Chapter 4
Advanced Topic: Knowledge Graph Completion

4.1 Introduction

Information extraction and entity resolution are clearly both important steps in domain-specific knowledge construction [66, 111]. However, even when done with high accuracy, they are rarely enough. Knowledge graphs constructed over raw data very often have missing and noisy information, including incorrect triples and missing relations. Put simply, such knowledge graphs have to be *refined* or 'completed' before they can be deployed in a good application [139, 145]. An example is illustrated in Fig. 4.1. The knowledge graph fragment in the figure describes political figures and their affiliations, and is possibly extracted from news articles. We assume for the moment that the entities and relations have been correctly extracted and reconciled (i.e. the techniques in Chaps. 2 and 3 achieved excellent performance). Given this KG, if we were to execute a query asking who the first lady was under President Barack Obama's presidency, we would not get any answer. On the other hand, we would get a response from the system if we replaced President Barack Obama in the query with President George Bush. This is because the fact that Laura Bush was the first lady in the Bush administration has been explicitly extracted from a source, perhaps because it was mentioned, while the same is not true for First Lady Michelle Obama. In general, it is not reasonable to assume that every possible fact or inference is ever going to be explicitly extracted from a raw input data source. Sometimes (as in the case above), this is because the 'missing' fact is not mentioned in the source explicitly, but many times, it is also because of noise in the extraction system. Similar reasoning can be applied to the presence of 'wrong' links.

In its broadest form, knowledge graph completion would take a graph with missing and wrong edges and nodes, and attempt to both correct and complete the graph. In other words, knowledge graph correction is included within knowledge graph completion. In the case of Fig. 4.1, a 'good' system would be able to take the graph and infer the fact that Michelle Obama was first lady in the Obama

M. Kejriwal, *Domain-Specific Knowledge Graph Construction*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-3-030-12375-8_4

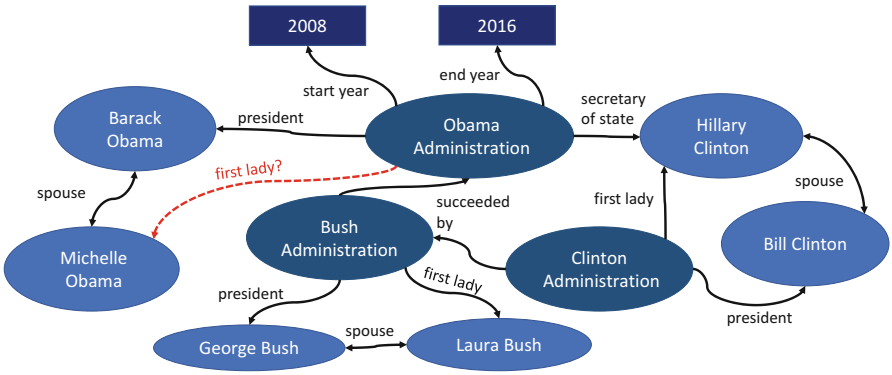

Fig. 4.1 A simplified illustration of the knowledge graph completion problem

administration. A 'bad' system would add noisy links, or remove correct links by incorrectly labeling them as noisy. Usually, the picture is much more nuanced and the evaluation of a knowledge graph completion system involves assessing whether, despite the potential introduction of some noise, the system ended up providing an overall benefit to knowledge graph quality.

Why is there reason to believe that knowledge graph completion works? One can intuitively see that the global graph exhibits some 'semantic regularities' that could be exploited. For example, if we had observed ten presidents in the KG, and found that 90% of their spouses were also explicitly designated first ladies in the KG, it is reasonable to believe that the other 10% are also first ladies, despite no explicit evidence. One can also see why this kind of inference can be a problem. The question of when and where knowledge graph completion is useful, and when it should be avoided has not been fully addressed by the research community. One disadvantage of completion, and of any method that relies solely on inference, is that state-of-the-art neural methods typically no provenance or 'explanation' of why some link was predicted between two nodes, or why some link was declared as noisy.

With these caveats in mind, we argue that knowledge graph completion is still a very useful, and actively researched, area within the broader community of knowledge graph construction. Multiple classes of methods have been proposed for the problem over the years. Before the modern renaissance of deep learning and neural networks, probabilistic graphical models constituted the primary line of attach for such 'relational' problems. Markov logic networks were particularly popular in the mid 2000s, but for various reasons, including scalability, were supplanted by models like probabilistic soft logic that offered a good tradeoff between expressibility, optimization and representation. Probabilistic soft logic and its variants have continued to be popular for various tasks, but the dominant line of research in the knowledge graph completion community (at the time of writing), and the one that we subsequently describe, is *knowledge graph embeddings* (KGEs) [176].

4.2 Knowledge Graph Embeddings

Knowledge Graph Embeddings (KGEs) are an applied offshoot of a broader emergent research area called *representation learning* [14]. In the 2000s, as machine learning systems continued to proliferate, it was realized (somewhat disquietingly) that the performance of a machine learning system was usually dependent very heavily on the *features* engineered over the datasets, as opposed to the virtues of the machine learning algorithms themselves. Feature engineering tended to be manual and ad-hoc, and in the general case, there was no good reason to suppose why one feature would perform better than another. Researchers also realized that the 'goodness' of a feature set also had a lot to do with the dataset itself i.e. it was quite possible for a particular set of features to perform well on one dataset but not another, all else being the same. Clearly, a less manually intensive, data-driven approach to learning good representations for data (whether it was natural language data like words and sentences, or image data) was motivated.

In the natural language community, representation learning algorithms like word2vec have become fairly standard and are widely used across the board for a range of tasks [118], especially information extraction (relation extraction, event extraction and named entity recognition). The key idea is to slide a window over text and to *embed* each word into a dense, real-valued vector space (typically between 50–200 dimensions) that is low-dimensional compared to alternatives like tf-idf, which require dimensionality in the range of 50,000–1 million, depending on the size of the language's vocabulary. The optimization function used for the embedding takes into account the other words in the window, called a *context*. Intuitively, words that have similar context would be embedded close together in the vector space. In natural language, this generally leads to common-concept instances (such as cat and dog) being embedded close together due to their similar context. This kind of embedding is reminiscent of topic models like Latent Dirichlet Allocation (LDA) [29], but LDA was a graphical (not neural) model designed to embed *documents*. In contrast, algorithms like word2vec are designed to embed words, based on context, rather than coarser-grained documents, although variants of word2vec can also be used to embed sentences, paragraphs and documents [51].

Because of their semantic dependence on context, rather than ontology, embeddings based on statistical models have been found to capture some remarkable analogical patterns in a completely unsupervised fashion (Fig. 4.2). Despite not being given ontological information, the embedding is able to deduce that words like 'generator' and 'battery' should be clustered closer together in a semantic space, rather than (say) 'teaching' and 'generator'. In the embedding space, one can also carry out vector operations like **King** − **Man** + **Woman** with the resulting vector being very close to **Queen** in the semantic space.

Primarily because of these semantic properties, and also (on a related note) because of superior performance on downstream natural language processing tasks like information extraction, the word2vec algorithm became so popular that numerous variants have emerged, and the algorithm has even been adopted to embed

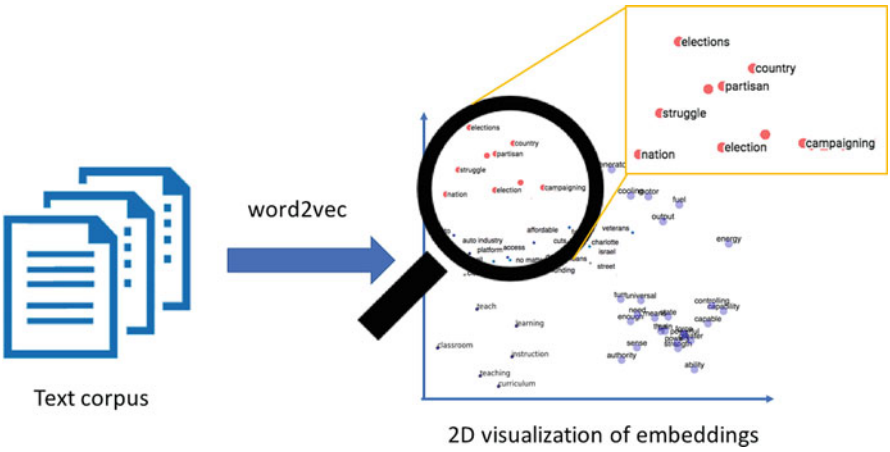

Fig. 4.2 An illustration of word embeddings using algorithms like word2vec. Words that occur in similar contexts (e.g., elections, campaigning) are clustered closer together in the vector space. 2D visualizations like these (from higher-dimensional vectors) can be rendered using neural dimensionality reduction algorithms like t-SNE [108]. Note that dimensions do not have any intrinsic meaning

nodes in networks and graphs (see e.g., the DeepWalk algorithm [141]). However, its application or adoption to knowledge graphs is not clear, and has not been usefully demonstrated. In part, the reason is that knowledge graphs are a richer, more structured data set, since even the simplest definition of a KG assumes that is a multi-relational, directed, labeled graph where entities are nodes and relations are different types of edges. Motived by this additional structure, novel approaches were proposed in the early 2010s to embed nodes (and often, but not always, relations) in the KG into a continuous vector space while preserving certain key structural properties.

In the rest of this section, it will be useful to think of a KG as a set of triples, where each triple is of the form (h, r, t), where h and t are referred to as head and tail entities respectively, and r is the relation. We do not assume constraints, although models like RDF impose many requirements on how relations and head/tail entities are actually represented. For example, the RDF model does not allow head entities to be modeled as 'literals' like strings or numbers. One reason for ignoring such constraints in the present discussion is that typical (and early) papers on knowledge graph embeddings have mostly arisen in the NLP and general AI communities, rather than the Semantic Web community, which is the major adopter of RDF. Although some recent work has attempted to embed knowledge graphs modeled specifically as RDF, even these models tend to be heavily inspired by the earlier models that were proposed in 'ontologically light' communities.

Furthermore, although many embedding models exist at the time of writing, almost all models represent h and t as points in the vector space, while relation r usually has a more flexible representation, since it is modeled as an *operation*. Thus,

it could be a vector, representing operations such as translation or projection, but in some cases, it can also be a matrix. In contrast, representing an entity as a matrix is far less common. The representations themselves are learned by minimizing a global loss function involving all entities and relations. As a result, even the embedding representation of a *single* entity or relation encodes global information from the *entire* knowledge graph. Subsequently, we describe some of the more established methods for achieving this kind of encoding.

4.2.1 TransE

TransE was one of the first KGE techniques proposed (shortly after the *Structured Embedding* method [31]) [30], and has largely survived the test of time. It continues to be widely used, and delivers extremely competitive performance. A range of alternatives (commonly referred to as Trans*) have been built using the same fundamental principles as TransE, but with richer optimizations and information sets. We present the TransE system in detail, and then briefly cover some alternatives.

First, TransE is an *energy-based* model for learning low-dimensional embeddings of entities; specifically, relationships are represented as translations in the embedding space: if (h, r, t) holds, then the embedding of the tail entity t should be close to the embedding of the head entity h plus some vector that depends on the relationship r. Put more mathematically, the algorithm attempts to generate an embedding for each h, r and t such that for triples observed in the knowledge graph, the *translation* relationship $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ should hold. Given enough triples, the hope is that the embedding is general enough to yield new information i.e. in the test phase, if we observe a relationship $\mathbf{h'} + \mathbf{r'} \approx \mathbf{t'}$ that was not observed during training, there is a non-trivial probability that (h', r', t') is a *true* triple (constituting missing information in the original KG) and can be added to the KG to complete it. A key strength of TransE is its reliance on a reduced set of parameters since it learns only *one* low-dimensional vector for *each* entity and *each* relationship. The energy-based optimization function (based on translation) is also simple and intuitive to understand.

Why would translation be expected to be so successful? One motivation is that hierarchical relationships are extremely common in KBs and translations are the natural transformations for representing them. For example, a natural representation of trees is to have the siblings be close to each other; in other words, with nodes at a given height organized on the x-axis, the parent-child relationship corresponds to a translation on the y-axis. Since a null translation vector corresponds to an equivalence relationship between entities, the model can then represent the sibling relationship as well. A secondary motivation arose from coincidental findings from the word embedding literature, where some authors found that many relations (e.g., capital-of, has-father) are represented by the model as translations in the embedding space. This suggested the existence of embedding spaces in which 1-to-1 relationships between entities of different types may potentially be represented by translations. The intent of TransE was to enforce such a structure of the embedding space.

4.2.2 TransE Extensions and Alternatives

The basic TransE model has been extended in numerous ways into a family of Trans* algorithms such as TransH, TransR, and TransD, to name a few [106, 176, 179]. The primary difference lies in the underlying assumptions about the embedding space,, which impacts the optimization function used during both training and testing. For example, to overcome limitations of TransE in dealing with 1-to-N, N-to-1, and N-to-N relations, an effective strategy is to allow an entity to have *distinct* representations when involved in *distinct* relations. To take an example, one could imagine learning a different embedding for the city 'Tokyo' in the context of a relation such as 'capital of' than in the context of the relation 'has population'. Intuitively, the first embedding would put (the entity vector representation of) Tokyo closer to other capital cities, while the second embedding may place it closer to cities with similar populations. in theory, this kind of embedding permits richer information sets to be captured, but at the cost of using more training data and observing more triples.

TransH follows this general idea [179], by introducing *relation-specific hyperplanes*. Similar to TransE, TransH models entities and relations as vectors, but the relation vector r is considered to be specific to a hyperplane (defined by its normal vector w_r). In other words, a relation is actually captured by two pieces of information (a vector, and a hyperplane normal). Given a triple (h, r, t), the entity representations of h and t are first *projected* onto the hyperplane, followed by the translation operation. The projections are assumed to be connected by r on the hyperplane with low error if (h, r, t) holds, with the scoring function being similar to that used by TransE. It is both possible and expected that, for some hyperplanes, the triple will have low error, while on other hyperplanes it won't, since h and t will not be connected through the relation underlying that hyperplane. Mathematically, the optimization is richer since it has to perform the dual task of hyperplane-specific translation (low error for true triples and high error for false triples), and discovering hyperplanes that are expressive and separable enough to accomplish such discrimination. As a consequence, TransH is slower than TransE, and (all else being equal) does not generalize as well to smaller or sparser graphs than TransE.

TransR is a similar variant [106], the difference being that it introduces relation-specific *spaces*, rather than (the more constrained) *hyperplanes*. In TransR, entities are represented as vectors in an entity space $\mathscr{R}^d$, and each relation is modeled as a translation vector in k-dimensional space $\mathscr{R}^k$ that doesn't necessarily have to be a hyperplane. More details on these operations are provided in the original paper. Herein, we note that, although powerful in modeling complex relations, TransR introduces a projection matrix for *each* relation, hence requiring $O(dk)$ parameters per relation. It ends up losing both the simplicity and efficiency of TransE and TransH, both of which require only $O(d)$ parameters per relation, d being the embedding dimensionality. More complicated versions of the same approach were also later proposed. We do not cover these here, but provide a list of some models (and their embedding spaces) published at the time of writing, in Fig. 4.3.

Method	Ent. embedding	Rel. embedding
TransE	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{r} \in \mathbb{R}^d$
TransH	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{r}, \mathbf{w}_r \in \mathbb{R}^d$
TransR	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{r} \in \mathbb{R}^k, \mathbf{M}_r \in \mathbb{R}^{k \times d}$
TransD	$\mathbf{h}, \mathbf{w}_h \in \mathbb{R}^d$ $\mathbf{t}, \mathbf{w}_t \in \mathbb{R}^d$	$\mathbf{r}, \mathbf{w}_r \in \mathbb{R}^k$
TranSparse	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{r} \in \mathbb{R}^k, \mathbf{M}_r(\theta_r) \in \mathbb{R}^{k \times d}$ $\mathbf{M}_r^1(\theta_r^1), \mathbf{M}_r^2(\theta_r^2) \in \mathbb{R}^{k \times d}$
TransM	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{r} \in \mathbb{R}^d$
ManifoldE	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{r} \in \mathbb{R}^d$
TransF	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{r} \in \mathbb{R}^d$
TransA	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{r} \in \mathbb{R}^d, \mathbf{M}_r \in \mathbb{R}^{d \times d}$
KG2E	$\mathbf{h} \sim \mathcal{N}(\boldsymbol{\mu}_h, \Sigma_h)$ $\mathbf{t} \sim \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)$ $\boldsymbol{\mu}_h, \boldsymbol{\mu}_t \in \mathbb{R}^d$ $\Sigma_h, \Sigma_t \in \mathbb{R}^{d \times d}$	$\mathbf{r} \sim \mathcal{N}(\boldsymbol{\mu}_r, \Sigma_r)$ $\boldsymbol{\mu}_r \in \mathbb{R}^d, \Sigma_r \in \mathbb{R}^{d \times d}$
TransG	$\mathbf{h} \sim \mathcal{N}(\boldsymbol{\mu}_h, \sigma_h^2 \mathbf{I})$ $\mathbf{t} \sim \mathcal{N}(\boldsymbol{\mu}_t, \sigma_t^2 \mathbf{I})$ $\boldsymbol{\mu}_h, \boldsymbol{\mu}_t \in \mathbb{R}^d$	$\boldsymbol{\mu}_r^i \sim \mathcal{N}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_h, (\sigma_h^2 + \sigma_t^2)\mathbf{I})$ $\mathbf{r} = \sum_i \pi_r^i \boldsymbol{\mu}_r^i \in \mathbb{R}^d$
UM	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	—
SE	$\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$	$\mathbf{M}_r^1, \mathbf{M}_r^2 \in \mathbb{R}^{d \times d}$

Fig. 4.3 Embedding spaces of TransE and its alternatives. $\mathcal{N}$ is the normal distribution with the usual mean μ and standard deviation σ parameters. d is the embedding dimensionality (set by the user, and generally in the range of tens to hundreds), n and m are the numbers of entities and relations respectively in the KG to be embedded. Other symbols are algorithm- or system-specific, although some (such as k) can be specified by the user. For example, in TranSparse θ is the average sparseness degree of projection matrices. For more formal definitions of parameters, we refer the reader to the individual papers or to a recent condensed survey

Method	Space complexity	Time complexity
TransE	$\mathcal{O}(nd + md)$	$\mathcal{O}(d)$
TransH	$\mathcal{O}(nd + md)$	$\mathcal{O}(d)$
TransR	$\mathcal{O}(nd + mdk)$	$\mathcal{O}(dk)$
TransD	$\mathcal{O}(nd + mk)$	$\mathcal{O}(\max(d, k))$
TranSparse	$\mathcal{O}(nd + (1-\theta)mdk)$	$\mathcal{O}(dk)$
TransM	$\mathcal{O}(nd + md)$	$\mathcal{O}(d)$
ManifoldE	$\mathcal{O}(nd + md)$	$\mathcal{O}(d)$
TransF	$\mathcal{O}(nd + md)$	$\mathcal{O}(d)$
TransA	$\mathcal{O}(nd + md^2)$	$\mathcal{O}(d^2)$
KG2E	$\mathcal{O}(nd + md)$	$\mathcal{O}(d)$
TransG	$\mathcal{O}(nd + mdc)$	$\mathcal{O}(dc)$
UM	$\mathcal{O}(nd)$	$\mathcal{O}(d)$
SE	$\mathcal{O}(nd + md^2)$	$\mathcal{O}(d^2)$

Fig. 4.4 Time and space complexity of selected KGE models. For details on notation, see caption of Fig. 4.3

4.2.3 Limitations and Alternatives

Recently, it was shown that KGEs can suffer from problems of generalization, reflected in poor performance, when the KG is either too sparse or noisy (or both) [144]. In such situations, alternate, more established methods such as Probabilistic Soft Logic (PSL) were found to work better [92]. Another issue is the time taken to train an embedding, and the tuning of hyperparameters. While an efficient implementation of TransE (and some of its extensions) exists at the time of writing, the original implementation was time consuming requiring on the order of hours to train medium-sized knowledge graphs. This makes trial-and-error-style training and tuning, problematic. Over time, the models have become steadily more complicated, in fact (Fig. 4.4).

4.2.4 Research Frontiers and Recent Work

Many of the models that have been proposed for knowledge graph embeddings are based on using assertions in a given KG as observations in the training data. However, this imposes a degree of locality on the embedding model, since there are other potential information sets that can be considered in the embedding optimization. Some possibilities (as alternate information sets) that have been recently proposed in addition to, or even instead of, assertions in the KG are covered below. These information sets can be used to augment and improve KGE training [176].

4.2.4.1 Ontological Information

Knowledge graphs do not just contain entities, since (as we described in detail in both Chaps. 1 and 2) many of the nodes are *typed* according to some ontology, whether implicit, explicit or shallow. Some ontologies, such as YAGO [167], can be extremely detailed containing full hierarchies of classes and sub-classes. For example, Sharon Stone is a person, but also an actor. Thus, all else the same, we would give higher weight to a triple that declares Sharon Stone to have starred in a movie, than if we had not known that Sharon Stone is an actress. Many similar examples can be formulated along these lines.

Modeling this intuition in a computational way is less straightforward. A possible avenue is to *not* treat the 'is-a' relation as special (recall that is-a was one of the few, and often the only, relations that serves as a 'glue' between the KG and the ontology) but to declare all is-a triples, and possibly other ontological triples, in the same vein as other KG triples e.g., $< SharonStone, is-a, Actor >$, $< SharonStone, is{-}a, Person >$. In this formulation, we are essentially *augmenting* the original training set (the assertions in the KG) with additional triples (the ontology, and the glue between KG and ontology). The effectiveness of this strategy is not completely clear at the time of writing, especially with respect to the relative sizes of ontologies and KGs. An advantage of the method is that it is simple. A disadvantage is that it may be simplistic e.g., is Sharon Stone is an actress, an actress is an entertainer, and an entertainer is a person, the embedding is not really capturing the fact that Sharon Stone is a person. Intuitively, the *special semantics* of is-a (and other) relations is not being taken into account by the embedding.

This has motivated more complex approaches that take into account the special nature of is-a triples. For example, as proposed in [70] using a method called *semantically smooth embedding* (SSE), one could *explicitly* design the optimization to require entities of the *same type* to stay close to each other in the embedding space e.g., Sharon Stone would be closer to Sylvester Stallone than to Roger Penrose, since Stallone is also an actor, while Penrose is a scientist. Technically, SSE employs two manifold learning algorithms, i.e., Laplacian eigenmaps and locally linear embedding to model such a 'smoothness assumption'. More specific details can be found in the original paper [70].

A second approach, proposed in [186], is *type-embodied knowledge representation learning* (TKRL), which can handle hierarchical entity categories and multiple category labels. TKRL is a translational distance model with type-specific entity *projections*. Given an assertion (h, r, t), TKRL first projects h and t with type-specific projection matrices, and then models r as a translation between the two projected entities. Because of the matrices, TKRL can have a high space complexity, and would likely not generalize well unless it had access to enough data. When it does, however, it has been found to have better performance in tasks and applications such as link prediction and triples classification. Whether this tradeoff makes sense for an application designer depends on the application and the size (and trustworthiness) of the KG to be embedded.

Currently, there continues to be ongoing interest in utilizing ontological information sets in knowledge graph embeddings, whether directly or indirectly. Most likely, there is still much work to be done in this area. What is clear, however, is that this information should not be ignored by the embedding model and can serve a useful purpose, whether in terms of boosting performance or (more abstractly) modeling human intuition more closely.

4.2.4.2 Text

Researchers have also explored incorporating textual descriptions of entities into the KGE model. This is motivated by the observation that, in many KGs, concise descriptions for entities are available, containing important semantic information about the entities. Even when this is not the case, one can always find and crawl sources such as news releases and Wikipedia articles to enrich entities with context.

Embedding KGs with text information dates back to the NTN model [165], which was proposed fairly early on. In NTN, text information is used in a fairly naive way since it is simply used to *initialize* entity embeddings. Namely, NTN first learns word vectors from a news corpus, and then initializes the representation of each entity by *averaging* the vectors of words contained in the entity's label. By way of example, the representation for 'Sharon Stone' would be initialized by averaging the word vectors for 'Sharon' and 'Stone'. This example also shows why the utility of text information is naive in this model, since Sharon and Stone individually show up in other contexts as well. This is also true for locations ('New York' vs. 'New Orleans', or 'Los Angeles' vs. 'Los Alomos'), and for other entity types as well. Later, another similar method was proposed, in which entities were represented as average word vectors of their descriptions rather than just their names. More generally, these kinds of methods are problematic because they do not take into account *joint contexts* of assertions and text but instead model textual information distinctly from assertions, and in the process, fail to leverage the potentially rich interactions between such information sets.

To the best of our knowledge, the first such joint model was proposed in [178]. The main idea was to *align* the KG with the text corpus, and then train both KG embedding and word embedding jointly, with the hope that both embeddings will be informed and improved by each other since the embeddings for entities, relations and words are all represented in the *same* vector space. Operations like inner product and similarity between these different elements become meaningful and insightful. The joint model has three 'sub-models': knowledge, text, and alignment. The knowledge sub-model simply embeds entities and relations in the KG and is actually a variant of TransE, with a special loss function for measuring fitness of the embeddings to KG facts. The text sub-model embeds words, and is a variant of the famous skip-gram word embedding model. Similar to the knowledge sub-model, it comes with a loss function that measures the fitness of the sub-model embedding to co-occurring word pairs. Finally, the alignment sub-model is designed to ensure that the embeddings of the two other sub-models lie in the *same space*. Different

such alignment mechanisms are introduced in their work and others that followed it, including by entity names, Wikipedia anchors, entity descriptions etc. with more such mechanisms continuing to be proposed in current research. Similar to the other sub-models, the alignment sub-model's loss function is defined to measure the quality ('fitness') of alignment. We highlight that one of the major contributions in using such joint models is the prediction of *out-of-KG* entities, i.e., phrases appearing in web text but not included in the KG yet.

Yet another approach along these lines is the *description-embodied knowledge representation learning* (DKRL) [185], which seeks to extend TransE to incorporate entity descriptions. DKRL associates each entity with two vector representations, one of which is structure-based and the other of which is description-based. The former captures structural information (just like the original TransE), while the latter captures textual information in the entity description. The description-based representation is constructed using word embeddings. Entity, relation, and word embeddings can all be learned simultaneously by minimizing a ranking loss function when training. Experimental results demonstrated the superiority of DKRL over TransE, particularly in the zero-shot scenario with out-of-KG entities.

Generally, incorporating text into the optimization tends to lead to empirical improvements. However, we are not aware of a full-scale empirical study that attempts to measure the extent of these improvements, and to assess the sensitivity of such improvements with respect to important parameters such as the size and quality of a KG, the relevance of the text corpus, and the noise in the text corpus. Beyond normal performance benefits, a primary benefit of the joint model, as we highlighted earlier, was its ability to gracefully handle entities that may not have been observed in the actual KG.

4.2.4.3 Other Extrinsic Information Sets

Incorporating text and ontological information into KGEs continue to be important directions of research, especially for improving KGEs using more context. However, these information sets are by no means the only ones. Below, we briefly cover some others.

Rules Ontologies are not just sets of inter-related concepts and properties. They can also contain constraints and rules to further express the domain. Can rules, as understood in this sense, be used to further influence KGEs in a positive direction? Wang et al. [177] proposed an approach utilizing rules to refine embedding models during KG completion. In their work, KG completion is formulated as an ILP (integer linear programming) problem. Specifically, the objective function is generated from embedding models, and the ILP constraints from pre-specified rules. Assertions inferred in this way will be the most preferred by the embedding models but would also comply with all the rules. A similar work that combines rules and embedding models via graphical models such as Markov logic networks was later introduced in [181]. However, in both the papers above, rules are modeled separately

from embedding models, serving as *post-processing* modules. They do not directly influence embeddings, and hence cannot be used to obtain 'better' embeddings.

A later approach that tried to influence embeddings in a joint model that leveraged rules directly in the embedding optimization was KALE [71], wherein a model was proposed that simultaneously embeds assertions and rules. In this framework, an assertion was modeled as a ground atom, with a well-defined truth value. Also, logical rules are first instantiated into ground rules, with ground rules then interpreted as complex formulae constructed by combining ground atoms with logical connectives (e.g., $\vee$), and modeled by t-norm fuzzy logics [93]. The truth value of a ground rule is a composition of the truth values of the constituent ground atoms, via specific t-norm based logical connectives.

The values of these connectives lie within the range of [0,1], indicating to what degree the ground rule is satisfied. In this way, KALE represents facts and rules in a unified framework, as atomic and complex formulae respectively. After unifying assertions and rules in this way, KALE minimizes a global loss involving both to learn both entity and relation embeddings. These learned embeddings are compatible not only with assertions in the training corpus but also with rules, which is hoped to lead to better performance of embeddings in downstream applications.

KALE has inspired other variants. For example, in [157], the overall approach is similar to KALE, but vector embeddings are introduced for entity pairs rather than individual entities, making it particularly useful for *relation extraction*. This is an example of an embedding which is (a priori) optimized keeping a target application in mind. However, since entities do not have their own embeddings, relations between unpaired entities cannot be effectively discovered.

Both KALE and the variant described above have the limitation that they have to instantiate universally quantified rules into ground rules before learning the embedding models. This grounding procedure is known to be time- and space-inefficient, especially when there are many entities in the KG or the rules are too complex. Some recent work has recognized this drawback, and proposed solutions to address it.

Generally speaking, the ongoing research shows that rules will continue to find more applications and uses in KGEs. The good performance of rule-supplemented KGEs, and the researchers investing in this approach, both show that there is an interesting synergy to be had between methods that were traditionally seen as disparate (statistical and logical), though by no means incompatible. Future research will show till what extent this synergy can be exploited, both in KGEs and other similar areas.

Temporal information In [82], the critical observation was made that KG assertions may often be time-sensitive, e.g., (Sharon Stone, ReceivedAward, Golden Globe) happened in 1995. Based on this observation, a *time-aware embedding* model was proposed, the idea being to impose temporal order constraints on time-sensitive relation pairs, e.g., StarredIn and ReceivedAward. Given such a pair (r_i, r_j), the prior relation is supposed to lie close to the subsequent relation after a temporal transition, i.e., $Mr_i \approx r_j$ where M is a transition matrix capturing the

temporal order information between relations. After imposing such constraints, the authors in [82] are able to learn temporally consistent relation embeddings. In other work, [58] tried to model the temporal *evolution* of KGs. That is, changes in a KG always arrive as events, represented by labeled quadruples such as $(h, r, t, s; True)$ or $(h, r, t, s; False)$, indicating that the assertion (h, r, t) appears or vanishes at time s, respectively. Each quadruple is then modeled as a four-way interaction among h, r, t, and s, where s is the vector representation of the time stamp. This model was shown to perform well in dynamic domains such as sensors or medicine. Overall, research has continued to intensify in this domain, and the link prediction problem that we study later has been extended to temporal link prediction i.e. the problem of predicting not just a link, but when it becomes stale (or active).

Paths and structures Relation paths may be understood as *multi-hop relationships* between entities. A relation path is typically defined as a sequence of relations $r_1 \rightarrow \ldots \rightarrow r_l$ through which two entities can be connected on the graph. For example, StarredIn $\rightarrow$ ShotIn is a path linking Sharon Stone to Nevada, via an intermediate movie node such as Casino. Relation paths contain semantic cues not otherwise found in the node itself and can useful for KG completion.

More generally, it is possible to learn such 'graph-aware embedding models' by leveraging different types of graph structures. In [62], such a model was proposed, leveraging three types of graph structures: neighbor context (equivalent to triples observed in a KG), path context (similar to relation paths just described) and edge context (defined as the relations linking to and from that entity). The last is primarily ontological e.g., the edge context of Sharon Stone might include relations such as StarredIn, LivesIn, ReceivedAward etc. Intuitively, all of these relations indicate collectively that Sharon Stone is a person, and more specifically, an entertainer. Experimental results have demonstrated the effectiveness of incorporating these graph structures in an embedding model. In other work, [83] suggested that the plausibility of a triple $tr = (h, r, t)$ could be estimated from its immediate context, defined as the set of triples sharing the same head as tr, the set of triples sharing the same tail, the set of triples with h as tail, the set of triples with t as head, and triples with arbitrary relations but where the two entities are h and t. By using such contexts, a system was found to perform better at the link prediction task (described subsequently) on multi-relational data, such as KGs.

Other Entity Attributes When introducing KGs in Chap. 1, we argued that relations in KGs can indicate both relationships *between* entities (e.g., StarredIn indicates a relationship between Movie and Actor entities) or be used to define entity attributes (e.g., the gender or birthdate of a person). Unfortunately, most KG embedding techniques such as TransE do not explicitly distinguish between these semantics. In [131], this distinction was made. Namely, entity-entity relations were encoded in a tensor, while entity-attribute relations in a separate entity-attribute matrix. The matrix and tensor are jointly factorized to learn representations simultaneously for entities, and both types of relations. Similar ideas have since been studied by other authors [105].

4.2.5 Applications of KGEs

Since a knowledge graph embedding is essentially just a mapping from nodes and edges to real-valued vectors, how can we tell when one embedding is better than another? An uncontroversial approach is an *ablation-style evaluation* [182] where, in the context of a given application, we evaluate an embedding against another embedding keeping all else constant (including datasets and metrics). Although such an evaluation is not without flaws, the biases (particularly, dataset bias [170]) are reduced if the benchmarks are large and real-world, and the applications have relevance. Below, we describe several viable applications. Note that one such application, Entity Resolution (ER) [66], has already been covered in detail in the previous chapter. There are two contexts in which we can use KGEs for ER. First, recall that there was a feature generation step whereby we attempted to extract a feature vector for each pair of entities that were consumed in the similarity step of ER. Feature engineering is a labor intensive process and there is always the possibility of missing something. By concatenating the embedding vectors of the two nodes, we can get an alternate feature representation that could potentially be used for better ER performance. Early results have been promising, though the hypothetical utility of embeddings over engineered features is still in the preliminary stages of ER research. A second possibility for utilizing embeddings for ER is to frame ER as a special, supervised case of a relation or link prediction problem (described below).

ER is a good example of an *in-KG* application, which is conducted *within the scope* of the KG where entity and relation embeddings are learned. Three other examples of in-KG applications are link prediction, triple classification and entity classification [176], all of which have been well-studied in the literature. It is not atypical to assume that all of these applications can be cast as special cases of knowledge graph *refinement*, with different definitions of refinement.

Link Prediction *Link prediction* is the problem of predicting whether a given entity has a specific relation with another 'hypothetical' entity, i.e., predicting h given (r, t) or conversely, t given (h, r), with the former task denoted as head entity prediction $(?, r, t)$, and the latter as tail entity prediction $(h, r, ?)$. Link prediction is a general problem that can be 'fed into' multiple out-of-KG applications e.g., question answering or even conversational AI [65, 168]. For example, (?, StarringIn, Terminator) is to predict the stars of the film Terminator, while (Sharon Stone, StarringIn, ?) amounts to predicting films that Sharon Stone has starred in. This example also shows that prediction can be a many-many problem i.e. there are multiple correct predictions for both cases. Link prediction is a quintessential KG completion task, i.e., adding missing knowledge to the graph, and has been tested extensively in previous literature. An alternate name for the problem (among others) is entity ranking. A similar idea can also be used to predict relations between two given entities, i.e., $(h, ?, t)$, which is usually referred to as relation prediction. In the social network community, link prediction has a much more specific meaning than in the KGE community; it is usually the problem of predicting future links (e.g., friendship) that might be formed between actors in the social network [110].

With entity and relation representations learned during training, link prediction can be done using ranking, similar to procedures developed over decades in the Information Retrieval community. Take the prediction task $(?, r, t)$ as an example, a ranking system can 'predict' the head entity by taking every entity h' in the KG as a *candidate* answer and calculating a score for each (h', r, t), using a scoring function. In descending order of scores, this yields a ranked list of candidate answers. If the embedding is 'good', the hope is that correct predictions will be ranked nearer to the top of the list than incorrect predictions.

Similar to IR, the common way to evaluate such rankings is to use metrics such as *mean rank* (the average of predicted ranks), *mean reciprocal rank* (the average of reciprocal ranks), *Hits@n* (the proportion of ranks no larger than n), and *AUC-PR* (the area under the precision-recall curve). Different metrics have different tradeoffs. AUC-PR takes a balanced view of precision and recall, for example, while Hitsn is oriented more towards recall than precision. For example, considering the Hits10 metric, and assuming there is only one correct prediction, a ranking where the correct entity is at rank 1 will have the same Hits10 (=1.0) as one where the correct entity is at rank 10. Similarly, if the correct entity is not in the top 10, but is at rank 11 vs. rank 100, both would receive a Hits10 of 0.0. Note that, for individual 'queries', Hits10 can only be 0 or 1, but when averaged over many such queries, ranges from 0 to 1 and can be used to assess the performance of an embedding on a test set, on average.

Entity Classification *Entity classification* is the problem of classing entities under different *semantic* categories, e.g., Sharon Stone is an Actor, Terminator is a Movie and so on [129]. Generally, the relation that is considered for classification purposes is the *is-a* relation. If the is-a relation has already been included in the embedding process (so an embedding for the is-a relation exists after training), entity classification can simply be treated as a special case of link prediction, and the same evaluation procedures can be applied for it. This similarity between link prediction, and both entity classification and entity resolution, highlights what was noted earlier, namely, that they can all be thought of as very specialized cases of the broader knowledge graph refinement (or completion) problem.

Triple Classification *Triple classification* can be thought of as a binary classification problem [129]: given an arbitrary triple (h, r, t), is the triple *true*? A trivial case is when the triple belongs in the training set, in which case it is clearly true. If this is not the case, then it is not necessary that the triple is untrue, since the training data was incomplete to begin with. One non-trivial issue with framing triple classification as binary classification is the consistent combination of the individual head, tail and relation embeddings in a way that can be used to predict the probability of truth. In systems like the Trans* KGEs (but also others), a density function is used to make such a prediction. The correct metric to use is accuracy, if the test data is balanced. In most benchmarks that have been released so far for triple classification, this has been the case. If the evaluation data is skewed, computing precision, recall or ROC curves may be more appropriate.

We conclude by noting that there are also *out-of-KG applications*, which are generally less controlled since they are designed to scale to broader domains. Such applications include relation extraction, question answering and recommender systems. We do not explore out-of-KG applications in this book, but focus on in-KG applications. More details on out-of-KG applications can be found in an overview of KGEs in [176].

4.3 Summary

Knowledge Graph Embeddings (KGEs) are a powerful set of techniques for representing entities, relations and even descriptions in a KG in a continuous real-valued vector space. Although some of the reasoning capabilities permitted by symbolic representations are lost in the process, the real-valued representations are much less brittle than discrete symbols, and hence, more robust to noise and missing information. Furthermore, recent efforts in the field have tried, with varying success, to reconcile the benefits of continuous and discrete KG representations. Research in this area is still ongoing, but it has become clear that KGEs are vital for the broader problem of knowledge graph completion (or identification). Other applications of KGEs include link prediction and entity classification.