Negative Sampling and Rule Mining for Explainable Link Prediction in Knowledge Graphs

Kamrul Islam, Sabeur Aridhi, Malika Smaïl-Tabbone

Negative Sampling and Rule Mining for Explainable Link Prediction in Knowledge Graphs

Md Kamrul Islam*, Sabeur Aridhi, Malika Smail-Tabbone

Universite de Lorraine, CNRS, Inria, LORIA, 54000 Nancy, France

Abstract

Several KG embedding methods were proposed to learn low dimensional vector representations of entities and relations of a KG. Such representations facilitate the link prediction task, in the service of inference and KG completion. In this context, it is important to achieve both an efficient KG embedding and explainable predictions. During learning of efficient embeddings, sampling negative triples was highlighted as an important step as KGs only have observed positive triples. We propose an efficient simple negative sampling (SNS) method based on the assumption that the entities which are closer in the embedding space to the corrupted entity are able to provide high-quality negative triples. As for explainability, it actually constitutes a thriving research question especially when it comes to analyse KGs with their rich semantics rooted in description logics. Hence, we propose in this paper a new rule mining method on the basis of learned embeddings. We extensively evaluate our proposals through several experiments. We evaluate our SNS sampling method plugged to several KG embedding models through link prediction task performances on well-known datasets. Experimental results show that the SNS improves the prediction performance of KG embedding models, and outperforms the existing sampling methods. To assess the performance of our rule mining method with and without SNS, we mine and evaluate rules on three popular datasets. The

*Corresponding author

Email addresses: kamrul.islam@loria.fr (Md Kamrul Islam), sabeur.aridhi@loria.fr (Sabeur Aridhi), malika.smail@loria.fr (Malika Smail-Tabbone)

extracted rules are evaluated as knowledge nuggets extracted from the KG and also as support for explainable link prediction. The overall results are good and open the way to many improvements and new perspectives.
Keywords: knowledge graph embedding, link prediction, negative sampling,, rule mining, explainability

---

1. Introduction

Today knowledge graphs are commonly adopted for knowledge representation. A knowledge graph (KG) is a graph-based representation of knowledge which models real-world entities and their relations in various domains [1]. Formally, a KG is represented as a collection of RDF triples, (head, relation, tail) where the head and the tail are two entities which are connected by a specific relation. KGs form building blocks for many applications, ranging from question answering to content-based recommendation. Large KGs contain millions of entities and billions of triples (*e.g.,* FreeBase, DBPedia, YAGO). Despite of their huge size, KGs are often incomplete. For example, birth place of more than 70% of person entities in Freebase is missing [2]. The incompleteness issue of KGs motivates researchers to study how to add or infer new triples in KGs. This task is known as *link prediction* in KGs.

In recent years, many embedding models have been proposed to learn vector representations (or embeddings) of entities and relations in KGs. These models perform the link prediction task based on the learned embeddings. The training of these models requires positive triples as well as negative (non-observed) triples. However, only positive triples are available in KGs. Importantly, the quality of negative triples does matter as it influences the overall results of the models [3, 4]. This brings the importance of sampling negative triples. However, this feature is not well covered in the literature. To the best of our knowledge, there exist some negative sampling methods such as uniform-random [5], GAN-based [3, 6, 7], NSCaching [4],Self-adversarial [8], and SANS [9]. All these methods, except the first one, look for hard negatives for a positive triple. A

negative triple, which is hard to differentiate from positive triples, is considered a hard negative [7]. However, each of them has its own pros-cons and the current state-of-art still lacks good negative sampling methods. We proposed a simple but efficient method called SNS to sample high-quality negative triples in KG [10] making a good balance between exploration and exploitation to improve the quality of negative triples. We design SNS as a general sampling method that can be plugged to any KG embedding model for link prediction. We provide a thorough description of the method principles (Section 2.1).

Today, the major limitation of KG embedding-based link prediction methods is their lack of explainability, which limits their ability to be deployed in many real-world applications. This seems to be particularly absurd given the richness of the initial graph. Hence we propose a new rule mining method that utilizes the KG and its embedding and a strategy to use the rules to explain the embedding-based link prediction (Section 3.1).

The rest of the paper is organized as follows. After introducing the SNS sampling method and its related works in Section 2, we present our rule mining method in Section 3. In Section 4, we present our experimental settings. In Section 5, we present the obtained results of the conducted experiments to evaluate our rule mining method in combination with the SNS sampling method. The last section is devoted to the conclusions and the future directions.

2. The Simple Negative Sampling (SNS) method

In this section, we describe our SNS sampling method (Section 2.1) and related works (Section 2.2). We summarize the notations in Table 1 which are used throughout the paper.

2.1. The SNS method

For link prediction, KG embedding models are trained with positive and negative triples to learn embeddings of entities and relations. A good trade-off between exploration and exploitation is crucial in searching for a high-quality

Table 1: Summary of notations

Symbol	Meaning	Symbol	Meaning
$\mathbb{E}$	Set of all entities	Θ	Set of all parameters an embedding method
$\mathbb{R}$	Set of all relations	T	Number of epochs
$\mathbb{S}$	Set of positive training triples	k	Number of sampled negative(s) for a positive
$\mathbb{D}$	Set of positive test triples	f	Scoring function of an embedding method
$\mathbb{Q}$	Set of positive triples	G	A bidirectional knowledge graph
d	Embedding dimension size	H	Head/target relation of a rule
m	Batch size	B_i	i-th body body relation of a rule
S_m	A batch of positive triples	e_i	Entity variable of a rule
S_m	A batch of negative triples	r^{-1}	Inverse relation of a relation r

negative triples [11]. Exploration corresponds to the capacity of the sampling method to select high-quality negative triples from unexplored areas whereas exploitation favours the utilization of already known negative triples to sample other negatives. SNS is concerned with achieving a fair balance between exploration and exploitation in the quest for good negatives. We first briefly describe a classical KG embedding model, and then our proposed SNS negative sampling method. The architecture of a classical KG embedding model is given in Figure 1 which starts with initializing the embeddings of entities and relations randomly from uniform/Gaussian distributions [12]. For the training of the model, a batch of positive train triples S_m is fetched and a negative sampling method is then used to generate a batch of negative triples. In the architecture, we inject our SNS method (shaded by yellowish color in Figure 1) to generate the batch of negative triples, S'_m for the batch of positive triples,S_m. The batches of positive and negative triples are then used to learn the embeddings. In this paper, we consider the pairwise training which is able to work under the 'open-world' assumption [13]. In pairwise training, the model tries to assign more plausibility score to a positive triple than its corresponding negative triple. The training objective is to optimize the embeddings of entities and relations

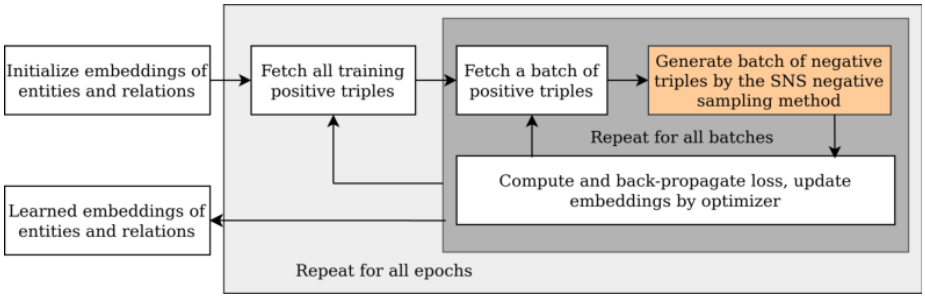


Figure 1: Architecture of a classical KG embedding model with SNS sampling

for minimizing the total pairwise loss as designed in Equation 1.

$$\min_{\Theta} \sum_{\forall_{(h,r,t)\in S_m,(h',r,t')\in S'_m}} L(f(h,r,t), f(h',r,t')) + \lambda reg(\Theta) \tag{1}$$

Here, f is the scoring function of the embedding model, $(h,r,t) \in S_m$ is a positive triple and $(h',r,t') \in S'_m$ is the corresponding negative triple. The pairwise loss for the positive and its negative triple is defined in Equation 2 [12].

$$L(f(h,r,t), f(h',r,t')) = \Big[\lambda - f(h,r,t) + f(h',r,t')\Big]_+ \tag{2}$$

Here, λ is the margin and $[.]_+ = max(0,.)$ is the hinge function. The embedding updating process is repeated for all batches of positive triples (shaded by dark gray color in Figure 1), and the whole training process (shaded by light gray color in Figure 1) is repeated for T times (or epochs). The model training process is similar to a traditional KG embedding model except we adapt our negative sampling method. We refer to [12] for more details about the traditional KG embedding.

In the following, we describe our method for negative sample generation.

Step 1. Triple perturbation: SNS, like other sampling methods, begins with perturbing a positive triple to generate an initial negative triple set. In triple perturbation, the head/tail of the positive triple is corrupted by replacing head/tail with other entities in the entity set($\mathbb{E}$). At the same time, the negative set is checked to make sure that it does not include any positive triple. To illustrate, consider the positive triple $q = (h,r,t) \in \mathbb{S}$. Corrupting the tail(t)

gives the initial negative set $q'_0(t) = \{(h, r, t') \notin \mathbb{Q} | t' \in \mathbb{E}\}$.

Step 2. Candidate set generation: Generally, the size of the set $q'_0(t)$ is large as KG contains large number of entities. Zhang et al. [4] describe that only some of initial negatives of the set are of good-quality. As we need few negatives for each positive triple, we randomly sample N_1 triples from $q'_0(t)$ to generate candidate negative set $q'_1(t)$(i.e. $q'_1(t) \subseteq q'_0(t)$) for user defined parameter N_1. As the training progresses, it is hoped that the quality of the negative(s) for the following step would be better or comparable to that of the current step's negative(s). For this purpose, the sampled negatives from the current step are kept in a small structure named least recently selected(LRS). $LRS[q'_h, q'_t]$ stores sampled negatives where q'_t and q'_h are the sampled negatives by head and tail corruption respectively in the preceding step. The stored negatives are then utilized to sample high-quality negatives in the following step. The candidate negative set, $q'_1(t)$ is updated to include the LRS negative(s) as $q'_1(t) := q'_1(t) \cup LRS[q'_t]$. The use of LRS is intended to favour the exploitation behavior of the proposed SNS method. It assures that the quality of the current step's negative(s) is better than, or at least comparable to, the quality of the preceding step's negative(s).

Step 3. Sampling probability computation: In this step, we compute the sampling probability of each negative in the candidate negative set $q'_1(t)$. Negatives having a higher probability are assumed to be of higher quality. The probability is defined based on the distance of each negative. The distance for a negative triple, $(h, r, t'_i) \in q'_1(t)$ is computed as the Euclidean distance between the corrupted(t) and the new entities(t'_i) as Equation 3.

$$d(t, t'_i) = ||\mathbf{t} - \mathbf{t'_i}|| \tag{3}$$

Here, $\mathbf{t}$ and $\mathbf{t'_i}$ are embeddings of the entities t and t'_i. The sampling probability of each candidate negative, $(h, r, t'_i) \in q'_1(t)$ is then computed using a softmax

function as Equation 4.

$$P(h, r, t'_i) = \frac{exp(\frac{1}{d(t,t'_i)})}{\sum_{j=1}^{N_1} exp(\frac{1}{d(t,t'_j)})} \tag{4}$$

For a candidate negative triple with a lower distance, the softmax function returns a higher sampling probability.

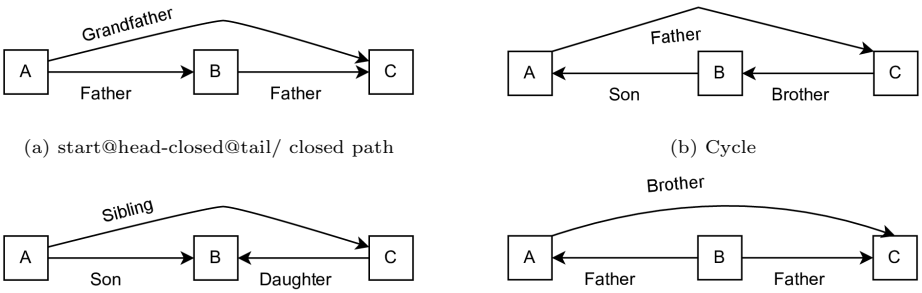
Step 4. Negative triple sampling and LRS updating: The triples from the candidate negative set,$q'_1(t)$ are ranked in decreasing order of their probabilities and k negative(s) are sampled. A natural choice could be sampling top-k (k=1 for pairwise training, $k > 1$ for maximum likelihood training) negative(s). However, sampling the top-k ranked negative(s) may lead to two issues. Firstly, because the current candidate negative set also includes the least recently sampled (LRS) high-quality negatives, there is a high chance of repeated sampling of the same negative(s) (even in several successive steps). This issue affects the exploration potential of SNS sampling. Secondly, the existence of false negative triples (that appear to be of high quality) cannot be overlooked [12]. To tackle these issues, SNS randomly samples k negative triples from N_2 top ranked triples in q'_t as $q'_t = \{(h, r, t')|rank_{(h,r,t')\leq N_2}\}$ for user-defined parameter N_2 where $N_2 > k$.

SNS repeats the above described process (from initial negative set generation to k negative(s) sampling) for head(h) corruption to sample k negative(s) as q'_h for the positive triple, q. The $LRS[q'_t, q'_h]$ is updated with the sampled 2k high-quality negative(s). Finally, we randomly sample k high-quality negative(s) as $q'_{h,r,t}$ from 2k negatives in $q'_h \cup q'_t$. The sampled k negative(s), q' and the corresponding positive, q are then used to train the KG embedding model.

2.2. Related works

During the training of KG embedding-based model, high-quality negative triples, which are not readily available, have a significant contribution. One simple way to sample negative triples is based on 'closed-world' assumption where all non-observed triples are assumed to be false and can be used as negatives.

However, this assumption is not entirely true for KGs due to their incompleteness [12]. Alternatively, most of the KG embedding models sample negatives from non-observed triples under 'open-world' assumption where a non-observed triple may be positive or negative. Generally, candidate negative triple set is generated by positive triple perturbation where head/tail entities are replaced with other possible entities [12] and then negative triples are sampled from the set. 'Uniform-random' is the mostly used negative sampling method where negative triples are randomly sampled from a uniform distribution of candidate negatives [4]. Though 'uniform-random' is simple, a sampled negative triple could be easily classified as negative. Consequently, 'uniform-random' method suffers from 'zero-loss'/'vanishing-gradient' problem [3]. To avoid this problem, the generative property of GAN frameworks was recently proposed to generate high-quality negatives. The generator of GAN is trained to pick high-quality negative triples whereas the discriminator part is trained to learn embeddings. IGAN [7], KBGAN [3], KSGAN [6] are three GAN-based sampling methods for KGs. Compared to 'uniform-random' sampling, these methods improve the quality of negative triples but they increase the number of model parameters and take extra costs on training for parameter optimization [4]. In addition, they suffer from instability and degeneracy problems because of adopting the complex reinforcement learning to train the generator [4]. To avoid the excessive training time of GAN-based methods, Zhang et al. [4] proposed a 'distilled' version of GAN-based methods, namely NSCaching which stores negatives with high scores in head and tail caches for each positive triple, and then samples negatives directly from the caches. With NSCaching sampling, KG embedding models show competitive link prediction performance. However, the memory requirement increases exponentially with the size of KGs. Also, the regular updating of caches increases the computational time. Thus, scalablity is a big issue for NSCaching and it is not recommended for large KGs. Self-adversarial [8] is another adversarial negative sampling method which computes high sampling probability for high triple scoring negatives. However, negatives with high scores may be positive though they are unobserved. This 'false-negative' case is over-



(a) start@head-closed@tail/ closed path

(b) Cycle

(c) start@head-closed@middle

(d) start@middle-closed@tail

Figure 2: Different types of paths for defining length-2 rules

looked in self-adversarial sampling. Apart from the above-mentioned sampling methods, the hard negative mining in contrastive learning motivates Ahrabian et al. [9] to study the neighborhood information of entities to sample negatives. They develop SANS method based on an assumption that neighbor entities without direct relation are good candidates for generating negatives. However, the one-time generation of negatives prior to the start of the embedding learning process is expensive in terms of memory requirements, as it requires the whole adjacency matrix of a KG in main memory.

3. The rule mining method

In this section, we start with describing our rule mining method which learns rules from a knowledge graph and its embedding (Section 3.1) and the strategy of using learned rules in explaining predictions (Section 3.2). Then we describe related works (Section 3.3).

3.1. Rule Mining from a Knowledge Graph and its Embedding

To mine rules from a KG, we focus on different path types in a KG. Figure 2 illustrates four types of paths of length 2 in the 'Family' KG. The existing embedding-based rule mining methods only handle the first type i.e. 'closed-path' (Figure 2a) and the rest three types (Figures 2b,2c,2d) are not handled. We focus on mining rules that consist of a head relation H and a sequence of

body relations $B_1, B_2, ..., B_n$ in the following form

$$B_1(e_0, e_1) \wedge B_2(e_1, e_2) \wedge ..... \wedge B_n(e_{n-1}, e_n) -> H(e_0, e_n) \tag{5}$$

where e_i is an entity variable, H is the head/target relation from e_0 to e_n and $B_1, B_2, ..., B_n$ are the sequence of body relations. In the rule body, $B_i(e_{i-1}, e_i)$ is either a relation (i.e. (e_{i-1}, r_i, e_i)) or its inverse (i.e. (e_{i-1}, r_i^{-1}, e_i)).

We give more importance to shorter paths than longer ones. A path is a sequence of triples in which one common entity appears in between two consecutive triples. We compute the score of a path, P as the average score over all triples in a path, penalized by the path length. We define the path scoring function as

$$Path\ score, PS(P) = \frac{1}{l^{1+\alpha}} \sum_{i=1}^{l} f(h_i, r_i, t_i) \tag{6}$$

where f is the embedding scoring function, l is the length of the path, α is the penalizing factor and $0 \leq \alpha \leq 1$. $\alpha = 0$ means we consider all paths equally whatever their length whereas larger values give more importance to shorter paths.

In the following we describe the procedure for rule mining which starts with graph generation, proceeds with triple sampling and ends with mining rules.

3.1.1. Bidirectional graph generation

To facilitate the search of the diverse paths to be used in in the rule bodies, we generate a bidirectional graph, G. To do so, we first generate a graph from KG training triples. For each triple $(h, r, t) \in G$, we add an inverse relation, i.e. $G \leftarrow G \cup (t, r^{-1}, h)$ if there is no relation from t to h. In this way we make the graph bidirectional so that for every triple the tail is accessible from head as well as the head is accessible from tail. This transformation helps us to extract diverse path types (examples in Figures 2b-2c).
For simplicity, we illustrate the procedure of mining rules for a specific relation, r in Figure 3. We describe the process in the following sub-sections.

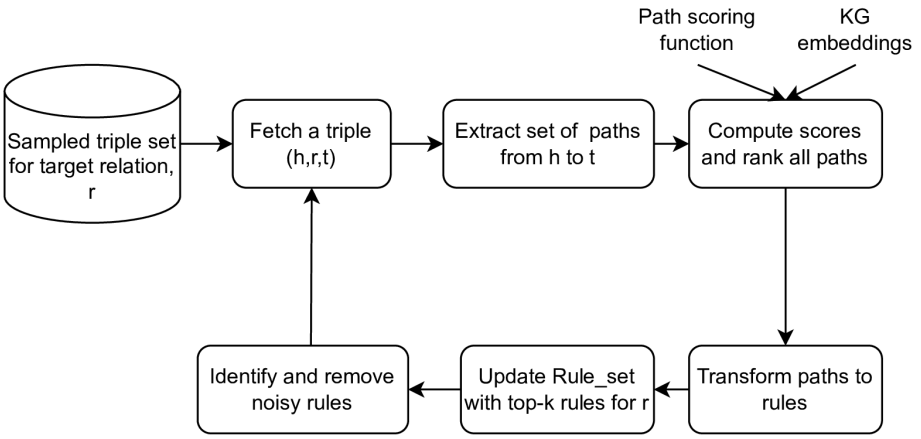


Figure 3: Rule mining for a target relation, r

3.1.2. Sampling triples

Real-world KGs contain millions of triples for a relation and it is intractable to process such a huge number of triples for rule mining. We assume that mining rules from a portion of the entire triple set well reflect the rules over the entire triple set for a relation. To mine rules, we randomly select a portion of the triple set from the whole triple set of a relation.

3.1.3. Initial rule set mining

The initial rule set mining procedure for a relation, r is illustrated in Figure 3. In the following, we describe each step in details.

Step 1 (Fetch a triple): We consider the triples to arrive sequentially from the sampled triple set. The procedure starts with fetching the arrived triple (h,r,t). Note that only one triple is fetched at a time and the next triple is fetched after complete processing of the current triple.

Step 2 (Extracting path chunk): After fetching the triple, the relation, r from h to t and its inverse(r^{-1}) (if it exists) are removed temporarily from the graph. Then a set of simple paths up to a length, MaxL from h to t are extracted. Here, MaxL is the predefined maximum body length, or simply maximum rule length. A simple path is a directed path from h to t with no repeated entities. After extracting the path set, the removed relations between h and t are added

again to make the graph in its original form.

Step 3 (Computing path scores and ranks): The scores of all extracted paths are computed using the paths scoring function in Equation 6. Note that a path may contain the inverse relation of the form (h_i, r_i^{-1}, t_i). In this case, the triple score is computed as $f(t_i, r_i, h_i)$. This computation allows us to include all the cases in Figure 2.

The paths are then ranked in decreasing order of their scores. Better ranking of a path means better chance that the path will generate a good rule.

Step 4 (Defining rules from paths): In this step, a path is transformed into a rule by replacing the entities of triples with variables. If the head and tail of a triple (h_i, r_i, t_i) are replaced with variables e_{i-1} and e_i then the triple is transformed into the body relation in the form $r_i(e_{i-1}, e_i)$. An inverse relation (h_i, r_i^{-1}, t_i) is transformed in the form $r_i(e_i, e_{i-1})$.

Step 5 (Update rule set): If a high-quality rule, 'Rule' does not exist in initial rule set, Rule_set of a relation then its is added by initializing its structure as follows

$$Rule.count = 1$$

$$Rule.energy = 1.0 + \gamma$$

The 'count' variable of the rule represents the number of occurrences of the rule and the energy variable takes the full energy and an additional decay rate (γ) as initial energy. The rule is then added to the initial rule set, Rule_set i.e. $Rule\_set \leftarrow Rule\_set \cup Rule$. In case the rule 'Rule' already exists in the Rule_set, then its count and energy values are updated as

$$Rule\_set[Rule].count = Rule\_set[Rule].count + 1$$

$$Rule\_set[Rule].energy = 1.0 + \gamma$$

We define the top-k ranked extracted rules as high-quality rules where k is an user-defined parameter.

Step 6 (Identify and remove noisy rules): The rules which are not useful in mining high-quality rules for a relation are identified as noisy and removed immediately. This removal reduces unnecessary memory and computing time consumption. In our method, a rule falls in one of three statutes. A 'potential' rule has positive energy but count below 2. A 'stable' rule has count above 1, and a noisy rule has count below 2 and negative energy. A 'potential' rule may become 'stable' or 'noisy' later. Every time a triple is fetched from the sampled triple set, the energy of each potential rule, 'Rule' from the Rule_set is decreased by the decay rate γ as

$$Rule\_set[Rule].energy = Rule\_set[Rule].energy - \gamma \tag{7}$$

Usually the decay rate is $0 < \gamma < 1$. Small value of γ means potential rules will be alive for longer period before dying if they don't last for long time. Generally, the number of paths between two entities is high in a dense KG and low in a sparse KG. Hence, we argue that the possibility of the presence of noisy rules for a relation is high in a dense KG and low in a sparse KG. We recommend setting a high decay rate for a dense KG and a low decay rate for a sparse KG. All potential rules with negative energies are identified as noisy rules and immediately removed from memory.

3.1.4. Final rule set generation

The above rule mining processes (Section 3.1.3) is repeated until all triples from the sampled triple set are processed. At the end of processing, the Rule_set contains 'stable' rules and a few 'potential' rules. The final rule set only contains the 'stable' ones.

Algorithm 1 describes the ranked path set extraction steps and Algorithm 2 describes the rule set mining steps based on the extracted path set.

[1] https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.simple_paths.all_simple_paths.html#networkx.algorithms.simple_paths.all_simple_paths

Algorithm 1: Extract_paths: Extracting paths for a given triple

Input: Bidirectional Knowledge graph(G), a triple (h,r,t), maximum path length (MaxL)

Output: Paths, Ranks

```
G←G-{h,r,t}    // temporarily remove the relation from h to t
Path_Scores← ∅
Ranks← ∅
Paths=all_simple_paths(G, source=h, target=t, cutoff=MaxL) [1]
/* extract all simple paths from h to t with maximum path
length, MaxL  */
foreach P∈Paths do
    Score_P ← Score of the path, P                  // using equation 6
    Path_Scores ← Path_Scores ∪ Score_P
Ranks ← Ranks of paths based on Path_Scores      // in decreasing
order of path scores
G←G∪{h,r,t}
```

3.2. Abduction for Explainable Link Prediction

Once the rules are learned, they can be exploited to provide plausible explanations to support certain predictions. We propose a procedure inspired by abductive reasoning over the rules to infer the best explanation(s) for a prediction. Indeed abductive reasoning using a body− >head rule allows us to infer (or at least assume) that body is satisfied whenever head is observed. Hence given a prediction as an observation, we look for rules whose head corresponds to the prediction. Each rule instantiation (using the bidirectional graph paths) constitutes then a plausible explanation for the prediction. Thus we have a way to further rank the top-k predictions based on their explainability.

3.3. Related works

Inducing first-order logical rules from a logical program (knowledge base containing facts and rules) in ILP (Inductive Logic programming) or multi-

Algorithm 2: Learn_rule: Learning rules for a relation

Input: Target relation (r), Bidirectional Knowledge graph (G), Triple set (T_r) with r, Decay rate (γ) , Sampling rate (β), Top-k

Output: Rule_set

$T'_r \leftarrow$ Randomly sample ($|T_r| * \beta$) triples from T_r
foreach *(h,r,t)*$\in T'_r$ **do**
    Paths, Ranks =Extract_paths(G,(h,r,t),L) `// Call Algorithm 1 to extract paths`
    `/* If a path exists then update its info, otherwise add the path to PathMetaInfo if it is ranked in top-k */`
    foreach $P \in Paths$ **do**
        Rule=Transform_path_to_rule(P) `// replace entities with literals`
        if $Rule \in Rule\_set$ **then**
            Rule_set[Rule].Count$\leftarrow$ Rule_set[Rule].Count + 1 `// increment counter`
            Rule_set[Rule].Energy$\leftarrow 1 + \gamma$ `// reset energy to full and additional` γ
        else
            `/* add a new rule */`
            if $Ranks[Rule] \leq top\text{-}k$ **then**
                Rule.Count$\leftarrow$ 1
                Rule.Energy$\leftarrow 1 + \gamma$
                Rule_set[Rule]$\leftarrow$ Rule_set $\cup$ Rule
    `/* Identify the noisy rules */`
    foreach $Rule \in Rule\_set$ **do**
        Rule.Energy$\leftarrow$Rule.Energy - γ
        if $Rule.Energy \leq 0$ & $Rule.Count \leq 1$ **then**
            Rule_set=Rule_set - Rule `// remove the noisy rule`
`/* generate the final rule set by removing potential rules */`
foreach $Rule \in Rule\_set$ **do**
    if $Rule.Count \leq 1$ **then**
        Rule_set=Rule_set - Rule `// remove the potential rule`

relational learning constituted seminal and important works in symbolic AI [14, 15]. The main limitation of such methods is their lack of scalability on large knowledge bases and their main strength is their high human-readability. In parallel, pattern mining algorithms were designed to extract association rules between frequent itemsets found in a binary data table [16, 17, 18]. In both cases, the achieved rule set constitute a good synthetic description of the data. Each rule can be evaluated using certain metrics and can also be exploited in prediction (or classification) mode using a deductive reasoning schema (*i.e., If*

the left (body) part of a rule is true *Then infer That* the right (head) part of the rule is true).

With the generalization of KGs and KG embeddings, two approaches aim at mining rules from learned embeddings: (1) EmbedRule [19] and (2) RLvLR (Rule Learning via Learning Representations) [20]. Our work differs from theirs in several points. Firstly, both existing methods mine rules based on closed path (CP) pattern, whereas our method considers more path patterns for mining better rules. Secondly, our method is able to detect and remove noisy rules early enough to reduce memory and computational time requirements. Finally, the existing methods only consider relation embedding for mining rules, thus losing the entity embedding information as the validity of a triple depends on both entity and relation embeddings. In contrast, our method utilizes the triple scoring function of an embedding method to define the path scoring function so as to exploit both entity and relation embeddings in mining high-quality rules.

4. Experimental settings

4.1. Experimental configuration

To evaluate the efficiency of SNS sampling, we plug it to a KG embedding model. The main focus of this paper is negative sampling for KGs. For evaluation, we choose TransH [21] to represent translational models and DistMult [19] to represent semantic matching models as they are popular baselines for link prediction task in KGs [12]. For details about the models, we refer to the original papers. The scoring functions of the models are given in Table 2. Each model is evaluated for four existing sampling methods (i.e. uniform-random, GAN-based, NSCaching, self-adversarial) and for the proposed SNS methods. We do not include SANS in the experiment due to its excessive memory requirement. For GAN-based method, we choose KBGAN as it is the only GAN-based sampling which has publicly available implementation (to best of our knowledge). We refer to the original articles for details about the sampling methods.

Table 2: Scoring functions of KG embedding models: w_r is the normal vector of the hyperplane for the relation r, $diag(r)$ is the diagonal matrix for the relation r, and $||.||_2$ represents l_2 norm.

Model	**Embeddings**	**Scoring function,** $f(h, r, t)$
TransH [21]	$\mathbf{h}^d, \mathbf{t}^d, \mathbf{r}^d, w_r^d$	$\|\|(\mathbf{h} - w_r^T \mathbf{h} w_r) + \mathbf{r} - (\mathbf{t} - w_r^T \mathbf{t} w_r)\|\|_2^2$
DistMult [19]	$\mathbf{h}^d, \mathbf{t}^d, \mathbf{r}^d$	$\mathbf{h^T} diag(r) \mathbf{t}$

To evaluate our rule mining method, we used the embeddings learned by DistMult with SNS and DistMult with random sampling. The source code of this paper can be found in our public GitLab repository [2]. In the repository, we provide instructions to execute the whole pipeline on the KGs mentioned as well as on a new KG.

4.2. Datasets

In the experiments, we use six widely used benchmark KG datasets, i.e., FB15K, FB15K-237, WN18, WN18RR, YAGO3-10, Family for link prediction task [5, 3, 4]. WN18 is derived from the WordNet which is a large semantic lexicon for the English language. FB15K is a subset of triples from Freebase KG which is a large collaborative general knowledge base. WN18RR and FB15K-237 datasets are derived from WN18 and FB15K respectively after removing the inverse-duplicate relations. The YAGO3-10 dataset is extracted from the open source YAGO knowledge base considering the entities with at least 10 relations. The last KG is Family KG, where the relations are family relationships among people [22, 23]. This KG is used to assess the rule mining method as the relationships in Family are easily interpretable. The KG datasets come with train/valid/test splits. Table 3 presents characteristics of the used datasets in terms of number of entities, number of relations, number of facts and number of triples in the training, test and validation sets.

[2] https://gitlab.inria.fr/kislam/kglp

Table 3: The experimental KG datasets

KG datasets	#Entity	#Relation	#Facts	#Training	#Validation	#Test
WN18RR	40,943	11	93,003	86,835	3,034	3,134
WN18	40,943	18	151,442	141,442	5,000	5,000
FB15K-237	14,541	237	310,116	272,115	17,535	20,466
FB15K	14,951	1,345	592,213	483,142	50,000	59,071
YAGO3-10	113,273	37	1,089,040	1,079,040	5,000	5,000
Family	3,007	12	28,356	25,517	1,410	1,429

4.3. Evaluation metrics

4.3.1. Evaluation metrics for the SNS method

To evaluate the performance of any sampling method, we plug it to a KG embedding model for link prediction task. The performance is defined with two widely used metrics: Hit@z, and mean reciprocal rank (MRR) [24]. The metrics are defined based on the rank of the positive test triple. Hit@z is defined as the average number of times a positive test triple is among the z highest ranked triples; whereas MRR is the average reciprocal rank of the positive test triple [25]. The range of both scores is 0 to 1. The higher value of MRR demonstrates the better ranking of positive test triples and better ranking provides better prediction performance. Also, higher Hit@z score indicates better performance. To illustrate, consider the positive test triple $q = (h, r, t) \in \mathbb{D}$. A set of negative triples $q'_t = \{(h, r, t') \notin \mathbb{Q} | t' \in \mathbb{E}\}$ is generated by simple triple perturbation (replacing tail with other entities) [5, 26] confirming that no positive triple exists in q'_t. The triples in $q \cup q'_t$ are then ranked in decreasing order of their scores (computed by embedding-based scoring functions in Table 2). The rank of the positive test triple q in $q \cup q'_t$ is defined as $rank^t_q$. Based on the rank of each positive test triple q, the performance metrics are defined in Equations 8 and 9.

$$Hit^t@z = \frac{1}{|\mathbb{D}|} \sum_{q \in \mathbb{D}} hit^t_q, \quad hit^t_q = \begin{cases} 1, & if\ rank^t_q \leq z \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

$$MRR^t = \frac{1}{|\mathbb{D}|} \sum_{q \in \mathbb{D}} \frac{1}{rank_q^t} \tag{9}$$

The whole evaluation process is also repeated by corrupting the head entity of each positive triple as well and $Hit^h@z$, MRR^h are computed. The final $Hit@z$, MRR metrics are the average of head and tails metrics i.e. $Hit@z = (Hit^t@z + Hit^h@z)/2$ and $MRR = (MRR^t + MRR^h)/2$. We re-scale the Hit@z score from the range 0-1 to 0-100 to facilitate the comparisons. As suggested by the most of the literature for link prediction in KGs, we consider $z \in \{1, 3, 10\}$.

4.3.2. Evaluation metrics for the rule mining method

To assess the quality of learned rules, we rely on two common statistical measures standard confidence (SC) and head coverage (HC) [20]. Both measures are defined based on the support of a rule in the following form.

$$Rule : r_1(e, e_1) \ \wedge \ r_2(e_1, e_2) \ \wedge \ .... \wedge \ r_n(e_{n-1}, e_n) \longrightarrow r(e, e_n)$$

The support of a rule is defined as the number of triples instantiating the rule [20].

$$Support(Rule) = \#(e, e_n) : body(Rule) \cap head(Rule) \tag{10}$$

SC is defined as the ratio of support and the number of entity pairs instantiating the body. The HC is the ratio of support and the number of entity pairs instantiating the head.

$$SC(Rule) = \frac{Support(Rule)}{\#(e, e') : body(Rule)} \tag{11}$$

$$HC(Rule) = \frac{Support(Rule)}{\#(e, e') : head(Rule)} \tag{12}$$

4.4. Configuration of rule-based explanation of predictions

As for explainable link prediction we need a metric to measure the potential of explanation of the rule set. We consider a positive test triple with a hit@10 score of 1 as a correct prediction. We aim to explain such predictions that are made by the embedding-based method. For each predicted triple, we identify

the rules which make the prediction true through abduction-based instantiation (Section 3.2). The authors of the literature rule mining methods use precision metric to assess prediction performance [19, 27]. Considering the incompleteness of a KG, we argue that precision could be a misleading metric as an unobserved triple is not necessarily false. Instead, we compute the recall score to evaluate the performance of our rule mining method. We define a true positive prediction as a test triple explained at least by one rule and the number of positives is the number of triples in the test set.

5. Results and discussion

We describe the link prediction performance of embedding-based methods with SNS sampling in Section 5.1 to assess the impact of SNS in improving embedding quality. In Section 5.2, we describe the performance of our rule mining method with its sensitivity to parameters. Finally, we explain a few embedding-based predictions with the help of mined rules in Section 5.3.

5.1. Link prediction performance

The proposed SNS method is implemented in Python with the well-known PyTorch and run on a 'NVIDIA A100-PCIE-40GB' GPU. For all of the experiments, we set the training epochs to 200, the embedding dimension to 100, the learning rate to 0.0001, and the margin value to 4.0. For the embedding optimization task, we use the popular Adam optimizer [28]. For all the datasets, the link prediction performance of embedding models with SNS sampling is computed and compared to other sampling methods. Then, for the WN18RR dataset, we do further analyses including parameter sensitivity and performance evolution with the number of epochs.

Prediction Performance: We train the KG embedding models from scratch for each sampling method, Random-uniform, KBGAN, NSCaching, SNS, self-adversarial (Self-Adv). For parameter setting, we follow the recommendations from the original papers ([4] for NSCaching, [3] for KBGAN, [8] for Self-Adv).

Table 4 shows the prediction metrics. With respect to most of the prediction metrics in most of the KG datasets, the proposed SNS clearly outperforms all other negative sampling methods when they are plugged into the translational model (TransH). When injected into the TransH, SNS improves hit@k scores by 2-5 percent compared to the second best sampling method in the WordNet KGs (WN18RR, WN18). SNS sampling produces the best results for the FB15K, Family datasets, with Hit@k scores improving by 5-8% when compared to the second best sample methods. For the YAGO 3-10 dataset, SNS also remains best or second best in almost all metrics. The most suitable reason for this success could be a good balance between exploration and exploitation. When the sampling methods are used with DistMult model, the hit@10 and hit@3 scores drop in all datasets. The metrics could be improved by training the model for more epochs. However, we are comparing different sampling strategies rather than different prediction models. SNS with the DistMult method shows a similar trend of improved prediction performance.

In this section, we provide a visualization of the closeness of the original and new entities in positives and their corresponding negatives in the second smallest KG dataset, WN18RR. Different negative sampling methods with TransH are used to learn embeddings. For the visualization, the negatives in the final epoch (i.e. epoch number 200) are analyzed. In Figure 4, we show the distribution of distances between the original and the new entities for different sampling methods. The box for SNS sampling shows the minimum median distance. Overall, the minimum median illustrates that the selected entities for negative sample generation are closer to the original entities in SNS than other sampling methods. The selection of close entities in SNS leads to the generation of hard negative triples and improves the prediction performance in turn, as seen in Table 4.

In the following, we describe further analyses of prediction by TransH with different sampling methods in the second smallest KG dataset, WN18RR.

Change in prediction performance for different epoch numbers: We present the MRR and Hit@10 scores of an embedding model in Figure 5 with

Table 4: Link prediction(LP) results: MRR, and Hit@z of different negative sampling(NS) methods. The best and second best metrics are marked in bold and underline faces.

LP models	**NS methods**	**WN18RR**				**WN18**			
		MRR	hit@10	hit@3	hit@1	MRR	hit@10	hit@3	hit@1
TransH	Random	0.1520	32.27	23.72	0.11	0.3199	79.43	64.25	11.85
	NSCaching	0.1713	40.68	31.43	0.93	0.4171	88.65	74.05	17.48
	KBGAN	0.1708	40.08	29.35	0.10	0.4183	87.34	73.67	18.09
	Self-Adv	0.1709	41.18	31.16	0.89	0.4191	89.48	75.84	7.08
	SNS	**0.1852**	**43.04**	**33.82**	**1.80**	**0.448**	**91.47**	**79.30**	**19.28**
DistMult	Random	0.1918	32.16	23.88	14.22	0.3453	52.82	37.33	25.75
	NSCaching	0.2262	37.37	**29.11**	17.84	0.3772	56.85	**42.18**	29.04
	KBGAN	0.2285	33.42	27.23	17.34	0.3791	57.28	41.97	28.39
	Self-Adv	0.2279	36.23	26.34	17.05	**0.3940**	**58.43**	41.31	31.29
	SNS	**0.2333**	**37.83**	25.12	**18.49**	0.3931	56.46	42.13	**31.38**
		FB15K237				**FB15K**			
TransH	Random	0.1988	36.68	22.50	11.50	0.3115	52.88	36.68	19.58
	NSCaching	0.2476	40.39	26.59	**17.33**	0.3926	61.22	44.99	25.50
	KBGAN	0.2162	40.58	23.52	15.23	0.3228	53.67	38.34	20.52
	Self-Adv	0.2350	41.01	26.49	14.73	0.3883	61.09	44.05	25.74
	SNS	**0.2514**	**42.90**	**29.44**	15.18	**0.4360**	**66.72**	**51.52**	**30.58**
DistMult	Random	0.1918	31.82	20.71	12.96	0.2188	33.25	21.84	12.95
	NSCaching	0.2205	34.87	25.69	15.72	0.2327	39.89	27.41	16.19
	KBGAN	0.2282	36.23	**27.23**	13.02	0.2203	35.54	23.12	15.92
	Self-Adv	0.2400	37.09	25.82	17.34	0.2493	39.80	28.38	18.53
	SNS	**0.2471**	**38.18**	26.97	**17.80**	**0.2937**	**45.62**	**32.66**	**20.79**
		YAGO3-10				**Family**			
TransH	Random	0.0850	18.96	9.05	1.24	0.3164	88.77	48.67	3.43
	NSCaching	0.1431	**26.76**	15.97	7.49	0.3434	90.34	55.04	4.65
	KBGAN	0.1467	26.08	16.03	8.34	0.3742	90.21	57.12	8.63
	Self-Adv	0.1443	26.46	**17.34**	**9.92**	0.3510	89.58	56.36	8.08
	SNS	**0.1488**	26.72	16.23	8.87	**0.4146**	**92.97**	**62.39**	**13.12**
DistMult	Random	0.0533	10.59	5.44	2.35	0.5002	77.36	60.14	37.45
	NSCaching	**0.0875**	14.22	**8.86**	**5.64**	**0.5274**	78.55	**61.34**	38.07
	KBGAN	0.0712	13.98	7.79	3.19	0.5120	78.01	60.75	38.01
	Self-Adv	0.0868	16.04	8.74	5.07	0.5122	78.59	60.95	**38.33**
	SNS	0.0804	**16.72**	8.39	4.98	0.5190	**79.32**	61.02	37.98

different sampling methods from epoch 10 to 200 with a 10 epochs interval to demonstrate how prediction performance varies as the number of training epochs grows. SNS has a similar MRR score (approximately 0.02) to other sampling methods at the start of epoch 10, with the exception of KBGAN, which has the

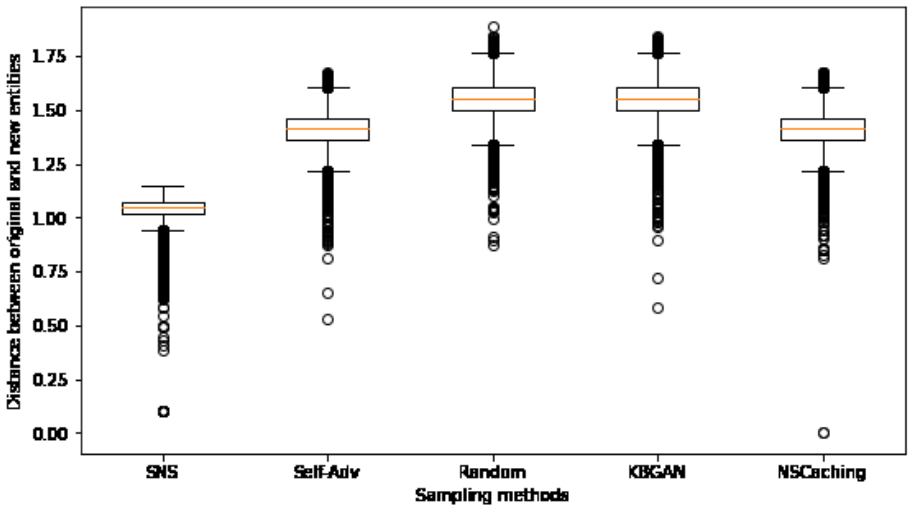
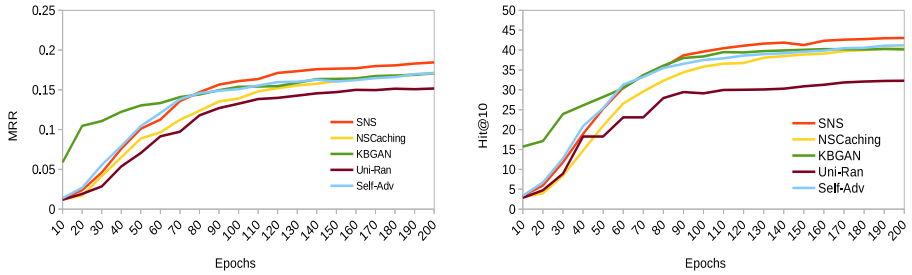


Figure 4: The distribution of distances between the original and the new entities in the positives and their corresponding negative triples in the WN18RR dataset: The embeddings of entities and relations are learned by TransH with different sampling methods.

highest MRR score (around 0.06).As the number of training epochs increases, the quality of the embeddings improves, resulting in a rise in MRR scores for all methods. The SNS method has a higher MRR improvement rate. SNS has the highest MRR score at epoch 70. The MRR scores of the proposed SNS method remain the highest among all sampling methods in the subsequent epochs, despite the fact that the improvement rate is not constant. The 'uniform-random' method has the lowest MRR because it does not manage to pick high-quality negatives. These improvements in rank are reflected in Hit@10 scores curves where SNS has the highest Hit@10 scores in later half of training epochs. These improvements in performance prove that our sampling method is able to provide better ranks of test triples than the state-of-art sampling methods.

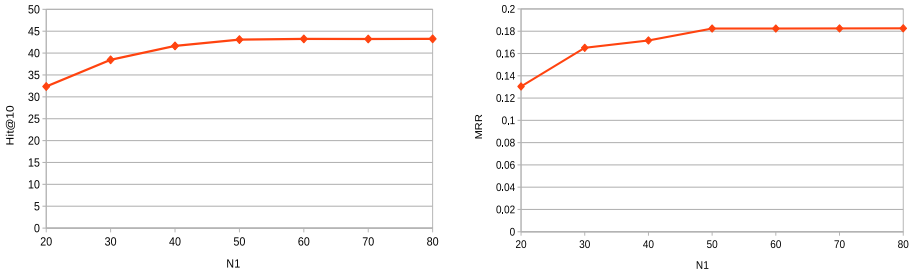
Parameter sensitivity analysis: SNS sampling method has two parameters, N_1 and N_2. To describe the changes in performance for different values of these parameters, we record Hit@10 and MRR scores for different values of N_1 with fixed $N_2 = 5$ which are plotted in Figure 6. As the value of N_1 increases, more initial negative triples are explored and the SNS sampling gets



(a) Epoch number vs MRR

(b) Epoch number vs Hit@10

Figure 5: Prediction scores of TransH with different samplings in different epochs



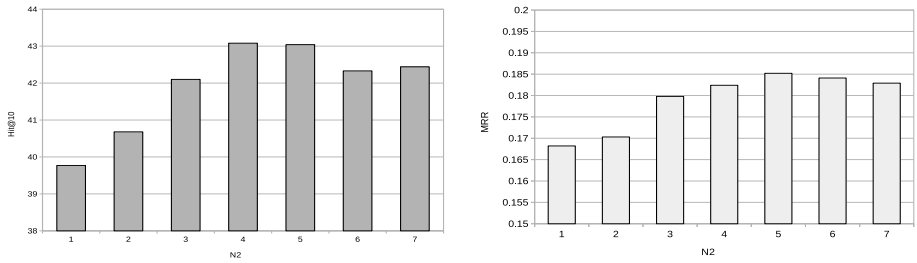
(a) N_1 vs Hit@10

(b) N_1 vs MRR

Figure 6: Sensitivity of SNS to N_1, size of candidate negative set

better exploration. As a consequence, the prediction performance improves as the value of N_1 increases from 20 to upper values as seen from Figs. 6a, 6b. The prediction performance is nearly stable for $N_1 = 50$ and above. In the point $N_1 = 50$, SNS sampling has good exploration to sample sufficient number of high-quality negatives. And this could be the cause of performance stability for $N_1 \geq 50$.

Again, to describe the sensitivity to the parameter N_2, we plot the performance metrics by varying N_2 among {1, 2, 3, 4, 5, 6, 7} with fixed N_1=50 in Figure 7. Figs. 7a and 7b show the change in Hit@10 and MRR for change in N_2. With setting N_2=1, SNS samples the top-most ranked negative(s). In this case, the chance of repeated sampling is high as SNS considers already known high-quality LRS negative(s) in addition to other candidate negatives. In case of high repeated sampling, SNS suffers from low exploration and high exploita-



(a) N_2 vs Hit@10 (b) N_2 vs MRR

Figure 7: Sensitivity of SNS to different N_2 values

tion effect leading to drops in MRR and Hit@10 metrics. With N_2=1, we see lowest prediction performance for SNS. As the value of N_2 increases, SNS gets better exploration and the best balance between exploration and exploitation is found for N_2=5 where the highest prediction metrics are recorded. However as the value of N_2 increases, the chance of sampling of known high-quality triple decreases (poor exploitation) and the chance of sampling less good-quality negative increases. As a result, the performance drops as seen in Figure 7 where both hit@10 and MRR drop for $N_2 > 5$.

Memory and computational efficiency Undoubtedly, random-uniform is the simplest, fastest, most memory efficient sampling method as it does not learn or store any parameter. GAN-based sampling makes the prediction model more complex, increases the number of training parameters, and makes the model harder to train due to use of reinforcement learning [4]. As a consequence, KBGAN needs extra memory and computational cost to store and optimize the parameters. Self-adversarial sampling is simpler and more memory efficient than GAN-based sampling as it does not require to train a generator module like GAN-based sampling. NSCaching stores set of high-quality negative triples in each positive triple cache which makes it worst memory efficient. In addition, the method takes additional time to update the cache periodically. The proposed SNS sampling method does not increase the training parameter like GAN-based method. It memorizes only the least recently sampled negative triple which takes very small amount of memory. Thus, intuitively SNS sampling is more memory

efficient than GAN-based sampling and NSCaching. The method does not use complex learning method like GAN-based method or does not take extra cache updating time like NSCaching. It takes very small amount of time to update the LRS structure. Considering the training time of each sampling method with the TransH embedding model, we see that the training time increases as the number of training samples increases, as expected. We also find 7-40% and 3-14% improvement in training time for SNS when it is compared to KBGAN and NSCaching respectively.

Examples of negative triples: In this section, we tabulate the sampled negative triples by different sampling methods with TransH for five randomly selected positive triples from the WN18RR KG dataset. From Table 5, the sampled negative triples by the 'random' method are almost unrelated to the positive triple. For example, the method generates (pakistan,has_part,deposit) which is totally unrelated to the positive triple (pakistan,has_part,tirich_mir) (at least seen with an open eye). Compared to other methods, SNS samples more semantically related negative triples, which is different from type relatedness. For example, SNS replaces 'tirich_mir' with 'amazon' which generates better negative triple than replacing 'tirich_mir' with 'malevolence' by NSCaching or 'tirich_mir' with 'chute' by KBGAN, or 'tirich_mir' with 'deposit' by the 'random' sampling method. The empirical results show an improvement in negative triple quality. As we deliberately limit the strength of sampled negative triples by N_1, some unrelated negative triples may remain. A small number of weak negative triples can even help the model's training by introducing a penalty in loss that reduces over-fitting.

Relation-wise performance in WN18RR dataset: To go beyond aggregated performance values, we intend to see the prediction performance on individual relation types in the WN18RR dataset. We choose TransH as the embedding method because it is seen from Table 4 that TransH with SNS shows better prediction performance (in terms of hit@10 and hit@3 than DistMult with random sampling. We train TransH with different sampling methods and record the MRR and Hit@10 metrics for each relation type for each sampling

Table 5: A few examples of sampled negative triples by different sampling methods in the WN18RR dataset. The entity in boldface is the one resulting from entity corruption.

Positive triple	**Negative triples by SNS**
	(point,derivationally_related_form,**clinch**)
	(**carelessness**,verb_group,play)
	(worry,hypernym,**announce**)
	(**akee**,member_meronym,tree)
	(pakistan,has_part,**amazon**)
	Negative triples by NSCaching
1. (point,derivationally_related_form,arrow)	(**rio_grande**,derivationally_related_form,point)
2. (meet,verb_group,play)	(meet,verb_group,**wheeler_peak**)
3. (worry,hypernym,mind)	(worry,hypernym,**seal**)
4. (forest,member_meronym,tree)	(**disjoin**,member_meronym,tree)
5. (pakistan,has_part,tirich_mir)	(pakistan,has_part,**malevolence**)
	Negative triples by KBGAN
	(point,derivationally_related_form,**wilt**)
	(**frequency**,verb_group,play)
	(worry,hypernym,**traversal**)
	(**pan**,member_meronym,tree)
	(pakistan,has_part,**chute**)
	Negative triples by Uniform-random
	(point,derivationally_related_form,**colored**)
	(**cranium**,verb_group,play)
	(**bladder_fern**,hypernym,mind)
	(forest,member_meronym,**barrow**)
	(pakistan,has_part,**deposit**)
	Negative triples by Self-Adv
	(**roping**,derivationally_related_form,arrow)
	(meet,verb_group,**convenience_food**)
	(worry,hypernym,**behave**)
	(forest,member_meronym,**bryaceae**)
	(**theism**, has_part, tirich_mir)

method. The metrics of the 11 relations in the WN18RR dataset are shown in Table 6. We see that two relations (hypernym, derivationally_related_form) cover about 75% of the total facts in the training set. From the table, the

Table 6: Relation-wise performance in WN18RR dataset: MRR, MR and Hit@10 percentage. Number beside a relation denotes the percentage of facts covered by the relation in the test set. The best metrics for each relation are highlighted in bold face.

Relations	SNS		KBGAN		NSCaching		Random	
	MRR	hit@10	MRR	hit@10	MRR	hit@10	MRR	hit@10
hypernym(39.92%)	**0.092**	**17.92**	0.04	8.18	0.0701	14.26	0.0357	11.28
derivationally_related_form(34.27%)	**0.3971**	95.44	0.4051	94.41	0.3899	**95.53**	0.3755	73.69
instance_hypernym(3.89%)	**0.0718**	**15.57**	0.0452	12.3	0.0439	9.84	0.0461	11.48
also_see(1.79%)	**0.0978**	26.79	**0.261**	**71.43**	0.0903	21.43	0.0766	25.0
member_meronym(8.07%)	**0.023**	**6.32**	0.0218	5.93	0.0134	3.56	0.012	4.74
synset_domain_topic_of(3.64%)	0.0412	8.77	**0.0465**	**14.04**	0.0375	7.89	0.0374	7.02
has_part(5.49%)	**0.0336**	**8.14**	0.0243	**8.14**	0.0197	4.65	0.019	3.49
member_of_domain_usage(0.77%)	0.0021	0.00	0.0015	0	0.0029	0.00	**0.0093**	**4.17**
member_of_domain_region(0.83%)	**0.0046**	0.00	0.004	0.00	0.0026	0.00	0.0034	0.0
verb_group(1.24%)	**0.1998**	**69.23**	**0.3571**	**97.44**	0.1497	48.72	0.1888	61.54
similar_to(0.09%)	0.0001	0.00	**0.2407**	**66.67**	0.0002	0.00	0.0008	0.00

prediction performance scores are higher for relations with a higher number of facts. We see the best MRR and hit@10 scores for the SNS method in almost 70% and 50% of the relations, respectively, which means that our sampling method ranks the triple better than other sampling methods. The performance metrics are poor in most of the relations with a low number of facts (similar_to, member_of_domain_usage, member_of_domain_region). The embedding quality of the relations suffers as a result of a lack of adequate triples, and the prediction performance of the relations drops as a consequences. The performance scores of other sampling methods are high in some particular relations. For example, KBGAN shows remarkable performance in also_see, similar_to and verb_group relations while the performance is poor for the hypernym relation. NSCaching is the second-best option in most cases.

5.2. Rule performance

The proposed rule mining method is implemented in Python with well-known PyTorch and run on a Intel(R) Core(TM) i7-1.90GHz CPU machine. We use the embeddings learned by an embedding method with SNS sampling strategy in rule mining task. To learn embeddings, we choose DistMult embedding method

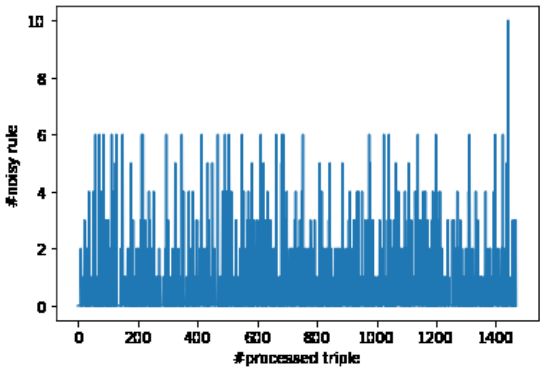


Figure 8: Identification of noisy rules for "brother" relations in the Family KG

due to its simplicity. We compare the rule-set mined based on DistMult with SNS and 'random' sampling. We set the sampling rate to 0.5, count threshold to 1, decay rate to 0.1, top-k to 5, and maximum rule length to 3 for experiments. We describe performance of our rule mining method in terms of prediction recall and quality metrics.

5.2.1. Noisy rule identification:

While the existing embedding-based rule mining methods identify noisy rules based on the quality metrics at the end of the mining process (depending on thresholds on quality metrics), our method is able to identify them early. Figure 8 shows the amount of identified noisy rules with different amounts of processed triples for 'brother' relation in Family KG. The early identification of noisy rules in our method reduces resource memory consumption as well as processing time.

5.2.2. Mining out-of closed path rule:

While existing embedding-based rule mining methods mine rules based on closed paths, our method is able to mine rules which do not follow closed path property. Below the examples of two mined rules out-of closed paths for 'aunt'

relation in Family KG.

$$niece(A, B) \wedge niece(C, B) \wedge aunt(C, D) -> aunt(A, D)$$
$$nephew(B, A) \wedge brother(B, C) -> aunt(A, C)$$

5.2.3. Prediction performance:

We compute the prediction performance of our rule mining method based on learned embeddings by our SNS and random sampling strategies. Table 7 tabulates the recall scores with their average, minimum and maximum numbers for Family, WN18RR and FB15K-237 KGs. And Table 8 tabulates the quality metrics scores with their average, minimum and maximum numbers for the same datasets. Overall, our the proposed rule mining method mines more rules when embeddings are learned DistMult with SNS than DistMult with random sampling. But, we see no significant different between the performance scores of mined rule sets. We see that the average number of mined rules per relation is highest for the densest FB15K-237 and lowest for the sparest WN18RR KGs. This is expected because the higher average node degree of a graph leads to generating more paths between pair of nodes and thus more rules to describe the relationship between the pair of nodes. The high number of rules for Family and FB15K-237 KGs give high recall scores in these two KGs. Looking at the minimum recall scores, we see the datasets except Family give a minimum scores of 0.0. Investigating the relation-wise recall, we find that the quality of learned rules for a relation with low number of training set is very low and even no rule is generated for few such relations (e.g. similar_to relation in WN18RR, /baseball/.../baseball_team_stats/season in FB15K-237). This situation affects the performance scores as well as the overall ruleset quality of a relation. In Table 8, we see the worst rule quality metrics for WN18RR. The average node degree of WN18RR is low which causes less number of paths between two entities to generate rules. This is most possible reason is that our method mines low-quality length-2 rules for WN18RR.

We try to investigate how the quality of embeddings affects the quality of mined ruleset as well as the prediction performance in WN18RR datasets. We

Table 7: Rule performance metrics: #Rules represents the total number of mined length-2 rules. The recall metric is given in the format (minimum score, average score, maximum score) which are computed over all relations in a KG. 'DistMult' is trained with two different sampling methods: SNS, random.

KGs	Random		SNS	
	#Rule	Recall (min.,avg.max.)	#Rules	Recall (min.,avg.max.)
Family	235	(0.569, 0.908, 1.0)	242	(0.5862, 0.910, 1.0)
WN18RR	69	(0.0, 0.184, 0.487)	76	(0.0, 0.185, 0.487)
FB15K-237	8490	(0.0, 0.684, 1.0)	8559	(0.0, 0.689, 1.0)

Table 8: Rule quality scores: HC and SC denote the head coverage and standard confidence metrics, respectively. The metrics are in given in the format (minimum score, average score, maximum score) which are computed over all relations in a KG. 'DistMult' is trained with two different sampling methods: SNS, random.

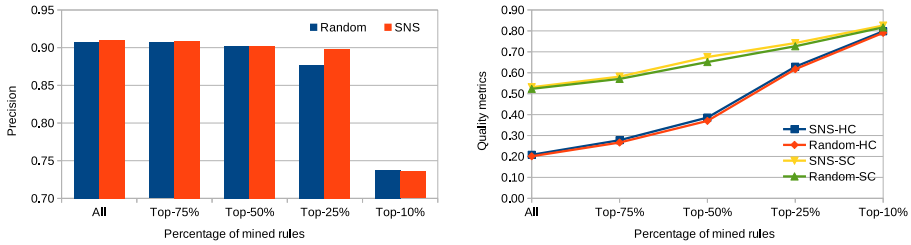
KGs	Random		SNS	
	HC(min., avg., max.)	SC(min., avg., max.)	HC(min., avg., max.)	SC(min., avg., max.)
Family	(0.135, 0.208, 0.241)	(0.325, 0.523, 0.674)	(0.119, 0.205, 0.254)	(0.285, 0.531, 0.695)
WN18RR	(0.0, 0.0337, 0.098)	(0.0, 0.0569, 0.179)	(0.0, 0.0311, 0.0869)	(0.0, 0.0689, 0.1433)
FB15K-237	(0.0, 0.176, 0.969)	(0.0, 0.196, 0.905)	(0.0, 0.175, 0.969)	(0.0, 0.199, 0.905)

choose WN18RR datasets because the overall prediction metrics and rule quality metrics in this dataset is lowest. We select the rules of length-2 mined based on embedding learned by DistMult with SNS sampling. We see that the results in Table 9 are nearly consistent with the prediction results by embedding-based method in Table 6. We see that no rule is mined for the relation similar_to because the relation has insufficient number of triples in the training set (see Table 9).

To see how the rule mining performance changes with different percentage rules, we sample all, top-75%, top-50%, top-25%, and top-10% ranked length-3 rules for each relation in Family KG and illustrate them in Figure 9. The rules for a relation are ranked in decreasing order of their support scores. As low-ranked rules are not considered, triggering rules for a few triples will be missing. As a result, it is seen in Figure 9a that the recall scores for both sampling methods drop as the number of rules drops. In contrast, as low-quality rules are removed, the average rule quality scores in Figure 9b improve.

Table 9: Relation-wise performance of mined length-2 rules in the WN18RR dataset: The embeddings are learned by DistMult with SNS sampling. The parameters for rule mining methods are set to default values except the maximum rule length is set to 2. Relations with recall scores above the average(0.184) are marked in boldface.

Relations	#Rule	Recall	HC	SC
hypernym	9	0.089	0.045	0.009
derivationally_related_form	12	0.224	0.087	0.019
instance_hypernym	5	0.197	0.017	0.028
also_see	7	0.214	0.048	0.040
member_meronym	6	0.099	0.013	0.016
synset_domain_topic_of	15	0.447	0.143	0.032
has_part	10	0.238	0.057	0.023
member_of_domain_usage	2	0.042	0.131	0.043
member_of_domain_region	2	0.00	0.142	0.015
verb_group	8	0.487	0.006	0.087
similar_to	0	0.000	0.00	0.00



(a) Prediction performance

(b) Rule quality

Figure 9: Change in prediction performance and quality with different percentages of mined rules of length-2 in the Family KG

In Table 10, we tabulate the top-5 ranked mined rules for the 'Father' relation in Family KG. Looking at the head coverage and standard confidence scores, we find that our method is able to compute low/better ranks for high-quality rules.

Table 10: Top-5 rules of length-3 for the 'father' relationship: ranks are computed in decreasing order of rule support values.

Rank	Rules	HC	SC
1	$husband(A,B) \wedge mother(B,C) -> father(A,C)$	0.765	0.882
2	$father(A,B) \wedge brother(B,C) -> father(A,C)$	0.599	0.407
3	$father(A,B) \wedge son(B,C) \wedge mother(C,D) -> father(A,D)$	0.550	0.298
4	$husband(A,B) \wedge mother(B,C) \wedge brother(C,D) -> father(A,D)$	0.546	0.386
5	$father(A,B) \wedge sister(B,C) -> father(A,C)$	0.524	0.420

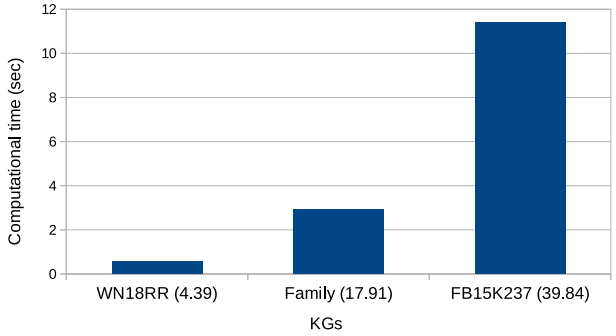


Figure 10: Graph complexity vs. computational time in different KGs for mining length-2 rules: The number on the horizontal axis in () denotes the average node degree. The computational time in the vertical axis is the average computational time over all relations in a KG.

5.2.4. Scalability:

To assess the behaviour of our rule mining method for different KGs with different scales of connectivity, we illustrate the computational time for mining length-2 rules for the Family, WN18RR and FB15K-237 KGs in Figure 10. Among the three KGs, WN18RR is the most sparse (average node degree<5.0) whereas the other two are denser. As expected, we see that the average rule mining time grows as node degree increases because the number of paths between a pair of nodes is usually higher in a dense graph.

5.2.5. Parameter sensitivity:

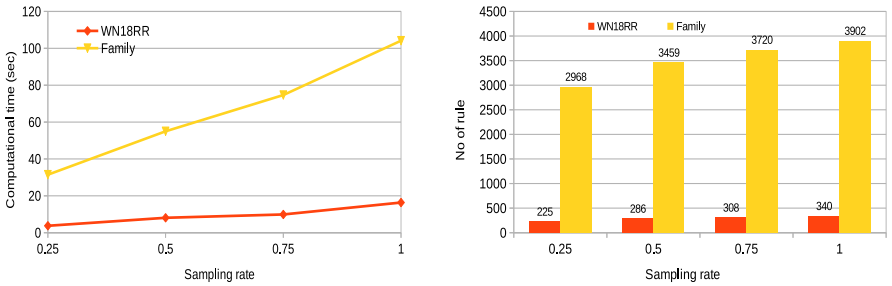
We assess parameter's sensitivity of our rule mining method, we implement the method on the WN18RR and Family KGs to learn length-3 rules. In learning embedding, we choose our SNS sampling method. There are four parameters

in our rule mining method and they are sampling rate, top-k paths, decay rate, and rule length.

Sampling rate: To assess how the rule mining method behaves, we illustrate computational time and performance metrics (recall, HC, SC) for different sampling rates in Figure 11. As expected, the computational time increases with increasing the sampling rate in Figure 11a for both datasets. The chance of generating more rules for a high sampling rate is high, and consequently the recall score is improved in Figure 11c. The rule mining time grows faster for the dense Family KG than the sparse WN18RR as the number of paths grows more quickly for the denser Family KG. The recall scores for WN18RR improved as we have more rules for higher sampling rate (Figure 11b). On the other hand, the recall scores for Family KG remain nearly very close for different sampling rate. The most suitable reason is that the high-quality embeddings help the proposed rule mining method in extracting high-quality rules frequently, even with low sampling rate. More rules including low-quality rules are mined for higher sampling rate which causes the overall quality metrics to be dropped for higher sampling rate. This situation is seen for both of the KGs. We see the sampling rate has impact on computational time as well as prediction and quality metrics. Undoubtedly, choosing a good sampling rate is a challenging but important task.

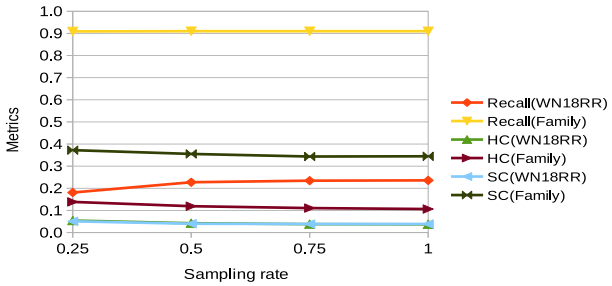
Top-k: We evaluate the total number of mined rules and the recall, HC, and SC of rules for different top-k values to describe the sensitivity of the "top-k" parameter in our rule mining method. Figure 12a shows that the number of mined rules increases with increasing top-k for WN18RR and Family KGs. Consideration of more paths between a pair of nodes, including high-quality and low-quality paths, is the most possible reason behind this rising trend in the number of rules. This assumption is further supported by the quality metrics in Figure 12b where both HC and SC drop as the top-k increases for both KGs.

Decay rate: To assess the sensitivity of our rule mining method to the "decay rate" parameter, we evaluate the total number of mined rules, recall, and quality metrics of our rule mining method for different decay rates. As the



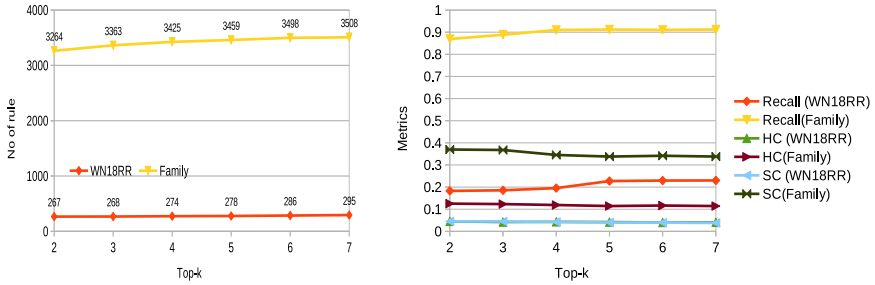
(a) Sampling rate vs time

(b) Sampling rate vs #Rule



(c) Sampling rate vs metrics

Figure 11: Sensitivity to sampling rate in the WN18RR KG

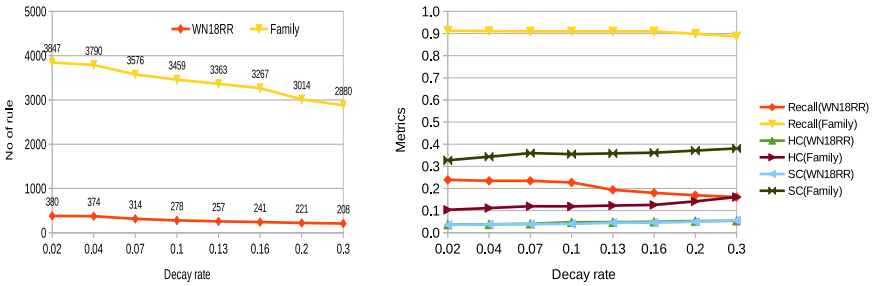


(a) Top-k vs #Rule

(b) Top-k vs metrics

Figure 12: Sensitivity to top-k: The default parameters setting are used (i.e. max. length=3, decay rate=0.1, sampling rate=0.5) except top-k varies from 2 to 7 with an interval 1.

decay rate increases, the energy of an unobserved rule drops faster, and as a result, it lives for a low period in the system. For this reason, we see that the total number of mined rules in Figure 13a falls and consequently the recall scores dropped for higher decay rate. In contrast, the rule quality scores increases for



(a) Decay rate vs #rule

(b) Decay rate vs metrics

Figure 13: Sensitivity to decay rate: The default parameters setting are used (i.e. max. rule length=3, sampling rate=0.5, top-k=5) except decay rates are from {0.02, 0.04, 0.07, 0.1, 0.13, 0.16, 0.2, 0.3}.

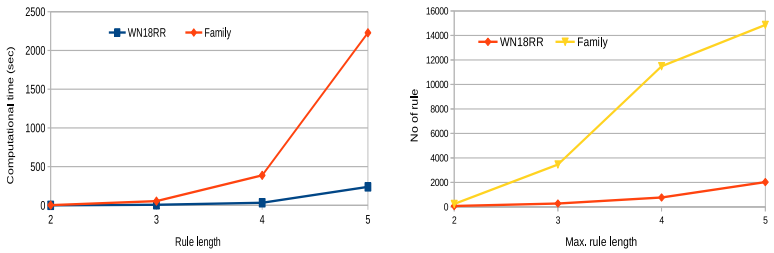
both KGs as only high-quality rules are extracted those happen very frequently for higher decay rate.

Maximum rule-length: Finally, we assess the sensitivity of the proposed rule mining method to "maximum rule length" parameter in terms of computational time, total number of rules, recall and quality metrics. The number of paths between a pair of nodes in a graph increases quickly as the maximum allowable path length increases. This results in rising computational time exponentially for mining rules as seen in Fig 14a. As the number of possible paths between a pair of nodes increases with increasing path length, the total number of rules also increases for higher rule length for both KGs in Fig 14b. As more rules are mined, the recall increases and quality metrics drops in Figure 14c.

5.3. Illustration of Explainable link prediction in FB15K-237

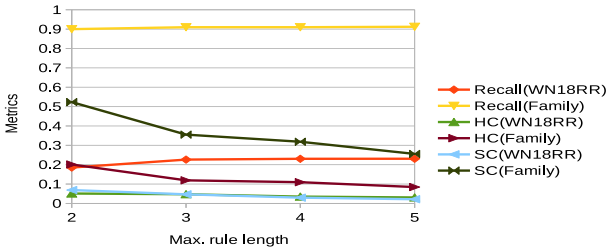
In this section, we take advantage from mined rules to explain the link prediction by embedding-based method. We consider the test triples which have hit@10 score 1 as correctly predicted triples. For a correctly predicted triple, we find the most suitable path that hits a rule from mined rule set. We give preference to shorter paths in case of occurring multiple paths.

We illustrate the link prediction explanation based on the mined length-2 rules in FB15K-237 KG. Note that the entities in this KG come with Free-



(a) Length vs time

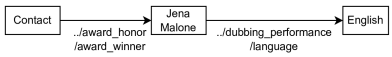
(b) Length vs #rule



(c) Length vs metrics

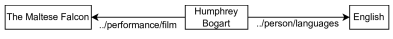
Figure 14: Sensitivity to maximum rule length: The default parameters setting are used (i.e. decay rate=0.1, sampling rate=0.5, top-k=5) except rule length varies from 2 to 5 with an interval 1.

base identifier and Freebase KG is no more maintained. We map the Freebase identifier to Wikidata identifier using Freebase data dump [29] and then the description of the entities are extracted from Wikidata using the Wikidata identifier. For illustration, we consider few triples with the '/film/film/language' relation from FB15K-237 KG which are correctly predicted by embedding-based method. The heads of these triples are film names and the tails are languages of films. For example, (Contact,../film/language, English) is interpreted as "The language of the film 'Contact' is 'English'". Fig 15 shows four such triples with their corresponding paths in the KG and in identified rules. The examples explain which paths help the target triple to be true. For instance, the prediction of the triple (Contact,../film/language,English) (in Fig 15a) could be explained as "'Jena Malone' is awarded for the 'Contact' film and the dubbing language of 'Jena Malone' is English." These two facts trigger the target triple, (Contact,../film/language, English) to be predicted as true.



(a) (Contact,../film/language,English)

(b) (Salmon Fishing in the Yemen, ../film/language,English)



(c) (The Maltese Falcon, ../film/language, English)

(d) (Space Cowboys, ../film/language, English)

Figure 15: A few examples of correctly predicted triples in FB15K-237 KG by embedding-based method with their triggered paths identified by mined rules. The relation in the KG is longer in length, describing its hierarchy. Due to space constraints, we only include the lower part of the hierarchy and replace the higher parts with dots in describing a relation.

6. Conclusion and Perspectives

In this paper, we propose a simple and efficient negative sampling method for knowledge graph embedding. The method is general and can be plugged to any knowledge graph embedding method. The method is able to generate high-quality negative triples and takes low computational time and memory while anticipating the 'vanishing-gradient' problem. Experimentally, we evaluate our method on six knowledge graph datasets for link prediction task and also describe its parameter sensitivity. The results show that the proposed SNS sampling brings consistent improvements in prediction performance. We also propose a new embedding-based rule mining method that is memory efficient and able to mine a diverse set of rule patterns. We take advantage of the rule mining method to explain the link prediction in order to minimize the 'explainability' limitation of embedding-based link prediction methods to some extent. To explain link prediction, the proposed rule mining method may be used in conjunction with any knowledge graph embedding method.

The poor performance of the studied methods on YAGO3-10 leaves the future work to explore other embedding models in the literature on this dataset. Another perspective of this study is to plug SNS to a recently published approach, TransO, leveraging ontology information for learning better embeddings [30]. Such improved embeddings may also be helpful for learning better rules. As

SNS uses only entity embeddings for sampling negative examples, it should also be usable in dynamic graph embedding approaches [31]. Implementing both our methods in a distributed environment to improve computational efficiency is another promising research direction. We also leave designing a good strategy for sampling the triples for our proposed rule mining method to future exploration, which can help to learn high quality rules with low sampling rate.

References

[1] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic Web 8 (3) (2017) 489–508.

[2] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, W. Zhang, Knowledge vault: A web-scale approach to probabilistic knowledge fusion, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 601–610.

[3] L. Cai, W. Y. Wang, KBGAN: Adversarial learning for knowledge graph embeddings, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2018, pp. 1470–1480.

[4] Y. Zhang, Q. Yao, Y. Shao, L. Chen, Nscaching: simple and efficient negative sampling for knowledge graph embedding, in: 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, 2019, pp. 614–625.

[5] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2, 2013, pp. 2787–2795.

[6] K. Hu, H. Liu, T. Hao, A knowledge selective adversarial network for link prediction in knowledge graph, in: CCF International Conference on Natu-

ral Language Processing and Chinese Computing, Springer, 2019, pp. 171–183.

[7] P. Wang, S. Li, R. Pan, Incorporating GAN for negative sampling in knowledge representation learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, 2018.

[8] 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, 2018.

[9] K. Ahrabian, A. Feizi, Y. Salehi, W. L. Hamilton, A. J. Bose, Structure aware negative sampling in knowledge graphs, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 6093–6101.

[10] M. K. Islam, S. Aridhi, M. Smaïl-Tabbone, Simple negative sampling for link prediction in knowledge graphs, in: International Conference on Complex Networks and Their Applications, Springer, 2021, pp. 549–562.

[11] J. Chen, B. Xin, Z. Peng, L. Dou, J. Zhang, Optimal contraction theorem for exploration–exploitation tradeoff in search and optimization, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 39 (3) (2009) 680–691.

[12] Q. Wang, Z. Mao, B. Wang, L. Guo, Knowledge graph embedding: A survey of approaches and applications, IEEE Transactions on Knowledge and Data Engineering 29 (12) (2017) 2724–2743.

[13] M. Nickel, K. Murphy, V. Tresp, E. Gabrilovich, A review of relational machine learning for knowledge graphs, Proceedings of the IEEE 104 (1) (2015) 11–33.

[14] S. Muggleton, Inverse entailment and progol, New Generation Computing 13 (3) (1995) 245–286.

[15] L. Dehaspe, H. Toivonen, Discovery of frequent datalog patterns, Data Mining and knowledge discovery 3 (1) (1999) 7–36.

[16] R. Srikant, R. Agrawal, Mining quantitative association rules in large relational tables, in: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 1996, pp. 1–12.

[17] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 1993, pp. 207–216.

[18] R. T. Ng, L. V. Lakshmanan, J. Han, A. Pang, Exploratory mining and pruning optimizations of constrained associations rules, ACM Sigmod Record 27 (2) (1998) 13–24.

[19] B. Yang, S. W.-t. Yih, X. He, J. Gao, L. Deng, Embedding entities and relations for learning and inference in knowledge bases, in: International Conference on Learning Representations, 2015.

[20] P. G. Omran, K. Wang, Z. Wang, An embedding-based approach to rule learning in knowledge graphs, IEEE Transactions on Knowledge and Data Engineering 33 (4) (2021) 1348–1359.

[21] Z. Wang, J. Zhang, J. Feng, Z. Chen, Knowledge graph embedding by translating on hyperplanes, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 28, 2014.

[22] S. Kok, P. Domingos, Statistical predicate invention, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 433–440.

[23] A. Sadeghian, M. Armandpour, P. Ding, D. Z. Wang, Drum: end-to-end differentiable rule mining on knowledge graphs, in: Proceedings of the 33rd International Conference on Neural Information Processing Systems, 2019, pp. 15347–15357.

[24] M. Wang, L. Qiu, X. Wang, A survey on knowledge graph embeddings for link prediction, Symmetry 13 (3) (2021) 485.

[25] A. Rossi, A. Matinata, Knowledge graph embeddings: Are relation-learning models learning relations?, in: EDBT/ICDT Workshops, 2020.

[26] G. Ji, S. He, L. Xu, K. Liu, J. Zhao, Knowledge graph embedding via dynamic mapping matrix, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (volume 1: Long papers), 2015, pp. 687–696.

[27] V. T. Ho, D. Stepanova, M. H. Gad-Elrab, E. Kharlamov, G. Weikum, Rule learning from knowledge graphs guided by embedding models, in: International Semantic Web Conference, Springer, 2018, pp. 72–90.

[28] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representation, 2015.

[29] Google, Freebase data dumps, https://developers.google.com/freebase/data (2022).

[30] Z. Li, X. Liu, X. Wang, P. Liu, Y. Shen, Transo: a knowledge-driven representation learning method with ontology information constraints, World Wide Web (2022) 1–23.

[31] G. Xue, M. Zhong, J. Li, J. Chen, C. Zhai, R. Kong, Dynamic network embedding survey, Neurocomputing 472 (2022) 212–223.