Generative Causal Explanations for Graph Neural Networks

Wanyu Lin 1 2 Hao Lan 2 Baochun Li 2

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, 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× 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, PGEExplainer (Luo et al., 2020) learns a multilayer perceptron (MLP) to explain multiple instances collectively. However, PGEExplainer heavily relies on node embeddings from the target GNN, which may not be obtained without knowing its internal model structure and parameters. Besides, PGEExplainer cannot explain any graph tasks without explicit motifs. Taking MUTAG as an example, PGEExplainer 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, PGEExplainer 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.

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 variable. 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 strategy 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 \( V_i = \{v_{i1}, v_{i2}, \cdots, v_{i|V_i|}\} \) is the node set of graph \( i \). In many applications, nodes are associated with \( d \)-dimensional node features \( X_i = \{x_{i1}, x_{i2}, \cdots, x_{i|V_i|}\}, x_{ij} \in \mathbb{R}^d \). 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 \( y_i \in \mathcal{Y} = \{c_1, c_2, \cdots, c_l\} \), 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 \( v_j \in \mathcal{V} \) of a graph \( G \) is associated with a corresponding node label \( y_j \). Examples of this kind include classifying papers in a citation network, or entities in a social network such as Reddit. The dataset \( \mathcal{D} = \{(v_j, y_j)\}_{j=1}^{|\mathcal{V}|} \) 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).
Figure 1. An illustration of the computation graph (best viewed in color). Node $i$ is the target node to be explained.

e 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 within its $L$-hop graph neighborhood.

Formally, a graph neural network (GNN) can be written as a function $f(\cdot) : G \rightarrow Y$ or $f(\cdot) : V \rightarrow Y$. 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 $\mathcal{L} : y \times \hat{y} \rightarrow s$ that computes a scalar loss $s \in \mathbb{R}$ after comparing the model’s predictive output $\hat{y}$ 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 $f(\cdot)_{exp}$, 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 $\hat{y}$. 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 $L$-hop subgraph; an example of $L = 2$ is 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 $\hat{y}$. 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 $\hat{y}$, efficiently and automatically. We use $G^* \subset G$ 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 $\hat{y}$ using all available information $U$ than if the information apart from variable $x_i$ had been used, we say that $x_i$ Granger-causes $\hat{y}$ (Granger, 1980), denoted by $x_i \rightarrow \hat{y}$.

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

---

1We are aware of the drawbacks of reusing notations. $x_i$ and $\hat{y}$ in this definition represent any random variables for simplicity.
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 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 \( p(\tilde{y} \mid G^c) = p(\tilde{y} \mid G^s) \) holds.

### 3.1. Causal Objective

Given a pre-trained/target GNN and an instance \( G^c \), we use \( \delta_{G^c} \) 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):

\[
\Delta_{s, e_j} = \delta_{G^c \setminus \{e_j\}} - \delta_{G^c} \tag{1}
\]

To calculate \( \delta_{G^c} \) 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:

\[
\hat{y}_{G^c} = f(G^c) \tag{2}
\]

\[
\hat{y}_{G^c \setminus \{e_j\}} = f(G^c \setminus \{e_j\}) \tag{3}
\]

Then we compare the outputs of the target GNN, e.g., \( \hat{y}_{G^c} \) and \( \hat{y}_{G^c \setminus \{e_j\}} \), 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):

\[
\delta_{G^c} = \mathcal{L}(y, \hat{y}_{G^c}) \tag{4}
\]

\[
\delta_{G^c \setminus \{e_j\}} = \mathcal{L}(y, \hat{y}_{G^c \setminus \{e_j\}}) \tag{5}
\]

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, \( \Delta_{s, e_j} \) 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 \( \Delta_{s, e_j} \), 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 causal 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 \(\Delta_{i,j}\) 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 generate an adjacency matrix \(\hat{A}^c\) as an explanation mask. With this mask, we are able to construct a corresponding explanation, 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 specific edge to the prediction of \(G^c\), if that edge exists in the target instance. Formally, the reconstruction process can be formulated as:

\[
Z = \text{GCNs} (A^c, X^c) \tag{6}
\]

\[
\hat{A}^c = \sigma (ZZ^T) \tag{7}
\]

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 \(\hat{y}\)?”, 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.

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 \(\hat{y}\). 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 $O(|E|)$, 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 PG-Explainer (Luo et al., 2020). GNNExplainer explains for a given instance at a time, while PG-Explainer 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 (Veličković 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

---

2 We use the source code released by the authors.
3 The source code can be found in https://github.com/wanyulin/IICML2021-Gem.
Table 1. Explanation Accuracy on Synthetic Datasets (%).

<table>
<thead>
<tr>
<th>K</th>
<th>BA-SHAPE</th>
<th>TREE-CYCLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>64.0</td>
<td>78.1</td>
</tr>
<tr>
<td>6</td>
<td>60.0</td>
<td>76.8</td>
</tr>
<tr>
<td>7</td>
<td>22.5</td>
<td>57.6</td>
</tr>
<tr>
<td>8</td>
<td>66.3</td>
<td>82.1</td>
</tr>
<tr>
<td>9</td>
<td>67.1</td>
<td>74.9</td>
</tr>
<tr>
<td>10</td>
<td>59.3</td>
<td>61.8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>K</th>
<th>MUTAG</th>
<th>NCI1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>64.0</td>
<td>78.1</td>
</tr>
<tr>
<td>6</td>
<td>60.0</td>
<td>76.8</td>
</tr>
<tr>
<td>7</td>
<td>22.5</td>
<td>57.6</td>
</tr>
<tr>
<td>8</td>
<td>66.3</td>
<td>82.1</td>
</tr>
<tr>
<td>9</td>
<td>67.1</td>
<td>74.9</td>
</tr>
</tbody>
</table>

Evaluation metrics. An essential criterion for explanations is that they must be human understandable, 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 visualized 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. \( p(\tilde{y} \mid G^0) \) should be close to \( p(\tilde{y} \mid G^c) \)). 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 PGEExplainer, 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 NC1.

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 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 PGEExplainer 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. PGEExplainer assumes NO or NH2 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 PGEExplainer 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 PGEExplainer-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 NC1, PGEExplainer 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.

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).
Table 3. Inference Time per Instance (ms).

<table>
<thead>
<tr>
<th>DATASETS</th>
<th>BA-shapes</th>
<th>Tree-cycles</th>
<th>MUTAG</th>
<th>NCI1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNNEXPLAINER</td>
<td>265.2</td>
<td>204.5</td>
<td>257.6</td>
<td>259.8</td>
</tr>
<tr>
<td>PGE</td>
<td>9.7</td>
<td>6.5</td>
<td>5.5</td>
<td>0.91</td>
</tr>
<tr>
<td>GEM</td>
<td>9.5</td>
<td>0.5</td>
<td>0.05</td>
<td>0.02</td>
</tr>
</tbody>
</table>
5. 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 (Granger, 1969), 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 do(·) calculus. Schwab et al. (?) framed the explanation task for deep 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. 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.

6. 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.

7. Acknowledgement

This project is supported by the Internal Research Fund at The Hong Kong Polytechnic University P0035763.

References


Weisfeiler, B. and Lehman, A. A. A Reduction of a Graph to a Canonical Form and an Algebra Arising during this Reduction. Nauchno-Technicheskaya Informatsia, 2(9):12–16, 1968.


A. A Unified Framework for Explaining Graph Neural Networks

We adopt the unified framework for explaining neural networks defined in (1). 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:

\[
\epsilon(A^*) = \psi_0 + \sum \psi_{ij} \times A^*_{ij}
\]  

where \(A^*\) is the adjacency matrix of the explanation, \(A^*_{ij} \in \{0, 1\}\) is a binary variable representing the existence of an edge.

In this section, we analyze that GNNExplainer (?) , PG-Explainer (?) 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:

\[
\max_{G^* \subseteq G^c} \text{MI}(Y, G^*) = \max_{G^* \subseteq G^c} (H(Y) - H(Y|G = G^*)) \quad \text{s.t.} \quad |G^*| < K
\]  

where \(K\) is a constraint on \(G^*\)’s size for a compact explanation.

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

\[
\min_{G^* \subseteq G^c} H(Y|G = \mathbb{E}(G^*)).
\]  

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 \(P(G_s) = \Pi_{(i,j) \in G^c} A^*_{ij}\). For explanations on graph structures, an explanation is selected based on the values of entries in the solution \(A^*\). GNNExplainer uses mean-field optimization to solve \(\min_{A^*} H(Y|G = \mathbb{E}(G^*))\). Therefore, the weights \(\psi_{ij}\) of GNNExplainer under the unified framework are chosen based on the solution \(A^*\).

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 \(\min_{G \sim \mathcal{N}(\mu, \sigma)} H(Y|G = G^*)\). The weights \(\psi_{ij}\) of Gem under the unified framework are then chosen based on the output of the graph generative model.

PGExplainer factorizes the explanation graph as \(P(G_s) = \Pi_{(i,j) \in G^c} P(A^*_{ij})\), where \(P(A^*_{ij})\) is instantiated with the Bernoulli distribution \(A^*_{ij} \sim \text{Bern}(\theta_{ij})\). PGExplainer utilizes the reparameterization trick and approximate the objective as \(\min_{G \sim \text{Uniform}(0,1)} H(Y|G = G^*)\). To collectively explain multiple instances, PGExplainer adopts a multilayer 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 \(\psi_{ij}\) 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 NO₂ or NH₂ motifs, which indicates that the assumption made by PGExplainer might not be reasonable.

Table 1. Data Statistics of Four Datasets.

<table>
<thead>
<tr>
<th>DATASETS</th>
<th>BA-SHAPES</th>
<th>TREE-CYCLES</th>
<th>MUTAG</th>
<th>NCI1</th>
</tr>
</thead>
<tbody>
<tr>
<td>#GRAPHS</td>
<td>1</td>
<td>1</td>
<td>4.337</td>
<td>4.110</td>
</tr>
<tr>
<td>#NODES</td>
<td>700</td>
<td>871</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>#EDGES</td>
<td>4,110</td>
<td>1,950</td>
<td>30</td>
<td>32</td>
</tr>
<tr>
<td>#LABELS</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>DATASETS</th>
<th>BA-SHAPES</th>
<th>TREE-CYCLES</th>
<th>MUTAG</th>
<th>NCI1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCURACY</td>
<td>94.1</td>
<td>97.1</td>
<td>88.5</td>
<td>78.6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>MOTIF</th>
<th>NO₂</th>
<th>NH₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTAGENIC</td>
<td>506</td>
<td>501</td>
</tr>
<tr>
<td>NON-MUTAGENIC</td>
<td>97</td>
<td>251</td>
</tr>
</tbody>
</table>

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.

Unless otherwise stated, all models, including GNN classification models and our explainer, are implemented using PyTorch ¹ 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.

Figure 1 (a) Mutag.  (b) NCI1.

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

---

¹https://pytorch.org
in log odds and investigate the distributions over the entire test set. The log-odds difference is formulated as:

$$\Delta \text{log-odds} = \text{log-odds}(p(G^{c})) - \text{log-odds}(p(G^{s}))$$  \hspace{1cm} (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 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. Note that, as Figure 4 shows, it takes 5.67 seconds to train Mutag with 1015 samples, which is quite fast (small but not zero). The distillation process is specifically designed for our framework and only occurs before our explainer training, which can be thought of as a “pre-processing” step. In other words, the time is a one-time cost. The computation time cost of the distillation process is provided in Table 5. 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 $G^{*}$ for a training instance.

The quality of the distilled ground truth provides a basis for our explanation performance. We also provide the performance evaluation of the distillation to see how well the distillation process works. We put the accuracy and associate time cost in Table 5.

*Figure 3. Log-odds difference comparisons on other datasets (more dense distribution around 0 is better).*
Generative Causal Explanations for Graph Neural Networks

![Figure 4. Training time comparisons.](image)

| Table 5. Distillation Performance on Real-World Datasets. |
|-----------------------------|-----------------------------|
| K | MUTAG | NCI1 |
| 15 | 90.7 | 82.0 |
| 20 | 94.2 | 91.7 |
| 25 | 95.2 | 96.8 |
| 30 | 95.8 | 97.1 |
| # of instance | 200 | 300 | 400 | 600 | 1000 | 1200 | 3000 |
| Time(s) | 18 | 30 | 37 | 74 | 186 |

**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 $\delta_{G^c}$ 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. $E^c_{\text{sorted}} \leftarrow$ sort the edges in ascending order based on the causal contributions.
6. Initialize $G^s \leftarrow G^c$
7. for all edge $e_j \in E^c_{\text{sorted}}$ 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. $e_j$.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. $E^* \leftarrow$ edges of $G^s$
21. $E^*_{\text{sorted}} \leftarrow$ sort $E^*$ in ascending order by weight
22. for all edge $e_j \in E^*_{\text{sorted}}$ do
23. if # of edge in $G^s \geq k$ then
24. $G^* \leftarrow G^s \setminus \{e_j\}$
25. if $G^*$ must be connected then
26. $G^* \leftarrow$ largest component of $G^*$
27. end if
28. end if
29. end for
30. return $G^*$