Supervised Knowledge Aggregation for Knowledge Graph Completion

Patrick Betz, Christian Meilicke, and Heiner Stuckenschmidt

University of Mannheim, Germany
{patrick,christian,heiner}@informatik.uni-mannheim.de

Abstract. We explore data-driven rule aggregation based on latent feature representations in the context of knowledge graph completion. For a given query and a collection of rules obtained by a symbolic rule learning system, we propose end-to-end trainable aggregation functions for combining the rules into a confidence score answering the query. Despite using latent feature representations for rules, the proposed models remain fully interpretable in terms of the underlying symbolic approach. While our models improve the base learner constantly and achieve competitive results on various benchmark knowledge graphs, we outperform current state-of-the-art with respect to a biomedical knowledge graph by a significant margin. We argue that our approach is in particular well suited for link prediction tasks dealing with a large multi-relational knowledge graph with several million triples, while the queries of interest focus on only one specific target relation.

1 Introduction

Knowledge graphs (KGs) represent structured knowledge in form of (subject, relation, object) facts. As KGs quite frequently suffer from missing data, the goal in link prediction or knowledge graph completion (KGC) is to predict new facts given an incomplete KG. While a vast amount of research is centered around knowledge graph embedding (KGE) models (e.g., [1,34]), rule-based approaches remain competitive in terms of performance [33]. They are fully interpretable as predictions are made by human-understandable symbolic rules and they easily allow for encoding domain knowledge into the overall model pipeline (e.g., [37]).

KGs are heavily used in the biomedical domain [11,18,25] in conjunction with general semantic web technologies to provide large linked data sources and ontologies such as Bio2RDF [2] and Hetionet[16]. These sources can be utilized for downstream tasks such as predictive diagnosis and processing KGs in the biomedical domain can differ from general KGC benchmark datasets. While the KGs may contain a substantial amount of relations, only one particular target relation might be of interest. The challenge is then to exploit the remaining graph context effectively, for instance, guiding models to exploit long-range dependencies while only providing supervision for the target relationships. For example, in the drug repurposing problem on the Hetionet dataset, we seek to find treatable diseases for given compounds by answering queries with respect to the target relationship *Compound-treats-Disease*. The remaining relations in the KG such as *Compound–binds–Gene* or *Disease–associates–Gene* affect the evaluation scheme of the task only by their usefulness in regard to predicting the target relationship correctly.

KGE models are not interpretable, which is an important aspect in general and has even higher relevance in the biomedical domain. They have shown to perform worse when long-range dependencies need to be utilized [19]. Path-based methods, on the other hand, are specifically tailored towards utilizing graph context that exceeds one-hop neighbourhoods and indeed, the neuro-symbolic model PoLo [18] is shown to be effective on the Hetionet KG. By using reinforcement learning agents are trained to perform policy-based walks and are additionally guided by logical rules. However, the model can only process cyclical rules and it has been shown empirically that more specific types, i.e., rules containing constants, are necessary for achieving results on-par with state-of-the-art models [20,21] in the context of KGC.

In this work, we first employ the simple rule learner AnyBURL [20,21]. We mine knowledge in the form of logical rules from the biomedical KG and find that a simple aggregation baseline already outperforms the current state-of-the-art. We then seek to improve the performance further by improving the aggregation of the mined rules. We formulate the problem as data-driven and aim to learn the aggregation from the training data. We propose a novel aggregator with a strong inductive bias, which we call the sparse aggregator, that generalizes the standard aggregation functions by using latent feature representations. We suggest to train the sparse aggregator by using black-box optimization [27,29,32] and directly optimize the mean-rank on the training KG. Furthermore, we propose a simple scaling scheme to reduce the variance of the gradients which improves the performance. Finally, we experiment with a more complex model based on a modified self-attention encoder [10], which we call the dense aggregator.

We find that the sparse aggregator improves the base performance of AnyBURL in all our experimental settings and outperforms current state-of-the-art [18] on the biomedical KG. We also present results on three standard KG benchmarks, where we achieve competitive results. Finally, we demonstrate that the sparse aggregator remains fully interpretable even though it uses latent feature representations. [1]

2 Related work

A KGE model is specified by a scoring function which outputs raw triple confidences and the models are trained by using likelihood based loss functions such as cross-entropy. A seminal model in the family of translation based models is TransE [4]. RESCAL [26] is based on tensor products and is extended by ComplEx [40] towards expressing asymmetric relations. Many alternative KGE models exist, for instance based on convolutional neural networks or graph-convolutions [9,41]. We refer to the studies in [46,33] for a more comprehensive

[1] The project repository and code can be found at this URL.

overview. KGE models achieve strong performances in the field of KGC and are efficient to use, however, their predictions are in general hard to interpret which can be of specific interest depending on the respective target domain. This gives rise to symbolic learning where predictions can be pinned to human-understandable rules.

Symbolic rule learning for larger sized KGs is introduced in [13,14] who propose to learn closed connected rules and improved over inductive logic programming systems in terms of scalability and performance. This type of rule learning is outperformed by RuleN [22], the predecessor of the AnyBURL system [20,21], which achieves results that are competitive to the state-of-the-art in the context of KGC. More recent approaches focus on differentiable rule learning [7,45] by representing rules as chained TensorLog [7] operators. An unsupervised approach based on the rulesets from AnyBURL is proposed in [28]. Rules are clustered by calculating the Jaccard index for every possible rule pair. Subsequently, the rules are aggregated using noisy-or aggregation. In the context of multi-modal KGE models, rules are used as features in [15] where feature weights are trained jointly in a product-of-experts scoring function consisting of different modalities.

Symbolic representations in form of rules are also used in the context of neuro-symbolic learning where the logical inference procedure is relaxed into fuzzy-logic formulations and not only the learning but also the application of rules is made differentiable. In [12], forward chaining is formulated to be differentiable and [31] relax backward chaining in Prolog by introducing Neural Theorem Provers which are further improved towards efficiency and flexibility in [23,24]. Please note that these models have not yet shown to be scalable towards KGs with a comparable size as used in this work.

Path-based methods traverse the graph starting from an entity and with sequential transitions to neighbouring nodes. Transitions are made by agents guided by stochastic policies which are learned within a reinforcement learning framework. This is applied to perform triple classification in [44] and extended towards query answering in MINERVA [8]. Finally, PoLo extends these approaches by enabling to inject rules learned by an external system [18] or given from domain knowledge and applies the approach to the drug repurposing problem.

3 Rule-based Knowledge Graph Completion

3.1 Preliminaries

A KG is a collection of (s, p, o) triples where s and o are entities while p is a relation. The KG forms a graph with relations represented as directed edges from s to o. In the context of KGC s is also called the head of the triple and o is called its tail. Models for KGC are concerned with predicting missing information in the KG, typically by answering queries of the form *(s, p, ?)* and *(?, p, o)*. These queries are answered by scoring a set of candidates which allows to derive a ranking. Training a model takes place on a training set $\mathcal{K}^{train}$ while a validation set and a test set are used for the evaluation. A common form of evaluating KG models is by calculating ranking based metrics on the respective evaluation set. The joint mean-rank is the average rank a model assigns to the true candidates when forming queries *(s, p, ?)* and *(?, p, o)* from all triples in the respective evaluation set. The mean reciprocal rank (MRR) takes the inverse $\frac{1}{rank_i}$ of the ranks and therefore lies in $[0, 1]$. The standard procedure is to filter the rankings with known triples from the remaining sets before calculating final ranks [4].

3.2 AnyBURL

AnyBURL [21] is a rule learning approach based on sampling paths from a given knowledge graph. These paths are generalized into rules by replacing constants with variables. The first edge in the path results into the head of the rule and the remaining edges yield the body of the rule. As edges in the graph represent relations a rule is composed of a set of non-grounded or partly grounded triples. AnyBURL is applicable to large datasets and can mine a large number of rules within a short period of time. We abstain from a more detailed description of the learning algorithm and refer to the respective publication. In the following, we briefly review the most relevant rule types AnyBURL learns and demonstrate how they are aggregated to create predictions.

Similar to related approaches, AnyBURL learns closed connected rules [14] also termed cyclic rules [21]. Here cyclic means the head variables X and Y which are connected by the target relation in the head of the rule are also connected via an alternative path represented by the body of the rule. These rules exclusively contain variables and we will give an example from the Hetionet KG in the following. Consider the rule $CtD(X, Y) \leftarrow \_PCiC(X, A) \wedge PCiC(A, B) \wedge CtD(B, Y)$. The rule is mined by AnyBURL and expresses with a certain confidence that a disease Y might be treated (CtD) with compounds which belong to the same pharmacological class (PCiC) as other compounds that are known to treat Y. We define the *confidence* of a rule as the number of body groundings that lead to a true prediction divided by the number of all body groundings estimated on the training data. Further examples for cyclical rules are given by Rule (R1) and (R2), discussed in Section 4.1.

Another rule type is given by acyclic rules with only one variable and constants, i.e., entities in the KG, at the remaining slots. AnyBURL is restricted in its default setting, which we used in our experiments, to learn rules of this type with only one body atom. The rule $citizen(X, UK) \leftarrow bornIn(X, London)$ is an example. It expresses that someone born in London is (probably) a UK citizen.

3.3 Knowledge Aggregation

Let us assume we are given a query $(s, p, ?)$,[2] a possible candidate answer c, and the set of rules $R_c := \{r_1, ..., r_k\}$ generating c, hence, all the rules that *fired* for the candidate. Note that R_c depends on the particular query but we do not reference this separately for brevity.

[2] All derivations throughout the work are equivalent for the head/subject direction.

The chosen type of rule aggregation defines how the associated rule confidences $conf(r_1), ..., conf(r_k)$ are aggregated into a score ψ. This score can be interpreted as the standalone confidence for the candidate representing a true answer or it can be used to generate a ranking in regard to multiple candidate answers to $(s, p, ?)$. The default aggregation in AnyBURL is max-aggregation:

$$\psi_{Max}(c) := \max \{conf(r_1), ..., conf(r_k)\}. \tag{1}$$

Potential ties in a ranking are resolved by comparing the respective candidates by their second strongest rule. Another common aggregation technique, which usually performs worse, is noisy-or aggregation:

$$\psi_{NO}(c) := 1 - \prod_{i=1}^{k} \big(1 - conf(r_i)\big). \tag{2}$$

In terms of restrictiveness, noisy-or and max-aggregation are placed on opposite ends of the spectrum which can be seen easily by comparing equations (1) and (2). Whereas only one rule determines the final score in max-aggregation, every single rule contributes when choosing noisy-or aggregation. The former is based on the underlying assumption that all rules are fully dependent, while the latter assumes rules to be mutually independent. Both assumptions are clearly wrong. Moreover, the score of noisy-or increases in the number of rules fired, *e.g*, it also increases when many rules with low confidences are added to the input rule set. These observation inspire the sparse aggregator presented in Section 5.

4 Supervised Knowledge Aggregation

4.1 Challenges

The full version of AnyBURL mines a large number of rules. For example, on the Hetionet KG it learns more than 40k rules for the target relation. Additionally, $|R_c|$ can be large, that is, a single candidate for a query might be generated by hundreds of rules which need to be aggregated. Two main additional characteristics cause the difficulty of the aggregation problem: varying rule set cardinalities and rule redundancy.

Different rule set cardinalities. As mentioned above the number of rules that generate a candidate answer, i.e., $|R_{c_i}|$ for some candidate c_i, can be large. However, this may depend strongly on the given candidate and it is also possible that only a few rules fired. This leads to the natural question of how to compare, for instance, a candidate for which only one strong rule fired to another candidate which was generated by multiple rules with lower confidences.

Rule redundancies. Many of the mined rules are dependent in the sense that they fire for similar reasons. When an aggregation method does not take into account these redundancies, it will overestimate the final score whenever multiple similar rules generated a candidate. Let us consider the following two rules:

$$speaks(A, B) \leftarrow lives(A, C) \wedge cityOf(C, D) \wedge hasLanguage(D, B) \tag{R1}$$

$$speaks(A, B) \leftarrow lives(A, C) \wedge locatedIn(C, D) \wedge hasLanguage(D, B) \tag{R2}$$

While the rules are not identical, they are partly redundant. The second rule does not provide much more additional evidence if we knew already that the first rule generated a candidate. Noisy-or aggregation would simply treat both rules as independent and overestimate the final score, while max-aggregation would ignore one of the rules which might be the better choice in this example. However, ignoring all rules except of the strongest rule will underestimate the final score whenever a candidate is generated by different rules which represent independent knowledge.

Note that often there is no schema available that contains the knowledge that *cityOf* is more specific than *locatedIn*, nor can this knowledge be derived from the given knowledge graph which is usually incomplete and noisy. Thus, it is not possible to filter out redundant rules. Moreover, in many cases the dependencies are less clear-cut. Think, for example, about replacing the *lives* relation by the *diedIn* relation in Rule (R1) or Rule (R2).

4.2 Supervised Rule Aggregation

We emphasized the challenges of rule aggregation in the last section. The goal of this work is to investigate if a data-driven view that utilizes supervision on the training KG can be beneficial for making a step towards solving the problem. We will introduce a formal perspective in the following paragraph and in the next section the respective aggregation models will be presented.

Given a KG, $\mathcal{K}^{train} = \{(s_i, p_i, o_i)\}_{i=1}^{N}$, let $\mathcal{S}^p$ be the set of rules that were mined for relation p, i.e., all rules mined with relation p in the rule head. Let $\mathcal{P}(\mathcal{S}^p)$ be the power set. Using the definitions from above, for a query $(s, p, ?)$, one possible candidate answer c, and the set of rules R_c that generated the candidate, we have that $R_c \in \mathcal{P}(\mathcal{S}^p)$. For supervised rule aggregation, we seek to learn the parameters Θ of an aggregation function f_Θ with signature $f_\Theta : \mathcal{P}(\mathcal{S}^p) \rightarrow \mathbb{R}$ by minimizing a loss criterion $L(\Theta) := \sum L_{q_j}(\Theta)$ which is the sum of all individual losses for the queries q_j that can be formed from $\mathcal{K}^{train}$.

The aggregation function takes as input the subset of rules R_c that generated c and outputs a real-valued score which we interpret as the confidence for the candidate being a true answer. In the definition f_Θ is parameterized globally over relations, however, the question of parameter sharing between relations is a modeling decision and we will argue that it might be beneficial to not share parameters. Furthermore, we will in the following sections inject a strong inductive bias into f by using the rule confidences as an additional input which we omitted for brevity in the formal treatment above.

5 Latent-based Aggregation

When viewing the problem as data-driven, a possibility would be to propositionalize the data, i.e., define a binary feature for every rule which is one if the rule fired for a particular candidate and zero otherwise. We could then train a simple discriminative model on $\mathcal{K}^{train}$. However, this model would not take into account rule dependencies and, more importantly, for many relations such as the target relation of the Hetionet KG, the number of rules exceeds the number of triples. Such a model is therefore not applicable as the candidates could be perfectly separated in the feature space without learning anything useful. Therefore, we choose an implicit feature representation over an explicit one by encoding rules with latent representations while assigning a strong inductive bias to our model in form of the rule confidences.

5.1 Sparse Aggregation

The two aggregation techniques introduced in Section 3.3 make two opposing assumptions which are both permanently violated in the data as we discussed. In the next paragrahs, we present the *sparse* aggregator which uses more rules for the final confidence calculation than max-aggregation but it uses less rules than noisy-or aggregation while also being able to represent rule redundancies. In fact, when we set the latent input dimensionality to one, we recover max-aggregation wheres a larger dimensionality leads to a behavior closer to noisy-or. Furthermore, our formulation enables to learn the parameters on the KG, i.e., we can utilize the whole training set.

We encode rules with latent feature vectors $\mathbf{x} \in \mathbb{R}^d$ where d is the dimensionality. For a query $(s, p, ?)$, a candidate answer c, and the k rules R_c that generated the candidate, let $\mathbf{x}_j \in \mathbb{R}^d$ be the latent representation of rule r_j and let $\mathbf{v} \in [0,1]^k$ be the confidence vector of the k rules, i.e., $\mathbf{v}_j = conf(r_j)$. We first normalize the latent vectors with the `SoftMax` function such that each vector sums to one. Subsequently, we multiply the normalized vectors with the respective rule confidence. Define the normalized vector multiplied with its confidence as

$$\mathbf{x}_j^* := \texttt{SoftMax}(\mathbf{x}_j) \cdot \mathbf{v}_j, \tag{3}$$

where $\mathbf{v}_j \in [0,1]$ and $\mathbf{x}_j^* \in [0,1]^d$. Subsequently, we apply the `Max` operator over the rules, that is, row-wise over the columns of the stacked matrix $\mathbf{X}^* \in [0,1]^{(d,k)}$

$$\mathbf{s} := \texttt{Max}(\mathbf{X}^*), \tag{4}$$

where $\mathbf{s} \in [0,1]^d$, i.e., $\mathbf{s}$ has the same dimensionality as each of the latent input vectors. It is important to note that only one single rule can contribute to a single entry of $\mathbf{s}$ due to the `Max` operator. Finally, we calculate the final score ψ for the candidate c by using the noisy-or product,

$$\psi(c) := 1 - \prod_{i=1}^{d} (1 - \mathbf{s}_i). \tag{5}$$

The derived expression for ψ can be differentiated approximately with respect to the latent features by using automatic differentiation frameworks.[3] We will now discuss some important properties of this aggregation function.

[3] The common gradient approximation for the `Max` function is, e.g., $\nabla max(y_1, y_2) = [1,0]$ for $y_1 > y_2$ and $[0,1]$ otherwise.

At most d rules can contribute to the final score. This follows from the fact that the `Max` function is taken over rules, i.e., for every row of $\mathbf{X}^*$ only one rule is considered and there exist d rows. We mentioned in the previous sections that hundreds of rules can fire for one candidate, with max-aggregation only regarding one rule and noisy-or aggregation using all the rules for the final score. With the sparse aggregator, we can balance this value by setting d accordingly, for instance, for the Hetionet dataset we set d=10.

When we set d=1, we recover simple max-aggregation. To see this, for $d = 1$ we have that $x^{\cdot}_j = 1 \cdot \mathbf{v_j} = conf(r_j)$, therefore, $s = max\{conf(r_1), ..., conf(r_k)\}$ with $\psi(c) = 1 - 1 - conf(r_{max}) = conf(r_{max})$ where r_{max} is the rule with the highest confidence. This property is ensured by the use of the `SoftMax` function as it normalizes a single value to 1 which would not be the case for alternative functions such as Sigmoid.

Potentially we can recover noisy-or aggregation by setting $d = |\mathcal{S}^p|$, and enforcing the matrix of all latent vectors to be the identity matrix, however, the goal is to learn to select a small strong subset of rules for a particular query. In the training stage, we aim to learn the latent representations by optimizing a loss criterion which will be discussed in the next section.

5.2 Optimizing Mean-Rank Using Black-Box Optimization

With the sparse aggregator we can easily represent rule redundancies by assigning similar latent features to redundant rules, reducing their joint influence on the score. In general, our model is closer to a discrete model than many machine learning or deep learning models. For instance, the `SoftMax` normalization restricts the updates of any learning algorithm to be a re-distribution of the rule confidences instead of a clear fitting of the data.

Our choice of supervised machine learning is not for the sake of it. Framing the model as purely discrete would result in an unfeasible search. For instance, only restricting the values to lie within $\{0, 1\}$ (without the SoftMax) would result in a search space with $2^{|\mathcal{S}^p| \cdot d}$ possible configurations for one relation p.

Fortunately, there exists recent work in machine learning that bridges the gap between continuous and discrete problems such as combinatorial optimization [3,27,29]. It has been demonstrated that these approaches can be applied to ranking-based metrics as they can be written as combinatorial optimization problems [32]. Using such a metric suits our problem because the model is not designed to fit the data tightly. Note that also rules with high confidences can fire for candidates which do not exist in the KG. A ranking metric can ignore these possible distortions as soon as the true candidate is ranked relatively high.

To that end, we define the loss criterion $L(\Theta)$ on the training data to be the mean-rank where Θ represents the latent features. For a given query $q = (s, p, ?)$ on the training data, let $\boldsymbol{\psi}$ be the vector of scores calculated according to equation (5) with $\psi(c_*)$ being the score for the true candidate c_*. The remaining scores belong to query candidates c_i that were generated by at least one rule and (s, p, c_i) does not appear in $\mathcal{K}^{train}$, that is, we exclusively filter with the training set. Note that conceptually these candidates can also be viewed as *negative* examples. We treat the maximum number of these negative candidates, sorted according to their ranking in max-aggregation, as a hyperparameter denoted by *top-n*. Moreover, let $\boldsymbol{rk}(\boldsymbol{\psi})$ be the vector of ranks and define rk_{c*} to be the rank of the true candidate. The mean-rank of the training data is simply the mean of ranks of the true candidates for the individual queries, therefore, the individual query loss is $L_q(\boldsymbol{rk}; \Theta) = rk_{c*}$. For applying gradient-based optimization, we need to calculate $\frac{dL_q}{d\Theta}$. From the chain rule, we obtain

$$\frac{dL_q}{d\Theta} = \frac{d\boldsymbol{\psi}}{d\Theta}\,\frac{dL_q}{d\boldsymbol{\psi}}, \tag{6}$$

where $\frac{d\boldsymbol{\psi}}{d\Theta}$ can directly be calculated using auto-differentiation libraries as mentioned in the previous section. For $\frac{dL_q}{d\boldsymbol{\psi}}$ we use black-box differentiation according to [32] and calculate

$$\frac{dL_q}{d\boldsymbol{\psi}} = -\frac{1}{\lambda}\left[\boldsymbol{rk}(\boldsymbol{\psi}) - \boldsymbol{rk}\big(\boldsymbol{\psi} + \lambda \cdot \frac{dL_q}{d\boldsymbol{rk}}\big)\right], \tag{7}$$

where $\frac{dL_q}{d\boldsymbol{rk}}$ is a vector which is one at the entry of the true candidate and zero otherwise which follows from the definition of the query loss above. In practice during the forward pass, we obtain the scores $\boldsymbol{\psi}$ for the query and calculate the ranks. Subsequently, in the backward pass, the scores are perturbed to $\boldsymbol{\psi}' = \boldsymbol{\psi} + \lambda \cdot \frac{dL_q}{d\boldsymbol{rk}}$ and the new ranks are calculated leading to (7).

Consider the inner expression $\boldsymbol{rk}(\boldsymbol{\psi}) - \boldsymbol{rk}(\boldsymbol{\psi}')$ in equation (7) and let us only focus on the entry of the true candidate. Let's assume its rank is #50 before and #1 after the perturbation. Then we obtain $-\frac{49}{\lambda}$ which is in absolute terms magnitudes stronger compared to a case where the true candidate only improved by a few ranks. We seek to reduce this variance by scaling the gradient in accordance to the true candidate, in particular we compute

$$\frac{dL_q}{d\boldsymbol{\psi}} = -\frac{1}{\lambda}\left[\boldsymbol{rk}(\boldsymbol{\psi}) - \boldsymbol{rk}(\boldsymbol{\psi}')\right]\frac{1}{rk_{c*} - 1}, \tag{8}$$

that is, we track the rank of the true candiate and scale the gradient with a proportional factor. This ensures a constant signal strength independent of the original position, for instance, in the example above we obtain $-\frac{49}{\lambda}\frac{1}{50-1} = -\frac{1}{\lambda}$.

5.3 Dense Aggregation

In the previous sections, we presented a data-driven perspective on knowledge aggregation and proposed a model based on latent features which has a strong inductive bias. We can relax this restriction and exploit the fact that language model architectures (e.g. [10,17,42]) are well calibrated for processing latent representations. In particular, we use a self-attention based architecture [10] that processes the input representations and outputs real-valued query confidences.

When dropping positional encodings, self-attention based architectures can be applied off-the-shelf on item set problems with varying input lengths.

Precisely, we use a slightly modified version of the PyTorch implementation of the BERT encoder [10]. The latent inputs $\{\boldsymbol{x}_1, ..., \boldsymbol{x_k}\}$ are fed into the encoder receiving the hidden representations $\{\boldsymbol{h}_1, ..., \boldsymbol{h}_k\}$ with same dimensionality. The modification that we implement affects how the self-attention is applied. We sort the inputs according to the rule confidences and we let the j'th rule only attend to the $j-1$ rules with higher confidence. This reduces the possible distraction that can be caused from many weak input rules towards the high-confident rules. For the aggregation, we use simple average pooling on the hidden representations and feed the resulting vector into a fully connected linear layer which outputs one final score. The aggregator is termed *dense* aggregator as every rule in the input set contributes to the final score.

The model is expensive to train and, on average, it is inferior to the sparse aggregator in terms of performance although we were not able to explore a large part of the hyperparameter space due to runtime considerations. The model is also not practical in more general terms, however, we are merely interested in the question if this model can learn specific aspects of the data which are hidden from the other models. To investigate this, we evaluate a joint model of the sparse and dense aggregators. We tune weights β_{sparse} and $\beta_{dense} = 1 - \beta_{sparse}$ per relation and direction on the validation set for discovering potential differences between the two models. To make sure that these differences are significant, we restrict β_{sparse} to lie in {0,1}. This setting will be denoted by D+S in the experimental section. It is not applicable to Hetionet where only one target relation exists.

6 Experiments

In the following we describe experiments for a specific biomedical KG and three benchmark KGs commonly used in the field of KGC. We are mainly interested in the question how our learnable aggregation compares to baseline aggregation functions and to other models that have a similar degree of explainability.

6.1 Datasets

The Hetionet network combines knowledge from a large amount of biomedical studies into a KG containing 47k typed entities and 2.24 million triples with 24 possible relations [16]. We focus on the task of drug repurposing, i.e., finding new use cases for existing compounds. In the Hetionet KG this is expressed by answering queries of the form $CtD(X, ?)$ where the relation CtD means *Compound-treats-Disease*. It is discussed in [18] that Hetionet significantly differs from other benchmark KG's, for instance exhibiting a higher average node degree. Moreover, while the dataset is of considerable size, the target relationship appears in only 755 facts which highlights the challenge of exploiting graph context without having strong supervision. We use the train, valid, and test splits according to the public documentation of [18].

Dataset	Entities	Relations	Triples
Hetionet	47,031	24	2,250,197
FB15k-237	14,505	237	310,116
WNRR	40,559	11	93,003
CoDEx-M	17,050	51	206,205

Table 1. Summary statistics.

We further evaluate our approach on three general benchmark KGs. FB15k-237 [39] and WNRR [9] are frequently used in the field of KGC and Codex-M was designed with the goal to be more challenging than previous benchmarks [36]. For these datasets we use the common train, valid, and test splits. Table 1 shows summary statistics for the KGs.

6.2 Experimental Settings

For the Hetionet KG we mostly focus on the comparisons in [18]. That is, we compare to the interpretable models PoLo [18], MINERVA [8], and pLogicNet [30] and we also include the KGE results presented in [18] for the models TransE [4], ComplEx [40], ConvE [9] and RESCAL [26]. Furthermore, we include the HittER [6] no-context model implementation of the libKGE library for which we run the hyperparameter search provided by the library with 15 trials. We also include the RotatE [38] results from the TorchDrug library.

For the remaining KGs we additionally add the rule-based methods DRUM [35] and Neural-LP [45] if results are available. We abstain from a comprehensive comparison against KGE but we include the results from the official libKGE library and the underlying work about training KGE models [5,34] to put our approach in the context of strong KGE models trained under a well-tested and unified codebase. Moreover, the results for HittER$_{nc}$ are based on [19] and we also add the results of the recent model M^2GNN proposed in [43].

For all the datasets, we mine rules on the training splits of the KGs using AnyBURL and subsequently we learn the aggregation functions on the same training sets while utilizing the valid sets for hyperparameter search and early stopping. We train the sparse aggregator on the mean-rank on the training data using the scaled gradients. The dense aggregator is trained on a standard cross-entropy loss, i.e., maximizing the likelihood of the training data while using negative examples. The sparse aggregator does not share parameters between relations, that is, queries for different relations can be treated independently and hyperparameters could be searched relation-wise. For the sake of simplicity, we train the model globally, however, we save checkpoints per epoch and pick for every relation in head and tail direction the checkpoint that results in the highest MRR on the valid set. For the Hetionet KG this is not necessary as only one target relation exists and only tails are predicted. Hyperparameter configurations and further training details can be found in Appendix A.

Approach	h@1	h@3	h@10	MRR
TransE [4]	0.099	0.199	0.444	0.205
ComplEx [40]	0.152	0.285	0.470	0.250
ConvE [9]	0.100	0.225	0.318	0.180
RESCAL [26]	0.106	0.166	0.377	0.187
HittER$_{nc}$	0.316	0.517	0.740	0.453
RotatE	0.185	0.282	0.403	0.257
CompGCN [41]	0.172	0.318	0.543	0.292
pLogicNet [30]	0.225	0.364	0.523	0.333
MINERVA[8]	0.264	0.409	0.593	0.370
PoLo [18]	0.314	0.428	0.609	0.402
PoLo (pruned)	0.337	0.470	0.641	0.430
Max	0.272	0.444	0.642	0.398
Noisy-or	0.377	0.509	0.642	0.472
Dense	0.306	0.514	0.701	0.441
Sparse	0.380	0.525	0.694	0.490

Table 2. Filtered MRR in tail direction for Hetionet. The results for the learnable aggregators are averages over 5 runs. The first, middle and last part represents embedding-based models, interpretable models, and the results of this work, respectively.

6.3 Results

Table 2 shows results on the test set for the filtered MRR and filtered hits@k in tail direction for the drug repurposing problem and Table 3 shows results for the test sets of the joint MRR and joint hits@k on the remaining datasets.

On the Hetionet dataset, the sparse aggregator improves the recent state-of-the-art [18] of the interpretable models by 4.3, 5.5, 5.3, and 6 percentage points for the metrics hits@1, hits@3, hits@10, and MRR, respectively. Interestingly, the noisy-or baseline already outperforms the state-of-the-art by a significant margin. Noteworthy, the sparse aggregator improves AnyBURL's max-aggregation by 9.2 percentage points for the MRR and shows improvements of 1.8 percentage points over noisy-or. The dense aggregator, on the other hand, performs significantly worse except for the hits@10 metric. Despite outperforming the previous models, it does not improve over the noisy-or baseline.

For the remaining datasets the sparse aggregator outperforms the interpretable methods and improves over the best AnyBURL baseline (note that noisy-or performs rather poor on these datasets) in all settings although the improvement is only marginal for the WNRR dataset. However, improvements of 2.3 percentage points on FB15k-237 and 2 percentage points on Codex-m are considered to be quite substantial in the KGC literature. Interestingly, the setting D+S which is explained in Section 5.3 achieves strong results on WNRR and Codex-M. This suggests that, despite being inferior overall, the dense aggregator captured some aspects of the data which are hidden from the sparse aggregator. We investigated this further and found that the dense aggregator is sensitive to

	FB15k-237			WNRR			Codex-M		
Approach	h@1	h@10	MRR	h@1	h@10	MRR	h@1	h@10	MRR
TransE	0.221	0.497	0.312	0.053	0.520	0.228	0.223	0.454	0.303
ComplEx	0.253	0.536	0.347	0.438	0.547	0.475	0.262	0.476	0.337
ConvE	0.248	0.521	0.338	0.411	0.505	0.442	0.239	0.464	0.318
RESCAL	0.263	0.541	0.355	0.439	0.517	0.467	0.244	0.456	0.317
HittER$_{nc}$	0.268	0.549	0.361	0.437	0.531	0.469	0.262	0.486	0.339
RotatE	0.240	0.522	0.333	0.439	0.553	0.478	-	-	-
M^2GNN	0.275	0.565	0.362	0.444	0.572	0.485	-	-	-
pLogicNet	0.237	0.524	0.332	0.398	0.537	0.441	-	-	-
MINERVA	0.217	0.456	0.293	0.413	0.513	0.448	-	-	-
DRUM [35]	0.255	0.516	0.343	0.425	0.586	0.486	-	-	-
Neural-LP [45]	-	0.362	0.240	0.371	0.566	0.435	-	-	-
Max	0.246	0.506	0.331	0.457	0.572	0.497	0.247	0.450	0.316
Noisy-or	0.247	0.494	0.329	0.391	0.559	0.446	0.218	0.427	0.289
Dense	0.245	0.510	0.335	0.466	0.587	0.507	0.261	0.465	0.331
Sparse	0.266	0.526	0.352	0.459	0.574	0.499	0.266	0.467	0.335
D+S	0.267	0.527	0.354	0.469	0.593	0.511	0.273	0.476	0.342

Table 3. Results for FB15k-237, WNRR and Codex-M for the joint filtered MRR. The first, middle and last part of the table represents embedding-based models, interpretable models, and the results of this work, respectively. The results for *Dense* and *Sparse* are averages over 3 runs.

negative signals. For instance, when adding rules with low confidences to an input set, the score of the dense aggregator might decrease. This behavior cannot be expressed by the sparse aggregator or the presented baselines which might open up interesting further directions.

Finally, Figures 1 and 2 compare the sparse aggregator under different training settings. Figure 1 shows test results for five runs on Hetionet when using the scaled gradients proposed in Section 5 and the default formulation. Training with the scaled gradients leads to lower variance and higher average performance. Figure 2 compares mean-rank training with using a standard cross-entropy loss on Codex-M.

7 Interpretability

In the following we discuss an example for a query where the sparse aggregator generates a ranking that differs significantly from the ranking suggested by the rule with the highest confidence. Moreover, in contrast to noisy-or where all rules that fired contribute to the score, by setting d=10 our model selects a compact subset of 10 rules of which we can further pick the ones with the highest impact. This procedure also resembles how a potential user can interact with the aggregation system. We use an example with short rules for the sake of simplicity.

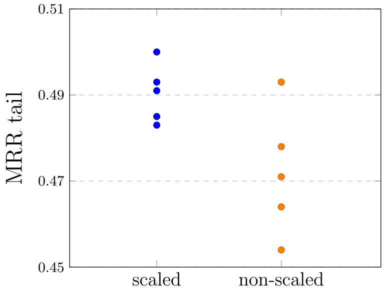

Fig. 1. MRR in tail direction for five runs on Hetionet when training under the scaled/non-scaled gradient.

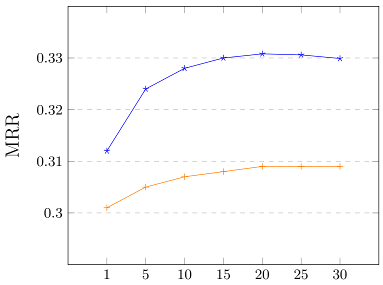

Fig. 2. Valid MRR per epoch on Codex-M for Mean-Rank training (⋆) and ordinary Cross-Entropy loss (+).

The target query asks for new diseases to which the compound *Methotrexate* (`DB00563`) can be applied. Our method ranks the correct answer *systemic lupus erythematosus* first, an autoimmune disease in which the body's immune system mistakenly attacks healthy tissue in many parts of the body.

To illustrate this example, we have chosen the eight rules with the highest confidences and depicted their normalized values for each of the ten dimensions side by side on the left part of Figure 3. Different rules can be distinguished by their color and position within the group of bars that reflects the value of a specific dimension. Note that the most confident rule that created a candidate for the query had a confidence of 0.357, while the #1 candidate proposed by our method received a score of 0.806. This is caused by the fact that the rules that recited this candidate differ significantly with respect to their latent features. There are several dimension (in particular 5,6, 8 and 9), where we learned a significantly higher value for that specific rule compared to all other rules. This means at the same time that a large fraction of the overall power of this rule is assumed to be independent from the other rules and increases the overall score. If the rules had similar values in most of the dimensions, the resulting score would be close to the score of the maximum strategy.

A domain expert, who wants to understand why a certain candidate is ranked at #1, is probably interested in a small set of rules that had the highest influence on the prediction. These are obviously the rules that dominate in some of the dimensions. To quantify their contribution we computed for each rule the sum of values over all dimensions taking only those dimension into account where the rule received the maximal value compared to the other rules. The resulting scores are depicted on the right part of Figure 3. This means that rules r_{229} and r_{264} are the most important two rules. Let us take a closer look at these rules and their meaning:

r_{229} `CtD(X,Disease::DOID:9074) <= CtD(X,Disease::DOID:7148)` - If a compound can be used as treatment for *rheumatoid arthritis* (`DOID:7148`), it might also be applied to treat the disease *lupus erythematosus* (`DOID:9074`).

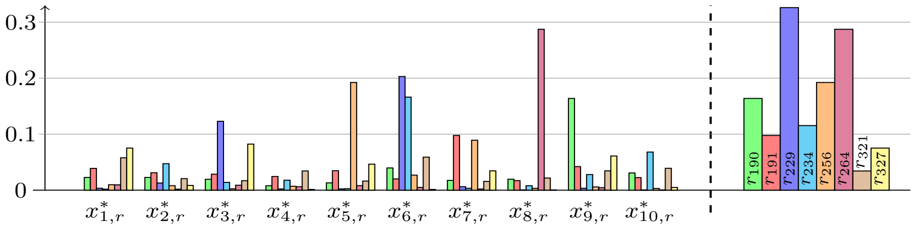

Fig. 3. Rule features when searching new treatments for *Methotrexate*.

Since *Methotrexate* is known to been used for *rheumatoid arthritis*, *lupus erythematosus* is predicted by the rule.

r_{264} `CtD(Compound::DB00563,Y) <= DlA(Y,Anatomy::UBERON:0000043)` - If a disease is diagnosed to affect a tendon (`UBERON:0000043`), that disease might be treated with *Methotrexate* (`DB0056`). Since *Methotrexate* is known to been used in such a context, it is predicted by the rule.

These two rules fire for different reasons. The first rule is based on observing that two different ailments which both correspond to autoimmune diseases are likely treatable by the same compound. The second rule focuses on a specific body part, the tendon, which – if afflicted by a disease – can often be treated with *Methotrexate*. In almost all cases tendons are either ruptured (no drug required) or inflamed which makes them treatable by *Methotrexate*. Figure 3 shows that our approach is capable to learn that these two rules are not redundant. It increases the score of a candidate that is proposed by both rules, which is in our case *lupus erythematosus*, a disease which can affect tendons and for which compounds have been applied that have also been applied to *rheumatoid arthritis*.

8 Conclusion

We showed empirically that simple rule learning approaches achieve strong results on the task of drug repurposing. We presented learnable knowledge aggregation in form of latent rule aggregation. To our best knowledge, this is a novel approach that differs fundamentally from symbolic and latent approaches proposed so far for KGC. In particular, we presented an aggregation function, the sparse aggregator, which can be learned on the training set and we proposed to employ black-box optimization. We found empirically that the sparse aggregator improves over baseline aggregation techniques. It is is on-par with standard KGE methods and is state-of-the-art on the Hetionet dataset, while still maintaining interpretability. The sparse model learns how to aggregate rules as positive evidence, however, it is not capable to learn that a rule or a combination of rules makes a prediction less likely. It might thus be beneficial to incorporate negative evidence into the model which opens up directions for future research.

A Experimental Details

A.1 Model Input

On the highest abstraction level, our models take as input a list of rules and output a real-valued score. More precisely, for a query $q = (s, p, ?)$ (same in head direction) we collect the *top_n* answer candidates c_i proposed by AnyBURL, that is, the candidates that were generated by at least one rule. For each of these candidates the respective list of rules defines the model input. Then, the descriptions of the main text apply. Finally, we obtain a vector of scores and likewise a ranking in regard to all candidate answers c_i. At test time, this ranking can directly be used for the evaluation. At training time, we distinguish the true answer/candidate c^* and the remaining candidates c' which we filter with the training set, i.e, we exclude a $c' \neq c^*$ if a triple (s, p, c') exists in train. For a ranking loss we can now calculate the query loss of q as explained in Section 5.2. For some arbitrary loss function such as cross-entropy, c^* defines the true candidate and the remaining candidates c' define the reference candidates or pseudo negative candidates.

A.2 Hyperparameters

For all the experiments, we use a max *top-n*=100, the Adagrad optimizer, and a batch-size of 256. Training is performed by using early stopping based on the validation set. LibKGE based configuration files for the experiments are provided in the supplementary materials.

Sparse aggregator. The hyperparameters that we are concerned with are dropout on the latent features, the latent dimension d, and the learning rate lr. For Hetionet we set d=10, dropout=0.15 and lr=0.9. On Fb15k-237 we set d=40, dropout=0.4, and lr=0.02. For WNRR we set d=50, dropout=0.4 and lr=0.03. For Codex-m we set d=40, dropout=0.4 and lr=0.02. For all the experiments we use a value of 5 for lambda when training on the mean-rank loss.

Dense aggregator. The dense aggregator follows in its architecture the PyTorch BERT encoder with the modification as explained in Section 5.3. We use 4 heads and 4 layers throughout all the experiments. The feed-forward dimensionality within the encoder is 256. For Hetionet we use d=20, dropout=0.15 and lr=0.01. For the remaining datasets we use d=56, dropout=0.15, and lr=0.005. We set the maximal number of rules per input list to 50 for the dense aggregator for all the experiments.

A.3 Rule sets

The base data for our experiments are the rules learned with AnyBURL. These are processed in a pre-processing pipeline to generate the inputs for the aggregators as explained above. For all the datasets we exclude AC2 rules and rules with an empty body. This leaves the AnyBURL performance mostly unchanged but we report newest AnyBURL results reported by the authors.

For WNRR rules are mined for 3600 seconds and we set the maximum length for cyclical rules equal to 5 as suggested in the AnyBURL documentation. All the learned rules are processed for training the models on this dataset. For the remaining datasets the default AnyBURL parameters are used. Here, we prune the learned rulesets slightly and only process rules that had at least 5 (10) true predictions for sparse (dense). On Hetionet rules are learned for 1000 seconds for dense and sparse. On Fb15k-237 rules are learned for 3600 (500) seconds for sparse (dense). Finally, on Codex-M rules are learend for 1000 (500) seconds for sparse (dense).

References

1. Ali, M., Berrendorf, M., Hoyt, C.T., Vermue, L., Galkin, M., Sharifzadeh, S., Fischer, A., Tresp, V., Lehmann, J.: Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework. IEEE Transactions on Pattern Analysis and Machine Intelligence (2021)
2. Belleau, F., Nolin, M.A., Tourigny, N., Rigault, P., Morissette, J.: Bio2rdf: towards a mashup to build bioinformatics knowledge systems. Journal of biomedical informatics **41**(5), 706–716 (2008)
3. Betz, P., Niepert, M., Minervini, P., Stuckenschmidt, H.: Backpropagating through markov logic networks. In: Proceedings of 15th International Workshop on Neural-Symbolic Learning and Reasoning. vol. 2986, pp. 67–81. CEUR (2021)
4. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Neural Information Processing Systems (NIPS). pp. 1–9 (2013)
5. Broscheit, S., Ruffinelli, D., Kochsiek, A., Betz, P., Gemulla, R.: Libkge-a knowledge graph embedding library for reproducible research. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 165–174 (2020)
6. Chen, S., Liu, X., Gao, J., Jiao, J., Zhang, R., Ji, Y.: Hitter: Hierarchical transformers for knowledge graph embeddings. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2020)
7. Cohen, W., Yang, F., Mazaitis, K.R.: Tensorlog: A probabilistic database implemented using deep-learning infrastructure. In: Journal of Artificial Intelligence Research. vol. 67, pp. 285–325 (2020)
8. Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., McCallum, A.: Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. arXiv preprint arXiv:1711.05851 (2017)
9. Dettmers, T., Minervini, P., Stenetorp, P., Riedel, S.: Convolutional 2d knowledge graph embeddings. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32, pp. 1811–1818 (2018)
10. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (Jun 2019)

11. Dörpinghaus, J., Jacobs, M.: Semantic knowledge graph embeddings for biomedical research: Data integration using linked open data. In: SEMANTICS Posters&Demos (2019)
12. Evans, R., Grefenstette, E.: Learning explanatory rules from noisy data. In: Journal of Artificial Intelligence Research. vol. 61, pp. 1–64 (2018)
13. Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast rule mining in ontological knowledge bases with amie. The VLDB Journal **24**(6), 707–730 (2015)
14. Galárraga, L.A., Teflioudi, C., Hose, K., Suchanek, F.: Amie: association rule mining under incomplete evidence in ontological knowledge bases. In: Proceedings of the 22nd international conference on World Wide Web. pp. 413–422 (2013)
15. García-Durán, A., Niepert, M.: Kblrn: End-to-end learning of knowledge base representations with latent, relational, and numerical features. UAI (2018)
16. Himmelstein, D.S., Lizee, A., Hessler, C., Brueggeman, L., Chen, S.L., Hadley, D., Green, A., Khankhanian, P., Baranzini, S.E.: Systematic integration of biomedical knowledge prioritizes drugs for repurposing. Elife **6**, e26726 (2017)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. In: Neural computation. vol. 9, pp. 1735–1780. MIT Press (1997)
18. Liu, Y., Hildebrandt, M., Joblin, M., Ringsquandl, M., Raissouni, R., Tresp, V.: Neural multi-hop reasoning with logical rules on biomedical knowledge graphs. In: European Semantic Web Conference. pp. 375–391. Springer (2021)
19. Meilicke, C., Betz, P., Stuckenschmidt, H.: Why a naive way to combine symbolic and latent knowledge base completion works surprisingly well. In: 3rd Conference on Automated Knowledge Base Construction (2021)
20. Meilicke, C., Chekol, M.W., Fink, M., Stuckenschmidt, H.: Reinforced anytime bottom up rule learning for knowledge graph completion (2020)
21. Meilicke, C., Chekol, M.W., Ruffinelli, D., Stuckenschmidt, H.: Anytime bottom-up rule learning for knowledge graph completion. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI). IJCAI/AAAI Press (2019)
22. Meilicke, C., Fink, M., Wang, Y., Ruffinelli, D., Gemulla, R., Stuckenschmidt, H.: Fine-grained evaluation of rule- and embedding-based systems for knowledge graph completion. In: Proceedings of the International Semantic Web Conference. pp. 3–20. Springer International Publishing (2018)
23. Minervini, P., Bošnjak, M., Rocktäschel, T., Riedel, S., Grefenstette, E.: Differentiable reasoning on large knowledge bases and natural language. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 5182–5190 (2020)
24. Minervini, P., Riedel, S., Stenetorp, P., Grefenstette, E., Rocktäschel, T.: Learning reasoning strategies in end-to-end differentiable proving. In: International Conference on Machine Learning. pp. 6938–6949. PMLR (2020)
25. Mohamed, S.K., Nounu, A., Nováček, V.: Drug target discovery using knowledge graph embeddings. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. pp. 11–18 (2019)
26. Nickel, M., Tresp, V., Kriegel, H.: A three-way model for collective learning on multi-relational data. In: Getoor, L., Scheffer, T. (eds.) Proceedings of the 28th International Conference on Machine Learning. pp. 809–816. Omnipress (2011)
27. Niepert, M., Minervini, P., Franceschi, L.: Implicit mle: Backpropagating through discrete exponential family distributions. NeurIPS (2021)
28. Ott, S., Graf, L., Agibetov, A., Meilicke, C., Samwald, M.: Scalable and interpretable rule-based link prediction for large heterogeneous knowledge graphs (2020)

29. Pogančić, M.V., Paulus, A., Musil, V., Martius, G., Rolinek, M.: Differentiation of blackbox combinatorial solvers. In: International Conference on Learning Representations (2020)
30. Qu, M., Tang, J.: Probabilistic logic neural networks for reasoning. In: International Conference on Learning Representations (2020)
31. Rocktäschel, T., Riedel, S.: End-to-end differentiable proving. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017. pp. 3788–3800 (2017)
32. Rolínek, M., Musil, V., Paulus, A., Vlastelica, M., Michaelis, C., Martius, G.: Optimizing rank-based metrics with blackbox differentiation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7620–7630 (2020)
33. Rossi, A., Barbosa, D., Firmani, D., Matinata, A., Merialdo, P.: Knowledge graph embedding for link prediction: A comparative analysis. ACM Transactions on Knowledge Discovery from Data (TKDD) **15**(2), 1–49 (2021)
34. Ruffinelli, D., Broscheit, S., Gemulla, R.: You CAN teach an old dog new tricks! on training knowledge graph embeddings. In: International Conference on Learning Representations (2020)
35. Sadeghian, A., Armandpour, M., Ding, P., Wang, D.Z.: DRUM: end-to-end differentiable rule mining on knowledge graphs. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS 2019, Vancouver, BC, Canada. pp. 15321–15331 (2019)
36. Safavi, T., Koutra, D.: CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 8328–8350. Association for Computational Linguistics, Online (Nov 2020)
37. Sola, D., Meilicke, C., Aa, H.v.d., Stuckenschmidt, H.: A rule-based recommendation approach for business process modeling. In: International Conference on Advanced Information Systems Engineering. pp. 328–343. Springer (2021)
38. Sun, Z., Deng, Z.H., Nie, J.Y., Tang, J.: Rotate: Knowledge graph embedding by relational rotation in complex space. International Conference on Learning Representations (2019)
39. Toutanova, K., Chen, D.: Observed versus latent features for knowledge base and text inference. In: Proceedings of the 3rd workshop on continuous vector space models and their compositionality. pp. 57–66 (2015)
40. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning. JMLR Workshop and Conference Proceedings, vol. 48, pp. 2071–2080. JMLR.org (2016)
41. Vashishth, S., Sanyal, S., Nitin, V., Talukdar, P.: Composition-based multi-relational graph convolutional networks. In: International Conference on Learning Representations (2020)
42. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. Advances in Neural Information Processing Systems pp. 6000–6010 (2017)
43. Wang, S., Wei, X., Nogueira dos Santos, C.N., Wang, Z., Nallapati, R., Arnold, A., Xiang, B., Yu, P.S., Cruz, I.F.: Mixed-curvature multi-relational graph neural network for knowledge graph completion. In: Proceedings of the Web Conference 2021. pp. 1761–1771 (2021)
44. Xiong, W., Hoang, T., Wang, W.Y.: Deeppath: A reinforcement learning method for knowledge graph reasoning. arXiv preprint arXiv:1707.06690 (2017)

45. Yang, F., Yang, Z., Cohen, W.W.: Differentiable learning of logical rules for knowledge base reasoning. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NeurIPS 2017, Long Beach, US (2017)
46. Zhang, J., Chen, B., Zhang, L., Ke, X., Ding, H.: Neural, symbolic and neural-symbolic reasoning on knowledge graphs. AI Open **2**, 14–35 (2021)