m-NLIL: Inducing Multiple Interpretable Rules from Graph-Structured Data

Anonymous Author
Anonymous Institution

Abstract

As the graph-structured data being widely used in real-world applications, the need for interpretable models on graphs is becoming increasingly pressing. In recent years, graph neural networks (GNNs) have been the SOTA models in dealing with large-scale graphs due to their expressive power. Nevertheless, GNNs' prediction procedure remains non-transparent and unintelligible. In this paper, we introduce m-NLIL, a multi-rules neural inductive logic programming model for learning interpretable logical classifiers on graph-structured data. We demonstrate that m-NLIL is capable of inducing multiple diverse rules from data and has a better performance in accuracy and speed than previous SOTA models. We also discuss m-NLIL's potential for providing post-hoc explanations for black-box GNN models.

1 INTRODUCTION

Recent years have witnessed the success of Graph neural networks (GNNs)(Scarselli et al., 2009; Dai et al., 2016) in a wide range of applications that deal with graph-structured data, including social, information, and chemical domains. The success of GNNs comes from their ability to recursively incorporate information from neighboring nodes in the graph, which can naturally represent both the node features and the structure of the input data.(Hamilton et al., 2017a; Kipf and Welling, 2017)

Despite the expressive power of GNNs, their decision-making processes are not interpretable or human-intelligible, which have drawn many concerns on reliabilities, effectiveness, and ethical problems of these black-box models.(Adadi and Berrada, 2018; Guidotti et al., 2018; Hooker, 2004; Koh and Liang, 2017) As the ongoing widespread of GNNs, it becomes increasingly essential to explain GNNs' decision-making process. Many previous works have focused on the interpretability of GNNs. One line of work use heuristic methods to explain by repeatedly masking part of nodes to evaluate their importance from differences of results and find the critical sub-graphs contributing most to classification results.(Ying et al., 2019) Other works focus on local approximation with simple interpretable surrogate models which can be considered as local explanations.(Ribeiro et al., 2016; Hamilton et al., 2017b) These approaches can make impressively accurate explanations for GNNs, yet their explanations are not capable of discovering and representing human-intelligible knowledge that GNNs have learned from data.

On the other hand, first-order logic (FOL) has shown a great ability to discover and represent knowledge in graph-structured data human-intelligibly way. It has been proven that GNNs are equivalent to Weisfeiler-Lehman (WL) test(Weisfeiler and Leman, 1968) and have rich logical expressiveness.(Barceló et al., 2020) Therefore, we are motivated to construct a logical model on graphs that can learn first-order logic to explain the underlying pattern in graphs and GNNs.

Some previous works have focused on logical reasoning and explanations on graphs. (Evans and Grefenstette, 2017; Payani and Fekri, 2019; Dong et al., 2019; Yang et al., 2017). A recent paper NLIL (Yang and Song, 2020) introduced a differentiable inductive logic programming (ILP) framework solving the "learning to explain" problem. It induces logical rules from data and serves as a classifier itself based on those rules, therefore being self-explaining. NLIL reaches a competitive score with deep and black-box models in some tasks while remaining light and interpretable. However, its performance will decline significantly when the distribution of data is mixed and disordered. This makes the scores fluctuate between labels sharply and puts a limit on the application of NLIL.

Preliminary work. Under review by AISTATS 2021. Do not distribute.

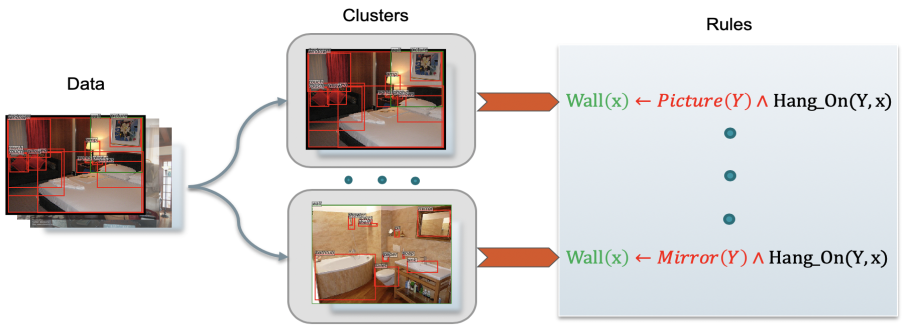
Data
Clusters
Rules
Wall(x) ← Picture(Y) ∧ Hang_On(Y, x)
Wall(x) ← Mirror(Y) ∧ Hang_On(Y, x)

Figure 1: m-NLIL can extract multiple rules from scene graph datasets

In this paper, we propose a novel framework called m-NLIL for generating multiple logical rules for reasoning and classifications on graphs, based on the previous model NLIL. An overview of m-NLIL is shown in Figure 1, We improved NLIL by introducing new methods to divide up the data and generate multiple rules respectively. m-NLIL can sort out and cluster the data iteratively to expand the searching space while keeping the data distribution inside each cluster highly centralized. These modifications guarantee m-NLIL a better performance than original NLIL, leading to an up to 10% increase in the overall score, 50% in several specific tasks. Besides, our methods are naturally parallelizable, which may reduce up to 30% of training time.

In the experiment part, we evaluate m-NLIL on its potential applications. m-NLIL can be used as a logical node classifier for graphs that reach a relatively high accuracy compared to both the original NLIL and some models involving black-box embedding methods. Furthermore, we have shown its impressive performance as a post-hoc explainer for GNNs, as long as the target model involves a message-passing process gathering structural information from graphs. All experiments are performed on scene graph datasets but can be extended to general heterogeneous graph datasets.

2 RELATED WORK

2.1 Graph Neural Networks

Graph neural networks define a general neural network architecture on structured data, e.g., protein-protein interaction networks, social networks, and knowledge graph.(Battaglia et al., 2018; Gilmer et al., 2017; Kipf and Welling, 2016a) This architecture can learn effective embedding of nodes by aggregating the feature vectors of each node with the feature vectors of its neighbors, which can represent both the node attributes and its local structure. The graph-level embedding can also be obtained from node embeddings by applying a global pooling over the node embeddings.

2.2 Inductive Logic Programming

Inductive logic programming (ILP) is a method that combines inductive methods with the power of first-order logic, which focused on learning the representation of the hypothesis as logic programs. This approach is suitable to be applied to structured data as an explanation. First, it offers a rigorous approach to the general knowledge-based inductive learning problem. Second, the use of first-order logic allows it to represent the structure and the relationship of the data rather than simply using the attributes. What's more, the form of the first-order logic is more consistent with human intelligence, which is more readable. These features allow the inductive logic programming systems can be applied in the scientific cycle of experimentation, hypothesis generation, and refutation.

2.3 Knowledge base reasoning

Knowledge bases consist of a set of predicates $P(\mathbf{x}, \mathbf{x}^{'})$, where $\mathbf{x}$ and $\mathbf{x}^{'}$ are entities and P is a binary relation between entities. The task of knowledge base reasoning consists of queries $q =< \mathbf{x}, P^{*}, \mathbf{x}^{'} >$. The task is to find a relational path $\mathbf{x} \xrightarrow{P_1} \dots \xrightarrow{P_T} \mathbf{x}^{'}$, which can connect two query entities. For each query in KB reasoning task, we hope to learn a chain-like logical rule of the following form:

$$P^{*}(\mathbf{x}, \mathbf{x}^{'}) \longleftarrow P_1(\mathbf{x}, \mathbf{y}_1) \wedge P_2(\mathbf{y}_1, \mathbf{y}_2) \wedge \dots \wedge P_T(\mathbf{y}_n, \mathbf{x}^{'})$$

Despite the chain-like logical rule can be learned easily, it has a limitation in its expressive power, since it is only a subset of the Horn clauses rule.

2.4 Connection between GNNs and Logic

It has been shown that first-order logic is powerful in representing graph-structured data. The main idea of the GNNs is to build a neural network that can reflect the input data structure. Thus it seems reasonable to use first-order logic to help us understand the procedural behavior of some fragments of GNNs. Previous work (Barceló et al., 2020) has shown that simple GNNs such as AC-GNNs are capable of capturing some of FOC_2 (first-order logic, which only allows formulas with two variables). However, it is not able to capture all the FOC_2 since GNNs are more likely to capture the local structure of the input data, and sometimes a global first-order logic on structured data might lose some local information. Thus when we try to use the first-order logic to explain the GNNs, this feature should be considered.

3 PRELIMINARIES

Graphs as a unique non-Euclidean data structure are used as the denotation of a large number of systems across various areas due to its great expressive power. Recent research on analyzing graphs with machine learning has been focused on node classification, link prediction, and graph classification. The main idea of these tasks is to learning classifiers, which can be considered as a function that maps an object from its input space to a categorical space.

Despite widespread adoption of those Graph neural networks, these models remain mostly not transparent and lack of interpretability. Recent works on interpretability have mainly focused on two aspects. One is to learn an interpretable model (such as linear regression, decision tree, and logistic regression) locally around the prediction.(Ribeiro et al., 2016; Hamilton et al., 2017b) This allows us to obtain a local explanation of the model. Another is to learn **first-order logic** rules which can be used to explain both the data pattern and the model prediction.(Evans and Grefenstette, 2017; Yang et al., 2017) To use the first-order logic rule as an explanation, we need to find the logical connection between the base classifiers or ground-truth in the data. This can be considered as a longstanding problem of first-order logic literature, i.e.,**Inductive Logic Programming** (ILP). There are several differentiable ILP models have been proposed which combine the neural network with the inductive logic programming such as Neural LP and **Neural Logic Inductive Learning**.(Yang and Song, 2020; Yang et al., 2017)

3.1 First-order Logic

The model of first-order logic(FOL) is based on domains and relations. The domain of a model is the set of objects or domain elements it contains. The objects in the model may be related in various ways. A relation is just the set of tuples of objects that are related. The certain kind of relationship can be considered as **functions**, which indicate a given object must be related to exactly one object in this way. For example, each person has one head, so the model has a unary "head" function that includes a following mapping:

$$(Person) \rightarrow Person's \quad head$$

The formula of first-order logic can be described by entities, predicate symbols, and function symbols. Here in the graph structure data $\mathcal{G} = \{\mathcal{X}, \mathcal{E}\}$, **entities** stand for nodes in the graph $\mathbf{x} \in \mathcal{X}$. For example, for a given social network, a certain person is an entity $\mathbf{x}$, and the set of all the people in the network is $\mathcal{X}$. **Predicate symbols** P represent the relations in a graph. In a social network, we can use `Friend`, `Brother` or `OnHead` as the predicate symbols. **Function symbols** simply represent the functions in first-order logic. Each predicate and function symbol comes with an arity that fixes the number of arguments.**Atom** is a predicate symbol applied to a logic variable, e.g. `Brother`($\mathbf{x}$) and `OnHead`($\mathbf{x}$, $\mathbf{y}$).

The **formula** of first-order logic is a combination of atoms using logical connectives such as `and`, `or` and `not`. Thus for a given predicate P_k, it can be explained by the following first-order logic entailment:

$$\forall \mathbf{x}, \mathbf{x}^{'} \exists Y_1, Y_2 \dots P_k(\mathbf{x}, \mathbf{x}^{'}) \leftarrow A(\mathbf{x}, \mathbf{x}^{'}, Y_1, Y_2, \dots)$$

where $P_k(\mathbf{x}, \mathbf{x}^{'})$ is the head of the entailment, it can also take in only one arguments, as $P_k(\mathbf{x})$, if it is an unary predicate. A defined the rule body of a general formula, which is made up of atoms with predicate symbols from P and logic variables including head variables $\mathbf{x}, \mathbf{x}^{'}$ and body variables $\mathcal{Y} = \{Y_1, Y_2, \dots\}$. Given the foregoing logic representation, the task of inductive logic programming(ILP) can be interpreted as learning the FOL formula for target predicate P_k.

3.2 Neural Logic Inductive Learning

To make the inductive logic programming(ILP) differentiable, NLIL introduces a TensorLog operator $\mathbf{M_k}$(Cohen, 2016). $\mathbf{M_k}$ is a matrix in $\{0,1\}^{|\mathcal{X}|\times|\mathcal{X}|}$ and its $m_{(i,j)}$ indicates the existence of certain type of relation between i-th entity and j-th entity. TensorLog defines an operator $\mathbf{M_k}$ for each relation k. With these notations, we can treat each predicate symbol as an operator. Then, a set of predicate symbols P can be considered as a subspace of the functions $\mathcal{P} = \{\rho_1, \dots, \rho_k\}$, where

$$\begin{cases} \rho_k() = \mathbf{M_k}1 & \text{if } k \in \mathcal{U}, \\ \rho_k(\mathbf{v}_x) = \mathbf{M_k}\mathbf{v}_x & \text{if } k \in \mathcal{B}. \end{cases}$$

where $\mathbf{v}_x$ is the one-hot encoding of the entity $\mathbf{x}$, $\mathcal{U}$ and $\mathcal{B}$ respectively correspond to the unary and binary predicates set. For a given object **entity** $\mathbf{x}$, ρ_k returns the set embedding of the object entities and the subject entities that satisfy the predicate P_k. By using the $\mathcal{P}$, a subspace of the functions, we can represent the variables as the operator calls. Therefore, the FOL formula can be converted into the operator form. Moreover, the task of learning the first-order rule can be transformed into searching for the appropriate chain of operator calls that can represent the variable which is equivalent to the relational path search task mentioned in section 2.3.

Despite the chain-like rule is simple and intelligible, it is not very expressive. In order to represent more complex rules, NLIL introduces the notion of **primitive statement** α. Similar to the definition of **atom**, the **primitive statement** is a predicate symbol applied to logic variables or variables that have been represented by operator calls. Primitive statements map the input from the input space to a scalar confidence score $\alpha : \mathbb{R}^{\mathcal{X}} \times \mathbb{R}^{\mathcal{X}} \rightarrow s \in [0, 1]$, their mappings are defined as:

$$\alpha_k(\mathbf{x}, \mathbf{x}^{'}) = \begin{cases} \sigma((\mathbf{M_k}1)^T(\prod_{t^{'}=1}^{T^{'}} \mathbf{M}^{(t^{'})}\mathbf{v}_{x^{'}})), & \text{if } k \in \mathcal{U}, \\ \sigma((\mathbf{M_k}\prod_{t=1}^{T} \mathbf{M}^{(t)}\mathbf{v}_x)^T(\prod_{t^{'}=1}^{T^{'}} \\ \mathbf{M}^{(t^{'})}\mathbf{v}_{x^{'}})), & \text{if } k \in \mathcal{B}. \end{cases}$$

where T is the length of the relation path and $\sigma(\cdot)$ is the sigmoid function. To find a relation path with proper length and operators, NLIL use the **path attention vector** $S_\alpha / S_\alpha^{'} = [s_\alpha^{(1)}, \dots, s_\alpha^{(T)}]^{\mathrm{T}}$ for the first and second argument of all statements, to pick the path length. And a **operator attention vectors** $S_\rho = [s_\rho^{(1)}, \dots, s_\rho^{(T)}]^{\mathrm{T}}$ to select the proper operators. Then the above equation can be relaxed into weighted sums form as follow:

$$\alpha_k(\mathbf{x}, \mathbf{x}^{'}) = \begin{cases} \sigma((\mathbf{M_k}1)^T(\kappa(s_{\alpha,k}^{'}, S_\rho)\mathbf{v}_{x^{'}})), & \text{if } k \in \mathcal{U}, \\ \sigma((\mathbf{M_k}\kappa(s_{\alpha,k}, S_\rho)\mathbf{v}_x)^T \\ (\kappa(s_{\alpha,k}^{'}, S_\rho)\mathbf{v}_{x^{'}})), & \text{if } k \in \mathcal{B}. \end{cases}$$

Furthermore, in order to extend the rule search space, NLIL constructs a hierarchical FOL formula space $\mathcal{F}$, which contains all the possible logical combinations of the primitive statement. This allows us to explore the logic combinations of primitive statements in the formula space.Since the collection of all possible combinations might be expensive, a constraint $\mathbf{C}$ is set as the maximum number of combinations. Similar to the path selection, NLIL uses **formula attention tensors** $\mathbf{S}_f, \mathbf{S}_f^{'}$ to help select the FOL formula.

Finally, in order to select the best explanation and compute the score for each query, a output attention $\mathbf{s}_o^T$ is defined over the final FOL formula space $\mathcal{F}_L$. The output score can be defined as:

$$score(\mathbf{x}, \mathbf{x}^{'}) = \mathbf{s}_o^T \mathbf{f}_L(\mathbf{x}, \mathbf{x}^{'}) \tag{1}$$

where $\mathbf{f}_L$ is the stacked outputs of all formulas in $\mathcal{F}_L$.With the above definition, a hierarchical rule space is built, in which the discrete selections of operators, statements and logic combinations are relaxed into the weight sums with respect to a series of attention parameters $S_\rho, S_\alpha, S_\alpha^{'}, S_f, S_f^{'}, \mathbf{s}_o^T$. This allows NLIL to learn more expressive FOL rules rather than chain-like rules. In order to learn the above attention parameters, NLIL proposes a stack of three Transformer networks as the attention generator. Each transformer generates the corresponding attention to make the soft selection. Given these logic representations, NLIL can be defined as optimization problem for the following equation:

$$\arg\min_{w} CrossEntropy(y, \mathbf{s_o}^T \mathbf{f}_L(\mathbf{x}, \mathbf{x}^{'})) \tag{2}$$

where w is the learnable parameter, y is the query label which indicates whether the relation exists or not. Compared with the traditional ILP method which can only learn search in chain-like rule space, NLIL can learn more expressive FOL rules due to the operator and the hierarchical rule space. And the introduces of attention allows NLIL to be differentiable and learn FOL rules that are globally consistent in the whole graph. However, since the softmax in attention will encourage extreme values, the attention usually becomes highly concentrated on one rule after convergence. As a result, some local information in the graphs might be ignored.

4 THE M-NLIL FRAMEWORK

Neural Logic Inductive Learning(NLIL) defines the values of formula in the attention vector as the score of the FOL rules as defined in Equation.1, and this model can be trained by optimizing Equation.2. However, we can only learn **one** rule for each label under NLIL, which may cause a lack of expressiveness in some complicated scenes. In this section, we introduce the m-NLIL framework based on NLIL, which can learn multiple expressive FOL rules for both graph-structured data and GNNs. Furthermore, we propose a practical optimization procedure with an effective initialization method and an alternating optimization-based algorithm to optimize m-NLIL.

4.1 Proposed Formulation

To obtain multiple rules for the target, we consider learning latent clusters in the original dataset and generating rules for each cluster. The splitting of the dataset allows us to attribute the graphs or nodes with similar patterns into a same cluster and helps us to learn distinctive rules for these patterns. For graph-structured data, we can directly learn the latent clusters in the dataset. But for GNNs, we first need to construct graph-structured data based on the predictions

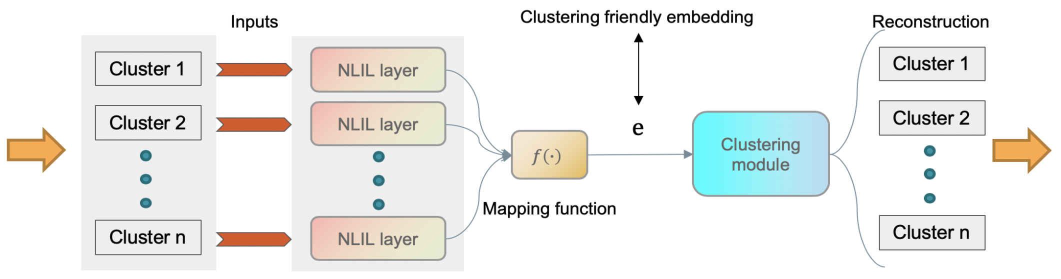

Figure 2: m-NLIL Framework

or results of GNNs which can represent the information learned by GNNs. Therefore, in order to make sure that the NLIL can learn different FOL formulas in each cluster, we can require the attention vector of each cluster to be orthogonal. And the optimization problem can be expressed as follow:

$$\begin{aligned}\arg\min_{w} \sum_{c_i \in C} \frac{1}{\mathcal{N}(c_i)} \sum_{x_i \in \mathcal{N}(c_i)} CrossEntropy(y, \mathbf{s}_{\mathbf{o}_i}^T \mathbf{f}_L(\mathbf{x}, \mathbf{x}^{'})) \\ s.t. \sum_{c_i, c_j \in C, i \neq j} \mathbf{s}_{\mathbf{o}_i}^{\mathrm{T}} \cdot \mathbf{s}_{\mathbf{o}_j} \geq \delta \end{aligned} \tag{3}$$

where $c_i \in C$ is the center of the cluster, and $\mathcal{N}(c_i)$ defines a local cluster with c_i as the center. In practice, the Equation (3) does not provide a gradient for NLIL to learn the best weight of each cluster. Moreover, the final attention vector $\mathbf{s}_{o_i}$ for each cluster might not share the same length. Thus we consider the orthogonal constraint as a clustering problem, and we hope to jointly learn both NLIL parameters and clusters. In order to jointly learn DNN parameters and clusters, some previous works (Xie et al., 2015; Yang et al., 2016) have connected a clustering module to the output layer deep learning models. Similarly, we connect a clustering module to the output layers of the NLIL, as shown in Figure2. The major difference is that in m-NLIL, we need to split the clusters and learn distinguishable NLIL parameters in each cluster. Here the clustering module is an optimization problem of the following form:

$$\min_{w,\Theta} \hat{L} = \sum_{x_i \in \mathcal{X}} l(f(x_i, w), \Theta) \tag{4}$$

where $f(x_i, w)$ is a mapping function that use the output of each NLIL to generate a clustering-friendly representation for a given data sample x_i and network parameters w, Θ is the parameters of the clustering model. The function $l(\cdot)$ is the loss function of clustering e.g., the Kullback-Leibler (KL) divergence loss (Xie et al., 2015) or agglomerative clustering loss (Yang et al., 2016).

Our motivation is to learn the latent clusters in the graph-structured data and train the NLIL to extract distinguishable FOL rules as an explanation. These two problems can be optimized simultaneously. One critical problem here is that clusters may "collapse" during the training process(i.e., entities from different clusters before are reassigned to the same cluster). One reason is that some datasets do not contain latent clusters. In this case, the splitting of the data will not attribute to the improvement of the accuracy of the model. Therefore, it is acceptable to have such "collapse". Another reason responsible for this problem is the learning of trivial FOL formulas. Since trivial rules will have a relatively high score in each cluster, most data will be assigned to the same cluster and results in the "collapse". In practice, these trivial rules will significantly reduce the expressiveness of our model, so it is one of the major problems we are focusing on.

Nevertheless, it is unavoidable to generate some trivial FOL rules during the NLIL training process. Thus the key to preventing this problem lies in the cluster reconstruction part. To make entities in the graph not so easily be reassigned to another cluster, we add a punish term according to the distance between the node and its new cluster center to control the reconstruction of clusters. The loss function we come up with is as follow:

$$\begin{aligned}\min_{w,M,\mathcal{N}} \sum_{c_i \in C} \frac{1}{\mathcal{N}(c_i)} \sum_{x_i \in \mathcal{N}(c_i)} \mathcal{L}(y_i, NLIL_{c_i}(\mathbf{x_i}, \mathbf{x_i}^{'})) \\ +\lambda ||f(\mathbf{x_i}, w_{c_i}) - c_i||_2^2 + \lambda ||f(\mathbf{x_i}^{'}, w_{c_i}) - c_i||_2^2 \end{aligned} \tag{5}$$

where the $\mathcal{L}(\cdot)$ is the loss function which can be the CrossEntropy as described above and w_{c_i} is the NLIL network parameters of cluster c_i. $NLIL_{c_i}(\mathbf{x_i}, \mathbf{x_i}^{'})$ is the output of NLIL network, as defined in Equation (7), in cluster c_i given the query $< \mathbf{x_i}, P_i, \mathbf{x_i}^{'}, y_i >$. Here, y_i indicates whether the predicate P_i exists or not. We have to note that different NLIL is trained respectively in each cluster. And λ is a regularization parameter that balances the reconstruction error and the distance to the cluster center. The $\mathcal{N}(\cdot)$ and c_i are the parameters of the clustering model which map the input node into cluster space. For instance, if K-Means(Lloyd, 1982) is used as the clustering model, then the $\mathcal{N}(\cdot)$ and c_i will correspond to the assignments $\{s_i\}$ and centroids M.

4.2 Optimization Procedure

Optimizing the Equation (5) is challenging since both the cost function and the constraints are non-convex. The splitting of clusters makes the dataset discrete, and we cannot jointly optimize all the parameters. Since m-NLIL is a framework that connects the NLIL network with the clustering network, we propose a pragmatic optimization procedure, including an initialization method and an alternating optimization algorithm. The optimization procedure of the m-NLIL framework is also shown in Figure 2. This approach allows us to transfer the original optimization problem into two sub-problems, which fix one of the clusters or FOL formula while keeping optimizing the other until the convergence of these two steps.

Initialization In order to learn the latent clusters in the graph-structured data, it is essential to initialize the cluster properly. For the explanation of GNN models, we can use the embedding as the input of the clustering model and perform KMeans to obtain initial values of $\mathcal{N}(\cdot)$ and c_i. But for the explanation of Graph-structured data, there is no embedding that can be used for initialization. One easy way to initialize in Graph-structured data is to split the data by the node class frequency; this method is simple but can have a good result on many datasets such as GQA.(Hudson and Manning, 2019) Another method we can consider is to use a pre-trained model to help generate the clustering-friendly embedding. For instance, we can use the ExpressGNN (Zhang et al., 2020) as the pre-trained model to generate the embedding for the Knowledge Base Graph. Besides, the initialization of different classifying tasks is also different. For node classification tasks, the initialization splits the original large-scale graph to obtain subgraph with more local structures information. For graph classification tasks, we hope to split the graph with similar structures feature into the same cluster which allows m-NLIL to extract the specific FOL.

Update NLIL parameters To optimize NLIL, we fix the clusters parameters $(\mathcal{N}(\cdot), c_i)$. Then the subproblem is to optimize NLIL with an additional penalty term on the clustering performance. Since NLIL is a differentiable ILP model, we can use back-propagation (SGD, 1988) based SGD to update the parameters. For each data sample x_i, we can optimize following this formula:

$$\min_{w} L_i = \mathcal{L}(y_i, NLIL_{c_i}(\mathbf{x_i}, \mathbf{x_i}')) + \lambda||f(\mathbf{x_i}, w_{c_i}) - c_i||_2^2 + \lambda||f(\mathbf{x_i}', w_{c_i}) - c_i||_2^2 \qquad (6)$$

Since NLIL is differentiable and the mapping function $f(\cdot)$ can be designed to be differentiable, the loss can back-propagate through the attentions into the Transformer networks for training.

Update clustering parameters To update the clustering parameters, we need to update both the node assignment matrix and centroids. The reassignment of the current sample can be naturally updated as follows:

$$\mathcal{N}(c_k) \leftarrow \mathbf{x_i}, \mathbf{x_i}' \quad k = \arg\min_{j} \mathcal{L}(y_i, NLIL_{c_j}(\mathbf{x_i}, \mathbf{x_i}')) + \lambda||f(\mathbf{x_i}, w_{c_j}) - c_i||_2^2 + \lambda||f(\mathbf{x_i}', w_{c_j}) - c_i||_2^2 \qquad (7)$$

This Equation (7) allows us to reassign the current sample to a cluster whose NLIL model has the best performance while constraining it to be attributed to a cluster with trivial FOL formulas.Also, we need to update the centroids, which can be done by simply taking the average of the current cluster's samples. The procedure of this alternating optimization is summarized in Algorithm 1

Algorithm 1 m-NLIL

```
Input: graph $\mathcal{G} = \{\mathcal{X}, \mathcal{E}\}$ , Input embedding $E$
Output: clusters $\mathcal{N}$ and the corresponding rules $rules$.
1: Initialization: $\mathcal{N}, C \leftarrow Clustering(E)$
2: while not converge do
3:     for $c_i$ in $C$ do
4:         $rules, w \leftarrow NLIL(\mathcal{N}(c_i))$
5:     end for
6:     update assignment by (7)
7:     update centroids
8: end while
```

5 Experiment

In the experiment section, we first evaluate our m-NLIL in triple classification task of the gqa scene graph dataset. (Hudson and Manning, 2019) We demonstrate the improvement in both accuracy and training cost of m-NLIL than the original model, besides its comparable classification capability with black-box embedding + neural network classifier. We also focus on its ability of concluding multiple rules for more accurate classification and more human-intelligible interpretations. What's more, we will show m-NLIL's performance on some practical applications, including serving as a post-hoc explainer for other GNNs.

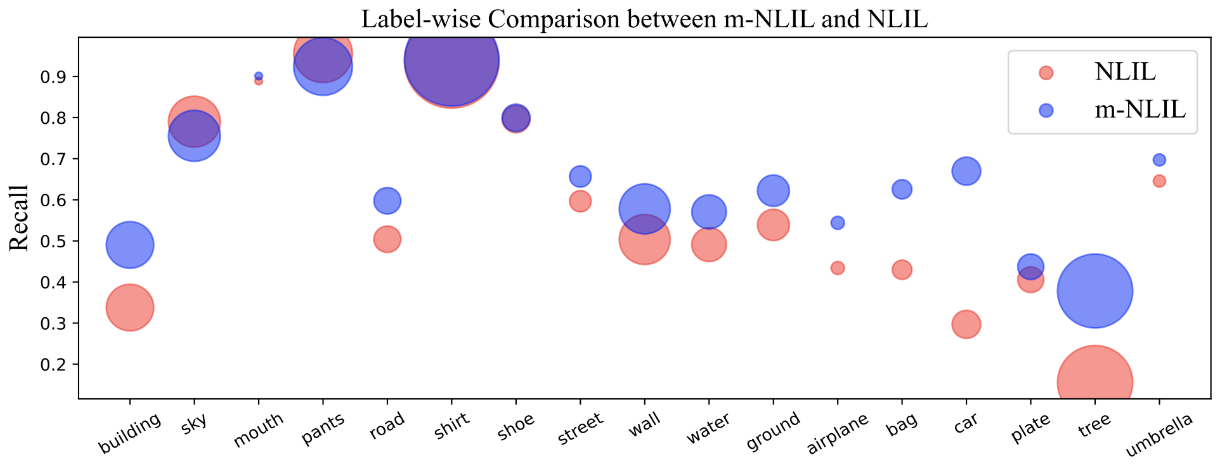

Figure 3: Recall scores comparison m-NLIL

5.1 Triple Classification on gqa

Triple Classification This task involves querying whether a certain triple $(ent1, pred, ent2)$ exists in a graph. In our experiment setting, all the triples are in the form like $(ent, type, ent)$, where the head and tail entity are the same nodes and the predicate becomes the type of this node. This makes the triple classification very similar to general classification tasks and therefore having a broader field of applications.

Table 1: Recall Scores of different models

Model	Recall Score
Baseline-freq	0.40
Baseline-MLP	0.53
NLIL	0.51
m-NLIL (Our Model)	**0.58**

The gqa Scene Graph Dataset The original gqa (Zellers et al., 2017) dataset is for "question answering on scene graphs generated from the image." In our experiments, we only use the scene graph part as the target for classification. There are two kinds of predicates in the gqa scene graph. 1) Edge predicates $(ent1, relation, ent2)$. 2) Node predicates $(ent, type)$. The latter will be our target of prediction.

The average recall results are shown in Table 1. Some of the data and baseline models come from (Yang and Song, 2020). Compared with the original NLIL, our m-NLIL shows a noticeably higher recall. And it even outperforms the baseline embedding classification method, which is a lot more complicated and a total black box while m-NLIL is only an interpretable, simple logical classifier.

A more detailed comparison with NLIL is shown in

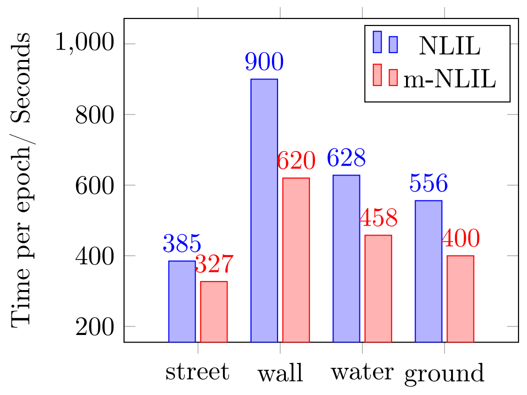

Figure 4: efficiency improvement of the training epoch on some labels

Figure 3 and 4. Figure 3 shows a label-wise recall score comparison between m-NLIL and NLIL, where the size of markers indicates the size of corresponding label in our training dataset. As is shown, m-NLIL outperforms NLIL in almost every label with up to 50 percent improvement. Figure 4 demonstrates the improvement of efficiency on NLIL. The training stage of our algorithm is naturally parallelizable without much need for sharing parameters; therefore, the speed-up ratio is close to theoretical maximum, in our case, depending on how long it takes to process the largest cluster.

From Figure 3, we can see that the improvement of labels with higher scores in the original NLIL is trivial, but those with low scores before gain a significant increase in classification performance. The major reason is that m-NLIL induces rules in smaller clusters rather than all the graphs in a label. NLIL is capable of finding the proper global rule for those labels which have a consistent global rule, while it will fail to extract rules from the smaller, local structure. As a result, the label with a low score in NLIL might have a significant improvement, since m-NLIL can learn multiple rules

Table 2: Explanations generated by m-NLIL

Label	**Rules**
`street`	Car(Y)$\wedge$On(Y, X) & Bus(Y)$\wedge$On(Y, X) & Road(Y)$\wedge$Near(Y, X)
`water`	Boat(Y)$\wedge$In(Y, X) & Surfboard(Y)$\wedge$In(Y, X) & Person(Y)$\wedge$In(Y, X)
`ground`	Snow(Y)$\wedge$On(Y, X) & Grass(Y)$\wedge$On(Y, X) & Person(Y)$\wedge$StandingOn(Y, X)
`wall`	Picture(Y)$\wedge$On(Y, X) & Mirror(Y)$\wedge$On(Y, X) & Clock(Y)$\wedge$On(Y, X)

(a) ground-truth

Label	**Rules**
`street`	Car(Y)$\wedge$On(Y, X) & Person(Y)$\wedge$On(Y, X) & Sign(Y)$\wedge$On(Y, X)
`water`	Boat(Y)$\wedge$On(Y, X) & Flowers(Y)$\wedge$In(Y, X) & Sand(Y)$\wedge$In(Y, X)
`ground`	Person(Y)$\wedge$On(Y, X) & Train(Y)$\wedge$On(Y, X) & Bird(Y)$\wedge$On(Y, X)
`wall`	Picture(Y)$\wedge$On(Y, X) & Mirror(Y)$\wedge$On(Y, X) & Bed(Y)$\wedge$Near(Y, X)

(b) GCN-gqa

Label	**Rules**
`street`	Car(Y)$\wedge$On(Y, X) & Bike(Y)$\wedge$On(Y, X) & Sign(Y)$\wedge$On(Y, X)
`water`	Boat(Y)$\wedge$Below(Y, X) & Boat(Y)$\wedge$Near(Y, X) & Person(Y)$\wedge$In(Y, X)
`ground`	Leaf(Y)$\wedge$On(Y, X) & Grass(Y)$\wedge$On(Y, X) & Basket(Y)$\wedge$On(Y, X)
`wall`	Picture(Y)$\wedge$On(Y, X) & Mirror(Y)$\wedge$On(Y, X) & Number(Y)$\wedge$Below(Y, X)

(c) MLP

in different local clusters. However, for the targets which can be explained with a single global rule, m-NLIL might not be beneficial. Besides, m-NLIL is less data-hungry, as it shows a much better performance on small labels than NLIL.

5.2 Post-hoc Explainer for GNN classifier

m-NLIL can directly serve as a post-hoc explainer for GNNs by changing the input label from ground truths to results from other GNNs to be explained. The output concluded logical rules could serve as an explanation of the given model classifier.

Target for Explanation We choose two other models on the gqa scene graph classification task. 1) GCN-gqa: This is a typical 2-layer Graph Convolution Network(GCN)(Kipf and Welling, 2016b) framework we built on the gqa dataset. There have been works on transferring GCN to small-scale relation graphs, like LCGN(Hu et al., 2019), a question-answering model on gqa based on Graph Convolution. Thus explaining them becomes more and more important. 2) Our MLP baseline: It uses a Multi-Layer Perceptron (MLP)(Rosenblatt, 1957) as the back-end classifier that directly processes node features and classifies them.

Table 2a, 2b and 2c are three tables showing explanations for ground-truth, GCN-gqa and MLP. Each model pay more attention to different parts in the scene graphs while the target labels for classification are the same. Notice that in Table 2, explanations to MLP is less reasonable than the ones to GCN-gqa. This is because MLP directly uses node features as input, while GCN involves Message Passing layers that utilize structural information in graphs. m-NLIL relies solely upon structures to induce explanations; therefore, it requires the target classification model uses messages from both nodes and edges to get better explanations.

6 Conclusion

In this paper, we proposed m-NLIL, a framework that combines a differentiable ILP framework with a clustering model. m-NLIL works by jointly optimize both NLIL and clustering, which provides a way to learn the local explanatory rules for both graph-structured data and GNN models. We demonstrate that m-NLIL can extract more diverse and accurate rules from the data compared with NLIL. Furthermore, the structure of m-NLIL enables parallel optimization which can improve efficiency. In the future, we plan to transfer this framework to explain graph adversarial attack models which can help provide robustness against adversarial attacks on GNNs.

Acknowledgements

All acknowledgments go at the end of the paper, including thanks to reviewers who gave useful comments, to colleagues who contributed to the ideas, and to funding agencies and corporate sponsors that provided financial support.

References

(1988). Neurocomputing: Foundations of research.

Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160.

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J., and Silva, J. (2020). The logical expressiveness of graph neural networks.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H. F., Ballard, A. J., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K. R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261.

Cohen, W. W. (2016). Tensorlog: A differentiable deductive database. *CoRR*, abs/1605.06523.

Dai, H., Dai, B., and Song, L. (2016). Discriminative embeddings of latent variable models for structured data. 48:2702–2711.

Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. (2019). Neural logic machines.

Evans, R. and Grefenstette, E. (2017). Learning explanatory rules from noisy data. *CoRR*, abs/1711.04574.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212.

Guidotti, R., Monreale, A., Turini, F., Pedreschi, D., and Giannotti, F. (2018). A survey of methods for explaining black box models. *CoRR*, abs/1802.01933.

Hamilton, W., Ying, Z., and Leskovec, J. (2017a). Inductive representation learning on large graphs. pages 1024–1034.

Hamilton, W., Ying, Z., and Leskovec, J. (2017b). Inductive representation learning on large graphs. pages 1024–1034.

Hooker, G. (2004). Discovering additive structure in black box functions. pages 575–580.

Hu, R., Rohrbach, A., Darrell, T., and Saenko, K. (2019). Language-conditioned graph networks for relational reasoning. *CoRR*, abs/1905.04405.

Hudson, D. A. and Manning, C. D. (2019). GQA: a new dataset for compositional question answering over real-world images. *CoRR*, abs/1902.09506.

Kipf, T. N. and Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

Kipf, T. N. and Welling, M. (2016b). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

Kipf, T. N. and Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks.

Koh, P. W. and Liang, P. (2017). Understanding black-box predictions via influence functions. 70:1885–1894.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

Payani, A. and Fekri, F. (2019). Inductive logic programming via differentiable deep neural logic networks. *CoRR*, abs/1906.03523.

Ribeiro, M., Singh, S., and Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. pages 97–101.

Rosenblatt, F. (1957). The perceptron - a perceiving and recognizing automaton. *Cornell Aeronautical Laboratory Report*, 85.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80.

Weisfeiler, B. Y. and Leman, A. A. (1968). Reduction of a graph to a canonical form and an algebra arising during this reduction.

Xie, J., Girshick, R. B., and Farhadi, A. (2015). Unsupervised deep embedding for clustering analysis. *CoRR*, abs/1511.06335.

Yang, F., Yang, Z., and Cohen, W. W. (2017). Differentiable learning of logical rules for knowledge base completion. *CoRR*, abs/1702.08367.

Yang, J., Parikh, D., and Batra, D. (2016). Joint unsupervised learning of deep representations and image clusters. *CoRR*, abs/1604.03628.

Yang, Y. and Song, L. (2020). Learn to explain efficiently via neural logic inductive learning.

Ying, R., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. (2019). GNN explainer: A tool for post-hoc explanation of graph neural networks. *CoRR*, abs/1903.03894.

Zellers, R., Yatskar, M., Thomson, S., and Choi, Y. (2017). Neural motifs: Scene graph parsing with global context. *CoRR*, abs/1711.06640.

Zhang, Y., Chen, X., Yang, Y., Ramamurthy, A., Li, B., Qi, Y., and Song, L. (2020). Efficient probabilistic logic reasoning with graph neural networks.