GNN Explainer: A Tool for Post-hoc Explanation of Graph Neural Networks

Rex Ying*, Dylan Bourgeois†,*, Jiaxuan You*, Marinka Zitnik*, Jure Leskovec*

*Stanford University, †EPFL
{rexying,dtsbourg,jiaxuan,marinka,jure}@cs.stanford.edu

ABSTRACT

Graph Neural Networks (GNNs) are a powerful tool for machine learning on graphs. GNNs combine node feature information with the graph structure by using neural networks to pass messages through edges in the graph. However, incorporating both graph structure and feature information leads to complex non-linear models and explaining predictions made by GNNs remains to be a challenging task. Here we propose *GnnExplainer*, a general model-agnostic approach for providing interpretable explanations for predictions of any GNN-based model on any graph-based machine learning task (node and graph classification, link prediction). In order to explain a given node's predicted label, GnnExplainer provides a local interpretation by highlighting relevant features as well as an important subgraph structure by identifying the edges that are most relevant to the prediction. Additionally, the model provides *single-instance explanations* when given a single prediction as well as *multi-instance explanations* that aim to explain predictions for an entire class of instances/nodes. We formalize GnnExplainer as an optimization task that maximizes the mutual information between the prediction of the full model and the prediction of simplified explainer model. We experiment on synthetic as well as real-world data. On synthetic data we demonstrate that our approach is able to highlight relevant topological structures from noisy graphs. We also demonstrate GnnExplainer to provide a better understanding of pre-trained models on real-world tasks. GnnExplainer provides a variety of benefits, from the identification of semantically relevant structures to explain predictions to providing guidance when debugging faulty graph neural network models.

ACM Reference Format:
Rex Ying*, Dylan Bourgeois†,*, Jiaxuan You*, Marinka Zitnik*, Jure Leskovec*. 2019. GNN Explainer: A Tool for Post-hoc Explanation of Graph Neural Networks. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In many real-world applications, such as social, information, chemical, and biological networks, data can be naturally represented as a graph [11, 47, 54]. Machine learning over graph data is challenging because it requires modeling node attribute information as well as the graph's structural information [50, 51].

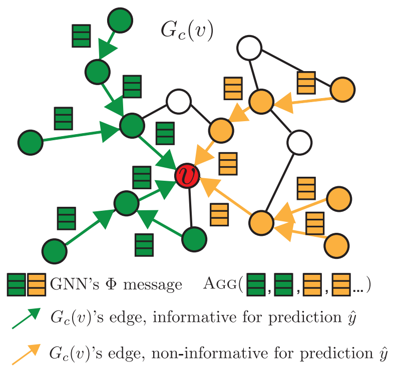
$G_c(v)$
v
GNN's Φ message
Agg(🟩, 🟩, 🟧, 🟧...)
$G_c(v)$'s edge, informative for prediction $\hat{y}$
$G_c(v)$'s edge, non-informative for prediction $\hat{y}$

Figure 1: GNN computation graph G_c for making a prediction $\hat{y}$ at node v. Some edges form important message-passing pathways (green) while others do not (orange). GNN **model aggregates informative as well as non-informative messages to form a prediction at node v. The goal of GnnExplainer is to identify a small set of important features and pathways (green) that are crucial for prediction.**

Graph Neural Networks (GNNs) provide a powerful tool for machine learning on graphs, thanks to their ability to recursively incorporate information/messages from neighboring nodes in the network, naturally capturing the graph structure simultaneously with the node features. GNNs have achieved state of the art performance on a wide range of tasks from node classification [22, 27], link prediction [49], or graph classification [46].

Despite their strengths, GNNs remain notoriously difficult to interpret and their predictions are hard to explain, a concern that is commonly raised for neural models in general. In its most general formulation, interpretability could be understood as "the degree to which a human can understand the cause of a decision" [32] or, in a more utilitarian variation, "the degree to which a human can consistently predict the model's result" [26]. The interpretability of explanations for machine learning models is important because it increases trust in the model, improves model's transparency and the lack of it can limit the adoption of machine learning methods in decision-critical domains, *e.g.*, medical or legal. Ensuring interpretability also contributes to other pertinent criteria such as fairness, privacy, or causality [15], in addition to helping the practitioner debug or even design models more effectively.

While currently no tools to interpret and explain GNNs exist, recent work aimed at explaining general neural networks has taken one of two main routes. The first approach locally approximates models with a simpler surrogate model, which can itself be probed for explanations [29, 34, 37]. The second approach carefully examines models for relevant components, for example, identifying relevant features in the input data [8, 17, 31, 39], or influential input samples [28, 44]. However these existing tools fall short in their inability to incorporate structural information. Since this aspect is crucial to the success of machine learning on graphs, any explanation of GNN's predictions should leverage rich relational information provided by the graph in addition to node feature.

Present work: Desiderata for GNN **explanations.** Here we propose the following desiderata which constitute desirable properties for the explanations of GNNs and their predictions:

Firstly, a good explainer should be able explain the structural graph patterns that the model used to make a prediction. Concretely, the explanation should shed light on the important (sub)graph structures for GNN message-passing that crucially influenced its prediction. This subgraph should ideally be as concise as possible, providing a minimal explanation to facilitate human interpretation without discarding relevant information.

Second is the ability to highlight the importance of not only a given node's features, but also correlations in the features of its neighbouring nodes. In GNNs, a given target node aggregates information from its neighboring nodes, then leverages this representation to make a prediction. The explainer's role is thus not only to identify important target node's features, but also the features of the neighbors that interact through the neural network to influence the prediction at the target node.

Third, the explainer should be able to provide an explanation for a prediction at a single target node (*single-instance explanation*) as well as an explanation for a given set of nodes (for example, all instances of a given class), *multi-instance explanation*. The goals of the two tasks are somewhat different. While single-instance explanation seeks to provide the most concise way to provide reasons for the prediction at a given node, a multi-instance explanation attempts to summarize a number of predictions into a single coherent and consistent explanation that is true over a number of nodes (*e.g.*, all nodes belonging to the same class).

Fourth, the explainer should be able to provide explanations for any machine learning task on graphs: node classification, link prediction, and graph classification.

And fifth, the explainer should be model-agnostic: it needs to apply to and be able to explain any variant of the GNN-family of models.

Present work: GnnExplainer. Here we propose *GnnExplainer*, a tool to explain predictions of GNNs (Figure 1). We operate under assumption that a GNN has already been trained and that now our aim is to explain its prediction(s). GnnExplainer can handle both single- as well as multi-instance explanations, and is agnostic to a particular variant of the GNN model. That is, it can be applied without any modifications to GCNs [27], GraphSAGE [22], PinSage [45], Jumping Knowledge Networks [43], Attention-based networks [40], Line-Graph Neural Networks [10], Gated Graph Neural Networks [30], DiffPool [46], and many more.

In the single-instance explanation scenario, we are given a GNN and its prediction. The goal of GnnExplainer is to identify a small subgraph and a small subset of features that play crucial role in GNN's prediction (Figure 1). We formalize this by maximizing the mutual information between the GNN input and the explanation, which we achieve by formulating a mean field variational approximation, and learning a real-valued *graph mask* which selects the relevant subgraph of the GNN's computation graph. Similarly, GnnExplainer also learns a *feature mask* that masks out unimportant features at all layers of the GNN. GnnExplainer further allows for incorporating explanation conciseness, encouraging discreteness of masks, and including priors such as Laplacian regularization.

In the multi-instance explanation scenario, GnnExplainer provides a single global and consistent explanation for a set of predictions. Here GnnExplainer first chooses a prototype reference explanation using the single-instance explanation, and then learns to align all other GNN subgraph explanations towards this reference point. Given that GnnExplainer single-instance explanations are generally concise, the task is feasible and best alignment can be efficiently computed.

We extensively validate GnnExplainer on synthetic as well as real-world networks. Experiments show that GnnExplainer provides consistent and concise explanations for GNN-based models. We demonstrate our method on both single- as well as multi-instance explanations. Using carefully designed synthetic data with "planted" pathways important for prediction, we show that GnnExplainer can accurately identify important topological structures used by the given GNN. Furthermore, we show GnnExplainer can also robustly identify most important features that influence GNN's prediction the most. We also demonstrate GnnExplainer on two real-world datasets: molecule classification and social network classification. We show that GnnExplainer is able to explain the graph structures that a given GNN model learned to use for prediction. For example, in the molecule classification task GnnExplainer identified important and domain-relevant topological structures, such as NO_2 functional groups and ring structures in molecules. Overall, experiments demonstrate that GnnExplainer provides consistent and concise explanations for GNN-based models for different machine learning tasks on graphs.

2 RELATED WORK

Interpretability and Explanations of Neural Models. To explain and interpret neural networks two main families of models have been proposed. The first family looks to create a simpler proxy model for the main, opaque model. This can be done in a model-agnostic way, usually by learning a locally faithful approximation around the prediction, for example with a linear model [34] or a set of rules, representing sufficient conditions on the prediction [4, 29]. Global distillations of the main model have also been proposed, for instance by reducing deep neural networks to decision trees [37, 52]. However, such approaches often produce intractably large surrogate models, which in practice are uninterpretable.

A second family of models instead aims to highlight relevant aspects of the computation within the provided model. The main approach here is to inspect feature gradients [17] but many other related ideas have also been proposed [38, 39]. When overlayed to the input data, these methods produce a saliency map [48] which reveals important features or raw pixels. However, saliency maps have been shown to be misleading in some instances [2] and prone to issues such as gradient saturation [38, 39]. These issues are exacerbated on discrete inputs such as graph adjacency matrices, since the gradient values can be very large but on a very small interval. This means such approaches are unsuitable for explaining relational structure of a GNN, which is our goal here.

Last, algorithms that find patterns of the input data [28, 44] to identify influential samples are an example of post-hoc interpretability methods. Instead of creating new, inherently interpretable models, thse approaches consider the model as a black box [1, 20] and then probe it for relevant information. Most techniques isolate individual input samples, with some methods allowing for important interactions to be highlighted [18, 23]. However, no work has been done to leverage stronger relational structures like graphs. In contrast, in many cases prediction on graphs can be induced by a complex composition of nodes and their paths. For example, in some tasks an edge could be important only when another alternative path exists to form a cycle, which determines the class of the node. Therefore their joint contribution cannot be modeled well using linear combinations of individual contributions.

Graph Neural Networks. Graph Neural Networks (GNN) [35] obtain node embeddings by recursively propagating information from its neighbours. This framework was later unified into a general Neural Message-Passing scheme [19], and more recently into the relational inductive bias model [6]. For a more detailed review of recent developments we please refer the reader to [6, 21, 50, 51]. Under this model, GNNs have achieved state-of-the-art performance across a variety of tasks, such as node classification [22, 27], link prediction [36, 49], graph clustering [14, 46] or graph classification [12, 16, 46]. These tasks occur in domains where the graph structure is ubiquitous, such as social networks [5], content graphs [45], biology [3], and chemoinformatics [16, 25, 54]. Recent instances of GNN models have been proposed to augment the interpretability properties of GNNs [33, 40, 41]. However, in contrast to our work here, these approaches design a novel GNN architectures explicitly for the purpose of interpretability. Our goal here is different, as we are already given a trained model in the GNN-family and aim to explain its predictions.

3 BACKGROUND

General Formulation of GNNs. A per-layer update of a GNN model involves three key computations [6, 50, 51]:

$$m_{ij}^l = \textsc{Msg}(\mathbf{h}_i^{l-1}, \mathbf{h}_j^{l-1}, r_{ij}) \tag{1}$$

$$M_i^l = \textsc{Agg}(\{m_{ij}^l | v_j \in \mathcal{N}_{v_i}\}) \tag{2}$$

$$\mathbf{h}_i^l = \textsc{Update}(M_i^l, \mathbf{h}_i^{l-1}) \tag{3}$$

First, the model computes neural messages between every pair of nodes (Equation 1). The message for node pair (v_i, v_j) is a function of v_i's and v_j's representations $\mathbf{h}_i^{l-1}$ and $\mathbf{h}_j^{l-1}$ in the previous layer, and the relation r_{ij} between the nodes. In different GNN models, r_{ij} can indicate, for example, an edge type [54], an edge weight [45], or a shortest path from v_i to v_j. Second, for each node v_i, the GNN model aggregates messages from v_i's neighborhood $\mathcal{N}_{v_i}$ and calculates an aggregated message M_i (Equation 2). Popular aggregation methods include mean/max pooling [22], permutation invariant neural networks, and Recurrent Neural Networks [42].

Here, the definition of a node neighborhood $\mathcal{N}$ is crucial, as it can affect the performance and scalability of the GNN model. Popular definitions include immediate network neighborhoods [27], multi-hop neighborhoods [43], sampling-based neighborhoods [7, 9, 22, 24], or PageRank-based neighborhoods [45]. Finally, the GNN model takes the aggregated message M_i^l along with v_i's representation $\mathbf{h}_i^{l-1}$ from the previous layer, and it non-linearly transforms them to obtain v_i's representation $\mathbf{h}_i^l$ at layer l (Equation 3).

These computations are performed for every layer, with the initial condition $\mathbf{h}_i^0 = \mathbf{x}_i$, where $\mathbf{x}_i$ is initial node feature vector. The final embedding for node v_i after L layers of computation is $\mathbf{z}_i = \mathbf{h}_i^L$. To predict the label for node i, we pass the node embedding $\mathbf{z}_i$ through a standard classification network, such as a multi-layer perceptron (MLP) with a softmax output layer. Our GnnExplainer is a model-agnostic framework that provides explanations for any GNN that can be formulated in terms of Equations 1–3.

Computation Graphs. In the context of a neural network architecture, the information that a GNN relies on for computing $\mathbf{z}_i$ is completely determined by v_i's computation graph, which is defined by Equation 2. The GNN uses that computation graph to generate v_i's representation $\mathbf{z}_i$ (Equations 1 and 3) [50, 51]. Importantly, the structure of the computation graph is different for each node v_i, and depends on how the neighborhood $\mathcal{N}_{v_i}$ is defined. Let $G_c(v_i)$ denote the computation graph used by the GNN to compute representation $\mathbf{z}_i$ of node v_i. The $G_c(v_i)$ can be obtained by performing a graph traversal of arbitrary depth L, *e.g.*, a Breadth-First Search (BFS), using $\mathcal{N}_{v_i}$ as the neighborhood definition. We further define $A_c(v_i)$ as the adjacency matrix corresponding to the computation graph $G_c(v_i)$.

Following this definition, in Graph Convolutional Networks (GCNs) [27], the computation graph $G_c(v_i)$ is simply an L-hop neighborhood of v_i in the input graph G. However, in other GNN models, such as Jumping Knowledge Networks [43], attention-based networks [40], and Line-Graph NNs [10], the computation graph $G_c(v_i)$ will be different from the exhaustive L-hop neighborhood.

4 GNN EXPLAINER: FORMULATION

4.1 Desired Features of GNN Explainer

An essential criterion for explanations is that they must be interpretable, *i.e.*, provide a qualitative understanding of the relationship between the input nodes and the prediction. Such a requirement implies that explanations should be easy to understand while remaining exhaustive. This means that a GNN explainer should take into account both the structure of the underlying graph G as well as the associated features, if they are available. More specifically, the following aspects of a GNN model should be incorporated into the design of explanations:

(1) **Local edge fidelity:** The explanation needs to identify the relational inductive bias used by the GNN. This means it needs to identify which message-passing edges in the computation graph G_c represent essential relationships for a prediction.
(2) **Local node fidelity:** The explanation should not only incorporate the specific node v_i's features $\mathbf{x}_i$, but also a number of

important features from the set $X_c(v_i)$ of features from other nodes present in computation graph $G_c(v_i)$.

(3) **Single-instance vs. multi-instance explanations:** The explanation should summarize where in a graph G the GNN model looks for evidence for its prediction and identify the subgraph of G most responsible for a given prediction. The GNN explainer should also be able to provide an explanation for a set of predictions, *e.g.*, by capturing a distribution of computation graphs for all nodes that belong to the same class.

(4) **Any GNN model:** A GNN explainer should be able to explain *any* model in GNN-family and be model-agostic, *i.e.*, treat GNN as a black box without requiring modifications of neural architecture or re-training.

(5) **Any prediction task on graphs:** A GNN explainer should be applicable to *any* machine learning task on graphs: node classification, link prediction, and graph classification.

4.2 GNN Explainer Problem Formulation

Next, we describe the setting of explaining GNN predictions. Without loss of generality, we consider the problem of explaining predictions for a node classification task. Let G denote a graph on nodes V and edges E, which is associated with a set of d node features $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, $x_i \in \mathbb{R}^d$, and a label function f on nodes $f : V \mapsto \{1, \dots, C\}$ that maps every node in V to one of the C classes. The GNN model Φ is optimized for all nodes in the training set, such that Φ can be used to approximate f on nodes in the test set.

Given a trained GNN model Φ and a predicted label (*i.e.*, single-instance explanation) or a set of predicted labels (*i.e.*, multi-instance explanations), a GNN explainer will generate an explanation by returning edge connectivity patterns and node features that provide insights on what the model has learned. To that end, the GNN explainer relies on the general GNN formulation (Section 5) and only accesses Φ in terms of forward and backward propagation, which means that it is agnostic to the specific type of GNN.

5 GNN EXPLAINER: METHOD

Next we describe our approach *GnnExplainer* that satisfies desiderata 1–5 from Section 4.1. We start by describing single-instance explanations (Section 5.1) that consider structural graph patterns and rich node features (Section 5.2) and proceed with multi-instance explanations based on the idea of class prototypes (Section 5.3). We conclude with a discussion on how GnnExplainer can be used for any machine learning task on graphs including link prediction and graph classification (Section 5.4) and we give details on GnnExplainer training (Section 5.5).

Overview of GnnExplainer. GnnExplainer is a tool for post-hoc interpretation of predictions generated by a pre-trained GNN. It is formulated in an optimization framework. The key insight here is that information used by GNN to compute prediction $\hat{y}$ is completely described by its computation graph $G_c(v_i)$ and the associated feature set $X_c(v_i) = \{x_j | v_j \in G_c(v_i)\}$ (Section 3); that is, GNN prediction is given as $\hat{y} = \Phi(G_c(v_i), X_c(v_i))$. As a result, to construct an explanation for $\hat{y}$ we only need to consider structural information present in $G_c(v_i)$ and node feature information present in $X_c(v_i)$.

GnnExplainer identifies a subgraph of the computation graph $G_c(v_i)$ and a subset of node features that are most influential for the model's prediction. Importantly, influential edges and node features in $G_c(v_i)$ are jointly determined. In the case of a set of predictions, GnnExplainer aggregates, in a class-specific way, individual explanations in the set. This way, individual explanations are summarized into a prototype computation graph.

5.1 Single-Instance Explanations

A GNN learns a distribution $P_\Phi(Y|G_c, X_c)$, where Y is a random variable representing output class labels $\{1, \dots, C\}$, indicating the probability of a node belonging to each of the C classes, as predicted by the GNN. Our goal is to identify a subgraph $G_S \subseteq G_c(v_i)$ with the associated features $X_S = \{\mathbf{x}_i | i \in G_S\}$ which are important for GNN's prediction of v_i's label. We formalize the notion of importance using mutual information MI as follows:

$$\max_{G_S} MI(Y, (G_S, X_S)) = H(Y) - H(Y|G = G_S, X = X_S). \quad (4)$$

In addition to the constraints $G_S \subseteq G_c(v_i)$, $X_S = \{\mathbf{x}_i | i \in G_S\}$, we also impose the size constraint $|G_S| \leq k$, which specifies that the subgraph for explanation should be concise and its size should not exceed k nodes. Since Φ is fixed, the entropy term $H(Y)$ in Equation 4 is also fixed. As a result, maximizing mutual information between predicted label distribution Y and explanation (G_S, X_S) is equivalent to minimization of conditional entropy $H(Y|G = G_S, X = X_S)$, which can be expressed as follows:

$$H(Y|G{=}G_S, X{=}X_S) = -\mathbb{E}_{Y|G_S, X_S}\left[\log P_\Phi(Y|G{=}G_S, X{=}X_S)\right] \quad (5)$$

In other words, explanation for v_i's prediction $\hat{y}$ is a subgraph G_S and the associated features X_S that minimize the uncertainty of the GNN Φ when the neural message-passing is limited to G_S. This means that the explanation G_S is given by a subgraph of G_c that maximizes probability of predicted class $\hat{y}$.

Intuitively, one can motivate the process of generating explanations as follows. Some edges in v_i's computation graph G_c form important message-passing pathways, which allow useful node information to be propagated across G_c and aggregated at v_i for prediction. However, some edges in G_c might not be informative for prediction. In particular, note that the GNN is trained on a number of examples/nodes and our aim is to explain the prediction of a particular node v_i. Thus, not all parts of the GNN Φ might be relevant or important for prediction at v_i. Hence messages passed along those edges do not provide any useful information and might even decrease GNN's performance. Indeed, the aggregator in Equation 2 needs to aggregate such non-informative messages together with useful messages, which can dilute the overall signal accumulated from v_i's neighborhood. In practice, many datasets and tasks support such motivation as illustrated in Figure 1. The objective of GnnExplainer thus aims to remove edges from v_i's computation graph that are not informative for the v_i's prediction. In other words, the objective is in a sense denoising the computation graph to keep at most k edges that have the highest mutual information between the computation graph and the GNN's prediction.

Model Optimization. Direct optimization of the objective in Equation 4 is not tractable as there are exponentially many discrete structures $G_S \subseteq G_c(v_i)$. We therefore use an approximation similar to soft-attention. Rather than optimizing G_S's adjacency matrix $A_S \in \{0,1\}^{n \times n}$ as a discrete structure where n is the size of the computation graph G_c, we allow a fractional adjacency matrix[1] for subgraph G_S: $A_S \in [0,1]^{n \times n}$. To enforce the subgraph constraint, the relaxed adjacency A_S is subject to the constraint that $A_S[i,j] \leq A_c[i,j], \forall i,j = 1,\ldots,n$. Such relaxation allows gradient descent to be performed on G_S.

The relaxed G_S can be interpreted as a variational approximation of subgraph distributions of $G_c(v_i)$ using the mean field assumption. If we treat $G_S \sim \mathcal{G}$ as a random graph variable, then objective in Equation 5 becomes:

$$\min_{\mathcal{G}} \mathbb{E}_{G_S \sim \mathcal{G}} H(Y|G = G_S, X = X_S) \qquad (6)$$

When Φ is convex, by Jensen's inequality, Equation 6 has the upper bound, which we can optimize:

$$\min_{\mathcal{G}} H(Y|G = \mathbb{E}_{\mathcal{G}}[G_S], X = X_S) \qquad (7)$$

Using mean field variational approximation, we decompose $\mathcal{G}$ into multi-variate Bernoulli distribution $P_{\mathcal{G}}(G_S) = \prod_{(i,j)\in G_c(v_i)} A_{S\,ij}$. Instead of optimizing $\mathbb{E}_{\mathcal{G}}[G_S]$, we optimize the expectation with respect to the mean-field approximation, namely A_S, whose (i,j) entry represents the expectation of Bernoulli distribution on whether edge e_{ij} exists. We found that despite non-convexity of GNN this procedure often converges to a good local minimum with the help of regularization that encourages discreteness [46]. To optimize the relaxed $A_S \in [0,1]^{n \times n}$, subject to the constraint $A_S[i,j] \leq A_c[i,j], \forall i,j = 1,\ldots,n$, we design a mask $M \in \mathbb{R}^{n \times n}$ as parameters to optimize. A_S can be computed by taking element-wise multiplication of mask $\sigma(M)$ with A_c, and always obey the continuous relaxation of the subgraph constraint. Here σ denotes the sigmoid function that maps M to $[0,1]^{n \times n}$, but other activation functions that squash the input can be used.

Concretely, we summarize the objective of learning a single-instance explanation with the following optimization problem:

$$\min_M -\mathbb{E}_{Y|A_S \odot \sigma(M), X_S} [\log P_\Phi(Y|G = A_S \odot \sigma(M), X = X_S)] \quad (8)$$

$$= \min_M - \sum_{c=1}^{C} \mathbb{1}[y = c] \log P_\Phi(Y = y|G = A_S \odot \sigma(M), X = X_S). \quad (9)$$

Here $\odot$ denotes element-wise multiplication. In the forward propagation, $P_\Phi(Y|G = A_S \odot M, X = X_S)$ is computed by replacing adjacency matrix A_c with $A_S \odot M$ in the aggregation step 2, meaning that the mask controls how much information is passed along each edge of G_c. An alternative to continuous relaxation is Gumbel Softmax, which allows learning of discrete adjacency A_S. We empirically compare both approaches in Section 6.

After mask M is learned, we use thresholding to remove low values in M. Finally, top k edges give us an explanation G_S for GNN's prediction at node v_i.

5.2 Including Node Features in Explanations

Node feature information plays an important role in computing messages between nodes. The explanations therefore should take into account the feature information. However, in a GNN node feature information is propagated over multiple layers, which means important features could differ between the nodes and the features might interact in non-linear ways across the multi-layer GNN. We tackle this problem by learning a feature mask (one for the entire GNN or one per node in the GNN computation graph). The mask then acts as a feature selector and also prevents the feature information from "leaking" through the layers of the GNN.

[1]For typed edges, we define $G_S \in [0,1]^{C_e \times n \times n}$, where C_e is the number of classes.

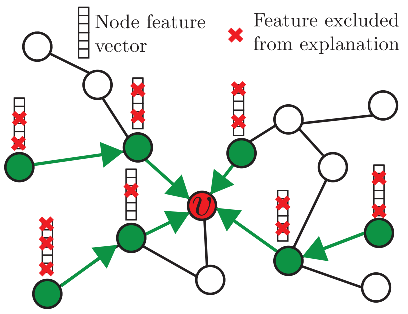

Figure 2: For v's explanation G_S (in green), GNNExplainer identifies what feature dimensions of G_S's nodes are essential for prediction at v by learning a node feature mask M_T.

We explicitly consider what feature dimensions are essential for message passing, *i.e.* feature selection for nodes in G_S. In addition to optimizing Equation 5 over G_S, we also perform optimization that selects important features in X_S. Two approaches are possible: (1) we can optimize the essential dimensions for each node's feature $x_i \in G_S$ separately, or (2) we can optimize a single set of important dimensions for all nodes' features involved in the GNN computation graph. The former allows for finer granularity of feature importance but provides more complex explanations, while the latter is computationally more efficient and provides more concise explanations. Here we adopt the latter approach, but the following analysis can easily extend to the former one as well.

For any given G_S (*i.e.*, an identified explanatory subgraph of the computational graph), instead of defining $X_S = \{x_i | v_i \in G_S\}$, we define X_S^T as a subset of features (over all nodes in the selected computation graph G_S) selected by the binary feature selector $T \in \{0,1\}^D$ (Figure 2):

$$X_S^T = \{x_i^T | v_i \in S\}, x_i^T = [x_{i,t_1}, \ldots, x_{i,t_j}], \text{where } t_j \in T, \qquad (10)$$

Therefore, GNNExplainer jointly optimizes for subgraph G_S and the subset of feature dimensions T for each node in G_S, and does so in an end-to-end way. The inclusion of node features changes mutual information from Equation 4 in the following manner:

$$\max_{G_S, T} MI(Y, (G_S, T)) = H(Y) - H(Y|G = G_S, X = X_S^T). \qquad (11)$$

Model Optimization. Due to the intractability of combinatorial optimization of the binary feature selector T, we perform continuous relaxation. We approximate X_S^T by $X_S \odot M_T$, where M_T is a feature selection mask matrix and needs to be estimated/learned. However, consider the case where some feature x_i takes value 0 but that value of zero provides useful information for GNN's prediction. In this case, the feature x_i should be retained and not be masked out, but our approximation proposed above would ignore it and filter the feature x_i out. However, if $x_i \neq 0$ and also being important for prediction, it would not be filtered out. To resolve this issue with important zero-valued features we marginalize over all possible choices of features in $\{0,1\}^D \backslash T$. In practice, we use Monte Carlo estimate of the marginals of these missing features [53]:

$$P_\Phi(Y|G_S, X_S^T) = \sum_{x_i \in D\backslash T} P(x_i) P_\Phi(Y|G = G_S, X = \{X_S^T, x_i\}). \quad (12)$$

Here, $X = \{X_S^T, x_i\}$ is a random variable due to the missing features x_i, whose distribution is approximated with the empirical distribution from data. However, in our setting, rather than computing an importance score for each feature dimension, we instead want to incorporate features into the objective in Equation 11. The key challenge here is how to perform backpropagation through a random variable X, which we address by using a reparametrization trick for X:

$$X = Z + (X_S - Z) \odot M_T, \quad s.t. \sum_j M_{Tj} \leq k \quad (13)$$

Here X_S is still defined as the deterministic node features $\{x_i | v_i \in G_S\}$. Such a formulation allows the backpropagation of gradients from the objective to the feature mask M_T.

During training, a random variable Z of dimension n-by-d is sampled from the empirical marginal distribution of all nodes $x_i \in X_S$. To see how the reparametrization allows optimization of feature mask, consider the scenario when i-th dimension is not important; that is, any sample of Z from the empirical marginal distribution does not affect GNN's prediction. In this case, the constraint will pull the corresponding mask value M_i towards 0, and as a result, the random variable Z is fed into the GNN. On the other hand, if i-th dimension is very important, the random sample of Z will degrade GNN's performance, resulting in a larger value of objective function. Therefore, to minimize the loss, the mask value M_i will go towards 1, when X becomes the deterministic X_S, which are the actual node features.

Imposing Additional Constraints on Explanations. The advantage of GnnExplainer's optimization framework is the flexibility of its objective function. Namely, it allows for the inclusion of additional constraints describing the structure of explanations. The following aspects are considered as side-objectives and are optimized jointly with the objective given in Equation 9.

(1) **Explanation size:** Explanations that are too complex often do not give useful insights. We regularize the size of explanations $|G_S| = \sum_{i,j} A_{S\,i,j} \leq k$. In the case of continuous relaxation, we instead compute its size as $\sum_{i,j} \sigma(M_{i,j})$. Practically, we use a corresponding regularization term, with a user-specified value.
(2) **Mask discreteness:** We encourage discrete masks M (*i.e.*, masks of structural graph patterns) and M_T (*i.e.*, masks of node features), which we optimize for by regularizing the element-wise entropy of M and M_T. For example, the entropy of M is computed by:

$$H(M) = -\sum_{1 \leq i,j \leq n} \left(M_{i,j} \log M_{i,j} + (1 - M_{i,j}) \log(1 - M_{i,j})\right) \quad (14)$$

(3) **Application-specific prior knowledge:** We include application-specific prior knowledge using Laplacian regularization. For example, in graphs that exhibit homophily, informative messages flow between nodes with similar labels. We can encourage such prior via a quadratic form regularization $f^T L_S f$, where f is the label function on each node, and $L_S = D_S - A_S$ is the graph Laplacian corresponding to adjacency matrix A_S.
(4) **Valid** GNN **computation graph:** The explanation of v_i's prediction should be a subgraph of v_i's computation graph that is itself a valid computation graph. In particular, the explanation needs to allow message passing towards the center node v_i in order to generate node prediction $\hat{y}$. In practice, we find that this property is naturally achieved in GnnExplainer without explicit regularization. This is because GnnExplainer jointly optimizes the mask across all edges. It encourages finding computation graphs that can pass information to the center node v_i and discourages disconnected edges (even if a disconnected edge is important for message-passing, it will not be selected if it cannot influence the center node).

Side-objectives 1-3 are included in GnnExplainer's objective function in the form of regularization terms.

5.3 Multi-Instance Explanations

The output of our single-instance GnnExplainer indicates for each node, which structure is important for prediction of its label. To obtain a deeper understanding of "why is a given set of nodes classified with label y", we want to also obtain a global explanation of the class, which can shed light on how the identified structure for a given node is related to a prototypical structure unique for its label. To this end, we propose an alignment-based multi-instance GnnExplainer.

For any given class, we first choose a reference node. Intuitively, this node should be a prototypical node for the class. Such node can be found by computing the mean of the embeddings of all nodes in the class, and choose the node whose embedding is the closest to the mean. Alternatively, if one has prior knowledge about the important computation subgraph, one can choose one which matches most to the prior knowledge.

Given the reference node for class c, v_c, and its associated important computation subgraph $G_S(v_c)$, we align each of the identified computation subgraphs for all nodes in class c to the reference $G_S(v_c)$. Utilizing the idea in the context of differentiable pooling [46], we use the a relaxed alignment matrix to find correspondence between nodes in an computation subgraph $G_S(v)$ and nodes in the reference computation subgraph $G_S(v_c)$. Let A_v and X_v be the adjacency matrix and the associated feature matrix of the to-be-aligned computation subgraph. Similarly let A^* be the adjacency matrix and associated feature matrix of the reference computation subgraph. Then we optimize the relaxed alignment matrix $P \in \mathbb{R}^{n_v \times n^*}$, where n_v is the number of nodes in $G_S(v)$, and n^* is the number of nodes in $G_S(v_c)$ as follows:

$$\min_P |P^T A_v P - A^*| + |P^T X_v - X^*|. \quad (15)$$

The first term in Equation 15 specifies that after alignment, the aligned adjacency for $G_S(v)$ should be as close to A^* as possible. The second term Equation 15 specifies that the features should for the aligned nodes should also be close.

In practice, it is often non-trivial for the relaxed graph matching to find a good optimum for matching 2 large graphs. However, thanks to the single-instance explainer, which produces concise subgraphs for important message-passing, a matching that is close to the best alignment can be efficiently computed.

Prototype by Alignment. We align the adjacency matrices of all nodes in class c, such that they are aligned with respect to the ordering defined by the reference adjacency matrix. We then take the average of each edge to generate a prototype $A_{\text{proto}} = \frac{1}{n}\sum_i A_i$, where A_i is the aligned adjacency matrix representing explanation for i-th node in class c. Prototype A_{proto} allows users to gain insights into structural graph patterns shared between nodes that belong to the same class. Users can then investigate a particular node by comparing its explanation to the class prototype.

5.4 Link Prediction and Graph Classification

GnnExplainer can provide explanations for any machine learning task on graphs. In particular, for link prediction and graph classification, we modify the GnnExplainer objective function in Equation 8 as follows.

When predicting a link between nodes v_i and v_j, Y represents a distribution of existence of links. Quantity P_Φ in Equation 8 is thus:

$$P_\Phi(Y|G_1 = A_{S_i} \odot M_1, G_2 = A_{S_j} \odot M_2, X = X_S), \qquad (16)$$

meaning that GnnExplainer learns two masks that identify important computation subgraphs for both nodes involved in the predicted link. When classifying graphs, Y represents a distribution of label classes for graphs. This means that quantity A_S in GnnExplainer objective function is no longer an adjacency matrix for the computation graph of a particular node v_i. Instead, it is the union of computation graphs across all nodes in the graph whose graph-level label we want to explain. Importantly, optimization procedure for explaining link prediction and graph classification is the same as that for node classification.

5.5 Efficiency of GnnExplainer

The number of parameters in GnnExplainer's optimization depends on the size of adjacency matrix of the computation graph $A_c(v_i)$, which is equal to the size of the mask M, which needs to be optimized. Most GNNs on large graphs use 2-3 hop neighborhoods [27], and thus computation graphs are much smaller than the the entire graph, allowing explanations to be efficiently computed. Very large graph systems, such as PinSage [45], use sampling- or PageRank-based neighborhoods to define computation graphs. In these cases, computation graphs are also relatively small, compared to the size of K-hop neighborhoods. In GNN models like relational reasoning [6], a computation graph for each node is the entire graph, but those models typically use attention mechanism [40]. As such, one can prune edges with low-attention weights and effectively reduce the size of computation graphs, thus allowing GnnExplainer to be efficient.

6 EXPERIMENTS

6.1 Experimental setup

Datasets. We first construct 4 synthetic datasets to assess the approach for different aspects detailed in the desiderata in a controlled environment.

(1) **BA-Shapes.** Given a base Barabási-Albert (BA) graph of size 300, a set of 80 five-node house-structure network motifs are attached to random nodes. The resulting graph is further perturbed by uniformly randomly adding $0.1n$ edges, where n is the size of the graph. In order to isolate structural information gathered by the GNN model, nodes are equipped with a set of constant features. Each node is assigned a label based on its role in the motif. There are 4 label classes.

(2) **BA-Community.** A union of 2 BA-Shapes graphs. These two graphs are then randomly connected, forming two bridged communities. Nodes have normally distributed feature vectors. To test feature explanation together with structure explanation, for dimension 2, The mean of the feature distribution for different communities differs by 1 standard deviation. The rest of the unimportant feature dimensions are drawn from the same distribution for all nodes in both communities. The label of each node is based on both its structural role and its community; hence there are 8 label classes.

(3) **Tree-Cycles.** We construct a balanced binary tree of height 8. A set of 80 six-node cycle motifs are attached the same way as BA-Shapes. This tests the model's ability to reach for multi-hop structural information in the assessment of a prediction. This task is more challenging since the degree distributions for nodes in the tree and the motif are similar.

(4) **Tree-Grid.** Same as Tree-Cycles except that a more complex 3-by-3 grid is attached to the tree instead of cycles.

In addition, we explore how GnnExplainer could provide insights to GNN models in real-world setting, in the area of molecular biology and social network analysis. The following **graph classification** datasets are considered:

(1) **Mutag.** A common benchmark of 4337 molecule graphs labeled according to their mutagenic effect on the Gram-negative bacterium *Salmonella typhimurium* [13].

(2) **Reddit-Binary.** A collection of 2000 graphs, each corresponding to an online discussion thread on Reddit. Nodes are users, connected if at least one responded to another's comment. These threads have different user interaction patterns: *r/IAmA* and *r/AskReddit* foster **Question-Answer** patterns, while *r/TrollXChromosomes* and *r/atheism* are **discussion-based** communities. The goal is to classify these communities into according to interaction types.

Baselines. Many existing explainability work cannot be directly applied on graphs. We consider the following baselines. All code for GnnExplainer and the baselines, and their parameter settings will be made public.

(1) **Grad.** Gradient-based method. We compute the gradient of model loss with respect to adjacency matrix and node features to be classified, and pick edges that have the highest absolute gradients, similar to saliency map. In graph classification, the gradient with respect to node features are averaged across nodes. This method allows explaining important subgraphs as well as features.

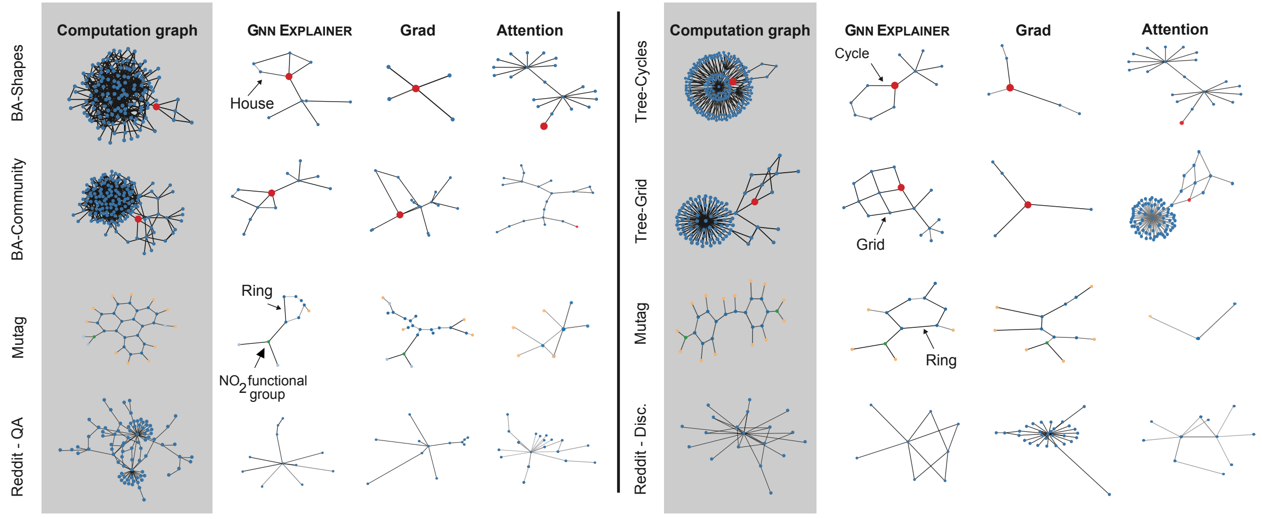

Figure 3: Examples of single-instance important subgraphs. The red node is the explained node.

(2) **ATT.** Graph Attention Network (GAT) provides explanation by attention weights on edges, serving as a proxy for edge importance in the computation graph [40]. However, this method cannot be directly estimate feature importance, and cannot be used as post-hoc analysis for other GNN models.

6.2 Single-instance prediction

To extract the subgraph, we first compute the importance weights on edges (gradients for GRAD baseline, attention weights for ATT baseline, and masked adjacency for GNNEXPLAINER).

A threshold is used to remove low-weight edges. For GNNEXPLAINER, we use 0.2 threshold throughout all experiments. However, as we find that the baselines are sensitive to thresholds, we tune the thresholds for each individual task. We choose the threshold such that the number of edges shown for each baseline matches our prior knowledge of the size of the explanation (cycle, grid *etc.*).

First, we show the explainer local edge fidelity (single-instance explanations) through its ability to highlight relevant message-passing pathways. In a strictly topology-based prediction task such BA-SHAPES and TREE-CYCLES, GNNEXPLAINER correctly identifies the motifs required for accurate classification (see top 2 rows of Figure 3). In these synthetic node classification tasks, the red node indicates the node to be classified. As seen in the figure, GNNEXPLAINER looks for a concise subgraph of the computation graph that best explains the prediction for a single queried node. All of house, cycle and tree motifs are identified in almost all instances.

In MUTAG, the color indicates the feature of the node (hydrogen, carbon etc.). In MUTAG, the carbon ring and the functional groups NH_2 and NO_2, that are known to be mutagenic, are correctly identified [13]. In the REDDIT-BINARY dataset, we observe that discussion patterns (4-th row to the right) have a high-degree center node, while the QA patters have more evenly distributed nodes. SUch pattern is identified by GNNEXPLAINER, which uses a star graph as explanation for QA graphs.

On the other hand, Gradient- and Attention-based methods often gives incorrect or incomplete patterns as explanation. For example, they miss the cycles, and the more complex grid structure. In the case of GRAD method for TREE-CYCLES and TREE-GRID, the gradients are very sparse, and even with threshold close to 0, very few edges can be identified. In general gradient and attention baselines are unstable and hard to tune for good explanation. Furthermore, although the importance weights in ATT baseline can be interpreted as importance in message passing, the graph attention weights are shared for all nodes in the graph.

In addition, both gradient and the attention baselines suffer from disconnectedness. Low threshold for pruning results in highly disconnected components in the case of node classification. However, we know a priori that for node classification, only the connected component containing the node to be classified can influence the node classification, and hence we only plot the connected component containing the node to be classified. While the objective and regularization in GNNEXPLAINER effectively ensure good properties of computation subgraph, the baselines often struggle in this.

Quantitative analysis. GNNEXPLAINER also performs well quantitatively, on synthetic dataset explanation where groundtruth is available. For any given threshold, the edges included in the explanation are compared with groundtruths (cycle, house, grid). We then plot a PR curve to compute the AUC score. GNNEXPLAINER significantly out-performs the baselines by a large margin (see Table 1).

The explanation should also highlight relevant feature information, not only from itself but even within the set of neighbours that propagated on its most influential message-passing pathways. In an experiment such as BA-COMM, the explainer is forced to also integrate information from a restricted number of feature dimensions. While GNNEXPLAINER indeed highlights a compact feature representation in Figure 4, gradient-based approaches struggle to cope with the added noise, giving high importance scores to irrelevant feature dimensions. We also re-iterate the model's ability to learn structure and feature-based information jointly in Figure 3, where GNNEXPLAINER is again able to identify the correct substructure, as is the Attention-based baseline.

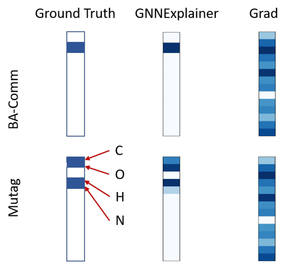

Figure 4: Feature importance visualization

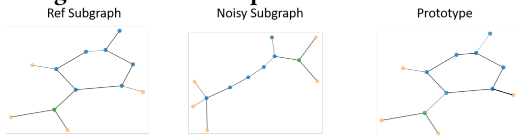

Figure 5: GnnExplainer is able to provide a prototype for a given node class, which can help identify functional subgraphs, e.g. a mutagenic compound from the Mutag dataset.

Table 1: GnnExplainer compared to baselines in identifying subgraphs using AUC.

	BA-Shapes	BA-Community	Tree-Cycles
GAT	0.957	0.974	0.914
Grad	0.936	0.883	0.885
GNN	**0.991**	**0.993**	**0.975**

6.3 Multi-instance prototype

In the context of multi-instance explanations, an explainer must not only highlight information locally relevant to a particular prediction, but also help emphasize higher-level correlations across instances. These instances can be related in arbitrary ways, but the most evident is class-membership. The assumption is that members of a class share common characteristics, and the model should help highlight them. For example, mutagenic compounds are often found to have certain characteristic functional groups that such NO_2, a pair of Oxygen atoms with a Nitrogen. While Figure 3 already hints to their presence, which the trained eye might recognize. The evidence grows stronger when a prototype is generated by GnnExplainer, shown in Figure 5. The model is able to pick-up on this functional structure, and promote it as archetypal of mutagenic compounds.

7 CONCLUSION

In this work, we present a novel explanation tool, GnnExplainer, which provides insights into any GNN that satisfies the neural message-passing scheme, without any modification to its architecture or re-training, and across any prediction task on graphs. It is able to leverage the recursive neighborhood-aggregation scheme of graph neural networks to identify not only important computational pathways but also highlight the relevant feature information that is passed along these edges.

REFERENCES

[1] A. Adadi and M. Berrada. 2018. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160.
[2] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim. 2018. Sanity checks for saliency maps. In *NeurIPS*.
[3] M. Agrawal, M. Zitnik, and J. Leskovec. 2018. Large-scale analysis of disease pathways in the human interactome. In *PSB*.
[4] M. Gethsiyal Augasta and T. Kathirvalavakumar. 2012. Reverse Engineering the Neural Networks for Rule Extraction in Classification Problems. *Neural Processing Letters* 35, 2 (April 2012), 131–150.
[5] L. Backstrom and J. Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*.
[6] P. W. Battaglia et al. 2018. Relational inductive biases, deep learning, and graph networks. (2018). arXiv:1806.01261
[7] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. In *ICLR*.
[8] J. Chen, Le Song, M. Wainwright, and M. Jordan. 2018. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. In *arXiv:1802.07814*.
[9] J. Chen, J. Zhu, and L. Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML*.
[10] Z. Chen, L. Li, and J. Bruna. 2019. Supervised Community Detection with Line Graph Neural Networks. In *ICLR*.
[11] E. Cho, S. Myers, and J. Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *KDD*.
[12] H. Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *ICML*.
[13] A. Debnath et al. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry* 34, 2 (1991), 786–797.
[14] M. Defferrard, X. Bresson, and P. Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.
[15] F. Doshi-Velez and B. Kim. 2017. Towards A Rigorous Science of Interpretable Machine Learning. (2017). arXiv: 1702.08608.
[16] D. Duvenaud et al. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*.
[17] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. 2009. Visualizing higher-layer features of a deep network. *University of Montreal* 1341, 3 (2009), 1.
[18] A. Fisher, C. Rudin, and F. Dominici. 2018. All Models are Wrong but many are Useful: Variable Importance for Black-Box, Proprietary, or Misspecified Prediction Models, using Model Class Reliance. (Jan. 2018). arXiv: 1801.01489.
[19] J. Gilmer, S. Schoenholz, P. Riley, O. Vinyals, and G. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML (JMLR)*, Vol. 70.
[20] R. Guidotti et al. 2018. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.* 51, 5 (2018), 93:1–93:42.
[21] W. Hamilton, R. Ying, and J. Leskovec. 2017. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin* (2017).
[22] W. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*.
[23] G. Hooker. 2004. Discovering additive structure in black box functions. In *KDD*.
[24] W.B. Huang, T. Zhang, Y. Rong, and J. Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. In *NeurIPS*.
[25] W. Jin, C. Coley, R. Barzilay, and T. Jaakkola. 2017. Predicting Organic Reaction Outcomes with Weisfeiler-Lehman network. In *NIPS*.
[26] B. Kim, R. Khanna, and O. Koyejo. 2016. Examples are not enough, learn to criticize! criticism for interpretability. In *NIPS*.
[27] T. N. Kipf and M. Welling. 2016. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[28] P. W. Koh and P. Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *ICML*.
[29] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec. 2017. Interpretable & Explorable Approximations of Black Box Models.
[30] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. 2015. Gated graph sequence neural networks. *arXiv:1511.05493* (2015).
[31] S. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *NIPS*.
[32] T. Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.* 267 (2019), 1–38.
[33] D. Neil et al. 2018. Interpretable Graph Convolutional Neural Networks for Inference on Noisy Knowledge Graphs. In *ML4H Workshop at NeurIPS*.
[34] M. Ribeiro, S. Singh, and C. Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *KDD*.
[35] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* (2009).
[36] M. Schlichtkrull, T. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*.
[37] G. J. Schmitz, C. Aldrich, and F. S. Gouws. 1999. ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. *IEEE Transactions on Neural Networks* (1999).
[38] A. Shrikumar, P. Greenside, and A. Kundaje. 2017. Learning Important Features Through Propagating Activation Differences. In *ICML*.
[39] M. Sundararajan, A. Taly, and Q. Yan. 2017. Axiomatic Attribution for Deep Networks. In *ICML*.

[40] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. 2018. Graph Attention Networks. In *ICLR*.

[41] T. Xie and J. Grossman. 2018. Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties. In *Phys. Rev. Lett.*

[42] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICRL*.

[43] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*.

[44] C. Yeh, J. Kim, I. Yen, and P. Ravikumar. 2018. Representer Point Selection for Explaining Deep Neural Networks. In *NeurIPS*.

[45] R. Ying, R. He, K. Chen, P. Eksombatchai, W. Hamilton, and J. Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*.

[46] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*.

[47] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. *NeurIPS*.

[48] M. Zeiler and R. Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *ECCV*.

[49] M. Zhang and Y. Chen. 2018. Link Prediction Based on Graph Neural Networks. In *NIPS*.

[50] Z. Zhang, Peng C., and W. Zhu. 2018. Deep Learning on Graphs: A Survey. *arXiv:1812.04202* (2018).

[51] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *arXiv:1812.08434* (2018).

[52] J. Zilke, E. Loza Mencia, and F. Janssen. 2016. DeepRED - Rule Extraction from Deep Neural Networks. In *Discovery Science*. Springer International Publishing.

[53] L. Zintgraf, T. Cohen, T. Adel, and M. Welling. 2017. Visualizing deep neural network decisions: Prediction difference analysis. In *ICLR*.

[54] M. Zitnik, M. Agrawal, and J. Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34 (2018).