GNN Interpretability Using Bayesian Inference

Shalin Patel

Advisor: Dr. Ritambhara Singh
Second Reader: Dr. Lorin Crawford

Division of Applied Mathematics and Department of Computer Science
Brown University
2023-04-21

Contents

GNN Interpretability Using Bayesian Inference

Shalin Patel[1,2]

[1] *Division of Applied Mathematics, Brown University*
[2] *Department of Computer Science, Brown University*

April 22, 2023

Abstract

Recently, GNNs have garnered a lot of attention as a flexible generalization of CNNs to operate on unstructured graphs of data. This has led to an explosion of different usecases as the general, unrestrictive nature of graphs allow researchers to train models that integrate many different forms of related data under one framework. While CNNs and other deep learning methods have had a good amount of research done on model interpretability, GNNs are still in a nascent stage for interpretation. The goal of this work is to look at the state of GNN Interpretability and suggest some new benchmarks for testing interpretation methods as well as providing a Bayesian Inference based approach to the problem. It will be seen that the Bayesian Inference approach performs better than the state of the art in GNN Interpretability and provides for a deeply introspectable tool to disect GNNs for the benefit of researchers using the model type.

1 Introduction

Graphs serve as a natural repository for information in many real-world applications ranging from social, informational, chemical, and biological domains [1]. Especially as data becomes more and more unstructured, graphs represent a flexible manner for storing and relating different nodes and their related features [2]. Indeed, graphs represent one of the most general mathematical structures for relating data and are seeing increasing use in modeling phenomenon such as social networks and gene regulatory networks [2, 3]. For the purposes of this work, given a set of vertices V, node features $\mathcal{X} : V \to \mathbb{R}^d$, a set of edges $E \subseteq V \times V$, and weights on the edges $W : E \to [0,1]$, we consider the graph $G = \{V, \mathcal{X}, E, W\}$. Additionally, we let the space of all graphs for a given set of vertices V be $\mathcal{G}$. Below in figure 1, a simple visualization of this definition can be seen for a cyclic graph of order 3.

1.1 Graph Neural Networks

To deal with the proliferation of graphs in computing and the need to construct models that consider graphs as a first-class member of the modeling process, a class of models known as Graph Neural Networks have emerged (GNN) with state of the art performance on a variety of classification and regression tasks [4]. Specifically, GNNs and their early iterations in GCNs took inspiration from CNNs that represented applying successive convolution operations on regular grids of information, such as images, to compose local features in the grid in to higher-level predictions [5]. At a high level, most GNN frameworks can be split into three steps `MSG`, `AGG`, and `UPD` representing a messaging step, aggregation step, and update step, respectively.

At a layer l in a GNN model ϕ, the update of the hidden state of the model occurs first by sending messages for all $(v_i, v_j) \in E$ as a function of the hidden state $\mathcal{H}_i^{l-1}$ and $\mathcal{H}_j^{l-1}$ as well as the weight $W_{ij} := W(v_i, v_j)$. Specifically, we have

$$m_{ij}^l := \texttt{MSG}(\mathcal{H}_i^{l-1}, \mathcal{H}_j^{l-1}, W_{ij})$$

Then, a GNN performs an aggregation step wherein it calculates an aggregate message for every vertex $v \in V$. Let $\mathcal{N}_k : V \times E \to \mathcal{P}(E)$ be a function that returns the edges in the k-hop neighborhood of a node. Then, we can formally write the aggregation step as

$$M_i^l := \texttt{AGG}(\{m_{ij}^l \mid v_j \in \mathcal{N}_k(v_i)\})$$

Then, finally, at each node, the GNN takes a nonlinear function (often a neural network of some sort) and applies it to this aggregated message M_i^l along with the hidden state $\mathcal{H}_i^{l-1}$

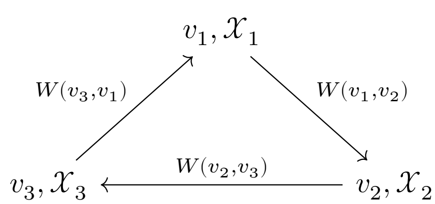

Figure 1: An example graph as related to the terminology laid out above

to get the new hidden state.

$$\mathcal{H}_i^l := \texttt{UPD}(\mathcal{H}_i^{l-1}, M_i^l)$$

When composed in layers, this forms a full Graph Neural Network. Note that in this framework, $\mathcal{H}_i^0 := \mathcal{X}_i$. Based on the task type, either node or graph classification in this work, further layers may be added on top of the final output. For example, in graph classification, it is often the case that the final node embeddings are concatenated and then run through an MLP to get a final classification for the whole graph [4].

The specific layers used in this work are Graph Convolution Layers also known as GCNs [6]. In the context of the framework above, the `MSG` sends weighted and normalized node features from the $l-1$st layer where the normalization is by the out-degree of the sending node and the weight of the message coming from $\mathcal{W}_{ij}$. In the `AGG` step, all of these messages are summed together. Finally, the `UPD` step applies a neural network, usually a trainable linear layer combined with a non-linear activation function to provide a non-linear update step.

1.2 Interpretation on GNNs

Given a GNN, it is natural in many fields such as computational biology to perform interpretation on the model in order to gain further insights. For example, in the case of RNA-seq data, a natural question is to determine important regulatory pathways between genes that could be related to eventual up or down regulation of a target gene [3]. One natural way to perform this task is to train a GNN on the RNA-seq data for a node-classification task and feed it a large graph G with many redundant edges. Given a GNN model ϕ with l layers and a target gene $v_i \in V$, we would like to determine $\mathcal{E}_i \subseteq \mathcal{N}_l(v_i, E)$ as well as $\mathcal{W}_i : \mathcal{E}_i \to [0,1]$ such that $\mathcal{E}_i, \mathcal{W}_i$ represent the most important subgraph for the model ϕ to perform its predictions for the input vertex v_i. While there are a few criteria for determining importance, we consider importance to be the maximal mutual information between the original model with its inputs and the same model with the estimated subgraph $\mathcal{E}_i, \mathcal{W}_i$. Written more formally, we wish to discover

$$\underset{\mathcal{E}_i, \mathcal{W}_i}{\arg\max}\ \texttt{MI}(\phi(v_i, \mathcal{X}, E, W), \phi(v_i, \mathcal{X}, \mathcal{E}_i, \mathcal{W}_i))$$

This is a computationally hard problem as a brute force search would take $O(2^{|E|})$ time even when discounting the weight array $\mathcal{W}_i$. In practice, discovering $\mathcal{E}_i$ is ignored and most of the importance discovery is done through learning a suitable $\mathcal{W}_i$ while letting $\mathcal{E}_i = E$.

However, because of the given task, and the various properties we would like to see in a reasonable interpretation of a GNN model, there are many routes that have been taken to interpret these models. As will be shown in §2, there have been a few non-bayesian attempts to solve this problem. The goal of this work is to analyze these methods and then suggest a new Bayesian method to solving the issue of searching for the best subgraph for a trained GNN to perform post-hoc analysis of importance. Furthermore, an added benefit of utilizing a Bayesian approach is that a full conditional distribution over the importance graph is learned giving researchers an even larger level of insight into their model that goes beyond just a simple edge mask.

1.3 Bayesian Inference

Recently, in deep learning literature there has been a rise in utilizing Bayesian methods to create Bayesian Neural Networks in order to provide uncertainty aware predictions [7]. While primarily concerned with giving estimates for failure modes and reducing overfitting within deep learning methods, the synthesis of Bayesian methods and deep learning models has imbued them with a greater sense of interpretability and introspectability. In the context of GNN interpretation, modeling the subgraph $\mathcal{E}_i, \mathcal{W}_i$ as a joint distribution allows for conditioning on certain edge weights and imbues the interpretations that are derived with a greater sense of introspectability. Figure 2 gives a good overview of the deep learning analogues for point estimate neural networks. In the same way, analogues will be utilized in the interpretation task in order to get the same benefits that have already been outlined in BNNs. In the Bayesian paradigm, a distribution $\mathcal{P}$ is treated as the belief in the occuance of a given event from the distribution rather than the limit of the frequencies of each event as in the frequentist scheme. Furthermore, prior beliefs are thought to inform posterior beliefs. In the context of interprebility this is important as the belief for a given interpretation is dependent on the domain that it is brought up in. For example, in social networks one may expect relatively dense explanations while in biology they would tend to be sparse [1] [3]. Generally speaking, given a hypothesis H representing the prior belief for the state of a system and some data D, the posterior probability $\mathcal{P}(H \mid D)$ can be calculated as

$$\mathcal{P}(H \mid D) = \frac{\mathcal{P}(D \mid H)\mathcal{P}(H)}{\mathcal{P}(D)}$$

and in this way, the posterior is conditioned by both the prior belief and the evidence that the data presents. While there are a variety of different techniques that can be used to

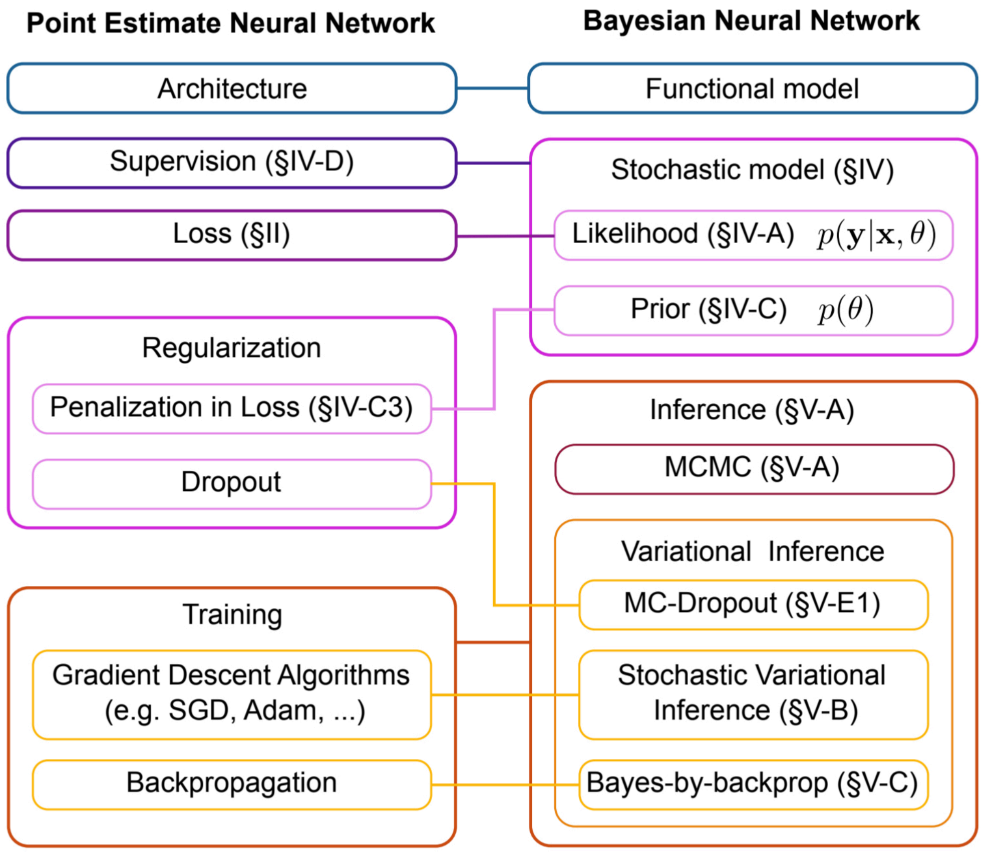

Figure 2: An overview of the corresponding structures in a standard neural network and a bayesian neural network. Originally appeared in [7].

update the prior belief into a posterior distribution as seen in figure 2, in this paper only stochastic variational inference (SVI) and Bayes-by-backprop are utilized.

1.3.1 Stochastic Variational Inference (SVI)

While other inference methods such as MCMC (with popular algorithms including HMC and NUTS [8] [9]), allow exact sampling from the posterior distribution, these methods have proven unpopular with the BNN community due to their algorithmic complexity and lack of scalability to larger models. Hence many communities use SVI which is not an exact method. In SVI, there is a family of distributions $q_\phi(H)$ which are parametrized by parameters ϕ. A common example would be the family of normal distributions parametrized

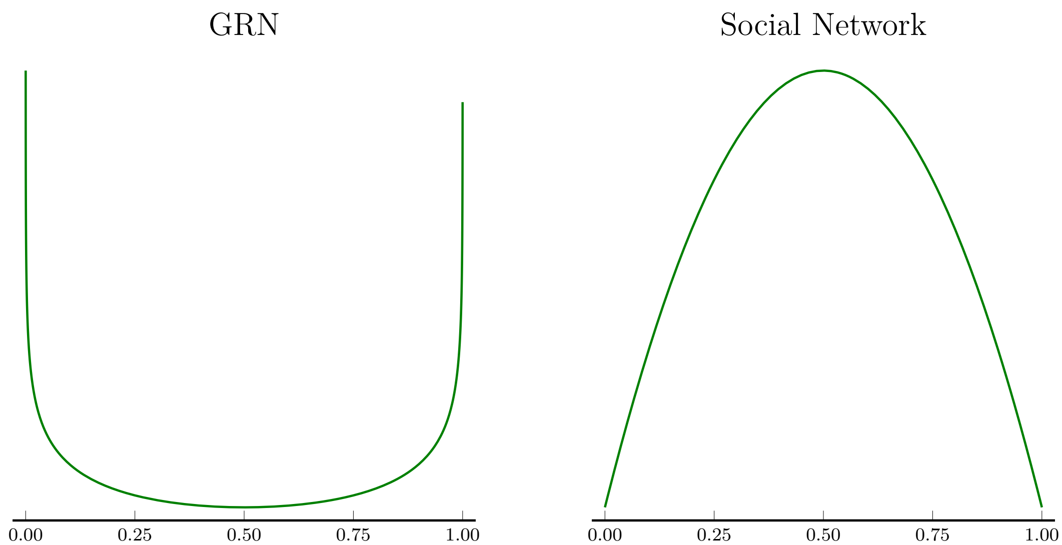

Figure 3: Idealized prior distributions for GRNs and Social Networks based on the Beta distribution. The displayed distributions are $\mathrm{B}(0.95, 0.95)$ and $\mathrm{B}(2, 2)$ respectively

by their mean and covariance structure. The goal of SVI is to approximate the posterior $\mathcal{P}(H \mid D)$ as closely as possible by $q_\phi(H)$. The most common measure of approximation in probability space is given by the Kullback-Leibler divergence (KL-divergence). While not a proper metric over the space of distributions, it does give a computationally-reasonable method to optimize against ϕ to get as close a match as possible. Specifically, SVI aims to minimize

$$D_{KL}(q_\phi \mid\mid \mathcal{P}) = \int_H q_\phi(H') \log \frac{q_\phi(H')}{\mathcal{P}(H' \mid D)} dH'$$

This is still problematic since the quantity $\mathcal{P}(H \mid D)$ would still need to be calculated. Hence, it is sufficient to optimize against the ELBO which serves as a lower-bound for the KL-divergence. The ELBO is defined as

$$\log \mathcal{P}(D) - D_{KL}(q_\phi \mid\mid \mathcal{P}) = \int_H q_\phi(H') \log \frac{\mathcal{P}(H', D)}{q_\phi(H')} dH'$$

Note here that $\log \mathcal{P}(D)$ is just a constant meaning that minimizing the KL-divergence is the same as maximizing the ELBO. Note that, generally speaking, the families q_ϕ tend to come from the exponential family of distributions and the parameters for these families are then just optimized using a typical SGD algorithm such as ADAM [10].

1.3.2 Bayes-by-backprop

While SVI provides a good framework for Bayesian inference, it does not quite work for deep learning applications because stochasticity stops backpropogation from going through a neural network. To mitigate this problem, the usual reparametrization technique used in creating variational autoencoders (VAEs) [11] is combined with SVI to create a deep-learning friendly SVI algorithm. In this variation, a simple non-parametrized random variable $\epsilon \sim q(\epsilon)$ is sampled. To obtain the family $q_\phi(\theta)$, a deterministic transformation $t(\epsilon, \phi)$ is applied such that $\theta = t(\epsilon, \phi)$ has the property that $\theta \sim q_\phi(\theta)$. To obtain such a t, only a certain class of functions can be utilized. These functions are broadly known as bijectors and require t to be a diffeomorphism. In more detail, let $t : M \to N$ be a differentiable map, then t is a diffeomorphism if it is a bijection and its inverse $t^{-1} : N \to M$ is differentiable as well.

Generally speaking, the exponential family of distributions can all be constructed from such transformations meaning that they are good candidates for Bayes-by-backprop. Note though, that because of the transformation t, the formula for the ELBO changes to the following

$$\int_\epsilon q_\phi(t(\epsilon, \phi)) \log \frac{\mathcal{P}(t(\epsilon, \phi), D)}{q_\phi(t(\epsilon, \phi))} |\det(\nabla_\epsilon t(\epsilon, \phi))| d\epsilon$$

This is much friendlier to compute since ϵ is now a constant with respect to ϕ and lets us perform SVI through multiple layers of transformations simply by using bijectors like t.

1.4 Normalizing Flows

The technique that was described in §1.3.2 is more generally known as a normalizing flow. While it was described earlier in the context of a reparametrization technique in which the t are fixed, there is no such restriction in reality. More concretely, the t do not have to be simple functions but, rather, can be learnable functions in their own right. This allows one to use the normalizing flow technique to perform tasks like density estimation and distribution fitting with a very flexible class of transforms that take a simple distribution like a standard multivariate normal and make them into any computable distribution [12]. Let the transformations of the base distribution be defined as $g = g_n \circ g_{n-1} \circ ... \circ g_1$ with inverse $f = g_1^{-1} \circ g_2^{-1} \circ ... g_n^{-1}$. Then we know that the determinant of the Jacobian of f is given by the product of the determinants of the Jacobians at each intermediate evaluation of the flow. This allows for more and more complicated transformations by introducing

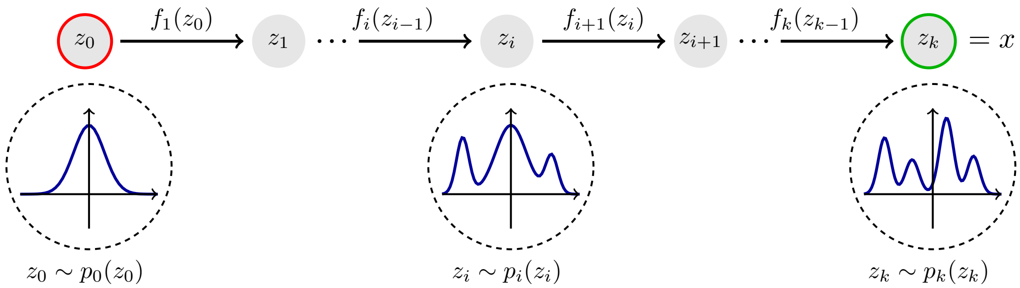

Figure 4: Chaining learnable bijectors to transform a base distribution to a more complicated distribution from [13]

more and more learnable layers as can be seen in figure 4. This structure is very similar to that of an artificial feed-forward neural network. As an example, the simplest form of a normalizing flow is

$$g_i(x) = Ax + b$$

with the learnable parameters here being the matrix A and bias vector b. As long as A is an invertible matrix, we have a bijective function that can be used as a normalizing flow layer. Note that these linear layers can be interleaved with activation functions to provide non-linear transformations. This is important as a linear transformation of an exponential family will remain exponential so an element of non-linearity is required (as is the case with MLPs). While RELU is not invertible, a formulation like leaky-RELU can be used for this task [14]. Still these are not super expressive. For a normalizing flow with universality, this paper utilizes rational quadratic spline based flows [15]. When combined with variational inference over the prior base-distribution, this allows for a very flexible estimation of the posterior distribution no matter how complex.

2 Related Work

When it comes to GNN interpretability, there are a few main methods. The first that started the field of GNN interpretability was GNN Explainer [4] which also provided a suite of general benchmarks that most methods have utilized as a framework for analyzing the effectiveness of their explainer framework. Another major piece of work in the field has been the parametrized-graph explainer (PGExplainer) [16] that took GNNExplainer and parametrized it with a deep neural network for faster inference times and more robust interpretation. Along with these two, a few other more recent explainers such as SubgraphX [17] and Gem [18] have introduced new ideas into the field with a variety of approaches to the problem of GNN interpretability. To date, there seems to be no fully Bayesian method to the problem of GNN interpretability.

In addition to these works, the work of SERGIO [19] will be introduced as it will be utilized later on to generate a new class of experiments that GNN Interpretability methods can be benchmarked against. This work provides causal graph structures that give a guaranteed groundtruth for interpretation.

2.1 GNN Explainer

The full version of GNNExplainer attempts to learn both a node interpretation and edge interpretation. For this work, only the edge interpretation part of the framework was utilized. GNN Explainer attempts to solve the objective outlined in §1.2, by only trying to learn $\mathcal{W}_i$ while treating $\mathcal{E}_i = E$. In this framework, GNN Explainer enforces that for any $e \in E$, $W(e) \geq \mathcal{W}_i(e)$. Then to get the argmax, GNNExplainer treats $\mathcal{W}_i$ as a random variable. Then the goal get transformed to

$$\underset{\mathcal{W}_i}{\arg\min}\ \mathbb{E}_{w \sim \mathcal{W}_i}[H(\phi(v_i, \mathcal{X}, E, w))]$$

This still remains intractible, so GNNExplainer attempts to make this simpler by using Jensen's inequality. Note that this is not a reasonable application of Jensen's since ϕ as a GNN has almost no hope of being convex. Nonetheless, using Jensen's inequality gives

$$\underset{\mathcal{W}_i}{\arg\min}\ H(\phi(v_i, \mathcal{X}, E, \mathbb{E}[\mathcal{W}_i]))$$

This is still quite intractable if $\mathcal{W}_i$ is a full joint distribution over all $e \in E$. Therefore, GNNExplainer attempts to use a mean field approximation for $\mathcal{W}_i$ where the edge interpretation is decomposed into the product of Bernoulli distributions meaning that each edge weight is an independent Bernoulli distribution with mean equal to the underlying probability of the variable. Specifically,

$$\mathcal{P}(\mathcal{W}_i) = \prod_{(v_j, v_k) \in E} \mathcal{W}_i[v_j, v_k]$$

with each $\mathcal{W}_i[v_j, v_k]$ is a value between $[0, 1]$ representing a Bernoulli variable for each edge in the underlying graph as defined by E. In this case, if the classification for a node is c, GNNExplainer performs direct gradient descent on this array of values to minimize

$$\underset{\mathcal{W}_i = \{\mathcal{W}_i[v_j, v_k] | (v_j, v_k) \in E\}}{\arg\min} - \sum_{c=1}^{C} \mathbb{1}_{y=c} \log \mathcal{P}(\phi(v_i, \mathcal{X}, E, \mathcal{W}_i) = c)$$

While there is some probabilistic formulation here, in effect, GNNExplainer optimizes an adjacency matrix in $[0, 1]$ against the mutual information of the model given the edge weights and the model with the original graph. This means that GNNExplainer learns no conditional structure between edges and does not take the graph dynamics into account while training. This is further emphasized with the fact that a mean-field approximation was used to assume conditional independence between the edges of the underlying graph. Hence, it is an algorithm that provides only a summary of the interpretation using assumptions that are not generally applicable to GNNs.

2.1.1 Benchmark Datasets

As one of the first explainer methods for GNNs, GNNExplainer created a set of synthetic datasets that serve as the canonical datasets for GNN Interpretability. The goal of this section is to describe these datasets and the perceived shortcomings in these datasets that led to an exploration of their validity and a setup for the experiments produced later that demonstrate the incorrectness of these datasets for the stated task.

The main dataset focused on in this paper is the Tree-Cycles dataset [4]. In this dataset trees of depth three are attached to cycles of length six in order to form and aggregate dataset. A GNN node classification task entails predicting whether a given node is either in a tree portion of the graph or in the cyclic portion of the graph with no given node features. The idea here is that the GNN can only rely upon the structure of the graph for its node classification and all its information must come from the edges. Hence interpretation on the edges of the GNN would reveal only information that could be gleaned from the graph structure. In figure 5, one can see an example of a portion of the dataset looking at a three-hop neighborhood around node 565.

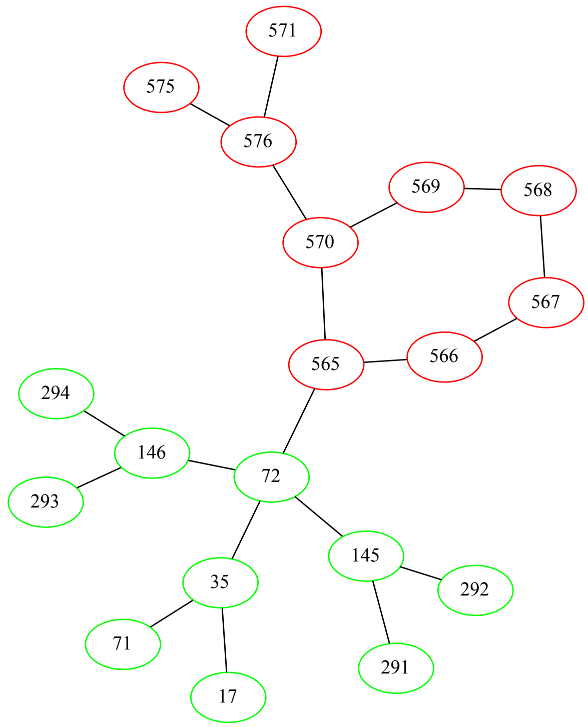

Figure 5: A look at the tree-cycles dataset at a three-hop neighborhood around node 565. In green are nodes classified as tree nodes and in red are nodes classified as cycle nodes. The dataset is almost 50% balanced between these two types.

While this is a great construction to test an interpretation method, given that the GNN only relies on the graph structure for prediction, it is difficult to imagine what the ground truth for a given dataset is. The paper that introduced GNNExplainer proposed that the groundtruths be motifs in the graph. So if a node was in a cycle portion, the groundtruth would be the edges in the cycle and if a node was in the tree portion, the groundtruth would be the edges of the tree. This can be seen in figure 6 which shows the proposed ground truths. While this is a valid task for an interpretation technique to attempt to solve, it does not have direct bearing on the task that the GNN was trained on and it does not have direct bearing on the mutual information framework that forms the theoretical underpinning for GNN interpretability.

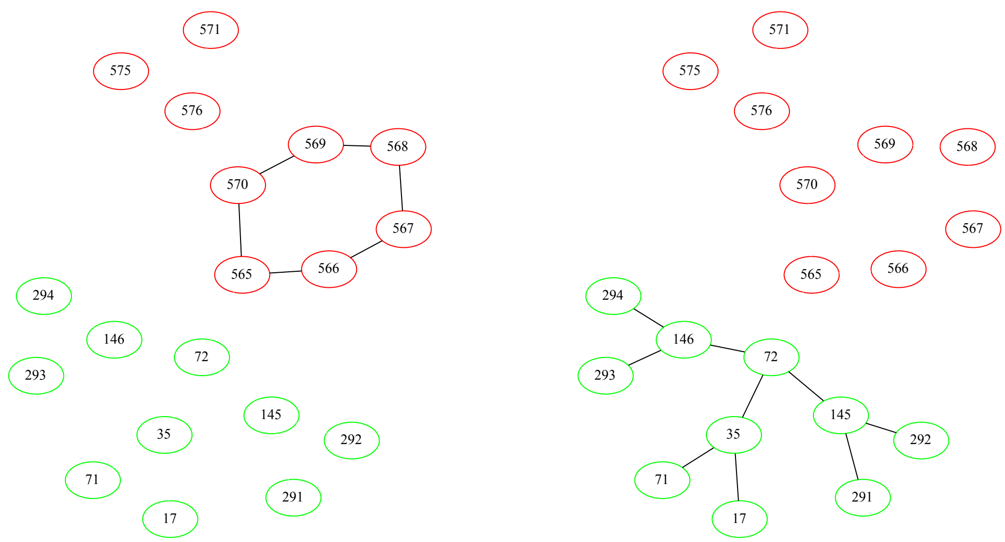

Figure 6: Demonstrations of the proposed groundtruths under [4] for the tree-cycles dataset. On the left is the proposed groundtruth for a node in the cycle (565) and on the right is the proposed ground truth for a node in a tree (72).

In a simple example, suppose a GNN is trained on the tree-cycles dataset. While it might be nice if the GNN needed all the edges in a cycle structure to determine that the node is, indeed, in the cycle, the GNN could have just as easily learned to use five edges from the cycle to make its predictions. Even if a theoretical GNN interpretation model was perfect, the fact that the GNN itself does not use the sixth edge means that an interpretation technique is doomed to max its accuracy score at 5/6 which would make it impossible to compare between methods. Furthermore, a GNN may learn to mix motifs when making its prediction. Consider node 565 in the exampel from figure 5. While this node is in the cycle portion of the graph, the GNN could very easily learn that the node is *adjacent* to the tree structure detected in node 72 and learn to make the inverse decision in this case. Notably, there is no garuntee that a GNN learns the motifs as the important substrucutres and there is little chance that across nodes, it consistently learns these rules. Indeed, it will be shown later that GNNExplainer itself struggles to meet its $> 90\%$ accuracy scores for this benchmark in replication studies [17] [18] and that a thorough search through all connected subgraphs fails to yield this structure as the ground truth.

While GNNExplainer suggests a few other benchmark datasets, they all suffer from the same issue. Namely, they claim that the groundthruth is the embedded motif structure, but a simple tests reveal that this is not consistently true and nor should it be true in the general case. Hence, a goal of this paper is to also provide a set of alternative benchmarks agaisnt which to evaluate GNN interpretability.

2.2 Parametrized-Graph Explainer

Parametrized-Graph Explainer (PGExplainer for short) aims to take the GNNExplainer process and parametrized so that after training a deep neural network, new explanations per node could be generated with the much smaller cost of conducting inference on a give node v_i.

In the PGExplainer model, they attempt to construct a generative probablistic model for the underlying subgraph by assuming that the explanatory graph is a Gilbert random graph [20] where the edges are conditionally independent of each other, much like GNNExplainer. Hence, the probability of a random subgraph being chosen is given in the same exact way as GNNExplainer. For clarity, this is

$$\mathcal{P}(\mathcal{W}_i) = \prod_{(v_j, v_k) \in E} \mathcal{W}_i[v_j, v_k]$$

and the learning objective, which starts the same as GNNExplainer, becomes the following as they avoid using Jensen's or the mean field approximation

$$\underset{\mathcal{W}_i = q_\Theta}{\arg\min}\ \mathbb{E}_{W_i \sim \mathcal{W}_i}[H(\mathcal{P}(\phi(v_i, \mathcal{X}, E, W_i))]$$

Note here that the distribution $\mathcal{W}_i$ is to be generated by a DNN that is specified by q and takes as input the parameters Θ. To sample these graphs in a manner that allows for training over the global view of the node classification task, the PGExplainer framework does the following.

1. Most node classification GNNs have a bunch of GNN layers before a final MLP that takes the graph embeddings obtained from the GNN layers and performs a final prediction on the nodes classification. The PGExplainer framework denotes this graph embedding step as ϕ_0 and extracts it from the GNN task. Let

$$Z = \phi_0(v_i, \mathcal{X}, E, W_i)$$

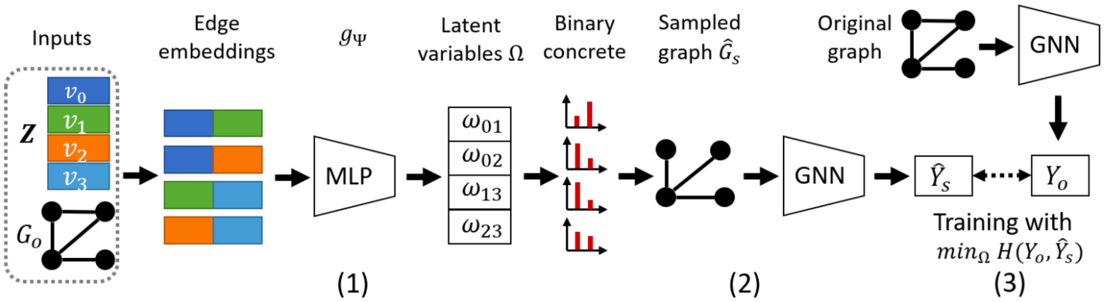

Figure 7: A convenient overview of the PGExplainer pipeline (with slightly different variable names and setting etc.) [16]

be the graph embedding calculated with a sampled subgraph W_i.

2. To calculate the distribution $\mathcal{W}_i(v_j, v_k)$, the PGExplainer lets g be a MLP network with shared parameters Θ. As an input, this MLP takes the concatenated values of the graph embedding Z for node i and the nodes j and k as they represent the edge (v_j, v_k). Hence, we have

$$\mathcal{W}_i(v_j, v_k) = g_\Theta([Z_i; Z_j; Z_k]) \qquad \forall (v_j, v_k) \in E$$

Note here that all edge distributions are independent with respect to i, j, k.

3. Finally, given this distribution. The parameters Θ are trained by sampling K subgraphs using the independent edge distributions (utilizing a reparametrization technique) and then optiming the above objective with these samples. Let $W_i^{(k)}$ represent the kth sampled subgraph. Then the following objective function represented the final target against which PGExplainer was optimized

$$\arg\min_{\Theta} \; - \sum_{v_i \in V} \sum_{k=1}^{K} \sum_{c=1}^{C} \mathcal{P}(\phi(v_i, \mathcal{X}, E, W) \log \mathcal{P}(\phi(v_i, \mathcal{X}, E, W_i^{(k)}))$$

Together this gives the PGExplainer framework. A convenient outline from their paper can be seen in figure 7. Note that this framework was tested against the same benchmarks that were given by GNNExplainer. While this gives a more probabilistic framework, it suffers, again, from learning a conditionally independent distribution per edge which disallows further introspection and also ignores the conditional structure while generating subgraphs.

This method is better in that it trains these distributions globally across nodes which means that more structure is being captured on the average and allows the model to be universal for a dataset. In effect, the combined training allows for a blurring of the edge importances across different nodes which makes it more useful and generalizable when looking at the edge importance of a specific node.

2.3 Other Explainer Frameworks

While there have been a few other explanation frameworks that have emerged in the time following GNNExplainer [4] and PGExplainer [16], most featured roughly similar performance or design. There are, though a couple frameworks that provide an interesting comparison to the two just mentioned.

2.3.1 SubgraphX

This method is different than most methods in that it does not attempt to learn a $\mathcal{W}_i$. Instead, it actively searches for a $\mathcal{E}_i$ and lets $\mathcal{W}_i(v_j, v_k) = 1\ \forall (v_j, v_k) \in \mathcal{E}_i$. It does this via a Monte-Carlo Tree Search algorithm that is informed by computing Shapley values for proposed subgraph structures and combining them in the tree search [17]. Additionally, this paper eschews the traditional accuracy/AUC evaluation metric used by GNNExplainer and PGExplainer in favor of a Fidelity/Sparsity framework. Crucially, they note that the task being defined by these interpretaion models is not detection of human-interpretable groundtruth motifs, but rather, fidelity to the original predictions made by the full GNN model.

2.3.2 GEM

The GEM framework attempts to use Granger Causality to guide its exploration process in learning the $\mathcal{W}_i$ function. The most interesting contribution of this paper to this study was the recognition that the groundtruths determined in the GNNExplainer and PGExplainer works were a bit arbitrary. Certainly, in real-life datasets the motifs that drive a GNN performance are not always apparent and need to be discovered rather than rediscovered. Hence, these benchmarks are a little suspect as they have little bearing on the process of using a GNN interpretation method on a real-world problem.

2.4 SERGIO

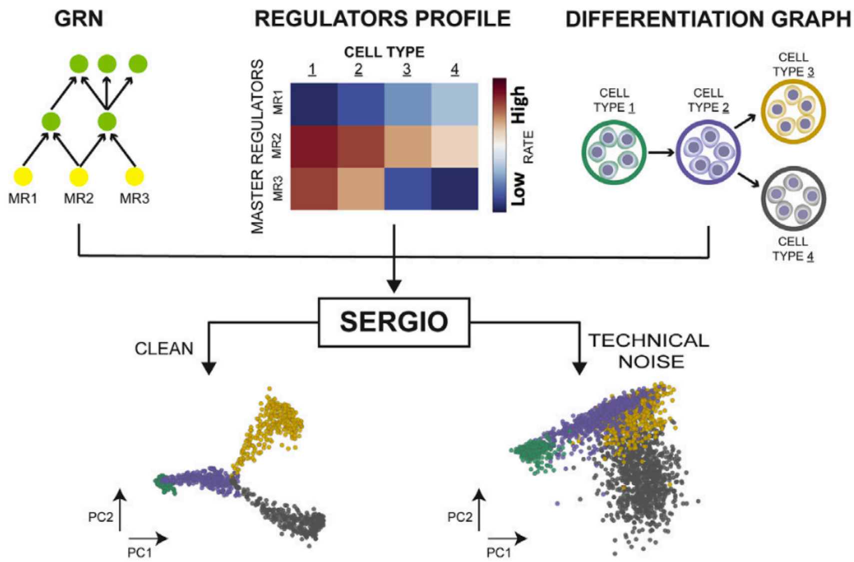

Figure 8: A graphic from the SERGIO paper detailing the inputs and outputs to the SERGIO process [19]. Specifically, a ground truth GRN is utilized to generate the data and recovering this GRN from the data could be a task for a GNN interpretation task.

SERGIO is a model for simulating single cell gene expression data with guidance from gene regulatory networks [19]. Before SERGIO, most methods of simulating single cell expression data came from analyzing the statistical properties of real-life single cell datasets and then matching these properties with appropriate distributions that could then be sampled from in order to generate new data. While these methods got more and more sophisticated in the number of statistics they matched against, there was no inclusion of gene regulatory networks (GRNs) which are crucial for single cell expression profiles both in the steady state and dynamical regimesWhile these methods got more and more sophisticated in the number of statistics they matched against, there was no inclusion of gene regulatory networks (GRNs) which are crucial for single cell expression profiles both in the steady state and dynamical regimes.

SERGIO aimed to bridge this gap and created a method that both matched the statistical distributions of experimental expression profiles while utilizing GRNs as the core of the modeling process as seen in figure 8. Crucially, though, the GRN underlying the final expression profile is entirely causal in driving the profiles that it generates and SERGIO maintains the ability to add technical noise into the proposed expression profile. This makes it a perfect setup for an experiment in GNN interpretability where the goal of an interpretation method would be to recover the underlying GRN from a GNN trained on predicting cell type from the expression data. While biological in origin, the abstract problem is perfect for a GNN interpretation task to be benchmarked against.

3 Methods

In this paper, three different probablistic formulations exist for the underlying edge importance model. All three of these models rely upon the excellent `Pyro` [21] framework for implementation and inference. Additionally, all GNN models are trained through the `pytorch_geometric` [22] and `pytorch` [23] frameworks.

3.1 Beta Model

The first method used in this paper is a prior in which all Beta distributions are considered independent of each other. Specifically, we assume that $\mathcal{E}_i = E$ and that

$$\mathcal{W}_i = \{\mathcal{W}_i(v_j, v_k) \mid (v, j, v_k) \in E\}$$

with

$$\mathcal{W}_i(v_j, v_k) = \mathbb{E}[Beta(\alpha_j, \beta_k)]$$

with $Beta(\alpha_{j,k}, \beta_{j,k})$ representing the prior Beta distribution on edge (v_j, v_k) with specific parameters $\alpha_{j,k}$ and $\beta_{j,k}$. Note, we define the Beta distribution as

$$\mathcal{P}(Beta(\alpha, \beta) = x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha, \beta)}$$
$$\mathrm{B}(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

where Γ is the canoncical Gamma function [24]. Given this, the goal of the interpretation task is to learn a variational family $q_\phi(H)$ that most closely resembles the posterior distribution when the categorical distribution

$$\phi(v_i, \mathcal{X}, E, \mathcal{W}_i)$$

is conditioned on the full model

$$\phi(v_i, \mathcal{X}, E, \mathcal{W})$$

where $\mathcal{W}(e) = 1$ for all $e \in E$. For the sake of this experiment, the posterior distribution is also assumed to be a set of Beta distributions but fully conditional on each other. The goal, then is to learn posterior values $\hat{\alpha}_{j,k}$ and $\hat{\beta}_{j,k}$. Given these values, the final explanation is

$$\mathcal{W}_i(v_j, v_k) = \mathbb{E}[Beta(\hat{\alpha}_j, \hat{\beta}_k)]$$

which has an easily derived closed form.

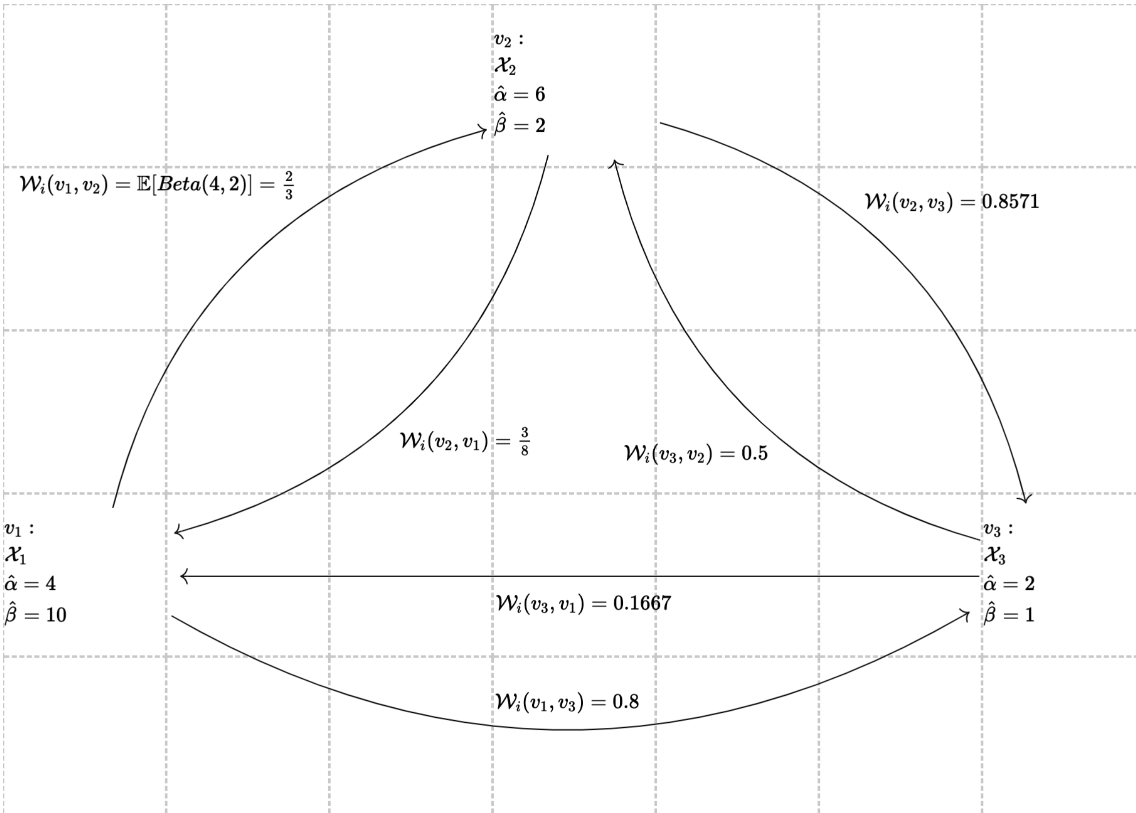

Figure 9: An example of a Beta posterior that shows information flowing from v_1 to v_3.

3.1.1 Interpretation of the Beta Model

There is a wide array of literature on the use of Exponential-families for the modeling of edge weight priors. In particular, the work of Bolla [25] is important. In this work, edge weights are modeled by conditionally independent Beta distributions. The parameters of these Beta distributions are updated via a closed-form solution but it serves as a good model for prediction propogation of information or material from one node to another and converging on a steady-state for the propogation. In the case of GNNs, one can imagine the `MSG` operation as the sending of information. The dense layers in the `Upd` step, then, learn the weight to assign to each incoming and outgoing message. In this vein, the Beta model can approximate this ”attention” that is paid in the `Upd` step. Hence, the goal of this model is to serve as an approximiation of this process and encode the steady-state flow of information in the model.

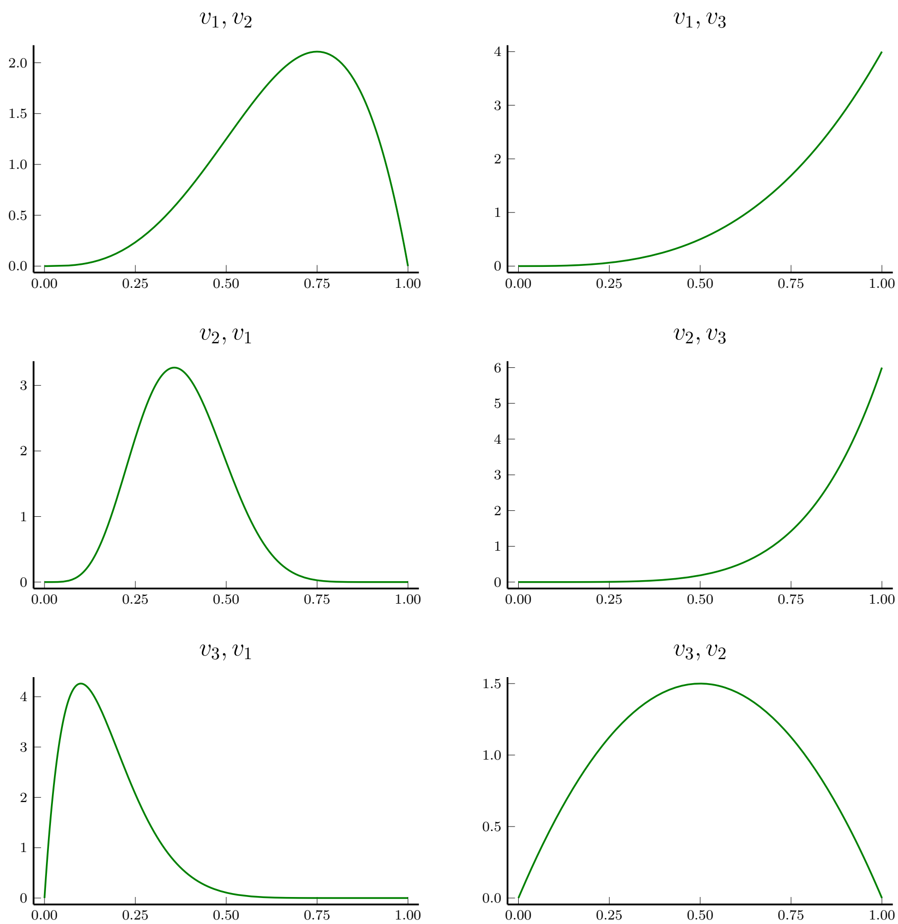

Figure 10: Posterior distributions for each edge in 9

In this model, the α values serve to desribe the affinity for information to flow out of a node while the β values tend to describe the affinity for information to flow into the node. Together, they describe the steady state flow of information and perfectly describe many networks in which pathways of information lead to end-goal expression. Indeed, since GNNs tend to embed this type of information flow in a deep-learning setting, it serves to reason that such a prior would be a good encapsulation of the process of information flow, especially at steady state. A typical example of this is gene expression in biology [3] where paths of edges represent regulatory pathways directly.

An example of this flow can be seen in figure 9. In this figure, we consider a hypothetical posterior for this model and demonstrate how it describes the flow of information from node

v_1 to v_3 via these α and β parameters. Additionally, in figure 10, one can see the posterior distribution of each edge. Notably, there is no conditional independence in this model as changing the parameter on one node changes all adjoining edges.

3.1.2 Training the Beta Model

While [25] demonstated a novel technique to train the parameters to get the true posterior in the limit, this paper will use a standard SVI setup to train the posterior. In the `Pyro` framework, all SVI runs need two things: the forward model and the variational guide. In listing 1, both the model is shown and the guide follows a very similar setup since it has the same esxact structure. Note that in both cases the mask is sampled from a beta distribution as defined above. Most GNNs have an effective computational graph size as denoted by the number of layers that are in the model. This effective computational graph is generally, the l-hop neighborhood around v_i so we only consider interpretation on $\mathcal{N}_l(v_i, E)$ as the other nodes cannot have any influence. Analogously, let all the nodes in this l-hop neighborhood be $V_{i,l}$.

Algorithm 1 The model setup for the Beta model

Require: $v_i \in V, \bar{\alpha} > 0, \bar{\beta} > 0$
Require: $c \sim \phi(v_i, \mathcal{X}, \mathcal{N}_l(v_i, E), \mathcal{W})$
$\alpha \leftarrow [\bar{\alpha} \mid \forall v_j \in V_{i,l}]$
$\beta \leftarrow [\bar{\beta} \mid \forall v_j \in V_{i,l}]$
$W_i(v_j, v_k) \sim Beta(\alpha_j, \beta_k) \qquad \forall (v_j, v_k) \in \mathcal{N}_l(v_i, E)$
$\hat{y} \leftarrow \phi(v_i, \mathcal{X}, \mathcal{N}_l(v_i, E), W_i)$
$\hat{c} \sim \hat{y}$

The SVI algorithm then optimizes against the α and β parameters to match the variational distribution $\hat{y}$ as closely as possible to the full distribution $\phi(v_i, \mathcal{X}, \mathcal{N}_k(v_i, E), \mathcal{W})$. Note that in the case of a graph classification task, the model ϕ simply loses its dependence on v_i and we compare against $\phi(\mathcal{X}, E, \mathcal{W})$. They are the exact same formulation. In each epoch of the SVI inference, only one sample is used to get an estimate of the ELBO as is the standard practice [7]. While this gives a noisy estimate, over time, the value of the ELBO should increase to indicate a better model fit.

3.2 Normalizing Flow Models

In this work, normalizing flow models that are tested. A simple KL-divergence is computed and the parameters of the normalizing flow are directly optimized using an appropriate SGD algorithm.

3.2.1 Spline Autoregressive Normalizing Flows

As mentioned in section §1.4, for a fully universal normalizing flow model, more expressive transformations are required than invertible affine transformations. One such method comes from spline autoregressive normalizing flow layers as described in [26] and implemented in [15]. In this work, spline transforms are combined with autoregressive layer transforms.

First, we describe autoregressive layer transforms in general. Given a random vector $Z = (Z_1, Z_2, \dots, Z_D)$ we can always write the distribution of this variable as

$$\mathcal{P}_Z(z) = \mathcal{P}_{Z_1}(z_1) \prod_{i=2}^{D} \mathcal{P}_{Z_i|Z_j \forall j<i}(z_i \mid z_j \forall j < i)$$

The aim of this representation is to chain D invertible transformations to get the full distribution over Z. Note here, any normalizing flow can be used for each transformation. In this case, the LRS layers from [26] are used.

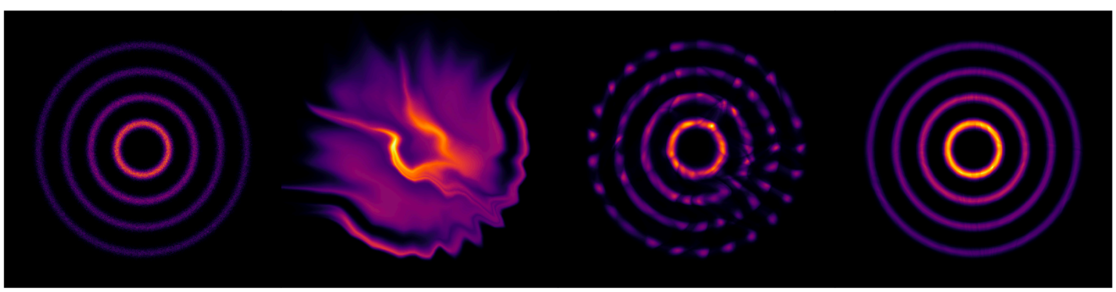

Figure 11: Different normalizing flow models on a synthetic conditionally distributed dataset. From left to right: ground truth, Glow, i-ResNet, Linear Rational Splines (used in this method). Taken from [26]

Most importantly, these coupling flows are able to capture full joint distributions and have been used in models such as VAEs and generative models [12] as seen in figure 11.

Crucially, this gives an extremely flexible family of distributions that ought to be able to capture the conditional structure that exists within a graph.

More specifically, when sampling a random graph, one can begin the construction by taking a base edge and sampling a value from it. Then, to sample another edge, the distribution must be conditional on the distribution of the first edge. This process continues with each successive edge that is sampled being contingent on all preceding edges which is exactly what happens in the autoregressive transform. Hence, this is a good fit for the graph sampling problem.

3.2.2 General Normalizing Flow Model

In general, the normalizing flow will will operate as follows:

1. A value will be sampled from a base distribution $b \sim \mathcal{B}$ with $\mathcal{B}$ being a fixed base distribution. In this case, we chose

$$\mathcal{B} = Beta(0.95, 0.95)$$

   in order to bias the explanation towards sparse explanations that are favored in the literature [17].

2. The value then gets put through a series of m linear rational spline coupling normalizing flows giving the edge mask distribution. Let all the layers be denoted by $\mathcal{L}_{\mathcal{B},\Theta}$ where Θ represents the parameters of the LRS layers. Therefore, we have

$$W_i = \sigma(\mathcal{L}_{\mathcal{B},\Theta}(b))$$

   where σ is the standard sigmoid function.

3. Finally, we get the conditional GNN classifier $\hat{y}$ as the distribution given by

$$\hat{y} = \phi(v_i, \mathcal{X}, \mathcal{N}_l(v_i, E), W_i)$$

   from which we apply a learning algorithm to obtain the weights Θ

4. Once this is done, the final edge interpretation is given by

$$\mathcal{W}_i = \mathbb{E}[\sigma(\mathcal{L}_{\mathcal{B},\Theta})]$$

   which is obtained simply by averaging Monte-Carlo sampling of the LRS layers.

For the graph edge interpretation model, this is the most general Bayesian algorithm that can be given due to the universality of the LRS coupling normalizing flow. Since the graph interpretation is expected to be quite a complicated distribution, the goal of this method is to capture all of the higher order complexities, including multi-hop relationships. The listing 2 succintly captures this algorithm.

Algorithm 2 The model setup for the Normalizing Flow model

Require: $v_i \in V, \bar{\alpha} > 0, \bar{\beta} > 0$
Require: $c \sim \phi(v_i, \mathcal{X}, \mathcal{N}_l(v_i, E), \mathcal{W})$
  $\mathcal{B} \leftarrow Beta(\bar{\alpha}, \bar{\beta})$
  $b \sim \mathcal{B}$
  $W_i \leftarrow \sigma(\mathcal{L}_{\mathcal{B},\Theta}(b))$
  $\hat{y} \leftarrow \phi(v_i, \mathcal{X}, \mathcal{N}_l(v_i, E), W_i)$
  $\hat{c} \sim \hat{y}$

3.2.3 KL-Divergence and Direct Optimization

The method used to train the parameters Θ is taking the disrtibution $\hat{c}$ and c and backpropogating against the KL-divergence of these two distributions. Because of the reparametrization trick and the fact that the GNN is fully differentiable, this gives the gradient of the KL-divergence against Θ and allows for direct optimization against these parameters. Any SGD algorithm can be used, and, in particular, the ADAM optimizer [10] is used in this model. Additionally, an L1 regularization is applied to the edge mask to enforce sparsity on the model. This ensures that the model does not converge on the full graph as its output and is forced to learn the truly important edges.

4 Experimental Setup

As was discussed in §2.1.1, there are many issues with the benchmarks that were proposed by [17]. Therefore, a major goal of this work was to confirm the intuition that these benchmarks were off. Then, the aim was to create a new set of benchmarks against which GNN interpretation models can be tested against. In this section, the methodology of these experiments will be walked through.

4.1 Validation of Incorrectness in Benchmark Datasets

As was shown in figure 6, the proposed ground truth for the tree cycles dataset that was introduced by GNNExplainer is supposed to be the graph motifs that are encoded as the node classes. Specifically, for a node that is in a cycle, the proposed groundtruth is all of the edges in the size six cycle. Similarly, for a node that is in the tree portion of the dataset, the supposed groundtruth is all the edges in the tree.

To validate our intuition that this was not what the GNN was necessarily learning, we conducted the following experiment:

1. We trained a GNN on the tree-cycles dataset and ensured it had good validation accuracy 95%. Since the nodes have no node features, it is clear that the GNN is *only* learning from the computational graph that it is given.

2. For a few sample nodes classified as cycle nodes, we collected every connected size six subgraph.

3. These subgraphs were fed to the GNN and were ranked according to the mutual information of the outputed classification distribution.

4. These samples were then plotted to compare them against the proposed ground truth.

While not a comprehensive review, the computational resources needed to check every node were prohibitive. Note that this metric for choosing the best subgraph comes directly from the GNNExplainer paper meaning that if their assumption was valid, the entire cycle would be chosen as the ground truth. The nodes chosen for this experiment were nodes 557, 566, and 511. In figure 12 are the resulting, subgraphs of size 6.

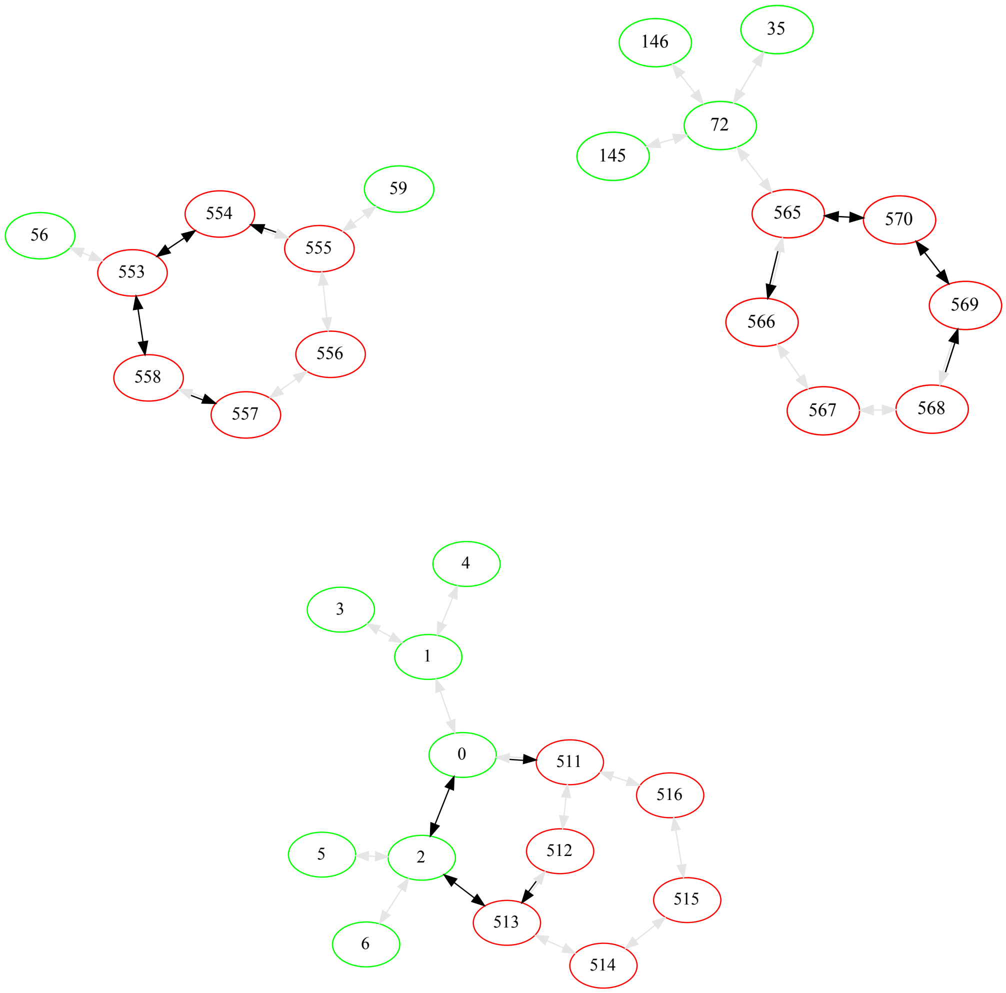

Figure 12: Found best subgraphs of size six for nodes 557, 566, and 511

Indeed, in none of the examples did a full ring structure get predicted. For nodes 557 and 566 it is clear that the most important subgraph uses about half the ring and has feedback structures inside of it with edges going both ways on intermediate nodes. In node 511, the interpretation barely comes from the ring structure and the GNN passes information through the edges of the tree structures. Clearly, it is not obvious that the motifs that the graph was defined on are not the groundtruth. Furthermore, the groundturth is *highly* dependent on the GNN that is trained. With regards to the tree-cycle dataset, there are many different GNNs that peform equally well but have different groundtruths. Hence, it is difficult to use this dataset for GNN interpretation validation. While one could train GNNs until the motifs do become the correct groundtruth, it is hard to validate that any training run has achieved this goal.

One interesting thing to note here is that all three of these examples have the same structure i.e. a node points to an intermediate node that is fully connected (forwards and backwards) with the next node. This node fully connects to a third node before finally connecting to the target node. In this sense, the GNN has learned a general method for identifying cycles but it is not dependent on sending the information through solely cycle edges as seen by the ground truth for node 511.

4.2 Noise Filtering Experiment

While the tree-cycles dataset has its issues, it does serve as a good dataset to test GNNs on due to the fact that all of the performance in the model must be coming from the graph structure that is fed into the GNN. The node features contribute nothing and without any graph at all, the GNN will perform poorly. Hence, the first experiment proposed is a noise filtering experiment where we test the ability of GNN interpretation methods to detect and filter true negatives from their prediction of what is important to a model.

This is done by training a GNN on the tree-cycles dataset and then sampling noisy edges into the graph that did not exist previously. Then the GNN Interpretation methods are run on top of the noisy graph to see whether or not they will mistakenly choose noisy edges as important. Crucially, we do not care about what the groundtruth is but, rather, what it cannot possibly be.

For adding the noise, inspiration is taken from [19] and their addition of technical noise to single-cell expression data. Specifically, noise is added as described in listing 3. As a short summary, for each node v in the graph a number of edges is sampled from $n \sim Possion(\lambda)$. This represents the amount of noisy edges that are going to be added going away from that node. Then, n nodes are chosen at random from the ρ-hop neighborhood of v with the probability of that noisy edge being added being defined as the inverse of the distance between the two nodes on the graph. Note, we restrict this to nodes that are more than distance one away. Once these noisy edges are sampled, they are added to a list that will all be added at once at the end of the sampling process. Note that in this case, we assume that the function $\mathcal{W}$ is always equal to one. With the right parameters λ and ρ, we want

Algorithm 3 Sampling noise into tree-cycle dataset

Require: $\lambda \geq 1$, $\rho \geq 2$.

$\bar{E} \leftarrow []$

for $v \in V$ **do**

$n \sim Possion(\lambda)$

$e \sim Categorical_n\left(\frac{1}{\sum_{v_i \in \mathcal{N}_\rho(v,E)} d(v,v_i)}[d(v,v_i) \mid \forall v_i \in \mathcal{N}_\rho(v,E)] \wedge d(v,v_i) \neq 1\right)$

$\bar{E} \leftarrow \bar{E} \cup e$

end for

$E \leftarrow E \cup \bar{E}$

$\mathcal{W}(e) \leftarrow 1 \qquad \forall e \in E$

to make sure the GNN performance does not degrade but we are adding in ample noise to the graph for the interpretation methods to filter out.

4.2.1 Validation of GNN Performance

To find the right parameters λ and ρ, we need to ensure that the noise that is added is not too high but also that there is a significant challenge for the interpretation methods. The heatmap in figure 13 shows the GNNs performance across a matrix of ρ and λ values.

Note that before noise was added to the graph, all nodes had 2-4 connections in and out. As can be seen, the performance of the model is very good across almost all λ values and ρ values up to four. This indicates that the GNN is quite resiliant to noise and that GNN Interpretation methods should be able to filter out most of the noise if constructed correctly. In this case, we choose $\lambda = 2.5$ and $\rho = 3$ since this would give each node a rougly 50% postitive to negative ratio on original edges to noisy edges. In this scheme, the GNN still performs well with roguhly 92% accuracy.

4.3 Graph Embedding Experiment

For a more comprehensive test, we aim to create a GNN task in which the ground truth is known before-hand. From this we can generate a dataset where the GNN, provably, only utilizes the ground truth and nothing else. Note, that there is no way to ensure that the GNN utilizes each edge in the ground truth, but the aim is to ensure that a subset of the edges is used for each example in the dataset.

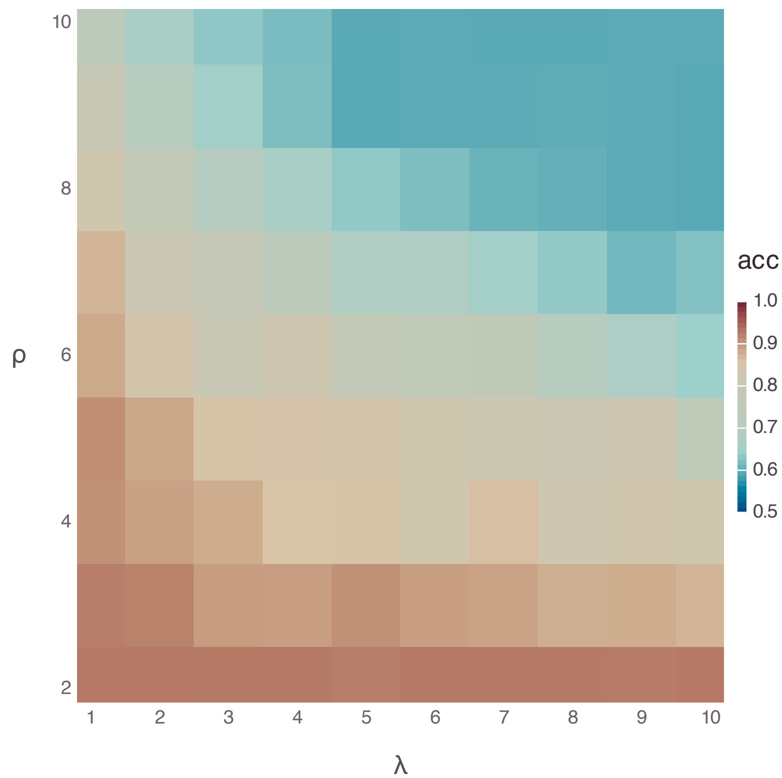

Figure 13: Performance of the noisy tree-cycle experiment across a variety of λ and ρ values

For this task, we define a graph classification task upon a complete graph of size seven. In this task, we define a Markov random walk over the nodes of the complete graph wherein a subset of the edges are biased. The level of bias is determined by a parameter p that can be set at different values to generate different classes. Specifically, in figure 14, one can see the embeded biased graph structure. Other than the biased edges, all other edges have equal probability given as $\frac{1-p}{5}$ except for node one which has all other edges having probability $\frac{1-p}{4}$ (there are no self-edges). Hence, given this structure and a random walk over the nodes, the node feature for each node in the dataset is the number of times it is visited only if it is visited via an edge in the ground truth. In this way, the counts distribution is necessarily dependent on the ground-truth structure and a GNN will pick up on that.

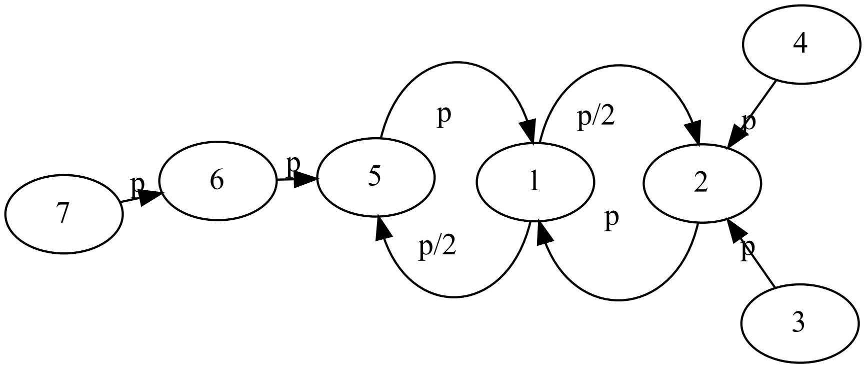

Figure 14: The biased portion of K_7 in the dataset generation. The rest of the edges are not drawn for clarity but have equal probability to each other node.

4.3.1 Dataset Details

In particular, the dataset was generated with class label zero corresponding to $p = 0.4$ while class label one had $p = 0.675$. Additionally, each point in the dataset was generated by taking 75 hops on the random walk and the counts were then aggregated on a per-node basis. Finally, we had that each class had 500 samples each for a total of 1000 samples for the GNN to classify. The dataset also exhibits bimodal distributions for nodes 1, 2, and 5 with the overall distribution being best fit as a mixture of binomial distributions. In figure 15, one can see the marginal counts distributions for each of nodes 1, 2, 5, and 6 (the rest of the nodes, by definition, cannot have any count). Since node six, can only get counts from node seven, the two classes have very little differentiation on that node.

4.3.2 GNN Performance Validation

We know that if the counts values were not conditional on the groundtruth, the counts distribution overall would represent the steady-state distribution of the Markov process. In this case, each node distribution would be independent of each other and the optimal classifier would necessarily be a Naive Bayes multinomial classifier [27]. The groundtruth conditional distributions mean that the features are not independent which means that

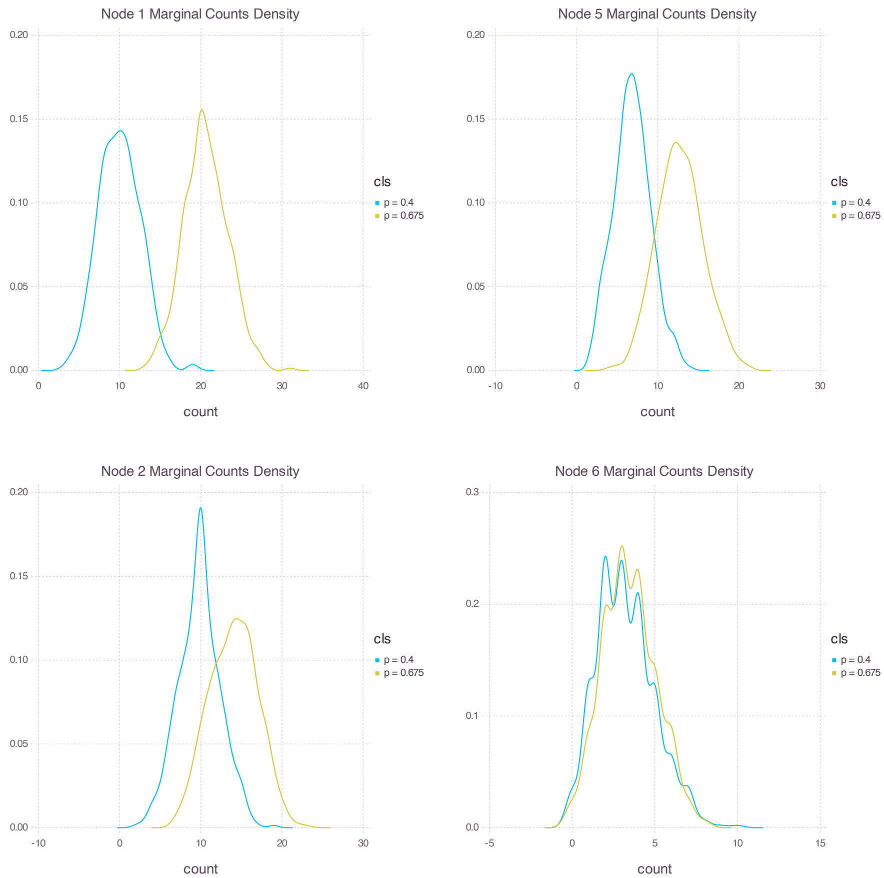

Figure 15: Marginal counts distributions for each node with counts

Naive Bayes is non-optimal. As a comparison, though, we trained a Naive Bayes classifier on this problem and got a test accuracy of 72.2% showing that there is a lot of information to be gained by utilizing the graph structure.

Indeed, we trained the GNN on this problem by feeding it the entire computational graph, but validated its accuracy on fresh examples and the entire groundtruth graph getting an accuracy of 99%. Hence, there is strong reason to believe that the GNN learned from the graph groundtruth. To validate this even further, we checked a few metrics with the trained GNN. First the accuracy of the GNN was measured when the entire computational graph was fed in, with the strength of the edges going from 0 to 1 in 0.1 increments. The same was repeated with the groundtruth and the inverse of the groundtruth (all of the noisy edges). In figure 16, the results of this experimentation can be seen.

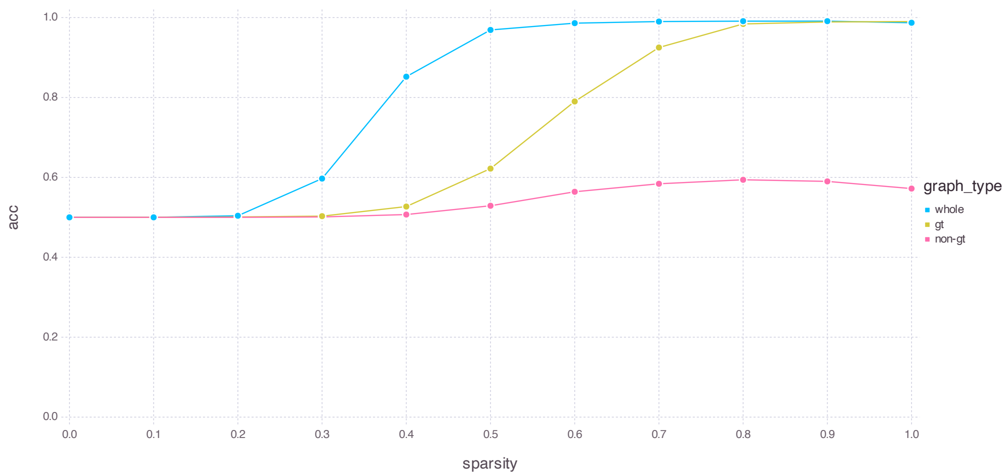

Figure 16: Performance of the GNN at different sparsity with (a) the entire computational graph (b) the groundtruth and (c) the inverse of the groundtruth

While the performance of the whole computational graph increases faster than that of the groundtruth, they ended up with the same accuracy at level 0.8 and it is clear that the inverse of the groundtruth contains very little information as its performance improved very little. What this indicates is that the groundtruth is very essential to the performance of the GNN and that a subset of it for any given node is essential for the predictive power of the GNN. Note that because for class zero the GNN automatically predicts the correct answer, only class one examples are used for the actual interpretation challenge. Hence, this seems like a strong candidate for evaluating GNN interpretation performance. While not all edges in the groundtruth may be needed for each example, all explainer tasks will be at the same disadvantage meaning that the results will be comprable.

5 Results

Below the results from the experiments mentioned in §4 are outlined. Specifically, there are two main experiments that were run: the noise filtering experiment and the graph embedding dataset. As will be seen, the performance of both the Beta model and Normalizing Flow model exceeds that of GNNExplainer and provides additional information into the edge mask distribution. Note that in [17] and [18], both GNNExplainer and PGExplainer demonstrate similar performance with their implementations coming from [22]. PGExplainer was unable to run on these examples with all predictred edge masks being very close to zero and demonstrating poor performance as a result. Hence, GNNExplainer is considered to be representative of the state of the art for the purposes of this work.

5.1 Noise Filtering Experiment Results

Note that for the following table, any prediction that was in the original graph structure is automatically given a correct. As mentioned before, since the goal of this experiment is not to determine the actual ground truth, but rather, to determine the ability of the explainer models to filter noise. For this experiment each of the methods was fed 25% of the nodes that have noisy edges as a test and the results were collated. Not all nodes had noisy edges because of the probablistic noise sampling mechanism, but on average 50% of the edges were noisy. This methodology means that the interpretation methods were exposed to slightly greater than 50% noise as all nodes with no noise were filtered out.

As can be seen from the results in table 1, GNNExplainer underperforms compared to both the Beta Model and the Normalizing Flow Model. The results indicate that all methods are doing a decent job at ensuring that noise is not picked up as important to the model. While there is room for improvement across the board, all methods demonstrate favorable performance with the bayesian learning methods leading the way in performance.

Model	Accuracy
GNNExplainer	0.923
Beta Explainer	**0.982**
Normalizing Flow Explainer	0.966

Table 1: Accuracy results for all three models in the noise filtering experiment

In particular, the Beta Model performs the best with the Normalizing Flow Model following it. While the Normalizing Flow Model is strictly more expressive than the Beta Model, it seems to exhibit higher variance in its training which leads to it picking up noise at times. This means that the Beta Model will perform better in experiments like this. Indeed, the restrictiveness of the Beta Model means that it would be more reluctant to pick any edge as positive. This probably explains its comparative advantage in this experiment and as will be shown later, the Beta Model is very good at picking out first-order pathways that are the most important to the GNN. While it may miss some information that is captured by the Normalizing Flow Model, this means that it is very good at filtering noise.

The probablistic approach to both the Beta Model and Normalizing Flow Model means that they are better suited for filtering noise as compared to GNNExplainer which experiences little penalty for picking up a noisy edge as it tends to have little effect on the performance of the GNN. Instead, GNNExplainer cares much more about finding the postive edges and has little incentive for dropping noisy edges as demonstrated in the GNN performance from figure 16 where the noisy edges added little value but did not decrease the performance of the model. In the probabilistic models, though, these edges add little value and are discarded due to their minimal effect on the model which is discovered when the edge is both sampled and not sampled.

5.2 Graph Embedding Experiment Results

With a known and confirmed groundtruth in this experiment, the interpretation methods were run through a few different metrics: accuracy, precision, recall, f1 score, and AUROC. In this case, the interpretation methods were compared directly against the groundtruth and the results were averaged across nodes to get the final results as seen in table 2. Additionally, the average of all explanation masks was taken in order to account for model variance and to get a sense of what a global interpretation score against the ground truth would give.

Based on these results, we can see that the Bayesian Inference interpretation methods are out-performing the current state-of-the-art in GNNExplainer. While the difference is not massive, these models perform better on all metrics and provide additional qualitative interpretation through the edge mask distributions that they generate. Specifically between the Beta Explainer and NF Explainer, the Beta Explainer has less recall but better precision which give it a higher accuracy and AUROC score while the better recall gives the NF Explainer a better F1 score. Overall, the NF Explainer has higher variance which leads

Model	Aggregation	Accuracy	Recall	Precision	F1 Score	AUROC
GNNExplainer	Avg Metrics	0.612	0.599	0.386	0.468	0.601
Beta Explainer	Avg Metrics	**0.634**	0.505	**0.387**	0.423	**0.658**
NF Explainer	Avg Metrics	0.588	**0.664**	0.376	**0.479**	0.631
GNNExplainer	Avg Mask	0.619	0.583	0.389	0.467	0.646
Beta Explainer	Avg Mask	**0.637**	0.583	**0.408**	0.480	**0.680**
NF Explainer	Avg Mask	0.595	**0.667**	0.381	**0.485**	0.669

Table 2: A summary of results from the graph embedding experiment

to its lesser performance on the other metrics but the mean of its distribution tends to capture important edges a bit better than the Beta Explainer.

5.2.1 Beta Model Edge Mask Distribution

Additional, insights can be drawn about the GNN by looking at the distribution of edge mask values that the Beta Model came up with. In figure 17, one can see the marginal distributions of edge mask values for each edge in the groundtruth. From these marginal

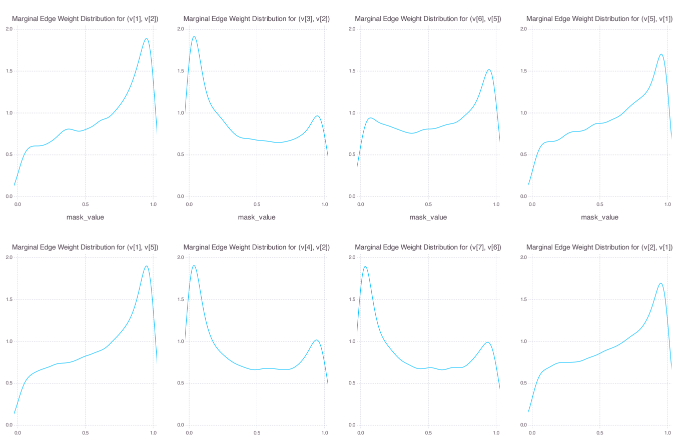

Figure 17: Marginal edge weight distributions of all groundtruth edges in the beta model for a particular training point

distributions, it is clear that messages coming out and into node 1 are highly important and that flows from node 6 to 5 are also highly important. The rest of the edges are not considered important. Note that this is for this particular example and not in general. This makes sense, though as a concentration of density in node 5 would force the model to focus on that region. Since the variational posterior is quite inflexible, the analysis from this model can be quite confident but shallow as it will only highlight the most important pathways.

Furthermore, insights can be drawn by looking at the joint distribution between pairs of edges. This can give insight into what edges are correlated, anti-correlated, and how many modes there are to the edge mask distribution. This is important as it gives researchers insights into the different mechanisms that drive performance in the GNN. If the GNN is being used in a physical sciences or biology context, for example, this can help elucidate the various mechanisms that map to physical phenomenom. As seen in figure 18, the Beta Model helps provide a great first order approximation to these types of pathways.

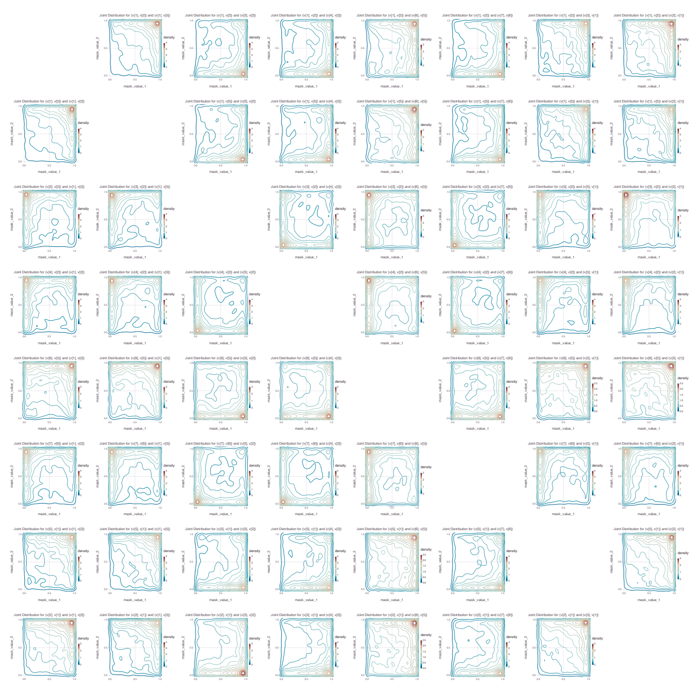

Figure 18: Joint edge weight distributions between pairs of groundtruth edges in the beta model for a particular training point

Specifically, by looking at the joint distributions of edges, one can tell when a certain pair of edges is postively coorelated, negatively correlated or lacks any correlation at all.

In figure 18, looking at edges $3 \to 2$ and $1 \to 5$ demonstrates that for this example, the model is certain that edge $1 \to 5$ is important while the edge $3 \to 2$ is not. Perhaps this is due to an accumulation of density in node 5 while there was not as much sampled at node 2. Indeed, this can be coroborrated by looking at $6 \to 5$ and $1 \to 5$ where both are very positively correlated and indicates that information is pooling at this node for this particular example. Finally, it is clear that bidirectional flows from node 1 and its neighbors are highly important as well. Overall, this gives a great unimodal intuition about the way in which information flows through the model.

5.2.2 Normalizing Flow Model Edge Mask Distribution

A similar analysis can be done for the Normalizing Flow Model. Note that this model is much more flexible than the Beta model which gives it a higher variance but allows it to capture multiple different pathways and local minima that the GNN may experience. This can already be seen in the marginal distributions as seen in figure 19.

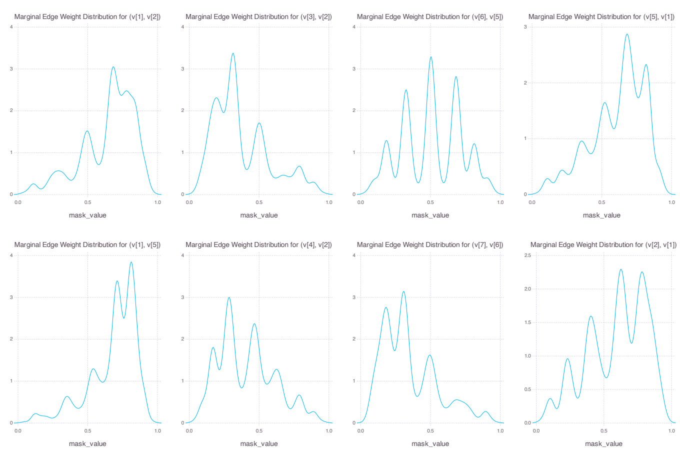

Figure 19: Marginal edge weight distributions for all groundtruth edges in the normalizing flow model for the same training point as 17

Indeed, the same distributional shape as in the Beta Model can be seen, but the Normalizing Flow model learns many more modes in the distribution. Note that this corresponds to the various local minima in the training example which is useful as it indicates that are many different modes through which the GNN passes information around. This helps explain the Normalizing Flow Model's weaker performance compared to the Beta Model.

The Beta Model can only pick out the most probable mode due to its relative inflexibility while the Normalizing Flow model picks up all the different peaks and troughs in the objective function giving it a higher variance but, essentially, correct overall distribution.

As with the Beta Model, the joint distribution between pairs of edges helps illustrate an even richer tale as seen in figure 20.

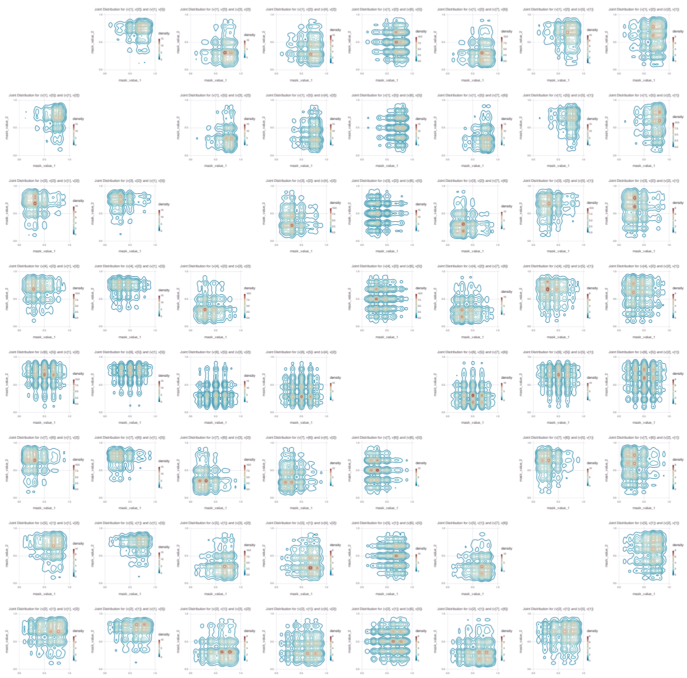

Figure 20: Joint edge weight distributions between pairs of groundtruth edges in the normalizing flow model for the same training point as 17

For example, we can now detect edge indepedence with edges $6 \to 5$ and $1 \to 2$ having little correlation with each other. If one were to condition on the modes of one of the edges, the conditional distribution could then indicate conditional pathways which clearly exist based on the multi-peaked joint distributions. While an example like this may not merit such analysis, in more complex datasets with graphs mapping to physical phenomenom, this could be of supreme importance and indicate all sorts of useful information.

6 Discussion

6.1 Comparison of Models

Overall, there a few takeaways in comparing the models. The first is that both of the Bayesian models for GNN interpretability perform better than GNNExplainer and represents and increase in the performance of GNN Interpretation methods. While the increase in performance is not too much higher than what GNNExplainer achieves, in a more complicated dataset the difference may be higher.

These Bayesian models not only offer better performance than GNNExplainer, but they also give a deeply introspectable look at the edge interpretations and allow for a detailed analysis of the influnce that the computational graph has on the GNNs performance. The Beta model serves as a low variance estimate for the first mode in the edge mask distribution and is very useful for understanding the high-level pathways of information flow in a GNN. The Normalizing Flow model, though, captures more modalities in the edge mask posterior and is able to give even further details about the GNN. This comes at the expense, though, of a higher variance estimate and more uncertainty in its predictions.

Together, though, these two models can be very helpful for extracting real-world value out of GNNs. When the computational graph maps to a physical phenomenom, the interpretations from these methods can deliver useful insights that the GNN has learned. GNNExplainer struggles to provide such value and, certainly, cannot capture all the different modalities that the normalizing flow explainer is able to pick up on.

That being said, these models are a lot more computationally expensive with GNNExplainer operating at 10x speed advantage compared to the Bayesian models. While this is quite slow, in reality, one rarely needs to perform analysis on every node on a graph and usually researchers pick notable examples for analysis [28]. Still, this is a significant speed advantage that GNNExplainer holds.

6.2 Future Work

Listed below is a inexhaustive list of various ways in which this research can be taken further.

6.2.1 Collation of Examples in Classification Datasets

One avenue of future work is to combine the ideas of PGExplainer [16] with the Bayesian inference models. Specifically, PGExplainer learns a parametrized network to generate node-level explanations that were trained across all nodes. In this sense, PGExplainer learns more general patterns in the GNN. While learning at each individual neighborhood of each node independently should converge on the same generic truth, leveraging an entire dataset in graph classification or node classification could greatly shorten the amortized inference time that is needed for this model. Especially in the graph embedding experiment, many explanations were similar and creating a parametrized version of the Bayesian inference models could help achieve lower variance estimates at a lower runtime cost.

6.2.2 Beta Model Conditional Parameter Selection

The Beta Model is quite restrictive in the set of posteriors that it can learn due to the fixed alpha and beta parameters at each node. While this allows it to achieve low variance estimates for the most important pathways in the GNN, it precludes it from learning the multi-modal distribution that the Normalizing Flow model manages to. Because of this, it may be useful to put a flexible distribution over the alpha and beta values at each node which is conditional on the edge that is being sampled. In this way, the Beta Model could become more flexible while retaining a lot of the nice qualities that it has.

6.2.3 Real-World Datasets (SERGIO and Beyond)

Finally, it would be great to apply these methods to a real-world dataset in order to demonstrate the ability of the full posterior edge mask distribution to capture important information in real-world context. A great usecase comes from computational biology and identifying gene-regulatory networks. In gene-regulatory networks, there are multiple different pathways that all interact in order to regulate gene expression. In a case like this, it would interesting to see what a GNN interpretation framework like the Normalizing Flow model and Beta model would be able to say about these networks. One place to find such a dataset is the SERGIO [19] method that was discussed in §2.4. While a synthetic dataset, it provides this exact type of causal playground upon which interpretation methods can be tested to determine their usefulness as a research tool in biological sciences and beyond.

References

[1] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 1082–1090. Association for Computing Machinery.

[2] Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. 5(1):59–68.

[3] Francesca Petralia, Won-Min Song, Zhidong Tu, and Pei Wang. New method for joint network analysis reveals common and different coexpression patterns among genes and proteins in breast cancer. 15(3):743–754.

[4] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. GNNExplainer: Generating explanations for graph neural networks.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering.

[6] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks.

[7] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. Hands-on bayesian neural networks – a tutorial for deep learning users. 17(2):29–48.

[8] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo.

[9] Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization.

[11] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. 12(4):307–392.

[12] Ivan Kobyzev, Simon J. D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. 43(11):3964–3979.

[13] Lilian Weng. Flow-based deep generative models. Section: posts.

[14] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network.

[15] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows.

[16] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network.

[17] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations.

[18] Wanyu Lin, Hao Lan, and Baochun Li. Generative causal explanations for graph neural networks.

[19] Payam Dibaeinia and Saurabh Sinha. SERGIO: A single-cell expression simulator guided by gene regulatory networks. 11(3):252–271.e11.

[20] E. N. Gilbert. Random graphs. 30(4):1141–1144. Publisher: Institute of Mathematical Statistics.

[21] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming.

[22] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch geometric.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library.

[24] Continuous univariate distributions, volume 2, 2nd edition | wiley.

[25] Marianna Bolla, Ahmed Elbanna, and Jozsef Mala. Estimating parameters of a directed weighted graph model with beta-distributed edge-weights.

[26] Hadi M. Dolatabadi, Sarah Erfani, and Christopher Leckie. Invertible generative modeling using linear rational splines.

[27] Harry Zhang. Exploring conditions for the optimality of naïve bayes. 19(2):183–198. Publisher: World Scientific Publishing Co.

[28] Jeremy Bigness, Xavier Loinaz, Shalin Patel, Erica Larschan, and Ritambhara Singh. Integrating long-range regulatory interactions to predict gene expression using graph convolutional networks. Pages: 2020.11.23.394478 Section: New Results.