

Generative Causal Explanations for Graph Neural Networks

Wanyu Lin Hao Lan Baochun Li

Abstract

This paper presents *Gem*, a model-agnostic approach for providing interpretable explanations for any GNNs on various graph learning tasks. Specifically, we formulate the problem of providing explanations for the decisions of GNNs as a causal learning task. Then we train a causal explanation model equipped with a loss function based on Granger causality. Different from existing explainers for GNNs, *Gem* explains GNNs on graph-structured data from a causal perspective. It has better generalization ability as it has no requirements on the internal structure of the GNNs or prior knowledge on the graph learning tasks. In addition, *Gem*¹, once trained, can be used to explain the target GNN very quickly. Our theoretical analysis shows that several recent explainers fall into a unified framework of *additive feature attribution methods*. Experimental results on synthetic and real-world datasets show that *Gem* achieves a relative increase of the explanation accuracy by up to 30% and speeds up the explanation process by up to 110 \times as compared to its state-of-the-art alternatives.

1. Introduction

Many problems in scientific domains, ranging from social networks (Lin et al., 2020) to biology (Zitnik et al., 2018) and chemistry (Zitnik & Leskovec, 2017), can be naturally modeled as problems of property learning on graphs. For example, in biology, identifying the functionality of proteins is critical to find the proteins associated with a disease, where proteins are represented by local protein-protein interaction (PPI) graphs. Supervised learning of graphs, especially with graph neural networks (GNNs), has had a significant impact on these domains, mainly owing to their efficiency and capability of inductive learning (Hamilton et al., 2017).

Despite their practical success, most GNNs are deployed as black boxes, lacking explicit declarative knowledge representations. Therefore, they have difficulty in generating the required underlying explanatory structure. The deficiency of

explanations for the decisions of GNNs significantly hinders the applicability of these models in decision-critical settings, where both predictive performance and interpretability are of paramount importance. For example, medical decisions are increasingly being assisted by complex predictions that should lend themselves to be verified by human experts easily. Model explanations allow us to argue for model decisions and exhibit the situation when algorithmic decisions might be biased or discriminating. In addition, precise explanations may facilitate model debugging and error analysis, which may help decide which model would better describe the data’s underlying semantics.

While explaining graph neural networks on graphs is still a nascent research topic, a few recent works have emerged (Luo et al., 2020; Vu & Thai, 2020; Ying et al., 2019; Yuan et al., 2020), each with its own perspective on this topic. In particular, XGNN (Yuan et al., 2020) was proposed to investigate graph patterns that lead to a specific class, while GNNExplainer (Ying et al., 2019) provided the local explanation for a single instance (a node/link/graph), by determining a compact subgraph leading to its prediction. PGM-Explainer (Vu & Thai, 2020) explored the dependencies of explained features in the form of conditional probability, which is naturally designed for explaining a single instance.

However, verifying if a target GNN works as expected often requires a considerable amount of explanations for providing a global view of explanations. For this end, PGExplainer (Luo et al., 2020) learns a multilayer perceptron (MLP) to explain multiple instances collectively. However, PGExplainer heavily relies on node embeddings from the target GNN, which may not be obtained without knowing its internal model structure and parameters. Besides, PGExplainer can not explain any graph tasks without explicit motifs. Taking MUTAG as an example, PGExplainer assumes that NH₂ and NO₂ are the motifs for the mutagen graphs and filters out the instances without these two motifs. We verified that the assumption might not be reasonable by looking at the dataset statistics (provided in Appendix B). More specifically, PGExplainer fails to explain the instances without explicit motifs under their assumptions. Motivated by this observation, we aim to provide fast and accurate explanations for any GNN-based models without the limitations above.

¹The source code and appendix are in the supplementary.

In this work, we propose a new methodology, called *Gem*, to provide interpretable explanations for any GNNs on the graph using causal explanation models. To the best of our knowledge, while the notion of causality has been used for interpretable machine learning on images or texts, this is the first effort from a causal perspective to explain graph neural networks. Specifically, our causal objective is built upon the notion of Granger causality, which comes from the pioneering work of Wiener and Granger (Bressler & Seth, 2011; Granger, 1969; Wiener, 1956). Granger causality declares a causal relationship $x_i \rightarrow Y$ between variables x_i and Y if we are better able to predict Y using all available information than if the information apart from x_i had been used. In the graph domain, if the absence of an edge/node x_i decreases the ability to predict Y , then there is a causal relationship between this edge/node and its corresponding prediction. Based on the insights from neuroscience (Biswal et al., 1997), we extend the notion of Granger causality to characterize the explanation of an instance by its local subgraphs.

We note that the concept of Granger causality is probabilistic, and the graph data is inherently interdependent, i.e., edges or nodes are correlated variables. Directly applying Granger causality may lead to incorrectly detected causal relations. In addition, we envision that the resulting explanations should be human-intelligible and valid. For example, in some applications such as chemistry, an explanation for the mutagen graph is a functional group and should be connected. Accordingly, we propose an approximate computation strategy that makes our method viable for graph data with interdependency, under reasonable assumptions on the causal objective.

In particular, we incorporate various graph rules, such as the connectivity check, to encourage the obtained explanations to be valid and human-intelligible. Then we train causal explanation models that learn to distill compact subgraphs, causing the outputs of the target GNNs. This approach is flexible and general since it has no requirements on the target model to be explained (commonly referred to as “model-agnostic”), or no assumptions on the learning tasks (explicit motifs for identifying a particular class), and can provide local and global views of the explanations. In particular, it does not require retraining or adapting to the original model. In other words, once trained, *Gem* can be used to explain the target GNN models with little time.

Highlights of our original contributions are as follows. We propose a new methodology to explain graph neural networks on the graph from the causal perspective; to the best of our knowledge, such an approach has not been used for interpreting GNNs on the graph so far. We introduce causal objectives for better estimates of the causal effect in our methodology and provide an approximate computation strat-

egy to deal with graph data with interdependency. Various graph rules are incorporated to ensure that the obtained explanations are valid. We then use a causal objective to train a graph generative model as the explainer, which can automatically explain the target GNNs with little time. We theoretically analyze that several recent methods, including *Gem*, all fall into the framework of additive feature attribution methods, which essentially solve the same optimization problem with different approximation methods (provided in Appendix A). We empirically demonstrate that *Gem* is significantly faster and more accurate than alternative methods.

2. Problem Setup

A set of graphs can be represented as $\mathcal{G} = \{G_i\}_{i=1}^N$, where $|\mathcal{G}| = N$. Each graph is denoted as $G_i = (V_i, E_i)$, where E_i denotes the edge set and $V_i = \{v_1^i, v_2^i, \dots, v_{|V_i|}^i\}$ is the node set of graph i . In many applications, nodes are associated with d -dimensional node features $X_i = \{x_1^i, x_2^i, \dots, x_{|V_i|}^i\}$, $x_j^i \in \mathbb{R}^d$. Without loss of generality, we consider the problem of explaining a graph neural network-based classification task. This task can be node classification or graph classification. For graph classification, we associate each graph G_i with a label, y_i , where $y_i \in \mathcal{Y} = \{c_1, c_2, \dots, c_l\}$, and l is the number of categories. The dataset $\mathcal{D} = \{(G_i, y_i)\}_{i=1}^N$ is represented by pairs of graph instances and graph labels. Examples of such task include classifying the drug molecule graphs according to their functionality.

In the node classification setting, each node $v_j \in V$ of a graph G is associated with a corresponding node label $y_j \in \mathcal{Y}$. Examples of this kind include classifying papers in a citation network, or entities in a social network such as Reddit. The dataset $\mathcal{D} = \{(v_j, y_j)\}_{j=1}^{|V|}$ is represented by pairs of nodes and node labels. In general, we use I_i to represent an instance, which is equivalent to v_i for node classification or G_i for graph classification.

GNN family models. Graph neural networks (GNNs) are a family of graph message passing architectures that incorporate graph structure and node features to learn a dense representation of a node or the entire graph. In essence, GNNs follow a neighborhood aggregation strategy, where the node representations are updated via iteratively aggregating the representations from its neighbors in the graph. Graph convolutional networks (GCNs) use mean pooling (Kipf & Welling, 2017) for aggregation, while GraphSage aggregates the node features via mean/max/LSTM pooling (Hamilton et al., 2017). Taking GCNs as an example, the basic operator for the neighborhood information aggregation is the element-wise **mean**. After L iterations of aggregation, a node’s representation can capture the structural information

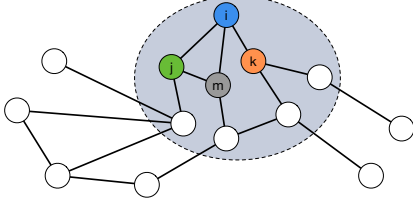


Figure 1. An illustration of the computation graph (best viewed in color). Node i is the target node to be explained.

within its L -hop graph neighborhood.

Formally, a graph neural network (GNN) can be written as a function $f(\cdot) : \mathcal{G} \rightarrow \mathcal{Y}$ or $f(\cdot) : V \rightarrow \mathcal{Y}$. The former is a graph-level classifier, and the latter is a node-level classifier. Typically, a GNN $f(\cdot)$ is trained with an objective function $\mathcal{L} : y \times \tilde{y} \rightarrow s$ that computes a scalar loss $s \in \mathbb{R}$ after comparing the model’s predictive output \tilde{y} to a ground-truth output y . The categorical cross-entropy for classification models is commonly used for such objectives.

Objective. We are given a pre-trained classification model, represented by $f(\cdot)$, and our ultimate goal is to obtain an explanation model, denoted as $f(\cdot)_{\text{exp}}$, that can provide fast and accurate explanations for the pre-trained model, which can also be called a target GNN. Intrinsically, an explanation is a subgraph that is the most relevant for a prediction — the outcome of the target GNN, denotes as \tilde{y} . Consistent with previous studies in the literature (Yuan et al., 2020), we focus on explanations on graph structures. In particular, we specifically do not require access to, or knowledge of, the process by which the classification model produces its output, nor do we require the classification model to be differentiable or any specific form. We allow the explainers to retrieve different predictions by performing queries on $f(\cdot)$.

3. Methodology

In essence, the core of the GNNs is a neighborhood-based aggregation process, where a prediction of an instance is fully determined by its computation graph. Let us use $G_i^c = (V_i^c, A_i^c, X_i^c)$ to represent the computation graph of an instance i , where V_i^c is the node set, $A_i^c \in \{0, 1\}$ indicates the adjacency matrix, and X_i^c is the feature matrix of the computation graph. Typically, a GNN learns a conditional distribution denoted as $\mathcal{P}(Y|G_i^c)$, where Y is a random variable representing the class labels. For clarity, let us see an example graph, shown in Figure 1, which will also be used throughout this paper. In this example, a target GNN is trained for node classification, and the node i is the target node to be explained. Oftentimes, the computation graph of node i is a two-hop subgraph, highlighted in Figure 1.

Therefore, the setting we focus on can be reformulated as

the following: we are given a GNN-based classification model that processes the computation graph of an instance (a node or a graph), denoted as G^c , and generates the corresponding outputs $p(Y|G^c)$ for predicting \tilde{y} . Unlike the node classification task, when the target GNN is trained for graph classification, the computation graph of an instance will be the entire graph. Accordingly, this work seeks to generate an explanation, a subgraph of G^c that is most relevant for predicting \tilde{y} , efficiently and automatically. We use G^s to denote the generated explanation. Our setting is general and works for any graph learning tasks, including node classification and graph classification. Our ultimate goal is to encourage a compact subgraph of the computation graph to have a large causal influence on the outcome of the target GNN.

Differences from PGExplainer. PGExplainer is the most closely related work to our study, as both PGExplainer and *Gem* adopt parameterized networks to provide local and global views for model explanations. However, PGExplainer relies on node embeddings from the target GNN to learn a multilayer perceptron, which may not be obtained without knowing its internal model structure. In contrast, to explain an instance (a node or a graph), *Gem* simply inputs the original computation graph into the explainer and outputs a compact explanation graph. In other words, *Gem* does not require any prior knowledge of the internal model structure (the target GNN) and parameters, or any prior knowledge of the motifs associated with the graph learning tasks. Therefore, it exhibits better generalization abilities. In what follows, we will present *Gem*, our model-agnostic approach for providing interpretable explanations for any GNNs on a variety of graph learning tasks. The design of *Gem* is based upon principles of causality, in particular Granger causality (Granger, 1969).

Granger causality (Granger, 1969; 1980). In general, Granger causality describes the relationships between two (or more) variables when one is causing the other. Specifically, if we are better able to predict variable \tilde{y} using all available information U than if the information apart from variable x_i had been used, we say that x_i Granger-causes \tilde{y} (Granger, 1980), denoted by $x_i \rightarrow \tilde{y}^2$.

The crux of our approach is to train an explanation model, or an *explainer*, to explain the target graph neural network. Specifically, *Gem* is trained with the guidance built on the first principles of Granger causality. Here we extend Granger causality to the case where a compact subgraph G^s of the computation graph is considered the main cause of the corresponding prediction \tilde{y} . This is inspired by a long-held belief in neuroscience that the structural connectivity local to a certain area somehow dictates the function of that

²We are aware of the drawbacks of reusing notations. x_i and \tilde{y} in this definition represent any random variables for simplicity.

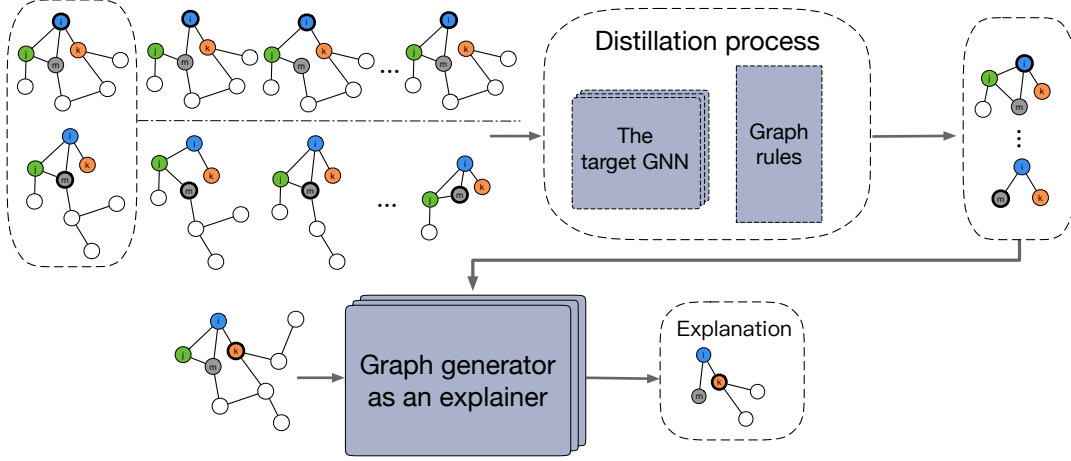


Figure 2. Illustration of *Gem*. 1) The distillation process of generating the ground-truth explanations based on the first principles of Granger causality; 2) Training the explainer that can be used to generate explanations for the target GNNs.

piece (Biswal et al., 1997). Due to the inherent property of GNNs, the computation graph contains all the relevant information that causes the prediction of the target GNN. Under the assumption that Granger causality was built upon, we can squeeze the cause of the prediction, G^s , from the computation graph. Therefore, we can use the given definition to quantify the degree to which part of the computation graph causes the prediction of the target GNN. In principle, the notion of Granger causality would hold if $p(\tilde{y} | G^c) = p(\tilde{y} | G^s)$ holds.

3.1. Causal Objective

Given a pre-trained/target GNN and an instance G^c , we use δ_{G^c} to denote the model error of the target GNN when considering the computation graph, while $\delta_{G^c \setminus \{e_j\}}$ represents the model error excluding the information from the edge e_j , where $e_j \in G^c$. With these two definitions and following the notion of Granger causality, we can quantify the causal contribution of an edge e_j to the output of the target GNN. More specifically, the causal contribution of the edge e_j is defined as the decrease in model error, formulated as Eq. (1):

$$\Delta_{\delta, e_j} = \delta_{G^c \setminus \{e_j\}} - \delta_{G^c} \quad (1)$$

To calculate δ_{G^c} and $\delta_{G^c \setminus \{e_j\}}$, we first compute the outputs corresponding to the computation graph G^c and the one excluding edge e_j , $G^c \setminus \{e_j\}$, based on the pre-trained GNN. For simplicity, the pre-trained GNN is denoted as $f(\cdot)$. Then, the associated outputs can be formulated as Eq. (2) and Eq. (3) respectively:

$$\tilde{y}_{G^c} = f(G^c) \quad (2)$$

$$\tilde{y}_{G^c \setminus \{e_j\}} = f(G^c \setminus \{e_j\}) \quad (3)$$

Then we compare the outputs of the target GNN, e.g., \tilde{y}_{G^c} and $\tilde{y}_{G^c \setminus \{e_j\}}$, with the ground-truth label y , respectively. In particular, we use the loss function of the pre-trained GNN

as the metric to measure the model error, denoted as \mathcal{L} . The mathematical formulations are shown as Eq. (4) and Eq. (5):

$$\delta_{G^c} = \mathcal{L}(y, \tilde{y}_{G^c}) \quad (4)$$

$$\delta_{G^c \setminus \{e_j\}} = \mathcal{L}(y, \tilde{y}_{G^c \setminus \{e_j\}}) \quad (5)$$

Now, the causal contribution of an edge e_j can be measured by the loss difference associated with the computation graph and the one deleting edge e_j .

Recall that we are seeking a ‘‘guidance’’ that can be used to train our explainer, encouraging its outcome to be effective explanations. Intrinsically, Δ_{δ, e_j} can be viewed as capturing the individual causal effect (ICE) (Goldstein et al., 2015) of the input G^c with values e_j on the output \tilde{y} . Therefore, it is straightforward to obtain the most relevant subgraph for predicting \tilde{y} based on Δ_{δ, e_j} , which we call the ground-truth distillation process.

Ideally, given the edges’ causal contributions in a computation graph, we can sort the edges accordingly and distill the top- K most relevant edges as a prediction explanation. However, due to the special representations of the graph data, the casual contributions from the edges are not independent, e.g., a 1-hop neighbor of a node can also be a 2-hop neighbor of the same node due to cycles. To this end, we further incorporate various graph rules to encourage the distillation process to be more effective. We believe that data characteristics are the most crucial factor in deciding which graph rules to use. It is necessary to understand the principle of the learning task, and the limitation of a human-intelligible explanation might be to prevent spurious explanations. In the application of bioinformatics, such as the MUTAG dataset, the explanation is a functional group, and therefore, the distilled top- K edges should be connected. Nevertheless, in graph representation-based Digital Pathology, such as the cell-graphs towards cancer subtyping, the explanation often contains subsets of cells and cellular interactions (Jaume

et al., 2020). In this particular scenario, the connectivity constraint is unnecessary. The detailed distillation process is presented in Appendix C.

Using the distilled ground truth, denoted as $\tilde{G}^s = (\tilde{V}^s, \tilde{A}^s, \tilde{X}^s)$, we can train supervised learning models to learn to explain any other GNN models based solely on its outputs, and without the need to retrain the model to be explained. The workflow of *Gem* is illustrated in Figure 2. Note that generating an explanation based on the explainer is not necessary in situations where ground-truth labels of the instances are readily available. In those cases, pre-calculated Δ_{δ, e_j} and our distillation process can directly be used to explain the pre-trained GNN.

3.2. Graph Generative Model as an Explainer

In principle, any graph generative models that can be trained to output graph-structured data can be used as a causal explanation model. Due to their simplicity and expressiveness, we focus on auto-encoder architectures utilizing graph convolutions (Kipf & Welling, 2016; Li et al., 2018a;b). In particular, we use a model consisting of a graph convolutional network encoder and an inner product decoder (Kipf & Welling, 2016). We leave the exploration of other generative models for future work.

More concretely, in our explainer, we first apply several graph convolutional layers to aggregate neighborhood information and learn node features. Then we use the inner product decoder to reconstruct the adjacency matrix \hat{A}^c , a compact subgraph that only contains the most relevant portion of a computation graph to its prediction. In particular, each value in \hat{A}^c denotes the contribution of a particular edge to the prediction of G^c . Formally, the reconstruction process can be formulated as:

$$Z = \text{GCNs}(A^c, X^c) \quad (6)$$

$$\hat{A}^c = \sigma(ZZ^T) \quad (7)$$

where A^c is the adjacency matrix of the computation graph for the target instance, X^c denotes the node features, and Z is the learned node features.

Explanation for node classification. The output of an explanation for a target node is a compact subgraph of the computation graph that is most influential for the prediction label. To answer the question of ‘‘How did a GNN predict that a given node has label \tilde{y} ?’’, the explainer should capture which node to explain. Specifically, we use a node labeling technique to mark nodes’ different roles in a computation graph. The generated node labels are then transformed into vectors (e.g., one-hot encodings) and treated as the nodes’ feature vector — X^c . Note that the labels here represent the structural information of nodes within a computation graph, which are different from the classification/prediction labels.

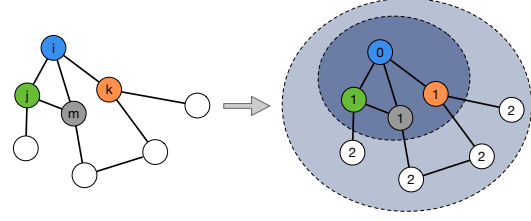


Figure 3. An illustration of the node labeling for node explanation generation (best view in color). The node i is the target node to be explained and the graph in the left-hand side is i ’s computation graph. i is labeled as 0, while j and k are labeled as 1 as they are one-hop away from i .

The intuition underlying this node labeling technique is that nodes with different relative positions to the target node may have different structural importance to its prediction label \tilde{y} . By incorporating the relative role features, *Gem* would capture which node to explain in a computation graph.

Specifically, our node labeling method is derived based on two principles: 1) The target node, denoted as i , always has the distinctive label ‘‘0.’’ 2) A node relative position within a computation graph can be described by its radius regarding the center/target node, i.e., $d(i, j)$ and $d(i, k)$. For two nodes, if $d(i, j) = d(i, k)$, then they have the same label. To further elucidate the used node labeling method, let us see an example shown in Figure 3. Node i is the target node, while both j and k are the one-hop neighbors of i . With our node labeling mechanism, node i is labeled as 0, while j and k are labeled as 1 as they are one-hop away from i .

Training the explainer. We envision that the reconstructed matrix \hat{A}^c is the weighted adjacency matrix that reflects the contributions of the edges to its prediction. Now we can apply the ‘‘guidance’’ distilled based on the notion of Granger causality, described in Sec. 3.1, to supervise the learning process. In particular, we use the root mean square error between the reconstructed weighted matrices and the true causal contributions distilled based on our proposed distillation process in Sec. 3.1.

One highlight of our explainer is its flexibility to choose the predictive model, which is commonly referred to as ‘‘model-agnosticism’’ (Ribeiro et al., 2016). Guided by the first principles of Granger causality, our explainer enables graph generative models to learn to generate compact subgraphs for model explanations. We do not need to retrain or adapt the predictive model to explain its decisions. Once trained, it can be used to construct explanations using the generative mapping for the target GNN with little time.

Computational complexity analysis. One may concern that it would be time-consuming to run through the training instances for obtaining the training ‘‘guidance.’’ We argue that our method amortizes the estimation cost by training

a graph generator to generate explanations for any given instances. In particular, *Gem* adopts a parameterized graph auto-encoder with GCN layers to generate explanations, which, once trained, can be utilized in the inductive setting to explain new instances. Specifically, the model parameter complexity of *Gem* is independent of the input graph size as it naturally inherits the advantages of GCNs (empirically verified in Appendix B). With the inductive property, the inference time complexity of *Gem* is $\mathcal{O}(|E|)$, where $|E|$ is the number of edges of the instance to be explained. Sec. 4 empirically verified the computation efficiency of *Gem*. In a nutshell, our solution transforms the task of producing explanations for a given GNN into a supervised learning task, trained based on the first principles of Granger causality. Then we can address the explanation task with existing supervised graph generative models.

Extensions to other learning tasks on graphs. Beyond node classification, our explainer can also provide explanations for link prediction and graph classification tasks without modifying its optimization algorithm. The key difference is the node labeling technique for marking the nodes’ roles in the computation graph. For example, to generate an explanation for the link prediction task, the explainer model should be able to identify which link to explain. An alternative approach is double-radius node labeling, marking the target link (connecting two target nodes) within the computation graph, proposed by Zhang *et al.* (Zhang & Chen, 2018). More concretely, node i ’s position is determined by its radius with respect to the two target nodes (x, y) , i.e., $(d(i, x), d(i, y))$.

Note that, due to properties of the graph structure invariant, there is no need to mark a particular node/link for graph classification tasks. Instead, we use a graph labeling method, called the Weisfeiler-Lehman (WL) algorithm (Weisfeiler & Lehman, 1968), to capture the structural roles of nodes within a computation graph, which has been widely used in graph isomorphism checking. For more details about the WL algorithm, We refer curious readers to (Weisfeiler & Lehman, 1968). In Sec. 4, we will empirically show that with the “guidance” based on Granger causality, complemented by node features from graph/node labeling, *Gem* can provide fast and accurate explanations for any graph learning tasks.

4. Experimental Studies

4.1. Datasets and Experimental Settings

Node classification with synthetic datasets. In the node classification setting, we built two synthetic datasets where ground truth explanations are available. In particular, we followed the data processing as reported in GNNExplainer (Ying *et al.*, 2019). The first dataset is BA-

shapes, where nodes are labeled based on their structural roles within a house-structured motif, including “top-node,” “middle-node,” “bottom-node,” and “none-node” (the ones not belonging to a house). The second dataset is Tree-cycles, in which nodes are labeled to indicate whether they belong to a cycle, including “cycle-node” and “none-node” (more details of the datasets are provided in Appendix B).

Graph classification with real-world datasets. For graph classification, we use two benchmark datasets from bioinformatics — Mutag (Debnath *et al.*, 1991) and NCI1 (Wale *et al.*, 2008). Mutag contains 4337 molecule graphs, where nodes represent atoms, and edges denote chemical bonds. The graph classes, including the non-mutagenic or the mutagenic class, indicate their mutagenic effects on the Gram-negative bacterium *Salmonella typhimurium*. NCI1 consists of 4110 instances, each of which is a chemical compound screened for activity against non-small cell lung cancer or ovarian cancer cell lines.

Baselines. We consider the state-of-the-art baselines that belong to the unified framework of additive feature attribution methods (The proof is provided in Appendix A) (Lundberg & Lee, 2017): GNNExplainer (Ying *et al.*, 2019) and PGExplainer (Luo *et al.*, 2020)³. GNNExplainer explains for a given instance at a time, while PGExplainer explains multiple instances collectively. Unless otherwise stated, all the hyperparameters of the baselines are the same in the source code. We do not include gradient-based method (Ying *et al.*, 2019), graph attention method (Velićković *et al.*, 2018), and Gradient (Pope *et al.*, 2019), since previous explainers (Luo *et al.*, 2020; Ying *et al.*, 2019) have shown their superiority over these methods.

Parameter settings of *Gem*. For all datasets on different tasks, associate explainers share the same structure (Kipf & Welling, 2016). Specifically, we first apply an inference model parameterized by a three-layer GCN with output dimensions 32, 32, and 16. Then the generative model is given by an inner product decoder between latent variables. The explainer models are trained with a learning rate of 0.01. We use hyperparameter K to control the size of the explanation subgraph and compare the performance of *Gem* with GNNExplainer and PGExplainer. The target GNN model accuracy on four datasets, more implementation details, and experimental results are presented in the Appendix B.

Evaluation metrics. An essential criterion for explanations is that they must be human interpretable, which implies that the generated explanations should be easy to understand. Taking BA-shapes as an example, the node label is determined by its position within a house-structured motif. The explanations for this dataset should be able to highlight the house structure. For interpretability, we use the visu-

³We use the source code released by the authors.

Table 1. Explanation Accuracy on Synthetic Datasets (%).

K	BA-SHAPES					TREE-CYCLES				
	5	6	7	8	9	6	7	8	9	10
<i>Gem</i>	93.4	97.1	97.1	97.1	99.3	86.1	87.5	92.5	93.9	95.4
GNNExplainer	82.4	88.2	91.2	91.2	94.1	14.3	46.8	74.6	91.4	96.1
PGExplainer	71.9	90.7	92.0	93.3	94.1	94.4	80.6	77.0	82.4	89.4

Table 2. Explanation Accuracy on Real-World Datasets (%).

K	MUTAG				NCI1			
	15	20	25	30	15	20	25	30
<i>Gem-0</i>	64.0	78.1	81.0	85.0	—	—	—	—
GNNExplainer-0	60.0	67.6	68.9	75.8	—	—	—	—
PGExplainer-0	22.5	38.5	57.6	72.3	—	—	—	—
<i>Gem</i>	66.3	78.0	82.1	83.4	56.9	65.3	68.9	72.8
GNNExplainer	67.1	74.9	75.8	80.9	59.3	61.8	69.6	72.0

alized explanations of different methods to analyze their performance qualitatively.

In addition, explanations seek to answer the question: when a GNN makes a prediction, which parts of the input graph are relevant? Ideally, the generated explanation/subgraph should lead to the same prediction outcome by the pre-trained GNN (e.g. $p(\tilde{y} | G^s)$ should be close to $p(\tilde{y} | G^c)$). In other words, a better explainer should be able to generate more compact subgraphs yet maintains the prediction accuracy while the associated explanations are fed into the pre-trained GNN. To this end, we generate the explanations for the test set based on *Gem*, GNNExplainer and PGExplainer, respectively. Then we use the predictions of the pre-trained GNN for the explanations to calculate the explanation accuracy.

The rationality of K selection is from the size of the ground-truth structure for the synthetic datasets. We do not have the ground-truth motif for the real-world datasets, therefore we select K according to the distribution of the graph size for each dataset, respectively (reported in the Appendix B). Note that a K that is too small may incur meaningless explanations; for example, the aromatic group is an essential component leading to the mutagenic effect. We evaluate the explanation performance under different K settings, starting from $K = 15$ for Mutag and NCI1.

4.2. Experimental Results

In what follows, we summarize the results of our experimental results and discuss our key findings. The explanation accuracy for synthetic datasets and real-world datasets in different K settings are reported in Table 1 and Table 2, respectively. As shown in Table 1, *Gem* consistently offers the best explanation accuracy in overall cases. In particular, *Gem* achieves 30% improvement when $K = 5$ on BA-shapes, compared with GNNExplainer. By looking at the explanations for a target node, shown in Figure 4, *Gem* can successfully identify the “house” motif that explains the node label (“middle-node” in red), when $K = 6$, while the GNNExplainer wrongly attributes the prediction to a

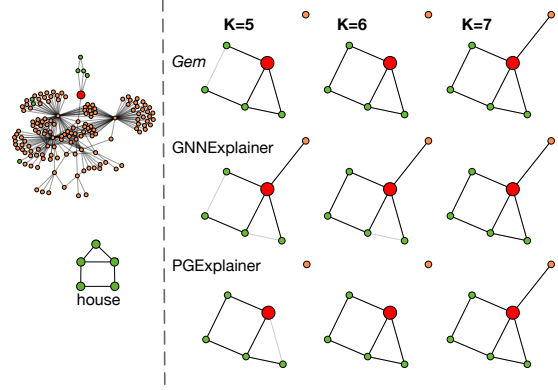


Figure 4. Explanation comparisons on BA-shapes. The “house” in green is the ground-truth motif that determines the node labels. The red node is the target node to be explained (better seen in color).

node (in orange) that is out of the “house” motif. On Tree-cycles, GNNExplainer failed to generate effective explanations when $K < 8$, while *Gem* and PGExplainer achieves favorable accuracy even when $K = 6$.

Note that, for the real-world datasets, there are no explicit motifs (no ground truth motifs) for classification. PGExplainer assumes NO_2 or NH_2 as the motifs for the mutagen graphs and trains an MLP for model explanation with the mutagen graphs including at least one of these two motifs. For fair comparisons, we report the results of PGExplainer following its setting reported in (Luo et al., 2020) and compare them with the results of GNNExplainer and *Gem* when explaining on mutagen graphs, indicated as PGExplainer-0, GNNExplainer-0, and *Gem-0* in Table 2. As GNNExplainer and *Gem* can explain both classes in the dataset, we report the results of explaining the entire test set using GNNExplainer and *Gem* (the 4-5th rows in Table 2⁴). For NCI1, PGExplainer fails to explain this dataset without the motif assumption, therefore, we report the results of GNNExplainer and *Gem*. In general, the results reported successfully verify that our proposed *Gem* can generate explanations that can consistently yield high explanation accuracies over all datasets.

To further check the interpretability of the generated explanations, we report the explanation results for Mutag in Figure 5 ($K = 15$). The first column shows the initial graphs and corresponding probabilities of being classified as “mutagenic” class by the pre-trained GNN, while the other columns report the explanation subgraphs. Associated probabilities belonging to the “mutagenic” class based on the pre-trained GNN are reported below the subgraphs.

In the first two cases (the first two rows), *Gem* can identify the essential components — the aromatic group (carbon

⁴The comparisons with PGExplainer on NCI1 were omitted as it fails to explain on NCI1, indicated as “—” in Table 2 and Table 3.

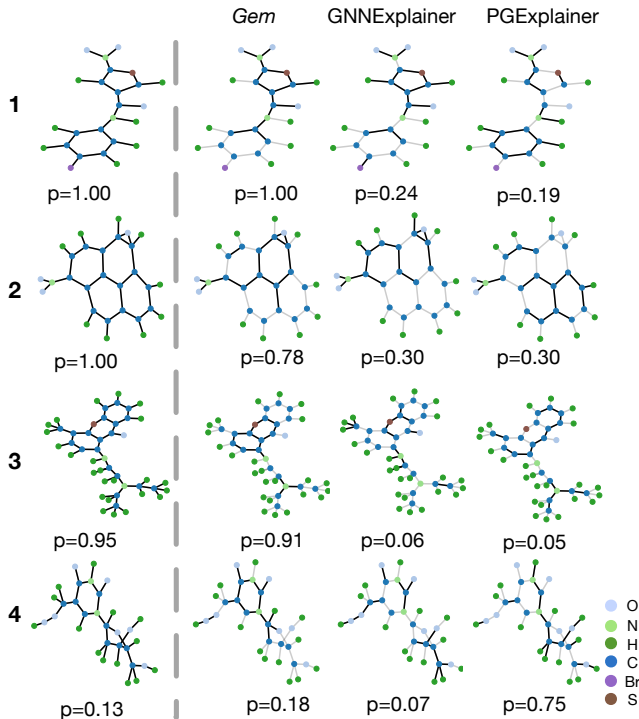


Figure 5. Explanation comparisons on Mutag. The explanation results of different methods are highlighted with black edges, where the gray edges are regarded as unimportant components for the prediction and are discarded by the explainers (better seen in color). The probability under each graph/subgraph denotes the likelihood of being classified into the “mutagenic” class, which is obtained by feeding the associated graph/subgraph into the pre-trained GNN.

ring) and NO_2 — leading to their labels (“mutagenic”). Nevertheless, GNNExplainer either only recognizes the aromatic group (the first row) or NO_2 group (the second row), which is not sufficient to be classified into “mutagenic” class by the pre-trained GNN. PGExplainer focuses on identifying the pre-defined motifs — NO_2 and NH_2 . In the first two rows, we observe that PGExplainer can successfully recognize the NO_2 motifs. However, when its explanation subgraphs are fed into the pre-trained GNN, the probabilities of being classified into the correct class are quite low. In the third row of Figure 5, we report an instance that belongs to the “mutagenic” class without NO_2 or NH_2 motifs. Only *Gem* can recognize the essential components of classifying into “mutagenic.” Note that, a good explainer for a pre-trained GNN should be able to highlight the important components that lead to its predictions for any instances. In the last row of Figure 5, we report an instance that belongs to the “non-mutagenic” class. Though there are no explicit motifs for this class, *Gem* can successfully generate the explanation, which can be recognized as the “non-mutagenic” class by the target GNN, with a probability of 0.82.

To verify the effectiveness of *Gem* in a more statistical view, we measure the resulting change in the pre-trained GNNs’

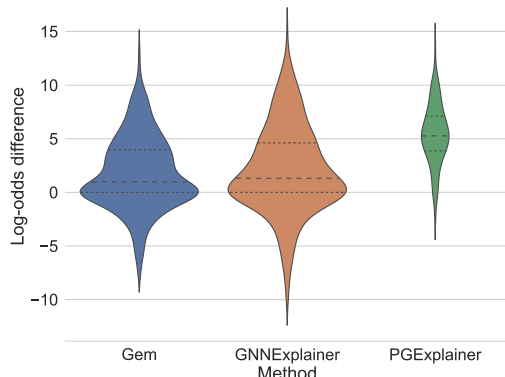


Figure 6. Log-odds difference comparisons on Mutag (more dense distribution around 0 is better).

Table 3. Inference Time per Instance (ms).

DATASETS	BA-SHAPES	TREE-CYCLES	MUTAG	NC11
GNNEXPLAINER	265.2	204.5	257.6	259.8
PGEXPLAINER	6.7	6.5	5.5	—
GEM	0.5	0.5	0.05	0.02

outcome by computing the difference (the initial graph and the explanation subgraph with $K = 15$ for Mutag) in log odds and investigate the distributions over the entire test set. The result on the Mutag dataset is reported in Figure 6 (The definition of log-odds difference and more results on other datasets are provided in Appendix B). We can observe that *Gem* performs consistently better than GNNExplainer and PGExplainer. Specifically, the log-odds difference of *Gem* is more concentrated around 0, which indicates *Gem* can well capture the most relevant subgraphs towards the predictions by the pre-trained GNNs.

Computational performance. PGExplainer and *Gem* can explain unseen instances in the inductive setting. We measure the average inference time for these two methods. As GNNExplainer explains an instance at a time, we measure its average time cost per explanation for comparisons. As reported in Table 3, we can conclude that *Gem* consistently explain faster than the baselines overall. Further experiments on the efficiency evaluation, such as training time comparisons, are provided in Appendix B.

5. Conclusion

In sum, we devised a new framework *Gem* for explaining graph neural networks that use the first principles of Granger causality. *Gem* has several advantages over existing work: it is model-agnostic, compatible with any graph neural network models without any prior assumptions on the graph learning tasks, can generate compact subgraphs, causing the outputs of the pre-trained GNNs very quickly after training. We show that causal interpretability could contribute to explaining and understanding graph neural networks. We believe this could be a fruitful avenue of future research that

helps better understand and design graph neural networks.

References

- Biswal, B. B., Kylen, J. V., and Hyde, J. S. Simultaneous assessment of flow and bold signals in resting-state functional connectivity maps. *NMR in Biomedicine*, 10(4-5): 165–170, 1997.
- Bressler, S. L. and Seth, A. K. Wiener–Granger Causality: a Well Established Methodology. *Neuroimage*, 58(2): 323–329, 2011.
- Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.
- Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.
- Granger, C. W. Investigating Causal Relations by Econometric Models and Cross-Spectral Methods. *Econometrica: Journal of the Econometric Society*, pp. 424–438, 1969.
- Granger, C. W. Testing for Causality: a Personal Viewpoint. *Journal of Economic Dynamics and Control*, 2:329–352, 1980.
- Hamilton, W., Ying, Z., and Leskovec, J. Inductive Representation Learning on Large Graphs. In *Proc. Advances in Neural Information Processing Systems*, 2017.
- Jaume, G., Pati, P., Foncubierta-Rodriguez, A., Feroce, F., Scognamiglio, G., Anniciello, A. M., Thiran, J.-P., Goksel, O., and Gabrani, M. Towards Explainable Graph Representations in Digital Pathology. *arXiv preprint arXiv:2007.00311*, 2020.
- Kipf, T. N. and Welling, M. Variational Graph Auto-Encoders. In *Proc. NIPS Bayesian Deep Learning Workshop*, 2016.
- Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. International Conference on Machine Learning*, 2017.
- Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning Deep Generative Models of Graphs. *arXiv preprint arXiv:1803.03324*, 2018a.
- Li, Y., Zhang, L., and Liu, Z. Multi-Objective De Novo Drug Design with Conditional Graph Generative Model. *Journal of Cheminformatics*, 10(1):33, 2018b.
- Lin, W., Gao, Z., and Li, B. *Guardian*: Evaluating Trust in Online Social Networks with Graph Convolutional Networks. In *Proc. IEEE International Conference on Computer Communications*, 2020.
- Lundberg, S. M. and Lee, S.-I. A Unified Approach to Interpreting Model Predictions. In *Proc. Advances in Neural Information Processing Systems*, pp. 4765–4774, 2017.
- Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., and Zhang, X. Parameterized Explainer for Graph Neural Network. In *Proc. Advances in Neural Information Processing Systems*, 2020.
- Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., and Hoffmann, H. Explainability Methods for Graph Convolutional Neural networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10772–10781, 2019.
- Ribeiro, M. T., Singh, S., and Guestrin, C. Model-Agnostic Interpretability of Machine Learning. In *Proc. ICML Workshop on Human Interpretability in Machine Learning*, 2016.
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.
- Vu, M. N. and Thai, M. T. PGM-Explainer: Probabilistic Graphical Model Explanations for Graph Neural Networks. In *Proc. Advances in Neural Information Processing Systems*, 2020.
- Wale, N., Watson, I. A., and Karypis, G. Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.
- Weisfeiler, B. and Lehman, A. A. A Reduction of a Graph to a Canonical Form and an Algebra Arising during this Reduction. *Nauchno-Tekhnicheskaya Informatsia*, 2(9): 12–16, 1968.
- Wiener, N. The Theory of Prediction. Modern Mathematics for Engineers. *New York*, pp. 165–190, 1956.
- Ying, Z., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. GNNExplainer: Generating Explanations for Graph Neural Networks. In *Proc. Advances in Neural Information Processing Systems*, pp. 9244–9255, 2019.
- Yuan, H., Tang, J., Hu, X., and Ji, S. XGNN: Towards Model-Level Explanations of Graph Neural Networks. In *Proc. SIGKDD. ACM*, 2020.

Zhang, M. and Chen, Y. Link Prediction Based on Graph Neural Networks. In *Proc. Advances in Neural Information Processing Systems*, 2018.

Zitnik, M. and Leskovec, J. Predicting Multicellular Function Through Multi-Layer Tissue Networks. *Bioinformatics*, 33(14):i190–i198, 2017.

Zitnik, M., Agrawal, M., and Leskovec, J. Modeling Polypharmacy Side Effects with Graph Convolutional Networks. *Bioinformatics*, 34(13):i457–i466, 2018.

Generative Causal Explanations for Graph Neural Networks

A. A Unified Framework for Explaining Graph Neural Networks

We adopt the unified framework for explaining neural networks defined in (?). *Additive feature attribution methods* have an explanation model that is a linear function of binary variables. Existing methods unified in this framework provide meaningful explanations for deep models on images. To specifically interpret graph neural networks on graph-structured data, we adapt the definition of the additive property for graph data attribution as follows:

$$\epsilon(A^s) = \psi_0 + \sum \psi_{ij} \times A_{ij}^s \quad (1)$$

where A^s is the adjacency matrix of the explanation, $A_{ij}^s \in \{0, 1\}$ is a binary variable representing the existence of an edge.

In this section, we analyze that GNNExplainer (?), PGExplainer (?) and our explainer, *Gem*, all fall into the framework of additive feature attribution methods. In what follows, we analyze, these methods for explaining GNNs essentially solve the same optimization problem with different approximation methods.

In general, to identify a compact subgraph that is important for the GNN’s prediction, the explanation problem can be formulated as following optimization problem based on mutual information:

$$\begin{aligned} & \max_{G^s \subseteq G^c} \text{MI}(Y, G^s) \\ & = \max_{G^s \subseteq G^c} (H(Y) - H(Y|G = G^s)) \quad (2) \\ & \text{s.t. } |G^s| < K \end{aligned}$$

where K is a constraint on G^s ’s size for a compact explanation.

When treating $G^s \subseteq G^c$ as a graph random variable and with convexity assumption, Eq. (2) can be approximated as solving following optimization problem:

$$\min_{G^s \subseteq G^c} H(Y|G = \mathbb{E}(G^s)) \quad (3)$$

The source code were enclosed in the supplementary material.

To make the optimization tractable, GNNExplainer relaxes the integer constraint on the adjacency matrix and searches for an optimal fractional adjacency matrix. In particular, GNNExplainer factorizes the explanation into a multivariate Bernoulli distribution, formulated as $P(G_s) = \prod_{(i,j) \in G^s} A_{ij}^s$. For explanations on graph structures, an explanation is selected based on the values of entries in the solution A^s . GNNExplainer uses mean-field optimization to solve $\min_{A^s} H(Y|G = \mathbb{E}(G^s))$. Therefore, the weights ψ_{ij} of GNNExplainer under the unified framework are chosen based on the solution A^s .

Gem approximates the solution of the optimization problem from a causal perspective and utilize a graph generative model to generate explanations in an inductive setting. Specifically, *Gem* assumes that the selections of edges from the G^c are conditionally independent to each other. With the assumption of conditional independence, we estimate the edge importance based on the notion of Granger causality. These estimations then are used as the ground-truth to train our explanation model. In particular, we use an auto-encoder architecture utilizing graph convolutions (??), due to its simplicity and expressiveness. Following the instantiation of graph auto-encode (?), we use the reparameterization trick for training, our objective is approximated as $\min_{Z \sim \mathcal{N}(\mu, \sigma)} H(Y|G = G^s)$. The weights ψ_{ij} of *Gem* under the unified framework are then chosen based on the output of the graph generative model.

PGExplainer factorizes the explanation graph as $P(G_s) = \prod_{(i,j) \in G^s} P(A_{ij}^s)$, where $P(A_{ij}^s)$ is instantiated with the Bernoulli distribution $A_{ij}^s \sim \text{Bern}(\theta_{ij})$. PGExplainer utilizes the reparameterization trick and approximate the objective as $\min_{\epsilon \sim \text{Uniform}(0,1)} H(Y|G = G^s)$. To collectively explain multiple instances, PGExplainer adopts a multi-layer perception (MLP) to learn to generate explanations from the target GNN, taking the node embeddings generated from the target GNN as the input. The weights ψ_{ij} of PGExplainer under the unified framework are chosen based on the output of the MLP.

B. Further Implementation Details and More Experimental Results

Datasets. BA-shapes was created with a base Barabasi-Albert (BA) graph containing 300 nodes and 80 five-node “house”-structured network motifs. Tree-cycles was built

with a base 8-level balanced binary tree and 80 six-node cycle motifs. The statistics of four datasets are presented in Table 1. Note that, we report the average number of nodes and the average number of edges over all the graphs for the real-world datasets. Table 2 reports the model accuracy on four datasets, which indicates that the models to be explained are performed reasonably well.

The motif statistics of Mutag are reported in Table 3. We found that there are 32% of non-mutagenic graphs containing at least NO_2 or NH_2 motifs, which indicates that the assumption made by PGExplainer might not be reasonable.

Table 1. Data Statistics of Four Datasets.

DATASETS	BA-SHAPES	TREE-CYCLES	MUTAG	NCI1
#GRAPHS	1	1	4,337	4,110
#NODES	700	871	29	30
#EDGES	4,110	1,950	30	32
#LABELS	4	2	2	2

Table 2. Model Accuracy of Four Datasets (%).

DATASETS	BA-SHAPES	TREE-CYCLES	MUTAG	NCI1
ACCURACY	94.1	97.1	88.5	78.6

Table 3. Motif Statistics on Mutag (Number of Instances Containing Associated Motif).

MOTIF	NO_2	NH_2
MUTAGENIC	596	501
NON-MUTAGENIC	97	251

Unless otherwise stated, all models, including GNN classification models and our explainer, are implemented using PyTorch¹ and trained with Adam optimizer. All the experiments were performed on a NVIDIA GTX 1080 Ti GPU with an Intel Core i7-8700K processor.

For GNN classification models, we use the same setting as GNNExplainer. Specifically, for node classification, we apply three layers of GCNs with output dimensions equal to 20 and perform concatenation to the output of three layers, followed by a linear transformation to obtain the node label. For graph classification, we employ three layers of GCNs with dimensions of 20 and perform global max-pooling to obtain the graph representations. Then a linear transformation layer is applied to obtain the graph label. Figure 1 (a) and 1 (b) are the model architectures for node classification and graph classification, respectively.

¹<https://pytorch.org>

Table 4. Data Splitting for Four Datasets.

DATASETS	#OF TRAINING	#OF TESTING	#OF VALIDATION
BA-SHAPES	300	50	50
TREE-CYCLES	270	45	45
MUTAG	3,468	434	434
NCI1	3,031	410	411

Figure 1 (c) depicts the model architecture of *Gem* for generating explanations. We use mean square error as the loss for training *Gem*. In particular, it was optimized using Adam optimizer with a learning rate of 0.01 and 0.001 for explaining graph and node classification model, respectively. We train at batch size 32 for 100 epochs. Table 4 shows the detailed data splitting for model training, testing, and validation. Note that both classification models and our explanation models use the same data splitting. Figure 2 depicts the distribution of the graph size on two real-world datasets in terms of the number of edges.

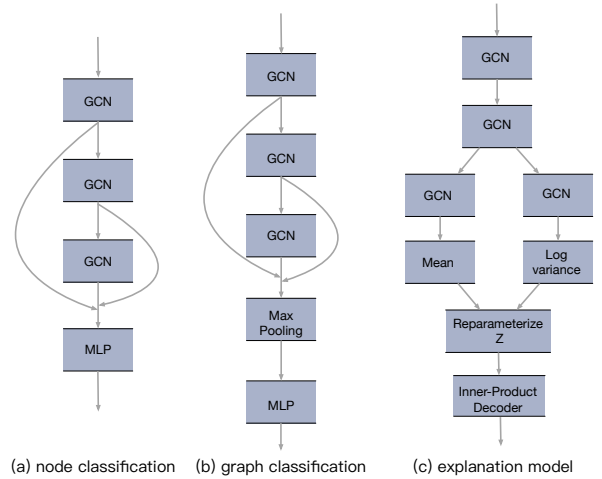


Figure 1. Model architectures.

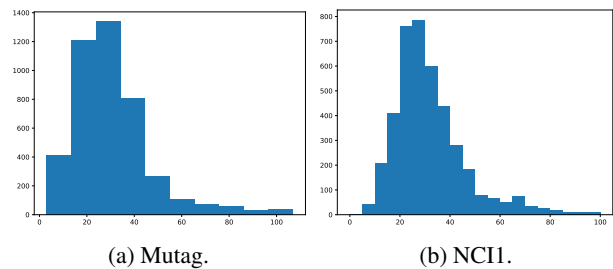


Figure 2. The distribution of the graph size on real-world datasets.

Log-odds difference. We measure the resulting change in the pre-trained GNNs' outcome by computing the difference

in log odds and investigate the distributions over the entire test set. The log-odds difference is formulated as:

$$\Delta \text{log-odds} = \text{log-odds}(p(G^c)) - \text{log-odds}(p(G^s)) \quad (4)$$

where $\text{log-odds}(p) = \log\left(\frac{p}{1-p}\right)$, and $p(G^c)$ and $p(G^s)$ are the outputs of the pre-trained GNN.

The results on the other datasets are reported in Figure 3. We can observe that *Gem* performs consistently better than GNNExplainer and PGExplainer in general. Specifically, the log-odds difference of *Gem* is more concentrated around 0, especially on the BA-shapes dataset, which indicates *Gem* has a better computation accuracy on identifying the most relevant subgraphs towards the predictions by the pre-trained GNNs.

Computational performance. For *Gem*, we measure the time for distilling the “ground-truth” explanation and the time for training the explainer with a varying number of training instances. As for GNNExplainer, we measure the overall time cost of explanations with a varying number of target instances. The training time of PGExplainer and the explanation time of GNNExplainer are measured by reusing the source code released by the authors. Figure 4 report the computational performance for four datasets. We can conclude that *Gem* amortizes the explanation cost by training a graph generator to generate explanations for any given instances and consistently explain faster than the baseline overall.

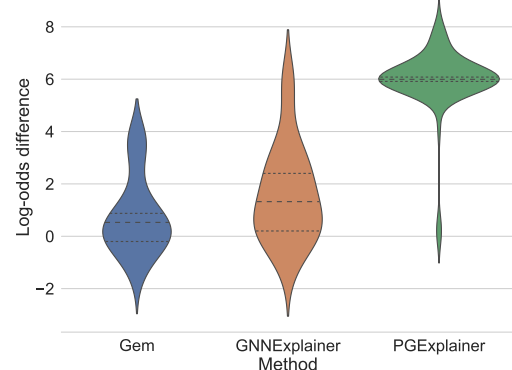
C. Distillation Process

We are given a pre-trained classification model, denoted as $f(\cdot)$, and the training set that is used to train $f(\cdot)$. **Procedure Distillation Process** presents the detailed algorithm of generating the ground-truth explanation G^s for a training instance.

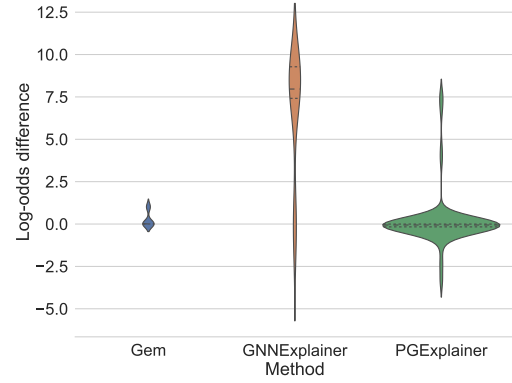
D. Other Related Work

Explanations seek the answers to the questions of “what if” and “why,” which are arguably and inherently causal. The theory of causal inference is one method by which such questions might be answered (?). Recently, causal interpretability has gained increasing attention in explaining machine learning models (???). There are several viable formalisms of causality, such as Granger causality (?), causal Bayesian networks (?), and structural causal models (?).

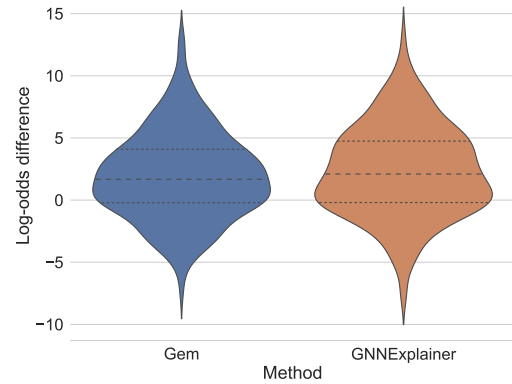
Prior works of this research line usually are designed to explain the importance of each component of a neural network on its prediction. Chattopadhyay *et al.* (?) proposed an attribution method based on the first principles of causality, particularly the Structural Causal Model and $\text{do}(\cdot)$ calculus. Schwab *et al.* (?) framed the explanation task for deep



(a) BA-shapes.



(b) Tree-cycle.



(c) NC11.

Figure 3. Log-odds difference comparisons on other datasets (more dense distribution around 0 is better).

learning models on images as a causal learning task, and proposed a causal explanation model that can learn to estimate the feature importance towards its prediction. Schwab *et al.*’s proposal is built upon the notion of Granger causality.

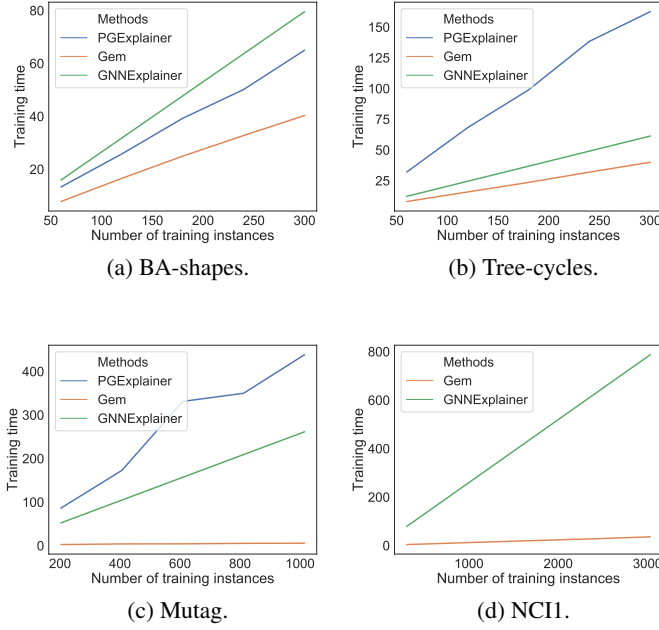


Figure 4. Training time comparisons.

While such methods can provide meaningful explanations for deep models on images, they cannot be directly applied to interpret graph neural networks, due to the inherent property of graph representations.

Algorithm 1 Distillation Process: Distill the top- k most relevant edges for each computation graph

Require: Given the computation graph $G^c = (V^c, A^c, X^c)$.

Ensure: Calculate the model error of the computation graph δ_{G^c} according to Eq. (4)

```

1: for edge  $e_j \in G^c$  do
2:   Calculate the causal contribution  $\Delta_{\delta, e_j}$  according to
   Eq. (1) – Eq. (5).
3: end for
4: Remove edges with the least casual contribution and
   re-calculate the causal contribution of the generated
   subgraph.
5:  $E_{\text{sorted}}^c \leftarrow$  sort the edges in ascending order based on
   the causal contributions.
6: Initialize  $G^s \leftarrow G^c$ 
7: for  $\forall$  edge  $e_j \in E_{\text{sorted}}^c$  do
8:   Calculate the model error of the subgraph, denoted
   as  $\delta_{G^s \setminus \{e_j\}}$  according to Eq. (3) and Eq. (5).
9:    $G^{s0} \leftarrow G^s \setminus \{e_j\}$ 
10:  if  $G^{s0}$  must be connected then
11:     $G^{s0} \leftarrow$  largest component of  $G^{s0}$ 
12:  end if
13:  if  $\delta_{G^{s0}} > \delta_{G^s}$  then
14:     $e_j.\text{weight} \leftarrow \delta_{G^s} - \delta_{G^{s0}}$ 
15:  else
16:     $G^s \leftarrow G^{s0}$ 
17:  end if
18: end for
19: Distill the subgraph with top- $k$  most relevant edges.
20:  $E^s \leftarrow$  edges of  $G^s$ 
21:  $E_{\text{sorted}}^s \leftarrow$  sort  $E^s$  in ascending order by weight
22: for  $\forall$  edge  $e_j \in E_{\text{sorted}}^s$  do
23:   if # of edge in  $G^s \geq k$  then
24:      $G^s \leftarrow G^s \setminus \{e_j\}$ 
25:     if  $G^s$  must be connected then
26:        $G^s \leftarrow$  largest component of  $G^s$ 
27:     end if
28:   end if
29: end for
30: return  $G^s$ 
    
```
