Knowledge-Based Systems 250 (2022) 109172

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Dynamic graph dropout for subgraph-based relation prediction

Sijie Mai[1], Shuangjia Zheng[1], Ya Sun[1], Ying Zeng, Yuedong Yang, Haifeng Hu *

Sun Yat-sen University, Guangzhou, 510006, Guangdong, China

ARTICLE INFO

Article history:
Received 19 February 2022
Received in revised form 27 May 2022
Accepted 28 May 2022
Available online 3 June 2022

Keywords:
Inductive relation prediction
Interpretability
Graph neural network
Explainability

ABSTRACT

Relation prediction for knowledge graphs aims at predicting missing relationships between entities. Inductive relation prediction methods have received increasing attention because of their capability of handling unseen entities. Among the inductive methods, the subgraph-based algorithms have emerged, which inductively predict relations using the subgraph surrounding the target entities. Despite the effectiveness, prior subgraph-based studies rarely focus on the explainability of subgraph reasoning, with which is critical for humans to understand and trust the prediction from GNNs. One of the reasons for the weak explainability of subgraph-based methods is the existence of noisy nodes and edges, which also hinders having higher performance of the model. In this paper, we present a dynamic graph dropout algorithm that prunes irrelevant nodes and edges to find the minimum sufficient subgraph for relation prediction. By this means, the proposed algorithm provides an explanation for which parts of the subgraph the model derives its results from. Specifically, we design an estimation function to evaluate the importance of the nodes and edges. Two dropout functions, i.e., soft and hard dropouts, are elaborately designed to filter out noisy nodes and edges based on their importance. Moreover, multiple restriction losses, including topological loss and penalty loss, are proposed to regularize the generation of the pruned subgraph. In this way, our model seeks to preserve the topology information in the subgraph and meanwhile maximally eliminate the redundant information. By removing noisy information, the proposed algorithm outperforms state-of-the-art models on eight inductive datasets.

1. Introduction

Knowledge graphs (KGs) are collections of factual information represented in the form of triplets [1,2]. Each triplet is organized as (s, r, t), where s and t represent the head entity and the tail entity, and r is the relation between s and t. KGs play an important role in various tasks due to their excellent ability to structure complex information [3–5]. However, due to the limitations of human knowledge and extraction algorithms, KGs are always incomplete because some links between entities are absent. To fulfill this absence, researchers design various algorithms to reduce the gap between KGs and real-world knowledge, which are referred to as the relation prediction or knowledge graph completion task.

Embedding-based methods are the mainstream approaches among all relation prediction algorithms [6–9]. After projecting entities and relations into a low-dimensional feature space, these algorithms learn the entity and relation embeddings, and use those embeddings to predict missing relations. Although promising, most of the embedding-based work is transductive, assuming that the entities in the KGs are unchanged and ignoring the evolving nature of KGs. In realistic scenarios, many KGs are ever-evolving with new entities emerging over time (e.g., new users in social media platforms) [10]. Since the embeddings of new entities are unavailable during training, embedding-based models cannot infer the relations for these entities. For practical applications, it is time and computationally consuming to perform tedious retraining every time a new entity is encountered. Therefore, the ability to infer relations for new entities without costly retraining from scratch is needed. In contrast to embedding-based models, another popular direction for relation prediction is to induce probabilistic logical rules by enumerating statistical regularities and patterns that occur in the KGs [11–13], which is entity-independent and inherently inductive. Moreover, logical rules provide an explanation for how the model infers relations between entities in a human-understandable way. However, these methods suffer from scalability issues due to having a rule-based nature or are less expressive since they do not account for the neighborhood structure surrounding the triplets [14,15].

To further promote the progress of inductive relation reasoning, recent subgraph-based methods provide ways to predict relations from the graph topology structure surrounding the target triplet [14–17]. Subgraph-based methods first extract the enclosing subgraph surrounding the target head and tail entities,

* Corresponding author.
E-mail address: huhaif@mail.sysu.edu.cn (H. Hu).
[1] These authors contribute equally.

https://doi.org/10.1016/j.knosys.2022.109172
0950-7051/

define the initial node embeddings according to the topology information, and finally score the subgraph using a graph neural network (GNN). Since the node embedding is defined as the topological position of each entity in the subgraph and there is no need for domain-related knowledge of these emerging entities, the subgraph-based methods are inherently inductive. In addition, a subgraph can be interpreted as the combination of valid paths between two target entities, and thus, it is more comprehensive and informative than a single rule. Despite the effectiveness of subgraph-based methods, few studies have attempted to make the predictions from subgraphs explainable. Compared to rule-based methods, subgraph-based methods are similar to a black-box whose predictions are not explainable, making the inferred results not trustworthy. In other words, we cannot infer which parts of the subgraph these algorithms derive their results from.

In fact, the explainability of GNNs is significant in attaining reliability of the models [18–22]. Only when we know how the model generates predictions based on the subgraph can we decide whether to trust the prediction from the model. Since relation prediction relies on logical rules that are not always intuitive to humans, generating the explanation of relation prediction for human understanding becomes more significant. Generally, the weak interpretability of common subgraph-based methods lies in the complexity of the graph structure. Compared with a single path, it is more difficult to explain the behavior of subgraphs in an intuitive way. Moreover, because a subgraph can be very large, it contains noisy information that interferes with the learning of the model. Therefore, filtering out noisy information can be helpful for the prediction, which is not considered in previous algorithms [14–16].

In this paper, we introduce an explainable subgraph-based relation prediction algorithm that focuses on improving the explainability and performance of the model. In contrast to the commonly considered *post-hoc explanation* applied in the popular graph explanation algorithms that generates explanation on trained GNNs [19,21,23], our algorithm manages to perform explanations and improve the performance of relation prediction simultaneously. From the perspective of improving model performance, the proposed algorithm can filter out noisy edges and nodes that negatively influence the prediction of relations. Note that previous graph explanation models tend to mask either nodes or edges [18,19,21,23]. We argue that when the noisy nodes contain numerous neighbors, it is not easy to explicitly mask all of the edges that connect the noisy nodes, especially in a dense graph where the degrees of the nodes are large. Moreover, noisy nodes can still influence the learning when generating the graph embedding for a prediction. In contrast, when the informative nodes contain noisy edges, masking nodes alone cannot filter out these edges. Therefore, we propose to prune the nodes and edges simultaneously to obtain a more compact and explainable subgraph (see Fig. 1).

Specifically, for each node and edge, we first learn an estimation function that comprehensively considers the relevant information of the specific node/edge and evaluates whether it can be dropped out. Two dropout functions, i.e., soft dropout and hard dropout, are then elaborately designed to filter out noisy edges and nodes, where hard dropout outputs a binary score for nodes/edges and soft dropout outputs a continuous score that is approximately binary. Moreover, we encourage the model to drop edges and nodes as much as possible under the constraint of penalty loss. Since subgraphs contain complex topological structures and relation prediction relies on logical rules that connect target entities, dropping out some nodes might cause the truncation of logical rules, especially when the degree of the node is large. Therefore, we further develop a topological loss on the node dropout to encourage the remaining part of the pruned subgraph to constitute a complete logical rule between target entities. By this means, the model can seek to preserve important logical paths for prediction and enhance the explainability. Moreover, we also encourage the pruned subgraph to be connective instead of separated by introducing the neighboring topological loss.

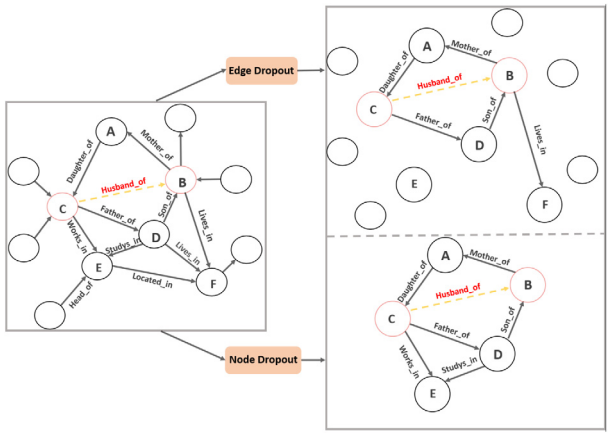


Fig. 1. The schematic diagram of the edge dropout and the node dropout.

In brief, the main contributions of this paper are listed below:

- We build an inductive relation prediction framework, termed GraphDrop, to enhance the explainability of relation prediction from enclosing subgraphs. Different from previous GNN explanation methods that explain well-trained GNNs [19,21,23], we train the model and perform explanation dynamically in an end-to-end manner, which not only generates the prediction but also reveals how the model obtains the prediction.
- We elaborately develop an estimation function and two dropout functions to estimate the importance of nodes/edges and then filter out noisy nodes/edges. We propose maximally removing nodes and edges in the subgraph via penalty loss to find the relational inference pattern of subgraphs. In addition, we seek to preserve the logical rules and topology structure in the subgraph via the proposed topological loss.
- We conduct extensive experiments to evaluate the effectiveness of our method. Moreover, we show that our subgraph-based framework is able to infer the logical rules to perform relation prediction in the subgraph.

2. Related work

2.1. Transductive relation prediction

Learning entity and relation embeddings for KGs has been an active research area in recent years due to its wide applications. Typical KG embedding models include TransE [6], Distmult [24], ComplEx [25], ConvE [7], RotatE [26], PairRE [27], etc. These methods learn relation and entity embeddings using the triplet data in the KGs and then infer the possibility of target triplet. However, they process each triplet separately without considering the semantic and structure information embedded in rich neighborhoods of triplets. Recently, GNN-based methods have been proposed to capture global structural information inherently stored in KGs and shown outstanding performance [8,28–31]. However, the above approaches mostly work on transductive settings and cannot process unseen entities. Moreover, the predictions from most of these methods are not always explainable and credible.

2.2. Inductive relation prediction

A research direction of inductive representation learning is to introduce additional attributes of entities, such as description text or images to embed unseen entities [32–34]. Although the generated embeddings can be utilized for relation prediction, these methods heavily rely on the external resources which are always unavailable in real-world scenarios. To resolve this issue, several inductive KG embedding models [35,36] are proposed to generate embedding of the emerging entity by aggregating information of its existing neighbors in the original KG. Nevertheless, these approaches require the new nodes to be surrounded by known nodes and cannot handle entirely new entities.

Another research direction is rule-based approaches which use observed co-occurrences of frequent patterns in the KG to recognize logical rules for each specific relation [13]. The rule-based methods are inherently inductive as logical rules are independent of entities. However, these approaches suffer from scalability issues and lack of expressive power due to their rule-based nature. Inspired by these statistical rule-induction approaches, several differentiable rule learners including NeuralLP [12], RuleN [11], and DRUM [37] are proposed to learn logical rules as well as confidence scores from KGs in an end-to-end paradigm. However, they do not take account of the neighbor structure surrounding the entities, and thus are not expressive enough when paths between the target head and tail entities are sparse.

Recently, to leverage the expressive power of GCNs and topological information between the target entities to infer relation, subgraph-based inductive relation prediction methods have been proposed and received very competitive results. GraIL [15] extracts the enclosing subgraph between target entities, and uses an attentive R-GCN [29] to score the subgraph. Furthermore, CoMPILE [14] extracts the directed subgraph where the relation flows from target head entity to target tail entity, and it introduces the bidirectional communicative message passing mechanism between edges and nodes to score the subgraph. Meta-iKG [16] focuses on handling few-shot relations, which proposes a meta-learning-based inductive algorithm. TACT [17] follows GraIL [15] to extract local subgraph between entities, and it further uses a relational correlation network to model topological patterns of the relations. However, these methods focus little on the explainability of the prediction from subgraph and do not consider filtering out irrelevant nodes and edges.

2.3. Graph explanation

Graph Neural Netstudies (GNNs) have emerged as powerful tools for effectively modeling graph structured data. Since GNNs are increasingly being applied in real-world applications such as drug repurposing [38,39], it becomes significant to ensure that humans can understand and trust their predictions [23,40]. Only when humans understand how models generate predictions can they determine when and how much to rely on these models and detect potential biases or errors of the models. To address the explainability issue, several algorithms have been proposed to explain the predictions from GNNs [18–20,23]. These approaches can be roughly characterized into perturbation-based [19,21,23], gradient-based [41], and surrogate model based [42,43] methods. These algorithms generally focus on one of the following aspects: (1) the importance of the edges; (2) the importance of the nodes; (3) the importance of node features; and (4) the discovery of graph patterns that can represent a certain class. Based on what types of explanations are provided, these algorithms can be further categorized into two main classes: instance-level methods and model-level methods. Instance-level methods provide input-dependent explanations for each graph. Given a graph, these methods explain GNNs by identifying important features, nodes, and edges for prediction [18–20,23,41]. In contrast, model-level methods explain GNNs in an input-independent perspective and focus on the general behaviors of the models. A typical model-level explanation method is XGNN [44]. It generates graph patterns to maximize the predicted probability for a specific class and uses graph patterns to explain this class.

The proposed algorithm can be classified as instance-level method, but it has major differences from other methods. To the best of our knowledge, we are the first to provide explanations from GNNs for subgraph-based inductive relation prediction. Moreover, an estimation function is designed before dropout to evaluate the contribution of each node/edge more comprehensively, and we seek to remove both the edges and nodes instead of either of them via the proposed soft/hard dropout functions. Furthermore, we design the topological loss to regularize the dropout function such that the generated pruned graph can still preserve the necessary topological and logical information to infer relation. Last but not least, our explanation is not *post-hoc interpretability*, i.e., we train the model and generate its explanation dynamically in an end-to-end manner. Note that Xu et al. [45] develop an explainable GNN to conduct multi-hop reasoning on KG. However, it differs greatly from our work. Firstly, the task in [45] is multi-hop reasoning which uses target head and the target relation to search for the unknown target tail. Its generated subgraph starts from the target head and expands until the target tail is found, where the attention mechanism is used to control the complexity of the subgraph. In contrast, we extract the full subgraph between target entities and then dynamically remove nodes and edges. Moreover, the explanation in [45] is provided by attention mechanism, while the proposed GraphDrop is provided by the proposed dropout function.

3. Algorithm

3.1. Notations and task definition

A triplet in the knowledge graph G is denoted as (s, r, t) where s, r, and t denote the head entity, relation, and tail entity, respectively. Inductive relation reasoning aims to score the plausibility of the target triplet (s_T, r_T, t_T), where the representations of s_T and t_T are unavailable during evaluation. In our algorithm, our task is to enable the relation reasoning from the enclosing subgraph to be explainable and improve the performance of the model by filtering out noisy information.

In the architecture of subgraph-based inductive relation prediction, an enclosing subgraph is used to represent the target triplet (s_T, r_T, t_T). The enclosing subgraph between the target head s_T and target tail t_T entity is denoted as $G_s = (V, E)$ where V and $E \in V \times V$ denote the sets of nodes and observed edges, respectively. We use N_e to represent the number of edges in the subgraph. The embeddings of the nodes is denoted as $\boldsymbol{N} \in \mathbb{R}^{N_n \times d_n}$ where N_n is the number of the nodes in the subgraph. The relation embedding is denoted as $\boldsymbol{R} \in \mathbb{R}^{N_r \times d}$ (N_r is the number of relations), which is parameterized as learnable parameter updated by gradient descent and is shared across train and test graphs. The summary of notations is shown in Table 1, the overall algorithm is summarized in Algorithm 1 and the pipeline of GraphDrop is illustrated in Fig. 2.

3.2. Subgraph extraction

In this section, we illustrate the procedure to extract the enclosing subgraph G_s between target entities in the knowledge graph G. Generally, the enclosing subgraph expands with the increase in the hop number h, where $h+1$ is the maximum distance

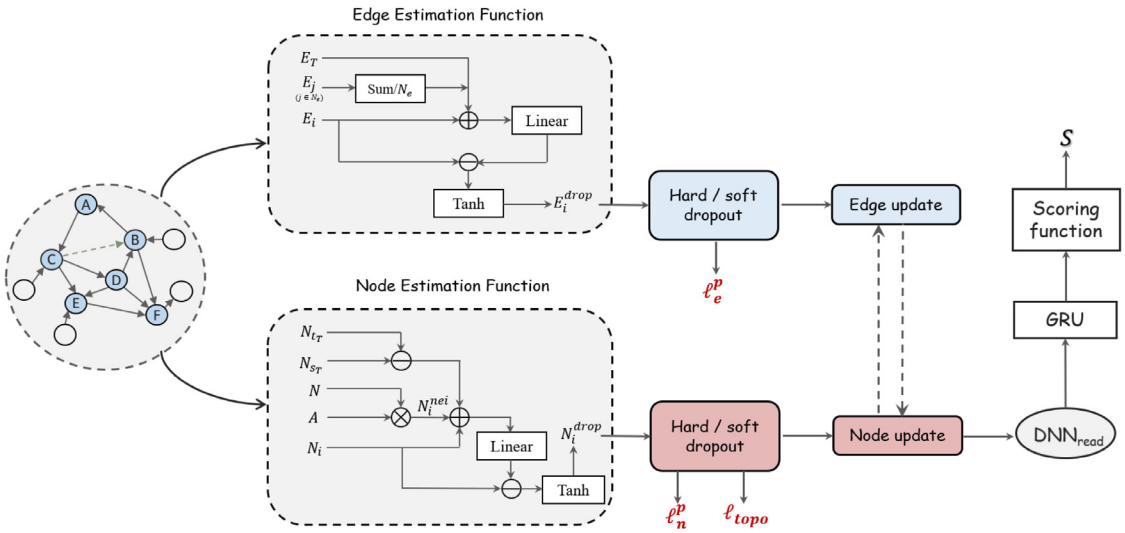


Fig. 2. The structure of the proposed GraphDrop. The estimation function, the hard/soft dropout, and the topological loss ℓ_{topo} are elaborately designed in GraphDrop to filter out uninformative nodes and edges, such that the predictive performance and the explainability of GNNs can be improved.

Table 1
Table of notations.

Notations	Descriptions
(s, r, t)	A triplet in the knowledge graph.
(s_T, r_T, t_T)	The target triplet to be predicted.
$\boldsymbol{N}$	Node embedding
$\boldsymbol{E}$	Edge embedding
$\boldsymbol{N}^0$	Node embedding after dropout.
$\boldsymbol{E}^0$	Edge embedding after dropout
$\boldsymbol{R}$	Relation embedding
$\boldsymbol{A}^{he}$	The adjacency matrix connecting head and edge.
$\boldsymbol{A}^{te}$	The adjacency matrix connecting tail and edge.
$\boldsymbol{A}^{re}$	The adjacency matrix connecting relation and edge.
$\boldsymbol{A}$	The adjacency matrix connecting head and tail.
s	The dropout value for node or edge.
l	The number of iterations in GNN.
α	A scalar determining the weight of the restriction losses.
β	A scalar determining the weight of the topological losses.
ℓ	The overall loss

Algorithm 1: GraphDrop

Input: The knowledge graph G, the target triplet (s_T, r_T, t_T).
Output: The prediction score S of the target triplet.
1. Extract subgraph G_s for target entities in the G
2. Initialize node embedding $\boldsymbol{N}$ and edge embedding $\boldsymbol{E}$, $k \longleftarrow 0$
3. Perform Edge Dropout and Node Dropout
4. While $k \leq l$:
5. Update Node Embedding
6. Update Edge Embedding
7. $k \longleftarrow k + 1$
8. Perform Graph Readout to obtain graph embedding
9. Score target triplet using triplet and graph embeddings
10. Update the GNN and Dropout Module according to ℓ

from the target head to the target tail entity. We follow [14–16] to set the h to 3. Specifically, for each target triplet, we extract the h-hop neighbors of the target head and target tail. Afterwards, we find the common entities of h-hop neighbors of the target head and target tail. Finally we add the edges (triplets) whose head and tail both belong to the common entities or target entities to construct the subgraph.

3.3. Node/edge embedding initialization

In the inductive setting, we need to define an entity-free initial embedding for each entity (node) in the subgraph. Here we use the relative positional embedding to initialize the node embedding. As adopted in GraIL [15] and CoMPILE [14], the common approach is to initialize the node embedding by the distance to the target head and target tail to capture the relative position of each node in the subgraph. The node embedding for node i is defined as $\bar{\boldsymbol{N}}_i = \text{one-hot}(d_{si}) \oplus \text{one-hot}(d_{it}) \in \mathbb{R}^{2(h+1)}$ where d_{si} denotes the minimum distance from the target head to node i, d_{it} denotes the minimum distance from node i to target tail, and $\oplus$ represents the feature concatenation operation. For edge i, i.e. (s_i, r_i, t_i), the initialized edge embedding is defined as $\bar{\boldsymbol{E}}_i = \bar{\boldsymbol{N}}_{s_i} \oplus \boldsymbol{R}_{r_i} \oplus \bar{\boldsymbol{N}}_{t_i} \in \mathbb{R}^{4(h+1)+d}$. Then $\bar{\boldsymbol{N}}$ and $\bar{\boldsymbol{E}}$ are mapped to the same dimensionality d as that of the relation embedding using fully-connected layers, after which we obtain the mapped node embedding $\boldsymbol{N} \in \mathbb{R}^{N_n \times d}$ and edge embedding $\boldsymbol{E} \in \mathbb{R}^{N_e \times d}$, respectively.

3.4. Explainable subgraph neural network

Given that the enclosing subgraph is extracted, the next procedure is to score the subgraph to identify the plausibility of the target triplet. In this section, we introduce how we get the pruned subgraph and use the pruned subgraph for relation reasoning. From a global perspective, the message passing between nodes and edges happens after the dropout for nodes and edges. The dropout procedure only conducts once for the sake of model complexity. We first introduce the proposed dropout operation in detail and then illustrate the message passing mechanism used in our algorithm.

The noisy nodes and edges in the subgraph negatively affect the learning of logical relation rules and hinder higher performance of GNN models. Existing studies try to identify edge importance with attention mechanisms [14,15], which can highlight useful edges. However, these methods cannot explicitly filter out noisy information and only tend to assign higher weights to informative edges. Moreover, the negative impact of noisy nodes and edges can severely affect the explainability of the prediction from subgraph, because we cannot identify which component the model focuses on when generating the prediction.

In contrast to previous subgraph-based methods [14–17], we propose a node and edge dropout module to selectively remove

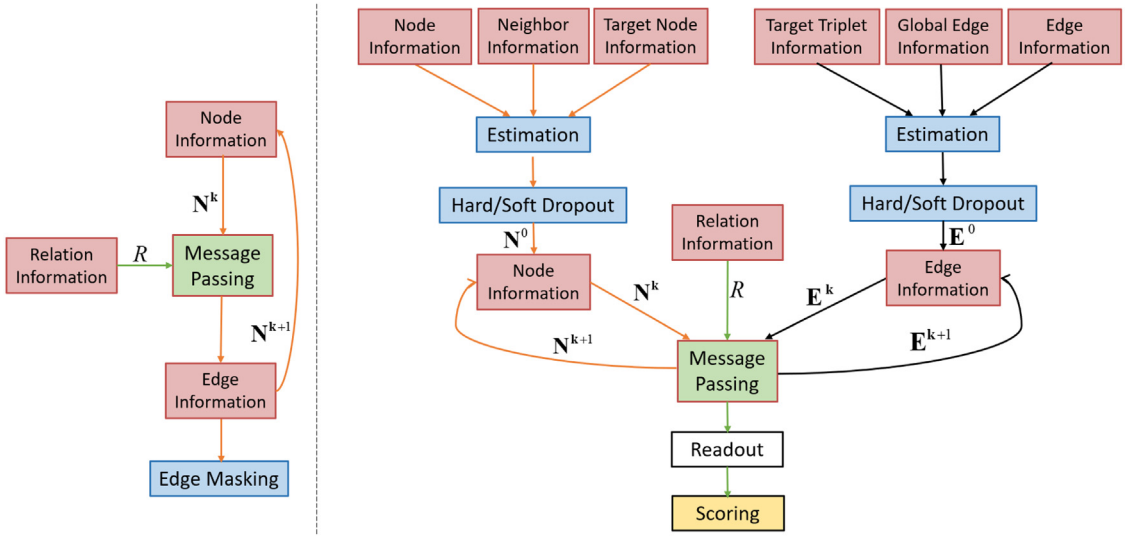


Fig. 3. Comparison between Other Graph Explanation Model and GraphDrop. The left figure illustrates the pipeline of GraphMask [19], a state-of-the-art graph explanation model that shares similarity with our model (the presence of relation information depends on the task). The right figure presents the pipeline of GraphDrop. Since GraphMask masks edges based on a trained GNN, it does not involve the scoring procedure. Moreover, GraphDrop introduces additional constraints on the topology information of the pruned subgraph, which are not included in GraphMask.

noisy information, in which way the negative impact of noisy information can be minimized to obtain better performance. Moreover, by pruning nodes and edges to discover the key components, our algorithm enables the prediction from subgraph to be explainable. Our algorithm is also different from popular graph explanation models [19,21,23]. Firstly, previous GNN explanation methods tend to explain a well-trained GNN and focus on the explainability of trained GNN (i.e., *post-hoc interpretability*). In contrast, we dynamically remove noisy edges and nodes in the subgraph during the training of GNN to improve both relational prediction capability and explainability. Secondly, different from many GNN explanation methods that mask either nodes or edges [19,21,23], we identify the contribution of both the nodes and edges to the prediction. Thirdly, the design of the dropout module, including the estimation function and soft dropout function, is different from other methods (see more details in Fig. 3). Lastly, we design topological loss to retain the connectivity and paths between target entities in the pruned subgraph. In the following section, we introduce the procedure to remove the nodes and edges in detail.

3.4.1. Edge dropout

To filter out noisy edges in the subgraph, we first propose an **estimation function** in the edge dropout module. Instead of merely using the edge embedding to determine whether the edge can be dropped as in previous studies [19,46], the estimation function incorporates the information from the subgraph and target triplet, such that the model can have a global view on the subgraph and the target to comprehensively estimate whether the edge should be removed. Mathematically, for each edge i, we first take the embeddings of each edge, the global edge embedding, and the target triplet as inputs, and then calculate the estimated embedding of each specific edge. The procedure can be formulated as:

$$\begin{aligned} \boldsymbol{E}_g &= \frac{1}{N_e}\sum_{j=1}^{N_e}\boldsymbol{E}_j \\ \boldsymbol{E}_i^e &= \text{Linear}(\boldsymbol{E}_i \oplus \boldsymbol{E}_T \oplus \boldsymbol{E}_g; \theta_L) \\ \boldsymbol{E}_i^{drop} &= \text{Tanh}(\boldsymbol{E}_i^e - \boldsymbol{E}_i) \end{aligned} \tag{1}$$

where $\boldsymbol{E}_T$ and $\boldsymbol{E}_g$ denote the embedding of the target triplet and the global edge embedding respectively, $\boldsymbol{E}_i^e \in \mathbb{R}^d$ represents the embedding that incorporates the information of target triplet, global edge embedding and estimated edge, and $\boldsymbol{E}_i^{drop}$ is the estimated embedding of edge i that is used to determine the dropout value of edge i. By using the information of target triplet to determine the noisy level of each edge, we can estimate the relevance of edge i to the target triplet. By utilizing the global edge embedding, the model can have a global view on the subgraph and then determine which edge is informative.

After the estimated embedding of each edge is obtained, we remove noisy edges based on estimated embedding by the following formula:

$$s^i, \ell_e^p = \text{Dropout}(\boldsymbol{E}_i^{drop}; \theta_{dr}) \tag{2}$$

where Dropout is the dropout function parameterized by θ_{dr}, which determines whether to filter out edge i based on its noisy level, s^i is the dropout value of edge i, and ℓ_e^p is the penalty loss to encourage model to filter out as many redundant edges as possible. The dropout function is trained across all subgraphs to obtain a strong inductive power to identify the noisy level of each edge. We put forward two candidates for the Dropout function, i.e., **soft dropout** and **hard dropout** in the following sections. After obtaining the output $\boldsymbol{s}^i$ from the dropout function, the updated embedding of the edge i can be determined:

$$\boldsymbol{E}_i^0 \longleftarrow s^i \cdot \boldsymbol{E}_i \tag{3}$$

where $\boldsymbol{E}_i^0$ represents the updated embedding of edge i, which contains much less noisy information. When s^i is equal to 0, the edge i is explicitly removed. The obtained $\boldsymbol{E}_i^0$ is then leveraged to conduct the message passing in the GNN.

With the proposed dropout module, our algorithm is capable of identifying and filtering out noisy nodes and edges. In this way, the GNN can dynamically retain informative nodes and edges in the subgraph and filter out redundant ones to obtain a better inference ability and enable the prediction to be more explainable. We illustrate the pipelines of hard dropout and soft dropout in detail in the following sections.

Hard Dropout: The main characteristic of hard dropout is that it produces a binary discrete dropout value s^i for each edge/node. Therefore, optimizing the model parameters with

traditional gradient-based methods will not be feasible due to the discrete nature of s^i. To tackle this issue, we utilize the hard concrete distribution [19,47–49] to generate s^i and enable the gradient-based update of the parameters. Specifically, hard concrete distribution is a mixed discrete–continuous distribution on the interval [0, 1]. It assigns a continuous probability to 0 or 1, and meanwhile it allows continuous outcomes in the unit interval such that the gradient can be computed via the reparameterization trick. The computation of s^i for hard dropout is illustrated as follows:

$$z^i = \text{ReLU}(\boldsymbol{E}_i^{drop}\ \boldsymbol{W}_{hard} + b_{hard}) + b_h \tag{4}$$

$$\hat{s}^i = \sigma(\frac{\log\frac{u}{1-u} + z^i}{\tau}) \tag{5}$$

$$\bar{s}^i = (\zeta - \omega)\cdot\hat{s}^i + \omega \tag{6}$$

$$s^i = \begin{cases} 1, & \bar{s}^i > 0.5 \\ 0, & \text{others} \end{cases} \tag{7}$$

$$l_e^p = \sum_{i=1}^{N_e}\sigma(z^i - \tau\cdot\log(-\frac{\omega}{\zeta})) \tag{8}$$

where b_h is the bias term for hard concrete distribution, $\boldsymbol{W}_{hard} \in \mathbb{R}^{d\times 1}$ is the parameter matrix for hard dropout, σ is the Sigmoid activation function, τ is the temperature hyperparameter that controls the sparsity of the edges, ζ and ω are the hyperparameters to stretch $\hat{s}^i$ such that it is closer to Bernoulli distribution ($\zeta > 1$ and $\omega < 0$), and $u \sim \mathcal{U}(0, 1)$ ($\mathcal{U}$ denotes Gaussian distribution). Notably, $\log\frac{u}{1-u}$ and τ are dismissed during testing [47]. l_e^p is the penalty loss that encourages the model to remove as many noisy edges as possible. Via the operations in Eqs. (4)–(8), $\bar{s}^i$ is close to Bernoulli distribution. When computing the gradients, the randomness is transferred to the Gaussian variable u via the reparameterization trick, and the gradient of z^i can be explicitly computed. For more details about the hard concrete distribution, please refer to [47]. Compared to using other methods such as reinforcement learning [50] to obtain the exact binary weight, using the hard concrete distribution is more elegant, without the need of adding additional optimization objectives or components. Via the hard dropout, the model can remove noisy edges, which cannot be realized by common attention mechanisms [14,15].

Soft Dropout: The binary dropout value of hard dropout is desirable for explainability. However, some edges may be difficult to be defined precisely as useful or useless. Under such circumstance, we always need to obtain a more feasible dropout value for the ambiguous edges such that the performance and explainability of the model can be balanced. Therefore, in this section, soft dropout mechanism is utilized to get the continuous output value for each edge. The procedure for the soft dropout is shown below:

$$\boldsymbol{z}^i = \text{ReLU}(\boldsymbol{E}_i^{drop}\ \boldsymbol{W}_{soft} + \boldsymbol{b}_{soft}) \tag{9}$$

$$z_k^i \longleftarrow \frac{e^{\lambda\cdot z_k^i}}{\sum_j e^{\lambda\cdot z_j^i}} \tag{10}$$

$$s^i \longleftarrow z_1^i,\quad \boldsymbol{z}^i = [z_1^i, z_2^i] \tag{11}$$

$$l_e^p = \sum_{i=1}^{N_e}(1 - (z_1^i - z_2^i)^2 + \beta_2\cdot z_1^i) \tag{12}$$

where $\boldsymbol{W}_{soft} \in \mathbb{R}^{d\times 2}$ is the parameter matrix for soft dropout, and $\boldsymbol{z}^i \in \mathbb{R}^2$ is the dropout vector used to determine whether the edge should be dropped. l_e^p is the penalty loss that encourages the elements of $\boldsymbol{z}^i$ to be close to 0 or 1, and β_2 is the coefficient that controls the sparsity of the edge. In Eq. (10), we normalize the dropout vector where λ is the scale factor to widen the distance between the elements in $\boldsymbol{z}^i$ [49]. In fact, we have $\frac{z_1^i}{z_2^i} = e^{\lambda\cdot(z_1^{i'} - z_2^{i'})} = (e^{(z_1^{i'} - z_2^{i'})})^\lambda$, which suggests that the distance increases exponentially with λ. In the visualization experiment, we show that this simple technique is practical and the elements of $\boldsymbol{z}^i$ are almost binary.

It is worth mentioning that soft dropout fundamentally differs from the commonly-considered attention mechanism in previous studies [14,15]. Firstly, soft dropout introduces the scale factor λ and the penalty loss l_e^p to reach better dropout effect by encouraging the distance among elements $\boldsymbol{z}^i$ to be larger so that the elements are almost binary. Moreover, it encourages the dropout module to filter out as many edges as possible via penalty loss. Lastly, soft dropout merely modifies the edge embedding and can be integrated with any message passing mechanisms. Compared to hard dropout, soft dropout is easier to be trained, which rarely encounters the 'collapse' problem (i.e., dropout values of all the edges are the same). In contrast, the hyperparameters of hard dropout need to be carefully designed to avoid 'collapse' problem. Moreover, because soft dropout allows continuous fine-grained output that is more expressive, its performance is expected to be better.

3.4.2. Node dropout

The dropout procedure of nodes is similar to that of edge, while distinction remains. Firstly, in the **estimation function**, we use the embedding of node i, the embedding of target entities, and the neighbor aggregation embedding to determine the noisy level of each node. The procedures are illustrated as follows:

$$\begin{aligned} \boldsymbol{N}_i^{nei} &= (\boldsymbol{A}\boldsymbol{N})_i \\ \boldsymbol{N}_i^{e} &= \text{FC}(\boldsymbol{N}_i \oplus (\boldsymbol{N}_{t_T} - \boldsymbol{N}_{s_T}) \oplus \boldsymbol{N}_i^{nei}; \theta_{FC}) \\ \boldsymbol{N}_i^{drop} &= \text{Tanh}(\boldsymbol{N}_i^{e} - \boldsymbol{N}_i) \end{aligned} \tag{13}$$

where FC denotes the fully-connected layer, $\boldsymbol{A}$ is the node-to-node adjacency matrix ($\boldsymbol{A}_{ij} = 1 \iff \exists\ r$, triplet (j, r, i) exists), $\boldsymbol{N}_i^{nei}$ is the neighbor aggregation information for node i. $\boldsymbol{N}_i^{drop}$ is the estimated node embedding which incorporates the information of node i, neighboring nodes and target entities, and then uses these information to comprehensively estimate the importance of node i. By leveraging the neighbor information, the model can to some extent estimate the relative position of the node in the graph. By leveraging the information of target entities, the estimation function is target-oriented. Note that the parameters of node and edge estimation function are not shared.

After the estimation function, we filter out the nodes via the soft/hard dropout function based on the output of the estimation function. The dropout module shares the same structure as that of edges. However, the parameters of the dropout function are not shared, and the input to the dropout function becomes $\boldsymbol{N}_i^{drop}$. After the dropout function, we obtain a new node embedding $\boldsymbol{N}_i^0$ and the node penalty loss ℓ_n^p.

3.4.3. Topological loss

One of the main novelties of our model is the design of topological loss to regularize the learning of the dropout functions. Subgraph has complex topology structures, and thereby removing a node might cause a chain reaction especially when the node has a large degree (i.e., if a node is removed, the logical connection between the target entities might be lost unexpectedly). Moreover, the pruned subgraph might consist of several subcomponents that are not connected. Therefore, to avoid such

chain reaction and retain the connectivity of the pruned subgraph, we put forward topological loss to regularize the learning of the node dropout.

Firstly, the target entities should not be removed so that the relation inference can be conducted. Moreover, inspired by the graph information bottleneck [46], we hope that if one node is retained, its neighbors should also have a higher possibility to be retained such that the generated subgraph is connective instead of consisting of several disconnected sub-components. To this end, we regularize the learning of the dropout module to encourage the learned subgraph to be connective and target-oriented. The equation for the regularization is shown as below:

$$\ell_{topo}^{nei} = \beta \cdot [(\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{H} - \boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S}) + (1 - s^{s_T}) + (1 - s^{t_T})] \tag{14}$$

where s^{s_T} and s^{t_T} are the dropout values for target head and target tail respectively, $\boldsymbol{H} \in \mathbb{R}^{N_n \times 1}$ is the ones matrix whose elements are all equal to one, $\boldsymbol{S} \in \mathbb{R}^{N_n \times 1}$ is the dropout matrix for nodes ($\boldsymbol{S} = [s^1; s^2; ...; s^{N_n}]$), β is a scalar which determines the weight of the topological loss. To be more specific, $\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{H}$ sums over the number of neighbors of the retained nodes in original subgraph, and $\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S}$ sums over the number of neighbors of the retained nodes in the pruned subgraph. Our algorithm forces $\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{H} - \boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S}$ to approximate 0 to encourage the neighbors of the retained nodes are also retained in the pruned subgraph. By this means, ℓ_{topo}^{nei} adds a restriction on the local topological structure of the subgraph and encourages the generated subgraph to be connective. One may argue that the learning of $\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{H} - \boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S}$ might fall into a sub-optimal solution where all nodes are filtered ($\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{H} - \boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S}$ is equal to 0 under such circumstance). Nevertheless, by adding $(1 - s^{s_T}) + (1 - s^{t_T})$ into ℓ_{topo}^{nei}, we force the target entities to be retained and the sub-optimal solution is not likely to happen. And due to the penalty loss, the algorithm will prevent the dropout function from selecting all the nodes, which is also a sub-optimal solution of $\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{H} - \boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S}$.

However, ℓ_{topo}^{nei} merely reflects the 1-hop structure information of the generated subgraph, while relation prediction relies on multi-hop relational paths between target entities. To this end, we further design a constraint, referred to as the connective topological loss ℓ_{topo}^{con}, to encourage the pruned subgraph to still contain the paths that connect the target head and target tail such that the relation can be inferred according to relational paths. Specifically, after node dropout, the updated node-to-node adjacency matrix can be represented as: $\widetilde{\boldsymbol{A}} = \boldsymbol{S}^T \odot (\boldsymbol{A} + \boldsymbol{A}^T) \odot \boldsymbol{S}$ where we define $\odot$ as the expanded element-wise multiplication of matrices. For example, for the given matrices $\boldsymbol{A} \in \mathbb{R}^{N_n \times N_n}$ and $\boldsymbol{S} \in \mathbb{R}^{N_n \times 1}$, the expanded element-wise multiplication of these two matrices is denoted as: $\forall j \in [1, N_n]$, we have $(\boldsymbol{A} \odot \boldsymbol{S})_{i,j} = \boldsymbol{A}_{i,j} \cdot \boldsymbol{S}_{i,1}$ where the second dimension of $\boldsymbol{S}$ can be seen as expanded. The equation for ℓ_{topo}^{con} is shown as below:

$$\ell_{topo}^{con} = \beta \cdot \frac{b}{\sum_{i=1}^{h+1} \widetilde{\boldsymbol{A}}^i_{s_T,t_T} + \gamma} \tag{15}$$

where h is the hop number of the subgraph, b is a constant scalar which is set to 32 in our experiment to scale ℓ_{topo}^{con}, γ is a scalar to prevent division by 0 which is set to 1, $\widetilde{\boldsymbol{A}}^1 = \boldsymbol{S}^T \odot (\boldsymbol{A} + \boldsymbol{A}^T) \odot \boldsymbol{S}$, $\widetilde{\boldsymbol{A}}^2 = \widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{A}}^T$, and $\widetilde{\boldsymbol{A}}^i = \widetilde{\boldsymbol{A}}^{i-1}\widetilde{\boldsymbol{A}}^T$. Intuitively, if the target head and target tail are still connected by a i-hop path after node dropout, $\widetilde{\boldsymbol{A}}^i_{s_T,t_T}$ is larger than 0 according to the definition of the adjacency matrix. Notably, the algorithm will preferentially reserve the nodes with more paths crossing (which are assumed as important), for the reason that preserving these nodes decreases the loss more. By minimizing ℓ_{topo}^{con} via the gradient descent, we encourage the model to retain the paths connecting two target entities, such that logical rules can be inferred. The overall topological loss is the sum of ℓ_{topo}^{con} and ℓ_{topo}^{nei}:

$$\ell_{topo} = \ell_{topo}^{con} + \ell_{topo}^{nei} \tag{16}$$

Recall that we also design a penalty loss for nodes which forces the generated subgraph to have as few nodes as possible. The penalty loss and topological loss compete with each other, and together they seek to generate a subgraph that is minimal and sufficient enough to conduct relation reasoning.

3.4.4. GNN for message passing

Generally, we use the node-edge communicative updating mechanism to conduct message passing in the subgraph as in [14]. Note that the edge attention mechanism is not used in our GNN since we already have the dropout function.

Node Embedding Updating: The node embedding is updated for totally l iterations. We use the edge embedding to update node representations:

$$\boldsymbol{N}^k_{agg} = \boldsymbol{A}^{\boldsymbol{te}}\boldsymbol{E}^{k-1} \tag{17}$$

$$\boldsymbol{N}^k = \mathrm{ReLU}((\boldsymbol{N}^k_{agg} + \boldsymbol{N}^{k-1})\boldsymbol{W}^k_n) \tag{18}$$

where $\boldsymbol{E}^{k-1}$ and $\boldsymbol{N}^k$ are the edge embedding at iteration $k-1$ and the node embedding at iteration k respectively, $\boldsymbol{N}^k_{agg}$ denotes the node aggregation information, and $\boldsymbol{A}^{\boldsymbol{te}} \in \mathbb{R}^{N_n \times N_e}$ is the adjacency matrix that connects the tail entity and the corresponding edge. Eq. (17) aggregates the edge information to each node, and only the edges whose tail is the corresponding node are selected for aggregation.

Edge Embedding Updating: To obtain the edge aggregation information and update the edge embedding, we have the following equations:

$$\boldsymbol{E}^k_{agg} = (\boldsymbol{A}^{\boldsymbol{he}})^T\boldsymbol{N}^k + (\boldsymbol{A}^{\boldsymbol{re}})^T\boldsymbol{R} - (\boldsymbol{A}^{\boldsymbol{te}})^T\boldsymbol{N}^k \tag{19}$$

$$\boldsymbol{E}^{k'} = \mathrm{ReLU}(\boldsymbol{E}^{k-1} + \mathrm{Tanh}(\boldsymbol{E}^k_{agg})) \tag{20}$$

$$\boldsymbol{E}^k = \mathrm{ReLU}(\boldsymbol{E}^{k'}\boldsymbol{W}^k_e + \boldsymbol{E}^0) \tag{21}$$

where $\boldsymbol{A}^{\boldsymbol{he}} \in \mathbb{R}^{N_n \times N_e}$, $\boldsymbol{A}^{\boldsymbol{re}} \in \mathbb{R}^{N_r \times N_e}$ is the adjacency matrix that connects head entity/relation and the corresponding edge, respectively. Note that we add the $\boldsymbol{E}^0$ to update the edge embedding in Eq. (21) to perform residual learning [51]. Recall that $\boldsymbol{E}^0$ is the dropped edge embedding and $\boldsymbol{N}^0$ is the dropped node embedding.

Graph Readout: To generate the final graph representation and then predict the score of the target triplet, we use a multi-layer perception network followed by a Gated Recurrent Unit (GRU) [52] to summarize the node embedding, as shown in the following equations:

$$\boldsymbol{N}^{e'}_i = \mathrm{DNN}_{read}(\boldsymbol{N}^l_{s_T} \oplus \boldsymbol{N}^l_{t_T} \oplus \boldsymbol{N}^l_i;\ \theta_{DNN_{read}}) \tag{22}$$

$$\boldsymbol{N}^e = \mathrm{GRU}(\boldsymbol{N}^{e'};\ \theta_{GRU}) \tag{23}$$

$$\boldsymbol{G}^e = \frac{1}{N_n}\sum_{i=1}^{N_n} \boldsymbol{N}^e_i \tag{24}$$

where $\boldsymbol{G}^e$ is the graph embedding that captures the subgraph information from a global perspective, and DNN_{read} is the deep neural network that communicates the target entity embeddings with all the node embeddings. This graph readout function is different from previous studies [14,15].

Scoring Function Definition: We adopt the idea of TransE [6] and CoMPILE [14] to design the scoring function. While differently, we apply the graph embedding in the design of the scoring

function to further capture the global information of the subgraph, and use the global subgraph information to help infer the relation pattern between the target triplet. All in all, the scoring function is defined as:

$$S = DNN((\boldsymbol{N}^l_{s_T} + \boldsymbol{R}_{r_T} - \boldsymbol{N}^l_{t_T}) \oplus \boldsymbol{G}^e; \theta_{DNN}) \tag{25}$$

where $\boldsymbol{N}^l_{s_T}$, $\boldsymbol{R}_{r_T}$, and $\boldsymbol{N}^l_{t_T}$ denote the final representation of target head, target relation, and target tail, respectively. We use a two-layer fully-connected network to process target triplet embedding and infer the score of target triplet.

Training Scheme: We train the overall framework in an end-to-end manner rather than adopt a two-stage training strategy where GNN is first trained and then the dropout module is trained [19,21]. The overall loss is the composition of the predictive loss, penalty loss and topological loss:

$$\ell_{pre} = max\{S' - S + \eta, 0\} \tag{26}$$

$$\ell = \ell_{pre} + \alpha(\ell^p_n + \ell^p_e + \ell_{topo}) \tag{27}$$

where ℓ_{pre} is the predictive loss, ℓ^p_n and ℓ^p_e and the penalty loss for node and edge respectively, ℓ_{topo} is the topological loss, and α is a scalar that determines the importance of the restriction losses (penalty loss and topological loss). Note that ℓ_{pre} is formatted as the hinge-loss which aims to widen the distance between scores of the positive and negative triplets (S' denotes the score of the negative triplet and η is a positive margin). Via penalty loss, the algorithm seeks to generate a minimal subgraph, and via the predictive loss and topological loss, the model seeks to generate a sufficient subgraph. Coupling together, the algorithm seeks to find a subgraph that is free of redundancy and simultaneously expressive enough for prediction.

4. Experiments

We evaluate the proposed GraphDrop on the widely-used inductive datasets for relation prediction. In our experiments, we mainly show that: (1) the proposed GraphDrop outperforms state-of-the-art methods on the commonly-used inductive datasets; (2) the proposed penalty and topological losses are helpful to the overall improvement of the algorithm; (3) the proposed dropout functions effectively filter out noisy nodes and edges and improve the performance of GNN; (4) the proposed dropout functions outperform other dropout baselines; (5) the proposed algorithm can generate human-understandable explanations for subgraph prediction; (6) the proposed method assigns a low score to the majority of nodes and edges in the subgraph and the dropout values of soft dropout approximate Bernoulli distribution.

4.1. Dataset

FB15K-237 [53] and NELL-995 [4] are widely-used databases for relation reasoning. We follow Meta-iKG [16] to extract four versions of inductive datasets for each database to ensure a more accurate evaluation of the models. In inductive datasets, we have filtered out the triplets that have no enclosing subgraph between the target entities as in [14,16]. The statistics of the four versions of inductive FB15k-237 and Nell-995 datasets are shown in Tables 2 and 3, respectively [16]. Each inductive dataset consists of train graph and test graph where the entities in the train graph and the test graph are not overlapped, and the relations are shared across the train and test graphs. Following the setting of [14–16], the subgraphs of training and validation triplets are extracted from the train graph, and the subgraphs of test triplets are extracted from the test graph. The training triplets exist in the train graph, while the test graph does not contain test triplets (test graph contains the entities and relations of test triplets).

4.2. Experimental details

To be consistent with previous algorithms [14–16], we use AUC-PR and Hits@10 to evaluate the models. To compute AUC-PR, we sample one negative triplet for each test triplet. For Hits@10, we rank each test triplet among the sampled 49 negative head/tail triplets and evaluate whether the score of the test triplet ranks top 10. The negative triplets are obtained by replacing the head or tail of each test triplet with other entities. We train the model for four times and average the testing results to obtain the final results.

We implement our algorithm on PyTorch 1.4.0, and use Adam [54] as optimizer with learning rate being 0.001. The hop number h is set to 3 which is consistent with previous studies [14–16]. The number of iterations l is set to 3. The α and β are both set to 0.01. The feature dimensionality d is set to 32.

Baselines: We evaluate our model with the following baselines: GraIL [15], CoMPILE [14], Relational GAT [55], Meta-iKG [16], and RuleN [11]. RuleN [11] is a rule-based method and the other baselines are subgraph-based methods. For the detailed introduction of these methods, please refer to the introduction and related work sections. Notably, Relational GAT refers to the implemented Graph Attention Network (GAT) [55] for inductive relation reasoning. Since the task in this paper involves relation embedding, we modify the procedures of the regular GAT as:

$$\boldsymbol{N}^k \longleftarrow \boldsymbol{W}^k \boldsymbol{N}^k \tag{28}$$

$$\alpha_{ij} = \frac{\exp(\sigma(\boldsymbol{a}^k[\boldsymbol{N}^k_i \oplus \boldsymbol{R}_{i,j} \oplus \boldsymbol{N}^k_j]))}{\sum_{g\in\mathcal{N}_i}\exp(\sigma(\boldsymbol{a}^k[\boldsymbol{N}^k_i \oplus \boldsymbol{R}_{i,g} \oplus \boldsymbol{N}^k_g]))} \tag{29}$$

$$\boldsymbol{N}^{k+1}_i = \mathrm{Tanh}(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\boldsymbol{N}^k_j + \boldsymbol{N}^k_i) \tag{30}$$

where $\boldsymbol{a}^k \in \mathbb{R}^{1\times 3\cdot d}$ is the attention parameter for iteration k, $\boldsymbol{R}_{i,j}$ denotes the embedding of the relation connecting entities i and j, and σ is a nonlinearity function which is chosen as LeakyReLU following [55]. The scoring function of Relational GAT is defined as:

$$\boldsymbol{N} \longleftarrow \oplus^l_{k=1}\boldsymbol{N}^k \tag{31}$$

$$S = DNN(\boldsymbol{N}_{s_T} \oplus \boldsymbol{R}_{r_T} \oplus \boldsymbol{N}_{t_T} \oplus \boldsymbol{G}^e; \theta_{DNN}) \tag{32}$$

where $\boldsymbol{G}^e$ is the graph embedding (see Eq. (24)). In Eq. (31), the node embeddings from all iterations are concatenated to generate the final node embedding.

4.3. Results and discussions

4.3.1. The performance on inductive datasets

The Comparison with baselines. In this section, we compare the proposed GraphDrop with baselines on eight inductive datasets. As presented in Tables 4 and 5, GraphDrop (both the soft dropout and hard dropout versions) demonstrates consistent improvement on the majority of the inductive datasets in terms of both the AUC-PR and Hits@10 metrics. Specifically, for almost all the datasets, the improvement on the Hits@10 is significant. It is worth mentioning that our GraphDrop outperforms the state-of-the-art method Meta-iKG by a considerable margin. Notably, both versions of GraphDrop outperform Relational GAT [55] on all datasets, which demonstrates the superiority of the proposed method over the attention-based method. We argue that this is because Relational GAT merely assigns lower weights to uninformative edges and higher weights to important edges, while GraphDrop can directly remove noisy information to the utmost extent by maximally filtering out noisy edges and

Table 2
Statistics of the inductive FB15k-237 datasets.

Version	Train relations	Train graph	Training triplets	Validation triplets	Test relations	Test graph	Test triplets
v1	183	4,245	4,040	475	146	1,993	108
v2	213	9,739	9,462	1,142	176	4,145	380
v3	218	17,986	17,703	2,179	187	7,406	779
v4	222	27,203	26,917	1,658	204	11,714	1,369

Table 3
Statistics of the inductive Nell-995 datasets.

Version	Train relations	Train graph	Training triplets	Validation triplets	Test relations	Test graph	Test triplets
v1	14	4,687	3,610	379	14	833	81
v2	88	8,219	7,118	921	79	4,586	430
v3	142	16,393	14,453	1,848	122	8,048	686
v4	77	7,546	6,710	419	61	7,073	638

Table 4
Comparison between models (Hits@10). The best results are highlighted in bold, and the second best results are marked with underlines.

Model	FB15k-237				NELL-995			
	v1	v2	v3	v4	v1	v2	v3	v4
RuleN [11]	65.35	71.68	67.84	70.53	53.70	69.77	64.29	57.92
Relational GAT [55]	64.78	71.58	68.29	67.64	58.64	73.72	75.43	74.11
GraIL [15]	66.52	73.82	70.15	68.30	55.56	76.40	75.66	71.24
CoMPILE [14]	66.52	72.37	69.77	70.27	62.35	76.51	75.58	68.19
Meta-iKG (MAML) [16]	66.52	72.37	68.81	74.32	60.49	74.07	77.99	71.63
Meta-iKG (Meta-SGD) [16]	66.96	74.08	71.89	72.28	64.20	77.91	77.41	73.12
GraphDrop (hard dropout)	**70.43**	75.13	75.67	72.75	66.05	77.91	80.32	74.61
GraphDrop (soft dropout)	68.70	**76.97**	**76.25**	**76.77**	**67.28**	**81.86**	**82.73**	**77.12**

Table 5
Comparison between models (AUC-PR).

Model	FB15k-237				NELL-995			
	v1	v2	v3	v4	v1	v2	v3	v4
RuleN [11]	79.60	82.67	83.03	84.01	67.12	80.52	73.91	77.07
Relational GAT [55]	81.57	84.03	81.48	81.62	68.95	80.78	81.89	81.64
GraIL [15]	80.45	83.66	84.35	83.08	69.35	85.04	84.43	80.19
CoMPILE [14]	79.95	83.56	83.97	83.87	68.36	85.50	84.04	79.89
Meta-iKG (MAML) [16]	80.31	82.95	82.52	84.23	72.12	84.11	82.47	79.25
Meta-iKG (Meta-SGD) [16]	81.10	84.26	84.57	83.70	72.50	85.97	84.05	81.24
GraphDrop (hard dropout)	**83.79**	85.25	84.40	85.69	74.91	86.03	84.62	81.97
GraphDrop (soft dropout)	83.37	**86.37**	**85.06**	**86.45**	**79.56**	**86.89**	**85.04**	**83.59**

nodes. Moreover, we seek to preserve the topological structure of the subgraph when pruning the subgraph, and develop estimation function to comprehensively evaluate the importance of the nodes/edges, which are not considered in Relational GAT. Overall, the results show that the proposed method achieves marked improvement over other algorithms, which highlights the necessity of removing noisy nodes and edges in the enclosing subgraph.

The comparison between soft dropout and hard dropout. As we can infer from the results in Tables 4 and 5, the soft dropout version of the GraphDrop outperforms its hard dropout version on the majority of datasets. We argue that this is because the soft dropout version of GraphDrop is more flexible. Although soft dropout encourages dropout values to be approximately binary, it allows continuous dropout values for nodes and edges, which are more fine-grained and representative. In addition, even though the binary dropout values generated by the hard dropout are more interpretable, the hard dropout suffers from limited expressive power. In a sense, the soft dropout version of GraphDrop can be seen as the trade-off between the performance and explainability.

4.3.2. Complexity analysis

The main innovation of GraphDrop is to insert the dropout module into subgraph-based models to improve predictive performance and interpretability. The dropout module mainly consists of several DNNs to perform estimation and dropout. To reduce the model complexity, the trainable layers in DNNs only include one or two fully-connected layers. Moreover, the attention mechanism is removed from our model, such that the complexity of the model is reduced. Therefore, the number of parameters only increases slightly. Quantitatively, as shown in Table 6, the number of parameters for the hard and soft dropout version of GraphDrop are 37795 and 36805, respectively. For GraIL and CoMPILE, the number of parameters are 28740 and 35777 respectively, which are slightly lower than that of our model. For the Meta-SGD version of Meta-iKG, the number of parameters is considerably higher than that of GraphDrop. Nevertheless, the performance of our model still is better than that of Meta-iKG. All in all, the model complexity slightly increases after the injection of the dropout module.

Table 6
The comparison of model complexity on FB15k-237-v1 dataset.

	The number of parameters
GraIL	**28,740**
CoMPILE	35,777
Meta-iKG (MAML)	35,777
Meta-iKG (Meta-SGD)	71,554
GraphDrop (hard dropout)	37,795
GraphDrop (soft dropout)	36,805

Table 7
Ablation Studies on Inductive Datasets.

	FB15k-237-v1		NELL-995-v1	
	AUC-PR	Hits@10	AUC-PR	Hits@10
W/O Edge Dropout	80.27	67.39	74.47	65.43
W/O Node Dropout	80.36	66.52	74.06	64.20
W/O ℓ_n^p	80.73	66.52	70.66	61.11
W/O ℓ_e^p	79.25	68.26	74.17	67.28
W/O ℓ_{topo}	80.73	64.35	73.51	62.35
GraphDrop (soft dropout)	**83.37**	**68.70**	**79.56**	**67.28**

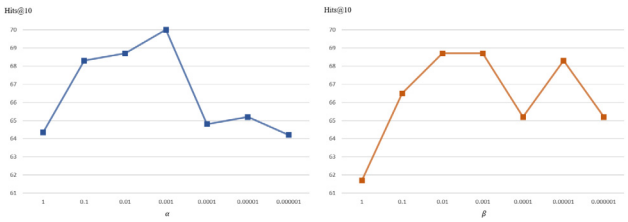


Fig. 4. Performance discrepancy under different hyperparameter settings on FB15k-237-v1 dataset.

4.3.3. Ablation studies

We conduct extensive experiments to evaluate the effectiveness of various components in this section, and the results are shown in Table 7. Note that here we present the results of the soft dropout version of GraphDrop because of its better performance, and similar results are observed on the hard dropout version. Firstly for the dropout module, we remove it from our model to analyze its contribution (see the cases of 'W/O Edge Dropout' and 'W/O Node Dropout' in Table 7). Moreover, we evaluate the contribution of the proposed losses, namely node penalty loss, edge penalty loss, and topological loss (see the cases of 'W/O ℓ_n^p', 'W/O ℓ_e^p', and 'W/O ℓ_{topo}').

From Table 7 we find that GraphDrop significantly outperforms its counterpart whose edge dropout module is removed, demonstrating the effectiveness of the proposed edge dropout. Specifically, edge dropout yields over 1.5% and 3.5% improvement on Hits@10 metric on the FB15k-237-v1 and NELL-995-v1 dataset, respectively. The performance drop on the AUC-PR metric is even more significant. Similar performance decrease is observed when the node dropout is removed. These results suggest that it is of great significance to consider filtering out noisy information that interferes with the relation prediction in the subgraph. Note that here we only remove either edge dropout or node dropout. Please refer to Section 4.3.4 for the results of the full removal of dropout module.

Moreover, from the results in Table 7, we can notice that the performance drops when any of the proposed losses is removed. Specifically, the removal of ℓ_e^p brings smaller impact than the removal of ℓ_n^p, indicting that noisy nodes have more severe influence on the prediction of the model. Moreover, the removal of ℓ_{topo} brings even more negative impact on the performance of the model than that of the removal of node dropout. It indicates that the commonly-considered regularization, i.e., encouraging the model to remove as many nodes as possible, is not necessarily effective from the perspective of performance. Therefore, we should add a restriction on the topological structure of the pruned subgraph such that the relation can be inferred and the performance of the overall system can be improved.

4.3.4. Comparison of the dropout functions

In this section, we evaluate the performance of different dropout functions to demonstrate the superiority of the proposed hard/soft dropout functions. Specifically, we choose two dropout baselines, i.e., random dropout and no dropout for comparison. Moreover, we compare the hard/soft dropout with another version of dropout function that only uses the node/edge embedding to estimate the importance of the corresponding node/edge (see the case of 'Hard Dropout (W/O Estimation)' and 'Soft Dropout (W/O Estimation)').

From the results in Table 8, we can notice that the model with the proposed dropout functions outperforms its counterpart without dropout on both datasets by a significant margin. Notably, the model without dropout (node dropout and edge dropout are both removed) performs considerably worse than its counterpart that either edge or node dropout is removed (see Table 7), indicting that only using node or edge dropout is still beneficial to the performance. As for random dropout, combining the results from both datasets, the model with 10% random dropout performs best, and it slightly outperforms the model without dropout (possibly due to the reason that dropout can reduce overfitting). Nevertheless, our model with hard/soft dropout still outperforms it by over 3% and 7% on the Hits@10 metric in FB15k-237-v1 and NELL-995-v1 dataset respectively. Furthermore, the hard/soft dropout without the proposed estimation function obtains better results and outperforms the random/no dropout, which demonstrates the effectiveness of learning a well-designed estimation function. However, it is still inferior to the model equipped with the proposed estimation function, suggesting that combining the relevant information of the edge/node and global information of the subgraph is beneficial to evaluate the importance of each edge/node. These results demonstrate that each component of the proposed dropout functions is important to the overall improvement of the model.

4.3.5. Hyperparameter evaluation

In this section, we investigate the influence of two hyperparameters in our architecture, namely α and β, and the results are shown in Fig. 4. Recall that α is the hyperparameter to control the weight of the overall restriction losses, and β controls the weight of the topological loss. The default values of α and β are both set to 1etmin2. From Fig. 4 we can infer that α should be a moderate value, and the model reaches the best performance when α is set to 1etmin3. As α becomes smaller, the model performance drops significantly, mainly because the influence of the proposed restriction losses becomes weaker and the optimization degenerates into a regular supervised optimization where only the predictive loss takes effect. In contrast, when α is 1, the performance of the model is also not that impressive because the restriction losses dominate the learning and weaken the influence of predictive loss.

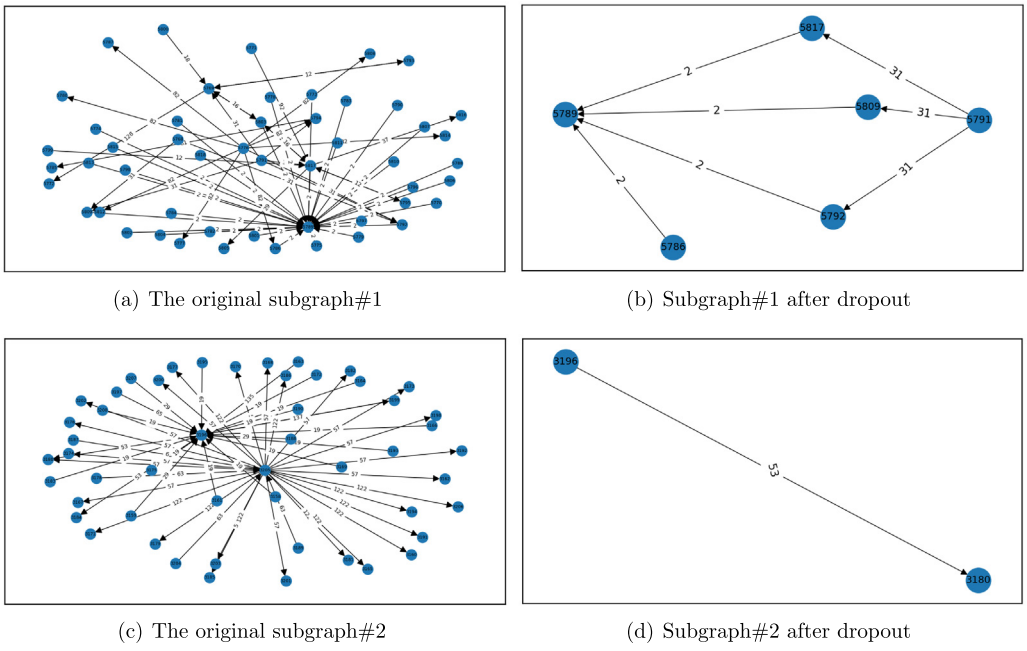
As for β, the model reaches a better performance when β is 1etmin2 or 1etmin3. When β is set to 1, the model performs poorly because the topological loss dominates the learning of the restriction losses. We observe that when β is set to a large value, the optimization of topological loss will fall into a sub-optimal solution where dropout values of the nodes and edges all become 0 or all become 1 (see the analysis in Section 3.4.3). Under such circumstance, the dropout module will have no effect and even negatively influence the prediction of the model, and thus the performance drops significantly. These results reveal how the restriction losses work.

4.3.6. Subgraph provides explanation

To demonstrate that the proposed model can explicitly filter out irrelevant nodes/edges and provide explanations of subgraph reasoning, we perform a case study where visualizations of the original subgraphs and pruned subgraphs are provided. The visualization results are presented in Fig. 5. Note that the pruned

Table 8
The comparison of dropout functions. For random dropout, we employ a regular dropout layer on edge/node embeddings. For the 'W/O Estimation' version of hard/soft dropout, we replace the estimation function (Eq. (1)) with the following operation: $\boldsymbol{E}_i^{drop} = \text{Linear}(\boldsymbol{E}_i;\ \theta_L)$, and the estimation function for node dropout vice verse.

	FB15k-237-v1		NELL-995-v1	
	AUC-PR	Hits@10	AUC-PR	Hits@10
Random dropout (10%)	80.41	65.65	70.73	58.26
Random dropout (50%)	79.76	62.61	73.95	59.88
Random dropout (90%)	79.50	64.78	67.91	58.02
No dropout	81.05	63.91	71.43	58.02
Hard dropout (W/O Estimation)	82.00	64.78	73.44	61.73
Soft dropout (W/O Estimation)	79.91	66.96	73.85	61.73
Hard dropout	**83.79**	**70.43**	74.91	66.05
Soft dropout	83.37	68.70	**79.56**	**67.28**



(a) The original subgraph#1 (b) Subgraph#1 after dropout

(c) The original subgraph#2 (d) Subgraph#2 after dropout

Fig. 5. Visualization of the original subgraphs and the pruned subgraphs. The data are taken from the FB15k-237-v1 dataset.

subgraph is generated by the hard dropout version of GraphDrop, and similar visualization results are observed on the soft dropout version if we do not visualize the edge/node whose dropout value is smaller than 0.5. It can be observed that the proposed method can cut a subgraph that contains a great deal of nodes and edges into a small subgraph, which only contains the target head node, target tail node and a few important paths between them. By this means, we can easily analyze how the model generates the prediction and which component of the subgraph it relies on.

Looking closer into subgraph#1, the target triplet for subgraph#1 is (5791, 31, 5786) where relation#31 is 'music /genre/ artist'. The subgraph contains another relation#2 'people/person/ profession' (note that the entity number does not have actual meaning). It can be inferred from Fig. 5(b) that the profession of entity#5786 is entity#5789. And the entities share the same profession with entity#5786 are all connected to entity#5791 via the relation#31 'music/genre/artist', which indicates the genre of these entities is entity#5791. Therefore, the model infers that the genre of entity#5786 is also entity#5791 as relation#2 and relation#31 are highly related, and thus the model assigns a high score to target triplet (5791, 31, 5786). An even more simple case is subgraph#2. The target triplet for subgraph#2 is (3180, 138, 3196) where relation#138 is 'location /country/ official_language'. The pruned subgraph contains the triplet (3196, 53, 3180) where relation#53 is 'language/human_language/countries_spoken_in'. By looking at the pruned subgraph, the model attains the knowledge that country#3180 speaks language#3196. Therefore, the model infers that it is of high possibility that the official language for country#3180 is language#3196 and thus it assigns a high score to the target triplet. The visualization to some extent suggests that the subgraph prediction from our model is explainable.

4.3.7. Analysis of dropout values

Fig. 6 presents the pie charts for the distributions of dropout value s^i generated by hard and soft dropout functions. For hard dropout, we present the proportion of nodes/edges with dropout value being 0 or 1. For soft dropout, we present the proportion of nodes/edges whose dropout value is within [0, 0.001), [0.001, 0.1), [0.1, 0.9), [0.9, 0.999), and [0.999, 1]. From Fig. 6 we can conclude that the majority of edges and nodes are filtered out in both the hard and soft dropouts, mostly due to the penalty loss l_e^p/l_n^p that encourages the sparsity of the subgraph. It indicates that only a few informative edges/nodes are retained for relation reasoning, which greatly reduces the difficulty of interpreting the prediction.

Moreover, observing Fig. 6, we can find that the dropout values generated by soft dropout are more expressive. In other words, soft dropout can assign a moderate weight between 0 and 1 to the ambiguous edges/nodes such that they will not be explicitly removed, and the contribution of edges/nodes can be highlighted

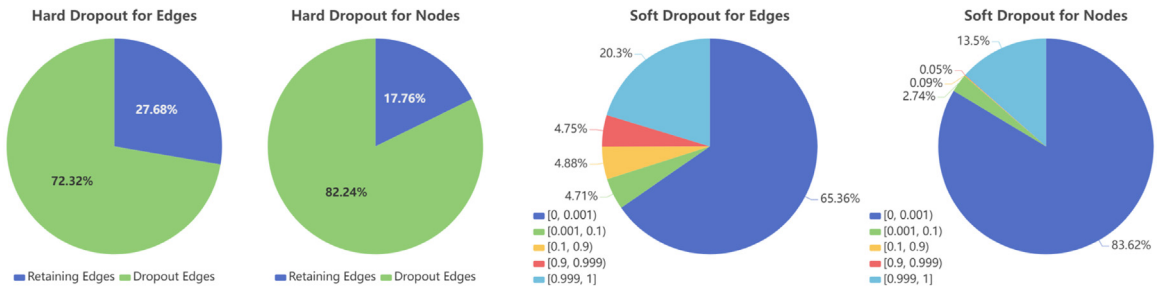


Fig. 6. The pie charts for dropout values of hard and soft dropouts. The subfigures illustrate the distributions of dropout values for edges and nodes in a batch of testing subgraphs from FB15k-237-v1 dataset.

or suppressed differently according to their importance in soft dropout. In addition, for soft dropout, we can observe that the distribution of dropout values of the majority of nodes/edges approximates the Bernoulli distribution. For instance, the proportion of nodes whose dropout value is within [0, 0.001) or [0.999, 1] is over 95%, and the proportion of edges whose dropout value is within [0, 0.1) or [0.9, 1] is over 90%. This is because we introduce the scale factor λ and the penalty loss to encourage s^i to be 0 or 1, which differs from existing attention mechanisms [14,15,55]. It indicates that apart from highlighting important nodes/edges as in attention mechanisms, our soft dropout can reach better dropout effect for noisy nodes/edges. Interestingly, we notice that the model tends to filter out more nodes than edges. This is consistent with the results in Table 7 that reveal filtering out nodes brings more benefit. We speculate that this is because the nodes generally are connected with many edges, and filtering out a node can remove much noisy information as the edges connecting the node are also removed.

5. Conclusion

In this paper, we present a framework for subgraph-based inductive relation reasoning which dynamically prunes the nodes and edges to generate a subgraph that is small but sufficient enough for relation prediction. We propose hard dropout and soft dropout to filter out nodes and edges, and introduce topological loss to regularize the pruned subgraph to preserve the topology information. Extensive experiments suggest that our model can effectively filter out noisy information and obtain a higher performance on relation prediction via the introduced hard/soft dropout. Moreover, we show that the prediction from the generated subgraph is highly explainable.

CRediT authorship contribution statement

Sijie Mai: Conceptualization, Methodology, Software, Writing – original draft. **Shuangjia Zheng:** Conceptualization, Methodology. **Ya Sun:** Visualization, Software, Writing – reviewing & editing. **Ying Zeng:** Writing – original draft, Writing – reviewing & editing. **Yuedong Yang:** Writing – reviewing & editing. **Haifeng Hu:** Supervision, Project administration, Writing – reviewing & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant 62076262.

References

[1] S. Ji, S. Pan, E. Cambria, P. Marttinen, P.S. Yu, A survey on knowledge graphs: Representation, acquisition, and applications, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–21, http://dx.doi.org/10.1109/TNNLS.2021.3070843.
[2] H. Liu, S. Zhou, C. Chen, T. Gao, J. Xu, M. Shu, Dynamic knowledge graph reasoning based on deep reinforcement learning, Knowl.-Based Syst. (2022) 108235.
[3] Y. Zhang, H. Dai, Z. Kozareva, A.J. Smola, L. Song, Variational reasoning for question answering with knowledge graph, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[4] C. Xiong, R. Power, J. Callan, Explicit semantic ranking for academic search via knowledge graph embedding, in: Proceedings of the 26th International Conference on World Wide Web, 2017, pp. 1271–1279.
[5] H. He, A. Balakrishnan, M. Eric, P. Liang, Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings, 2017, arXiv preprint arXiv:1704.07130.
[6] A. Bordes, N. Usunier, A. García-Durán, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: Advances in Neural Information Processing Systems, 2013, pp. 2787–2795.
[7] T. Dettmers, P. Minervini, P. Stenetorp, S. Riedel, Convolutional 2d knowledge graph embeddings, 2017, arXiv preprint arXiv:1707.01476.
[8] Q. Li, D. Wang, S. Feng, C. Niu, Y. Zhang, Global graph attention embedding network for relation prediction in knowledge graphs, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–14, http://dx.doi.org/10.1109/TNNLS.2021.3083259.
[9] Z. Zhou, C. Wang, Y. Feng, D. Chen, JointE: Jointly utilizing 1D and 2D convolution for knowledge graph embedding, Knowl.-Based Syst. (2022) 108100.
[10] R. Trivedi, H. Dai, Y. Wang, L. Song, Know-evolve: Deep temporal reasoning for dynamic knowledge graphs, in: International Conference on Machine Learning, PMLR, 2017, pp. 3462–3471.
[11] C. Meilicke, M. Fink, Y. Wang, D. Ruffinelli, R. Gemulla, H. Stuckenschmidt, Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion, in: International Semantic Web Conference, Springer, 2018, pp. 3–20.
[12] F. Yang, Z. Yang, W.W. Cohen, Differentiable learning of logical rules for knowledge base reasoning, in: Advances in Neural Information Processing Systems, 2017, pp. 2319–2328.
[13] L. Galárraga, C. Teflioudi, K. Hose, F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE, VLDB J. 24 (6) (2015) 707–730.
[14] S. Mai, S. Zheng, Y. Yang, H. Hu, Communicative message passing for inductive relation reasoning, Assoc. Adv. Artif. Intell. (AAAI) (2021).
[15] K.K. Teru, E. Denis, W.L. Hamilton, Inductive relation prediction by subgraph reasoning, in: ICML, 2020.
[16] S. Zheng, S. Mai, Y. Sun, H. Hu, Y. Yang, Subgraph-aware few-shot inductive link prediction via meta-learning, 2021, arXiv preprint arXiv:2108.00954.
[17] J. Chen, H. He, F. Wu, J. Wang, Topology-aware correlations between relations for inductive link prediction in knowledge graphs, 2021, arXiv preprint arXiv:2103.03642.
[18] C. Agarwal, M. Zitnik, H. Lakkaraju, Towards a rigorous theoretical analysis and evaluation of gnn explanations, 2021, arXiv preprint arXiv:2106.09078.
[19] M.S. Schlichtkrull, N. De Cao, I. Titov, Interpreting graph neural networks for nlp with differentiable edge masking, 2020, arXiv preprint arXiv:2010.00577.
[20] H. Yuan, H. Yu, S. Gui, S. Ji, Explainability in graph neural networks: A taxonomic survey, 2020, arXiv preprint arXiv:2012.15445.
[21] T. Funke, M. Khosla, A. Anand, Hard masking for explaining graph neural networks, 2020.
[22] R. Shimizu, M. Matsutani, M. Goto, An explainable recommendation framework based on an improved knowledge graph attention network

with massive volumes of side information, Knowl.-Based Syst. 239 (2022) 107970.
[23] R. Ying, D. Bourgeois, J. You, M. Zitnik, J. Leskovec, Gnnexplainer: Generating explanations for graph neural networks, Adv. Neural Inf. Process. Syst. 32 (2019) 9240.
[24] B. Yang, W.-t. Yih, X. He, J. Gao, L. Deng, Embedding entities and relations for learning and inference in knowledge bases, 2014, arXiv preprint arXiv:1412.6575.
[25] T. Trouillon, C.R. Dance, E. Gaussier, J. Welbl, S. Riedel, G. Bouchard, Knowledge graph completion via complex tensor factorization, J. Mach. Learn. Res. 18 (1) (2017) 4735–4772.
[26] Z. Sun, Z.-H. Deng, J.-Y. Nie, J. Tang, RotatE: Knowledge graph embedding by relational rotation in complex space, in: International Conference on Learning Representations, 2019.
[27] L. Chao, J. He, T. Wang, W. Chu, Pairre: Knowledge graph embeddings via paired relation vectors, in: ACL, 2021, pp. 4360–4369.
[28] Z. Li, H. Liu, Z. Zhang, T. Liu, N.N. Xiong, Learning knowledge graph embedding with heterogeneous relation attention networks, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–13, http://dx.doi.org/10.1109/TNNLS.2021.3055147.
[29] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: European Semantic Web Conference, Springer, 2018, pp. 593–607.
[30] C. Shang, Y. Tang, J. Huang, J. Bi, X. He, B. Zhou, End-to-end structure-aware convolutional networks for knowledge base completion, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 3060–3067.
[31] D. Nathani, J. Chauhan, C. Sharma, M. Kaul, Learning attention-based embeddings for relation prediction in knowledge graphs, 2019, arXiv preprint arXiv:1906.01195.
[32] R. Xie, Z. Liu, H. Luan, M. Sun, Image-embodied knowledge representation learning, 2016, arXiv preprint arXiv:1609.07028.
[33] R. Xie, Z. Liu, J. Jia, H. Luan, M. Sun, Representation learning of knowledge graphs with entity descriptions, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016.
[34] B. Shi, T. Weninger, Open-world knowledge graph completion, 2017, arXiv preprint arXiv:1711.03438.
[35] T. Hamaguchi, H. Oiwa, M. Shimbo, Y. Matsumoto, Knowledge transfer for out-of-knowledge-base entities: a graph neural network approach, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017, pp. 1802–1808.
[36] P. Wang, J. Han, C. Li, R. Pan, Logic attention based neighborhood aggregation for inductive knowledge graph embedding, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 7152–7159.
[37] A. Sadeghian, M. Armandpour, P. Ding, D.Z. Wang, DRUM: End-to-end differentiable rule mining on knowledge graphs, in: Advances in Neural Information Processing Systems, 2019, pp. 15347–15357.
[38] Q. Wang, K. Huang, P. Chandak, N. Gehlenborg, M. Zitnik, Interactive visual explanations for deep drug repurposing, 2021.
[39] M. Zitnik, M. Agrawal, J. Leskovec, Modeling polypharmacy side effects with graph convolutional networks, Bioinformatics 34 (13) (2018) i457–i466.
[40] M. Ribeiro, S. Singh, C. Guestrin, "Why should I trust you?": Explaining the predictions of any classifier, in: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations, Association for Computational Linguistics, San Diego, California, 2016, pp. 97–101, http://dx.doi.org/10.18653/v1/N16-3020, URL https://aclanthology.org/N16-3020.
[41] P.E. Pope, S. Kolouri, M. Rostami, C.E. Martin, H. Hoffmann, Explainability methods for graph convolutional neural networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 10772–10781.
[42] Q. Huang, M. Yamada, Y. Tian, D. Singh, D. Yin, Y. Chang, Graphlime: Local interpretable model explanations for graph neural networks, 2020, arXiv preprint arXiv:2001.06216.
[43] M. Vu, M.T. Thai, Pgm-explainer: Probabilistic graphical model explanations for graph neural networks, Adv. Neural Inf. Process. Syst. 33 (2020) 12225–12235.
[44] H. Yuan, J. Tang, X. Hu, S. Ji, Xgnn: Towards model-level explanations of graph neural networks, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 430–438.
[45] X. Xu, W. Feng, Y. Jiang, X. Xie, Z. Sun, Z.-H. Deng, Dynamically pruned message passing networks for large-scale knowledge graph reasoning, in: International Conference on Learning Representations, 2019.
[46] J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, R. He, Graph information bottleneck for subgraph recognition, in: International Conference on Learning Representations, 2020.
[47] C. Louizos, M. Welling, D. Kingma, Learning sparse neural networks through L_0 regularization, in: International Conference on Learning Representations, 2018.
[48] D.P. Kingma, M. Welling, Auto-encoding variational bayes, 2013, arXiv preprint arXiv:1312.6114.
[49] Y. Zeng, S. Mai, H. Hu, Which is making the contribution: Modulating unimodal and cross-modal dynamics for multimodal sentiment analysis, 2021, arXiv preprint arXiv:2111.08451.
[50] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: A survey, J. Artificial Intelligence Res. 4 (1996) 237–285.
[51] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[52] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: EMNLP, 2014, pp. 1724–1734.
[53] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, M. Gamon, Representing text for joint embedding of text and knowledge bases, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, pp. 1499–1509.
[54] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: ICLR, 2015.
[55] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: International Conference on Learning Representations, 2018.