graph pooling function to the node representations (before the projection head) and followed by another shared projection head. The mutual information between nodes and graphs in different views is maximized through:

$$\mathscr{L}_{\text{ssl}} = -\frac{1}{|\mathscr{V}|}\sum_{v_i \in V}(\text{MI}(\mathbf{z}^1_{\text{GNN},i}, \mathbf{z}^2_{\text{GNN},\mathcal{G}}) + \text{MI}(\mathbf{z}^2_{\text{GNN},i}, \mathbf{z}^1_{\text{GNN},\mathcal{G}})), \tag{18.52}$$

where the MI represents the mutual information estimator and four estimators are explored, which are noise-contrastive estimator, Jensen-Shannon estimator, normalized temperature-scaled cross-entropy, and Donsker-Varadhan representation of the KL-divergence. Note that the mutual information in Eq. equation 18.52 is averaged over all graphs in the original work (Hassani and Khasahmadi, 2020). Additionally, their results demonstrate that Jensen-Shannon estimator achieves better results across all graph classification tasks, whereas in the node classification task, noise contrastive estimation achieves better results. They also discover that increasing the number of views does not increase the performance on downstream tasks.

18.7 Discussion

Existing methods employing self-supervision to graph neural networks achieve performance improvements and numerous insightful results are also discovered in the meantime. While most of the self-supervised pretext tasks are helpful for the downstream tasks, there are still a fair proportion of pretext tasks that bring weak or even fail to boost the performance (Gao et al, 2021; Jin et al, 2020d; Manessi and Rozza, 2020; You et al, 2020c). This is either because these pretext tasks are highly unrelated to the primary task, i.e., the encoded features useful for pretext tasks are useless or even harmful (Manessi and Rozza, 2020) for downstream tasks or because the information learned from completing pretext tasks can already be learned from completing downstream tasks by GNNs (Jin et al, 2020d). Besides, the strength of the performance improvement depends on the specific GNN architecture used for completing pretext and downstream tasks. The improvements are more significant for basic GNNs such as GCN, GAT, and GIN while less for more advanced GNNs such as GMNN (You et al, 2020c). Furthermore, one pretext task is not universally the best across multiple datasets (Gao et al, 2021; Manessi and Rozza, 2020). Therefore, whether a self-supervised pretext task helps GNNs in the standard target performance is determined by first whether the dataset allows the GNNs to extract extra feature information through completing pretext tasks, and second whether the extra self-supervised information complement, contradict to or has already been covered by information extracted from existing architecture (You et al, 2020c). Numerous works focus on applying contrastive learning as a form of self-supervised learning

(Chen et al, 2020b; Hassani and Khasahmadi, 2020; Veličković et al, 2019; You et al, 2020b; Zhu et al, 2021). Generally they find that while composing different augmentations benefits the performance (You et al, 2020b), increasing the number of views generated from the same graph augmentation technique to more than two cause no further improvement (Hassani and Khasahmadi, 2020), which is different from visual representation learning. Moreover, the beneficial combinations of augmentations are data-specific because of the highly heterogeneous nature of the graph-structured data and harder contrastive tasks are more helpful than overly simple ones (You et al, 2020b). Therefore, designing viable pretext tasks requires domain specific knowledge and should be targeted towards specific types of networks, GNN architectures and downstream tasks.

18.8 Summary

In this chapter, we provided a systemic, categorical and comprehensive overview on the recent works leveraging self-supervised learning in graph neural networks. Despite recent successes achieved by applying self-supervised learning in the text and image domains, self-supervised learning applied to the graph domain, especially for graph neural networks, is still in its emerging stage. Several promising directions could be pursued to further advance this field. First, although a large surge of research focuses on designing effective pretext tasks boosting the performance of graph neural networks, few works focus on visualizing, interpreting and explaining the underlying reason causing such beneficial performance improvements. Deeply understanding the intrinsic mechanism as to why and how SSL helps GNNs could help us design more powerful pretext tasks. Second, similar to the work defining the architectural design space for GNNs to quickly query the best GNN design for a novel task on a novel dataset (You et al, 2020a), we should collect and classify various pretext tasks and create a design space for SSL in GNNs. This allows for transferring the best designs of pretext tasks across different downstream tasks, GNN architectures and datasets. We hope that this chapter can shed some light on the main ideas of applying self-supervised learning to graph neural networks and related applications in order to encourage progress in the field.

Editor's Notes: Although methods introduced in the previous chapter (chapter 4, 5, 6, 15, and 16) have achieved state-of-the-art performances in corresponding tasks, they require large annotated datasets. Self-supervised learning seeks to create and utilize pretext labels on unlabeled data. Pretext tasks are relevant to traditional graph analysis tasks, such as node-level tasks (chapter 4) and graph level tasks (chapter 9), while pretext tasks use pseudo labels. The development of self-supervised GNN is of great significance to domains where labeled data are difficult to obtain, such as drug development (chapter 24). Besides, domains that have accumulated a large number of unlabeled data sets, such as computer vision (chapter 20) and natural language processing (chapter 21), also benefit from self-supervised learning.