Differentiable Learning of Rules with Constants in Knowledge Graph

Anonymous ACL submission

Abstract

Knowledge reasoning, helping overcome the incompleteness issue of knowledge graph(KG), significantly contributes to the development of large KG, which consists of relations and constants. Rule mining studies the problem of capturing interpretable patterns over KG, which is one of the key tasks of knowledge reasoning. However, previous works mainly focus on the combination of different relations, and are limited for ignoring the importance of constants. In this paper, we propose that constants should be considered in rule mining process, and introduce an Elegant Differentiable rUle learning with Constant mEthod (**EduCe**). Based on soft constant operator and dynamic weight, the model we proposed can mine more diverse and accurate logical rules while controlling the number of parameters, which is also a great challenge to this problem. Experiment results on several benchmark datasets demonstrate the effectiveness and accuracy of our approach.

1 Introduction

Vast amounts of knowledge based on web about abstract and real-world is always a major component of Artificial Intelligence (AI). One way to represent knowledge is Knowledge Graph (KG), and there are well known KGs such as Wordnet (Miller, 1995) and Freebase (Bollacker et al., 2008) have been built. Such KGs represent facts as a graph of constants(*e.g., iPhone, Apple*) connected by relations(*e.g., brandIs*), which could be formally represented as a set of binary grounded atoms, called triplets or facts, such as *brandIs(iPhone, Apple)*.

Due to the incompleteness of KGs, many methods have been proposed to KG completion including knowledge graph embedding(KGE) (Wang et al., 2017; Bordes et al., 2013; Lin et al., 2015; Yang et al., 2015; Trouillon et al., 2016; Dettmers et al., 2018; Sun et al., 2019), graph neural networks (Schlichtkrull et al., 2018; Vashishth et al., 2020; Nathani et al., 2019; Bansal et al., 2019) and rule learning (Meilicke et al., 2019; Ortona et al., 2018a; Chen et al., 2016; Galárraga et al., 2013). Compared to deep learning approaches like KGE, rule learning is preferred due to its interpretability and robustness in transfer tasks. To mine the structure and confidence of rules at the same time in a fast way, differentiable rule learning methods (Yang et al., 2017; Sadeghian et al., 2019) are introduced and attract many research interests in recent years.

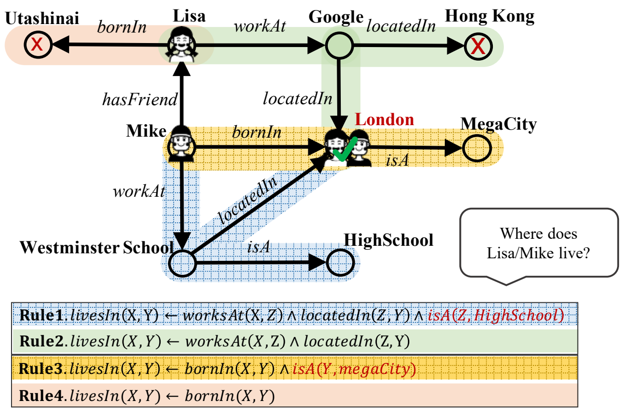

Figure 1: Examples of rules with and without constant.

Existing work such as Neural-LP (Yang et al., 2017) and DRUM (Sadeghian et al., 2019) learn to sequentially compose the primitive operations which are inspired by TensoLog (Cohen, 2016) with gradient-based optimization. At each stage of computation, they 'softly' choose a subset of TensorLog's operation with high weight, which is used to connect rule application with matrix multiplication. While their target is learning **chain-like** logical rules such as *livesIn(X, Y)* $\leftarrow$ *worksAt(X, Z)* $\wedge$ *locatedIn(Z, Y)*, and the model only focuses on choosing suitable parameters of every relation for each step.

But suppose we add another limitation factor to the above rule and make it *livesIn(X, Y)* $\leftarrow$ *worksAt(X, Z)* $\wedge$ *locatedIn(Z, Y)* $\wedge$ *isA(Z, HighSchool)*.

Obviously, the confidence is higher than the first rule, since a high school is commonly located in one city, while it is not the case for many large companies because they usually have multiple offices located in different cities, such as Google, and usually there is only one city that a person lives in. The atom *isA(Z, HighSchool)* is called a constant atom which has one variable and a constant (*HighSchool*), and the rules with such atoms are *rules with constants*. Our goal in this paper is to enable differentiable learning of rules with constants in knowledge graphs, to facilitate higher completion results and more accurate rule learning.

However, ensuring efficiency of this problem is difficult. On the one hand, a KG usually contains hundreds or even thousands of times as many constants than relations, which makes the search space for constant atoms much larger than variable atoms. On the other hand, not only one constant atom is possibly added in each step, which also leads to higher time complexity.

In this paper, we propose a differentiable framework named EduCe that can mine logical rules with constants. In EduCe, we define a relevant operator to select constants, a 'soft' way to use it, and dynamic weight mechanism to reduce the amount of parameters.

Experimentally, we apply EduCe to several knowledge graph datasets, and evaluate the capability of EduCe on both link prediction and rule mining tasks. The results show that EduCe is able to recover rules containing constants and yield more accurate prediction results compared to previous differentiable rule learning methods, and even some embedding methods. At the same time, the results also show that rules with constants usually have higher quality.

Thus our contributions are as follows:

- We draw attention to expanding the diversity of target rules for differentiable rule learning method and emphasize the importance of constants to rule.
- We propose EduCe, a new end-to-end differentiable rule learning method mining rules with constants.
- We experimentally demonstrate that EduCe outperforms existing differentiable rule learning methods, and even some embedding methods on link prediction task and successfully outputs high quality symbolic rules with constants.

2 Related work

2.1 Symbolic-based rule learning and reasoning

The problem of learning collection of relational rules is a type of statistical relational learning (Koller et al., 2007), and it can also be called inductive logic programming (ILP) (Muggleton, 1995) when the learning process involves proposing new logical rules. Although ILP methods can learn from relational data, most methods in this field require negative examples and can't handle modern large knowledge graph.

AMIE (Galárraga et al., 2013) concentrates on association rule mining following two steps. The first step is rule extending, which extends candidate rules by several kinds of operations. The second step is rule pruning according to the predefined evaluation metrics like confidence. AMIE+ (Galárraga et al., 2015) revises the rule extending process and improves evaluation method. They suffer from the predefined metrics and discrete counting.

Rudik (Ortona et al., 2018b) mine positive and negative rules in knowledge graph, while the former class infers new facts in KG, and the latter class is crucial for other tasks, such as detecting erroneous triples. Anyburl (Meilicke et al., 2019) propose an efficient way to mine rules, but the rule it mined is hard to transfer to other KG.

2.2 Neural-based rule learning and reasoning

A common neural-based reasoning method for KG is Knowledge Graph Embeddings (KGEs) (Wang et al., 2017), which has been proved to be effective for KGC. These methods embed entities and relations into vectors space and measure the true value of triplets via calculation in vector space. Most KGEs like TransE (Bordes et al., 2013), TransD (Ji et al., 2015), TransH (Wang et al., 2014) and DistMult (Yang et al., 2015), concentrate on encoding the true value of triplets constructed with two entities and one relation. And KR-EAR (Lin et al., 2016) propose to distinguish attributes and relations in KG since attributes and relations exhibit rather distinct characteristics, like entity set size. It also inspires us to learn rules with constants because attributes are more likely to participate in partially grounded atoms in rules. Some of the KGEs such as DistMult (Yang et al., 2015) are also used for rule learning based on well-trained relation embeddings, while their performance is limited by the huge search space as the incremental of the rule length.

Some neural-based methods such as KALE (Guo et al., 2016) and RUGE (Shu et al., 2017), learn the entity and relation embeddings not only based on triplets observed in KGs but also triplets inferred from rules that are learned from symbolic-based rule learning methods such as AMIE (Galárraga et al., 2013). They benefit from symbolic-based rule reasoning while they can't conduct rule learning. Thus IterE (Zhang et al., 2019) learns rules based on updated embedding at each iteration and injects new facts inferred by these rules into KGE.

More recently, end-to-end differentiable rule learning methods based on TensorLog (Cohen, 2016) are proposed. Neural-LP (Yang et al., 2017) is the first differentiable rule learning method aiming at learning probabilistic chain-like logic rules with learning parameters and structure of rules simultaneously with the basic idea that expressing the logical relationships between two entities by matrix operations. Extensions based on Neural-LP like DRUM (Sadeghian et al., 2019) are also proposed. To extend the diversity of target rules, Neural Logic Inductive Learning (NLIL) (Yang and Song, 2019) tackles the non-chain-like rules by incorporating a primitive statement. Neural-Num-LP (Wang et al., 2019) extends Neural-LP to learn the numerical rules, which is a great inspiration for fully understanding the possible reasoning patterns.

3 Problem Formulation

Knowledge Graph $\mathcal{G}$ is composed by a set of grounded atoms like $\{r(e_1, e_2)|r \in \mathcal{R}, e_1, \in \mathcal{E}, e_2 \in \mathcal{E}\}$ where $\mathcal{E}$ is a countable set of constants, which is also called entities, and $\mathcal{R}$ is a set of binary relations, respectively.

Rule is in the form of $head \leftarrow body$, where the $head$ of rule is an atom and the $body$ of rule is a conjunction of atoms. Each atom is defined as over $\mathcal{R} \cup \mathcal{E} \cup \mathcal{X}$, where $\mathcal{X}$ is a countable set of variables. Based on this, we define four kinds of atoms, $r\left(e_1, e_2\right)$, $r\left(X, e\right)$, $r\left(e, X\right)$ and $r\left(X, Y\right)$, where upper-case letters are variables $\{X, Y\} \in \mathcal{X}$, and lower-case letters $\{e, e_1, e_2\} \in \mathcal{E}$ are constants. The first kind is also called fact that is barely used in rules, and we name the second and third kinds **constant atom** and the last one **variable atom**.

Rules without constants is in the following form:

$$r\left(X,\ Y\right) \leftarrow r_1\left(X,\ Z_1\right) \wedge ... \wedge r_T\left(Z_{T-1},\ Y\right)$$

An example of chain-like rule is

$$livesIn(X, Y) \leftarrow worksAt(X, Z) \wedge locatedIn(Z, Y)$$

This type of rule is defined over only variable atoms without taking constant atoms into consideration.

Rules with constants refer to rules whose body is composed of variable atoms and constant atoms, which is in the following form:

$$r\left(X,\ Y\right) \leftarrow \underbrace{r_1\left(X,\ Z_1\right) \wedge \underbrace{\left(\bigwedge_{i=1}^{n_1} r_1^i\left(Z_1,\ e_1^i\right)\right)}_{\text{constant atoms for the 1st step}}}_{\text{the 1st step}} \wedge ...$$

$$\underbrace{\wedge\, r_T\left(Z_{T-1},\ Y\right) \wedge \underbrace{\left(\bigwedge_{i=1}^{n_T} r_T^i\left(Z_T,\ e_T^i\right)\right)}_{\text{constant atoms for the } T\text{th step}}}_{\text{the } T\text{th step}}$$

where $e_t^i \in \mathcal{E}$ are entities formed constant atoms. T is the length of chain in the rule and n_i is the number of constant atoms related to each step of the chain. More specifically, if $T = 2$, $n_1 = 2$ and $n_2 = 1$, a rule with constants is like Figure 2(a).

Learning rules with constants is not easy since the quantity of parameters to be learned could be extremely large. In particular, for each step, there are $2|\mathcal{R}|$ possible relations to choose with automatically inverse relations considered, and the number of candidate constants to be chosen in constant atoms is $|\mathcal{E}|$. Thus intuitively, the time complexity of learning rules with constants is

$$O\left(|\mathcal{R}|^T \times (|\mathcal{R}||\mathcal{E}|)^T\right) \qquad (1)$$

The first part in Equation 1 indicates selecting a suitable relation to expand the path, and the second part means choosing one or several constant atoms for each step. As we can see, the time complexity of this problem is enormous, which is a great challenge to this problem.

4 Method

To enable differentiable rule learning of chain-like rules, Neural-LP has reduced the first part in Equation 1 to $O(T|\mathcal{R}|)$. What we consider here is reducing the complexity of constant atom selection,

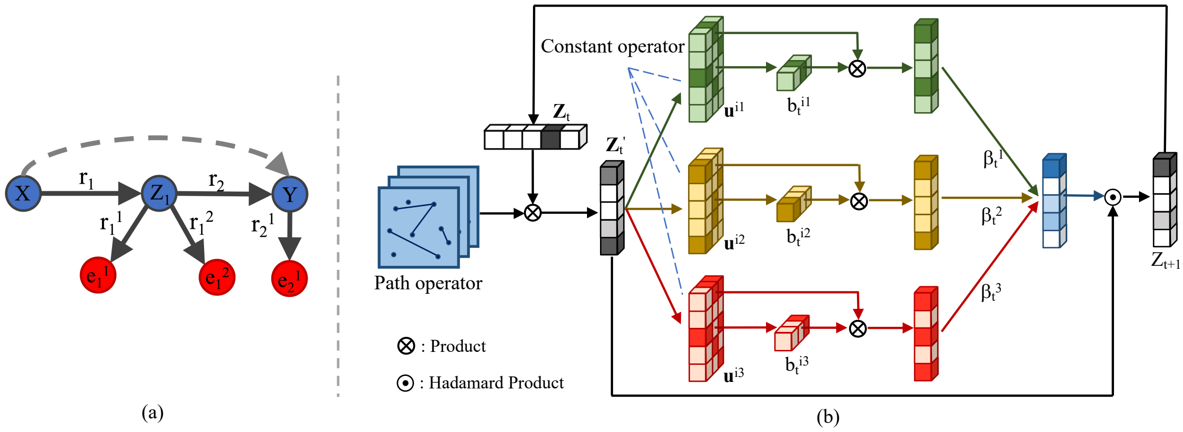

Figure 2: The form of rule with constants and Reasoning process of EduCe.

which is shown as the second part in Equation 1, to facilitate constant rule learning. By using constant operator and dynamic weight in EduCe framework, we successfully reduced the complexity to $O(2T|\mathcal{R}|)$. Next, we will introduce the details of EduCe, including the operators we define, model architecture and training objectives, and describe how to decode symbolic rules based on well-trained EduCe.

4.1 Operators

Given a KG $\mathcal{G} = \{r(e_1, e_2)|r \in \mathcal{R}, e_1, \in \mathcal{E}, e_2 \in \mathcal{E}\}$, we firstly represent each entity e_i as an one-hot vector $\mathbf{v}^{e_i} \in \{0,1\}^{|\mathcal{E}|}$, and represent each relation r_k as an adjacent matrix $\mathbf{M}^{r_k} \in \{0,1\}^{|\mathcal{E}|\times|\mathcal{E}|}$ where $\mathbf{M}^{r_k}_{ij} = 1$ if $r_k(e_i, e_j) \in \mathcal{G}$, else $\mathbf{M}^{r_k}_{ij} = 0$.

For an inference query $r(e_1, ?)$ to predict the tail entity with head entity e_1 and relation r, in order to conduct inference process, two kinds of operations are necessary. One is path operator which maps one entity to other entities following a certain relation, the other one is constant operator selecting entities that satisfy a specific constant atom.

Path Operator $\mathcal{O}^P$ is already defined by `TensorLog` and applied in previous works like (Yang et al., 2017):

$$\mathcal{O}_P(\mathbf{v}^i, \mathbf{M}^{r_k}) = \mathbf{v}^i\mathbf{M}^{r_k} \tag{2}$$

Via recursively applying path operators, path queries could be answered by expanding path with path operator.

Constant Operator $\mathcal{O}^C$ is an operator we define, with variables vector v^i, a constant vector v^c and a relation matrix $\mathbf{M}^{r_k}$ as input and output variables satisfying given constant. For constant atom (X, r_k, c) where $X \in \mathcal{X}, r \in \mathcal{R}$ and $c \in \mathcal{E}$, with $X = e_i$, the constant operator could be framed as

$$\mathcal{O}^C(\mathbf{v}^i, \mathbf{M}^{r_k}, \mathbf{v}^c) = \mathbf{v}^i \circ \underbrace{(\mathbf{v}^c(\mathbf{M}^{r_k})^\top)}_{\mathbf{u}^{ck}} \tag{3}$$

where $\circ$ stands for Hadamard product. As we can see, a constant operator is determined by a relation and a constant. $\mathbf{v}^c(\mathbf{M}^{r_k})^\top$ could be computed in advance, and we can rewrite it as $\mathbf{u}^{ck}$.

4.2 EduCe

With the two operators we mentioned before, a naive framework, **Educe**$_N$(naive version EduCe), can be proposed that softly uses path and constant operator, and rules can be learned in theory. Specifically, suppose target rules are with T steps, Educe$_N$ is defined as a recurrent architecture with the following function for step t:

$$\mathbf{z}_t = \mathcal{O}^C_\beta(\mathcal{O}^P_\alpha(\mathbf{z}_{t-1})) \tag{4}$$

where $\mathcal{O}^P_\alpha$ is a soft function of path operator $\mathcal{O}^P$ with α as parameters and $\mathcal{O}^C_\beta$ is a soft function of constant operator $\mathcal{O}^C$ with β as parameters.

Soft path operator $\mathcal{O}^P_\alpha$ is defined the same as DRUM and Neural-LP which softly choose relations along the path in each step:

$$\mathbf{z}'_t = \mathbf{z}_{t-1} \times \sum_{i=1}^{|\mathcal{R}|+1} \alpha^i_t \mathbf{M}^{r_i} \tag{5}$$

where $\mathbf{M}^{r_i}$ is the adjacent matrix of relation $r_i \in \mathcal{R}$ as introduced before. Consider that we want to mine rules with maximum length of T, we add a new relation $r_{|\mathcal{R}|+1}$ with an identity adjacency matrix. α^i_t is a scalar representing the possibility of the relation r_i as the relation in rules at step t, and

$\mathbf{z}'_t \in \mathbb{R}^{|\mathcal{E}|\times 1}$ could be interpreted as entities that could be reached after soft path operator in step t.

Soft constant operator $\mathcal{O}^C_\beta$ is defined with $\mathbf{z}'_t$ as input to choose necessary constants in step t, which could be written as

$$\mathbf{z}''_t = \sum_{k=0}^{|\mathcal{R}|}\sum_{i=1}^{|\mathcal{E}|} \beta_t^{ik} \times \mathbf{z}'_t \circ \mathbf{u}^{ik} \tag{6}$$

where β_t^{ik} is the weight for constant operator $\mathcal{O}^C(\mathbf{z}'_t, \mathbf{u}^{ik})$ that should be learned in step t. Note that maybe no constant should be considered for a step, we also add a special relation r_0 with $\mathbf{u}^{i0}$(i=1) which is all-one vector.

Using soft path operator to learn chain-like rules can successfully reduce the parameters from $O\left(|\mathcal{R}|^T\right)$ to $O(T|\mathcal{R}|)$. Similarly, the complexity in Equation 1 can be reduced to $O(T|\mathcal{R}|+T|\mathcal{R}||\mathcal{E}|)$ with soft path operator and soft constant operator. This is what Educe$_N$ does.

Unfortunately, this naive way is not applicable because the number of constants in $\mathcal{G}$ is large. Although we have greatly reduced the time complexity, the number of learnable parameters of Educe$_N$ is still huge for direct optimization because of the limited number of data samples. Considering all types, the number of β_t, i.e. $|\mathcal{R}| \times |\mathcal{E}|$, is much larger than $|\mathcal{R}|$, making the problem more difficult.

The next thing we need to consider is to further reduce the amount of β_t and thus we propose EduCe. As we mentioned before, a constant operator is determined by a relation and a constant. The parameters β_t indicate relation selection and constant selection in a step, so Equation 6 can be redefined as Equation 7:

$$\mathbf{z}''_t = \sum_{k=1}^{|\mathcal{R}|} \beta_t^k \sum_{i=1}^{|\mathcal{E}|} b_t^{ik} \times \mathbf{z}'_t \circ \mathbf{u}^{ik} \tag{7}$$

where β_t^{ik} is replaced by β_t^k and b_t^{ik}, which indicate selecting relations and constants.

Now instead of regarding b_t^{ik} as parameter to be learned, we propose dynamic weight, which computes b_t^{ik} as below by utilizing intermediate inference results:

$$b_t^{ik} = \mathbf{z}'_t \cdot \mathbf{u}^{ik} \tag{8}$$

In this case, b_t^{ik} indicates the relevance between $\mathbf{z}'_t$ and constant e_i via relation r_k. It will be larger if they are connected and smaller if not. Since it's computed from the reasoning process, we name it dynamic weight. The key idea behind this is to use the intermediate entities to select constants, and then the constants can be used to select entities.

Thus, Equation 6 can be rewrite as

$$\mathbf{z}_{t+1} = \mathbf{z}'_t \circ \sum_{k=0}^{|\mathcal{R}|} \beta_t^k Scale(\sum_{i=1}^{|\mathcal{E}_k|} \mathbf{z}'_t \cdot \mathbf{u}^{ik} \times \mathbf{u}^{ik}) \tag{9}$$

where $Scale(\mathbf{x})$ means scaling vector $\mathbf{x}$ to range $(0, 1)$ by dividing by the maximum value of $\mathbf{x}$, and $\mathcal{E}_k$ stands for the tail entity of r_k.

Generating weights α and β. We estimate weights α and β via a BiLSTM function $\mathcal{F}$ and fully-connected layers. Given the embedding of head relation r, denoted as $\mathbf{r}$ which is initialized randomly, BiLSTM aims to sequentially generate α_t and β_t for step t.

$$\mathbf{h}_0, \mathbf{h}'_{2T} = \mathcal{F}(\mathbf{r}) \tag{10}$$

$$\mathbf{h}_t = \mathcal{F}(\mathbf{r}, \mathbf{h}_{t-1},), t > 0 \tag{11}$$

$$\mathbf{h}'_{2T-t-1} = \mathcal{F}(\mathbf{r}, \mathbf{h}'_{2T-t}), t > 0 \tag{12}$$

With the vectors output from $BiLSTM$, two types of fully-connected layers are applied to generate α and β :

$$\alpha_t = \mathcal{S}(f_a(\mathbf{h}_t + \mathbf{h}'_t)), t = 1, 3, ..., 2T-1 \tag{13}$$

$$\beta_t = \mathcal{S}(f_b(\mathbf{h}_t + \mathbf{h}'_t)), t = 2, 4, ..., 2T \tag{14}$$

where f_a and f_b are fully connected neural network, and $\mathcal{S}$ is a softmax function.

The final score of a target triplet (h, r, t) is the similarity between the predicted vector $\mathbf{z}_T$ and the answer vector $\mathbf{v}^t$

$$\phi(t|h, r) = \mathbf{v}^t \cdot log[\mathbf{z}_T, \gamma]_+ \tag{15}$$

where $[\mathbf{x}, \gamma]_+$ denotes the maximum value between each element of $\mathbf{x}$ and γ. The objective of EduCe is

$$\min \left(-\sum_{(h,r,t)\in\mathcal{G}} \phi(t|h, r)\right) \tag{16}$$

4.3 Rule Decoder of EduCe

To decode symbolic rules from the neural network of EduCe, we propose a rule parsing algorithm using the parameters learned from training process. The basic idea is to select appropriate relations and constants with high weight. Specifically, we recover possible rules for each triplet via parameters α, β, b and output symbolic rules with high confidence for each query. The detailed procedure is shown in Algorithm 1.

Algorithm 1 Decode symbolic logical rules from EduCe

Input: path operator attention $\{\alpha_t | t = 1,2...T\}$, constant operator attention$\{\beta_t | t = 1,2...T\}$, dynamic weight$\{b_t | t = 1,2...T\}$

Initialize: $R = \{([P_r, P_e, P_v], w)\}$, $P_r = \emptyset$, $P_v = \emptyset$, $P_e =$ head entity, $\alpha = 1$ represents confidence

for *t=1:T* **do**
  for $r_p \in \alpha_t > thr1$ **do**
    for $([P_r, P_e, P_v], w) \in R$ **do**
      `// Expand the path if possible`
      if $P_e[-1]$ *can link other entity* n *via* r_p **then**
        Flag = False
        for $r_c \in \beta_t > thr2$ **do**
          for $con \in b_t > thr3$ **do**
            `// a new path with constant`
            $w' = w \times \alpha_t^{r_p} \times (1 + \beta_t^{r_c} \times b_t^{r_c con}])$
            if $w' > thr\_rule$ **then**
              add $([P_r + r_p, P_e + n, P_v + (r_c, con)], w')$
              Flag = True
            end
          end
        end
        if *Flag=False* **then**
          `// a new path without constant`
          $w' = w \times \alpha_t^{r_p}$
          add $([P_r + r_p, P_e + n, P_v + \emptyset], \alpha'')$
        end
      end
      Delete $([P_r, P_e, P_v], w)$ from R
    end
  end
end

The thr in this algorithm is not simply manually pre-defined, but also related to the maximum weight of this step. Because there might be several operator choices within a step, but because of the softmax function we use in equation 13, their weights might all be relatively small.

We use this algorithm for one query triplet, and the output includes all rules used in the inferring process. After all triplets are input into the decoder, most of the rules will repeat many times. For a repeated rule r, the average confidence $\sum_{i=1}^{|r|} w_i$ will be computed. Also, the number of occurrence of a rule is considered to revise confidence, to avoid overfitting rules.

5 Experiment

5.1 Datasets and Experiment Setting

Our experiments were conducted on four different datasets which are introduced as follows, and Table 1 summarizes the data statistics.

- *Constant* is a synthetic dataset. We define several different rules which are divided into several groups, and the body of each rule in the same group contains a special constant that distinguishes it from others, and the other part is the same. This only difference leads to a different head relation of the rule.
- *Family-gender* contains the bloodline relationships between individuals of multiple families, and we add gender of each person.
- *UMLS* (Kok and Domingos, 2007): Unified Medical Language System, is a set that brings together many health and biomedical vocabularies and standards.
- *FB15K-237* (Toutanova et al., 2015): This dataset contains knowledge base relation triplets and textual mentions of Freebase entity pairs.

Dataset	#Triple	#Relation	#Entity
Constant	30000	10	18363
Family-gender	29854	12	3008
UMLS	5960	46	135
FB15K237	310116	237	14541

Table 1: Knowledge base completion datasets statistics.

For each dataset, we split it into four parts: *fact, train, valid and test*. *Fact set* is a subset, which is randomly extracted about 70% from the original *train set*. We use it to construct the path operator and constant operator and but don't use the data to train the model.

We implement our model with Pytorch framework and train our model on RTX3090 GPU. The ADAM optimizer was used to parameter tune with learning rate of 0.0001. Batch size is different for every dataset, respectively. We set both the hidden state dimension of BiLSTM and head relation vector size to 256.

5.2 Link Prediction

We compare EduCe to several embedding methods and rule mining methods, which include TransE(Bordes et al., 2013), RotatE(Sun et al., 2019), ConvE(Dettmers et al., 2018), DistMult(Yang et al., 2015), ComplEx(Trouillon et al., 2016) for embedding methods and Neural-LP(Yang et al., 2017), DRUM(Sadeghian et al., 2019) for differentiable rule mining methods on link prediction task. Meanwhile, experiments were conducted with **Educe**$_N$ that just utilizes soft path operator.

Category	Methods	UMLS				FB15K-237			
			Hit				Hit		
		MRR	**@1**	**@3**	**@10**	**MRR**	**@1**	**@3**	**@10**
KGE	TransE(Bordes et al., 2013)	0.668	0.468	0.845	0.930	0.290	0.199	-	0.471
	ConvE(Dettmers et al., 2018)	0.908	0.862	0.944	0.981	0.325	0.237	0.356	0.501
	DistMult(Yang et al., 2015)	0.753	0.651	0.821	0.930	0.241	0.155	0.263	0.419
	ComplEx(Trouillon et al., 2016)	0.961	0.935	0.985	0.992	0.247	0.158	0.275	0.428
	RotatE(Sun et al., 2019)	**0.948**	**0.914**	**0.980**	**0.994**	**0.338**	**0.241**	**0.375**	**0.533**
Differentiable Rule Learning	Neural-LP(Yang et al., 2017)	0.75	0.62	0.86	0.92	0.240	-	-	0.362
	DRUM(T=2)(Sadeghian et al., 2019)	0.81	0.67	0.94	**0.98**	0.250	0.187	0.271	0.373
	DRUM(T=3)(Sadeghian et al., 2019)	0.80	0.66	0.92	0.97	0.343	0.255	0.378	0.516
	EduCe(T=2)	0.852	0.745	**0.957**	0.975	0.368	0.275	0.414	0.546
	EduCe(T=3)	**0.857**	**0.789**	0.911	0.965	**0.419**	**0.314**	**0.471**	**0.619**
	Educe_N(T=2)	0.805	0.669	0.930	0.976	0.243	0.179	0.266	0.368
	Educe_N(T=3)	0.821	0.688	0.946	0.975	0.345	0.258	0.378	0.516

Table 2: Link prediction results on *UMLS* and *FB15K-237*.

	Constant		Family-gender	
	MRR	Hit@1/3/10	MRR	Hit@1/3/10
Neural-LP	.52	.43/ .50/ .67	.87	.79/ .93/ .99
DRUM(T=2)	.38	.28/ .49/ .50	.95	.92/ .98/ .99
DRUM(T=3)	.58	.36/ .77/ .99	.95	.92/ .98/ .99
EduCe(T=2)	.45	.41/ .49/ .50	**.96**	**.94/** .98/.99
EduCe(T=3)	**.76**	**.62/ .88/ 1.0**	.94	.91/ .97/ .99

Table 3: Link prediction results on *Constant* and *Family-gender*.

Following the evaluation method in (Bordes et al., 2013), hit1, hit3, hit10 and MRR(Mean Reciprocal Rank) was reported after filtered ranking.

As the Tables 2 and 3 show, EduCe significantly outperforms other rule mining methods on the synthetic dataset *Constant* as expected, which proves the ability of EduCe to utilize constants. More convincing is that EduCe also outperforms other differentiable rule mining methods for all metrics on both real-world datasets obviously. Meanwhile, the result of EduCe with different length of rule is closer than DRUM, and we think it's because constants make the rules more accurate. Notably, our approach is able to achieve better results than some pure embedding methods, especially on *FB15K-237*. On dataset *UMLS*, it is also competitive. We have to point out that EduCe can provide symbolic logical rules with Algorithm 1, which is an advantage to pure embedding methods.

Performance of Educe_N is close to DRUM, and we analyzed the parameters β of soft constant operator and found out almost all weight is assigned to the special relation r_0 that we introduced before, which means in the optimization process, Educe_N cannot handle too many constants, so a compromise option is selected, which is ignoring constants. This optimization failure proves the necessity of dynamic weight.

The results also show that previous rule-based methods get worse performance than embedding methods on this task generally, but the effectiveness of EduCe proves the promising future of neural-symbolic method. Also, the most important thing is it proves constants play an important role in inferring, taking this part of KG into consideration will significantly improve the results of reasoning.

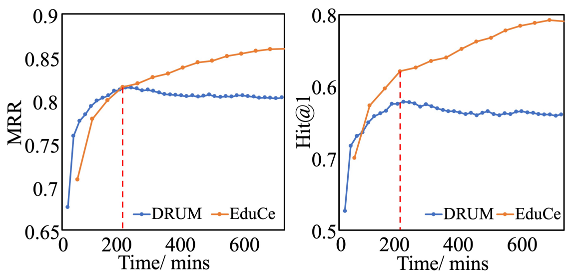

Figure 3: Behaviour of DRUM and EduCe on *UMLS*.

We also recorded the training time and represented the results in Figure 3, and each epoch is marked as a point on the curve. The figure shows EduCe starts at a high level and improves more quickly than DRUM within the same epoch. Although the training time of one epoch is longer, EduCe converge with fewer epochs. Also, at 200 minute, DRUM nearly achieves its best performance, and EduCe can achieve same or better accuracy at the same time. This proves the efficiency of our model.

5.3 Rule Decoding and Quality Evaluation

As we stated in the previous sections, the key advantage of rule-based methods is the interpretability of inferring process. In order to have an intuitive

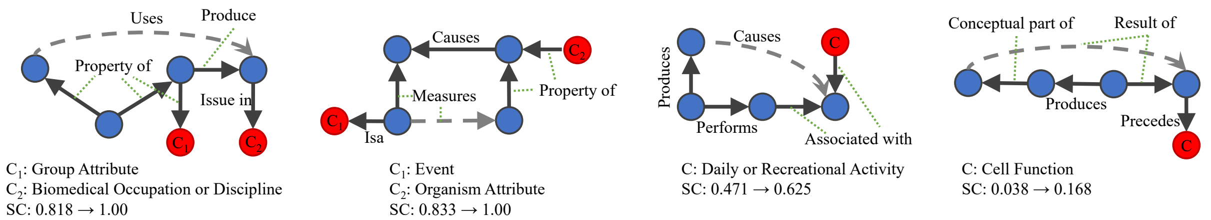

Figure 4: Rules examples learned by EduCe on *UMLS*.

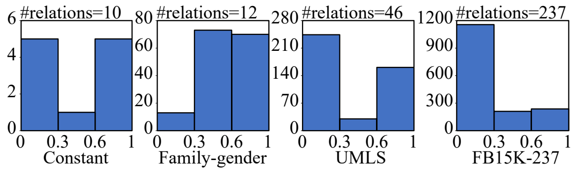

Figure 5: Number of rules on different datasets.

understanding of the results of rule mining. We performed Algorithm 1 on the datasets(*Constant*, *Family-gender*, *UMLS*, *FB15K-237*), and every dataset yielded useful results.

We counted the statistical number that is represented in Figure 5. The rules are sorted by the assigned confidence of Algorithm 1, and the figure shows different numbers of rules with different confidence. We use the highest and lowest confidence of rules on each dataset as standard, and divide this interval into three parts according to the ratio of 3:3:4, which is represented by the horizontal axis. On all datasets, the model has parsed out an appropriate number of rules.

Methods	UMLS	FB15K-237
	Top 50/100/200	Top 50/200/500
Neural-LP(T=2)	.228/.239/.221	.020/.044/.033
Neural-LP(T=3)	.104/.145/.153	.020/.031/.034
DRUM(T=2)	.400/.350/.303	.058/.036/.048
DRUM(T=3)	.340/.284/.202	.020/.039/.027
EduCe(T=2)	.541/**.482**/**.446**	.363/.339/.278
EduCe(T=3)	**.546**/.386/.424	**.405**/**.383**/**.399**

Table 4: Average confidence of ranked rules on *UMLS* and *FB15K-237*.

To reach an objective assessment of the rule quality, the rules with higher confidence, which is calculated by decoding algorithm, are selected, and we calculated the average **Standard Confidence** (Galárraga et al., 2013) of rules. The result is shown in Table 4. We can see that with constant, the quality of rules improves a lot.

For demonstration purpose, examples of the rule mined by EduCe on *UMLS* are shown in Figure 4. We choose four rules of different types, which appear frequently in the inference process. These examples illustrate the diversity of the rules we mined. Note that even though all of the examples are with constants, EduCe can also mine rules without constants.

Like the first example, the rule is *Uses(X, Y)* $\leftarrow$ *PropertyOf*$(Z_1, X)\wedge$ *PropertyOf*$(Z_1, Z_2)\wedge$ *Produce*$(Z_2, Y)\wedge$ *IssueIn(Y, Biomedical Occupation or Discipline)*. In the dataset, *Y*, as the tail entity of relation *Uses*, can only be an instance of medicine or medical device. However, the tail entity set of the third relation *Produce* contains other type entities like *Regular or Law* and *Age Group*, so the rule uses *IssueIn(Y, Biomedical Occupation or Discipline)* to choose eligible entities from the candidate set, which is *medicine* or *medical device* here. The standard confidence (SC) of rule removing constant from the body is also calculated. Specifically, without constants, it is 0.818 for the first example, and it will be improved to 1.00 with constants considered.

It is worth to mention that on dataset *Constant*, the pre-defined rules, which are used to build the dataset, are parsed by EduCe precisely. Figure 5 illustrates on this dataset, there are 5 rules in the range of high confidence, which includes the pre-defined rules we use.

6 Conclusion

In this article, we addressed the problem of learning rules with constants from KGs. In particular, we considered rules in a new form which is based on the constant operator and dynamic weight and proposed a rule mining model, EduCe, which allows us to learn such rules from KGs effectively in a differentiable way. The experiment result shows that our approach is superior to previous works, which do not take constants into consideration, both in terms of the consequence of link prediction task and quality evaluation of rule mining. Future research may focus on further expansion of current method by designing more complex forms of rule.

References

Trapit Bansal, Da-Cheng Juan, Sujith Ravi, and Andrew McCallum. 2019. A2n: attending to neighbors for knowledge graph inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4387–4392.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795.

Yang Chen, Sean Goldberg, Daisy Zhe Wang, and Soumitra Siddharth Johri. 2016. Ontological pathfinding. In *Proceedings of the 2016 International Conference on Management of Data*, pages 835–846.

William W Cohen. 2016. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. 2015. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24(6):707–730.

Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422.

Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. 2016. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 192–202.

Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. Openke: An open toolkit for knowledge embedding. In *Proceedings of EMNLP*.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. 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)*, pages 687–696.

Stanley Kok and Pedro M. Domingos. 2007. Statistical predicate invention. In *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 433–440. ACM.

Daphne Koller, Nir Friedman, Sašo Džeroski, Charles Sutton, Andrew McCallum, Avi Pfeffer, Pieter Abbeel, Ming-Fai Wong, Chris Meek, Jennifer Neville, et al. 2007. *Introduction to statistical relational learning*. MIT press.

Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2016. Knowledge representation learning with entities, attributes and relations. *ethnicity*, 1:41–52.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. 2019. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, pages 3137–3143. ijcai.org.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Stephen Muggleton. 1995. Inductive logic programming: Inverse resolution and beyond. In *Proceedings of the 14th international joint conference on Artificial intelligence-Volume 1*, pages 997–997.

Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. 2019. Learning attention-based embeddings for relation prediction in knowledge graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4710–4723.

Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. 2018a. Robust discovery of positive and negative rules in knowledge bases. In *ICDE*, pages 1168–1179. IEEE Computer Society.

Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. 2018b. Robust discovery of positive and negative rules in knowledge bases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1168–1179. IEEE.

Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. 2019. Drum: End-to-end differentiable rule mining on knowledge graphs. *arXiv preprint arXiv:1911.00055*.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer.

G. Shu, W. Quan, L. Wang, B. Wang, and G. Li. 2017. Knowledge graph embedding with iterative guidance from soft rules.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *EMNLP*, pages 1499–1509. The Association for Computational Linguistics.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. 2020. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Po-Wei Wang, Daria Stepanova, Csaba Domokos, and J Zico Kolter. 2019. Differentiable learning of numerical rules in knowledge graphs. In *International Conference on Learning Representations*.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations*.

Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. *arXiv preprint arXiv:1702.08367*.

Yuan Yang and Le Song. 2019. Learn to explain efficiently via neural logic inductive learning. *arXiv preprint arXiv:1910.02481*.

Wen Zhang, Bibek Paudel, Liang Wang, Jiaoyan Chen, Hai Zhu, Wei Zhang, Abraham Bernstein, and Huajun Chen. 2019. Iteratively learning embeddings and rules for knowledge graph reasoning. In *The World Wide Web Conference*, pages 2366–2377. ACM.

Appendices

Definition of Standard Confidence

As stated in (Galárraga et al., 2013), the standard confidence measure regards facts that are not in KG as false, in an other word, it implements a closed world setting. Based on this, the standard confidence of a rule is defined as:

$$conf(\mathbf{B} \rightarrow r(x, y)) = \frac{supp(\mathbf{B} \rightarrow r(x, y))}{(x, y) : \exists z_1, \ldots, z_m : \mathbf{B}}$$

where $\mathbf{B}$ is rule body, z are the variables in the rule body apart from x and y according to (Galárraga et al., 2013), and we expand them to variables and constants. This indicates the ratio of its predictions that are in KG. $supp(\mathbf{B} \rightarrow r(x, y))$ is defined as follows:

$$supp(\mathbf{B} \rightarrow r(x, y)) = (x, y) : \exists z_1, ..., z_m : \mathbf{B} \wedge r(x, y)$$

More Details about Datasets

In the previous section, we introduced that dataset *Constant* is constructed based on different groups of pre-defined rules. Here is an example of one group.

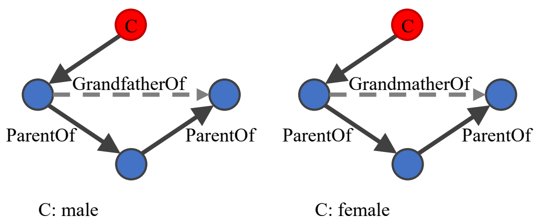

Figure 6: Rule example in *Constant*.

The only difference in the body of these two rules is the constant, which we mean *male* and *female* here. There are several such groups of rules used in *Constant*.

We use OpenKE (Han et al., 2018) to get the results of embedding methods on link prediction task about *UMLS*, and the results of *FB15K-237* are from (Sadeghian et al., 2019).

More cases of mined rules

Causes(A,D)	Produces(B,A), Performs(B,C), Associated_with(C,D), Associated_with(Research_Activity,D), Associated_with(Health_Care_Activity,D),Associated_with(Finding,D),Associated_with(Geographic_Area,D),Associated_with(Daily_or_Recreational_Activity,D),Associated_with(Laboratory_Procedure, D),Associated_with(Therapeutic_or_Preventive_Procedure,D)
	Produces(B,A),Performs(B,C),Associated_with(C,D),Associated_with(Research_Activity,D), Associated_with(Finding,D),Associated_with(Daily_or_Recreational_Activity,D),Associated_with (Laboratory_Procedure,D)
	Produces(B,A),Performs(B,C),Associated_with(C,D), Associated_with(Research_Activity,D), Associated_with(Finding,D), Associated_with(Daily_or_Recreational_Activity,D)
	Produces(B,A),Performs(B,C),Associated_with(C,D), Associated_with(Daily_or_Recreational_Activity,D)
	Produces(B,A),Property_of(Group_Attribute,B),Performs(B,C),Associated_with(C,D),Associated_with(Research_Activity,D),Associated_with(Health_Care_Activity,D),Associated_with(Finding,D), Associated_with(Geographic_Area,D),Associated_with(Daily_or_Recreational_Activity,D), Associated_with(Laboratory_Procedure,D), Associated_with(Therapeutic_or_Preventive_Procedure,D)
	Produces(B,A),Property_of(Group_Attribute,B),Performs(B,C),Associated_with(C,D),Associated_with(Research_Activity,D),Associated_with(Finding,D),Associated_with(Daily_or_Recreational_Activity,D),Associated_with(Laboratory_Procedure,D)
Consists_of(A,D)	Produces(A,B),Produces(C,B),Produces(C,D),Issue_in(D,Occupation_or_Discipline),Issue_in(D,Bio medical_Occupation_or_Discipline)
Ingredient_of(D,A)	Produces(B,A), Property_of(Group_Attribute,B), Performs(B,C), Analyzes(C,D)
	Produces(B,A), Property_of(Group_Attribute,B), Produces(C,B), Complicates(D,C)
Ingredient_of(A,D)	Causes(A,B),Occurs_in(B,Patient_or_Disabled_Group),Occurs_in(B,Family_Group),Occurs_in(B,Pop ulation_Group),Occurs_in(B,Professional_or_Occupational_Group),Occurs_in(B,Group),Produces(B, C), Ingredient_of(C,D)
	Causes(A,B),Occurs_in(B,Patient_or_Disabled_Group),Occurs_in(B,Family_Group),Occurs_in(B,Age _Group), Occurs_in(B,Population_Group), Occurs_in(B,Professional_or_Occupational_ Group), Occurs_in(B,Group), Produces(B,C), Ingredient_of(C,D)
	Produces(B,A),Produces(B,C),Produces(C,D),Issue_in(D,Occupation_or_Discipline),Issue_in(D,Biomedical_Occupation_or_Discipline)
Isa(A,D)	Performs(B,A), Property_of(Group_Attribute,B), Performs(B,C), Isa(C,D)
Issue_in(A,D)	Produces(B,A), Property_of(Group_Attribute,B), Performs(B,C), Issue_in(C,D)
Measures(D,A)	Property_of(A,B), Property_of(Organism_Attribute,B), Causes(B,C), Measures(D,C), Isa(D,Event), Isa(D,Activity)
	Property_of(A,B),Property_of(Organism_Attribute,B),Property_of(Clinical_Attribute,B),Causes(B,C), Isa(C,Natural_Phenomenon_or_Process), Isa(C,Biologic_Function), Measures(D,C), Isa(D,Event), Isa(D,Activity)
Occurs_in(A,D)	Prevents(B,A),Prevents(B,C),Co-occurs_with(D,C),Associated_with(Educational_Activity,D), Associated_with(Health_Care_Activity,D), Associated_with(Geographic_Area,D),Associated_with(Daily _or_Recreational_Activity,D), Associated_with(Therapeutic_or_Preventive_Procedure,D)
Treats(A,D)	Produces(B,A), Performs(B,C), Associated_with(C,D), Associated_with(Research_Activity,D), Associated_with(Health_Care_Activity,D), Associated_with(Daily_or_Recreational_Activity,D), Associated_with(Therapeutic_or_Preventive_Procedure,D)
	Produces(B,A), Performs(B,C), Associated_with(C,D), Associated_with(Research_Activity,D), Associated_with(Daily_or_Recreational_Activity,D)
Treats(A,D)	Produces(B,A),Performs(B,C),Associated_with(C,D),Associated_with(Daily_or_Recreational_Activity ,D)
Uses(A,D)	Property_of(B,A), Property_of(B,C), Property_of(Group_Attribute,C), Produces(C,D)
	Property_of(B,A),Property_of(B,C),Property_of(Group,C), Produces(C,D),Issue_in(D,Occupation_or_Discipline), Issue_in(D,Biomedical_Occupation_or_Discipline)