ProtGNN: Towards Self-Explaining Graph Neural Networks

Zaixi Zhang[1], Qi Liu[1*], Hao Wang[1], Chengqiang Lu[1], Cheekong Lee[2*]
[1] Anhui Province Key Lab. of Big Data Analysis and Application,
School of Computer Science and Technology, University of Science and Technology of China
[2] Tencent America
{zaixi, wanghao3, lunar}@mail.ustc.edu.cn, qiliuql@ustc.edu.cn
cheekonglee@tencent.com

Abstract

Despite the recent progress in Graph Neural Networks (GNNs), it remains challenging to explain the predictions made by GNNs. Existing explanation methods mainly focus on post-hoc explanations where another explanatory model is employed to provide explanations for a trained GNN. The fact that post-hoc methods fail to reveal the original reasoning process of GNNs raises the need of building GNNs with built-in interpretability. In this work, we propose Prototype Graph Neural Network (ProtGNN), which combines prototype learning with GNNs and provides a new perspective on the explanations of GNNs. In ProtGNN, the explanations are naturally derived from the case-based reasoning process and are actually used during classification. The prediction of ProtGNN is obtained by comparing the inputs to a few learned prototypes in the latent space. Furthermore, for better interpretability and higher efficiency, a novel conditional subgraph sampling module is incorporated to indicate which part of the input graph is most similar to each prototype in ProtGNN+. Finally, we evaluate our method on a wide range of datasets and perform concrete case studies. Extensive results show that ProtGNN and ProtGNN+ can provide inherent interpretability while achieving accuracy on par with the non-interpretable counterparts.

Introduction

Graph Neural Networks (GNNs) have become increasingly popular since many real-world relational data can be represented as graphs, such as social networks (Bian et al. 2020), molecules (Lu et al. 2019) and financial data (Yang et al. 2020). Following a message passing paradigm to learn node representations, GNNs have achieved state-of-the-art performance in node classification, graph classification, and link prediction (Kipf and Welling 2017; Veličković et al. 2017; Xu et al. 2019). Despite the remarkable effectiveness of GNNs, explaining predictions made by GNNs remains a challenging open problem. Without understanding the rationales behind the predictions, these black-box models cannot be fully trusted and widely applied in critical areas such as medical diagnosis. In addition, model explanations can facilitate model debugging and error analysis. These indicate the necessity of investigating the explainability of GNNs.

Recently, extensive efforts have been made to study explanation techniques for GNNs (Yuan et al. 2020b). These methods can explain the predictions of node or graph classifications of trained GNNs with different strategies. For example, GNNExplainer (Ying et al. 2019) and PGExplainer (Luo et al. 2020) are proposed to select a compact subgraph structure that maximizes the mutual information with the GNN's predictions as the explanation. PGM-Explainer (Vu and Thai 2020) firstly obtains a local dataset by random node feature perturbation. Then it employs an interpretable Bayesian network to fit the local dataset and to explain the predictions of the original GNN model. In addition, XGNN (Yuan et al. 2020a) generates graph patterns to maximize the predicted probability for a certain class and provides model-level explanation. Despite the tremendous developments in the interpretation of GNNs, most existing approaches can be classified as *post-hoc* explanations where another explanatory model is used to provide explanations for a trained GNN. *Post-hoc* explanations can be inaccurate or incomplete in revealing the actual reasoning process of the original model (Rudin 2018). Therefore, it is more desirable to build models with inherent interpretability where the explanations are generated by the model themselves.

We leverage the concept of prototype learning to construct GNNs with *built-in* interpretability (i.e. self-explaining GNNs). In contrast to *post-hoc* explanation methods, the explanations generated by self-explaining GNNs are actually used during classification and are not generated *post-hoc*. Prototype learning is a form of case-based reasoning (Kolodner 1992; Schmidt et al. 2001), which makes the predictions for new instances by comparing them with several learned exemplar cases (i.e. prototypes). It is a natural practice in solving problems with graph-structured data. For example, chemists identify potential drug candidates based on known functional groups (i.e. key subgraphs) in molecular graphs (He et al. 2010; Zhang et al. 2021c). Prototype learning provides better interpretability by imitating such a human problem-solving process. Recently the concept of the prototype has been incorporated in convolutional neural networks to build interpretable image classifiers (Chen et al. 2018; Rymarczyk et al. 2021). However, so far prototype learning is not yet explored for explaining GNNs.

Building self-explaining GNNs based on prototype learning brings unique challenges. First, the discreteness of the edges

*Qi Liu and Cheekong Lee are the corresponding authors.

makes the projection and visualization of the graph prototypes difficult. Second, the combinatorial nature of graph structure makes it hard to build self-explaining models with both efficiency and high accuracy for graph modeling.

In this paper, we tackle the aforementioned challenges and propose **Prot**otype **G**raph **N**eural **N**etwork (ProtGNN), which provides a new perspective on the explanations of GNNs. Specifically, various popular GNN architectures can be employed as the graph encoder in ProtGNN. Prediction on a new input graph is performed based on its similarity to the prototypes in the prototype layer. Furthermore, we propose to employ the Monte Carlo tree search algorithm (Silver et al. 2017) to efficiently explore different subgraphs for prototype projection and visualization. In addition, in ProtGNN+, we design a conditional subgraph sampling module to identify which part of the input graph is most similar to each prototype for better interpretability and efficiency. Finally, extensive experiments on several real-world datasets show that ProtGNN/ProtGNN+ provides *built-in* interpretability while achieving comparable performance with the non-interpretable counterparts. The implementation is publicly available at https://github.com/zaixizhang/ProtGNN.

Related Work

Graph Neural Networks

Graph neural networks have demonstrated their effectiveness on various graph tasks. Let $G = (V, E)$ denotes a graph with node attributes X_v for $v \in V$ and a set of edges E. GNNs leverage the graph connectivity as well as node and edge features to learn a representation vector (i.e., embedding) h_v for each node $v \in V$ or a vector h_G for the entire graph G. Generally, GNNs follows a message passing paradigm, in which the representation of node v is iteratively updated by aggregating the representations of v's neighboring nodes $\mathcal{N}(v)$. Here we use Graph Convolutional Network (GCN) (Kipf and Welling 2017) as an example to illustrate such message passing procedures:

$$h_v^{k+1} = \sigma\Big(\sum_{u \in \mathcal{N}(v)} \big(W^k h_u^k \tilde{A}_{uv}\big) \Big), \tag{1}$$

where h_u^k is the representation vector of node u at the k-th layer and $\tilde{A} = \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix. $\hat{A} = A + I$ is the adjacency matrix of the graph G with self connections added and $\hat{D}$ is a diagonal matrix with $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$. $\sigma(\cdot)$ in Eq. (1) is the ReLU function and W^k is the trainable weight matrix of the k-th layer.

Explainability in Graph Neural Networks

As the application of GNNs grows, understanding why GNNs make such predictions becomes increasingly critical. Without understanding the mechanisms of making predictions, GNNs can hardly be used in critical or sensitive areas (Zhang et al. 2021a,b). Recently, the study of the explainability in GNNs is experiencing rapid developments. As Suggested by a recent survey (Yuan et al. 2020b), existing methods for explaining GNNs can be categorized into several classes: gradients/features-based methods (Baldassarre and Azizpour 2019; Pope et al. 2019), perturbation-based methods (Ying et al. 2019; Luo et al. 2020; Yuan et al. 2021; Schlichtkrull, De Cao, and Titov 2020), decomposition methods (Schwarzenberg et al. 2019; Schnake et al. 2020), and surrogate methods (Vu and Thai 2020; Huang et al. 2020).

Specifically, the gradients/features-based methods employ the gradients or the feature values to indicate the importance of different input features. These methods simply adapt existing explanation techniques in the image domain to the graph domain without incorporating the properties of graph data. Perturbation-based methods monitor the changes in the predictions by perturbing different input features and identifies the most influential features. Decomposition methods explain GNNs by decomposing the original model predictions into several terms and associating these terms with graph nodes or edges. Given an input example, surrogate methods firstly sample a dataset from the neighborhood of the given example and then fit a simple and interpretable model, e.g., a decision tree to the sampled dataset. The surrogate models are usually easier to interpret, shedding light into the inner-workings of more complex models.

However, all the above methods are *post-hoc* explanation methods. Compared with *post-hoc* explanation methods, *built-in* interpretability (Chen et al. 2018; Ming et al. 2019) is more desirable since *post-hoc* explanations usually do not fit the original model precisely (Rudin 2018). Therefore, it is necessary to build models with inherent interpretability and high accuracy.

The Proposed ProtGNN

In this section, We introduce the architecture of ProtGNN/ProtGNN+, formulate the learning objective and describe the training procedures.

ProtGNN Architecture

We let $\{x_i, y_i\}_{i=1}^n$ be a labeled training dataset, where x_i is the input attributed graph and $y_i \in \{1, ..., C\}$ is the label of the graph. We aim to learn representative prototypical graph patterns that can be used for classification references and analogical explanations. For a new input graph, its similarities with each prototype are measured in the latent space. Then, the prediction of the new instance can be derived and explained by its similar prototype graph patterns.

In Figure 1, we show the overview of the architecture of our proposed ProtGNN. The network consists of three key components: a graph encoder f, a prototype layer g_P, and a fully connected layer c appended by softmax to output the probabilities in multi-class classification tasks.

For a given input graph x_i, the graph encoder f maps the whole graph into a single graph embedding h with a fixed length. The encoder could be any backbone GNN e.g., GCN,

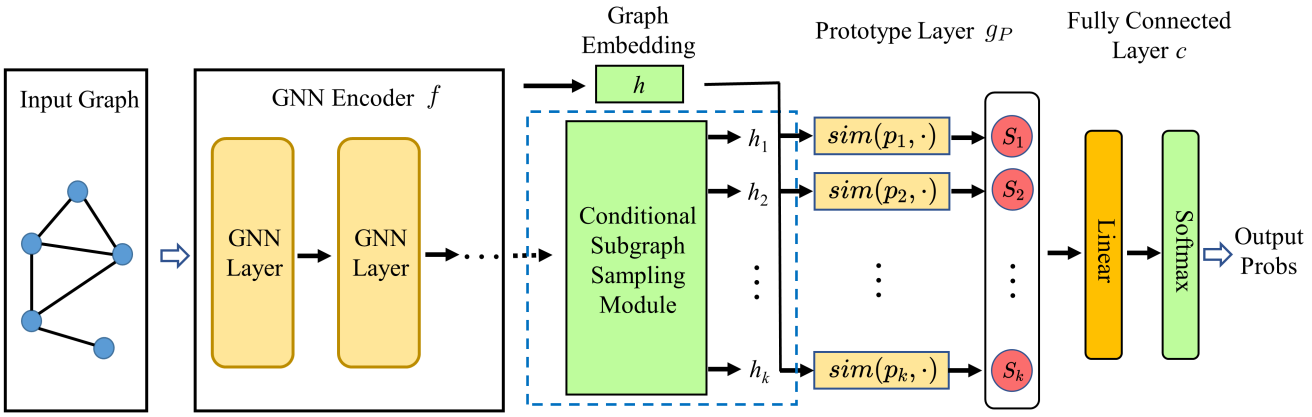

Figure 1: The architecture of our proposed ProtGNN/ProtGNN+. The model mainly consists of three parts: GNN encoder f, prototype layer g_P, and the fully connected layer c appended by softmax to output probabilities in multi-class classification tasks. ProtGNN calculates the similarity score ($sim(p_k, \cdot)$ in the illustration) between the graph embedding and the learned prototypes in the prototype layer. For further interpretability, the conditional subgraph sampling module (in the dashed bounding box) is incorporated in ProtGNN+ to output subgraphs most similar to each learned prototype.

GAT or GIN. The graph embedding h could be obtained by taking a sum or max pooling of the last GNN layer.

In the prototype layer, we allocate a pre-determined number of prototypes m for each class. In the final trained ProtGNN, each class can be represented by a set of learned prototypes. The prototypes should capture the most relevant graph patterns for identifying graphs of each class. For each input graph x_i and its embedding h, the prototype layer computes the similarity scores:

$$sim(p_k, h) = log(\frac{\|p_k - h\|_2^2 + 1}{\|p_k - h\|_2^2 + \epsilon}) \tag{2}$$

where p_k is the k-th prototype with the same dimension as the graph embedding h. The similarity function is designed to be monotonically decreasing to $\|p_k - h\|_2$ and always greater than zero. In experiments, ϵ is set to a small value e.g., 1e-4. Finally, with the similarity scores, the fully connected layer with softmax computes the output probabilities for each class.

Learning Objective

Our goal is to learn a ProtGNN with both accuracy and inherent interpretability. For accuracy, we minimize the cross-entropy loss on the training dataset: $\frac{1}{n}\sum_{i=1}^{n} \text{CrsEnt}(c \circ g_p \circ f(x_i), y_i)$. For better interpretability, we consider several constraints in constructing prototypes for the explanation. Firstly, the cluster cost (Clst) encourages that each graph embedding should at least be close to one prototype of its own class. Secondly, the separation cost (Sep) encourages that each graph embedding should stay far away from prototypes not of its class. Finally, we found in experiments that some learned prototypes are very close to each other in the latent space. We encourage the diversity of the learned prototypes by adding the diversity loss (Div) which penalizes prototypes too close to each other.

To sum up, the objective function we aim to minimize is

$$\frac{1}{n}\sum_{i=1}^{n} \text{CrsEnt}(c \circ g_p \circ f(x_i), y_i) + \lambda_1 \text{Clst} + \lambda_2 \text{Sep} + \lambda_3 \text{Div}, \tag{3}$$

$$\text{Clst} = \frac{1}{n}\sum_{i=1}^{n} \min_{j:p_j \in P_{y_i}} \|f(x_i) - p_j\|_2^2 \tag{4}$$

$$\text{Sep} = -\frac{1}{n}\sum_{i=1}^{n} \min_{j:p_j \notin P_{y_i}} \|f(x_i) - p_j\|_2^2 \tag{5}$$

$$\text{Div} = \sum_{k=1}^{C} \sum_{\substack{i \neq j \\ p_i, p_j \in P_k}} \max(0, cos(p_i, p_j) - s_{max}) \tag{6}$$

where λ_1, λ_2, and λ_3 are hyper-parameters controlling the weights of the losses. P_{y_i} is the set of prototypes belonging to class y_i. s_{max} is the threshold of the cosine similarity measured by $cos(\cdot, \cdot)$ in the diversity loss.

Prototype Projection

The learned prototypes are embedding vectors that are not directly interpretable. For better interpretation and visualization, we design a projection procedure performed in the training stage. Specifically, we project each prototype p_j ($p_j \in P_k$) onto the nearest latent training subgraph from the same class as that of p_j (see Eq. (7)). In this way, we can conceptually equate each prototype with a subgraph, which is more intuitive and human-intelligible. To reduce the computational cost, the projection step is only performed every few training epochs:

$$\begin{aligned} p_j &\leftarrow \arg\min_{\widetilde{h} \in \mathcal{H}_j} \|\widetilde{h} - p_j\|_2, \\ \mathcal{H}_j &= \{\widetilde{h} : f(\widetilde{x}), \widetilde{x} \in \text{Subgraph}(x_i)\ \forall i\ s.t.\ y_i = k\}. \end{aligned} \tag{7}$$

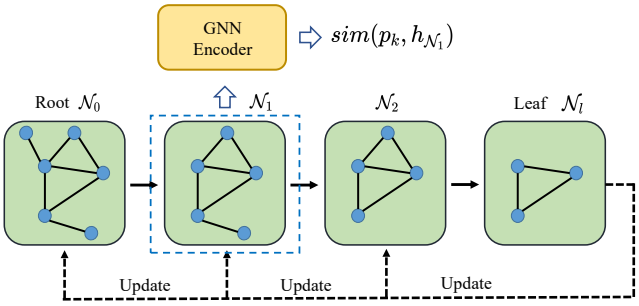

Figure 2: An illustration of graph prototype projection with Monte Carlo Tree Search. The bottom shows one selected path from the root to leaves in the search tree, which corresponds to one iteration of MCTS. Nodes that are not selected are ignored for simplicity. For each node, its subgraph is evaluated by computing the similarity score via GNN Encoder and the similarity function. In this figure, we show the computation of similarity score for the node $\mathcal{N}_1$ (shown in blue dashed box). In the backward pass, the model updates the statistics of each node.

Unlike grid-like data such as images, the combinatorial characteristic of graph makes it unrealistic to find the nearest subgraph by enumeration (Chen et al. 2018). In graph prototype projection, we employ the Monte Carlo tree search algorithm (MCTS) (Silver et al. 2017) as the search algorithm to guide our subgraph explorations (see Figure 2). We build a search tree in which the root is associated with the input graph and each of other nodes corresponds to an explored subgraph. Formally, we define each node in the search tree $\mathcal{T}$ as $\mathcal{N}_i$ and $\mathcal{N}_0$ denotes the root node. The edges in the search tree represent the pruning actions. In the search tree, the graph associated with a child node can be obtained by performing node-pruning from the graph corresponding to its parent node. To limit the search space, we have added two additional constraints: $\mathcal{N}_i$ has to be a connected subgraph and the size of the projected subgraph should be small.

During the search process, the MCTS algorithm records the statistics of visiting counts and rewards to guide the exploration and reduce the search space. Specifically, for the node and pruning action pair $(\mathcal{N}_i, a_j)$, we assume that the subgraph $\mathcal{N}_j$ is obtained by action a_j from $\mathcal{N}_i$. The MCTS algorithm records four variables for $(\mathcal{N}_i, a_j)$:

- C$(\mathcal{N}_i, a_j)$ denotes the number of counts for selecting action a_j for node $\mathcal{N}_i$.
- W$(\mathcal{N}_i, a_j)$ is the total reward for all $(\mathcal{N}_i, a_j)$ visits.
- Q$(\mathcal{N}_i, a_j)$ is the averaged reward for multiple visits.
- R$(\mathcal{N}_i, a_j)$ is the immediate reward for selecting a_j on $\mathcal{N}_i$, which is measured by the similarity between the prototype and the subgraph embedding in this paper. The subgraph embedding is obtained by encoding the subgraph with the GNN encoder f.

Guided by these statistics, MCTS searches for the nearest subgraphs in multiple iterations. Each iteration consists of two phases. In the forward pass, MCTS selects a path starting from the root $\mathcal{N}_0$ to a leaf node $\mathcal{N}_l$. To keep subgraphs connected, we select to prune peripheral nodes with minimum degrees. The leaf node can be defined based on the numbers of nodes in subgraphs such that $|\mathcal{N}_l| \leq N_{\min}$. The action selection criteria at node $\mathcal{N}_i$ is:

$$a^* = \underset{a_j}{\operatorname{argmax}}\, Q(\mathcal{N}_i, a_j) + U(\mathcal{N}_i, a_j) \tag{8}$$

$$U(\mathcal{N}_i, a_j) = \lambda R(\mathcal{N}_i, a_j) \frac{\sqrt{\sum_k C(\mathcal{N}_i, a_k)}}{1 + C(\mathcal{N}_i, a_j)}, \tag{9}$$

where λ is a hyper-parameter to control the trade-off between exploration and exploitation. The strategy initially prefers to select child nodes with low visit counts to explore different pruning actions, but asympotically prefers actions leading to higher similarity scores.

In the backward pass, the statistics of all node and action pairs selected in this path are updated:

$$C(\mathcal{N}_i, a_j) = C(\mathcal{N}_i, a_j) + 1 \tag{10}$$

$$W(\mathcal{N}_i, a_j) = W(\mathcal{N}_i, a_j) + sim(p_k, h_{\mathcal{N}_l}), \tag{11}$$

where $h_{\mathcal{N}_l}$ is the embedding of the subgraph associated to the leaf node $\mathcal{N}_l$. In the end, we select the subgraph with the highest similarity score from all the expanded nodes as the new projected prototype.

Conditional Subgraph Sampling Module

We further propose ProtGNN+ with a novel conditional subgraph sampling module to provide better interpretation. In ProtGNN+, we not only show the similarity scores to prototypes, but also identify which part of the input graph is most similar to each prototype as part of the reasoning process. In Figure 1, the conditional subgraph sampling module outputs different subgraph embeddings for each prototype. While this task can also be accomplished by MCTS, the exponentially-growing time complexity to the graph size and the difficulty of parallelization and generalization make MCTS algorithm an undesirable choice. Instead, we adopt a parameterized method for efficient similar subgraph selection conditioned on given prototypes.

Formally, we let $e_{ij} \in \{0, 1\}$ be the binary variable indicating whether the edge between node i and j is selected. The matrix of e_{ij} is denoted as $\mathcal{E}$. The optimization objective of the conditional subgraph sampling module is:

$$\max_{\mathcal{E}} sim(p_k, f(\mathcal{G}_s)) \; s.t. \; |\mathcal{G}_s| \leq B, \tag{12}$$

where $\mathcal{G}_s$ is the selected subgraph whose adjacency matrix is $\mathcal{E}$. B is the maximum size of the subgraph.

The combinatorial and discrete nature of graph makes the direct optimization of the above objective function intractable. We first consider a relaxation by assuming that the explanatory graph is a Gilbert random graph (Gilbert 1959) where the state of each edge is independent to each other. Furthermore, for ease of gradient computation and update, we relax

$\mathcal{E} \in \{0,1\}^{N\times N}$ into convex space $\mathcal{E} \in [0,1]^{N\times N}$. N is the number of nodes in the input graph. For efficiency and generalizability, we adopt deep neural networks to learn e_{ij}:

$$e_{ij} = \sigma(\text{MLP}_\theta([z_i; z_j; p_k])), \tag{13}$$

where $\sigma(\cdot)$ here is the Sigmoid function. MLP is a multi-layer neural network parameterized with θ and $[\cdot;\cdot;\cdot]$ is the concatenation operation. z_i and z_j are node embedding obtained from the GNN Encoder, which encodes the feature and structure information of the nodes' neighborhood. Then the objective in Eq. (12) becomes

$$\begin{aligned} &\max_\theta sim(p_k, f(\mathcal{G}_s)) - \lambda_b R_b \\ &R_b = \text{ReLU}(\sum_{e_{ij}\in\mathcal{E}} e_{ij} - B), \end{aligned} \tag{14}$$

where λ_b is the weight for the budget regularization R_b. In our experiments, we adopt stochastic gradient descent to optimize the objective function.

Comparison with MCTS: Our designed conditional subgraph sampling module is much more efficient than MCTS and easier for parallel computation. The parameters of our conditional subgraph sampling module are fixed and independent of the graph size. To sample from a graph with $|\mathcal{E}|$ edges, the time complexity of our method is $\mathcal{O}(|\mathcal{E}|)$. One limitation of the conditional subgraph sampling module is that it requires additional training. Therefore, MCTS is still used in the prototype projection step of ProtGNN+ for the stability of optimization.

Theorem on Subgraph Sampling

To provide more understandable visualization, ProtGNN+ prunes the input graph to find the subgraphs most similar to prototypes and then calculates the similarity scores. Compared with ProtGNN, the subgraph sampling procedure may affect the classification accuracy. The following theorem provides some theoretical understanding of how input graph sampling affects classification accuracy.

Theorem 1: Let $c \circ g_p \circ f$ be a ProtoGNN. The embedding of the input graph is h. We assume that the number of prototypes is the same for each class, and is denoted as m. For each class k, the weight connection in the last layer c between a class k prototype and the class k logit is 1, and that between a non-class k prototype and the class k logit is 0. We denote p_l^k as the l-th prototype for class k and h_l^k the embedding of the pruned subgraph. ProtGNN and ProtGNN+ has the same graph encoder f. We make the following assumptions: there exists some δ with $0 < \delta < 1$,

- for the correct class, we have $\|h - h_l^k\|_2 \leq (\sqrt{1+\delta} - 1)\|h - p_l^k\|_2$ and $\|h - p_l^k\|_2 \leq \sqrt{1-\delta}$;
- for the incorrect classes, $\|h - h_l^k\|_2 \leq \theta\|h - p_l^k\|_2 - \sqrt{\epsilon}$, $\theta = \min(\sqrt{1+\delta} - 1, 1 - \frac{1}{\sqrt{2-\delta}})$.

For one correctly classified input graph in ProtGNN, if the output logits between the top-2 classes are at least $2mlog((1+\delta)(2-\delta))$, then ProtGNN+ can classify the input graph correctly as well.

Algorithm 1: Overview of ProtGNN/ProtGNN+ Training

Input: Training dataset $\{x_i, y_i\}_{i=1}^n$
Parameter: Training epochs T, Warm-up epoch T_w, Projection epoch T_p, Prototype projection period τ, ProtGNN+

```
Initialize model parameters.
for training epochs t = 1, 2, ··· , T do
    Optimizing objective function in Eq. (3)
    if t>T_p and t%τ = 0 then
        Performing prototype projection with MCTS
    end if
    if ProtGNN+ enabled and t>T_w then
        Optimizing the objective function in Eq. (14).
    end if
end for
```

Output: Trained model, prototype visualization

The intuition behind Theorem 1 is that if the subgraph sampling does not change the graph embedding too much, ProtGNN+ will have the same correct predictions as ProtGNN. The proof is included in the appendix.

Training Procedures

Before training starts, we randomly initialize the model parameters. We let w_c be the weight matrix of the fully connected layer c and $w_c^{(k,j)}$ be the weight connection between the output of the j-th prototype and the logit of class k. In particular, for a class k, we set $w_c^{(k,j)} = 1$ for all j with $p_j \in P_k$ and $w_c^{(k,j)} = 0$ for all j with $p_j \notin P_k$. Intuitively, such initialization of w_h encourages prototypes belonging to class k to learn semantic concepts that are characteristic to class k. After training begins, we employ gradient descents to optimize the objective function in Eq. (3). If the training epoch is larger than the projection epoch T_p, we perform the prototype projection step every few training epochs. Furthermore, if we train ProtGNN+, the conditional subgraph sampling module and ProtGNN are iteratively optimized after the warm-up epoch T_w when the optimization of GNN encoder and prototypes are stabilized.

ProtGNN for Generic Graph Tasks

In the above sections and illustrations, we have described ProtGNN/ProtGNN+ using graph classification as an example. It is worth noting that ProtGNN/ProtGNN+ can be easily generalized to other graph tasks, such as node classification and link prediction. For example, in the node classification task, the explanation target is to provide the reasoning process behind the prediction of node v_i. Assuming the GNN encoder has L layers, the prediction of node v_i only relies on its L-hop computation graph. Therefore, prototype projection and conditional subgraph sampling are all performed in the L-hop computation graph.

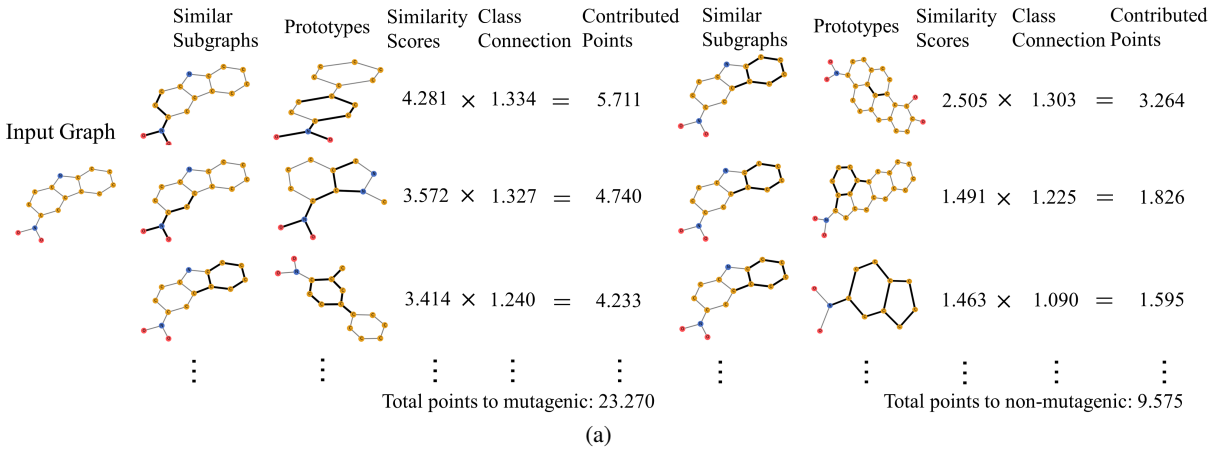

Figure 3: The reasoning process of ProtGNN+ in deciding whether the molecular graph is mutagenic. The predictions are based on the similarity between the latent input representations against the prototypes. The network tries to find evidence by looking at which subgraph was mostly similar to the prototypes. The selected subgraphs are highlighted. Due to space constraint, we only show several prototypes with the largest weights for each class.

Datasets	GCN			GIN			GAT		
	Original	ProtGNN	ProtGNN+	Original	ProtGNN	ProtGNN+	Original	ProtGNN	ProtGNN+
MUTAG	73.3±5.8	**76.7±6.4**	73.3±2.9	**93.3±2.9**	90.7±3.2	91.7±2.9	75.0±5.0	78.3±4.2	**81.7±2.9**
BBBP	84.6±3.4	**89.4±4.1**	88.0±4.6	86.2±1.1	**86.5±1.6**	85.9±4.0	83.0±2.6	**85.9±2.5**	85.5±0.8
Graph-SST2	89.7±0.5	**89.9±2.4**	89.0±3.0	92.2±0.3	92.0±0.2	**92.3±0.4**	88.1±0.8	**89.1±1.2**	88.7±0.9
Graph-Twitter	67.5±1.9	**68.9±5.9**	66.1±6.5	66.2±1.3	75.2±2.8	**76.5 ±3.4**	**69.6±6.5**	64.8±4.0	66.4±3.3
BA-Shape	91.9±1.7	**95.7±1.4**	94.3±3.7	92.9±0.5	95.2±1.3	**95.5 ±2.4**	92.9±1.2	**93.4±3.4**	93.2±2.0

Table 1: The classification accuracies and standard deviations (%) of ProtGNN, ProtGNN+, and the original GNNs.

Experimental Evaluation

Datasets and Experimental Settings

Datasets: We conduct extensive experiments on different datasets and GNN models to demonstrate the effectiveness of our proposed model. These datasets are listed as below:

- MUTAG (Debnath et al. 1991) and BBBP (Wu et al. 2018) are molecule datasets for graph classification. In these datasets, nodes represent atoms and edges denote chemical bonds. The labels of molecular graphs are determined by the molecular compositions.
- Graph-SST2 (Socher et al. 2013) and Graph-Twitter (Dong et al. 2014) are sentiment graph datasets for graph classification. They convert sentences to graphs with Biaffine parser (Gardner et al. 2018) that nodes denote words and edges represent the relationships between words. The node embeddings are initialized with Bert word embeddings (Devlin et al. 2018). The labels are determined by the sentiment of text sentences.
- BA-Shape is a synthetic node classification dataset. Each graph contains a base graph obtained from the Barabási-Albert (BA) mode (Albert and Barabási 2002) and a house-like five-node motif attached to the base graph. Each node is labeled based on whether it belongs to the base graph or the different spatial locations of the motif.

Methods	GCN	ProtGNN	ProtGNN+	ProtGNN+*
Time	177.9 s	506.3 s	632.7 s	$>$ 2 hrs

Table 2: Efficiency studies of different methods on BBBP

Experimental Settings: In our evaluation, we use three variants of GNNs, i.e. GCN, GAT, and GIN. The split for train/validation/test sets is $80\% : 10\% : 10\%$. All models are trained for 500 epochs with an early stopping strategy based on accuracy on the validation set. We adopt the ADAM optimizer with a learning rate of 0.005. In Eq.(3), the hyper-parameters λ_1, λ_2, and λ_3 are set to 0.10, 0.05, and 0.01 respectively. s_{max} is set to 0.3 in Eq. (6). The number of prototypes per class m is set to 5. In MCTS for prototype projection, we set λ in Eq. (9) to 5 and the number of iterations to 20. Each node in the Monte Carlo Tree can expand up to 10 child nodes and $N_{\min}$ is set to 5. The prototype projection period τ is set to 50 and the projection epoch T_p is set to 100. In the training of ProtGNN+, the warm-up epoch T_w is set to 200. We employ a three-layer neural network to learn edge weights. In Eq. (14), λ_b is set to 0.01 and B is set to 10. We select hyper-parameters based on related works or grid search, an analysis on hyper-parameters is included in the appendix. Generally, we find our method is robust to various hyperparameters. All our experiments are conducted

with one Tesla V100 GPU.

Evaluations on ProtGNN/ProtGNN+

Comparison with Post-hoc Methods

Furthermore, even though ProtGNN+ and post-hoc methods fall into different categories (built-in and post-hoc), we try to provide comparisons here. Specifically, we use the sampled subgraph provided by the conditional subgraph sampling module as the explanation. We compare ProtGNN+ with two post-hoc methods: PGExplainer (Luo et al. 2020) and GNNExplainer (Ying et al. 2019). In Figure 5, we show that the explanation provided by post-hoc methods may be inaccurate or incomplete while ProtGNN+ can capture the NO2 functional group and the house-like motif well.

Comparison with Baselines In Table 1, we compare the classification accuracy of ProtGNN/ProtGNN+ with the original GNNs. We apply 3 independent runs on random data splitting and report the means and standard deviations. In the following sections, we use GCN as the default backbone model. As we can see, ProtGNN and ProtGNN+ achieve comparable classification performance with the corresponding original GNN models, which also empirically verifies Theorem 1.

Reasoning Process of Our Network In Figure 3, we perform case studies on MUTAG to qualitatively evaluate the performance of our proposed method. We visualize the prototypes and show the reasoning process of our ProtGNN+ in reaching a classification decision on input graphs. In particular, given an input graph x, the network finds the likelihood to be in each class by comparing it with prototypes from each class. The conditional subgraph sampling module finds the most similar subgraphs in x. These similarity scores are calculated, weighted, and summed together to give a final score for x belonging to each class. For example, Figure 3 shows the reasoning process of ProtGNN+ in deciding whether the input molecular graph is mutagenic. Based on chemical domain knowledge (Debnath et al. 1991), carbon rings and NO_2 groups tend to be mutagenic. In the Prototype column of the mutagenic class, we can observe that the prototypes can capture the structures of NO_2 and carbon rings well. Moreover, in the column of Similar Subgraphs, the conditional subgraph sampling module can effectively identify the most similar subgraphs. For instance, in the first row of the mutagenic class, the NO_2 group and part of the carbon ring can be identified, which is quite similar to the prototype on the right.

Overall, our method provides interpretable evidence to support classifications. Such explanations participate in the actual model computation and is always faithful to the classification decisions. More examples and case studies are reported in appendix.

t-SNE Visualization of Prototypes In Figure 5 we show the visualization on BBBP dataset of the graph and prototype embeddings using t-SNE method. We can observe that the prototypes can occupy the centers of graph embeddings, which verifies the effectiveness of prototype learning.

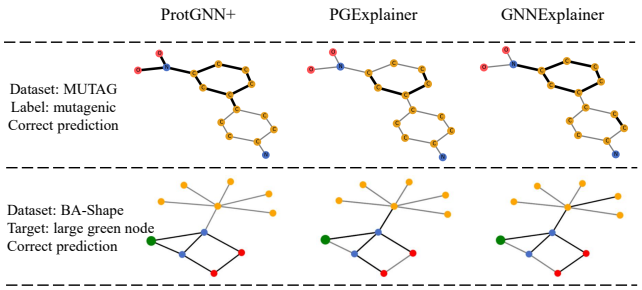

Figure 4: Explanation provided by ProtGNN+ and baselines. ProtGNN+ can capture the NO2 functional group and the house-like motif well while the baselines fail.

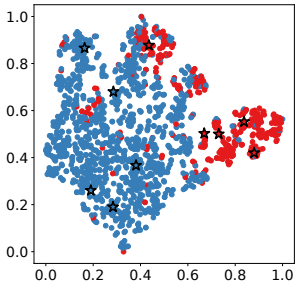

Figure 5: Visualization on BBBP dataset of the graph (dots) and prototype (stars) embeddings using the t-SNE method. Different colors indicate different classes.

Efficiency Studies Finally, we study the efficiency of our proposed methods. In Table 2, we show the time required to finish training for each model. Here ProtGNN+* denotes using MCTS for subgraph sampling in the training of ProtGNN+. The time complexity of ProtGNN+* is extremely high due to the complexity of MCTS. The proposed conditional subgraph sampling module can effectively reduce the time cost of ProtGNN+. Although ProtGNN and ProtGNN+ have a larger time cost compared to GCN (largely due to prototype projection with MCTS), the time cost is still acceptable considering the provided built-in interpretability.

Conclusion

While extensive efforts have been made to explain GNNs from different angles, none of existing methods can provide *built-in* explanations for GNNs. In this paper, we propose ProtGNN/ProtGNN+ which provides a new perspective on the explanations of GNNs. The prediction of ProtGNN is obtained by comparing the inputs to a few learned prototypes in the prototype layer. For better interpretability and higher efficiency, a novel conditional subgraph sampling module is proposed to indicate the subgraphs most similar to prototypes. Extensive experimental results show that our method can provide a human-intelligible reasoning process with acceptable classification accuracy and time-complexity.

Acknowledgments

This research was partially supported by grants from the National Natural Science Foundation of China (Grants No. 61922073 and U20A20229) and 2021 Tencent Rhino-Bird Research Elite Training Program. Qi Liu gratefully acknowledges the support of the Youth Innovation Promotion Association of CAS (No. 2014299).

References

Albert, R.; and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1): 47.

Baldassarre, F.; and Azizpour, H. 2019. Explainability techniques for graph convolutional networks. *ICML workshop*.

Bian, T.; Xiao, X.; Xu, T.; Zhao, P.; Huang, W.; Rong, Y.; and Huang, J. 2020. Rumor detection on social media with bi-directional graph convolutional networks. In *AAAI*, volume 34, 549–556.

Chen, C.; Li, O.; Tao, C.; Barnett, A. J.; Su, J.; and Rudin, C. 2018. This looks like that: deep learning for interpretable image recognition. *NeurIPS*.

Debnath, A. K.; Lopez de Compadre, R. L.; Debnath, G.; Shusterman, A. J.; and Hansch, C. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2): 786–797.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dong, L.; Wei, F.; Tan, C.; Tang, D.; Zhou, M.; and Xu, K. 2014. Adaptive recursive neural network for target-dependent twitter sentiment classification. In *ACL*, 49–54.

Gardner, M.; Grus, J.; Neumann, M.; Tafjord, O.; Dasigi, P.; Liu, N.; Peters, M.; Schmitz, M.; and Zettlemoyer, L. 2018. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*.

Gilbert, E. N. 1959. Random graphs. *The Annals of Mathematical Statistics*, 30(4): 1141–1144.

He, Z.; Zhang, J.; Shi, X.-H.; Hu, L.-L.; Kong, X.; Cai, Y.-D.; and Chou, K.-C. 2010. Predicting drug-target interaction networks based on functional groups and biological features. *PloS one*, 5(3): e9603.

Huang, Q.; Yamada, M.; Tian, Y.; Singh, D.; Yin, D.; and Chang, Y. 2020. Graphlime: Local interpretable model explanations for graph neural networks. *arXiv preprint arXiv:2001.06216*.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. *ICLR*.

Kolodner, J. L. 1992. An introduction to case-based reasoning. *Artificial intelligence review*, 6(1): 3–34.

Lu, C.; Liu, Q.; Wang, C.; Huang, Z.; Lin, P.; and He, L. 2019. Molecular Property Prediction: A Multilevel Quantum Interactions Modeling Perspective. *AAAI*.

Luo, D.; Cheng, W.; Xu, D.; Yu, W.; Zong, B.; Chen, H.; and Zhang, X. 2020. Parameterized explainer for graph neural network. *NeurIPS*.

Ming, Y.; Xu, P.; Qu, H.; and Ren, L. 2019. Interpretable and steerable sequence learning via prototypes. In *SIGKDD*.

Pope, P. E.; Kolouri, S.; Rostami, M.; Martin, C. E.; and Hoffmann, H. 2019. Explainability methods for graph convolutional neural networks. In *CVPR*, 10772–10781.

Rudin, C. 2018. Please stop explaining black box models for high stakes decisions. *stat*, 1050: 26.

Rymarczyk, D.; Struski, Ł.; Tabor, J.; and Zieliński, B. 2021. ProtoPShare: Prototype Sharing for Interpretable Image Classification and Similarity Discovery. *SIGKDD*.

Schlichtkrull, M. S.; De Cao, N.; and Titov, I. 2020. Interpreting graph neural networks for nlp with differentiable edge masking. *arXiv preprint arXiv:2010.00577*.

Schmidt, R.; Montani, S.; Bellazzi, R.; Portinale, L.; and Gierl, L. 2001. Cased-based reasoning for medical knowledge-based systems. *International Journal of Medical Informatics*, 64(2-3): 355–367.

Schnake, T.; Eberle, O.; Lederer, J.; Nakajima, S.; Schütt, K. T.; Müller, K.-R.; and Montavon, G. 2020. XAI for graphs: explaining graph neural network predictions by identifying relevant walks. *arXiv e-prints*, arXiv–2006.

Schwarzenberg, R.; Hübner, M.; Harbecke, D.; Alt, C.; and Hennig, L. 2019. Layerwise relevance visualization in convolutional text graph classifiers. *arXiv preprint arXiv:1909.10911*.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 1631–1642.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Vu, M.; and Thai, M. T. 2020. PGM-Explainer: Probabilistic Graphical Model Explanations for Graph Neural Networks. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *NeurIPS*, volume 33.

Wu, Z.; Ramsundar, B.; Feinberg, E. N.; Gomes, J.; Geniesse, C.; Pappu, A. S.; Leswing, K.; and Pande, V. 2018. MoleculeNet: a benchmark for molecular machine learning. *Chemical science*, 9(2): 513–530.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *ICLR*.

Yang, S.; Zhang, Z.; Zhou, J.; Wang, Y.; Sun, W.; Zhong, X.; Fang, Y.; Yu, Q.; and Qi, Y. 2020. Financial Risk Analysis for SMEs with Graph-based Supply Chain Mining. In *IJCAI*, 4661–4667.

Ying, R.; Bourgeois, D.; You, J.; Zitnik, M.; and Leskovec, J. 2019. Gnnexplainer: Generating explanations for graph neural networks. *NeurIPS*, 32: 9240.

Yuan, H.; Tang, J.; Hu, X.; and Ji, S. 2020a. Xgnn: Towards model-level explanations of graph neural networks. In *SIGKDD*, 430–438.

Yuan, H.; Yu, H.; Gui, S.; and Ji, S. 2020b. Explainability in graph neural networks: A taxonomic survey. *arXiv preprint arXiv:2012.15445*.

Yuan, H.; Yu, H.; Wang, J.; Li, K.; and Ji, S. 2021. On explainability of graph neural networks via subgraph explorations. *ICML*.

Zhang, Z.; Jia, J.; Wang, B.; and Gong, N. Z. 2021a. Backdoor attacks to graph neural networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, 15–26.

Zhang, Z.; Liu, Q.; Huang, Z.; Wang, H.; Lu, C.; Liu, C.; and Chen, E. 2021b. GraphMI: Extracting Private Graph Data from Graph Neural Networks. In *IJCAI*.

Zhang, Z.; Liu, Q.; Wang, H.; Lu, C.; and Lee, C.-K. 2021c. Motif-based Graph Self-Supervised Learning for Molecular Property Prediction. *NeurIPS*, 34.