Chapter 7
Interpretability in Graph Neural Networks

Ninghao Liu and Qizhang Feng and Xia Hu

Abstract Interpretable machine learning, or explainable artificial intelligence, is experiencing rapid developments to tackle the opacity issue of deep learning techniques. In graph analysis, motivated by the effectiveness of deep learning, graph neural networks (GNNs) are becoming increasingly popular in modeling graph data. Recently, an increasing number of approaches have been proposed to provide explanations for GNNs or to improve GNN interpretability. In this chapter, we offer a comprehensive survey to summarize these approaches. Specifically, in the first section, we review the fundamental concepts of interpretability in deep learning. In the second section, we introduce the post-hoc explanation methods for understanding GNN predictions. In the third section, we introduce the advances of developing more interpretable models for graph data. In the fourth section, we introduce the datasets and metrics for evaluating interpretation. Finally, we point out future directions of the topic.

7.1 Background: Interpretability in Deep Models

Deep learning has become an indispensable tool for a wide range of applications such as image processing, natural language processing, and speech recognition. Despite the success, deep models have been criticized as "black boxes" due to their complexity in processing information and making decisions. In this section, we introduce the research background of interpretability in deep models, including the

Ninghao Liu
Department of CSE, Texas A&M University, e-mail: nhliu43@tamu.edu

Qizhang Feng
Department of CSE, Texas A&M University, e-mail: qf31@tamu.edu

Xia Hu
Department of CSE, Texas A&M University, e-mail: xiahu@tamu.edu

definition of interpretability/interpretation, the reasons for exploring model interpretation, the methods of obtaining interpretation in traditional deep models, the opportunities and challenges to achieve interpretability in GNN models.

7.1.1 Definition of Interpretability and Interpretation

There is no unified mathematical definition of interpretability. A commonly used (nonmathematical) definition of interpretability is given as below (Miller, 2019).

Definition 7.1. *Interpretability* is the degree to which an observer can understand the cause of a decision.

There are three elements in the above definition: "understand", "cause", and "a decision". According to different scenarios, it is common that these elements are re-weighted or even some elements are replaced. First, in the context of machine learning systems where the role of humans needs to be emphasized, the definition of interpretability is usually revised to adapt to humans (Kim et al, 2016), where interpretation results that better facilitate human understanding and reasoning habits are more desirable. Second, from the term "cause" in the definition, it is natural to think that interpretation studies causality properties in models. While causality is important in developing certain types of interpretation methods, it is also common that interpretation is obtained beyond the framework of causal theories. Third, there is an increasing number of techniques that jump out of the scheme of explaining "a decision", and try to understand a broader range of entities such as model components (Olah et al, 2018) and data representations.

The interpretation is one mode in which an observer may obtain understanding of a model or its predictions. A general and widely followed definition is as below (Montavon et al, 2018).

Definition 7.2. An *interpretation* is the mapping of an abstract concept into a domain that the human can understand.

Typical examples of human-understandable domains include arrays of pixels in images or words in texts. There are two elements that merit attention in the above definition: "concept" and "understand". First, the "concept" to be explained could refer to different aspects, such as a predicted class (i.e., the logit value of the predicted class), the perception of a model component, or the meaning of a latent dimension. Second, in specific scenarios where user experience is important, it is necessary to transfer raw interpretation to the format that facilitates user comprehension, sometimes even with the cost of sacrificing interpretation accuracy.

It is also worth noting that, in this work, we distinguish between "interpretation" and "explanation". Although their differences have not been formally defined, in literatures, explanation mainly refers to the collection of important features for a given prediction (e.g., classification or regression) (Montavon et al, 2018). Meanwhile, "explanation" is more likely to be used if we are studying post-hoc interpretation

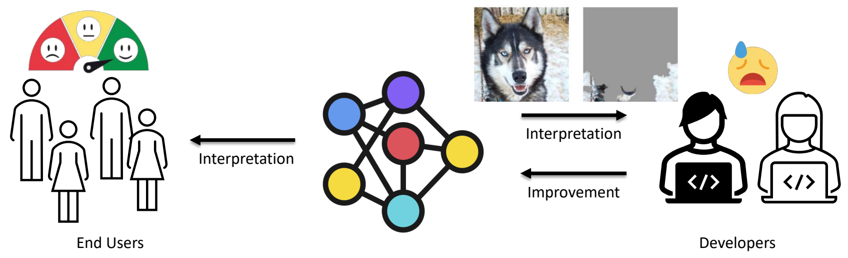

Fig. 7.1: Left: Interpretation could benefit user experiences in interaction with models. Right: Through interpretation, we could identify model behaviors that are not desirable according to humans, and work on improving the model accordingly (Ribeiro et al, 2016).

or human-understandable interpretation. "Interpretation" usually refers to a broader range of concepts, especially to emphasize that the model itself is intrinsically interpretable (i.e., the transparency of the model).

7.1.2 *The Value of Interpretation*

There are several pragmatic reasons that motivate people to study and improve model interpretability. Depending on who finally benefits from interpretation, we divide the reasons into model-oriented and user-oriented, as shown in Fig. 7.1.

7.1.2.1 Model-Oriented Reasons

Interpretation is an effective tool to diagnose the defects in models and provide directions on how to improve. Therefore, after several iterations of model updates, it is possible to obtain better models with particular properties coming about, and we could apply these models to our advantage. There are several properties that have been considered in literatures that are summarized as below.

1. *Credibility:* A model is regarded as credible if the rationale used behind predictions is consistent with the well-established domain knowledge. Through interpretation, we could observe whether the predictions are based on proper evidences, or they are simply from the exploitation of artifacts in data. By extracting explanations from a model and making the explanations to match human-annotated evidences in data, we are able to improve the model's credibility when making decisions (Du et al, 2019; Wang et al, 2018c).
2. *Fairness:* Machine learning systems have the risk of amplifying societal stereotypes if they rely on sensitive attributes, such as race, gender and age, in making predictions. Through interpretation, we could observe whether the predictions

are based on sensitive features that are required to be avoided in real applications.

3. *Adversarial-Attack Robustness:* Adversarial attack refers to adding carefully-crafted perturbations to input, where the perturbations are almost imperceptible to humans, but can cause the model to make wrong predictions (Goodfellow et al, 2015). Robustness against adversarial attacks is an increasingly important topic in machine learning security. Recent studies have shown how interpretation could help in discovering new attack schemes and designing defense strategies (Liu et al, 2020d).
4. *Backdoor-Attack Robustness:* Backdoor attack refers to injecting malicious functionality into a model, by either implanting additional modules or poisoning training data. The model will behave normally unless it is fed with input containing patterns that trigger the malicious functionality. Studying model robustness against backdoor attacks is attracting more interest recently. Recent research discovers that interpretation could be applied in identifying if a model has been infected by backdoors (Huang et al, 2019c; Tang et al, 2020a).

7.1.2.2 User-Oriented Reasons

The interpretation could contribute to the construction of interfaces between humans and machines.

1. *Improving User Experiences:* By providing intuitive visual information, interpretation could gain user trust, and increase a system's ease of use. For example, in healthcare-related applications, if the model could explain to patients how it makes diagnoses, the patients would be more convinced (Ahmad et al, 2018). For another example, in a recommender system, providing explanations can help users to make faster decisions and persuade users to purchase the recommended products (Li et al, 2020c).
2. *Facilitating Decision Making:* In many applications, a model plays the role as an assistant, while humans will make the final decision. In this case, interpretation helps shape human understandings towards instances, thus affecting subsequent decision-making processes. For example, in outlier detection, some outliers own malicious properties that should be handled with caution, while some are benign instances that simply happens to be "different". With interpretation, it is much easier for human decision-makers to understand whether a given outlier is malicious or benign.

7.1.3 Traditional Interpretation Methods

In general, there are two categories of techniques in improving model interpretability. Some efforts have been paid to building more transparent models, and we are able to grasp how the models (or parts of the models) work. We call this direction

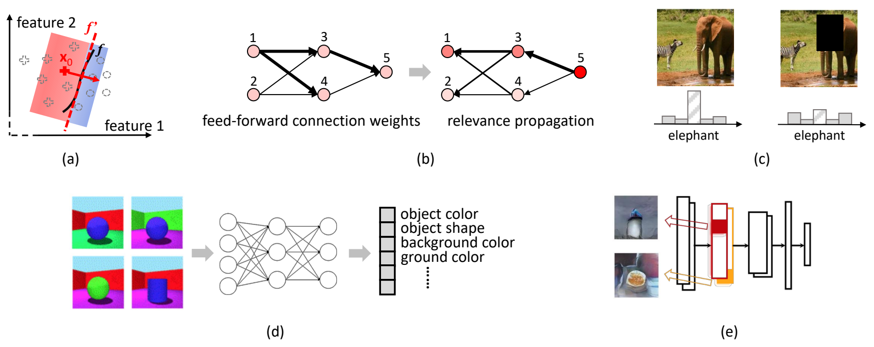

Fig. 7.2: Illustration of post-hoc interpretation methods. (a): Local approximation based interpretation. (b): Layer-wise relevance propagation. (c): Explanation based on perturbation. (d): Explaining the meaning of latent representation dimensions. (e): Explaining the meaning of neurons in a convolutional neural network via input generation.

as *interpretable modeling*. Meanwhile, instead of elucidating the internal mechanisms by which models work, some methods investigate *post-hoc interpretation* to provide explanations to models that are already built. In this part, we introduce the techniques of the two categories. Some of the methods provide motivation for GNN interpretation which will be introduced in later sections.

7.1.3.1 Post-Hoc Interpretation

The post-hoc interpretation has received a lot of interests in both research and real applications. Flexibility is one of the advantages of post-hoc interpretation, as it put less requirement on the model types or structures. In the following paragraphs, we briefly introduce several commonly used methods. The illustration of the basic idea behind each of these methods is shown in Fig. 7.2.

The first type of methods to be introduced is *approximation-based methods*. Given a function f that is complex to understand and an input instance $\mathbf{x}^* \in \mathbb{R}^m$, we could approximate f with a simple and understandable surrogate function h (usually chosen as a linear function) locally around $\mathbf{x}^*$. Here m is the number of features in each instance. There are several ways to build h. A straightforward way is based on the first-order Taylor expansion, where:

$$f(\mathbf{x}) \approx h(\mathbf{x}) = f(\mathbf{x}^*) + \mathbf{w}^\top \cdot (\mathbf{x} - \mathbf{x}^*), \tag{7.1}$$

where $\mathbf{w} \in \mathbb{R}^m$ tells how sensitive the output is to the input features. Typically, $\mathbf{w}$ can be estimated with the gradient (Simonyan et al, 2013), so that $\mathbf{w} = \nabla_{\mathbf{x}} f(\mathbf{x}^*)$. When gradient information is not available, such as in tree-based models, we could build h through training (Ribeiro et al, 2016). The general idea is that a number of training instances $(\mathbf{x}^i, f(\mathbf{x}^i))$, $1 \leq i \leq n$ are sampled around $\mathbf{x}^*$, i.e., $\|\mathbf{x}^i - \mathbf{x}^*\| \leq \varepsilon$. The instances are then used to train h, so that h approximates f around $\mathbf{x}^*$.

Besides directly studying the sensitivity between input and output, there is another type of method called *layer-wise relevance propagation (LRP)* (Bach et al, 2015). Specifically, LRP redistributes the activation score of output neuron to its predecessor neurons, which iterates until reaching the input neurons. The redistribution of scores is based on the connection weights between neurons in adjacent layers. The share received by each input neuron is used as its contribution to the output.

Another way to understand the importance of a feature $\mathbf{x}_i$ is to answer questions like "What would have happened to f, had $\mathbf{x}_i$ not existed in input?". If $\mathbf{x}_i$ is important for predicting $f(\mathbf{x})$, then removing/weakening it will cause a significant drop in prediction confidence. This type of method is called the *perturbation method* (Fong and Vedaldi, 2017). One of the key challenges in designing perturbation methods is how to guarantee the input after perturbation is still valid. For example, it is argued that perturbation on word embedding vectors cannot explain deep language models, because texts are discrete symbols, and it is hard to identify the meaning of perturbed embeddings.

Different from the previous methods that focus on explaining prediction results, there is another type of method that tries to understand how data is represented inside a model. We call it *representation interpretation*. There is no unified definition for representation interpretation. The design of methods under this category is usually motivated by the nature of the problem or the properties of data. For example, in natural language processing, it has been shown that a word embedding could be understood as the composition of a number of basis word embeddings, where the basis words constitute a dictionary (Mathew et al, 2020).

Besides understanding predictions and data representations, another interpretation scheme is to understand the role of *model components*. A well-known example is to visualize the visual patterns that maximally activate the target neuron/layer in a CNN model (Olah et al, 2018). In this way, we understand what kind of visual signal is detected by the target component. The interpretation is usually obtained through a *generative process*, so that the result is understandable to humans.

7.1.3.2 Interpretable Modeling

Interpretable modeling is achieved via incorporating interpretability directly into the model structures or learning process. It is still an extremely challenging problem to develop models that are both transparent and could achieve state-of-the-art performances. Many efforts have been paid to improve the intrinsic interpretability of deep models. Some details are discussed as below.

A straightforward strategy is to rely on *distillation*. Specifically, we first build a complex model (e.g., a deep model) to achieve good performance. Then, we use another model, which is readily recognized as interpretable, to mimic the predictions of the complex model. The pool of interpretable models includes linear models, decision trees, rule-based models, etc. This strategy is also called *mimic learning*. The interpretable model trained in this way tends to perform better than normal training, and it is also much easier to understand than the complex model.

Attention models, originally introduced for machine translation tasks, have now become enormously popular, partially due to their interpretation properties. The intuition behind attention models can be explained using human biological systems, where we tend to selectively focus on some parts of the input, while ignoring other irrelevant parts (Xu et al, 2015). By examining attention scores, we could know which features in the input have been used for making the prediction. This is also similar to using post-hoc interpretation algorithms that find which input features are important. The major difference is that attention scores are generated during model prediction, while post-hoc interpretation is performed after prediction.

Deep models heavily rely on learning effective representations to compress information for downstream tasks. However, it is hard for humans to understand the representations as the meanings of different dimensions are unknown. To tackle this challenge, *disentangled representation learning* has been proposed. Disentangled representation learning breaks down features of different meanings and encodes them as separate dimensions in representations. As a result, we could check each dimension to understand which factors of input data are encoded. For example, after learning disentangled representations on 3D-chair images, factors such as chair leg style, width and azimuth, are separately encoded into different dimensions (Higgins et al, 2017).

7.1.4 Opportunities and Challenges

Despite the major progress made in domains such as vision, language and control, many defining characteristics of human intelligence remain out of reach for traditional deep models such as convolutional neural networks (CNNs), recurrent neural networks (RNNs) and multi-layer perceptrons (MLPs). To look for new model architectures, people believe that GNN architectures could lay the foundation for more interpretable patterns of reasoning (Battaglia et al, 2018). In this part, we discuss the advantages of GNNs and challenges to be tackled in terms of interpretability.

The GNN architecture is regarded as more interpretable because it facilitates learning about entities, relations, and rules for composing them. First, entities are discrete and usually represent high-level concepts or knowledge items, so it is regarded as easier for humans to understand than image pixels (tiny granularity) or word embeddings (latent space vectors). Second, GNN inference propagates information through links, so it is easier to find the explicit reasoning path or subgraph that contribute to the prediction result. Therefore, there is a recent trend of transforming image or text data into graphs, and then applying GNN models for predictions. For example, to build a graph from an image, we can treat objects inside the image (or different portions within an object) as nodes, and generate links based on the spatial relations between nodes. Similarly, a document can be transformed into a graph by discovering concepts (e.g., nouns, named entities) as nodes and extracting their relations as links through lexical parsing.

Although the graph data format lays a foundation for interpretable modeling, there are still several challenges that undermine GNN interpretability. First, GNN still maps nodes and links into embeddings. Therefore, similar to traditional deep models, GNN also suffers from the opacity of information processing in intermediate layers. Second, different information propagation paths or subgraphs contribute differently to the final prediction. GNN does not directly provide the most important reasoning paths for its prediction, so post-hoc interpretation methods are still needed. In the following sections, we will introduce the recent advances in tackling the above challenges to improve the explainability and interpretability of GNNs.

7.2 Explanation Methods for Graph Neural Networks

In this section, we introduce the post-hoc explanation methods for understanding GNN predictions. Similar to the categorization in Section 7.1.3, we include approximation-based methods, relevance-propagation based methods, perturbation-based methods, and generative methods.

7.2.1 Background

Before introducing the techniques, we first provide the definition of graphs and review the fundamental formulations of a GNN model.

Graphs: In the rest of the chapter, if not specified, the graphs we discuss are limited to homogeneous graphs.

Definition 7.3. A *homogeneous graph* is defined as $\mathscr{G} = (\mathscr{V}, \mathscr{E})$, where $\mathscr{V}$ is the set of nodes and $\mathscr{E}$ is the set of edges between nodes.

Furthermore, let $A \in \mathbb{R}^{n \times n}$ be the adjacency matrix of $\mathscr{G}$, where $n = |\mathscr{V}|$. For unweighted graphs, $A_{i,j}$ is binary, where $A_{i,j} = 1$ means there exists an edge $(i, j) \in \mathscr{E}$, otherwise $A_{i,j} = 0$. For weighted graphs, each edge (i, j) is assigned a weight $w_{i,j}$, so $A_{i,j} = w_{i,j}$. In some cases, nodes are associated with features, which could be denoted as $X \in \mathbb{R}^{n \times m}$, and $X_{i,:}$ is the feature vector of node i. The number of features for each node is m. In this chapter, unless otherwise stated, we focus on GNN models on homogeneous graphs.

GNN Fundamentals: Traditional GNNs propagate information via the input graph's structure according to the propagation scheme:

$$H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l}W^{l}), \tag{7.2}$$

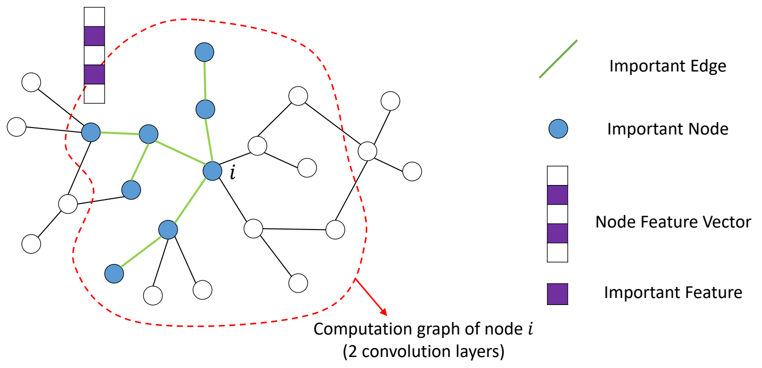

Fig. 7.3: Illustration of explanation result formats. Explanation results for graph neural networks could be the important nodes, the important edges, the important features, etc. An explanation method may return multiple types of results.

where H^l denotes the embedding matrix at layer l, and W^l denotes the trainable parameters at layer l. Also, $\tilde{A} = A + \mathbf{I}$ denotes the adjacency matrix of the graph after adding the self loop. The matrix $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$, i.e., $\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$. Therefore, $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ normalizes the adjacency matrix. If we only focus on the embedding update of node i, the GCN propagation scheme could be rewritten as:

$$H_{i,:}^{l+1} = \sigma\left(\sum_{j \in \mathscr{V}_i \cup \{i\}} \frac{1}{c_{i,j}} H_{j,:}^{l} W^{l}\right), \tag{7.3}$$

where $H_{j,:}$ denotes the j-th row of matrix H, and $\mathscr{V}_i$ denotes the neighbors of node i. Here $c_{i,j}$ is a normalization constant, and $\frac{1}{c_{i,j}} = (\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})_{i,j}$. Therefore, the embedding of node i at layer l can be seen as aggregating neighbor embeddings of nodes that are neighbors of node i, followed by some transformations. The embeddings in the first layer H^0 is usually set as the node features. As the layer goes deeper, the computation of each node's embedding will include further nodes. For example, in a 2-layer GNN, computing the embedding of node i will use the information of nodes within the 2-hop neighborhood of node i. The subgraph composed by these nodes are called the *computation graph* of node i, as shown in Fig. 7.3.

Target Models: There are two common tasks in graph analysis, i.e., graph-level predictions and node-level predictions. We use classification tasks as the example. In graph-level tasks, the model $f(\mathscr{G}) \in \mathbb{R}^C$ produces a single prediction for the whole graph, where C is the number of classes. The prediction score for class c could be written as $f^c(\mathscr{G})$. In node-level tasks, the model $f(\mathscr{G}) \in \mathbb{R}^{n \times C}$ returns a matrix, where each row is the prediction for a node. Some explanation methods are designed solely for graph-level tasks, some are for node-level tasks, while some could handle both scenarios. The computation graphs introduced above are commonly used in explaining node-level predictions.

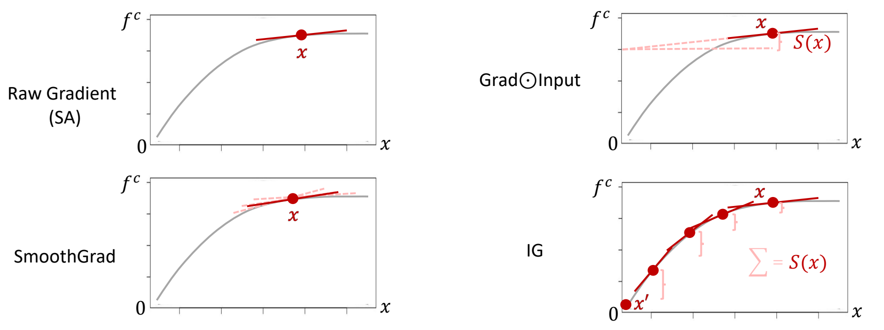

Fig. 7.4: Illustration of several gradient-based explanation methods. Methods relying on local gradients may suffer from the saturation problem or noises in input, where a feature's local sensitivity is not consistent with its overall contribution. SmoothGrad removes noises in an explanation by averaging multiple explanations on nearby points. IG is more accurate than Grad ⊙ Input in measuring feature contribution.

Interpretation Formats: According to the introduction above, there are several input modes that could be included in the explanation as shown in Fig. 7.3. Specifically, explanation methods could identify what are the *important nodes*, *important edges* and *important features* that contribute most to the prediction. Some explanation methods may identify multiple types of input modes simultaneously.

7.2.2 *Approximation-Based Explanation*

The approximation-based explanation has been widely used to analyze the prediction of models with complex structures. Approximation-based approaches could be further divided into white-box approximation and black-box approximation. The white-box approximation uses information inside the model, which includes but is not limited to gradients, intermediate features, model parameters, etc. The black-box approximation does not utilize information propagation inside the model. It usually uses a simple and interpretable model to fit the target model's decision on an input instance. Then, the explanation can be easily extracted from the simple model. The details of commonly used methods for both categories are introduced as below.

7.2.2.1 White-Box Approximation Method

Sensitivity Analysis (SA) Baldassarre and Azizpour (2019) studies the impact of a particular change in an independent variable on a dependent variable. In the context of explanation, the dependent variable refers to the prediction, while the independent variables refer to the features. The local gradient of the model is commonly used as sensitivity scores to represent the correlation between the feature and the prediction result. The sensitivity score is defined as:

$$\mathscr{S}(\mathbf{x}) = \|\nabla_{\mathbf{x}} f(\mathscr{G})\|^2, \tag{7.4}$$

where $\mathscr{G}$ is the input instance graph to be explained, $f(\mathscr{G})$ is the model prediction function. Here $\mathbf{x}$ refers to the feature vector of a node of interest. Node features with higher sensitivity scores are more important because they can lead to drastic changes to model decisions.

Although SA is intuitive and straightforward, its effectiveness is still limited. It assumes input features are mutually independent, and does not necessarily pay attention to their correlations in the actual decision-making process. Also, sensitivity analysis only measures the impact of local changes to the decision function $f(\mathscr{G})$, rather than thoroughly explaining the decision function value itself. Explanation results provided by sensitivity analysis are usually relatively noisy and challenging to comprehend. Therefore, some follow-up techniques have been developed trying to overcome this limitation (as shown in Fig. 7.4).

GuidedBP(Baldassarre and Azizpour, 2019) is similar to SA except that it only detects the features that positively activate the neurons, with the assumption that negative gradients may confuse the contribution of important features and makes the visualizing noisy. To follow this intuition, GuideBP modifies the process of back-propagation of SA and discards all negative gradients.

Grad $\odot$ Input Sanchez-Lengeling et al (2020) measures the feature contribution scores as the element-wise product of the input features and the gradients of decision function with respect to the features:

$$\mathscr{S}(\mathbf{x}) = \nabla_{\mathbf{x}}^{\top} f(\mathscr{G}) \odot \mathbf{x}. \tag{7.5}$$

Therefore, Grad $\odot$ Input considers not only the feature sensitivity, but also the scale of feature values. However, the methods mentioned above all suffered from the *saturation problem*, where the scope of the local gradients is too limited to reflect the overall contribution of each feature.

Integrated Gradients (IG) Sanchez-Lengeling et al (2020) solves the saturation problem by aggregating feature contribution along a designed path in input space. This path starts from a chosen baseline point $\mathscr{G}'$ and ends at the target input $\mathscr{G}$. Specifically, the feature contribution is computed as:

$$\mathscr{S}(\mathbf{x}) = \left(\mathbf{x} - \mathbf{x}'\right) \int_{\alpha=0}^{1} \nabla_{\mathbf{x}} f\left(\mathscr{G}' + \alpha\left(\mathscr{G} - \mathscr{G}'\right)\right) d\alpha \tag{7.6}$$

where $\mathbf{x}'$ denotes a feature vector in the baseline point $\mathscr{G}'$, while $\mathbf{x}$ is a feature vector in the original input $\mathscr{G}$. The choice of baseline $\mathscr{G}'$ is relatively flexible. A typical strategy is to use a null graph as the baseline, which has the same topology but its nodes use "unspecified" categorical features. This is motivated by the application of

IG in explaining image classification models (Sundararajan et al, 2017), where the baseline is usually chosen as a pure black image or an image with random noises.

The explanations obtained by the above methods usually contain a lot of noises. Therefore, Smilkov et al (2017) proposes SmoothGrad to alleviate the problem. **SmoothGrad** averages attributions evaluated on a number of noise-perturbed versions of the input. This method initially aims at sharpening the saliency maps on images. Furthermore, Sanchez-Lengeling et al (2020) applies it to the Grad $\odot$ Input method by adding Gaussian noise to node and edge features, and averaging multiple explanations to a smoothed one.

Class Activation Mapping (CAM) (Pope et al, 2019) is an explanation method that is initially developed for CNNs. This method only works under a specific model architecture, where the last convolutional layer is followed by a global average pooling (GAP) layer before the final softmax layer. The feature maps (i.e., activations) in the last convolutional layer are aggregated and re-scaled to the same size as the input image, so that the activations highlight the important regions in the image. The idea of CAM can also be adapted to graph neural networks. Specifically, the GAP layer in a GNN could be defined as averaging the embeddings of all nodes in the last graph convolution layer: $\mathbf{h} = \frac{1}{n}\sum_{i=1}^{n} H^L_{i,:}$, where L is the last graph convolution layer. CAM treats each dimension of the final node embeddings (i.e., $H^L_{:,k}$) as a feature map. The logit value for class c is:

$$f^c(\mathscr{G}) = \sum_k w^c_k \mathbf{h}_k \tag{7.7}$$

where $\mathbf{h}_k$ denotes the k-th entry of $\mathbf{h}$, w^c_k is the GAP-layer weight of k-th feature map with respect to class c. Therefore, the contribution of node i to the prediction is:

$$\mathscr{S}(i) = \frac{1}{n}\sum_k w^c_k H^L_{i,k}. \tag{7.8}$$

Although CAM is simple and efficient, it only works on models with certain structures, which greatly limits its application scenarios.

Grad-CAM (Pope et al, 2019) combines gradient information with feature maps to relax the limitation of CAM. While CAM uses the GAP layer to estimate the weight of each feature map, Grad-CAM employs the gradient of output with respect to the feature maps to compute the weights, so that:

$$w^c_k = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial f^c(\mathscr{G})}{\partial H^L_{i,k}}, \tag{7.9}$$

$$\mathscr{S}(i) = ReLU\left(\sum_k w^c_k H^L_{i,k}\right). \tag{7.10}$$

The *ReLU* function forces the explanation to focus on the positive influence on the class of interest. Grad-CAM is equivalent to CAM for GNNs with only one fully-connected layer before output. Compared to CAM, Grad-CAM can be applied to more GNN architectures, thus avoiding the trade-off between model explainability and capacity.

7.2.2.2 Black-Box Approximation Methods

Different from white-box approximation methods, black-box approximation methods manage to bypass the need to obtain internal information of complex models. The general idea is to use models that are intrinsically interpretable (such as linear regressions, decision trees) to fit the complex model. Then, we can explain the decision based on the simple models. The fundamental assumption behind is that: Given an input instance, the model's decision boundary within the neighborhood of that instance can be well approximated by the interpretable model. The major challenge is how to define the neighborhood space given an input graph which is a discrete data structure.

We introduce several approaches, including GraphLime (Huang et al, 2020c), RelEx (Zhang et al, 2020a), and PGM-Explainer (Vu and Thai, 2020). These methods share a similar procedure: First, a neighborhood space is defined around the target instance. Second, data points are sampled within this space and their predictions are obtained after being fed into the target model. A training dataset is built, where each instance-label pair consists of a sampled point and its prediction. Finally, an interpretable model is trained by using the dataset. The key difference between these methods lies in two aspects, i.e., the definition of the neighborhood, and the choice of the interpretable model.

GraphLime is a local explanation method for GNN predictions on graph nodes. Given the prediction result on a target node v_t, GraphLime defines the neighborhood space as a set of nodes which are in the k-hop neighborhood of the target node in the input graph:

$$\mathcal{V}_t = \{v \mid distance(v_t, v) \leq k, v \in \mathcal{V}\}, \tag{7.11}$$

where the k-hop neighborhood refers to the nodes which are within k hops from the target node. GraphLime collects the features of nodes in $\mathcal{V}_t$ as the corpus, and employs HSIC Lasso (Hilbert-Schmidt independence criterion Lasso) to measure the independence between features and predictions of the nodes. The top important features are selected as the explanation result, so GraphLime cannot provide explanations based on structural information of the graph.

RelEx defines the neighborhood space as a set of perturbed graphs to the computation graph of the target node. Similar to GraphLime, RelEx explains GNN predictions on nodes. The computation graph $\mathcal{G}_t$ of the target node v_t is composed of the k-hop neighbor nodes around node v_t and the edges that connect them. First, RelEx proposes a BFS sampling strategy to sample multiple perturbed graphs $\{\mathcal{G}'_{t,1}, \mathcal{G}'_{t,2}, ..., \mathcal{G}'_{t,I}\}$ from the computation graph. These perturbed graphs are fed into the original GNN f to build a training set $\{\mathcal{G}'_{t,i}, f(\mathcal{G}'_{t,i})\}_{i=1}^{I}$. Then, a new GNN f' is trained upon the training set to approximate f. After that, a mask M is trained for explanation. The mask is applied to the adjacency matrix of $\mathcal{G}_t$. The value of each mask entry is in $[0,1]$, so it is a soft mask. There are two loss terms for training the mask: (1) $f'(\mathscr{G}_t \odot M)$ is close to $f'(\mathscr{G}_t)$, (2) the mask M is sparse. The resultant mask entry values indicate the importance score of edges in $\mathscr{G}_t$, where a higher mask value means the corresponding edge is more important.

PGM-Explainer applies probabilistic graphical models to explain GNNs. To find the neighbor instances of the target, PGM-Explainer first randomly selects nodes to be perturbed from computation graphs. Then, the selected nodes' features are set to the mean value among all nodes. After that, PGM-Explainer employs a pair-wise dependence test to filter out unimportant samples, aiming at reducing the computational complexity. Finally, a Bayesian network is introduced to fit the predictions of chosen samples. Therefore, the advantage of PGM-Explainer is that it illustrates the dependency between features.

7.2.3 Relevance-Propagation Based Explanation

Relevance propagation redistributes the activation score of output neuron to its predecessor neurons, iterating until reaching the input neurons. The core of relevance propagation methods is about defining a rule for the activation redistribution between neurons. Relevance propagation has been widely used to explain models in domains such as computer vision and natural language processing. Recently, some work has been proposed to explore the possibility of revising relevance propagation method for GNNs. Some representative approaches include LRP (Layer-wise Relevance Propagation) (Baldassarre and Azizpour, 2019; Schwarzenberg et al, 2019), GNN-LRP (Schnake et al, 2020), ExcitationBP (Pope et al, 2019).

LRP is first proposed in (Bach et al, 2015) to calculate the contribution of individual pixels to the prediction result for an image classifier. The core ides of LRP is to use back propagation to recursively propagate the relevance scores of high-level neurons to low-level neurons, up to the input-level feature neurons. The relevance score of the output neuron is set as the prediction score. The relevance score that a neuron receives is proportional to its activation value, which follows the intuition that neurons with higher activation tend to contribute more to the prediction. In (Baldassarre and Azizpour, 2019; Schwarzenberg et al, 2019), the propagation rule is defined as below:

$$\begin{aligned} R_i^l &= \sum_j \frac{z_{i,j}^+}{\sum_k z_{k,j}^+ + b_j^+ + \varepsilon} R_j^{l+1} \\ z_{i,j} &= x_i^l w_{i,j} \end{aligned} \tag{7.12}$$

where R_i^l, R_j^{l+1} is the relevance score of neuron i in layer l and neuron j in layer $l+1$, respectively. x_i^l is the activation of neuron i in layer l. $w_{i,j}$ is the connection weight. ε prevents the denominator from being zero. This propagation rule only allows positive activation values. Also, explanations obtained using this method are limited to nodes and node features, where graph edges are excluded. The reason is that the adjacency matrix is treated as part of the GNN model. Therefore, LRP is unable to analyze topological information which nevertheless plays an important role in graph data.

ExcitationBP is a top-down attention model originally developed for CNNs (Zhang et al, 2018d). It shares a similar idea as LRP. However, ExcitationBP defines the relevance score as a probability distribution and uses a conditional probability model to describe the relevance propagation rule.

$$P(a_j) = \sum_i P(a_j \mid a_i) P(a_i) \tag{7.13}$$

where a_j is the j-th neuron in the lower layer and a_i is the i-th parent neuron of a_j in the higher layer. When the propagation process passes through the activation function, only non-negative weights are considered and negative weights are set to zero. To extend ExcitationBP for graph data, new backward propagation schemes are designed for the softmax classifier, the GAP (global average pooling) layer and the graph convolutional operator.

GNN-LRP mitigates the weakness of traditional LRP by defining a new propagation rule. Instead of using the adjacency matrix to obtain propagation paths, GNN-LRP assigns the relevance score to a walk, which refers to a message flow path in the graph. The relevance score is defined by the T-order Taylor expansion of the model with respect to the incorporation operator (graph convolutional operator, linear message function, etc.). The intuition is that the incorporation operator with greater gradients has a greater influence on the final decision.

7.2.4 Perturbation-Based Approaches

An assumption behind prediction explanations is that important input parts significantly contribute to the output while unimportant parts have minor influences. It thus implies that masking out the unimportant parts will have a negligible impact on the output, and masking out the important parts will have a significant impact. The goal is to find a mask M to indicate graph component importance. The mask could be applied to nodes, edges or features in graphs. The mask value can either be binary $M_i \in \{0,1\}$ or continuous $M_i \in [0,1]$. Some recent perturbation-based approaches are introduced as below.

GNNExplainer (Ying et al, 2019) is the first perturbation-based explanation method for GNNs. Given the model's prediction on a node v, GNNExplainer tries to find a compact subgraph $\mathcal{G}_S$ from the computation graph of node v that is most crucial for the prediction. The problem is defined as maximizing the mutual information (MI) between the predictions of the original computation graph and the predictions of the subgraph:

$$\max_{\mathscr{G}_S} MI\left(Y, (\mathscr{G}_S, X_S)\right) = H(Y) - H\left(Y \mid \mathscr{G} = \mathscr{G}_S, X = X_S\right), \tag{7.14}$$

where $\mathscr{G}_S$ and X_S is the subgraph and its nodes' features. Y is the predicted label distribution, and its entropy $H(Y)$ is a constant. To solve the optimization problem above, the authors apply a soft-mask M on adjacency matrix:

$$\min_{M} -\sum_{c=1}^{C} \mathbf{1}[y = c] \log P_{\Phi}\left(Y = y \mid G = A_c \odot \sigma(M), X = X_c\right), \tag{7.15}$$

where A_c is the adjacency matrix of the computation graph, X_c is the corresponding feature matrix, and M denotes the trainable parameters. The sigmoid function projects the mask value in [0,1]. Finally, an subgragh is built by selecting the edges (and the nodes connected by these edges) corresponding to the high values in M. Besides providing explanations based on graph structures, GNNExplainer could also offer feature-wise explanations by applying a similar masking learning process on features. Moreover, regularization techniques could be applied to enforce the explanation to be sparse. As a model-agnostic approach, GNNExplainer is suitable for any graph-based machine learning tasks and GNN models.

PGExplainer (Luo et al, 2020) shares the same idea with GNNExplainer and learns a discrete mask applied on edges to explain the predictions. The main idea is to use a deep neural network to generate edge mask values:

$$M_{i,j} = \mathrm{MLP}_{\Psi}\left([\mathbf{z}_i; \mathbf{z}_j]\right), \tag{7.16}$$

where Ψ denotes the trainable parameters of the MLP. $\mathbf{z}^i$ and $\mathbf{z}^j$ are the embedding vector for node i and j, respectively. $[\cdot;\cdot]$ denotes concatenation. Similar to the GNNExplainer, the mask generator is trained by maximizing the mutual information between the original prediction and the new prediction.

GraphMask (Schlichtkrull et al, 2021) also produces the explanation by estimating the influences of edges. Similar to PGExplainer, GraphMask learns an erasure function that quantifies the importance of each edge. The erasure function is defined as:

$$z_{u,v}^{(k)} = g_{\pi}\left(\mathbf{h}_u^{(k)}, \mathbf{h}_v^{(k)}, \mathbf{m}_{u,v}^{(k)}\right) \tag{7.17}$$

where $\mathbf{h}_u$, $\mathbf{h}_v$ and $\mathbf{m}_{u,v}$ refers to the hidden embedding vectors for node u, node v and the message sent through the edge in graph convolution. π denotes the parameters of function g. One difference between GraphMask and PGExplainer is that the former also takes the edge embedding as input. Another difference is that GraphMask provides the importance estimation for every graph convolution layer, and k indicates the layer that the embedding vectors belong to. Instead of directly erasing the influences of unimportant edges, the authors then propose to replace the message sent through unimportant edges as:

$$\tilde{\mathbf{m}}_{u,v}^{(k)} = z_{u,v}^{(k)} \cdot \mathbf{m}_{u,v}^{(k)} + \left(1 - z_{u,v}^{(k)}\right) \cdot \mathbf{b}^{(k)}, \tag{7.18}$$

where $\mathbf{b}^{(k)}$ is trainable. The work shows that a large proportion of edges can be dropped without deteriorating the model performance.

Causal Screening (Wang et al, 2021) is a model-agnostic post-hoc method that identifies a subgraph of input as an explanation from the cause-effect standpoint. Causal Screening exerts causal effect of candidate subgraph as the metric:

$$S(\mathscr{G}_k) = MI(\mathrm{do}(\mathscr{G} = \mathscr{G}_k); \hat{y}) - MI(\mathrm{do}(\mathscr{G} = \emptyset); \hat{y}) \tag{7.19}$$

where $\mathscr{G}_k$ is the candidate subgraph, k is the number of edges and MI is the mutual information. The intervention $\mathrm{do}(\mathscr{G} = \mathscr{G}_k)$ and $\mathrm{do}(\mathscr{G} = \emptyset)$ means the model input receives treatment (feeding $\mathscr{G}_k$ into the model) and control (feeding $\emptyset$ into the model), respectively. $\hat{y}$ denotes the prediction when feeding the original graph into the model. Causal Screening uses a greedy algorithm to search the explanation. Starting from an empty set, at each step, it adds one edge with the highest causal effect into the candidate subgraph.

CF-GNNExplainer (Lucic et al, 2021) also proposes to generate counterfactual explanations for GNNs. Different from previous methods that try to find a sparse subgraph to preserve the correct prediction, CF-GNNExplainer proposes to find the minimal number edges to be removed such that the prediction changes. Similar to GNNExplainer, CF-GNNExplainer employs the soft mask as well. Therefore, it also suffers from the "introduced evidence" problem (Dabkowski and Gal, 2017), which means that non-zero or non-one values may introduce unnecessary information or noises, and thus influence the explanation result.

7.2.5 *Generative Explanation*

Many methods introduced in previous subsections define the explanation as selecting sub-graphs that contains part of nodes, edges or features of the original input. Recently, XGNN (Yuan et al, 2020b) proposes to obtain explanation by *generating* a graph that maximizes the prediction of the given GNN model. Some methods that share a similar idea have been proposed for computer vision tasks. For example, the role of a neuron could be understood by finding the input prototypes that maximally activates the neuron's activation (Olah et al, 2018). The problem of finding prototype samples can be defined as an optimization problem, which can be solved by gradient ascent. However, this method can not be directly used on GNNs because the gradient ascent method is not compatible with the discrete and topological nature of graph data. To tackle this problem, XGNN defines graph generation as a reinforcement learning task.

To be more specific, the generator follows the steps below. First, it randomly picks one node as the initial graph. Second, given an intermediate graph, the generator adds a new edge to the graph. This action is carried out in two steps: choosing the edge's starting point as well as the end point. XGNN employs another GNN as the policy to determine the action. The GNN learns nodes features, and two MLPs then take the learned features as input to predict the possibility of a start point and an end point. The end point and the edge between the two points are added to update the intermediate graph as an action. Finally, it calculates the reward of the action, so that we can train the generator via policy gradient algorithms. The reward consists of two terms. The first term is the score of the intermediate graph after feeding it to the target GNN model. The second one is a regularization term that guarantees the validity of the intermediate graph. The above steps are executed repeatedly until the number of action steps reaches the predefined upper limit. As a generative explanation method, XGNN provides a holistic explanation for graph classification. There could be more generative explanation methods for other graph analysis tasks to be explored in the future.

7.3 Interpretable Modeling on Graph Neural Networks

Following the introduction in Section 7.1.3.2, we introduce two categories of interpretable modeling approaches, i.e., GNN models with attention mechanism and disentangled representation learning on graphs.

7.3.1 GNN-Based Attention Models

Attention mechanisms benefit model interpretability by highlighting relevant parts of the graph for the given task through attention scores. According to the graph types, we introduce attention models built upon homogeneous graphs and heterogeneous graphs, respectively.

7.3.1.1 Attention Models for Homogeneous Graphs

Graph Attention Networks (GATs) enable assigning different weights to different node embeddings in a neighborhood when aggregating information (Veličković et al, 2018). Specifically, let $\mathbf{h}^i$ denote the column-wise embedding of node i, then the embedding update is written as:

$$\mathbf{h}^i_{l+1} = \sigma\Big(\sum_{j \in \mathcal{V}_i \cup \{i\}} \alpha_{i,j} W \mathbf{h}^j_l\Big), \tag{7.20}$$

where $\alpha_{i,j}$ is the attention score, and $\mathcal{V}_i$ denotes the set of neighbors of node i. Also, GAT uses a shared parameter matrix W independent of the layer depth. The attention score is computed as:

$$\alpha_{i,j} = \mathrm{softmax}(e_{i,j}) = \frac{\exp(e_{i,j})}{\sum_{k \in \mathcal{V}_i \cup \{i\}} \exp(e_{i,k})}, \tag{7.21}$$

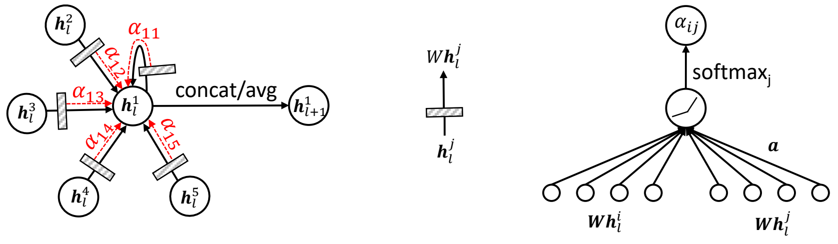

Fig. 7.5: Left: An illustration of graph convolution with single head attentions by node 1 on its neighborhood. Middle: The linear transformation with a shared parameter matrix. Right: The attention mechanism employed in (Veličković et al, 2018).

where self-attention mechanism is applied,

$$e_{i,j} = \text{LeakyReLU}(\mathbf{a}^\top[W\mathbf{h}_l^i \| W\mathbf{h}_l^j]), \tag{7.22}$$

where $\|$ denotes vector concatenation. In general, the attention mechanism can also be denoted as $e_{i,j} = attn(\mathbf{h}_l^i, \mathbf{h}_l^j)$. Therefore, the attention mechanism is a single-layer neural network parameterized by a weight vector $\mathbf{a}$. The attention score $\alpha_{i,j}$ shows the importance of node j to node i.

The above mechanism could also be extended with multi-head attention. Specifically, K independent attention mechanisms are executed in parallel, and the results are concatenated:

$$\mathbf{h}_{l+1}^i = \|_{k=1}^K \sigma(\sum_{j\in\mathscr{V}_i\cup\{i\}} \alpha_{i,j}^k W^k \mathbf{h}_l^j), \tag{7.23}$$

where $\alpha_{i,j}^k$ is the normalized attention score in the k-th attention mechanism, and W^k is the corresponding parameter matrix.

Besides learning node embeddings, we could also apply attention mechanisms to learn a low-dimensional embedding for the whole graph (Ling et al, 2021). Suppose we are working on an information retrieval problem. Given a set of graphs $\{\mathscr{G}_m\}$, $1 \le m \le M$, and a query q, we want to return the graphs that are most relevant to the query. The embedding of each graph $\mathscr{G}_m$ with respect to q could be computed using the attention mechanism. In the first step, we could apply normal GNN propagation rules as introduced in Equation 7.2, to obtain the embeddings of nodes inside each graph. Let $\mathbf{q}$ denote the embedding of the query, and $\mathbf{h}^{i,m}$ denote the embedding of node i in a graph $\mathscr{G}_m$. The embedding of graph $\mathscr{G}_m$ with respect to the query can be computed as:

$$\mathbf{h}_{\mathscr{G}_m}^q = \frac{1}{|\mathscr{G}_m|}\sum_{i=1}^{|\mathscr{G}_m|} \alpha_{i,q}\mathbf{h}^{i,m} \tag{7.24}$$

where $\alpha_{i,q} = \text{attn}(\mathbf{h}^{i,m}, \mathbf{q})$ is the attention score, and attn() is a certain attention function. Finally, $\mathbf{h}_{\mathscr{G}_m}^q$ can be used to compute the similarity of $\mathscr{G}_m$ to the query in the graph retrieval task.

7.3.1.2 Attention Models for Heterogeneous Graphs

A heterogeneous network is a network with multiple types of nodes, links, and even attributes. The structural heterogeneity and rich semantic information bring challenges for designing graph neural networks to fuse information.

Definition 7.4. A *heterogeneous graph* is defined as $\mathscr{G} = (\mathscr{V}, \mathscr{E}, \phi, \psi)$, where $\mathscr{V}$ is the set of node objects and $\mathscr{E}$ is the set of edges. Each node $v \in \mathscr{V}$ is associated with a node type $\phi(v)$, and each edge $(i, j) \in \mathscr{E}$ is associated with an edge type $\psi((i, j))$.

We introduce how the challenge in embedding could be tackled using Heterogeneous graph Attention Network (HAN) (Wang et al, 2019m). Different from traditional GNNs, information propagation on HAN is conducted based on meta-paths.

Definition 7.5. A *meta-path* Φ is defined as a path with the form $v_{i_1} \xrightarrow{r_1} v_{i_2} \xrightarrow{r_2} \cdots \xrightarrow{r_{l-1}} v_{i_l}$, abbreviated as $v_{i_1} v_{i_2} \cdots v_{i_l}$ with a composite relation $r_1 \circ r_2 \circ \cdots \circ r_{l-1}$.

To learn the embedding of node i, we propagate the embeddings from its neighbors within the meta-path. The set of neighbor nodes is denoted as $\mathscr{V}_i^{\Phi}$. Considering that different types of nodes have different feature spaces, a node embedding is first projected to the same space $\mathbf{h}^{j'} = M_{\phi_i} \mathbf{h}^j$. Here M_{ϕ_i} is the transformation matrix for node type ϕ_i. The attention mechanism in HAN is similar to GAT, except that we need to consider the type of meta-path that is currently sampled. Specifically,

$$\mathbf{z}^{i,\Phi} = \sigma\left(\sum_{j \in \mathscr{V}_i^{\Phi}} \alpha_{i,j}^{\Phi} \mathbf{h}^{j'}\right), \tag{7.25}$$

where the normalized attention score is

$$\alpha_{i,j}^{\Phi} = \text{softmax}(e_{i,j}^{\Phi}) = \text{softmax}(attn(\mathbf{h}^{i'}, \mathbf{h}^{j'}; \Phi)). \tag{7.26}$$

Given a set of meta-paths $\{\Phi_1, ..., \Phi_P\}$, we can obtain a group of node embeddings denoted as $\{\mathbf{z}^{i,\Phi_1}, ..., \mathbf{z}^{i,\Phi_P}\}$. To fuse embeddings across different meta-paths, another attention algorithm is applied. The fused embedding is computed as:

$$\mathbf{z}^i = \sum_{p=1}^{P} \beta_{\Phi_p} \mathbf{z}^{i,\Phi_p}, \tag{7.27}$$

where the normalized attention score is

$$\beta_{\Phi_p} = \text{softmax}(w_{\Phi_p}) = \text{softmax}\left(\frac{1}{|\mathscr{V}|} \sum_{i \in \mathscr{V}} \mathbf{q}^{\top} \cdot \text{MLP}(\mathbf{z}^{i,\Phi_p})\right), \tag{7.28}$$

where $\mathbf{q}$ is a learnable semantic vector. MLP$(\cdot)$ denotes a one-layer MLP module. w_{Φ_p} can be explained as the importance of the meta-path Φ_p. Besides modeling heterogeneous types of nodes and edges, HetGNN (Zhang et al, 2019b) extends the discussion by considering heterogeneity in node attributes (e.g., images, texts, categorical features).

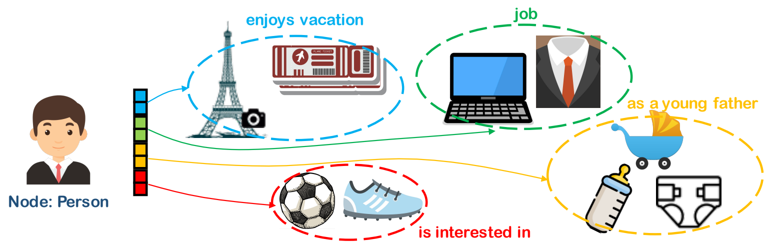

Fig. 7.6: Using multiple embeddings to represent the interests of a user. Each embedding segment corresponds to one aspect in data (Liu et al, 2019a).

7.3.2 *Disentangled Representation Learning on Graphs*

Traditional representation learning is limited in interpretability due to the opacity of the representation space. Different from manual feature engineering where the meaning of each resultant feature dimension is specified, the meaning of each dimension of the representation space is unknown. Representation learning on graphs also suffers from this limitation. To tackle this issue, several approaches have been proposed to enable assigning concrete meanings to different representation dimensions, thus improving the interpretability of representation learning on graphs.

7.3.2.1 Is A Single Vector Enough?

Many existing representation learning methods on graphs focus on learning a single embedding for each node. However, for those scenarios where some nodes have multiple facets, is a single vector enough to represent each node? Solving such a problem is of great practical value for applications such as recommender systems, where users could have multiple interests. In this case, we could use multiple embeddings to represent each user, and each embedding corresponds to one interest. An example is shown in Fig. 7.6. Specifically, if $\mathbf{h}^i \in \mathbb{R}^D$ denotes the embedding of node i, then $\mathbf{h}^i = [\mathbf{h}^{i,1}; \mathbf{h}^{i,2}; ...; \mathbf{h}^{i,K}]$, where $\mathbf{h}^{i,k} \in \mathbb{R}^{D/K}$ is the embedding for the k-th facet. There are two challenges in learning disentangled representations, i.e., how to discover the K facets, and how to distinguish the update of different embeddings during the training process. The facets could be discovered in an unsupervised manner by using clustering, where each cluster represents a facet. In the following parts, we introduce several approaches for learning disentangled node embeddings on graphs.

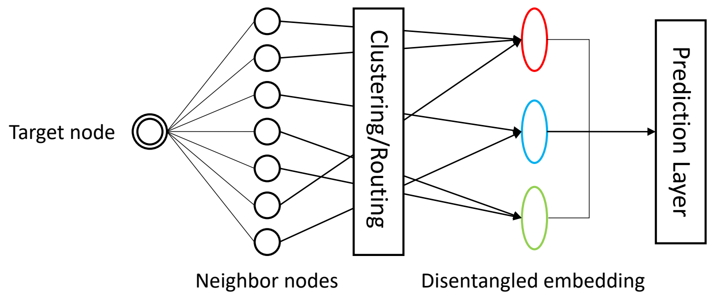

Fig. 7.7: The high-level idea of learning the disentangled node embedding for a target node by using clustering or dynamic routing.

7.3.2.2 Prototypes-Based Soft-Cluster Assignment

We discuss the techniques in the context of recommender system design. Facets that represent item types are discovered as we learn user and item embeddings. Here we assume that each item only has one facet, while each user could still have multiple facets. The embedding of item t is simply $\mathbf{h}^t$, while the embedding of user u is $\mathbf{h}^u = [\mathbf{h}^{u,1}; \mathbf{h}^{u,2}; ...; \mathbf{h}^{u,K}]$. Each item t is associated with a one-hot vector $\mathbf{c}_t = [c_{t,1}, c_{t,2}, ..., c_{t,K}]$, where $c_{t,k} = 1$ if t belongs to facet k, and $c_{t,k} = 0$ otherwise. Besides node embeddings, we also need to learn a set of *prototype* embeddings $\{\mathbf{m}^k\}_{k=1}^K$. The one-hot vector is drawn from the categorical distribution as below:

$$\mathbf{c}_t \sim \text{categorical}(\text{softmax}([s_{t,1}, s_{t,2}, .., s_{t,K}])), \quad s_{t,k} = cos(\mathbf{h}^t, \mathbf{m}^k)/\tau, \tag{7.29}$$

where τ is a hyper-parameter that scales the cosine similarity. Then, the probability of observing an edge (u,t) is

$$p(t|u, \mathbf{c}_t) \propto \sum_{k=1}^{K} c_{t,k} \cdot \text{similarity}(\mathbf{h}^t, \mathbf{h}^{u,k}). \tag{7.30}$$

Besides the fundamental learning process introduced above, the variational autoencoder framework could also be applied to regularize the learning process (Ma et al, 2019c). The item embeddings and prototype embeddings are jointly updated until convergence. The embedding of each user $\mathbf{h}^u$ is determined by aggregating the embeddings of interacted items, where $\mathbf{h}^{u,k}$ collects embeddings from items that also belong to facet k. In the learning process, the cluster discovery, node-cluster assignments, and embedding learning are jointly conducted.

7.3.2.3 Dynamic Routing Based Clustering

The idea of using dynamic routing for disentangled node representation learning is motivated by the Capsule Network (Sabour et al, 2017). There are two layers of capsules, i.e., low-level capsules and high-level capsules. Given a user u, the set of items that he has interacted with is denoted as $\mathscr{V}_u$. The set of low-level capsules is $\{\mathbf{c}_i^l\}$, $i \in \mathscr{V}_u$, so each capsule is the embedding of an interacted item. The set of high-level capsules is $\{\mathbf{c}_k^h\}$, $1 \leq k \leq K$, where $\mathbf{c}_k^h$ represents the user's k-th interest.

The routing logit value $b_{i,k}$ between low-level capsule i and high-level capsule k is computed as:

$$b_{i,k} = (\mathbf{c}_k^h)^\top S\,\mathbf{c}_i^l, \tag{7.31}$$

where S is the bilinear mapping matrix. Then, the intermediate embedding for high-level capsule k is computed as a weighted sum of low-level capsules,

$$\begin{aligned} \mathbf{z}_k^h &= \sum_{i \in \mathscr{V}_u} w_{i,k}\, S\,\mathbf{c}_i^l, \\ w_{i,k} &= \frac{\exp(b_{i,k})}{\sum_{k'=1}^{K} \exp(b_{i,k'})} \end{aligned} \tag{7.32}$$

so $w_{i,k}$ can be seen as the attention weights connecting the two capsules. Finally, a "squash" function is applied to obtain the embedding of high-level capsules:

$$\mathbf{c}_k^h = \text{squash}(\mathbf{z}_k^h) = \frac{\|\mathbf{z}_k^h\|^2}{1 + \|\mathbf{z}_k^h\|^2} \frac{\mathbf{z}_k^h}{\|\mathbf{z}_k^h\|^2}. \tag{7.33}$$

The above steps constitute one iteration of dynamic routing. The routing process is usually repeated for several iterations to converge. When the routing finishes, the high-level capsules can be used to represent the user u with multiple interests, to be fed into subsequent network modules for inference (Li et al, 2019b), as shown in Fig. 7.7.

7.4 Evaluation of Graph Neural Networks Explanations

In this section, we introduce the setting for evaluating GNN explanations. This includes the *datasets* that are commonly used for constructing and explaining GNNs, as well as the *metrics* that evaluate different aspects of explanations.

7.4.1 Benchmark Datasets

As more approaches have been proposed for explaining GNNs, a variety of datasets have been used to assess their effectiveness. As such a research direction is still in the initial stage of development, a universally accepted benchmark dataset, such as the COCO dataset for image object detection, has not yet been proposed. Here we list a number of datasets that have been used for developing GNN explanation approaches, including synthetic datasets and real-world datasets.

7.4.1.1 Synthetic Datasets

It is difficult to evaluate explanations because there are no ground truths in datasets to compare with. A strategy to mitigate this problem is to use synthetic datasets. In this case, motifs designed by humans could be added to data to play the role as ground truths, and these motifs are assumed to be relevant to the learning task. Some synthetic graph datasets are listed as below.

- **BA-Shapes** (Ying et al, 2019): A Barabási-Albert graph with 300 nodes, to which 80 house-shaped motifs are attached randomly. It is then further augmented by adding 10% random edges.
- **BA-Community** (Ying et al, 2019): A graph consists of two BA-Shapes, with node features in different BA-Shapes following different normal distributions to distinguish them.
- **Tree-Cycle** (Ying et al, 2019): A graph based on an eight-level balance tree, to which 80 hexagonal motifs are attached randomly to the tree.
- **Tree-Grid** (Ying et al, 2019): A graph similar to Tree-Cycle, but with 80 3-by-3 grid motifs instead of the hexagonal motifs.
- **Noisy BA-Community, Noisy Tree-Cycle, Noisy Tree-Grid** (Lin et al, 2020a): These four datasets are obtained by adding 40 important and 10 unimportant node features to the corresponding datasets list above. This design can help to test a method's ability to identify important node features.
- **BA-2Motifs** (Luo et al, 2020): A dataset contains 800 independent graphs that are obtained by adding either a pentagon motif or a house motif to the base BA graph. This dataset is designed for graph classification task while previous ones are for node classification task.

7.4.1.2 Real-World Datasets

Some examples of real-world graph datasets are listed as below.

- **MUTAG** (**?**): A dataset consisting of 4,337 molecule graphs that are labeled mutagenic or non-mutagenic. The nodes and edges in a graph represent the atoms and chemical bonds. Related studies have shown that molecules with carbon rings and Nitro group (NO_2) may lead to mutagenic effects. Also, there are several other molecule datasets, such as **BBBP**, **BACE** and **TOX21** (Pope et al, 2019).
- **REDDIT-BINARY** (Yanardag and Vishwanathan, 2015): A online-discussion interaction dataset. It contains 2,000 graphs, and each of them is labeled as a

question-answer based or a discussion based community. The nodes and edges represent the users and their interactions, respectively.

- **Delaney Solubility** (Delaney, 2004): A molecule dataset with 1,127 molecule graphs, and their labels are the water-octanol partition coefficient. This dataset is usually for graph regression tasks.
- **Bitcoin-Alpha, Bitcoin-OTC** (Kumar et al, 2016): Two trust-weighted signed networks. Each of them consists of a graph whose nodes are accounts trading on the Bitcoin-Alpha or Bitcoin-OTC platform. The nodes are labeled trustworthy or not according to other members' ratings.
- **MNIST SuperPixel-Graph** (Dwivedi et al, 2020): An image dataset in the form of graphs. Each sample is a graph converted from the corresponding image in the MNIST dataset. Every node is a super-pixel that represents the intensity of corresponding region.

7.4.2 Evaluation Metrics

An appropriate evaluation metric is crucial for methods comparison. Explanation visualization such as heat-map, due to its intuitiveness, has been widely used in explanation for image and text data. However, it loses this advantage since graph data is not intuitive to understand. Only experts with the domain knowledge can make judgement. In this section, we introduce several commonly used metrics.

- **Accuracy** is only appropriate for datasets with ground truth. The synthetic datasets usually contain the ground truth that is defined by the rule they are constructed. For example, in molecule datasets, the molecule with NO_2 and carbon ring is mutagenic. Considering that carbon ring also occurs in non-mutagenic molecule, the NO_2 group is considered as ground truth. F1 score and ROC-AUC are commonly used accuracy metrics. The limitation of the accuracy metrics is that it is unknown whether the GNN model makes predictions in the same way as humans (i.e., whether the pre-defined ground truth is really valid).
- **Fidelity** (Pope et al, 2019) follows the intuition that removing the truly important features will significantly decrease the model performance. Formally, fidelity is defined as:

$$fidelity = \frac{1}{N}\sum_{i=1}^{N}\left(f^{y_i}\left(\mathcal{G}_i\right) - f^{y_i}\left(\mathcal{G}_i \setminus \mathcal{G}_i'\right)\right) \tag{7.34}$$

  where f is the output function target model. $\mathcal{G}_i$ is the i-th graph, $\mathcal{G}_i'$ is the explanation for it, and $\mathcal{G}_i \setminus \mathcal{G}_i'$ represents the perturbed i-th graph in which the identified explanation is removed.
- **Contrastivity** (Pope et al, 2019) uses Hamming distance to measure the differences between two explanations. These two explanations correspond the model's prediction of one instance for different classes. It is assumed that models would highlight different features when making predictions for different

classes. The higher the contrastivity, the better the performance of the interpreter.

- **Sparsity** (Pope et al, 2019) is calculated as the ratio of explanation graph size to input graph size. In some cases, explanations are encouraged to be sparse, because a good explanation should include only the essential features as far as possible and discard the irrelevant ones.
- **Stability** (Sanchez-Lengeling et al, 2020) measures the performance gap of the interpreter before and after adding noise to the explanation. It suggests that a good explanation should be robust to slight changes in the input that do not affect the model's prediction.

7.5 Future Directions

Interpretation on graph neural networks is an emerging domain. There are still many challenges to be tackled. In this section, we list several future directions towards improving the interpretability of graph neural networks.

First, some online applications require real-time responses from models and algorithms. It thus puts forward high requirements on the efficiency of explanation methods. However, many GNN explanation methods conduct sampling or highly iterative algorithms to obtain the results, which is time-consuming. Therefore, one future research direction is how to develop more efficient explanation algorithms without significantly sacrificing explanation precision.

Second, although more and more methods have been developed for interpreting GNN models, how to utilize interpretation towards identifying GNN model defects and improving model properties is still rarely discussed in existing work. Will GNN models be largely affected by adversarial attacks or backdoor attacks? Can interpretation help us to tackle these issues? How to improve GNN models if they have been found to be biased or untrustworthy?

Third, besides attention methods and disentangled representation learning, are there other modeling or training paradigms that could also improve GNN interpretability? In the interpretable machine learning domain, some researchers are interested in providing causal relations between variables, while some others prefer using logic rules for reasoning. Therefore, how to bring causality into GNN learning, or how to use incorporate logic reasoning into GNN inference, may be an interesting direction to explore.

Fourth, most existing efforts on interpretable machine learning have been devoted to getting more accurate interpretation, while the human experience aspect is usually overlooked. For end-users, friendly interpretation can promote user experience, and gain their trust to the system. For domain experts without machine learning background, an intuitive interface helps integrate them into the system improvement loop. Therefore, another possible direction is how to incorporate human-computer interaction (HCI) to show explanation in a more user-friendly format, or how to design better human-computer interfaces to facilitate user interactions with the model.

Acknowledgements The work is, in part, supported by NSF (#IIS-1900990, #IIS-1718840, #IIS-1750074). The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agencies.

Editor's Notes: Similar to the general trend in the machine learning domain, explainability has been ever more widely recognized as an important metric for graph neural networks in addition to those well recognized before such as effectiveness (Chapter 4), complexity (Chapter 5), efficiency (Chapter 6), and robustness (Chapter 8). Explainability can not only broadly influence technique development (e.g., Chapters 9-18) by informing model developers of useful model details, but also could benefit domain experts in various application domains (e.g., Chapters 19-27) by providing them with explanations of predictions.