DIFFERENTIABLE LEARNING OF NUMERICAL RULES IN KNOWLEDGE GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Rules over a knowledge graph (KG) capture interpretable patterns in data and can be used for KG cleaning and completion. Inspired by the `TensorLog` differentiable logic framework, which compiles rule inference into a sequence of differentiable operations, recently a method called Neural LP has been proposed for learning the parameters as well as the structure of rules. However, it is limited with respect to the treatment of numerical features like *age*, *weight* or *scientific measurements*. We address this limitation by extending Neural LP to learn rules with numerical values, *e.g.*, "*People younger than 18 typically live with their parents*". We demonstrate how dynamic programming and cumulative sum operations can be exploited to ensure efficiency of such extension. Our novel approach allows us to extract more expressive rules with aggregates, which are of higher quality and yield more accurate predictions compared to rules learned by the state-of-the-art methods, as shown by our experiments on synthetic and real-world datasets.

1 INTRODUCTION

Due to the availability of vast amounts of knowledge on the web, advances in information extraction have led to large graph-structured knowledge bases, also known as *knowledge graphs* (KGs), which are widely used in web search, question answering, and data analytics. Such KGs represent data as a graph of entities (*e.g.*, $john$, $article1$) connected via relations (*e.g.*, $citedIn$), or more formally as a set of binary grounded atoms (*e.g.*, $citedIn(john, article1)$). A common task in such settings is that of *link prediction*, determining whether a relation exists between two entities in the graph even if the relation is not included explicitly in the graph. Although most work on this topic has focused on statistical rule-extraction techniques (Meilicke et al. (2019); Galarraga et al. (2015); Ortona et al. (2018a)), recent methods have shown the benefit of using deep learning approaches for this link prediction task (see Wang et al. (2017) for overview). And while most deep approaches (for example, those based upon graph embedding methods) are inherently difficult to interpret, the Neural LP method of Yang et al. (2017) is particularly appealing in that it allows for interpretable resulting rules for the link prediction task while still preserving the flexibility of a learning approach. Unfortunately, Neural LP is also quite limited in the types of rules it is capable of representing, and notably no rules that depend on *numerical features* can be efficiently learned within this framework.

In this paper, we propose an extension to Neural LP that allows for fast learning of numerical rules. Specifically, although numerical rules would result in dense matrix operations in the generic Neural LP framework, we show that using dynamic programming and cumulative sum operations, we can efficiently express the operators for numerical comparators within the Neural LP framework. By defining the relevant operators implicitly in this manner, we show that we can extend Neural LP to efficiently learn rules that make use of numerical features, while retaining the interpretability of the Neural LP framework. More generally, this is an instance of integrating so-called "aggregates" (*i.e.* external oracle queries, in this case binary queries that reflect numerical comparison) within a rule-learning framework. Learning such rules with aggregates is very much an open problem in the KG community (Galárraga & Suchanek (2014)), and our approach is the first work to learn rules with these numerical aggregates.

We apply our approach to several knowledge graph datasets, and show that we are able to answer queries more accurately than the previous Neural LP approach, as well as more accurately that a state-of-the-art rule extraction method, the AnyBurl package proposed by Meilicke et al. (2019).

Specifically, we show on two synthetic and two real-world datasets that our extension to Neural LP is able to more accurately recover rules that depend on numerical information, and thus make much more accurate link predictions in the knowledge graph. Further, the extracted rules are still interpretable as in the original Neural LP framework, and unlike the pure graph embedding strategies (Bordes et al. (2013)).

2 Related Work

Relational Data Mining. The problem of learning rules from the data has been traditionally addressed in the area of relational data mining (Raedt (2017)) and inductive logic programming (ILP) (Muggleton (1995)). Works most related to ours concern learning non-monotonic rules (Law et al. (2018); Inoue & Kudoh (1997)) and decision trees with aggregates (Vens et al. (2006)) from relational data.

In the context of KGs, the problem of rule learning has recently gained a lot of attention. In Ortona et al. (2018b) rules with negation and simple numerical comparison have been considered. Contrary to our approach, in Ortona et al. (2018b) aggregates are not supported and their method is designed to find a small set of rules that cover the majority of positive and as few negative examples as possible, which differs from our objective of learning rules in an unsupervised fashion.

Neural-based Rule Learning. Several works utilize embedding models and neural architectures for rule learning (Yang et al. (2017); Manhaeve et al. (2018); Rocktäschel & Riedel (2017); Evans & Grefenstette (2018); Zhang et al. (2019); Ho et al. (2018)). The closest to ours is the work of Yang et al. (2017), which reduces the rule learning problem to algebraic operations on neural-embedding-based representations of a given KG. However, in Yang et al. (2017) negations and aggregates are not supported in contrast to our work.

Embedding Models with Numerical Attributes. The problem of KG incompleteness has been tackled by methods that predict missing relational edges between existing entities. Several approaches rely on statistics and include tensor factorization (Nickel et al. (2011)). Other models are based on neural-embedding (Bordes et al. (2013)). For overview see Wang et al. (2017).

The most relevant for us is the work of García-Durán & Niepert (2018), which presents a novel approach to combine relational, latent (learned) and numerical features, *i.e.* features taking large or infinite number of real values for the KG completion task. While this work operates on KGs with numerical values, in contrast to ours its goal is primarily fact prediction rather than rule learning.

3 Preliminaries

Knowledge Graphs. We assume countable sets $\mathcal{C}$ of constants, $\mathcal{N} \subset \mathbb{R}$ of numerical values and $\mathcal{R}$ of binary relations. A KG $\mathcal{G}$ is defined by a finite set of ground atoms, *a.k.a.* facts, of the form $p(x, y)$, where $p \in \mathcal{R}$, $x \in \mathcal{C}$ and $y \in \mathcal{C} \cup \mathcal{N}$ (*e.g.*, $\textit{citedIn}(\textit{john}, \textit{article1})$). The set $\mathcal{R}_n \subseteq \mathcal{R}$ stores all numerical predicates p, such that $p(x, y) \in \mathcal{G}$, where $x \in \mathcal{C}$ and $y \in \mathcal{N}$. The set of numerical facts, *i.e.* facts over numerical predicates, is denoted by $\mathcal{G}_n \subseteq \mathcal{G}$. We use lower-case letters for constants and upper-case letters for variables.

As KGs are incomplete, one can assume that missing facts, *i.e.* facts that are not in $\mathcal{G}$, are either *unknown* or *false*. Typically the *open world assumption* (OWA) is employed, which means that missing facts are considered to be unknown rather than false. Alternatively, the *local closed world assumption* (LCWA) can be considered to generate negative facts by assuming that the KG is locally complete as data is usually added to KGs in batches. More precisely, it means that for any $x \in \mathcal{C}$ we can conclude that $p(x, y)$ is false if $\exists z \in \mathcal{C} \cup \mathcal{N}$ such that $p(x, z) \in \mathcal{G}$ and $p(x, y) \notin \mathcal{G}$.

Numerical rules. A *rule* is an expression of the form

$$p(Y, X) \leftarrow q_n(Y, Z_n) \wedge \ldots \wedge q_1(Z_1, X). \tag{1}$$

where $p, q_1, \ldots, q_n \in \mathcal{R}$, left-hand side of the rule is referred to as the rule head and right-hand side as the rule body, and every conjunct in the rule head or body is referred to as an atom. The

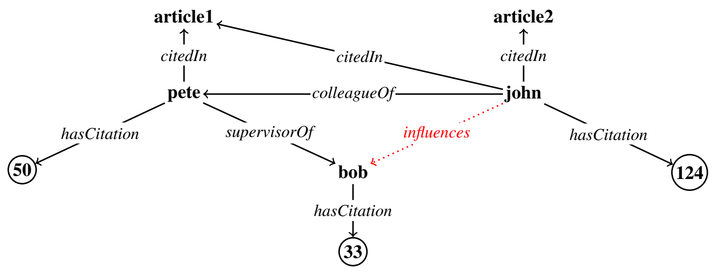

Figure 1: An exemplar KG about publications, their authors and relations among them. Relations are presented in italics, entities in bold black and numerical values in circes, true facts as solid black lines and the missing ones as dashed in red.

rule $\textit{influences}(Y, X) \leftarrow \textit{colleagueOf}(Y, Z) \wedge \textit{supervisorOf}(Z, X)$ intuitively states that typically students are influenced by colleagues of their supervisors. Aside from conventional rules, we can also have *numerical rules*, *i.e.* rules that contain numerical *comparison* among variables (*e.g.*, number of citations of two people), or a variable and a numerical constant.

To simplify presentation, numerical values in $\mathcal{N}$ linked to an entity from $\mathcal{C}$ are sometimes treated as its "features", and numerical relations in $\mathcal{R}_n$ as functions that depend on those features. In this case, for $p(X, Y)$ we also use a shortcut notation $X.p = Y$. For instance, $john.hasCitation = 124$ stands for $hasCitation(john, 124)$, and for compactness, $r^{\circ}_{pq}(X, Y)$ stands for $X.p \circ Y.q$, where $\circ \in \{\leq, >\}$. The second subscript in r°_{pq} is omitted if it is clear from the context that $p = q$.

Example 1. *For example, consider a KG in Fig 1 and the rule* $\textit{influences}(X, Y) \leftarrow \textit{colleagueOf}(Y, Z) \wedge \textit{supervisorOf}(Z, X) \wedge r^{>}_{hasCitation}(Y, Z)$ *states that students are influenced by colleagues of their supervisors with a higher number of citations*[1].

We can also define a classification relation mapping the feature to the probability of a logistic classification, $\sigma(w^T X.\text{features} + b)$, where w and b are parameters and σ is the sigmoid function. As we demonstrate later such rules can be integrated and learnt naturally in the Neural LP framework.

Rule Learning. Given a KG $\mathcal{G}$ the goal of *rule learning* is to extract rules from $\mathcal{G}$, such that their application to $\mathcal{G}$ results in an approximation of the ideal KG, which stores all correct facts.

The Neural LP method (Yang et al. (2017))is among rule learning proposals, which learns a distribution over rules of the form in Eq. (1) without comparison operators in an end-to-end fashion by making use of gradient-based optimization. This approach relies on the `TensorLog` framework (Cohen et al. (2017)), which connects rule application with sparse matrix multiplications. In `TensorLog` all entities are mapped to integers, and each entity i is associated with a one-hot encoded vector $v_i \in \{0, 1\}^{|\mathcal{C}|}$ such that only its i-th entry is 1.

For example, every KG entity $c \in \mathcal{C}$ in Fig. 1 is encoded as a $0/1$ vector of length 5, since $|\mathcal{C}| = 5$. For every relation $p \in \mathcal{R} \setminus \mathcal{R}_n$ and every pair of entities $x, y \in \mathcal{C}$ a matrix $M_p \in \{0, 1\}^{|\mathcal{C}| \times |\mathcal{C}|}$ is defined such that its (x, y) entry, denoted by $(M_p)_{xy}$, is 1 iff $p(x, y) \in \mathcal{G}$. For example, by considering the KG in Fig. 1, for the relation $p = \textit{citedIn}$ we have

$$M_p = \begin{array}{ccccc|l} \text{john} & \text{pete} & \text{bob} & \text{article1} & \text{article2} & \\ 0 & 0 & 0 & 1 & 1 & \text{john} \\ 0 & 0 & 0 & 1 & 0 & \text{pete} \\ 0 & 0 & 0 & 0 & 0 & \text{bob} \\ 0 & 0 & 0 & 0 & 0 & \text{article1} \\ 0 & 0 & 0 & 0 & 0 & \text{article2} \end{array}$$

The idea of `TensorLog` is to imitate application of rules for any entity $X = x$ by performing matrix multiplications $M_{q_1} M_{q_2} \dots M_{q_n} v_x = s$, where v_x is the indicator of entity x. Non-zero entries in the vector s point to the entities y for which $p(x, y)$ is derived by applying

[1] *I.e.* $\textit{influences}(Y, X) \leftarrow \textit{colleague}(Y, Z) \wedge \textit{supervisorOf}(Z, X) \wedge \textit{hasCitation}(Z, U) \wedge \textit{hasCitation}(Y, V) \wedge U > V$.

the above rule on $\mathcal{G}$. For example, the inference for the following rule $\mathit{influences}(Y, X) \leftarrow \mathit{colleagueOf}(Y, Z), \mathit{supervisorOf}(Z, X)$ can be translated to

$$M_{supervisorOf}\ M_{colleagueOf}\ v_x = s\ .$$

By setting $v_x = [1, 0, 0, 0, 0]^\top$ as indicator of $john$ and applying the matrix multiplications, we obtain $s = [0, 0, 1, 0, 0]^\top$, the indicator of bob. As $M_{q_1}, \dots, M_{q_n}$ are sparse, the matrix-vector multiplication can be done efficiently, and the inference process is parallelizable on GPUs.

In Neural LP (Yang et al. (2017)) above operators are used to learn for every head the formula

$$f(\alpha) = \sum_i \alpha_i \prod_{j \in \beta_i} M_{q_j} \tag{2}$$

where i indexes over all possible rules, α_i is the confidence associated with the rule r_i and β_i is an ordered list of all relations appearing in these rules. The rules are read off from the solution of the following optimization problem

$$\max_{\{\alpha_i, \beta_i\}} \log \left(\sum_{\{x,y\}} v_y^\top \Big(\sum_i \alpha_i \Big(\prod_{j \in \beta_j} M_{q_j} \Big) v_x \Big) \right) .$$

4 Learning Rules with Numerical Features and Negations

As the main contribution of the paper we extend the Neural LP framework to allow us to use comparison operators with numerical values in the rule bodies, and also to handle negations of atoms. These extensions are non-trivial as the Neural LP framework does not handle facts over numerical values: naively treating numerical constants as entities in $\mathcal{C}$ is intractable due to the explosion of the number of non-zero elements in the respective matrices. Similarly, naive treatment of negated clauses would introduce dense matrices that would not be practical to operate on. Intuitively, the main idea of our approach is to represent the necessary matrix operations implicitly, either using dynamic programming, cumulative sums and permutations (for numerical comparison features) and low rank factorizations (for negated atoms). Exploiting this structure lets us formulate the associated `TensorLog` operators efficiently, and effectively integrate them into the Neural LP framework for rule extraction.

4.1 Comparison Operators

Pair-wise comparison. We start by implicitly representing the operators associated with numerical comparators. Let $p, q \in (\mathbb{R} \cup \{\texttt{NaN}\})^{|\mathcal{C}|}$ be the vector of two specific features, where `NaN` means missing values. The comparison operator $M_{r^{\leq}_{pq}}$ is defined as

$$(M_{r^{\leq}_{pq}})_{ij} = \begin{cases} 1 & \text{if } p_i \leq q_j \text{ and } p_i, q_j \text{ is not } \texttt{NaN}, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, this matrix including the binary indicator of the comparison, over all pairs of entities in the knowledge graph that contain p and q. Unlike conventional sparse relations, the matrix $M_{r^{\leq}_{pq}}$ is usually dense (*i.e.* it has $O(n^2/2)$ non-zero elements), thus a naive materialization would exceed the typical GPU memory limit. However, in reality there is no need to explicitly materialize the TensorLog relation matrix. Note that in the Neural LP inference chain we described above, all that is needed is to efficiently compute the matrix-vector product between a relation matrix and some vector representing the current probabilities in the inference chain.

Consider the special case that both p and q are sorted in an ascending order as $\tilde{p}$ and $\tilde{q}$ with operator $(\leq)$[2], and call the corresponding comparison matrix $\tilde{M}_{r^{\geq}_{pq}}$. Because $\tilde{q}_i \leq \tilde{q}_{i+1}$ and $\tilde{p}_j \leq \tilde{p}_{j+1}$, we have the following property (**P1**)

$$(\tilde{M}_{r^{\leq}_{pq}})_{i,j} = 1 \implies (\tilde{M}_{r^{\leq}_{pq}})_{i+1,j} = 1$$
$$(\tilde{M}_{r^{\leq}_{pq}})_{i,j} = 0 \implies (\tilde{M}_{r^{\leq}_{pq}})_{i,j+1} = 0,$$

[2]This way, the comparison involving `NaN` always yields false and they will be stacked at the end.

i.e., the resulting matrix $\tilde{M}_{r^{\leq}_{pq}}$ is always effectively *lower triangular* in form (or more precisely, the transition from 1 to 0 is always monotonic in the matrix, even if the non-zero pattern is not precisely lower triangular in the usual sense).

Now define $\gamma_i = \arg\max_j$ such that $(\tilde{M}_{r^{\leq}_{pq}})_{ij} = 1$, i.e., γ_i is the index of the last element equal to one. The main observation is that we can compute the required matrix-vector product using just this γ vector, *i.e.* for any vector v,

$$(\tilde{M}_{r^{\leq}_{pq}} v)_i = \sum_{1 \leq j \leq \gamma_i} v_j \ = \texttt{cumsum}(v)_{\gamma_i} \ .$$

The respective values of γ for $\tilde{M}_{r^{\leq}_{pq}} v$ can be precomputed on a CPU with linear complexity by dynamic programming since its value is monotonically increasing because of the property (**P1**). Also, the `cumsum` operator can be calculated in $O(|\mathcal{C}|)$ time, with an efficient GPU parallelization that in practice is even faster for large vectors.

For the general case when p and q are not sorted, we can first permute the input v to the sorted order, perform the matrix-vector multiplication, then permute the result back to the original order. Since permutation (aka, index slicing) is a simple linear time operation, this does not affect the complexity of the overall approach. Specifically, let P_p and P_q be the permutation matrix corresponding to the argsort of p and q, respectively. Then the matrix-vector multiplication corresponding to the comparison operator can be written as

$$M_{r^{\leq}_{pq}} v = P_q^\top \texttt{cumsum}(P_p v)_\gamma \ ,$$

which can be computed in $O(|\mathcal{C}|)$ in parallel given β, which are precomputed once in $O(|\mathcal{C}| \log |\mathcal{C}|)$. Thus, the comparison operator needed for inference can be computed efficiently on a GPU.

Efficient use of numerical comparisons via multi-atom symbol matching. Although the above numerical comparison operator provides an efficient means for implementing such comparisons within the Neural LP framework, it has significant drawbacks as well. Specifically, because the comparison operator is dense, when using it to match potential entities in the graph, it has the potential to create a huge number of candidate matches. For example, the operator $(X.p \leq Y.p)$ will link the entity with smallest p to all other entities with attribute p and will decrease the probability of finding the correct target. To make the comparison operator more useful, it is natural to use it jointly with other sparse operator.

$$colleagueOf(X, Y) \wedge (X.p > Y.q)$$

which would search only over neighbors of X in the graph that also obeyed this numerical relation. Let the two parallel rules above corresponds to operator $M_{colleagueOf}$ and $M_{r^{\leq}_{pq}}$. The above relation can be implemented in TensorLog via

$$(M_{colleagueOf} v) \odot (M_{r^{\leq}_{pq}} v) = \text{diag}(M_{colleagueOf} v)(M_{r^{\leq}_{pq}} v),$$

where the symbol $\odot$ denotes the element-wise multiplication. Unfortunately, the above relation is not learnable in the standard Neural LP framework which only operators a single chain of matrix-vector operations $M_{q_1} M_{q_2} \ldots M_{q_m} v_x$, which doesn't allow easy computation of this Hadamard product, as it includes two "copies" of the vector v. However, we note that it is trivial to simply cache intermediate values of v in the multiplication chain, which allows us to compute such Hadamard products; in the knowledge graph setting, this exactly corresponds to the ability to integrating symbol matching at multiple points in the inference chain.

Classification Operators. We may also consider rules more general rules, where the comparison is performed not necessarily among two numerical attributes of a certain entity but rather functions over such attributes. Note that such rules M for all entities can readily be expressed by `TensorLog` operators, that is, the corresponding matrix for a given numerical value Z is a diagonal matrix. We model the numerical value Z by making use a logistic model. Namely, for each entity we collect the feature vector φ, which consists of all the numerical values from $\mathcal{N}$ that is in relation with the given entity. The i^{th} element of the diagonal in M is defined as sigmoid$(w^\top \varphi + b)$, where the weight vector w and the bias vector b assumed to be learned. These parameters can easily be learnt in the Neural LP framework via backpropagation.

Dataset	#Entities	#Relations	Feature dim	#Features	#Facts	#Testset
FB15K-237Num	12493	237	116	27899	82992	10359
DBP15K	12867	278	251	48105	79345	9789
Numerical1	1000	2	1	1000	5785	98
Numerical2	1000	2	2	2000	5800	100

Table 1: Dataset statistics.

Negated Operators. The negation of a relation $p \in \bar{\mathcal{R}}$ obtained by naively flipping all zeros to ones and vice versa in the corresponding (sparse) matrix M_p results in a dense matrix, which are not supported directly in `TensorLog`. To compute the negated operator $\bar{M}_p \in \{0,1\}^{|\mathcal{C}|\times|\mathcal{C}|}$ we employ the *local closed-world assumption*. For a given M_p only the elements, that are in such rows that contain at least one non-zero element, should be flipped. The matrix-vector multiplication for the negated operator $\bar{M}_p$ can be written as

$$\bar{M}_p v := 1_p 1_p^\top v - M_p v \ , \tag{3}$$

where $1_p \in \{0,1\}^{|\mathcal{C}|}$ is the indicator vector for p such that $(1_p)_i = 0$ iff $p_i = \mathtt{NaN}$. Note that for any `TensorLog` operator M_p the products $M_p v$ and $1_p(1_p^\top v)$ can be computed efficiently, therefore the negated operator $\bar{M}_p$ can be computed efficiently as well.

The trick in Eq. (3) generalizes to the comparison operators $M_{r_{pq}^{\circ}}$, namely,

$$\bar{M}_{r_{pq}^{\circ}} v = 1_p 1_q^\top v - M_{r_{pq}^{\circ}} v \ .$$

For example, $\bar{M}_{r_{pq}^{<}} v = 1_p 1_q^\top v - M_{r_{pq}^{\geq}} v$. This way we can learn rules with negated atoms in the body.

Once the rules have been learned by our approach, we rely on the same procedure as in Yang et al. (2017) to decode them back to the form Eq. 1.

Connection to aggregates in knowledge graphs Note that, importantly, the rules that we extract using the described procedure fall into the language of logic rules with external computations in the spirit of Eiter et al. (2012), a concept known as *aggregates* in knowledge bases. Indeed, much of the formulation we have presented here can be viewed as an instance of learning rules for knowledge with aggregates. This is an active area of current research, and our work here is significant in connection to this area in that we present one of the first methods for learning rules using (a limited form of) aggregates. However, the discussion requires substantial additional notation in order to be concrete, and so we defer this discussion to Appendix A

5 Experimental Results

In this section we report the results of our experimental evaluation, which focuses on the effectiveness of our method against the state-of-art rule learning systems with respect to the predictive quality of the learned rules. Specifically, we conduct experiments on a canonical knowledge graph completion task as described in Yang et al. (2017). In the task, the query and tail are given to the algorithm, and the goal is to retrieve the related head. For example, if $supervisorOf(turing, church)$ is not present in the knowledge graph, then when presented with the relation $supervisorOf$ and the entity $church$, the goal is to exploit the existing triples in the KG to retrieve $turing$. In order to represent the query as a continuous input to the neural controller, for each query we learn the embedding of the lookup table. As in Yang et al. (2017), the embedding has dimension 118 and is randomly initialized to unit norm vectors. The only difference between the parameters of the Neural-LP and our system is that set the learning rate 10^{-2}, while in Yang et al. (2017) it is set to 10^{-3}, but both systems are run to convergence, and this learning rate does not affect the final performance materially except for making it converge faster. In all cases, we extracted rules with a maximum length of 5.

5.1 Experimental Setup

Datasets. To evaluate and compare the developed approach for learning numerical rules, we considered the following datasets containing knowledge graphs:

Dataset	FB15K-237-num		DBP15K-num		Numerical1		Numerical2	
	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR
AnyBurl	0.4262	0.2438	0.5217	0.3711	0.0306	0.0085	0.6850	0.5087
Neural-LP	0.362*	0.240*	0.4363	-	0.2398	-	0.2950	-
Neural-LP-N	0.415	**0.259**	**0.682**	**0.451**	**1.000**	**0.941**	**1.000**	**0.837**

Table 2: Comparing our approach against current state-of-the-art rule learning methods. * annotated entries obtained from Yang et al. (2017).

- *FB15K-237-num* is a variant of Freebase knowledge base with numerical values, where the reverse relations have been removed (García-Durán & Niepert (2018)).
- *DBPedia15K* is extracted from the DBPedia knowledge graph Lehmann et al. (2015), a standard KG over encyclopedia facts, and restricted to consider numerical quantities García-Durán & Niepert (2018).
- *Numerical1* is a synthetic dataset with 1000 entities, each containing a single numerical value (generated uniformly from 1 to 1000). Each entity has 50 randomly-chosen neighbors, and the goal is to find neighbors with the closest value to the current entity, with the constraint that the neighbor's value must be higher.
- *Numerical2* is a variant on the *Numerical1* task, where each entity and two numerical values, "balance" and "debt"; under the same generation process as previously, the goal of this task is to find the neighbor of each node with the largest delta between balance and debt.

The statistics of the knowledge graphs used in our experiments is presented in Table 1, where apart from the number of KG entities, facts and the size of the test set, we also report the number of numerical relations (**Feature dim**) and numerical facts (**#Features**). We use 80% of the KG as the training set, and 10% for test set and the same for validation. The KG is split randomly with the constraint that only non-numerical facts appear in the test set, since we do not learn rules capable of predicting missing numerical entities.

Baselines. We compared our proposed approach, which we refer to as *Neural-LP-N* against the following two baselines:

- *AnyBURL*[3] (Meilicke et al. (2019)) is an anytime bottom-up method for learning Horn rules, *i.e.* rules with only positive atoms and no comparison operators. To tune the system we use the default parameters as described on the system webpage and set the timeout to 5000 seconds.
- *Neural-LP*[4] (Yang et al. (2017)) is a differential rule learning system described in Section 3. As in Yang et al. (2017) for Neural-LP we set the learning rate to 10^{-2}.

Following the common practice Meilicke et al. (2019); Yang et al. (2017) we compute the standard evaluation metrics used for the link prediction task Bordes et al. (2013): *Hit@10*, the number of correct head terms predicted out of the top 10 predictions; and mean reciprocal rank (*MRR*), the mean of one over the rank of the correct answer in the predictions. We have implemented our approach for learning numerical rules from knowledge graphs in python using the PyTorch library, and conduct all experiments on a machine GTX 1080 TO GPU with 11 GB RAM.

5.2 RESULTS

In Table 2 we report the quality of predictions obtained by our method and the baselines. Since the Neural LP framework Yang et al. (2017) cannot handle the Freebase with numerical information, we present the results for *FB15K-237* without numerical facts instead, which are taken from Yang et al. (2017). The MRR values are missing for Neural LP in several places, as the implementation provided by the authors does not have the respective function implemented.

[3]http://web.informatik.uni-mannheim.de/AnyBURL/

[4]https://github.com/fanyangxyz/Neural-LP

As expected, on the synthetic datasets *Numerical1* and *Numerical2*, our method significantly outperforms the baselines. This is natural, since these datasets are constructed so that reasoning about numerical attributes is required for almost any prediction task presented to the algorithms. And notably, the proposed approach is able to achieve 100% Hit@10 rates, as it is able to correctly identify these relevant numerical properties. This contrasts to the baseline Neural LP approach, which is unable to incorporate such information, and thus predicts the heads of each relation more or less randomly. The datasets are also particularly challenging for AnyBURL, because AnyBURL treats each numerical value as an independent entity, and thus cannot perform efficient comparative reasoning.

Most compellingly, however, similar observations can be made about the real-world datasets as well. Indeed, since all entities in the KG including numerical ones are treated equally by the available systems, intuitively both AnyBurl and Neural LP try to find frequent patterns in KGs and use these to predict the missing facts. The numerical rules mined by our system are much more expressive and substantially improve the performance of the approach in some cases. Specifically, our method outperforms the Neural LP approach in terms of all metrics on both the *FB15K-237-num* and *DBPedia15K* datasets. The AnyBURL dataset is still competitive with our approach on the *FB15K-237-num* dataset (better in terms of Hit@10 but worse in terms of MRR), but our approach substantially outperforms it on the *DBPedia15K*, where our numerical comparison reasoning is able to substantially improve upon the existing methods.

6 Conclusion

In this paper we have addressed the problem of learning numerical rules from large knowledge graphs. Especially we have considered rules with aggregates (*i.e.* external oracle queries) that enable us to use numerical comparison operators in the rule body. The Neural LP method is a recent appealing learning approach based on `TensorLog`, however it does not support numerical rules, as they would result in dense matrix operations. We have introduced an extension to the Neural LP framework that allows for learning such rules from KGs by efficiently expressing comparison and classification operators, negation and multi-atom symbol matching as well. We have shown that our proposed extension outperforms previous techniques that do not support numerical information with respect to the quality of predictions that they produce. The future research might focus on a further extension of our current approach by allowing for more general rule forms with complex external computations as well as rules with existential variables in the head.

References

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, pp. 2787–2795, 2013.

William W. Cohen, Fan Yang, and Kathryn Mazaitis. Tensorlog: Deep learning meets probabilistic dbs. *CoRR*, abs/1707.05390, 2017.

Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl. Conflict-driven ASP solving with external sources. *TPLP*, 12(4-5):659–679, 2012.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *JAIR*, 61: 1–64, 2018.

Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, 175(1):278–298, 2011.

Luis Galarraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. In *VLDB*, volume 24, pp. 707–730, 2015.

Luis Galárraga and Fabian M. Suchanek. Towards a numerical rule mining language. *AKBC*, 2014.

Alberto García-Durán and Mathias Niepert. Kblrn: End-to-end learning of knowledge base representations with latent, relational, and numerical features. In *UAI*, pp. 372–381, 2018.

Vinh Thinh Ho, Daria Stepanova, Mohamed H. Gad-Elrab, Evgeny Kharlamov, and Gerhard Weikum. Rule learning from knowledge graphs guided by embedding models. In *ISWC*, pp. 72–90, 2018.

Katsumi Inoue and Yoshimitsu Kudoh. Learning extended logic programs. In *IJCAI (1)*, pp. 176–181. Morgan Kaufmann, 1997.

Mark Law, Alessandra Russo, and Krysia Broda. The complexity and generality of learning answer set programs. *AIJ*, 259:110–146, 2018.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6: 167–195, 2015.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *NeurIPS*, pp. 3753–3763, 2018.

Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 3137–3143, 2019.

Stephen Muggleton. Inductive logic programming: Inverse resolution and beyond. In *IJCAI (1)*, pp. 997. Morgan Kaufmann, 1995.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, pp. 809–816, 2011.

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

Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Rudik: Rule discovery in knowledge bases. *PVLDB*, 11(12):1946–1949, 2018b.

Luc De Raedt. Multi-relational data mining. In *Encyclopedia of Machine Learning and Data Mining*, pp. 892–893. Springer, 2017.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NeurIPS*, pp. 3791–3803, 2017.

Celine Vens, Jan Ramon, and Hendrik Blockeel. Refining aggregate conditions in relational learning. In *PKDD*, volume 4213 of *Lecture Notes in Computer Science*, pp. 383–394. Springer, 2006.

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

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*, pp. 2316–2325, 2017.

Wen Zhang, Bibek Paudel, and *et al.* Liang Wang. Iteratively learning embeddings and rules for knowledge graph reasoning. In *WWW*, pp. 2366–2377. ACM, 2019.

A Interpretation as Rules with External Computations

We have presented our approach for learning rules that support numerical comparison among entities, and have also highlighted the algorithmic and numerical approaches to handling such rules. In the general case such features are inlined with *logic rules* that allow for *external computations*, *i.e.* "oracle programs" that might appear in the body of the rules as part of a generic black-box computation. In the case of simplest numerical comparisons, for example, the corresponding oracle may check, *e.g.*, whether one quantity is larger than another or aggregate the numerical attributes for an entity, *e.g.*, count the number of citations a person has and compare to the number of citations that another person has. The challenge of learning rules with aggregates is a central one in much work on KGs, and our proposed procedure is the first approach that is capable of learning such rules (even though we of course do not allow for arbitrary aggregates, the fact that we incorporate such learning at all is a substantial contribution to the KG literature). Because this is a key capability of our approach, in this section we further highlight the perspective of our approach as a method for learning such aggregate rules. The notation and definitions here are not crucial for understanding the remainder of the paper, but will provide context and clarify the contribution of the current work from the perspective of the KG and knowledge base communities.

A.1 Rules with Aggregates

We consider aggregates of the form $f\{Y\colon p(X,Y)\} \circ Z$, where f is an aggregate function symbol, $\{Y : p(X,Y)\}$ is called an aggregate element, $\circ \in \{<,>,=,\leq,\geq\}$ and Z is either a variable or a numerical value. For any aggregate in the above form we assume that if $p \notin \mathcal{R}_n$, then $f = \texttt{count}$, otherwise $f \in \mathcal{A} = \{\texttt{count}, \texttt{sum}, \texttt{max}, \texttt{min}, \texttt{avg}\}$. For example the aggregate element $\{Y\colon \textit{citedIn}(X,Y)\}$ denotes the set of all Y in which X is cited, and the aggregate $\texttt{count}\{Y\colon \textit{citedIn}(X,Y)\} > 5$ checks whether the number of X's citations exceeds 5.

A rule is an expression of the following form

$$h \leftarrow b_1 \wedge \cdots \wedge b_k \wedge \textit{not}\, b_{k+1} \wedge \cdots \wedge \textit{not}\, b_m \ , \tag{4}$$

where every $h, b_{k+1}, \ldots, b_m$ is an *atom* (*e.g.*, $b(X,Y)$), and every $b_1, \ldots, b_k$ is either an *atom* or an *aggregation conjunction* of the form $b^{\circ}_{pq}(Y,Z)$:

$$b^{\circ}_{pq}(Y,Z) \leftarrow p(Y,V) \wedge f\{V' : p(Y,V')\} = N \wedge q(Z,U) \wedge f\{U' : q(Z,U')\} = M \wedge N \circ M \ .$$

The right-hand side of the rule (4) is the conjunction of atoms $\textit{body}_r$ called the rule body; $\textit{body}^+_r = \{b_1, \ldots, b_k\}$ and $\textit{body}^-_r = \{b_{k+1}, \ldots, b_m\}$. A rule r is called *Horn rule*, if $\textit{body}^-_r r = \emptyset$, and $\textit{body}^+_r$ contains only standard atoms. We focus on rules, in which every variable must appear in two different atoms, one of which is in $\textit{body}^+_r$.

The following example of a rule with an aggregate presented below states that a patent is relevant for a person if it has a higher average rating than another relevant patent having common keywords with the former. Such rule contains an aggregate computing the average rating of a patent, and it is formally represented as follows:

$$\begin{aligned} \textit{relevant}(X,Z) \leftarrow & \textit{hasRating}(Y,V) \wedge \texttt{avg}\{V'\colon \textit{hasRating}(Y,V')\} = N \wedge \\ & \textit{hasRating}(Z,U) \wedge \texttt{avg}\{U'\colon \textit{hasRating}(Z,U')\} = M \wedge N \leq M \\ & \textit{relevant}(X,Y) \wedge \textit{hasCommonKeywordsWith}(Y,Z) \ . \end{aligned}$$

The execution of rules with default negation and aggregates (Faber et al. (2011)) over KGs is defined in the standard way. More precisely, let $\mathcal{G}$ be a KG, r a rule over $\mathcal{G}$, and a be an atom from $\mathcal{G}$. Then, $r \models_{\mathcal{G}} a$ holds if there is a variable assignment that maps atoms in $\textit{body}^+_r$ in $\mathcal{G}$ such that it does not map any of the atoms in $\textit{body}^-_r$ in $\mathcal{G}$. $\mathcal{G}_r = \mathcal{G} \cup \{a \,|\, r \models_{\mathcal{G}} a\}$ extends $\mathcal{G}$ with predicates derived from $\mathcal{G}$ by applying r. Note that to avoid propagating uncertain predictions, given a set of rules R we execute every rule in R on $\mathcal{G}$ independently, *i.e.* $\mathcal{G}_R = \bigcup_{r \in R} \mathcal{G}_r$. Given additional syntactic restrictions on rules in R, which disallow cycles through negation and recursive aggregates, consistency is ensured.