

A Review of Methods for Explainable and Interpretable Graph Neural Networks

Art Tay

1 Introduction

Graphs provide an incredibly flexible structure for modeling complex data. Data can naturally appear as graphs, like molecules. We can reduce data to a graph, such as the key points of a image. We can even use graphs to add structure, such as grammatical relationships. Graph Neural Networks (GNNs) have become a popular choice for prediction and inference on graph data. At their core, GNNs work by iteratively updating node embeddings based on information from neighboring nodes. The idea is to use the graph’s structure to engineer better features. This message passing scheme allows GNNs to capture complex dependencies and patterns present within the graph structure. GNN architectures typically consist of multiple layers, each performing message passing and aggregation operations to refine the embeddings. These layers are often followed by pooling and dense prediction layers to produce the final output. Some important applications of graph classification include predicting chemical toxicity ([Bai et al. 2019](#)), classifying proteins ([Gallicchio and Micheli 2019](#)), and even detecting cancer from pathology slides ([Xiao et al. 2023](#)). While GNNs achieve remarkable predictive power, their complexity prevents the exaction of the scientific rationale. Like many deep learning models, the black box nature of GNNs prevents wide adoption. Without strong methods for understanding their predictions, GNNs are more susceptible to adversarial attacks and undetected discrimination. Inferential methods also serve to direct modeling efforts by highlighting common structures or features that may be predictive in other applications. These reasons underscore the critical importance of methods for explaining and interpreting GNN models.

Although explainability and interpretability are sometimes used interchangeably in the literature, we will adopt the distinction expressed in Yuan et al. (2022). In this article the authors say that a model is explainable if the models predictions can be reasoned post-hoc (Yuan et al. 2022). For example, the effect of a specific input variable can be estimated by either dropping or randomly permuting the variable and accessing the effect on the output (Breiman 2001). This paradigm could be directly applied to GNNs; however, researchers are often not interested in the statistical significance of tabular node and edge-level features. Most scientific questions focus on the importance of specific graph-level substructures. The challenge with GNN explanations is that, in addition to computing the importance of features, the high-level features themselves must first be identified. On the other hand, a model is interpretable if the model’s decision process can be readily understood by humans (Yuan et al. 2022). For example, a linear regression model is interpretable because each coefficient clearly defines the relationship between any input and output. An interpretable model is typically superior to an explainable one because similar to a partial F-test, explanation methods might identify that certain variables enhance the model, but they cannot explain how these variables contribute. On the flip side, interpretable models may not reach the performance levels of black-box models. GNN models are generally not interpretable because the relation expressed by the parameters tends to exceed human understanding. A direct translation from traditional statistics would be a circuit type analysis, which has work for image processing convolution neural networks (Olah et al. 2020). For graphs, this would involve using coefficients on subgraphs to produce predictions. Similar to the difficulties of explanation methods, these subgraphs would have to be identified, which can be computational complex and expensive given the combinatorial nature of graphs. Overcoming these challenges has substantial scientific impacts. In applications where GNNs demonstrate strong predictive power, it enables the formulation of testable scientific hypotheses about the nature of the classification. Conversely, in cases where GNNs exhibit weak predictive power, it helps identify and understand potential misunderstandings within the model.

1.1 General Notation

- Any graph G can be describe by X, A, E . The node feature, matrix, edge feature matrix, and adjacency matrix respectively.

- Let $X = [X_c, X_d]$, where X_c is the subset of continuous node features and X_d is the subset of one-hot discrete node features.
- Let $E = [E_c, E_d]$, denoted in the same manner.
- Let n represent the number of nodes in the graph and v represent the number of edges.
- For any graph, let ν denote the set of node and \mathcal{E} denote the set of edges.
- Let $\text{feat}_{(\cdot)}$ denote the number of features or columns in the the corresponding feature matrix.
- A is a binary $n \times n$ matrix where $A[i, j] = 1$ indicates that an edge exists between nodes labeled i and j .
- Let $\text{explaineer}(G; \Omega) = h_G^{(1)}, \dots, h_G^{(L)}, \rho_G$ be an L layer GNN model with parameters Ω that we would like to explain.
- Let \hat{Y}_G be the predicted class label for graph G predicted from $\text{explaineer}()$.
- Unless otherwise noted, assume any GNN model discussed is a graph-level classification model.

2 Analysis of Core Papers

2.1 GNNInterpreter (Wang and Shen 2024)

GNNInterpreter (Wang and Shen 2024) is a method for generating model-level explanations of GNN graph classification models. In general, explanation methods serve to elucidate which features within the data influence disparate predictions. These methods typically fall into two categories: instance-level and model-level. Instance-level explanations aim to unveil the model’s rationale behind a particular prediction. In domains such as image and text analysis, a prevalent approach involves masking or perturbing the instance and assessing the impact on the model’s prediction. On the other hand, model-level explanations seek to understand how a model generally distinguishes between classes. In image and text analysis, for instance, one common technique involves treating the input as a trainable parameter and optimizing the model’s prediction towards a specific class. Consequently, the resulting optimized input comprises a set of features strongly associated with the targeted class. GNNInterpreter provides model level explanations for GNN

in this manner. Formally, GNNInterpreter tries to learn the graph generating distribution for each class. GNNInterpreter works by optimizing the parameters of a generic graph generating distribution to produce samples that closely match the explaine’s understanding of the targeted class.

Graph generating distributions are hard to specify because there can be discrete and continuous elements of X , E and A . Furthermore, the interactions between these matrices can be complex. The authors tackle these issues by making two simplify assumptions. First, they assume that every graph is a *Gilbert random graph* (Gilbert 1959), where every possible edge as an independent fixed probability of occurring. Second, the author assume that the features of every node and edge are independently distributed. The justification of the first assumption is that the other common types of random graphs are not suitable for this application. Erdo-Renyi random graphs (ERDdS and R&wi 1959) have a fixed number of edges, which limits the diversity of explanations, Rado random graphs (Rado 1964) are infinite in size, and the random dot-product graph model is just a generalization of Gilbert’s model. The second assumption is justified by the fact that the parameters of the independent distributions will be updated jointly using the *explaine* model. Therefore, the *explaine*’s understanding of the latent correlation structure should be contained in the final estimates.

Although the graphs discuss in this paper only contain discrete features, X_c and E_c can be sampled from any continuous distribution that can be expressed as a location-scale family. Separating the stochastic and systematic components is necessary for gradient based optimization. It is commonly known as the *reparametrization trick*. The discrete feature matrices, (X_d, E_d, A) , need to be sampled from a continuous distribution for gradient based optimization, but the distribution has to have sampling properties close to a discrete distribution. The author assume that the true underlying distribution for every discrete node and edge feature is *categorical*. The categorical distribution is also know as the multi-bernoulli, where every sample has a fixed probability of being in one of the discrete categories. Let θ_{cat} represent the associated vector of un-normalized or relative probabilities, where each entries is > 0 . Then,

$$\operatorname{argmax} \log \theta_{\text{cat}} + \text{Gumbel}(0, 1) \sim \text{Cat}(\theta_{\text{cat}}) \quad (1)$$

The intuition is that the Gumbel distribution, which is the density of the maximum order statistic of i.i.d. standard normals, makes it a good candidate for modeling the winning or maximum probability category. Adding Gumbel noise to the logits should maintain the true relative proportions, but enough skewness such that every category has some probability of having the maximum noised logit. A proof of equation 1 is provide in section Section 4.3. Approximating the argmax function with the Softmax function allows for approximate categorical sampling that is differentiable w.r.t. θ_{cat} . The associate inverse uniform CDF sampling formula below is referred to as the *Gumbel-Softmax trick* or the *concrete distribution* (Maddison, Mnih, and Teh 2017).

$$\text{Softmax} \left(\frac{\log \theta_{\text{Cat}} - \log(-\log \epsilon)}{\tau} \right), \quad \epsilon \sim U[0, 1]. \quad (2)$$

τ is a hyperparameter that controls the degree of relaxation. Smaller value of τ approximate the discrete sampling better, but can result in numerical issues. The adjacency matrix can be sampled in a similar manner since the Bernoulli is just a special case of the categorical.

$$\text{sigmoid} \left(\frac{\log(\theta_A/(1 - \theta_A)) + \log \epsilon - \log(1 - \log \epsilon)}{\tau} \right), \quad \epsilon \sim U[0, 1], \quad (3)$$

where θ_A is an $n \times n$ matrix with each $[i, j]$ entry representing the relative probability that the ij edge exists. Equation 3 samples from the *binary concrete distribution* (Maddison, Mnih, and Teh 2017). Taken together, let

$$G_{\text{gen}} \sim \text{gen}(\Theta)$$

notate the combined graph generating distribution, where Θ is the set of all parameters from the independently sampled distributions.

An obvious objective is to maximize the likelihood that the *explainee* model predicts a sampled graph to be a member of the target class. Let $\tilde{\rho}$ denote the desired predicted probability vector. Then the above objective can be expressed as:

$$\mathcal{L}_{\text{pred}}(\Theta \mid G_{\text{gen}}) = \mathbb{E}_{G_{\text{gen}}} \text{CrsEnt}(\text{explainee}(G_{\text{gen}}), \tilde{\rho}) \quad (4)$$

While the above objective enforces a desirable property, it fails to be restrictive enough to generate realistic graphs. This is because the final prediction, $\rho_{G_{\text{gen}}}$ is compute using only final embeddings, $h_{G_{\text{gen}}}^{(L)}$. Normally $h_{G_{\text{gen}}}^{(L)}$ contains all the needed information from the graphs structure; however, the generation scheme allows the feature distribution to be optimized directly. This means that an explanation can ignore the graph structure and optimize towards the desired final embeddings. It is dangerously common for nonsensical features and structures at the graph level to end up on the desired side of the decision boundary. Empirically, the authors found that even completely random graphs can produce confidence and consistent predictions.

Dataset [Method]	Predicted Class Probability by GNN		Training Time Per Class
Is_Acyclic [XGNN]	Cyclic 0.076 ± 0.000	Acyclic 0.927 ± 0.000	45 s
Is_Acyclic [Ours]	Cyclic 0.999 ± 0.001	Acyclic 1.000 ± 0.000	20 s
Is_Acyclic [Random]	Cyclic 0.143 ± 0.155	Acyclic 0.857 ± 0.155	-
MUTAG [XGNN]	Mutagen 0.986 ± 0.057	Nonmutagen 0.991 ± 0.083	128 s
MUTAG [Ours]	Mutagen 1.000 ± 0.000	Nonmutagen 1.000 ± 0.000	12 s
MUTAG [Random]	Mutagen 0.068 ± 0.251	Nonmutagen 0.932 ± 0.251	-
ColorConsistency [Ours]	Consistent 0.968 ± 0.110	Inconsistent 1.000 ± 0.000	58 s
ColorConsistency [Random]	Consistent 0.017 ± 0.118	Inconsistent 0.983 ± 0.118	-

Figure 1: An abridged copy of Table 5 from Wang and Shen (2024).

For example, Figure 1 shows that random graphs had an average predicted probability of being non-mutagenic in the MUTAG dataset (Debnath et al. 1991) if 93.2%. In order to mitigate this issue, the authors proposed additional minimizing the cosine distance between the average embedding of all the observed graph from the targeted class, $\bar{h}_{G_c}^{(L)}$, and the embedding of the generated explanation:

$$\mathcal{L}_{\text{embed}}(\Theta \mid G_{\text{gen}}) = \mathbb{E}_{G_{\text{gen}}} \text{CosDist} \left(\bar{h}_{G_c}^{(L)}, h_{G_{\text{gen}}}^{(L)} \right). \quad (5)$$

Additionally, the author wanted to encourage sparsity for ease of interpretation. This was done by employing an L_1 , L_2 , and a budget penalty on the edge probabilities:

$$\mathcal{L}_{\text{spars}}(\theta_A) = \|\theta_A\|_1 + \|\theta_A\|_2 + \text{softplus}(\text{sigmoid}\|\theta_A\|_1 - B)^2, \quad (6)$$

where B is the expected maximum number of edge for generated explanation graphs. Connectivity is another desirable property as it ensures a cohesive explanation. To encourage connectivity the author minimize the *KL-Divergence* between edge probabilities that share a common node.

$$\mathcal{L}_{\text{conn}}(\theta_A) = \sum_{i \in \nu} \sum_{j, k \in \mathcal{E}(i)} D_{KL}(\text{sigmoid}(\theta_A[i, j]) \parallel \text{sigmoid}(\theta_A[i, k])), \quad (7)$$

where $\mathcal{E}(i)$ is the set of edges that connect to node i .

The final generator model is trained by sampling $G_{\text{gen}} \sim \text{gen}(\Theta)$ and then iteratively updating Θ via gradient descent on the full loss:

$$\begin{aligned} \mathcal{L}_{\text{GNNInterpreter}}(\Theta \mid G_{\text{gen}}) = & \lambda_1 \mathcal{L}_{\text{pred}}(\Theta \mid G_{\text{gen}}) + \lambda_2 \mathcal{L}_{\text{embed}}(\Theta \mid G_{\text{gen}}) + \\ & \lambda_3 \mathcal{L}_{\text{spars}}(\Theta \mid G_{\text{gen}}) + \lambda_4 \mathcal{L}_{\text{conn}}(\Theta \mid G_{\text{gen}}) \end{aligned} \quad (8)$$

GNNInterpreter achieves remarkable accuracy on most target classes, with most intervals in Figure 2 being tight and very close to 1. This indicates that the generated examples are almost always classified as the targeted class. The explanations for the house motif and the lollipop shape perform worse in terms of predictions. Although the authors critique the use of predictions as the sole objective, they do not employ any other quantitative metrics to evaluate the validity of their explanations.

Dataset [Method]	Predicted Class Probability by GNN				Training Time Per Class
MUTAG [XGNN]	Mutagen	Nonmutagen			128 s
	0.986 ± 0.057	0.991 ± 0.083			
MUTAG [Ours]	Mutagen	Nonmutagen			12 s
	1.000 ± 0.000	1.000 ± 0.000			
Cyclicity [Ours]	Red Cyclic	Green Cyclic	Acyclic		49 s
	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000		
Motif [Ours]	House	House-X	Complete-4	Complete-5	83 s
	0.918 ± 0.268	0.999 ± 0.032	1.000 ± 0.000	0.998 ± 0.045	
Shape [Ours]	Lollipop	Wheel	Grid	Star	24 s
	0.742 ± 0.360	0.989 ± 0.100	0.996 ± 0.032	1.000 ± 0.000	

Figure 2: A copy of the quantitative modeling results from Table 2 in Wang and Shen (2024).

Qualitatively, Figure 3 displays limitations in terms of realism. For instance, for the mutagen class, the explanation correctly identifies the importance of the N02 group; however, the generated graph is unrealistic and might not even be chemically feasible. Additionally, the non-mutagen example




























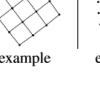

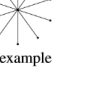
Dataset [Method]	Generated Model-Level Explanation Graphs							
MUTAG [XGNN]	Mutagen		Nonmutagen					
					<div> ● C ● N ● O ● F </div> <div> ● I ● Cl ● Br </div>			
MUTAG [Ours]	Mutagen		Nonmutagen					
					<div> ● C ● N ● O ● F </div> <div> ● I ● Cl ● Br </div>			
Cyclicity [Ours]	Red Cyclic		Green Cyclic		Acyclic			
								
Motif [Ours]	House		House-X		Complete-4		Complete-5	
								
Shape [Ours]	Lollipop		Wheel		Grid		Star	
								

Figure 3: A copy of the qualitative modeling results from Figure 1 in Wang and Shen (2024).

lacks clear patterns or identifiable structures. The explanations for the Cyclicity dataset, as well as the Wheel and Grid classes, do not seem to be from their respective underlying data distributions. In summary, GNNInterpreter offers a method for generating example graphs that would be classified as a targeted class by a GNN model, without requiring domain-specific rules. However, optimizing for predictions and even embeddings alone does not seem to be a sufficient objective for producing in-distribution graphs. While the authors correctly note that focusing on predictions can lead to unrealistic graphs, they also inadvertently show that optimizing embeddings alone is not necessarily sufficient either.

2.2 D4Explainer (Chen et al. 2023)

D4Explainer (Chen et al. 2023), or in-Distribution GNN Explanations via Discrete Denoising Diffusion, directly addresses the realism of generated graphs in model-level explanations by using a diffusion based generator model. In the image domain, diffusion models have proven to produce more realistic images than other generative AI methods, such as Generative Adversarial Networks (GANs). These models work by iteratively adding noise to an observation until it becomes pure noise. A denoising model is then trained to predict the noise added at any step. New observations

can be generated by reversing this process, passing pure noise through the denoising model, a technique known as reverse sampling. Additional objectives, such as a desired class label, can be added to the loss function to generate samples with certain properties.

The authors here are focused on *discrete structural diffusion*. D4Explainer generate example graph by noising and denoising the adjacency matrices of observed graphs. The sampled graphs have the same features, but different structures. The process of gradually adding noise to the input data is called *forward diffusion*. During forward diffusion, random noise is added iteratively until the data becomes pure noise at the final iteration. This ensures that the denoising model can start with pure noise. Forward diffusion is usually a Markov process. Let $t \in [0, T]$ denote the current iteration. Let β_t be the common probability that any edge changes state at time step t . $(\beta_1, \dots, \beta_T)$ is known as the variance schedule and is a set hyperparameter. Let A_t be a one-hot encoded version of the t^{th} noised adjacency. Then the forward diffusion process can be expressed as:

$$A_t[i, j] \sim q(A_t[i, j] \mid A_{t-1}[i, j]) = \text{Cat}(A_{t-1}[i, j] \cdot Q_t) \quad (9)$$

or

$$A_t[i, j] \sim q(A_t[i, j] \mid A_0[i, j]) = \text{Cat}\left(A_0[i, j] \prod_{i=1}^t Q_i\right) \quad (10)$$

where,

$$Q_t = \begin{bmatrix} 1 - \beta_t & \beta_t \\ \beta_t & 1 - \beta_t \end{bmatrix}$$

the t^{th} element-wise transition matrix. Even though only the adjacency matrix is being noised, the authors still wanted to use all of the available information during the *backwards diffusion* process. Thus the denoising model was parameterized as:

$$p(A_0 \mid A_t, t, X_0, E_0; \Omega) \quad (11)$$

where Ω is the set of trainable parameters. The authors employed a *Provably Powerful Graph Network* (PPGN) (Maron et al. 2020) as their denoising architecture; however, they have added an additional neural network to learn the time or noise level effect. Like most diffusion models, the primary objective is to minimize the distance between the predicted denoised observation and the

original. The authors have added an additional weight term to focus the model on noisier or more difficult training examples. The core loss function is expressed as:

$$\mathcal{L}_{dist}(\Omega \mid A_0) = \sum_{t=1}^T \left(1 - 2\bar{\beta}_t + \frac{1}{T}\right) \mathbb{E}_{\hat{A}_0} \text{CrsEnt}(A_0, \hat{A}_0), \quad (12)$$

where $\hat{A}_0 = p(A_0 \mid A_t, t, X_0, E_0; \hat{\Omega})$, $A_t[i, j] \sim q(A_t[i, j] \mid A_0[i, j])$, and

$$\bar{\beta}_t = \frac{1}{2} - \frac{1}{2} \prod_{i=1}^t (1 - 2\beta_i),$$

the cumulative transition probability. After the denoising model has been trained, model-level explanations are produced by iterative sampling candidate adjacency structures for an observed set of graph features and selecting the candidate with the highest probability of being from the targeted class. This is a multi-step procedures, where at each step A_t is denoised to k candidates $A_0, 1 \dots k$. The candidate with the best predicted probability is then noised to a level of $t - 1$ and the process is repeated. The pseudocode is reproduced below.

Algorithm 1 D4Explainer Model-level Explanation Reverse Sampling Algorithm

Require: $\hat{\Omega}$: trained denoising parameters; $q(A_t \mid A_0)$: forward diffusion process.

Input: N: maximum number of nodes; T: maximum noise level; K: number of candidates per iteration; $\tilde{\rho}$ targeted prediction vector; (X, E) : node and edge features.

Output: \hat{A} : adjacency matrix for model-level explanation.

Sample $A_T[1 : n, 1 : n] \sim \text{Bernoulli}(0.5)$

for t in T to 1 **do**

 Sample candidates $\{\hat{A}_{0,k} \sim p(A_t, t, X, E; \hat{\Omega}) : k \in 1, \dots, K\}$

 Select the best candidate $\underset{j \in 1, \dots, K}{\text{argmin}} \text{CrsEnt}(\text{explaine}(G = (X, A_{0,j}, E)), \tilde{\rho})$

 Sample $A_{t-1}[1 : n, 1 : n] \sim q(A_{t-1}, A_{0,j})$

end for

return A_0

To assess how well a sampled explanation matches the observed graph distribution, the authors compute various maximum mean discrepancy (MMD) statistics. MMD is a general technique used to measure the distance between two data distributions based solely on observed data. The distributions are estimated using *kernel density estimation*, and then the distance between their means is compared. As long as the chosen kernel is *characteristic*, an MMD statistics defined as the l2 distance between *kernel mean embeddings* is representative of the distance between the true

distributions (Gretton et al. 2012). The *Gaussian Earth Mover’s Distance kernel* was selected here. The author use MMD statistics to compare the degree, clustering, and spectrum distributions of the generated explanation against the observed data. The degree distribution of a graph indicates how frequently nodes have different numbers of connections, reflecting the overall connectivity pattern. The clustering coefficient of a node measures the proportion of the node’s neighbors that are also connected to each other, representing local clustering. The spectrum distribution, which is the distribution of eigenvalues of the adjacency matrix or Laplacian matrix, provides insights into the graph’s structural characteristics and dynamic properties. Similar to GNNInterpreter, the author of D4Explainer also value the sparsity of their generate example. They measure the density of a graph as the number of present edges divided by the number of possible edges or

$$\text{Density} = |\mathcal{E}|/|\nu|^2. \quad (13)$$

In additional to model-level explanations, D4Explainer can provide *counterfactual explanations* by adding a term to the loss function. The authors define a counterfactual explanation for a particular observed graph to be G^c such that $\hat{Y}_{G^c} \neq \hat{Y}_G$, but the difference between G^c and G is minimal. \mathcal{L}_{dist} (12) ensures that the G^c generated does not stray too far from the observed graph distribution, and adding \mathcal{L}_{CF} minimize the probability that the generated graph is classified the same as the input.

$$\mathcal{L}_{CF}(\Omega \mid A_0) = \mathbb{E}_{\hat{A}_0} - \log(1 - \rho_{G^c}[\hat{Y}_G]), \quad (14)$$

where $G = (X, A, E)$ is an observed graph, $G^c = (X, \hat{A}_0, E)$, and $\rho_{G^c}[\hat{Y}_G]$ denote the probability that G^c belongs to the same class as G . To quantify the quality of generated counterfactual explanations, the author report *Counterfactual Accuracy*, *Fidelity*, and *modification ratio*. CF-ACC measure the proportion of G^c s that have different predicted labels than there associated observed graph.

$$\text{CF-ACC} = \mathbb{E}_{\hat{A}_0} I(\hat{Y}_{G^c} \neq \hat{Y}_G) \quad (15)$$

Fidelity measures the difference in the predicted probability of G and G^c with respected to the

original class label.

$$\text{Fidelity} = \mathbb{E}_{\hat{A}_0} \rho_G[\hat{Y}_G] - \rho_{G^c}[\hat{Y}_G] \quad (16)$$

Finally, modification ratio measures difference in edges between G^c and G relative to the size of the original graph.

$$\text{MR} = \mathbb{E}_{\hat{A}_0} \frac{|\sum_{i,j} A[i,j] - \hat{A}_0(i,j)|}{\sum_{i,j} A[i,j]} \quad (17)$$

Similar to GNNInterpreter, the author report the mean predicted probability for the targeted class. Using a maximum of 6 nodes, the author were able to achieve a mean of 0.832 on the MUTAG dataset. Since the counterfactual explanation are generated on a per observation basis, the author’s compared their method to other popular instance-level explanation methods.

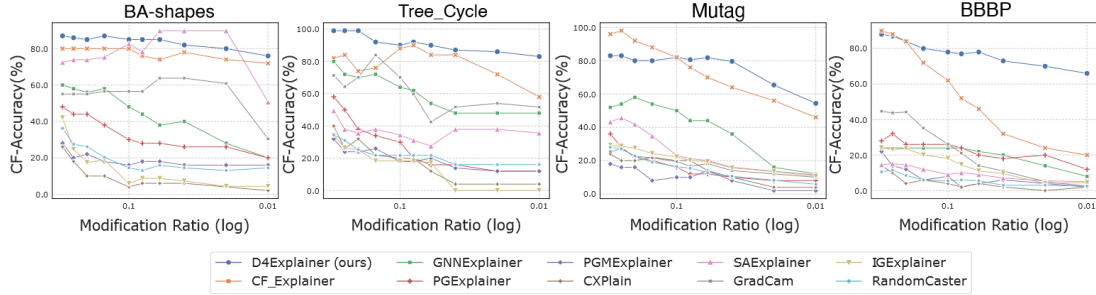


Figure 4: Comparison of the CF-ACC relative to MR of various methods and datasets. A copy of Figure 4 from Chen et al. (2023).

Figure 4 displays that D4Explainer can flip the prediction on roughly 80% of observed graphs at most modification levels. Missing from the plot is a variance measure. It is possible that the counterfactual accuracy varies more between different data splits or hyperparameter than the other methods. Figure 5 indicates that the D4Explainer generated graphs tend to have feature distributions that are closer to the original dataset; however, there might be a hidden class bias. The distributions might only be close for the majority class, skewing the averages.

Models	Mutag				BBBP				NCII			
	Deg.	Clus.	Spec.	Sum.	Deg.	Clus.	Spec.	Sum.	Deg.	Clus.	Spec.	Sum.
RandomCaster	0.1593	0.0247	0.0417	0.2257	0.1693	0.0072	0.0397	0.2162	0.1847	1.9769	0.0404	2.2020
GNNEExplainer	0.1614	<u>0.0002</u>	0.0409	0.2025	0.1615	0.0002	0.0395	0.2012	0.1577	0.0005	0.0405	0.1987
SAExplainer	0.0940	0.0032	0.0412	0.1384	0.1594	0.0032	0.0402	0.2028	0.189	0.0002	0.0408	0.2300
GradCam	0.1122	0.0083	0.0416	0.1621	<u>0.0699</u>	0.0026	<u>0.0384</u>	<u>0.1109</u>	0.1638	0.0003	0.0404	0.2045
IGExplainer	0.1292	0.0000	0.0411	0.1703	0.0908	0.0000	0.0394	0.1302	0.4288	0.0002	0.0398	0.4688
PGExplainer	0.1475	<u>0.0002</u>	0.0418	0.1895	0.2014	0.0018	0.0403	0.2435	0.1937	0.0000	<u>0.0396</u>	0.2333
PGMEExplainer	0.1800	<u>0.0002</u>	0.0419	0.2221	0.1916	0.0003	0.0403	0.2322	0.2199	0.0000	0.0404	0.2603
CXPlain	0.1734	1.2706	0.0417	1.4857	0.1768	<u>0.0001</u>	0.0394	0.2163	0.1629	<u>0.0001</u>	0.0404	0.2034
CF-GNNEExplainer	0.1172	0.0000	<u>0.0380</u>	0.1552	0.0870	<u>0.0001</u>	0.0393	0.1264	<u>0.1224</u>	<u>0.0001</u>	0.0404	<u>0.1629</u>
D4Explainer	0.1172	0.0000	0.0244	<u>0.1416</u>	0.0530	0.0000	0.0331	0.0861	0.1006	0.0000	0.0353	0.1359

Figure 5: MMD statistic for various methods and datasets copied from Table 2 in Chen et al. (2023).

2.3 ProtGNN (Zhang et al. 2021)

Up until now, we have been discussing post-hoc explanation methods; however, Rudin (2019) raises two important concerns with this approach. First, parity in the predictions does not guarantee parity in the reasoning. Explanations may bear strong predictions, but for distinct and sometimes nonsensical reasons. Recall that Figure 1 showed that completely random graphs can illicit confident predictions. Second, feature importance indicates that a feature is significant to the model, but it does not reveal how the model uses the feature. For instance, a substructure might lead to a prediction due to a strong relation with the target class, or it could be important because of a strong negative relation with other classes. ProtGNN (Zhang et al. 2021) uses the prototype modeling paradigm to create a GNN that is *interpretable*. Prototype based models make prediction by comparing new inputs to exemplar cases or *learned prototypes* of each class. This results in an interpretable model because, as long as the prediction function applied to the prototype similarity scores is simple, the reasoning for any given prediction will be clear. Intuitively, this can be thought of as using a black-box model to engineer features, which are then passed to a straightforward white-box model.

ProtGNN initializes m random prototype vectors for each of the C classes. Let p_k denote a prototype vector and P_{Y_G} denote the set of prototypes vectors assigned to the ground truth label of G , Y_G . Now, let $f(G; \Omega_f) = \hat{p}_G$ be a graph encoder function that maps any graph into the prototype vector space, where Ω_f is the set of trainable parameters. f is generally one of the standard GNN architectures. Since ProtGNN was primarily designed to be a graph classification model, the author optimized for predictive power using:

$$\mathcal{L}_{\text{prot}}(\Omega_f, \Omega_\psi, p_1, \dots, p_{mC} \mid G) = \mathbb{E}_G \text{CrsEnt}[\psi(\text{sim}(\hat{p}_G, p_1), \dots, \text{sim}(\hat{p}_G, p_{mC}); \Omega_\psi), Y_G] \quad (18)$$

where ψ is any simple prediction model with trainable parameters Ω_ψ and sim is any vector similarity function. ProtGNN uses a full connected layer with a softmax output activation for ψ , which is similar to a multinomial regression model. To address similar out-of-distribution concerns discussed earlier, the authors include three additional loss terms as constraints. The cluster loss ensure that each embedding is close to at least one prototype assigned to its ground truth class:

$$\mathcal{L}_{\text{clst}}(\Omega_f, \Omega_\psi, p_1, \dots, p_{mC} \mid G) = \mathbb{E}_G \min_{p_k \in P_{Y_G}} \|\hat{p}_G - p_k\|_2^2. \quad (19)$$

A separation loss ensures that embeddings are far from the prototypes of other classes:

$$\mathcal{L}_{\text{sep}}(\Omega_f, \Omega_\psi, p_1, \dots, p_{mC} \mid G) = -\mathbb{E}_G \min_{p_k \notin P_{Y_G}} \|\hat{p}_G - p_k\|_2^2. \quad (20)$$

Finally, the diversity loss encourages each prototype within a class to learn different information:

$$\mathcal{L}_{\text{div}}(\Omega_f, \Omega_\psi, p_1, \dots, p_{mC} \mid G) = \sum_{k=1}^C \sum_{p_i \neq p_j \in P_k} \max(0, \cos(p_i, p_j) - s_{\max}) \quad (21)$$

where s_{\max} is a set similarity threshold.

The learned prototypes are latent embedding vectors that are not directly interpretable. To address this issue, the authors use a Monte Carlo Tree Search (MCTS) algorithm (Coulom 2006) to identify the subgraph within the observed graphs of the prototype’s class that has an embedding closest to the given prototype. Specifically, the author use the MCTS to select an optimal sequence of pruning actions. Let $\mathcal{G}_{0, i, \dots, j}$ represent the subgraph resulting from the sequence of pruning actions $\text{act}_i, \dots, \text{act}_j$, where \mathcal{G}_0 , the root of the search tree, is an observed graph. The MCTS algorithm proceeds in 4 core steps. First, an existing node in the search tree $\mathcal{G}_{0, i, \dots, j}$ is *selected*. Second, the chosen node is *expanded* by adding a child node $\mathcal{G}_{0, i, \dots, j, k}$ resulting from action act_k . To limit the search space, the authors restrict act_k to be the removal of a peripheral node with minimum degree. Third, the reward for the sequence of actions i, \dots, j, k is estimated via a Monte Carlo *simulation* where additional random actions are taken until a subgraph of the desired size is

produced. The sampled embeddings are then compared to the prototype vector and the similarities are averaged. Finally, the child node’s estimated score is then *backpropagated* to update the estimated scores of all its parent nodes. After a fixed number of iterations through the observed graphs from the prototype’s class, the decision path with the highest resulting score is selected to produce the ultimate prototype projection. In addition to finding the closest subgraph in the observed class, the authors also suggest a method for finding similar subgraphs within each input. This is done by training a neural network to predict an edge mask conditional on a prototype vector:

$$\hat{A}[i, j] = \text{nn}(G, p_k; \Omega_{\text{nn}}). \quad (22)$$

The resulting induced subgraphs’ embeddings are then optimized to be as close as possible to the given prototype:

$$\mathcal{L}(\Omega_{\text{nn}}, \Omega_f, p_k \mid G) = -\mathbb{E}_G \text{sim}(\hat{p}_G, p_k), \quad (23)$$

thus producing a similar observed subgraph per learned prototype. The authors denote the version of the model that utilizes this *conditional subgraph sampling module*, in addition to the above losses, as **ProtGNN+**.

ProtGNN achieves similar or better graph classification accuracy than state of the art models across 5 standard benchmark datasets (see Figure 6).

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

Figure 6: A copy of the quantitative modeling results report in Table 1 of Zhang et al. (2021).

Figure 6 also indicates that ProtGNN is robust to the choice of embedding model as there seems to be benefits across various GNN architectures. The authors do note that the computational complexity of ProtGNN can be significantly higher than a standard GNN model, taking roughly 5 times longer to train. Most of this complexity is attributable to the use of the MCTS. On the other hand, ProtGNN allows for easier and clearer inference. Figure 7 demonstrates that the output of

ProtGNN is similar to a simple regression model. The prototypes can be thought of as features, the similar subgraphs and similarity scores as the observed feature values, and the class connections as the marginal effects on the prediction.

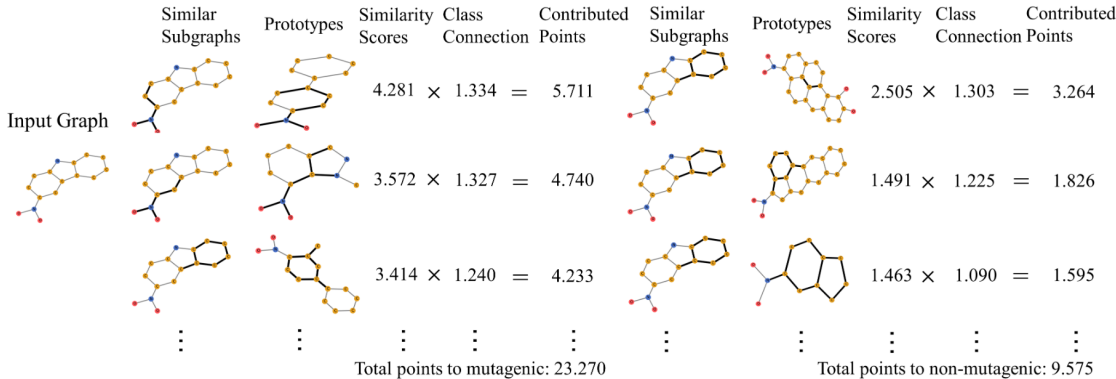


Figure 7: An example ProtGNN+ output for the MUTAG dataset taken from Figure 3 in Zhang et al. (2021).

Since prototypes from every class are used for the classification, ProtGNN can also be used for counterfactual type analysis. For any given predictions, researcher can determine why a certain classification was made and why the other classes were not selected. Figure 7 display both a strong similarity to the mutagenic prototypes as well as a weaker relation to the non-mutagenic prototypes.

3 Synthesis of Core Papers

4 Technical Details

4.1 Methodology

4.2 Results

4.3 Proofs

Proof of Equation 1: WLOG assume we want to sample for a categorical distribution $\text{cat}(\theta_{\text{cat}})$ with a set of W categories each with a probability

$$\pi_{\omega} = \frac{\theta_{\text{cat}}[\omega]}{\sum_{i \in W} \theta_{\text{cat}}[i]}.$$

Define random variable

$$D = \operatorname{argmax}_{i \in W} \log \theta_{\text{cat}} + \text{Gumbel}(0, 1) = \operatorname{argmax}_{i \in W} \log \theta_{\text{cat}}[i] + \text{Gumbel}(0, 1)[i],$$

here $\text{Gumbel}(0, 1)[i]$ denotes the i^{th} i.i.d. sample from the associated Gumbel. In order for D to be a true categorical random variable, $\Pr[D = \omega]$ need to be π_ω . $D = \omega$ if and only if $\log \theta_{\text{cat}}[\omega] + \text{Gumbel}(0, 1)[\omega] > \log \theta_{\text{cat}}[i] + \text{Gumbel}(0, 1)[i] \forall i \in W \setminus \omega$. Now, let M_i denote a random variable that follows a $\text{Gumbel}(\log \theta_{\text{cat}}[i], 1)$ distribution. Then,

$$\begin{aligned} \Pr[D = \omega] &= \mathbb{E}_{M_\omega} \prod_{i \in W \setminus \omega} \Pr(M_i < m_\omega) \text{ i.i.d location shifted Gumbel distributions.} \\ &= \mathbb{E}_{M_\omega} \prod_{i \in W \setminus \omega} \exp(-e^{\log \theta_{\text{cat}}[i] - m_\omega}) \text{ Gumbel CDF.} \\ &= \mathbb{E}_{M_\omega} \exp\left(-\sum_{i \in W \setminus \omega} e^{\log \theta_{\text{cat}}[i] - m_\omega}\right) \\ &= \int_{-\infty}^{\infty} \exp(\log \theta_{\text{cat}}[\omega] - m_\omega) \exp(-e^{\log \theta_{\text{cat}}[\omega] - m_\omega}) \cdot \exp\left(-\sum_{i \in W \setminus \omega} e^{\log \theta_{\text{cat}}[i] - m_\omega}\right) dm \\ &\quad \text{Gumbel PDF.} \\ &= \int_{-\infty}^{\infty} \exp(\log \theta_{\text{cat}}[\omega] - m_\omega) \exp\left(-\sum_{i \in W} e^{\log \theta_{\text{cat}}[i] - m_\omega}\right) dm \\ &= \int_{-\infty}^{\infty} \theta_{\text{cat}}[\omega] \exp(-m_\omega) \exp\left(-e^{-m_\omega} \sum_{i \in W} \theta_{\text{cat}}[i]\right) dm \\ &= \pi_\omega \sum_{i \in W} \theta_{\text{cat}}[i] \int_{-\infty}^{\infty} \exp(-m_\omega) \exp\left(-e^{-m_\omega} \sum_{i \in W} \theta_{\text{cat}}[i]\right) dm \text{ From the above definition of } \pi_\omega \\ &= \pi_\omega \sum_{i \in W} \theta_{\text{cat}}[i] \frac{\exp\left(-e^{-m_\omega} \sum_{i \in W} \theta_{\text{cat}}[i]\right)}{\sum_{i \in W} \theta_{\text{cat}}[i]} \Big|_{-\infty}^{\infty} \\ &= \pi_\omega \sum_{i \in W} \theta_{\text{cat}}[i] \frac{1}{\sum_{i \in W} \theta_{\text{cat}}[i]} = \pi_\omega. \end{aligned}$$

Reference: Huijben et al. (2022)

5 Future Directions

6 References

- Bai, Yunsheng, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. 2019. “Unsupervised Inductive Graph-Level Representation Learning via Graph-Graph Proximity.” <https://arxiv.org/abs/1904.01098>.
- Breiman, Leo. 2001. “Random Forests.” *Machine Learning* 45 (1): 5–32. <https://doi.org/10.1023/A:1010933404324>.
- Chen, Jialin, Shirley Wu, Abhijit Gupta, and Rex Ying. 2023. “D4Explainer: In-Distribution GNN Explanations via Discrete Denoising Diffusion,” no. arXiv:2310.19321 (October). <https://doi.org/10.48550/arXiv.2310.19321>.
- Coulom, Rémi. 2006. “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search.” In *Computers and Games*. <https://api.semanticscholar.org/CorpusID:16724115>.
- Debnath, Asim Kumar, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. 1991. “Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity.” *Journal of Medicinal Chemistry* 34 (2): 786–97. <https://doi.org/10.1021/jm00106a046>.
- ERDdS, P, and A R&wi. 1959. “On Random Graphs i.” *Publ. Math. Debrecen* 6 (290-297): 18.
- Gallicchio, Claudio, and Alessio Micheli. 2019. “Fast and Deep Graph Neural Networks.” <https://arxiv.org/abs/1911.08941>.
- Gilbert, E. N. 1959. “Random Graphs.” *The Annals of Mathematical Statistics* 30 (4): 1141–44.
- Gretton, Arthur, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. “A Kernel Two-Sample Test.” *Journal of Machine Learning Research* 13 (25): 723–73.
- Huijben, Iris A. M., Wouter Kool, Max B. Paulus, and Ruud J. G. van Sloun. 2022. “A Review of the Gumbel-Max Trick and Its Extensions for Discrete Stochasticity in Machine Learning,” no. arXiv:2110.01515 (March). <https://doi.org/10.48550/arXiv.2110.01515>.
- Maddison, Chris J., Andriy Mnih, and Yee Whye Teh. 2017. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables,” no. arXiv:1611.00712 (March). <https://doi.org/10.48550/arXiv.1611.00712>.
- Maron, Haggai, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2020. “Provably Powerful

- Graph Networks,” no. arXiv:1905.11136 (June). <https://doi.org/10.48550/arXiv.1905.11136>.
- Olah, Chris, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. 2020. “Zoom in: An Introduction to Circuits.” *Distill*. <https://doi.org/10.23915/distill.00024.001>.
- Rado, Richard. 1964. “Universal Graphs and Universal Functions.” *Acta Arithmetica* 9: 331–40. <https://api.semanticscholar.org/CorpusID:118279788>.
- Rudin, Cynthia. 2019. “Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead,” no. arXiv:1811.10154 (September). <https://doi.org/10.48550/arXiv.1811.10154>.
- Wang, Xiaoqi, and Han-Wei Shen. 2024. “GNNInterpreter: A Probabilistic Generative Model-Level Explanation for Graph Neural Networks,” no. arXiv:2209.07924 (February). <https://doi.org/10.48550/arXiv.2209.07924>.
- Xiao, Guanghua, Shidan Wang, Ruichen Rong, Donghan Yang, Xinyi Zhang, Xiaowei Zhan, Justin Bishop, et al. 2023. *Deep Learning of Cell Spatial Organizations Identifies Clinically Relevant Insights in Tissue Images*. Preprint. In Review. <https://doi.org/10.21203/rs.3.rs-2928838/v1>.
- Yuan, Hao, Haiyang Yu, Shurui Gui, and Shuiwang Ji. 2022. “Explainability in Graph Neural Networks: A Taxonomic Survey,” no. arXiv:2012.15445 (July). <https://doi.org/10.48550/arXiv.2012.15445>.
- Zhang, Zaixi, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. 2021. “ProtGNN: Towards Self-Explaining Graph Neural Networks,” no. arXiv:2112.00911 (December). <https://doi.org/10.48550/arXiv.2112.00911>.

7 Appendix