GNN-EGG: Graph Neural Network Explanations via Graph Generation

Art Taychameekiatchai *Southern Methodist University*

***Abstract*—Graph Neural Networks (GNNs) provide a means for modeling inherently graphical data, such as transportation, social, and molecular networks, but also for enhancing and reducing unstructured data, including text and images. A potential shortcoming is that their predictions are opaque — a black box — hindering broader adoption and refinement. In this paper, we propose a novel architecture agnostic algorithm for generating model-level post-hoc explanations of GNN based graph classifiers. The algorithm attempts to learn the data generating distribution for each class of graph the model can predict. While this idea is not new, we introduce a novel loss function aimed at reducing the "class lean problem" [1] and allowing for better comparisons of structural consistency. The primary contribution of this work is the use of a differentiable approximation to Graph Edit Distance (GED) in the loss function. This term enable us to ensure consistency in both the graph space and the embedding space for our representative examples. We demonstrate the the theoretical benefits of our loss function through a simulation study, and benchmark our algorithm using the MUTAG dataset [2].**

***Index Terms*—Graph Neural Networks, Interpretable AI**

I. Introduction

GRAPHS provide an incredibly flexible structure for modeling complex data relationships. 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 grammatically modifiers in text data. With the emergence of Graph Neural Networks (GNNs) graphs have become a popular choice for prediction and inference. Some important applications of graph classification include predicting chemical toxicity [2], classifying proteins [3], and even detecting cancer from pathology slides [4]. 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. The lack of strong inference methods leave GNNs more susceptible to adversarial attacks and undetected discrimination. Explanation methods also serve to direct modeling efforts by highlighting common structures of features that may be predictive in other applications.

II. Related Work

Explaining GNNs is a less explored area than methods for image and text models. The primary challenges lie in the discrete nature of the adjacency matrix, and possibly some of the node or edge features [5]. The discreteness of graph data prevents many of the gradient based methods that have been developed for images to be easily ported over. This issue can be overcome on an instance level because the prediction is continuous. These methods primarily work by making small changes to an input graph and then optimizing the change in the prediction. This can be achieve through masking or perturbations [5]. Problematically, these methods only explain a model's prediction for a particular observation, and are not reflective of the GNN's overall decision process. Model level explanations are typically done via generative methods [5]. The goal is to illuminate the decision-making processes of GNNs by generating illustrative instances for each possible prediction outcome. XGNN [6] use a reinforcement learning approach to generate example graphs iteratively. While this does allow for more granular structural checks, the computation time makes it infeasible for medium to large size graphs. Furthermore, defining rules to enforce requires specific domain expertise that might not be available. GNNInterpreter [1], on the other hand, uses continuously relaxed discrete distributions to generate the graph matrices independently. The authors argue that a strong *explainee* model should impose the correct correlation structure between the distributions as the parameters are optimized through back-propagation. This solves the speed issue, but can result in less structurally correct examples.

This work attempt to improve upon the interpretability and visualization of structurally consistency of GNNInterpreter by making main modification to the objective. First, we utilize multiple embedding levels of the *explainee* model in addition to the final layer. Second, we incorporate Graph Edit Distance (GED) in the object, as a visualization tool, and as a more human understandable measure of similarity between observed and generated graphs.

III. Methods

A. Notation

Let G be a graph with X, E, and A being the node feature matrix, edge feature matrix, and adjacency matrix respectively. Let $X = [X_c, X_d]$ and $E = [E_c, E_d]$ where c and d denote continuous and one-hot discrete features.

B. Generator Model

X_c and E_c can both be sampled from any function with the form:

$$f(z\ ;\ \Omega) : \mathbb{R}^Z \to \mathbb{R}^{dim}$$

and dim denotes the dimensionality of the matrix we want to generate. z could be sampled as pure random noise, but z could also be a function of the data. In most instances, we generate continuous feature matrices using a random vector

from a multivariate normal pass through a standard dense layer with linear activation.

X_d and E_d can both be sampled from the *concrete distribution* [7]. Using the reparameterization trick and continuous relaxation, we can sample one-hot categorical vectors based on two parameters θ_{Cat}, a trainable parameter vector of probabilities, and τ, a hyperparameter that controls the degree of relaxation (smaller value approximate the discrete sampling better, but can result in numerical issues).

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

Like z, θ_{Cat} could be a function of the data and other trainable parameters. One concrete distribution would be required per discrete feature.

Similarly, the adjacency matrix can be sampled from the *binary concrete distribution*:

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

Identical to GNNInterpreter, we set θ_A to be the same dimensionality as adjacency matrix, where each entry in θ is the probability that an edge exists between nodes i and j.

Algorithm 1 Graph Generation Algorithm

function GENERATOR(z_1, z_2, $\theta_{\text{Cat}X}$, $\theta_{\text{Cat}E}$, θ_A, τ)
  $X_c = f(z_1,\ \Omega_1)$
  $E_c = f(z_2,\ \Omega_2)$
  $X_d = []$
  for each discrete node feature i **do**
    $X_{di} \sim \text{Concrete}(\theta_{\text{Cat}X}[i],\ \tau[X,\ i])$
    $X_d = [X_d,\ X_{di}]$
  end for
  $E_d = []$
  for each discrete edge feature i **do**
    $E_{di} \sim \text{Concrete}(\theta_{\text{Cat}E}[i],\ \tau[E,\ i])$
    $E_d = [E_d,\ E_{di}]$
  end for
  $A \sim \text{BinaryConcrete}(\theta_A, \tau[A])$
  return X_c, X_d, E_c, E_d, A
end function=0

C. GNN Prediction & Embedding Loss

Most GNN use a message passing paradigm. Each node receives feature information from all of its neighbors which is then aggregated by a shared model. The distilled messages are then combined with the original node features to establish that layers node embeddings. Most architectures use several message passing layers followed by pooling and dense prediction layers. The idea is to use the graph's structure to engineer better node features to draw predictions. In order to train our generator model to sample graph of the desired class we minimize the cosine distance between the pooled embeddings of our generated graph as the average embeddings compute from all the observed graph of the target class. Let $\bar{h}_l$ denote this average for the l^{th} layer and h_l denote the embedding from the current generated sample. Let $g()$ denote the prediction layers and let t denote the desired prediction. The loss contribution of each generated example is define as,

$$\mathcal{L}_1 = \text{CrtEnt}(g(h_L),\ t) + \sum_{l=1}^{L} \text{CosDist}(h_l,\ \bar{h}_l) \tag{1}$$

D. Structural Consistency Loss & The Class Lean Problem

As mentioned above, GNN model use the graph structure to learn better node features from which to make predictions; however the generation scheme used here allows for the node features to be generated independently of the graph structure. This means that it is possible that the generator will ignore the graph structure and leap to training the final features as opposed to using the message passing. This problem is exacerbate be the fact generator models are primarily judged based on their prediction intervals. It is dangerously common for nonsensical features and structures at the graph level to end up on the desired side of the decision boundary. X. Wang and Shen (2023) [1] found that completely random graph produced significantly skewed predictions across 4 datasets. This phenomenon is what we are calling the "class lean problem". Using multiple embedding level should help to mitigate this issue by forcing the generated graph to be on the correct side of multiple embedding spaces, but this still lacks quantification. We could use the cosine distance between the embedding vector to compare different generators, but the raw value lacks a human interpretation. Furthermore, Steck, Ekanadham, and Kallus (2024) [8] found that similarities compute on latent embedding vectors can vary wildly from the ground truth based on how the embeddings are calculated.

Our proposed solution is to use an approximation to the graph edit distance (GED). GED measures the similarity between two graphs by counting the minimum number of edits required to make the graphs isomorphic (or subgraph isomorphic). This has the key advantage of being human interpretable. Finding the exact optimal GED is a known NP-complete problem, and most solution are not differentiable [9]. Fortunately, approximating GED as a quadratic assignment problem (QAP) solves both the time complexity and derivative issue [10]. Let G_1 and G_2 be graphs we wish to compute the GED between. QAP approximates this be solving the following optimization problem,

$$\mathcal{L}_2 = \max_M \text{vec}(M)\ K(G_1, G_2)\ \text{vec}(M)^T \tag{2}$$

here M is a $n_1 \times n_2$ node matching matrix and K is a $n_1 n_2 \times n_1 n_2$ quadratic affinity matrix that encode both node and edge similarity information. The efficient of QAP comes from the fact that when any two nodes are matched together, the above objective will implicitly account for the possible edges between the nodes. In order to compute GED, each entry of K should range from $[-1, 0]$, where -1 signals the need for a complete edit and 0 means no edits are required [10]. Specifics are in appendix A.

We also increase data utilization by weighting $\mathcal{L}_2$ in an manner akin to the triplet loss. Let G_{obs} and G_{gen} denote a observed and a generated graph respectively. Furthermore, let u denote an *uninformative prediction*.

$$\omega = [\text{CrtEnt}(t,\ u) - \text{CrtEnt}(t,\ \text{explainee}(G_{obs}))]^3 \tag{3}$$

$$\mathcal{L}_3 = \omega \cdot (\mathcal{L}_2(G_{obs}, G_{gen}) + \sum_{l=1}^{L} \text{CosDist}[h_l(G_{obs}),\ h_l(G_{gen})] \tag{4}$$

If a particular observation is predicted to be closer to the target class than to some uninformative or uniform random probability vector, $\omega > 0$, and the model will be encouraged to generate graphs closer to the observed data, and vice a versa. This allows us to use observed graph from all classes as well as their individual embeddings. The full loss function is thus,

$$\mathcal{L}(\Theta \mid G_{obs}) = \lambda_1 \cdot \mathcal{L}_1 + \lambda_2 \cdot \mathcal{L}_3 \tag{5}$$

where Θ is the set of all generator parameters and the λs are tuneable hyperparameters. Note that the loss is compute per batch of observed graphs. Further details on discrete gradient issues in appendix B.

IV. Results

A. Simulation Study

To access the theoretical benefits of the above loss function we conducted a simulation study using simplified graph generation. We generated 3 clusters of graph data using a normal distribution of the continuous node and edge features and a Bernoulli for the discrete node and edge features. We trained a GNN model using neural network convolutions message passing layers, that utilized all the features, on the first two clusters. For each of the 25 trials, we generated 50 training and 50 testing samples and evaluated the cosine embedding distance as well as the approximate GED.

V. Conclusion

The conclusion goes here.

Appendix A
Solving the Quadratic Assignment Problem

We use the following similarity metric

$$-0.25\,[1 - \theta(\text{Continuous Features})] + \\ -0.5\,[1 - \phi(\text{Discrete Features})]$$

where θ is normalized cosine similarity and [ϕ is unnormalized cosine similarity]. $1-\theta()$ will return 2 is the vectors are in opposite directions, 1 is the vectors are orthogonal, and 0 if they are pointing in the same direction; however, one-hot encoded vectors will never be in opposite directions, so the different types of feature must be weighted correctly. Computing affinity this way allows for K to be differentiable wrt generated graph features. Furthermore we use the reweighted random walks algorithm [11] to solve, which generates an M that is differentiable wrt to K.

Appendix B
Discrete Gradient Details

Most GNN explainee model will require further discretization of the generator's output. The typically transformation are $A' = I(A \geq 0.5)$, $X_d = \text{argmax}(X_d)$, $E_c = \text{argmax}(E_c)$; however, each one will prevent gradient backpropagation with respect to their corresponding parameters. This issue can be mitigated in two ways. First, $\mathcal{L}_2$ does not require that the explainee model be called, and therefore does not require discrete transformations. Specifically, we append θ_A as a edge feature and convert A to be fully connected. This allows $\mathcal{L}_2$ to explicitly learn the graph structure. There is no additional penalty using a fully connected adjancency, since deletion are given an edit cost of zero. Second, a REINFORCE [12] term can be added based on the likelihood of the generating distributions. For example if the A' transformation is required, for following term can be added to ensure training of the edge probabilities.

$$L_R = -\frac{l(\theta_A \mid G)}{\mathcal{L}}$$

l denotes the log-likelihood. In short, this term moves θ_A to move the likelihood of G proportional to the inverse of the loss ie if $\mathcal{L}(G)$ is small, θ_A will be moved to increase the likelihood of G and vice-a-versa. Empirically, we find that reinforcing only $\mathcal{L}_3$ as well as dropping the regularization terms present in GNNInterpreter lead to the best results.

Acknowledgment

The authors would like to thank...

References

[1] X. Wang and H.-W. Shen, "Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks," no. arXiv:2209.07924, Feb. 2023, arXiv:2209.07924 [cs]. [Online]. Available: http://arxiv.org/abs/2209.07924

[2] Y. Bai, H. Ding, Y. Qiao, A. Marinovic, K. Gu, T. Chen, Y. Sun, and W. Wang, "Unsupervised inductive graph-level representation learning via graph-graph proximity," 2019.

[3] C. Gallicchio and A. Micheli, "Fast and deep graph neural networks," 2019.

[4] G. Xiao, S. Wang, R. Rong, D. Yang, X. Zhang, X. Zhan, J. Bishop, C. Wilhelm, S. Zhang, C. Pickering, M. Kris, J. Minna, and Y. Xie, *Deep Learning of Cell Spatial Organizations Identifies Clinically Relevant Insights in Tissue Images*, Jul 2023. [Online]. Available: https://www.researchsquare.com/article/rs-2928838/v1

[5] H. Yuan, H. Yu, S. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," no. arXiv:2012.15445, Jul 2022, arXiv:2012.15445 [cs]. [Online]. Available: http://arxiv.org/abs/2012.15445

[6] H. Yuan, J. Tang, X. Hu, and S. Ji, "Xgnn: Towards model-level explanations of graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Aug 2020, p. 430–438, arXiv:2006.02587 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2006.02587

[7] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," no. arXiv:1611.00712, Mar. 2017, arXiv:1611.00712 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1611.00712

[8] H. Steck, C. Ekanadham, and N. Kallus, "Is cosine-similarity of embeddings really about similarity?" Mar. 2024, arXiv:2403.05440 [cs]. [Online]. Available: http://arxiv.org/abs/2403.05440

[9] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, and M. Vento, "A quadratic assignment formulation of the graph edit distance," no. arXiv:1512.07494, Dec. 2015, arXiv:1512.07494 [cs]. [Online]. Available: http://arxiv.org/abs/1512.07494

[10] R. Wang, Z. Guo, W. Pan, J. Ma, Y. Zhang, N. Yang, Q. Liu, L. Wei, H. Zhang, C. Liu, Z. Jiang, X. Yang, and J. Yan, "Pygmtools: A python graph matching toolkit," *Journal of Machine Learning Research*, vol. 25, no. 33, pp. 1–7, 2024. [Online]. Available: https://jmlr.org/papers/v25/23-0572.html

[11] M. Cho, J. Lee, and K. M. Lee, "Reweighted random walks for graph matching," in *Computer Vision – ECCV 2010*, ser. Lecture Notes in Computer Science, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer, 2010, p. 492–505.

[12] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," 1992.