Combining Embeddings and Rules for Fact Prediction

Armand Boschin
Télécom Paris, Institut Polytechnique de Paris, France

Nitisha Jain
Hasso Plattner Institute, University of Potsdam, Germany

Gurami Keretchashvili
Télécom Paris, Institut Polytechnique de Paris, France

Fabian Suchanek
Télécom Paris, Institut Polytechnique de Paris, France

Abstract

Knowledge bases are typically incomplete, meaning that they are missing information that we would expect to be there. Recent years have seen two main approaches to guess missing facts: Rule Mining and Knowledge Graph Embeddings. The first approach is symbolic, and finds rules such as "If two people are married, they most likely live in the same city". These rules can then be used to predict missing statements. Knowledge Graph Embeddings, on the other hand, are trained to predict missing facts for a knowledge base by mapping entities to a vector space. Each of these approaches has their strengths and weaknesses, and this article provides a survey of neuro-symbolic works that combine embeddings and rule mining approaches for fact prediction.

2012 ACM Subject Classification Information systems → Information systems applications

Keywords and phrases Rule Mining, Embeddings, Knowledge Bases, Deep Learning

Digital Object Identifier 10.4230/OASIcs.AIB.2022.4

Category Invited Paper

Funding This work was partially funded by ANR-20-CHIA-0012-01 ("NoRDF").

1 Introduction

A knowledge base (KB) is a computer-processable collection of knowledge about the world. KBs typically contain real-world entities (such as organizations, people, movies, or locations) and their relationships (who was born where, which movie plays where, etc.). Thousands of such KBs are publicly available, including, e.g., Wikidata [60], DBpedia [4], and YAGO [53]. These KBs contain millions of entities and relationships between them, saying, e.g., who was born in which city, which actor acted in which movie, or which city is located in which country. Such KBs are used for question answering, Web search, text understanding, personal assistants, and other AI applications [66].

KBs are usually never complete; there are always facts that are missing from the KB. This is due to the way in which KBs are constructed: Some of them are constructed automatically by extracting facts from Web sources. Such an extraction may fail to extract all information, and the underlying sources can be incomplete themeselves. Other KBs are fed by a community, and may be incomplete simply because not all facts have yet been added. *Fact prediction* is the task of predicting facts that are true in the real world, but missing in the KB. Although

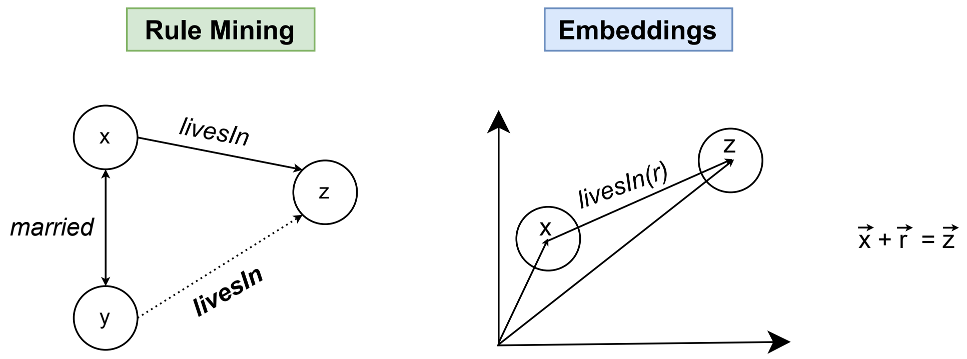

Figure 1 Rule Mining and Embeddings.

this may never make the KB complete, it will at least add facts that were missing. There are two major approaches to this end: Rule Mining and Knowledge Graph Embeddings. *Rule mining* is a symbolic approach. It finds rules such as the following in a KB:

$$\mathit{married}(x, y) \wedge \mathit{livesIn}(x, z) \Rightarrow \mathit{livesIn}(y, z)$$

This rule means that if some person x is married to some person y, and x lives in a city z, then y also lives in that city. Such rules are usually not true in all instances, and typically come with a confidence score. Modern systems [30, 34, 40] can find such rules automatically on KBs of millions of entities. These rules can then be used to predict missing facts: If we know that some person lives in some city, but we do not know the place of residence of their spouse, we can use the rule to predict that, with high likelihood, the spouse lives in the same city.

The other methods to predict missing facts are *embedding-based methods*. These methods are a gift of the renaissance of neural networks in the 2010's. They project the entities and facts of a KB into a vector space. In its simplest variant, an entity x is mapped to its embedding, the vector $\vec{x}$. A relationship r, likewise, is mapped to a vector $\vec{r}$. These embeddings have the following property: If $\vec{r}$ is the vector for the *livesIn* relationship, then we can walk from the embedding $\vec{x}$ of a person x to the embedding $\vec{z}$ of their place of residence z by computing $\vec{z} = \vec{x} + \vec{r}$. This gives us another way of guessing the place of residence for some person y: We just find the city whose embedding is closest to $\vec{y} + \vec{r}$.

Each of these methods has its advantages and disadvantages: While rules are easy to understand for humans (and embeddings are less intuitively accessible), embeddings can take into account signals from all facts in which an entity occurs (and not just the ones mentioned in the rule, which are typically few). Therefore, recent years have seen fruitful endeavors to combine neural methods with symbolic methods. Both rule mining techniques and embedding techniques have been surveyed in recent articles [62, 9, 46, 73], among which is our own previous tutorial article [54]. Hence, in this tutorial, we survey approaches that combine both techniques.

The article is structured as follows: Section 2 introduces knowledge bases, rule mining techniques, and embedding techniques, following largely [54]. Section 3 discusses embeddings in more detail. Section 4 discusses embedding techniques that use rule mining techniques. Section 5, vice versa, discusses rule mining techniques that use embedding techniques. We conclude in Section 6.

2 Preliminaries

2.1 Knowledge Bases

Knowledge Bases. To define a knowledge base [54], we need a set $\mathcal{I}$ of *entities*. An entity is anything that can be an object of thought [67]. General-purpose KBs are typically concerned with entities such as places (e.g., *Paris*, or *India*), people (such as politicians, scientists, or actors), organizations (such as companies or associations), or artworks (such as movies, books, etc.). But knowledge bases can also be concerned with biomedical entities, geological formations, scientific articles, or any other type of entities.

In what follows, we assume a set $\mathcal{R}$ of binary *relation names* (also called *relations*, *relationships*, or *predicates*). For example, the relation *locatedIn* holds between a city and a country; the relation *actedIn* holds between an actor and a movie; and the relation *president-Of* holds between a person and a country. Finally, we need a set $\mathcal{L}$ of *literals*. These are strings or numbers. A *fact* (or an *assertion*, *triple*, or *statement*) is then of the form $\langle s, r, o\rangle$ with a *subject* $s \in \mathcal{I}$, a *relation* $r \in \mathcal{R}$ and an *object* $o \in \mathcal{I} \cup \mathcal{L}$ [30][1]. An example of a fact is $\langle$*Paris*, *locatedIn*, *France*$\rangle$. The *inverse* of a relation r is a relation r^-, so that $\langle x, r, y\rangle$ holds if and only if $\langle y, r^-, x\rangle$ holds. For example, the inverse of *hasNationality* is *hasCitizen*. A *knowledge base* $\mathcal{K}$ over the sets $\mathcal{I}, \mathcal{R}, \mathcal{L}$ is then a set of facts over these sets. Whenever $\mathcal{K}$ is clear from the context, we write $\langle s, r, o\rangle$ to mean $\langle s, r, o\rangle \in \mathcal{K}$.

Taxonomies. Knowledge bases typically also define *classes*. Intuitively, a class can be understood as a set of entities, its *instances*. For example, the class of capital cities contains the city of Paris, the city of Beijing, etc. Many formalisms use unary predicates to express class membership, stating, e.g., *city(Paris)*. If every instance of some class y is also an instance of some class y', then y is called a *subclass* of y'. For example, the class *capitalCity* is a subclass of the class *city*, which is itself a subclass of *geographicLocation*. This gives us a hierarchy of classes – the *taxonomy*. Figure 2 shows an example of a taxonomy of classes.

Many KBs express the taxonomy by binary relations. To say that an entity x belongs to a class y, the KB adds the triple $\langle x$, *type*, $y\rangle$. To say that a class y is a subclass of a class y', we add $\langle y$, *subclassOf*, *y'*$\rangle$. However, a taxonomy has an inherent semantics that is different from other facts that hold between entities, and therefore, one is usually ill-advised to treat the link $\langle$*Paris*, *type*, *city*$\rangle$ in the same way as $\langle$*Paris*, *locatedIn*, *France*$\rangle$.

Axioms. KBs typically come with a set of logical constraints. For example, we can impose that if x is an instance of a class y, and if y is a subclass of the class y', then x must also be an instance of y':

$$\langle x, \textit{type}, y\rangle \wedge \langle y, \textit{subclassOf}, y'\rangle \Rightarrow \langle x, \textit{type}, y'\rangle$$

Typical axioms are the following:
- **Domain and Range Constraints** say that the subject (resp. object) of a relation must belong to a certain class, as in "People are born in places (and not, say, in organizations)".
- **Cardinality Constraints** say that the number of objects per subject for a certain relation is restricted, as in "People can have at most one birth place".
- **Symmetry, transitivity, and inverse constraints** say that a relation is symmetric, transitive, or the inverse of another relationship.
- **Disjointness constraints** say that two classes cannot have instances in common, e.g., places and people.

[1] For our purpose, in line with the other works [17, 18, 40], we do not consider blank nodes.

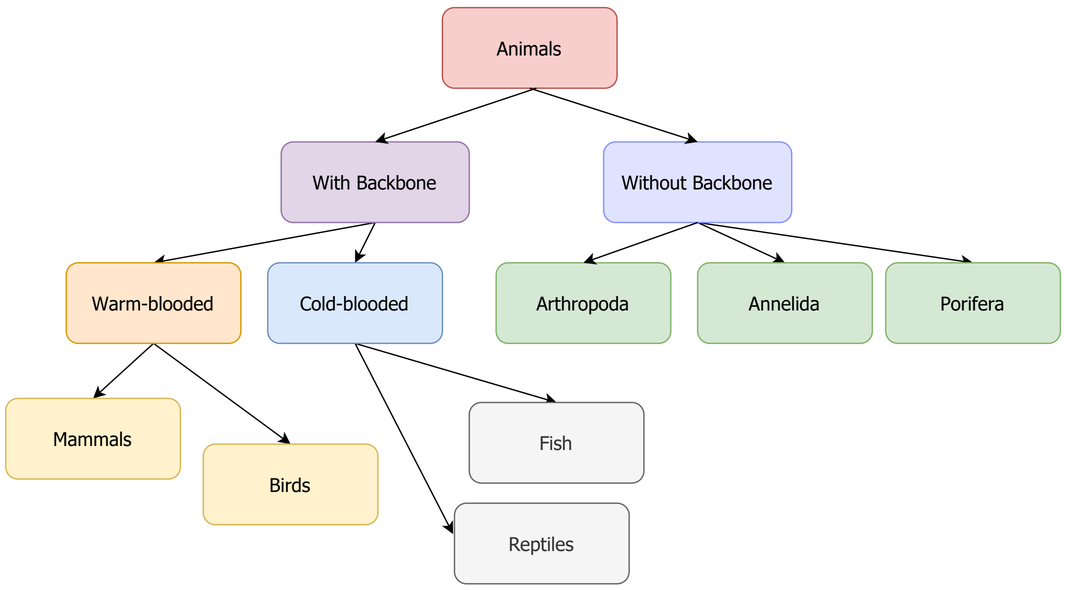

Figure 2 Taxonomy Example.

Such axioms exist in packages of different complexity: The Resource Description Framework Schema *RDFS* is a system of basic axioms that are concerned mainly with class membership. The axioms are so basic that they cannot result in contradictions. The Web Ontology Language *OWL* is a system of axioms that exists in several flavors – from the simple to the undecidable [54]. Such packages of axioms, together with the taxonomy, are sometimes called *ontology* or *schema*. Automated reasoners can be used to (1) predict facts that follow logically from these axioms and (2) determine whether a KB is inconsistent with respect to these axioms.

Fact Prediction. In what follows, we will assume an ideal knowledge base $\mathcal{K}^*$, which contains all facts of the real world (see [44] for a discussion of such a KB). One typically assumes that all facts in some given KB $\mathcal{K}$ are also true in the real world, i.e., $\mathcal{K} \subseteq \mathcal{K}^*$. However, the KBs are typically incomplete, i.e., there are facts in the real world that are not in the KB (i.e., $\mathcal{K} \subsetneq \mathcal{K}^*$). Predicting a fact f that is true in the real world, but not yet in the KB, is called the problem of *fact prediction*.

World Assumptions. Fact prediction is complicated by the fact that the KBs typically do not store negative information [43]. That is: while a KB may store that Elvis Presley has sung the song “All Shook Up”, it will not store the fact that he did not sing the song “The Winner Takes It All”. This raises the question what we should do if the KB does not contain certain statements (e.g., the KB does not contain the fact that Elvis sang “Always on my mind”, which is true in the real world). In a database, one would assume that any fact that does not appear in our data is not true in the real world – an assumption known as the *Closed World Assumption*. This assumption, however, is usually false for KBs, as KBs are highly incomplete and miss many facts from the real world. Hence, it is more appropriate to make the *Open World Assumption*, which says that if an assertion is not in the KB, it may or may not be true in the real world. Thus, in our example, if the KB does not contain the assertion that Elvis sang “Always on my mind”, we would not be entitled to conclude that this assertion would be false in the real world (which it is indeed not).

Negative assertions. A negative assertion is a statement that is known to be false. Such statements are essential as counter-examples in rule mining and fact prediction, so as to avoid an over-generalization. For example, Woody Allen married his step-daughter. If we find 10 other people who married their step-daughter, and no person who is *not* married to their step-daughter, we would conclude that people in general marry their step-daughters. The problem is now that KBs do not contain negative assertions. No KB tells us that Elvis Presley was *not* married to his step-daughter. And the Open World Assumption prevents us from assuming this negative assertion from the facts that are in the KB. This means that we have, in theory, no way to generate counter-examples for rule mining and fact prediction. Hence, we could mine the rule "Everybody is married to their step-daughter" without any obstruction.

Several remedies have been proposed. One is the *Partial Completeness Assumption*, or *Local Closed World Assumption* [17]. It says that if a KB contains the facts $\langle s, r, o_1 \rangle$, ..., $\langle s, r, o_n \rangle$, then any fact $\langle s, r, o' \rangle$ with $o' \notin \{o_1, ..., o_n\}$ must be false in the real world. The rationale is that if some contributor made the effort to add the objects $o_1, ..., o_n$, they would for sure also have added any remaining object o'. It can be shown that this assumption is generally true for relations that have few objects, such as *hasBirthDate* or *hasNationality* [18]. Indeed, in most KBs, the relations are designed in such a way that the average number of objects per subject is lower than the average number of subjects per object [18]. For example, a KB is more likely to contain the relation *hasNationality* (one person has few nationalities) rather than *hasCitizen* (one country has millions of citizens). A relation that has a higher average number of objects per subject than subjects per object can simply be replaced by its inverse [18]. With this, the PCA works generally well.

The method can be used as follows to generate a large number of negative examples: take any fact $\langle s, r, o \rangle$ from the KB, replace o by a randomly chosen object o' such that $\langle s, r, o' \rangle$ is not in the KB, and assume that $\langle s, r, o' \rangle$ is a negative assertion. The assertion $\langle s, r, o' \rangle$ is called a *corrupted* variant of $\langle s, r, o \rangle$. The method is also often applied in the same way to the subjects of the triples. This, however, creates a problem: Since relations generally have more subjects per object than vice versa, the PCA is much less plausible in this setting. For example, while it is, under the PCA, safe to assume that if some person Mary is American, she is not French, it is not safe to assume there are no more Americans than those in the KB. This is why the original PCA is applied only to the objects.

2.2 Rule Mining

Rules and Axioms. We have already seen that KBs can come with axioms, such as the symmetry of a relation. These axioms are usually defined manually, and they allow no exceptions. In what follows, we will be concerned with *rules*. These also express constraints on the data, but different from axioms, they are not imposed on the data, but automatically mined from the data. As such, they also allow for exceptions. For example, we can find that *marriedTo* is "usually" symmetric in the data of a given KB, meaning that for most couples, the *marriedTo* fact holds in both directions – although there are some couples for which the relation holds only in one direction, presumably because of missing data. This is why such rules are also called *soft rules* (as opposed to the "hard" axioms). Let us now make this idea more formal.

Atoms and Rules. An *atom* is an expression of the form $\langle \alpha, r, \beta \rangle$, where r is a relation and α, β are either entities or variables [30] (we write variables in lower case, and entities in upper case). For example, $\langle x, livesIn, Berlin \rangle$ is an atom with one variable, x. An atom is *instantiated* if at least one of its arguments is an entity. If both arguments are entities, the atom is *grounded* and tantamount to a fact. A *conjunction* of atoms $B_1, ..., B_n$ is of the form $B_1 \wedge ... \wedge B_n$. For example, we can build the conjunction $\langle x, livesIn, Paris \rangle \wedge \langle x, wasBornIn, Berlin \rangle$, which, intuitively, designates all people x who were born in Berlin and live in Paris. To make this intuition more formal, we need the notion of a substitution. A *substitution* σ is a partial mapping from variables to entities. Substitutions can be straightforwardly extended to atoms and conjunctions. For example, the substitution $\sigma = \{x \rightarrow Mary\}$ can be applied to our conjunction above, and it yields $\langle Mary, livesIn, Paris \rangle \wedge \langle Mary, wasBornIn, Berlin \rangle$.

A (Horn) *rule* is a formula of the form $B_1 \wedge ... \wedge B_n \Rightarrow H$, where the $B_1 \wedge ... \wedge B_n$ is a conjunction of *body atoms*, and H is the *head atom*. An example for a rule is

$$\langle x, married, y \rangle \wedge \langle x, livesIn, z \rangle \Rightarrow \langle y, livesIn, z \rangle$$

Let us call this rule R^* in what follows. Two atoms A, A' are *connected* if they have common variables. It is common [17, 18, 40] to impose that all atoms in a rule are transitively connected and that rules are closed. A rule is *closed* if every variable in the head appears in at least one atom in the body. A rule is *grounded* if all of its atoms are grounded.

Predictions. Given a rule $R = B_1 \wedge ... \wedge B_n \Rightarrow H$ and a substitution σ, we can apply σ to both the body and the head of R, and obtain an *instantiation* of R, which we denote by $\sigma(R)$. In our example, we could instantiate the above rule R^* by $\sigma = \{x \rightarrow Mary, y \rightarrow Bob, z \rightarrow Paris\}$, and obtain $\sigma(R^*)$ as

$$\langle Mary, married, Bob \rangle \wedge \langle Mary, livesIn, Paris \rangle \Rightarrow \langle Bob, livesIn, Paris \rangle$$

If $\sigma(B_i) \in \mathcal{K}\ \forall i \in \{1, ..., n\}$, we call $\sigma(H)$ a *prediction* of R from $\mathcal{K}$, and we write $\mathcal{K} \wedge R \models \sigma(H)$. Suppose, e.g., that we have a KB $\mathcal{K} = \{\langle Paris, locatedIn, France \rangle, \langle Mary, married, Bob \rangle, \langle Mary, livesIn, Paris \rangle\}$. Here, our example rule R^* can be instantiated as before by $\sigma = \{x \rightarrow Mary, y \rightarrow Bob, z \rightarrow Paris\}$. Then, all body atoms of the instantiated rule $\sigma(R^*)$ appear in $\mathcal{K}$. Hence, the rule predicts the head atom of $\sigma(R^*)$, which is $\langle Bob, livesIn, Paris \rangle$. Hence, we write $\mathcal{K} \wedge R^* \models \langle Bob, livesIn, Paris \rangle$.

Mining Rules. Inductive Logic Programming (ILP) is the task of finding rules automatically [54]. Typically, one provides a set of *positive examples* (i.e., facts that the rules shall predict), and a set of *negative examples* (facts that the rules must not predict). In the context of KBs, ILP faces several challenges: First, KBs usually do not provide negative examples. We have discussed a method to generate negative examples above, the Partial Completeness Assumption (Section 2.1). Another challenge is that a strict application of the definition of ILP to rule mining would find only rules that are true in all instantiations. However, in real-world KBs, there can be exceptions to rules, e.g., due to faulty or missing data. Hence, rule mining typically aims for rules that have a high *support* (the number of positive examples predicted by the rule), and a high *confidence* (the proportion of examples it predicts that are positive). In this way, the methods can find rules even if they do not apply in all instances, such as "If two people are married, then the children of one of them are also the children of the other".

AMIE [17] was one of the first rule mining systems for large KBs under the Open World Assumption. It starts with the most general rules (such as "everybody is married with each other"), and refines them until their confidence is high enough (e.g., "if two people are parents of the same children, they are most likely married"). This relies on the observation that the support of a rule decreases monotonically when a rule is made more specific. The RuDiK system [40] can mine logical rules like AMIE, but brings a number of improvements: First, RuDiK can also mine negative rules, such as "If two people are siblings, they are not married". Second, RuDiK can mine relations between literals, such as "Someone's birth date is always before someone's death date". Finally, RuDiK removes facts that have been covered by a rule, so that subsequent rules are forced to predict facts that have not already been predicted. This allows not just for some optimizations of the mining algorithm, but also to mine rules that predict more unknown facts correctly.

The AnyBURL system [34] is a bottom-up rule mining system: It starts with path rules that are specific to one instance, and generalizes them to achieve good support. A particular advantage of the system is that the user can trade running time for rule quality, i.e., get better rules by waiting longer.

The DRUM system [49] is a linear formulation of the rule mining problem using one-hot-encoding vectors for entities and adjacency matrices for relations. As it is linear, the problem is fully differentiable and can then be solved using gradient descent techniques. This solving appraoch proved to be very good for predictions involving previously unseen entities or relations.

Let us now turn to the second family of methods that can be used to predict missing facts: Knowledge Graph Embeddings.

2.3 Embeddings

Embeddings. An embedding for a group of objects (e.g. words, relations, or entities) is an injective function that maps each object to a real-valued vector, so that the intrinsic relations between the objects are maintained [54]. In the case of KBs, we are looking to embed entities and relations. In particular, given a KB, we would want the entities that are semantically similar in the KB to be mapped to vectors that are close to each other in the vector space.

The most basic embeddings [7] are designed so that, for a fact $\langle s, r, o\rangle$, we have $\vec{s}+\vec{r} \approx \vec{o}$, where $\vec{\cdot}$ is the embedding vector of the underlying entity or relation. For example, if we know $\langle$*Elvis*, *marriedTo*, *Priscilla*$\rangle$, then we would want the vector $\overrightarrow{Elvis} + \overrightarrow{marriedTo}$ to be close to the vector $\overrightarrow{Priscilla}$. An embedding with these properties has several advantages: First, the embedding allows us to feed entities and relations into machine learning methods that work on vectors (e.g., classification algorithms). The vectors are typically low in dimension (e.g., a few hundred), which makes them particularly suited for such applications. Second, the embedding provides a natural way of grouping together similar entities, so that given one entity, we can find its peers by scanning the vector space. In our example, we would expect Elvis to be close in the vector space to other singers. Finally, the embeddings allow for link prediction: If we do not know the spouse of Elvis, we can just compute the vector $\overrightarrow{Elvis} + \overrightarrow{marriedTo}$ and propose that the person that we find there is the spouse. If the embedding is well designed, that would actually work.

Terminology. In the literature about KB embeddings, the KB is often called a *knowledge graph* (KG) instead of a *knowledge base*. This is because embedding approaches typically project away literals and facts with literals. Consequently, fact prediction is known as *link prediction* in this scenario. Furthermore, the approaches typically do not deal with classes, taxonomies, or axioms. What remains is then a graph where the nodes are entities, and the edges are relations. In this scenario, facts are usually called *triples*, the subject is called the *head* of the triple, and the object is called the *tail*.

Link prediction with embeddings. Knowledge graph embeddings are created by trainable machine-learning models, typically neural networks. We will discuss these methods in detail in Section 3. All of these models take as input a fact $\langle h,\ r,\ t\rangle$, and output a score of its likelihood of being true: the higher the score, the more likely the model believes the fact to be. This score is typically denoted by $f(\langle h, r, t\rangle)$ or $f_{\vec{r}}(\vec{h}, \vec{t})$. To train such a model, we need a KB of true facts. We train the model to give a high score to these facts. To avoid over-generalization, we also have to train the model with counter-examples. These are typically generated by corrupting the facts from the KB (Section 2.1), i.e., by taking a fact $\langle h, r, t\rangle$ from the KB and replacing the tail by a random entity t'. The model is then trained to give the true triples from the input KB a higher score than the corrupted triples.

We can then use the models for link prediction as follows: We take a partially-filled triple for which we would like to know the head or tail entity, e.g., $\langle$*Elvis*, *marriedTo*, *?*$\rangle$. We try out all possible tail entities from the KB, and score the resulting triple using the scoring function. The predicted entity is intuitively the one with the highest resulting score. All entities can be sorted according to the scores of their triple. Each entity is then associated to its *prediction rank*, i.e., to the position that it has in the ranked list of predictions.

In the supervised setting, we often know the true answer (*Priscilla*), and we can compute its prediction rank $PR_{\langle Elvis, marriedTo, ?\rangle}(Priscilla)$. Several metrics are computed from the prediction ranks of head and tail entities. If $\mathcal{T}$ the set of known true facts, the metrics are the following:

- Mean Rank (MR): the average prediction ranks of the correct entities

$$MR = \frac{1}{2\,|\mathcal{T}|}\left(\sum_{(h,r,t)\in\mathcal{T}} PR_{\langle ?,r,t\rangle}(h) + PR_{\langle h,r,?\rangle}(t)\right)$$

- Mean Reciprocal Rank (MRR): the average of the inverse of the prediction ranks

$$MRR = \frac{1}{2\,|\mathcal{T}|}\left(\sum_{(h,r,t)\in\mathcal{T}} \frac{1}{PR_{\langle ?,r,t\rangle}(h)} + \frac{1}{PR_{\langle h,r,?\rangle}(t)}\right)$$

- Hit at k (Hit@k): proportion of the tests in which the prediction rank is better than k (typical values for k are 1, 3 and 10)

$$Hit@k = \frac{1}{2\,|\mathcal{T}|}\left(\sum_{(h,r,t)\in\mathcal{T}} \mathbb{1}\{PR_{\langle ?,r,t\rangle}(h) \leq k\} + \mathbb{1}\{PR_{\langle h,r,?\rangle}(t) \leq k\}\right)$$

Both MRR and Hit@k have values between 0 and 1, higher values indicate better results. In some cases, multiple entities can be correct answers (e.g. for 1-N relations) and the model should not be penalized for predicting another true answer that is simply more likely than the one at hand. Those metrics are usually computed in a *filtered* setting in which prediction ranks are computed by removing the other true entities ranked better than the one at hand.

3 Embedding Models

In the last decade, numerous methods for computing knowledge graph embeddings have been proposed. The methods differ from one another in terms of how they relate the entities and relations of the KG in the latent space. The existing models can be categorized as geometric, tensor-based or convolutional. In this section, we introduce and discuss a few popular models from each category.

In the following, let us consider a KG with n entities $\mathcal{E} = \{e_1, \dots e_n\}$ and m relations $\mathcal{R} = \{r_1 \dots r_m\}$ that is to be embedded in a d-dimensional vector space. $\mathbb{R}$ (resp. $\mathbb{C}$) is the field of real (resp. complex) numbers.

3.1 Geometric models

Geometric models interpret relations as geometric operations in the vector space. The earliest of these models is TransE, which we now describe in detail.

TransE [7] is a *translation-based model*, i.e., it uses a geometric distance to measure the similarity of the entities. Given a fact $\langle h, r, t\rangle$, its goal is to find vectors $\vec{h}, \vec{r}, \vec{t}$, so that $\vec{h} + \vec{r} \approx \vec{t}$.

One way to do that is to design a neural network [54]. We first create a *vocabulary*, i.e., an ordered list of all entities in the KG. Then we create, for each entity, its *one-hot encoding*. This is simply a vector that has as many dimensions as there are entities. Every element of the vector is set to zero, and only the i^{th} element is set to one, where i is the position of the entity in the vocabulary. The same is done for the relations. Then we design a network as follows: The input is the one-hot encoding of the head, the one-hot encoding of the relation, and the the one-hot encoding of the tail of a given fact from the KB. That is, if n is the number of entities, and m is the number of relations, the network has $m + 2 \times n$ input neurons. The first hidden layer of the network then maps each of these vectors to a d-dimensional real vector in $\mathbb{R}^d$. An entity e is mapped to a vector $\vec{e}$, and a relation r is mapped to $\vec{r}$. The further layers then reduce these vectors to a single output that scores the input assertion. More precisely, the network computes, for an input fact $\langle h, r, t\rangle$ from the KB, the function $f_{\vec{r}}(\vec{h}, \vec{t}) = -||\vec{h} + \vec{r} - \vec{t}||$ (where $||\cdot||$ is either the 1-norm or the 2-norm). The network is then trained with facts from the KB to maximize this score for these facts. It is trained with negative assertions to minimize this score. This leads to embeddings that verify the simple arithmetic equation $\vec{h} + \vec{r} \approx \vec{t}$ [54]. The important thing here is that the later hidden layers take their decision based solely on the output of the first hidden layer. The vectors computed by the first layer thus contain all the necessary information to assess the truth value of an assertion – and this is what we want from a good embedding. Thus, we will use the vectors that the first layer outputs as the embeddings of the input entities.

One limitation of TransE is the inability to model symmetric relationships [65]: if r is symmetric (i.e. $\langle h, r, t\rangle$ true implies $\langle t, r, h\rangle$ to be true as well), then r tends to have an embedding vector close to $\vec{0}$ because minimizing both $||\vec{h} + \vec{r} - \vec{t}||_2$ and $||\vec{t} + \vec{r} - \vec{h}||_2$ simultaneously happens if and only if $\vec{r} = \vec{0}$. Another problem appears with one-to-many relations. Consider for example the facts $\langle$*ElonMusk*, *founderOf*, *SpaceX*$\rangle$ and $\langle$*ElonMusk*, *founderOf*, *Tesla*$\rangle$. TransE would give very similar embeddings to both *SpaceX* and *Tesla*, and thus fail to differentiate between the two companies. TransE also has problems modeling many-to-one, reflexive, and transitive relations, and to capture multiple semantics of a relation.

TransH [65] tries to alleviate some limitations of TransE by allowing an entity to have different representations in the embedding space depending on the relation it is involved with. Each relation r is represented not only by a vector $\vec{r}$, but also by an hyperplane (i.e. a sub-space of one dimension less than the embedding space). Algebraically an hyperplane can be defined by a single vector, namely the vector that is orthogonal to it. Thus, each relation r is associated with a set of two vectors: $\vec{r}$ for the relation itself, and $\vec{h_r}$ for its hyperplane.

To compute the score of a triple $\langle h, r, t\rangle$, the embeddings $\vec{h}$, $\vec{t}$ of the entities are first projected onto the hyperplane defined by $\vec{h_r}$, and they are then connected by the translation vector $\vec{r}$ of the relation. Given a relation r, let p_r be the linear orthogonal projection on the hyperplane defined by $\vec{h_r}$. Then the loss function of TransH can be written as $f(\langle h, r, t\rangle) = f_{\vec{r}}(\vec{h}, \vec{t}) = -||p_r(\vec{h}) + \vec{r} - p_r(\vec{t})||_2^2$.

This is designed to solve the limitations of TransE: a reflexive relation r can have a translation vector $\vec{r}$ close to $\vec{0}$, since all information is contained in $\vec{h_r}$. For relations with several objects, likewise, the objects can be embedded in the same place in the hyperplane only for that specific relation.

TransR [31] extends the idea of sub-space projection of TransH by proposing that the projection step is now done on any sub-space of a given dimension. Let d be the dimension of the embedding space and d' the dimension of the relation-specific sub-spaces. Algebraically a linear projection from a vector space of dimension d into one of its sub-spaces of dimension d' is simply represented by a matrix of dimension $d \times d'$. Each relation r is then represented by a vector $\vec{r}$ and a projection matrix M_r. Thus, TransR is simply an evolution of TransH that increases the expressiveness of the model by increasing the number of parameters. Intuitively, this should allow the model to *learn* a greater amount of useful information from the known facts it is trained on. CTransR [31] is an extension of TransR, which operates by clustering diverse head-tail entity pairs into groups and learning distinct relation vectors for each group.

TransD [25] in turn proposes to keep the idea of projecting on any possible sub-space but reduces the number of parameters compared to TransD in order to limit the risk of overfitting. This is done by allowing only the sub-space projections that are defined by a *low-rank* matrix: that is a matrix that can be decomposed as a product of vectors.

Several other improvements have also been proposed in the direction of translation embedding methods, including TransG [68], TransF [16], and KG2E [21]. Other geometric models perform rotation-like transformations in the vector space instead of pure translations. Its most prominent examples are RotatE [55] and HAKE [72].

RotatE [55] aims to be particularly suited for relations that are symmetric, anti-symmetric, inverses of each other, and compositions of each other, which are typical for KGs. For instance, the relation *marriedTo* is a symmetric relation: $\langle x$, *marriedTo*, $y\rangle$ implies $\langle y$, *marriedTo*, $x\rangle$. Further, many relations such as familial relations are compositional. For example, $\langle x$, *hasParent*, $y\rangle$ and $\langle y$, *hasParent*, $z\rangle$ imply $\langle x$, *hasGrandParent*, $z\rangle$. RotatE captures these relation patterns by defining each relation as a rotation from the head entity to the tail entity in the vector space. Specifically entities and relations are now embedded in $\mathbb{C}^d$ and for any relation r, the modulus of each component $\vec{r_i}$ is 1. For a triple $\langle x, r, y\rangle$, the model then tries to achieve $\vec{y} \approx \vec{x} \circ \vec{r}$, where $\circ$ is the element-wise product. Intuitively, a relation r applies a coordinate-wise rotation on the head entity so as to come close to the tail entity. The score function is then $||\vec{x} \circ \vec{r} - \vec{y}||$. A relation is symmetric if and only if its embedding belongs to $\{-1, +1\}^d$ (i.e. coordinate-wise rotations of 0 or π radians), r_1 and r_2 are are symmetric if and only if their embeddings are complex conjugates, and a relation r_3 is the composition of two relations r_1 and r_2 if and only if $\vec{r_3} = \vec{r_1} \circ \vec{r_2}$ (i.e., the coordinate-wise rotations of r_3 are the successive rotations of r_1 and r_2).

HAKE [72] extends the RotatE embeddings by taking into account and preserving the semantic hierarchies of the entities in the KGs. For example, the entity *Paris* is part of *France*, which is a part of the EU. Such hierarchies between entities are quite common in most KGs such as Yago and Freebase. To model these relations between entities, HAKE represents an entity e (and a relation r) in the vector space in two parts: as $\vec{e}_m$ and $\vec{r}_m$

in the modulus part and as $\vec{e}_p$ and $\vec{r}_p$ in the phase part. The modulus part is aimed at differentiating entities at different hierarchies from each other, such as *Paris* from *France*, while the phase part distinguishes the different entities at the same hierarchy level, e.g. *Paris* and *Lyon*. In this manner, HAKE is able to represent the semantic hierarchies associated with KGs, and outperform other techniques by learning better embeddings.

3.2 Semantic Matching models

Another common category of embedding methods compares the vector of the subject and the vector of the object directly in order to assess how likely the fact is to be true.

RESCAL [38] is the simplest model in this category. Entities are represented as vectors and relations become bilinear functions (simply represented as square matrices). A triple $\langle h, r, t\rangle$ is then scored by the application of the relation-specific bilinear function to the entity embeddings: $f(\langle h, r, t\rangle) = \vec{h}^t \cdot M_r \cdot \vec{t}$, where $\vec{h}$ (resp. $\vec{t}$) is the embedding of h (resp. t) and $M_r \in \mathbb{R}^{d\times d}$ is the representing matrix of r. Intuitively, this bi-linear scoring function can be interpreted as some sort of *scalar product* between the entities in some relation-specific distortion of the embedding space. This is simply an intuition as no sufficient constraints are imposed on the relation matrices to make them scalar products. Precisely, they are not forced to be symmetric nor positive definite.

DistMult [69] is a variation of the RESCAL models where the relation matrices are all forced to be diagonal. This simplifies the computations, and reduces the parameter space. As a drawback, DistMult gives the same score for the triples $\langle h, r, t\rangle$ and $\langle t, r, h\rangle$. Thus, it is unable to model asymmetric relations such as *sonOf*, *actedIn* etc. Despite these limitations, DistMult has been recently shown to perform as well as many recently proposed models, presumably due to its simplicity and scalability [48].

ComplEx [58] improves upon the DistMult model by using the same diagonal constraint, but with complex-valued embedding vectors. Entities and relations are then simply represented as vectors in $\mathbb{C}^d$ and the Hermitian product is used instead of the bi-linear product in the scoring function. This allows the approach to take into account asymmetric relations in the KGs, as in the triple $\langle$*Paris*, *capitalOf*, *France*$\rangle$ (where France is not the capital of Paris). The scoring function is defined as $f(\langle h, r, t\rangle) = Re(\vec{h} \cdot M_R \cdot \overline{\vec{t}})$ where $Re(c)$ is the real part of $c \in \mathbb{C}$ and M_r is the diagonal matrix with $\vec{r}$ on its diagonal. The fact that the Hermitian product is not commutative solves the problem of representing asymmetric relations and switching to complex vector space doubles the number of parameters thus increasing the expressiveness of the model.

SimplE [27] proposes to extend one of the most generic multiplicative methods: Canonical Polyadic (CP) decomposition [22]. This method is used for decomposing tensors into a sum of products. It can be applied to KG embeddings because a KG with n entities and m relations is simply represented as a 3-dimensional adjacency tensor $\mathcal{T} \in \{0,1\}^{n\times n\times m}$: $\mathcal{T}[i,j,k] = 1$ if $\langle e_i, r_k, e_j\rangle$ is true and 0 else. As explained in [27], CP decomposition represents entities e with two vectors $(\vec{h_e}, \vec{t_e}) \in (\mathbb{R}^d)^2$ and relations r with a vector $\vec{r} \in \mathbb{R}^d$ where d is the dimension of the embedding. These vectors are learned in order to be able to reconstruct the tensor $\mathcal{T}$ by estimating $\hat{\mathcal{T}}[i,j,k] = \langle \vec{h_{e_i}}, \vec{t_{e_j}}, \vec{r_k}\rangle = \sum_{\ell=1..d} \vec{h_{e_i}}[l] \times \vec{t_{e_j}}[l] \times \vec{r_k}[l]$. This estimation is used in the case of KG embeddings as a scoring function of triples. SimplE just proposes to represent relations r with two vectors $\vec{r}$ and $\vec{r^{-1}}$, the scoring function being now $f(e_i, r, e_j) = \frac{1}{2}(\langle \vec{h_{e_i}}, \vec{t_{e_j}}, \vec{r}\rangle + \langle \vec{h_{e_j}}, \vec{t_{e_i}}, \vec{r^{-1}}\rangle)$. The authors show that their model is fully expressive, meaning that if given enough embedding dimensions it can exactly represent any KG. It is then argued that simple logical constraints can be implemented in the model by applying constraints on the relation embeddings. We will later see one such application in Section 4.3.

3.3 Deep Models

Deeper neural architectures have also been introduced for KB embeddings, with the hope that hidden layers can capture more complex interaction patterns between entities and relations (and then estimate more complex scoring functions). In such models, the first part of the network (which, in shallow networks, just maps facts to their embeddings or their projections) now adds additional layers (possibly numerous) that receive as inputs the embeddings, and produce as outputs some extracted features. The second part of the network now computes the scoring function from the features extracted by the first part of the network, and not directly from the embedding (or its projection) as in shallow models. The scoring function also becomes a parameter of the model (to be trained) and is not defined a priori anymore. This often entails that we lose the interpretability of the scoring function [54]. There are many deep neural network based models that have been proposed over the years, early examples of such models are SME, NTN and MLP [6, 52, 13].

NTN [52] was introduced by Socher et al. as a generalization of the RESCAL model. It employs a combination of linear transformations and nonlinear activation functions to obtain head and tail embeddings. As such, while this is a more expressive model, it is also quite complex with a large number of parameters that are harder to train. Better and lightweight architectures have been since proposed, such as MLP, where the paramaters are shared among all the relations.

ConvE and ConvKB [11, 10] are popular examples of models that are based on convolutional neural networks (CNN). These can learn complex nonlinear features of the entities and relations with fewer parameters by using 2D convolutions over embeddings. ConvE has been shown to be particularly effective for complex graph with nodes having a high number of incoming edges. The model introduced the 1-N scoring scheme where for a given triple $\langle h, r, t\rangle$ where t is to be predicted, the matching is performed with all the tail entities at the same time, leading to speedier training. ConvE has proven to be a competitive embedding model and a popular baseline for more recent deep learning approaches.

Graph Convolutional Networks (GCNs) have recently gained popularity for performing link prediction in knowledge graphs in tandem with standard embedding techniques. GCNs are a form of message passing multi-layer neural networks, first introduced by [28] for semi-supervised node classification on graph structured datasets. One layer of GCN encodes information about the immediate neighbours of a node in feature vector, and k layers stacked on top of each other can encode the information of the neighbourhood k hops away. GCNs can overcome the limitations of knowledge graph embedding models in terms of neglecting the attributes of the entities and ignoring the graph structure by encoding the entities based on their neighbours in the graph. Several extensions of GCNs have been suggested for multi-relational knowledge graphs.

Relational GCNs (R-GCNs) [50] are GCNs for graphs with a large number of relations, which makes them particularly suitable for knowledge graphs. Link prediction is essentially an auto-encoder framework: An encoder creates the feature representations for the entities from its neighbours (these features are generated from relation-specific transformations that are dependent on the type and direction of the relations). A decoder (in this case, DistMult factorization) is a scoring function to predict the labelled edges. R-GCNs show improvements compared to DistMult, HolE and ComplEx for the link prediction task. While R-GCN extended the GCN models on knowledge graphs by including the different types of relations during the generation of entity representations, they do not represent relations themselves.

VR-GCN [70] is an extension of the R-GCN model that generates both entity and relation embeddings explicitly. It ensures relation representation and different entity roles (head or tail in different triples), and it conforms to the translation representation from translational embeddings where $\vec{h} + \vec{r} \approx \vec{t}$. While the primary goal of this technique is to enable graph alignment, the performance of VR-GCN is also discussed in terms of the link prediction task, with VR-GCN acting as the encoder and DistMult as the decoder for scoring the triples.

SACN [51] leverages a variant of the existing knowledge graph embeddings ConvE and TransE as decoder along with a variant of GCN (weighted GCN) as encoder. The weighted GCN encoder learns representations for the entities in the graph by utilizing the graph structure, node attributes and the associated relations while weighing different relations differently and learning these weights during the training. The entity representations are given to the decoder, which is a combination of ConvE and TransE (based on ConvE but having the translational property of TransE model) that performs better than the ConvE model. The encoder and decoder are trained jointly to learn the entity representations and score triples to verify and improve the representations.

CompGCN [59] generalizes previous GCN methods by jointly learning the representation of the nodes and the relations in the multi-relational KGs while leveraging composition functions from embedding approaches. CompGCN is able to scale well with the increasing number of relations and outperforms several previous models including TransE, DistMult, ComplEx, R-GCN and SACN.

3.4 Evaluation of Embedding Methods

3.4.1 Evaluation Protocol

Evaluation. Rule methods are typically evaluated under the *open world assumption*, i.e., any fact that is predicted is manually evaluated to see whether it holds in the real world or not. Thus, even a fact that does not appear in the KB can be counted as correct. This evaluation is obviously very labor-intensive, but it targets what rule mining is interested in: the prediction of yet-unknown facts. KB embedding models, in contrast, are typically evaluated under the closed world assumption: Given a KB, one removes a certain portion of it to obtain a training KB. One then trains the embeddings on this reduced KB, and uses the embeddings to predict facts. If these facts appear in the original KB, they count as correct, otherwise they count as incorrect.

Datasets. Three very common KBs for evaluating embedding approaches are FB15k [7], WN18 [6] and Yago3-10 [11]. FB15k and WN18 were both proposed by Bordes et al. respectively in 2013 and 2014. FB15k is an extraction from Freebase where entities were selected based on the number of citations in the original KB. WN18 is a subset of Wordnet in which entities are *synsets*, that is semantic senses of words (selected on their popularity in the KB) and predicates are lexical relations between those senses. Yago3-10 was proposed by Dettmers et al. in 2018 as a subset of Yago3 [33] in which most of the facts describe people (e.g., by citizenship, gender, and profession). The three KBs have been initially randomly split into training, validation and test subsets and those splits always stay the same. Table 1 shows some statistics about these datasets.

Table 1 Details on the various KBs used for embedding evaluation.

Dataset	Number of entities	Number of relations	Number of training facts	Number of evaluation facts	Number of test facts
FB15k	14,951	1,345	483,142	50,000	59,071
FB15k-237	14,541	237	272,115	17,535	20,466
WN18	40,943	18	141,442	5,000	5,000
WN18RR	40,943	11	86,835	3,034	3,134
Yago3-10	123,182	37	1,079,040	5,000	5,000

3.4.2 Shortcomings of Benchmarks

While embedding models have gained popularity for the link prediction task and obtained state-of-the-art results, several studies have recently taken a critical look at the performance and evaluation aspects of these models. The benchmark datasets on which the embedding models are trained have also been scrutinized.

Toutanova et al. [57] were the first to find data leakage issues in the FB15k dataset. More precisely, the authors noted that, for certain relations r, the inverse relation r^- was also present in the data. This makes the prediction of a fact $\langle x, r, y\rangle$ trivial if the fact $\langle y, r^-, x\rangle$ is already there. As a remedy, the authors constructed the dataset FB15k-237 by removing the inverse triples and keeping only one relation out of the reverse relations. Dettmers et al. [11] similarly found issues with the WN18 dataset and created the WN18RR dataset. Table 1 shows the statistics about these datasets. With the introduction of these new datasets and their adoption for the evaluation of newer embedding models, it could be ensured that the models are not just learning trivial entailment, but learning to correctly predict non-trivial facts that require actual inference. However, most papers still showed the results for the evaluation of new models on both the old and new version of the datasets.

Akrami et al. [2] conducted a further detailed study questioning the performance of embedding models in the presence of data leakage and data redundancy. The study found a sizeable percentage of inverse, duplicate, and Cartesian product relations in the popular datasets FB15k, WNRR and Yago3-10. Duplicate relations are relations with different names that share the same facts (e.g., *hasCitizenship* and *hasNationality*). Cartesian product relations are relations that hold between all instances of a class (e.g., *sameSpeciesAs*). Such relations can be predicted trivially. Hence, the authors argued, the performance of these models would be significantly worse for link prediction on actual unseen data in realistic settings. Their experiments analysed various popular embeddings models including TransE, TransH, TransR, TransD, DistMult, ComplEx, ConvE, Tucker, and RotatE and showed substantial drops in performance with different datasets after removing the unrealistic triples, so much so that simple rule based techniques could achieve better accuracy than complex embedding techniques. The authors therefore strongly advocated the need to re-evaluate existing embedding approaches to find an effective solution for the link prediction task.

Rossi et al. [47] take a critical look at the properties of the entities in the benchmark datasets that are used to evaluate link prediction performance of embedding models. They focused on the Freebase and Wordnet datasets and performed a detailed experimental analysis of the features of these datasets and their limiting effect on the performance of embedding models. For instance, the authors showed that embedding models perform artificially better for the most frequent entities in the dataset. In FB15k, the entity *United States* appears in a lot of triples, and therefore, the TransE and DistMult models show better scores while predicting this entity as the missing entity. If the most frequent entities were removed from these datasets, the model performance (counter-intuitively) improved, indicating the over-fitting of the models on the most representative entities. Therefore, the authors advocated that better benchmarking practices and metrics are needed to determine the capability and fairness of the models.

Pujara et al. [42] performed an interesting study on the effect of sparsity and unreliable data on the performance of embeddings. Existing curated KGs like Wordnet and Freebase were modified in different experiments to introduce sparsity (in terms of relations or entities) and unreliable or corrupted triples, so that they resemble real-world KGs derived from text (such as NELL [8]). The authors found that performance is closely linked with sparsity, i.e. embeddings work well for relations and entities that have a dense representation and sparsity adversely affects their performance. Experiments showed that unreliable triples also degraded the performance. However, the authors made an interesting conclusion, namely that corrupted triples still improved embeddings marginally, therefore it is better to have a large noisy KG rather than a small set of very high quality triples.

These studies helped in bringing into focus the flaws that are inherent in all the popular benchmark datasets due to which global metrics for the evaluation of embedding models are proved to be insufficient and misleading. Thus, there is a need for careful and fine-grained evaluation of the performance of embedding models for their application in realistic use cases.

3.4.3 Shortcomings of the protocol

Several works have studied the shortcomings of the evaluation protocol for KB embeddings.

Pezeshkpour et al. [41] focused on the evaluation metrics and pointed out the need and importance of calibration of the embedding models before they can be deployed in real-world scenarios. For example, if the model says with 0.5 confidence that a triple is true, then the actual probability of the triples with this confidence should also be 0.5. In particular, they found that the model calibration as well as the ranking metrics were highly susceptible to the choice of negative sampling during training, with random replacement of subject or object entity *(Random-N)* leading to worst results. In order to improve the evaluation techniques, the authors proposed the *CarefulN* method to select negative samples. Here, the highest scoring negative sample having an entity type which is different from the target entity type is selected as a negative sample. E.g. given a triple $\langle$*Barack Obama*, *presidentOf*, *USA*$\rangle$, if *USA* is the target entity to be predicted, and the ranked list of predicted entities is *(USA, Hawaii, United Nations, Michelle Obama, . . .)*, then we choose $\langle$*Barack Obama*, *presidentOf*, *Michelle Obama*$\rangle$ as the negative sample since the type for *Michelle Obama* is different from *USA*. This technique explicitly ensures that the negative sample being generated is a true negative. Following this technique, they derived a new benchmark dataset Yago3-TC for evaluating KG completion that consists of both true and false facts for facilitating the correct measurement of triple classification accuracy.

Sun et al. [56] looked into the very specific issue of the recent neural-networks based embedding models showing inconsistent performance gains across different datasets such as FB15k-237 and WNRR18. They investigated in detail the models ConvKB [10] and CapsE [61] and found an unusual score distribution to be the reason for this discrepancy. For instance, many negatively sampled triples were given the same score as the correct triple. To break ties for the triples with the same score, they proposed a RANDOM evaluation protocol, i.e. if multiple triples have the same score, one of them is chosen randomly. Experiments demonstrated that recent deep models such as ConvKB, and CapsE were indeed affected by different evaluation protocols (unlike other models like ConvE and RotatE) and this could be detected with the proposed RANDOM protocol.

Kadlec et al. [26] were among the first to question the performance gains achieved by the newly proposed model architectures. The authors were able to perform suitable fine tuning the hyper-parameters for DistMult, one of the first embedding models proposed, and outperform several new models. This raised concerns on the performance gains by the newer models, advocating for a closer inspection of the training practices and objectives.

Ruffinelli et al. [48] re-implemented several existing models and performed extensive analysis of the performance of these models to compare them on a common ground. Going beyond previous works such as [36] (which studied the loss functions) and [29] (which looked into the negative sampling strategies), this paper performed a comprehensive and empirical evaluation of the effect of different training strategies and parameters such as the regularizer, optimizer and loss functions on a number of new and old embedding models. Their analysis indicated that the training parameters play a major role in the embedding performance. With a systematic fine tuning of these parameters, even the older model architectures such as RESCAL can match or outperform the recently introduced improved models. This work makes a strong point of the need for re-assessing the individual benefits claimed by recent and newer embedding models in light of the older models. The authors caution that the performance gain reported by newer models could be mitigated by merely fine tuning of the training strategies and therefore, this warrants close inspection.

Jain et al. [23] raised questions regarding the very semantic representation learned by the embeddings models in the first place. They performed classification and clustering experiments on the embeddings in order to analyse their semantic capability. The authors challenge the common notion that entities having similar meaning (i.e. belonging to the same class or type) such as *politicians*, *actors* etc. would be represented by similar vectors. They constructed a dataset with entities belonging to different levels of the taxonomy for Yago3-10 and DBpedia datasets, such as from *person* to *artist* to *painter*. Detailed experiments demonstrated that both clustering and classification showed poor results for entities having fine-grained classes. This means that embeddings are unable to capture the semantics of entities beyond the top level classes (*person*, *organization*, *places* in Yago). These surprising results indicated that though embeddings might show good performance on the link prediction task, their utility for other semantic tasks such as reasoning etc. should be carefully examined.

Wang et al. [64] inspected the evaluation protocol of the embedding techniques for the KB completion task. They argue that the Entity Ranking (ER) protocol, where the missing head or tail entity is predicted for a triple, is more suitable for evaluating a question answering task, but not for the KB completion task. This is due to the fact that the context of the missing information in terms of a head or tail entity would not be available when attempting to find new missing triples of the form $\langle ?, r, ? \rangle$. With the ER protocol, the models may not be penalized for ranking certain incorrect triples higher since they are not encountered at all. The paper instead proposes a Pairwise Ranking (PR) protocol where all possible entity pairs are considered and ranked with respect to a particular relation. Extensive experiments show that popular embedding models provide worse performance with PR protocol than with the ER protocol, even on seemingly easy datasets.

These studies have emphasized the need for better evaluation protocols and a critical look at the training strategies of the embedding models for the task of KB completion in realistic settings.

4 Embedding Methods with Logical Components

4.1 Rationale

Rule mining methods and embedding methods are complementary for the purpose of link prediction:

- Rule mining produces patterns that can be understood by humans. Thus, their predictions can be explained and justified.
- Rule mining can, in principle, work together with the schema of the KB, axioms, and other types of logical constraints.
- Rule mining methods can deal with literals and numerical values, while embedding methods typically project these away. Rule mining can find, e.g., that the death date is always later than the birth date.
- Rule mining is typically evaluated under the open world assumption: it is explicitly designed to predict facts that are not yet in the KB. Embedding methods, in turn, are typically tuned to predict what is already there.
- On the flip-side, rule mining methods typically predict based on a single rule; it is harder to predict facts with several rules that reinforce or contradict each other.
- Rule mining methods do not have a holistic view on an entity, with all its relations; they are restricted to the relations that appear in the rules.
- Rule mining methods often generate rules with a low confidence, i.e., with a high rate of false predictions.

For these reasons, several works have taken to combine symbolic and embedding methods. Here, the symbolic component does not necessarily come from Rule Mining, but can also come from the logical axioms of the ontology of the input KB. The existing approaches fall into three classes:

- **Adding simple axioms.** Some approaches constrain the embeddings by simple axioms, which concern inverse, symmetric, or equivalent relations. We discuss them in Sections 4.2 [12], 4.3 [15], and 4.4 [35].
- **Complex constraints.** Other approaches support more general and complex logical constraints on the embeddings. We discuss them in Section 4.5 [23], 4.6 [14], and 4.7 [63].
- **Joint learning.** Finally, a number of approaches jointly embeddings and confidences for rules. We discuss them in Section 4.8 [19] and 4.9 [20, 71].

4.2 Improving Knowledge Graph Embeddings Using Simple Constraints

A first simple combination of logical rules and embeddings is presented by Ding et al. [12]. The authors focus on relation entailments, i.e., rules of the form $\langle x, r, y \rangle \Rightarrow \langle x, r', y \rangle$ that can also be denoted as $r \Rightarrow r'$. For example, if two people are married, then they also know each other: $marriedTo \Rightarrow knows$. Such entailments can either be axioms from the ontology, or they can be soft rules, i.e., rules with a confidence score that do not hold in all instantiations. For example, a soft rule can be: If a person is born in a country, then the person probably has the citizenship of that country. This is very often the case though not always. Such soft rules can be mined by a rule mining system, and from now on, a set of such entailments is assumed to be available.

If it is known that a relation r entails a relation r' and that $\langle x, r, y\rangle$ holds for some entities x, y, then $\langle x, r', y\rangle$ holds. Thus, the score that an embedding model gives to the fact $\langle x, r, y\rangle$ should not be larger than the score it gives to $\langle x, r', y\rangle$:

$$f(\langle x, r, y\rangle) \leq f(\langle x, r', y\rangle) \tag{1}$$

The authors enforce this condition on the ComplEx model (see Section 3) by imposing that for a given entity x, all the real parts of the components of the embedding vector $\vec{x} \in \mathbb{C}^d$ have to be non-negative, and all the imaginary parts have to be smaller than or equal to one. Given an entity x, d the embedding dimension, the constraints are formalized in Equation 2 where $Re(\cdot)$ (resp. $Im(\cdot)$) returns the real (resp. imaginary) part of a complex number and $\vec{x}_i$ is the i^{th} component of the vector $\vec{x}$.

$$\forall i \in \{1, \dots, d\} : Re(\vec{x}_i) \geq 0 \land Im(\vec{x}_i) \leq 1 \tag{2}$$

This constraint can be intuitively justified by seeing each component of $\vec{x}$ as a feature, whose value is zero if the feature does not apply to the entity x, and greater than 0 if the feature applies to x, but never below zero. The constraint on the imaginary component serves as a kind of normalization. With this non-negativity constraint, the desideratum of Equation 1 can be achieved by requiring:

$$\forall i \in \{1, \dots, d\} : Re(\vec{r}_i) \leq Re(\vec{r'}_i) \land Im(\vec{r}_i) = Im(\vec{r'}_i) \tag{3}$$

If the entailment does not hold strictly, but only with a certain confidence, the condition can be relaxed by introducing a real-valued confidence level λ and vector slack variables $\vec{\alpha}, \vec{\beta}$, which turn Equation 3 into

$$\forall i \in \{1, \dots, d\} : \lambda \times (Re(\vec{r}_i) - Re(\vec{r'}_i)) \leq \vec{\alpha}_i, \lambda \times (Im(\vec{r}_i) - Im(\vec{r'}_i))^2 \leq \vec{\beta}_i \tag{4}$$

The larger the confidence level λ, and the smaller the slack variables $\vec{\alpha}$ and $\vec{\beta}$, the more Equation 4 resembles the hard constraint of Equation 3.

When these constraints are imposed on the ComplEx model, then the model is forced to give a high score to facts that are logically entailed by other facts to which it gave a high score. The authors then show that this improves the performance of link prediction over the original model.

4.3 Improved Knowledge Graph Embedding Using Background Taxonomic Information

Fatemi et al [15] introduce another way to improve knowledge graph embeddings, which uses the taxonomy of the KB. For example, a knowledge base might contain the information that *Emmanuel Macron* is a *president*, but it does not contain information that he is a *mammal*, because it is implied by taxonomical knowledge. With this knowledge, if we know that mammals are warm-blooded, we can conclude that *Emmanuel Macron* is warm-blooded as well, without having explicit facts about this relation in the KG. Going one step further than relation entailment, this work leverages the subsumption property of the relations as well as the classes in KG. For example, the relation *presidentOf* is a sub-property of *managerOf*, which in turn is a sub-property of *employedBy*. Formally, if a relation r_1 is a sub-property of a relation r_2, then $\forall x, y : \langle x, r_1, y\rangle \Rightarrow \langle x, r_2, y\rangle$. To represent class subsumption, the authors model the entities as the characteristic functions of the class they belong to. This means that if entity e is in class C i.e. $\langle e, \mathit{type}, C\rangle$, then the characteristic function between e and C is true – written as $\langle e, C, true\rangle$. Hence, class subsumption can be expressed as a special case of relation subsumption. For instance, if *president* is subclass of *mammal* in the taxonomy, then $\langle EmmanuelMacron, president, true\rangle \Rightarrow \langle EmmanuelMacron, mammal, true\rangle$.

The proposed framework is a modification of the SimplE [27] embedding model (see Section 3.2), which makes use of these axioms. SimplE considers two embeddings for each relation: one embedding r^+ for relation itself and another r^- for its inverse relation. Similarly, there are two embeddings for each entity: one as a head entity e^+, and another as the tail entity e^-. These embeddings are concatenated to obtain the final embedding for a relation or entity. The proposed modification of this model is restricting the entity embeddings to be element-wise non-negative.

In order to enforce the axiom that a relation r is a sub-relation of a relation s ($\forall x, y : \langle x, r, y\rangle \Rightarrow \langle x, s, y\rangle$), the model adds an equality constraint as $\vec{r} = \vec{s} - \vec{\delta_r}$ where $\vec{\delta_r}$ is a non-negative vector, which expresses how r is different from s. This vector is learned during training. With this, the function μ (that maps embeddings to the probability of a triple) obeys the constraint $\mu\langle x, s, y\rangle \geq \mu\langle x, r, y\rangle$.

Thus, the resulting $SimpleE^+$ model is able to enforce subsumption properties for entities and relations and therefore, incorporate taxonomic knowledge in the embeddings to learn more interpretable representation for words [37]. The experimental evaluation shows that the proposed model is able to outperform traditional SimplE for the KG completion task and also has a faster convergence rate when taxonomic information is available.

4.4 Regularizing Knowledge Graph Embeddings via Equivalence and Inversion Axioms

We have so far seen approaches that concentrate on subproperty axioms. We shall now look into two other types of axioms [35]: Given two relations r_1 and r_2, an equivalence axiom $r_1 \equiv r_2$ means that r_1 and r_2 are semantically equivalent though distinct in the KB (e.g., *part of* and *component of*). An inverse axiom $r_2 \equiv \bar{r_2}$ means that r_1 is the inverse predicate of r_2 (e.g., *part of* and *has part*). The approach assumes that these axioms are defined in the ontology of the input KB.

Given the two sets of equivalence and inversion axioms, constraints are enforced in the training of the models. Let $r_1 \equiv r_2$ be an equivalence (resp. inversion) axiom. This means that relations r_1 and r_2 are equivalent (resp. inverse) and then the scoring function f of an embedding model should verify $f(\langle h, r, t\rangle) = f(\langle h, r_2, t\rangle)$ (resp. $f(\langle h, r, t\rangle) = f(\langle t, r_2, h\rangle)$) given any entities h and t.

In the case of equivalence, this is simply implemented by forcing the embeddings of r_1 and r_2 to be the identical. In the case of an inversion axiom, the constraint has to be specified for each model in the form of a model-dependent function Φ such that the constraint $\vec{r_2} = \Phi(\vec{r_1})$ results in $f(\langle h, r_1, t\rangle) = f(\langle t, r_2, h\rangle)$. For example, in the case of TransE [7], using $\Phi : \vec{r_2} \mapsto -\vec{r_1}$, one gets $f(\langle h, r_1, t\rangle) = ||\vec{h} + \vec{r_1} - \vec{t}|| = || - \vec{h} - \vec{r_1} + \vec{t}||$ (by homogeneity of the norm) and then $f(\langle h, r_1, t\rangle) = ||\vec{t} + \Phi(\vec{r_1}) - \vec{h}|| = f(\langle t, r_2, h\rangle)$. Note that Φ needs to be an involution, i.e., $\forall r, \Phi(\Phi(r)) = r$.

These constraints are called *hard constraints* because they entirely determine some embeddings. Another possibility is to use soft constraints in order to enforce axioms that are not entirely true. For example *married with* and *partner of* are not entirely semantic equivalents but their embeddings are similar to one another. Intuitively the objective of soft constraints is to nudge the model to adopt some desired properties rather than enforce hard-coded requirements. This is done by adding two weighted terms to the usual training loss: $\hat{\mathcal{L}} = \mathcal{L} + \lambda \left[\sum_{r_1 \equiv r_2} ||\vec{r_1} - \vec{r_2}||_2^2 + \sum_{r_1 \equiv \bar{r_2}} ||\vec{r_2} - \Phi(\vec{r_1})||_2^2\right]$ where λ is an hyper-parameter that needs to be determined during training.

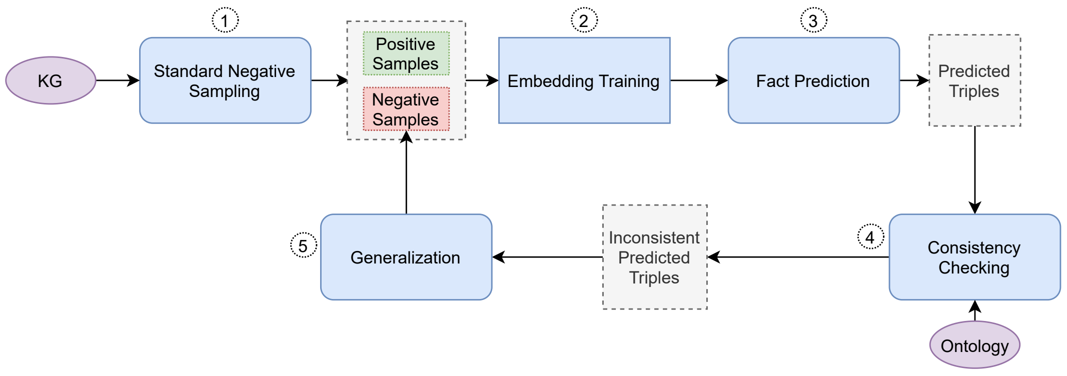

Figure 3 *ReasonKGE* Framework.

4.5 Improving Knowledge Graph Embeddings with Ontological Reasoning

Until now, we have concentrated mainly on very simple types of axioms to improve embeddings. ReasonKGE [24] is a method that can use complex constraints as well. The idea is to use symbolic reasoning to find predictions by the model that are logically inconsistent, and to feed these as negative samples into a retraining step. Traditionally, embedding methods generate negative triples by randomly replacing the head entity or tail entity in a triple from the KB (Section 2). This method, however, has two problems: First, as we have already discussed, it does not work as well for the head entities (Section 2.1). Furthermore, traditional methods do not necessarily create negative statements that violate domain and range constraints. For example, a triple such as ⟨*Elvis*, *hasNationality*, *Priscilla*⟩ cannot be true since *hasNationality* requires a country as object. If such triples are not generated as negative examples, the model may produce them as predictions. Therefore, ReasonKGE sets out to generate negative examples by axioms – inspired by the NELL system [8], which also uses axioms for the generation of examples.

The framework of the proposed method is shown in Figure 3. The inputs of the framework are the KG and its ontology, whereas the outputs are negative samples, which can then be used for training the model in the next iteration. The first iteration simply generates the baseline model with a default sampling method. Here, traditional sampling methods are used to generate the negative facts, and the model is trained based on positive and negative facts to obtain the predictions. The predicted triples are checked for inconsistencies with respect to the underlying ontology with the help of a reasoner. The inconsistent triples are then generalized to other semantically similar triples which would also cause the same inconsistencies. Lastly, all the generated negative samples are fed back to the model for the next iteration of training. With each round of training, the model learns to identify inconsistencies and therefore make more consistent predictions.

The **consistency checking procedure** (step 4) is one of the main steps, that detects which predictions made by embedding model are inconsistent with the original KG ($\mathcal{G}$) and ontology $\mathcal{O}$. For computational purposes, this check is done only on the subset of relevant facts. The *relevant set* is defined as follows:

$$Relv(\alpha, \mathcal{G}) = \{\alpha\} \cup \{\beta \in \mathcal{G} | Ent(\beta) \cap Ent(\alpha) \neq \varnothing\} \tag{5}$$

Here, α is the predicted triple and β are triples in the KG. For example, consider α = ⟨*Samsung*, *locatedIn*, *Emmanuel Macron*⟩. For this prediction the relevant set could consist of the following triples:

$$\begin{aligned}Relv(\alpha, \mathcal{G}) = \{&\langle Emmanuel\ Macon, livesIn, France\rangle,\\ &\langle Emmanuel\ Macron, spouse, Brigitte\ Macron\rangle,\\ &\langle Emmanuel\ Macron, type, person\rangle,\\ &\langle Samsung, type, company\rangle\}\end{aligned}$$

It is sufficient to perform consistency checking on $Relv(\alpha, \mathcal{G}) \cup \mathcal{O}$, instead of $\alpha \cup \mathcal{G} \cup \mathcal{O}$. In our example, a reasoner would flag this prediction as inconsistent, because the ontology tells us that *locatedIn* requires a location, and hence ⟨*Samsung*, *locatedIn*, *Emmanuel Macron*⟩ implies that the type of *Emmanuel Macron* is *location.* That contradicts the fact ⟨*Emmanuel Macron*, *type*, *person*⟩, together with the axiom that states that people and locations are disjoint. Thus, this triple can serve as a negative sample.

Further, in Step 5, the negative samples obtained via consistency checking are fed to a **generalization** module to obtain multiple similar inconsistent facts that have a similar structure within KG. This is beneficial in 2 ways: firstly, by generating several negative samples that cause the same inconsistency, the model would be able to learn the inconsistency pattern and thus, the prediction of similar incorrect triples in next training iterations would be avoided. Secondly, it enables us to obtain sufficient number of negative samples for a given triple during the training of the model. The generalization of inconsistent predictions is done in the following way: in an inconsistent predicted fact $\langle h, r, t\rangle$, t can be replaced with another entity k that has similar triples. For example, if α = ⟨*Samsung*, *locatedIn*, *Emmanuel Macron*⟩ is a predicted inconsistent triple, then we can take $\alpha = \langle Samsung, locatedIn, Joe\ Biden\rangle$ as another negative example if *Joe Biden* has the same neighbour triples as *Emmanuel Macron*, i.e. ⟨*Joe Biden*, *type*, *person*⟩.

The authors show experimentally that *ReasonKGE* achieved better results on the link prediction task as compared to traditional methods (TransE, ComplEx). Experiments conducted on the Yago3-10 dataset were particularly significant, as the model achieved more than 10% improvement for all the measures as compared to TransE. Additionally, *ReasonKGE* reduced the ratio of inconsistent predictions over the test set when compared to other models that employ static or random sampling techniques. A limitation of this method is the use of DL-Lite [3] ontologies, due to which, theoretically, not all possible similar negative samples will be obtained based on a given inconsistent prediction in the generalization step.

4.6 Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs

Similar to *ReasonKGE*, this paper proposes to improve KG embeddings by injecting available background knowledge in the form of ontological axioms [14]. The authors propose *TransOWL* and *TransROWL* models, as improved versions of the traditional embedding methods TransE and TransR respectively.

The injection of background knowledge during the training phase involves two main components - *reasoning* to add negative samples and *Background Knowledge (BK) injection* to add constraints on the scoring function.

- During **reasoning**, negative samples are generated by leveraging the ontological properties such as *domain*, *range*, *disjointWith*, *functionalProperty* with the help of the Apache Jena framework. For example, if a particular entity type (or concept in ontology terminology), let's say *location* is disjoint with another type e.g. *person*, then negative samples are generated by replacing the person entity in a triple with all location entities present in the

KG. Thus, for a triple $\langle Samsung, locatedIn, South\ Korea\rangle$, a list of negative samples can be generated by replacing *South Korea* with *Joe Biden*, *Barack Obama*, *John Smith* and so on.

- During **BK injection**, ontological properties such as *equivalentClass*, *equivalentProperty*, *inverseOf* and *subClassOf* are applied for the definition of additional constraints on the scoring function such that resulting embedding vectors can reflect these properties. New triples corresponding to these properties are generated and added to the training set of the model. For example, for the *equivalentClass* property, if class A is equivalent to class B, then for a triple $\langle entity1, type, A\rangle$, it is possible to generate another triple $\langle entity1, type, B\rangle$ as well. Similarly this is performed for other properties as well and a considerable number of additional triples is generated before training the model.

The basic loss function for TransE is defined as

$$\sum_{\langle h,r,t\rangle\in\Delta,\langle h',r,t'\rangle\in\Delta'} [\gamma + f_r(t,h) - f_r(t',h')] \tag{6}$$

here $\gamma\ \geq 0$ is the hyperparameter *margin*. For TransOWL, this loss function is more complex due to the additional constraints from the axioms. For example, the addition of the the *inverseOf* axiom would add a term to the loss function as

$$\sum_{\langle t,s,h\rangle\in\Delta,\langle t',s,h'\rangle\in\Delta'} [\gamma + f_s(t,h) - f_q(t',h')] \tag{7}$$

where f is the scoring function, Δ refers to the set of additional triples generated by a reasoner and s is the inverse relation of r. Similarly, the constraints are added in the loss function for the other axioms as well. Experimental evaluation shows that the models generated through this procedure show improvement for link prediction as well as triple classification in KGs as compared to the original TransE and TransR models.

4.7 Knowledge Base Completion Using Embeddings and Rules

In [63], the authors propose to constrain knowledge graph embeddings by an altogether different type of axioms: cardinality axioms. This is done by an Integer Linear Programming problem: the objective function is computed using the scoring function of an embedding model under the constraints from the symbolic axioms.

Let the $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$ and $\mathcal{R} = \{r_1, r_2, \dots, r_m\}$ be the sets of entities and relations in a KG at hand. The linear problem is defined with mn^2 decision variables $\{x^k_{i,j}, 1 \leq i, j \leq n, 1 \leq k \leq m\}$ such that $x^k_{i,j}$ indicates whether the fact $\langle e_i, r_k, e_j\rangle$ is true or false. The weight of a triple is computed using the scoring function f of an embedding model. This results in an objective function of the form:

$$\max_{x^k_{i,j}} \sum_i \sum_j \sum_k f(\langle e_i, r_k, e_j\rangle) \cdot x^k_{i,j}$$

The constraints of this optimization problem are derived from four types of rules:

- **Type 1: noisy observation**. Observed triples are very likely to be true but KBs are prone to noise. In order to take into account the rare cases in which an observed fact is false, slack variables $\epsilon^k_{i,j}$ are introduced for each observed triple and the R1 constraint is added along with a penalization term in the objective function. This is a classical method in linear programming, which allows the easy identification of noisy triples.

- **Type 2: argument type expectation**. Some predicate-specific type constraints should be respected by the head and tail entities. This results in the R2 constraint in which $\mathcal{S}_k$ (resp. $\mathcal{O}_k$) contain the indexes of the entities that have the type of the head (resp. tail) of the relation r_k.
- **Type 3: at-most-one restraint**. Some relations can handle at most one head per tail (many-to-one) or one tail per head (one-to-many). For example, the relation *cityLocatedInCountry* is a one-to-many relation meaning that a city can be located in at most one country. Other relations are one-to-one. Those three types of relations result in three constraints R3.1, R3.2 and R3.3 in which $\mathcal{R}_{1-M}$, $\mathcal{R}_{M-1}$ and $\mathcal{R}_{1-1}$ are respectively the sets of one-to-many, many-to-one and one-to-one relations.
- **Type 4: simple implication**. A relation r_1 can imply another relation r_2, if $\langle x, r_1, y\rangle \Rightarrow \langle x, r_2, y\rangle$ for any entities x and y. It is denoted $r_1 \Rightarrow r_2$. This gives us the constraint R4.

With this, the final Integer Logic Program is:

$$\max_{x_{i,j}^k} \sum_i \sum_j \sum_k f(\langle e_i, r_k, e_j\rangle) \cdot x_{i,j}^k$$

$$(R1)\ x_{i,j}^k + \epsilon_{i,j}^k = 1, \forall (i,j,k) : \langle e_i, r_k, e_k\rangle \text{ is observed}$$

$$(R2)\ x_{i,j}^k = 0, \forall k, \forall i : e_i \notin S_k, \forall j : e_j \notin O_k$$

$$(R3.1)\ \sum_i x_{i,j}^k \leq 1, \forall k : r_k \in \mathcal{R}_{1-M}, \forall j$$

$$(R3.2)\ \sum_j x_{i,j}^k \leq 1, \forall k : r_k \in \mathcal{R}_{M-1}, \forall i$$

$$(R3.3)\ \sum_i x_{i,j}^k \leq 1, \sum_j x_{i,j}^k \leq 1, \forall k : r_k \in \mathcal{R}_{1-1}, \forall i, \forall j$$

$$(R4)\ x_{i,j}^{k_1} \leq x_{i,j}^{k_2}, \forall k_1, k_2 \text{ s.t. } r_{k_1} \Rightarrow r_{k_2}$$

$$\text{where } x_{i,j}^k \in \{0,1\}, \forall i,j,k;\ \epsilon_{i,j}^k \in \{0,1\}, \forall (i,j,k) : \langle e_i, r_k, e_k\rangle \text{ is observed}$$

In spite of their promising results, the authors highlight two main limitations to this approach. First, constraints do not take into account the possible many-to-many relations, possibly missing out some ontology information. Second, solving the integer linear programming problem is time consuming and the approach then lacks scalability. In this regard, the authors propose a divide-and-conquer strategy for future work.

4.8 Jointly embedding KGs and Rules

So far, we have constrained embeddings by axioms. There are, however, also approaches that use soft rules instead of axioms, and that learn embeddings jointly with confidence scores for these soft rules. The first of these [19] improves the training procedure of the TransE model [7] by a new training loss that integrates both observed triples and groundings of some logical rules. The method focuses on rules of only two shapes: $\forall x, y,\ \langle x, r_1, y\rangle \Rightarrow \langle x, r_2, y\rangle$ and $\forall x, y,\ \langle x, r_1, y\rangle \cap \langle y, r_2, z\rangle \Rightarrow \langle x, r_3, z\rangle$, where r_1, r_2 and r_3 are relations from the graph.

Following Rocktäschel et al. [45], the truth value of a grounded rule is computed from the truth values of the constituent triples and t-norm logic principles. For this, the truth value of a single triple is first defined as:

$$f(\langle x, r, y\rangle) = 1 - \frac{1}{3\sqrt{d}} ||\vec{x} + \vec{r} - \vec{y}||$$

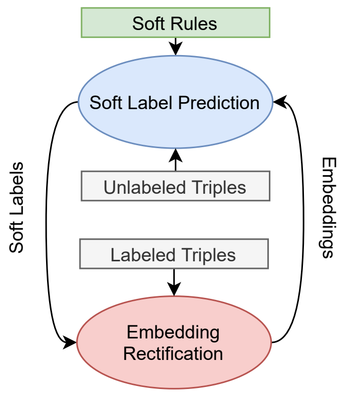

Figure 4 RUGE.

This is simply a normalization of the TransE [7] scoring function. To compute the truth value of more complicated logical formulae, this definition has to be broaden to negation, conjunction, and disjunction. The truth value of a negated triple $\neg p$ is simply $1 - f(p)$. The truth value of a conjunction is given by a t-norm, i.e., a function that is commutative, associative, and monotonous, and that has 1 as the identity element. The work of [19] uses simply the product as the t-norm, i.e., $f(p \wedge q) = f(p) \times f(q)$. With this, the truth value f of an implication is

$$\begin{aligned} f(p \Rightarrow q) &= f(\neg p \vee q) = f(\neg(p \wedge \neg q)) \\ &= 1 - f(p) \times (1 - f(q)) \\ &= 1 - f(p) + f(p) \times f(q) \end{aligned}$$

The only question left is how to generate the logical rules that are taken as input of this improved training procedure. A natural method could be to run a logical approach such as AMIE or RuDiK [17, 40]. The authors of [19] have a different approach that uses their method of scoring rule groundings to select the best ranking rules in a greedy manner.

4.9 Knowledge Graph Embedding with Iterative Guidance from Soft Rules (RUGE)

The Rule-Guided Embedding (RUGE) algorithm [20] is another method that learns embeddings jointly with confidence scores for logical rules. Its main steps are shown in Figure 4. The system starts out with soft rules (top of the figure), mined by the AMIE system [17]. These are instantiated to make predictions (see Section 2.1) – each with a confidence. The Embedding Step (bottom of the figure) takes as input the predictions of the rules as well as labeled triples from the KB. The embedding is trained on these two sources. This allows the prediction of new facts, which will in turn predict new facts by help of the rules. This process is iterated, thereby amplifying automatically the number of labeled examples.

The rule mining system gives each rule a confidence. However, this confidence concerns the rule as a whole, not an individual grounded variant of the rule, where all variables are instantiated. To compute the confidence of an individual grounded rule, the approach uses the scores $\phi(\cdot)$ of the embeddings of the facts that appear in the rule, as well as the score $s(\cdot)$ of the fact that the rule predicts, and proceeds according to the definitions of t-norm based fuzzy logics, in much the same way as [19].

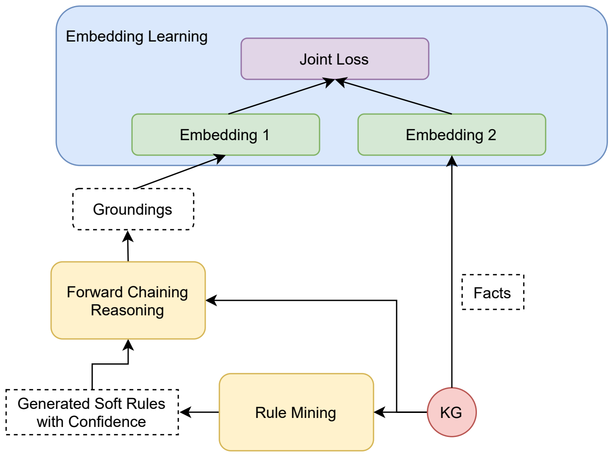

Figure 5 SoLE Architecture (stage 1 in yellow, stage 2 in blue).

The approach then aims to find a scoring function $s(\cdot)$ that is as close as possible to the current scoring function $\phi(\cdot)$, while at the same time making the confidences $\pi(\cdot)$ of all grounded rules as close as possible to 1 (the maximum). This is done by solving an optimization problem. This yields scores $s(\cdot)$ for facts that are predicted by the rules.

In the second step, the approach then corrects its embeddings $\phi(\cdot)$ so that they mirror (1) the truth value of facts that appear already in the KB and (2) the score $s(\cdot)$ for facts that do not appear in the KB, but were predicted by the rules. This updated embedding is then fed again into the rules, and the process is iterated. Experiments show that this method achieves significant improvements in link prediction task on Freebase and YAGO.

A variant of this approach is the SoLE system [71] ("Soft Logical rules enhanced Embeddings"), whose architecture is shown in Figure 5. Like RUGE, SoLE takes as input a KB and rules. It uses the rules to predict new facts in an iterative manner until no more facts can be predicted (a technique called *forward chaining*). The rules are then grounded, and a confidence score is computed for each grounded rule, not unlike this is done in RUGE as well. Different from RUGE, SoLE then minimizes a joint loss so as to find embeddings that can (1) predict the labels of triples contained the KB, while also (2) imitating the confidences of rules.

5 Rule Mining with embedding techniques

In the previous section, we have discussed several embedding methods that use logical techniques to improve their performance. The other direction is much less common: there are few methods that use embedding models in order to improve logical rule mining techniques. We will now present the most prominent ones.

5.1 ILP Rule Mining

A first small application of embeddings for rule mining is presented in an extension of RuDiK [40] by Ahmadi et al. [1]. The new system can also mine rules about class membership, such as "Politicians are not married to officeholders of a different party":

$$\begin{aligned}&[\langle x, party, x_p\rangle \wedge \langle y, party, y_p\rangle \wedge x_p \neq y_p \\ &\qquad \wedge \langle x, type, Politician\rangle \wedge \langle y, type, \mathit{Office\ Holder}\rangle] \Rightarrow \neg\langle x, spouse, y\rangle\end{aligned}$$

The question is now what classes should be considered in such rules. Considering all classes may lead to rules that are too fine-grained. It would also be inefficient. Using only the top-level classes, in contrast, may miss out on useful rules that hold in a subclass.

RuDiK therefore clusters the instances of the KB. The method of choice here are entity embedding methods. The authors observe that the clusters obtained this way are more uniform in what concerns the structural similarity of entities (i.e., the outgoing relations that they share) than class membership. This is because two entities with different relations can belong to the same class, and entities with the same relations can belong to different classes. The embedding, in contrast, groups entities by their relations, which is more amenable to the rule mining.

It turns out that entities with popular classes, such as *Person*, can be spread across multiple clusters, but classes with finer granularity, such as *Politician* and *OfficeHolder* are grouped together. For each cluster, RuDiK then determines a class (e.g., the class that most entities in the cluster belong to). This class is then used for mining rules such as the one above.

5.2 Few-shot learning for label propagation

As stated in Section 2.1, a recurrent problem when working on KBs is the lack of negative statements. That makes it difficult to classify a prediction of any model. In an ideal situation, an operator would be available during training in order to manually tag generated facts as positive or negative. This is rarely the case because it is very costly but it could be very useful in the generation of false statements for example. This problem of manually tagging samples (here triples) is not specific to KB processing and it has given birth to a field of research called few-shot learning. This is the study of learning algorithms that work on a very small number of samples. It often applies in fields were the creation of supervision labels is costly, for example computer vision.

In [32] the authors propose a few-shot rule-based knowledge validation framework that uses an embedding model (HypER [5]) in order to propagate the decisions of a human operator to whom triples to tag are submitted. The goal of the method is to enrich the KB with positive and negative examples that allow a better evaluation of a set of rules. The proposed *propagation* method relies on a measure of similarity between facts. To compute the similarity, a vector representing each triple is computed by concatenating the embeddings of the entities. The propagation of manual labels is done locally to triples sharing the same relation, and so their embedding is omitted in the concatenation. For example, let's say that an operator labeled the fact $\langle \mathit{Barack\ Obama}, \mathit{marriedTo}, \mathit{Sasha\ Obama} \rangle$ as false, this label is going to be propagated to triples involving *Barack Obama* and *MarriedTo* or *MarriedTo* and *Sasha Obama* that are similar enough to the initial one. Eventually the set of manually labeled triples along with the automatically labeled ones improve the evaluation process of the rules. The authors apply their method to rules mined with AMIE [17] and RuDiK [40]. The proposed method uses HypER as embedding model but the authors insist on the fact that any model can be used for this task.

5.3 Approximate algorithms

AMIE [17] is an exhaustive rule mining system, i.e., it finds all rules above user-specified confidence and support thresholds. This makes AMIE quite heavy to run on large knowledge bases. AMIE+ [18] improved the runtime by approximating the computation of the confidence value of rules. This comes at a minor cost in the precision of the algorithm but allows reducing the computation time by several orders of magnitude.

Another way to speed up the rule mining is by *sampling*. The underlying intuition is that a rule of the form $\langle e, r_1, e_1\rangle \wedge \langle e_1, r_2, e_2\rangle \wedge \dots \wedge \langle e_n, r_{n+1}, e'\rangle \Rightarrow \langle e, r, e'\rangle$ can be seen as the co-occurrence in the knowledge graph (KG) of two paths from e to e': one of length 1 (passing through the relation r), and one of length $2n+1$ (through the entities e_1, e_2, ..., e_n and the relations r_1, r_2, ..., r_{n+1}). Exploring the possible rules then comes down to finding possible paths from one entity to another. This graph exploration is computationally expensive, and so the authors of [39] propose a two-step acceleration of the graph exploration and of the evaluation of the rules.

- First, the size of the graph is reduced by *sampling* the KG. Given a relation r that should appear in the head atom of the rule and a maximum length $l \geq 2$, the neighborhood of r is computed iteratively. We start from a set E_0, which includes any entity involved in a fact with r. We then compute E_i for $1 \leq i \leq l-2$ by including entities linked to some entity of E_{i-1} by any predicate. The neighborhood of r is then defined as $\mathcal{N}(r) = \cup_{i=0}^{l-2} E_i$ and it includes all entities relevant to find paths of maximum length l and then rules involving l atoms in the body and p in the head.
- Subsequently, instead of exhaustively exploring all the possible paths in the neighborhood of the relation r, the authors suggest to use a bilinear embedding model to learn matrix representations of relations (see Section 3.2). A relation path $r_1, r_2, \dots, r_l$ in the graph can then be represented as the product of the matrices of the relations $M_{r_1} \cdot M_{r_2} \cdot \dots \cdot M_{r_l}$. The similarity between the path (corresponding to the body of the potential rule) and r is computed using the matrix Frobenius norm $sim(r, [r_1, r_2, \dots, r_l]) = \exp(-||M_r - M_{r_1} \cdot M_{r_2} \cdot \dots \cdot M_{r_l}||_F)$.

The authors compare their approach to AMIE+, and show that the new approach mines more rules, and rules of better quality in terms of confidence. Furthermore, the process is much faster for rules that have the shape of paths.

6 Conclusion

Knowledge Bases (KBs) find many uses in AI applications, such as personal assistants, question answering systems, or text analysis. And yet, KBs are usually incomplete and miss facts. Two avenues of research have taken to predict missing facts: a symbolic one, based on rule mining, and a neural one, based on embeddings. Each of them has their respective strengths, and in this article we have presented an overview of both. We have also discussed recent studies on the criticism of the benchmark and protocols used during evaluation of embedding models. We have then presented approaches that successfully combine both symbolic and neural methods to perform fact prediction in KBs. While there are several approaches that use rules in order to improve embeddings, there are rather few approaches that use embeddings to improve rule mining. This may thus be an interesting direction for further research.

References

1 Naser Ahmadi, Viet-Phi Huynh, Vamsi Meduri, Stefano Ortona, and Paolo Papotti. Mining expressive rules in knowledge graphs. *Journal of Data and Information Quality (JDIQ)*, 12(2):1–27, 2020.

2 Farahnaz Akrami, Mohammed Samiul Saeef, Qingheng Zhang, Wei Hu, and Chengkai Li. Realistic re-evaluation of knowledge graph completion methods: An experimental study. In *ACM SIGMOD*, 2020.

3 Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The dl-lite family and relations. *Journal of artificial intelligence research*, 36:1–69, 2009.

4 Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, 2007.

5 Ivana Balažević, Carl Allen, and Timothy M Hospedales. Hypernetwork knowledge graph embeddings. In *ICANN*, 2019.

6 Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

7 Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, 2013.

8 Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

9 Yuanfei Dai, Shiping Wang, Neal N Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5):750, 2020.

10 Tu Dinh Nguyen Dai Quoc Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *NAACL*, 2018.

11 Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

12 Boyang Ding, Quan Wang, Bin Wang, and Li Guo. Improving knowledge graph embedding using simple constraints. *arXiv preprint*, 2018. `arXiv:1805.02408`.

13 Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *ACM SIGKDD*, 2014.

14 Claudia d'Amato, Nicola Flavio Quatraro, and Nicola Fanizzi. Injecting background knowledge into embedding models for predictive tasks on knowledge graphs. In *ESWC*, 2021.

15 Bahare Fatemi, Siamak Ravanbakhsh, and David Poole. Improved knowledge graph embedding using background taxonomic information. In *AAAI*, volume 33, 2019.

16 Jun Feng, Minlie Huang, Mingdong Wang, Mantong Zhou, Yu Hao, and Xiaoyan Zhu. Knowledge graph embedding by flexible translation. In *KR*, 2016.

17 Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.

18 Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. In *VLDBJ*, 2015.

19 Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *EMNLP*, 2016.

20 Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Knowledge graph embedding with iterative guidance from soft rules. In *AAAI*, 2018.

21 Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao. Learning to represent knowledge graphs with gaussian embedding. In *CIKM*, 2015.

22 Frank Lauren Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6:164–189, 1927.

23 Nitisha Jain, Jan-Christoph Kalo, Wolf-Tilo Balke, and Ralf Krestel. Do embeddings actually capture knowledge graph semantics? In *ESWC*, 2021.

24 Nitisha Jain, Trung-Kien Tran, Mohamed H Gad-Elrab, and Daria Stepanova. Improving knowledge graph embeddings with ontological reasoning. In *ISWC*, 2021.

25 Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *ACL*, 2015.

26 Rudolf Kadlec, Ondřej Bajgar, and Jan Kleindienst. Knowledge base completion: Baselines strike back. In *RepL4NLP*, 2017.

27 Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *NeurIPS*, 2018.

28 Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint*, 2016. `arXiv:1609.02907`.

29 Bhushan Kotnis and Vivi Nastase. Analysis of the impact of negative sampling on link prediction in knowledge graphs. *arXiv preprint*, 2017. `arXiv:1708.06816`.

30 Jonathan Lajus, Luis Galárraga, and Fabian M. Suchanek. Fast and Exact Rule Mining with AMIE 3. In *ESWC*, 2020.

31 Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2015.

32 Michael Loster, Davide Mottin, Paolo Papotti, Jan Ehmüller, Benjamin Feldmann, and Felix Naumann. Few-shot knowledge validation using rules. In *TheWebConf*, 2021.

33 Farzaneh Mahdisoltani, Joanna Asia Biega, and Fabian M. Suchanek. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*, 2015.

34 Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. An introduction to anyburl. In *KI*, 2019.

35 Pasquale Minervini, Luca Costabello, Emir Muñoz, Vít Nováček, and Pierre-Yves Vandenbussche. Regularizing knowledge graph embeddings via equivalence and inversion axioms. In *ECML PKDD*, 2017.

36 Sameh K Mohamed, Vít Novácek, Pierre-Yves Vandenbussche, and Emir Muñoz. Loss functions in knowledge graph embedding models. *DL4KG@ ESWC*, 2377:1–10, 2019.

37 Brian Murphy, Partha Talukdar, and Tom Mitchell. Learning effective and interpretable semantic models using non-negative sparse embedding. In *COLING*, 2012.

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

39 Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. An embedding-based approach to rule learning in knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1348–1359, 2021.

40 Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Robust discovery of positive and negative rules in knowledge bases. In *ICDE*, 2018.

41 Pouya Pezeshkpour, Yifan Tian, and Sameer Singh. Revisiting evaluation of knowledge base completion models. In *AKBC*, 2020.

42 Jay Pujara, Eriq Augustine, and Lise Getoor. Sparsity and noise: Where knowledge graph embeddings fall short. In *EMNLP*, 2017.

43 Simon Razniewski, Hiba Arnaout, Shrestha Ghosh, and Fabian M. Suchanek. Completeness, Recall, and Negation in Open-World Knowledge Bases. In *VLDB*, 2021.

44 Simon Razniewski, Fabian M. Suchanek, and Werner Nutt. But What Do We Actually Know? In *AKBC workshop*, 2016.

45 Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *NAACL*, 2015.

46 Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49, 2021.

47 Andrea Rossi and Antonio Matinata. Knowledge graph embeddings: Are relation-learning models learning relations? In *EDBT/ICDT*, 2020.

48 Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *ICLR*, 2019.

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

50 Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018.

51 Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *AAAI*, 2019.

52 Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *NeurIPS*, 2013.

53 Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago - A Core of Semantic Knowledge . In *WWW*, 2007.

54 Fabian M. Suchanek, Jonathan Lajus, Armand Boschin, and Gerhard Weikum. Knowledge Representation and Rule Mining in Entity-Centric Knowledge Bases. In *RW*, 2019.

55 Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2018.

56 Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *ACL*, 2020.

57 Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *CVSC workshop*, 2015.

58 Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

59 Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *ICLR*, 2019.

60 Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

61 Thanh Vu, Tu Dinh Nguyen, Dat Quoc Nguyen, Dinh Phung, et al. A capsule network-based embedding model for knowledge graph completion and search personalization. In *NAACL*, 2019.

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

63 Quan Wang, Bin Wang, and Li Guo. Knowledge base completion using embeddings and rules. In *ICOAI*, 2015.

64 Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, Samuel Broscheit, and Christian Meilicke. On evaluating embedding models for knowledge base completion. In *RepL4NLP*, 2019.

65 Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.

66 Gerhard Weikum, Luna Dong, Simon Razniewski, and Fabian M. Suchanek. Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases. In *Foundations and Trends in Databases*, 2021.

67 Alfred North Whitehead and Bertrand Russell. *Principia mathematica*. Cambridge University Press, 1913.

68 Han Xiao, Minlie Huang, Yu Hao, and Xiaoyan Zhu. Transg: A generative mixture model for knowledge graph embedding. *arXiv preprint*, 2015. `arXiv:1509.05488`.

69 Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint*, 2014. `arXiv:1412.6575`.

70 Rui Ye, Xin Li, Yujie Fang, Hongyu Zang, and Mingzhong Wang. A vectorized relational graph convolutional network for multi-relational network alignment. In *IJCAI*, 2019.

71 Jindou Zhang and Jing Li. Enhanced knowledge graph embedding by jointly learning soft rules and facts. *Algorithms*, 12(12):265, 2019.

72 Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *AAAI*, 2020.

73 Xiang Zhao, Weixin Zeng, Jiuyang Tang, Wei Wang, and Fabian M. Suchanek. An Experimental Study of State-of-the-Art Entity Alignment Approaches . In *TKDE*, 2020.