A Review of Relational Machine Learning for Knowledge Graphs

From Multi-Relational Link Prediction to Automated Knowledge Graph Construction

Maximilian Nickel, Kevin Murphy, Volker Tresp, Evgeniy Gabrilovich

Abstract—Relational machine learning studies methods for the statistical analysis of relational, or graph-structured, data. In this paper, we provide a review of how such statistical models can be “trained” on large knowledge graphs, and then used to predict new facts about the world (which is equivalent to predicting new edges in the graph). In particular, we discuss two different kinds of statistical relational models, both of which can scale to massive datasets. The first is based on tensor factorization methods and related latent variable models. The second is based on mining observable patterns in the graph. We also show how to combine these latent and observable models to get improved modeling power at decreased computational cost. Finally, we discuss how such statistical models of graphs can be combined with text-based information extraction methods for automatically constructing knowledge graphs from the Web. In particular, we discuss Google’s Knowledge Vault project.

Index Terms—Statistical Relational Learning, Knowledge Graphs, Knowledge Extraction, Latent Feature Models, Graph-based Models

I. INTRODUCTION

I am convinced that the crux of the problem of learning is recognizing relationships and being able to use them.

Christopher Strachey in a letter to Alan Turing, 1954

MAChINE learning typically works with a data matrix, where each row represents an object characterized by a feature vector of attributes (which might be numeric or categorical), and where the main tasks are to learn a mapping from this feature vector to an output prediction of some form, or to perform unsupervised learning like clustering or factor analysis. In Statistical Relational Learning (SRL), the representation of an object can contain its relationships to other objects. Thus the data is in the form of a graph, consisting of nodes (entities) and labelled edges (relationships between entities). The main goals of SRL include prediction of missing edges, prediction of properties of the nodes, and clustering the nodes based on their connectivity patterns. These tasks arise in many settings such as analysis of social networks and biological pathways. For further information on SRL see [1, 2, 3].

In this article, we review a variety of techniques from the SRL community and explain how they can be applied to large-scale knowledge graphs (KGs), i.e., graph structured knowledge bases (KBs) which store factual information in form of relationships between entities. Recently, a large number of knowledge graphs have been created, including YAGO [4], DBpedia [5], NELL [6], Freebase [7], and the Google Knowledge Graph [8]. As we discuss in Section II, these graphs contain millions of nodes and billions of edges. This causes us to focus on scalable SRL techniques, which take time that is linear in the size of the graph.

In addition to typical applications of statistical relational learning, we will also discuss how SRL can aid information extraction methods to “grow” a KG automatically. In particular, we will show how SRL can be used to train a “prior” model based on an existing KG, and then combine its predictions with “noisy” facts that are automatically extracted from the web using machine reading methods (see e.g., [9][10]). This is the approach adopted in Google’s Knowledge Vault project, as we explain in Section VIII.

The remainder of this paper is structured as follows. In Section II we introduce knowledge graphs and some of their properties. Section III discusses SRL and how it can be applied to knowledge graphs. There are two main classes of SRL techniques: those that capture the correlation between the nodes/edges using latent variables, and those that capture the correlation directly using statistical models based on the observable properties of the graph. We discuss these two families in Section IV and Section V, respectively. Section VI describes approaches for combining these two approaches, to get the best of both worlds. In Section VII we discuss relational learning using Markov Random Fields. In Section VIII we describe how SRL can be used in automated knowledge base construction projects. In Section IX we discuss extensions of the presented methods, and Section X presents our conclusions.

II. KNOWLEDGE GRAPHS

In this section, we discuss knowledge graphs: how they are represented, how they are created, and how they are used.

A. Knowledge representation

Relational knowledge representation as used in KGs has a long history in logic and artificial intelligence [11]. More recently, it has been used in the Semantic Web to represent information in machine-readable form, in order to enable intelligent agents operating on a “web of data” [12]. While the original vision of the Semantic Web remains to be fully realized, parts
of it have been achieved. In particular, the concept of linked data \cite{W3C,Decker2009} has gained traction, as it facilitates publishing and interlinking data on the Web in relational form using the W3C Resource Description Framework (RDF) \cite{Soininen15,Decker14}.

In this article, we will loosely follow the RDF standard and represent facts in the form of binary relationships, in particular (subject, predicate, object) (SPO) triples, where subject and object are entities and predicate is the type of a relation. (We discuss how to represent higher-order relations in Section \ref{sec:triple}. The existence of a particular SPO triple indicates an existing fact, i.e., that the respective entities are in a relationship of the respective type. For instance, the information

Leonard Nimoy was an actor who played the character Spock in the science-fiction movie Star Trek can be expressed via the following set of SPO triples:

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>(LeonardNimoy,)</td>
<td>profession</td>
<td>Actor</td>
</tr>
<tr>
<td>(LeonardNimoy,)</td>
<td>starredIn</td>
<td>StarTrek</td>
</tr>
<tr>
<td>(LeonardNimoy,)</td>
<td>played</td>
<td>Spock</td>
</tr>
<tr>
<td>(Spock,)</td>
<td>characterIn</td>
<td>StarTrek</td>
</tr>
<tr>
<td>(StarTrek,)</td>
<td>genre</td>
<td>ScienceFiction</td>
</tr>
</tbody>
</table>

We can combine all the SPO triples together to form a multi-graph, where nodes represent entities (all subjects and objects), and directed edges represent relationships. The direction of an edge indicates whether entities occur as subjects or objects, i.e., an edge points from the subject to the object. The different relation types are represented via different types of edges (also called edge labels). This construction is called a knowledge graph (KG), or sometimes a heterogeneous information network \cite{Kagal2012}.) See Figure \ref{fig:kg} for an example.

In addition to a collection of facts, knowledge graphs often provide type hierarchies (Leonard Nimoy is an actor, which is a person, which is a living thing) and type constraints (e.g., a person can only marry another person, not a thing).

B. Open vs closed world assumption

While existing triples always encode true relationships (facts), there are different paradigms for the interpretation of non-existing triples:

- Under the closed world assumption (CWA), non-existing triples indicate false relationships. For example, the fact that in Figure \ref{fig:kg} there is no starredIn edge from Leonard Nimoy to Star Wars is interpreted as saying that Nimoy definitely did not star in this movie.
- Under the open world assumption (OWA), a non-existing triple is interpreted as unknown, i.e., the associated relationship can be either true or false. Continuing with the above example, the missing edge is not interpreted as saying Nimoy did not star in Star Wars. This more cautious approach is justified, since KGs are known to be very incomplete. For example, sometimes just the main actors in a movie are listed, not the complete cast. Also, even the place of birth attribute, which you might think would be typically known, is missing for 71% of all people included in Freebase \cite{Wang2018}.

RDF and the Semantic Web make the open-world assumption. In Section \ref{sec:triple} we also discuss the local closed world assumption (LCWA), which is often used for training relational models.

C. Knowledge base construction

Completeness, accuracy, and data quality are important parameters that determine the usefulness of knowledge bases and are influenced by the way KBs (with KG’s being a special form) are constructed. We can classify KB construction methods into 4 main groups:

- In curated approaches, triples are created manually by a closed group of experts.
- In collaborative approaches, triples are created manually by an open group of volunteers.
- In automated semi-structured approaches, triples are extracted automatically from semi-structured text (e.g., infoboxes in Wikipedia) via hand-crafted rules, learned rules, or regular expressions.
- In automated unstructured approaches, triples are extracted automatically from unstructured text via machine learning and natural language processing (NLP) techniques (see e.g., \cite{Gardiner2012} for a review).

Construction of curated knowledge bases typically leads to highly accurate results, but this technique does not scale well due to its dependence on human experts. Collaborative knowledge base construction, which was used to build Wikipedia and Freebase, scales better but still has some limitations. For instance, the place of birth attribute is missing for 71% of all people included in Freebase, even though this is a mandatory property of the schema \cite{Wang2018}. Also, a recent study \cite{Wang2018} found that the growth of Wikipedia has been slowing down. Consequently,

---

![Fig. 1. Small example knowledge graph. Nodes represent entities, edge labels represent types of relations, edges represent existing relationships.](image-url)

TABLE I

<table>
<thead>
<tr>
<th>Creation Method</th>
<th>Schema-Free Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collabo</td>
<td>ReVerb \cite{Wang2012}, OLLIE \cite{Wang2012}, PRISMATIC \cite{Wang2012}</td>
</tr>
<tr>
<td>Auto. Unstructured</td>
<td></td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th>Creation Method</th>
<th>Schema-Based Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curated</td>
<td>Cyc/OpenCyc \cite{Wang2012}, WordNet \cite{Wang2012}, UMLS \cite{Wang2012}</td>
</tr>
<tr>
<td>Collaborative</td>
<td>WikiData \cite{Wang2012}, Freebase \cite{Wang2012}</td>
</tr>
<tr>
<td>Auto. Semi-Structured</td>
<td>YAGO \cite{Wang2012}, DBpedia \cite{Wang2012}, Freebase \cite{Wang2012}</td>
</tr>
<tr>
<td>Auto. Unstructured</td>
<td>Knowledge Vault \cite{Wang2012}, NELL \cite{Wang2012}, PATTY \cite{Wang2012}, DeepDive/Elementary \cite{Wang2012}, PROSPERA \cite{Wang2012}</td>
</tr>
</tbody>
</table>

---

TABLE III

<table>
<thead>
<tr>
<th>Table I. Knowledge Base Construction Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation Method</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>Curated</td>
</tr>
<tr>
<td>Collaborative</td>
</tr>
<tr>
<td>Auto. Semi-Structured</td>
</tr>
<tr>
<td>Auto. Unstructured</td>
</tr>
</tbody>
</table>
TABLE II
SIZE OF SOME SCHEMA-BASED KNOWLEDGE BASES

<table>
<thead>
<tr>
<th>Knowledge Graph</th>
<th>Entities</th>
<th>Relation Types</th>
<th>Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freebase</td>
<td>40 M</td>
<td>35,000</td>
<td>637 M</td>
</tr>
<tr>
<td>Wikidata</td>
<td>13 M</td>
<td>1,643</td>
<td>50 M</td>
</tr>
<tr>
<td>DBpedia(^1)</td>
<td>4.6 M</td>
<td>1,367</td>
<td>68 M</td>
</tr>
<tr>
<td>YAGO2</td>
<td>10 M</td>
<td>72</td>
<td>120 M</td>
</tr>
<tr>
<td>Google Knowledge Graph</td>
<td>570 M</td>
<td>35,000</td>
<td>18,000 M</td>
</tr>
</tbody>
</table>

automatic knowledge base construction (AKBC) methods have been attracting more attention. ABKC methods can be divided into two main approaches. The first approach exploits semi-structured data, such as Wikipedia infoboxes, which has lead to large knowledge graphs with high accuracy; example projects include YAGO \([4, 23]\) and The first approach exploits semi-structured data, such as Wikipedia infoboxes, which has lead to large knowledge graphs with high accuracy; example projects include YAGO \([4, 23]\) and DBpedia \([5]\). However, semi-structured text still covers only a small fraction of the information stored on the Web. Hence the second approach tries to “read the Web”, extracting facts from natural language in Web pages. Example projects include NELL \([6]\) and the Knowledge Vault \([24]\). In Section VIII we show how we can reduce the level of “noise” in such automatically extracted facts by combining them with knowledge extracted from existing, high-quality KGs.

KGs, and more generally KBs, can also be classified based on whether they employ a fixed or open lexicon of entities and relations. In particular, we distinguish two main types of KB:

- In schema-based approaches, entities and relations are represented via globally unique identifiers and all possible relations are predefined in a fixed vocabulary. For example, Freebase might represent the fact that Barack Obama was born in Hawaii using the triple \((/m/02mjmr, /people/person/born-in, /m/03gh4)\), where \(/m/02mjmr\) is the unique machine ID for Barack Obama.

- In schema-free approaches, entities and relations are identified using open information extraction (OpenIE) techniques \([32]\) and represented via normalized but not disambiguated strings (also referred to as surface names). For example, an OpenIE system may contain triples such as (“Obama”, “born in”, “Hawaii”), (“Barack Obama”, “place of birth”, “Honolulu”), etc. Note that it is not clear from this representation whether the first triple refers to the same person as the second triple, nor whether “born in” means the same thing as “place of birth”. This is the main disadvantage of OpenIE systems.

Table I lists current knowledge base construction projects classified by their creation method and data schema. In this paper, we will only focus on schema-based KBs. Table II shows a selection of such KBs and their sizes.

D. Uses of knowledge graphs

Knowledge graphs provide semantically structured information that is interpretable by machines — a property that is regarded as an important ingredient to build more intelligent machines \([33]\). Such knowledge graphs have a variety of commercial and scientific applications. A prime example is the integration of Google’s Knowledge Graph, which currently stores 18 billion facts about 570 million entities, into the results of Google’s search engine \([8]\). The Google KG is used to identify and disambiguate entities in text, to enrich search results with semantically structured summaries, and to provide links to related entities. (Microsoft has a similar KB, called Satori, integrated with its Bing search engine \([34]\).)

Enhancing search results with semantic information from knowledge graphs can be seen as an important step to transform text-based search engines into semantically aware question answering services. Another prominent example demonstrating the value of knowledge graphs is IBM’s question answering system Watson, which was able to beat human experts in the game of Jeopardy!. Among others, this system used YAGO, DBpedia, and Freebase as its sources of information \([35]\).

Knowledge graphs are also used in several specialized domains. For instance, Bio2RDF \([36]\), Neurocommons \([37]\), and LinkedLifeData \([38]\) are knowledge graphs that integrate multiple sources of biomedical information. These have been been used for question answering and decision support in the life sciences.

E. Main tasks in knowledge graph construction and curation

In this section, we review a number of typical KG tasks.

Link prediction is concerned with predicting the existence (or probability of correctness) of (typed) edges in the graph (i.e., triples). This is important since existing knowledge graphs are often missing many facts, and some of the edges they contain are incorrect \([39]\). It has been shown that relational models which take the relationships of entities into account can significantly outperform non-relational ML methods for this task (e.g., see \([40, 41]\)). As we describe in detail in Section VIII link prediction can support automated knowledge base construction by estimating the plausibility of triples from already known facts. For instance, suppose an information extraction method returns a fact claiming that Barack Obama was born in Kenya, and suppose (for illustration purposes) that the true place of birth of Obama was not already stored in the knowledge graph. An SRL model can use related facts about Obama (such as his profession being US President) to infer that this new fact is unlikely to be true and should not be included.

Entity resolution (also known as record linkage \([42]\), object identification \([43]\), instance matching \([44]\), and deduplication \([45]\)) is the problem of identifying which objects in relational data refer to the same underlying entities. In a relational setting, the decisions about which objects are assumed to be identical can propagate through the graph, so that matching decisions are made collectively for all objects in a domain rather than independently for each object pair (see, for example, \([46, 47, 48]\)). In schema-based automated knowledge base construction, entity resolution can be used to match the extracted surface names to entities stored in the knowledge graph.

Link-based clustering extends feature-based clustering to a relational learning setting and groups entities in relational data...
based on their similarity. However, in link-based clustering, entities are not only grouped by the similarity of their features but also by the similarity of their links. As in entity resolution, the similarity of entities can propagate through the knowledge graph, such that relational modeling can add important information for this task. In social network analysis, link-based clustering is also known as community detection [49].

III. STATISTICAL RELATIONAL LEARNING FOR KNOWLEDGE GRAPHS

Statistical Relational Learning is concerned with the creation of statistical models for relational data. In the following sections we discuss how statistical relational learning can be applied to knowledge graphs. We will assume that all the entities and (types of) relations in a knowledge graph are known. (We discuss extensions of this assumption in Section IX-C). However, triples are assumed to be incomplete and noisy; entities and relation types may contain duplicates.

Notation: Before proceeding, let us define our mathematical notation. (Variable names will be introduced later in the appropriate sections.) We denote scalars by lower case letters, such as \( a \); column vectors (of size \( N \)) by bold lower case letters, such as \( \mathbf{a} \); and tensors (of size \( N_1 \times N_2 \)) by bold upper case letters, such as \( \mathbf{A} \). We denote the \( k \)’th “frontal slice” of a tensor \( \mathbf{A} \) by \( \mathbf{A}_k \) (which is a matrix of size \( N_1 \times N_2 \)), and the \( (i,j,k) \)’th element by \( a_{ijk} \) (which is a scalar). We use \( \mathbf{a}; \mathbf{b} \) to denote the vertical stacking of vectors \( \mathbf{a} \) and \( \mathbf{b} \), i.e., \( \mathbf{a}; \mathbf{b} = \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix} \). We can convert a matrix \( \mathbf{A} \) of size \( N_1 \times N_2 \) into a vector \( \mathbf{a} \) of size \( N_1 N_2 \) by stacking all columns of \( \mathbf{A} \), denoted \( \mathbf{a} = \text{vec} (\mathbf{A}) \). The inner (scalar) product of two vectors (both of size \( N \)) is defined by \( \mathbf{a}^\top \mathbf{b} = \sum_{i=1}^{N} a_i b_i \). The tensor (Kronecker) product of two vectors (of size \( N_1 \) and \( N_2 \)) is a vector of size \( N_1 N_2 \) with entries \( \mathbf{a} \otimes \mathbf{b} = \begin{pmatrix} a_1 \mathbf{b} \\ \vdots \\ a_{N_1} \mathbf{b} \end{pmatrix} \).

Matrix multiplication is denoted by \( \mathbf{A} \mathbf{B} \) as usual. We denote the \( L_2 \) norm of a vector by \( ||\mathbf{a}||_2 = \sqrt{\sum_i a_i^2} \), and the Frobenius norm of a matrix by \( ||\mathbf{A}||_F = \sqrt{\sum_{i,j} a_{ij}^2} \). We denote the vector of all ones by \( \mathbf{1} \), and the identity matrix by \( \mathbf{I} \).

A. Probabilistic knowledge graphs

We now introduce some mathematical background so we can more formally define statistical models for knowledge graphs. Let \( \mathcal{E} = \{e_1, \ldots, e_{N_e}\} \) be the set of all entities and \( \mathcal{R} = \{r_1, \ldots, r_{N_r}\} \) be the set of all relation types in a knowledge graph. We model each possible triple \( x_{ijk} = (e_i, r_k, e_j) \) over this set of entities and relations as a binary random variable \( y_{ijk} \in \{0,1\} \) that indicates its existence. All possible triples in \( \mathcal{E} \times \mathcal{R} \times \mathcal{E} \) can be grouped naturally in a third-order tensor (three-way array) \( \mathbf{Y} \in \{0,1\}^{N_e \times N_r \times N_e} \), whose entries are set such that

\[
    y_{ijk} = \begin{cases} 
        1, & \text{if the triple } (e_i, r_k, e_j) \text{ exists} \\
        0, & \text{otherwise}.
    \end{cases}
\]

We will refer to this construction as an adjacency tensor (cf. Figure 2). Each possible realization of \( \mathbf{Y} \) can be interpreted as a possible world. To derive a model for the entire knowledge graph, we are then interested in estimating the joint distribution \( P(\mathbf{Y}) \), from a subset \( \mathcal{D} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E} \) of observed triples. In doing so, we are estimating a probability distribution over possible worlds, which allows us to predict the probability of triples based on the state of the entire knowledge graph. While \( y_{ijk} = 1 \) in adjacency tensors indicates the existence of a triple, the interpretation of \( y_{ijk} = 0 \) depends on whether the open world, closed world, or local-closed world assumption is made. For details, see Section Section III-D.

Note that the size of \( \mathbf{Y} \) can be enormous for large knowledge graphs. For instance, in the case of Freebase, which currently consists of over 40 million entities and 35,000 relations, the number of possible triples \( |\mathcal{E} \times \mathcal{R} \times \mathcal{E}| \) exceeds \( 10^{19} \) elements. Of course, type constraints reduce this number considerably.

Even amongst the syntactically valid triples, only a tiny fraction are likely to be true. For example, there are over 450,000 thousands actors and over 250,000 movies stored in Freebase. But each actor stars only in a small number of movies. Therefore, an important issue for SRL on knowledge graphs is how to deal with the large number of possible relationships while efficiently exploiting the sparsity of relationships. Ideally, a relational model for large-scale knowledge graphs should scale at most linearly with the data size, i.e., linearly in the number of entities \( N_e \), linearly in the number of relations \( N_r \), and linearly in the number of observed triples \( |\mathcal{D}| = N_d \).

B. Types of SRL models

As we discussed, the presence or absence of certain triples in relational data is correlated with (i.e., predictive of) the presence or absence of certain other triples. In other words, the random variables \( y_{ijk} \) are correlated with each other. We will discuss three main ways to model these correlations:

M1) Assume all \( y_{ijk} \) are conditionally independent given latent features associated with subject, object and relation type and additional parameters (latent feature models)

M2) Assume all \( y_{ijk} \) are conditionally independent given observed graph features and additional parameters (graph feature models)

M3) Assume all \( y_{ijk} \) have local interactions (Markov Random Fields)

In what follows we will mainly focus on M1 and M2 and their combination; M3 will be the topic of Section VII.
The model classes M1 and M2 predict the existence of a triple \( x_{ijk} \) via a score function \( f(x_{ijk}; \Theta) \) which represents the model’s confidence that a triple exists given the parameters \( \Theta \). The conditional independence assumptions of M1 and M2 allow the probability model to be written as follows:

\[
P(Y|D, \Theta) = \prod_{i=1}^{N_x} \prod_{j=1}^{N_y} \prod_{k=1}^{N_r} \text{Ber}(y_{ijk} | \sigma(f(x_{ijk}; \Theta)))
\]  

(1)

where \( \sigma(u) = 1/(1 + e^{-u}) \) is the sigmoid (logistic) function, and

\[
\text{Ber}(y|p) = \begin{cases} 
  p & \text{if } y = 1 \\
  1 - p & \text{if } y = 0
\end{cases}
\]  

(2)

is the Bernoulli distribution.

We will refer to models of the form Equation (1) as probabilistic models. In addition to probabilistic models, we will also discuss models which optimize \( f(\cdot) \) under other criteria, for instance models which maximize the margin between existing and non-existing triples. We will refer to such models as score-based models. If desired, we can derive probabilities for score-based models via Platt scaling [50].

There are many different methods for defining \( f(\cdot) \), which we discuss in Sections [IV] and [V]. Before we proceed to these modeling questions, we discuss how to estimate the model parameters in general terms.

C. Penalized maximum likelihood training

Let us assume we have a set of \( N_d \) observed triples and let the \( n \)-th triple be denoted by \( x^n \). Each observed triple is either true (denoted \( y^n = 1 \)) or false (denoted \( y^n = 0 \)). Let this labeled dataset be denoted by \( D = \{(x^n, y^n) | n = 1, \ldots, N_d\} \). Given this, a natural way to estimate the parameters \( \Theta \) is to compute the maximum a posteriori (MAP) estimate:

\[
\max_{\Theta} \sum_{n=1}^{N_d} \log \text{Ber}(y^n | \sigma(f(x^n; \Theta))) + \log p(\Theta | \lambda)
\]  

(3)

where \( \lambda \) controls the strength of the prior. (If the prior is uniform, this is equivalent to maximum likelihood training.) We can equivalently state this as a regularized loss minimization problem:

\[
\min_{\Theta} \sum_{n=1}^{N} \mathcal{L}(\sigma(f(x^n; \Theta)), y^n) + \lambda \text{reg}(\Theta)
\]  

(4)

where \( \mathcal{L}(p, y) = -\log \text{Ber}(y|p) \) is the log loss function. Another possible loss function is the squared loss, \( \mathcal{L}(p, y) = (p - y)^2 \). We discuss some other loss functions below.

D. Where do the negative examples come from?

One important question is where the labels \( y^n \) come from. The problem is that most knowledge graphs only contain positive training examples, since, usually, they do not encode false facts. Hence \( y^n = 1 \) for all \( (x^n, y^n) \in D \). To emphasize this, we shall use the notation \( D^+ \) to represent the observed positive (true) triples: \( D^+ = \{x^n \in D | y^n = 1\} \). Training on all-positive data is tricky, because the model might easily overgeneralize.

One way around this is to assume that all (type consistent) triples that are not in \( D^+ \) are false. We will denote this negative set \( D^- = \{x^n \in D | y^n = 0\} \). Unfortunately, \( D^- \) might be very large, since the number of false facts is much larger than the number of true facts. A better way to generate negative examples is to “perturb” true triples. In particular, let us define:

\[
D^- = \{(e_t, r_k, e_j) | e_t \neq e_t \wedge (e_t, r_k, e_j) \in D^+ \}
\]  

and

\[
\cup \{(e_t, r_k, e_j) | e_j \neq e_j \wedge (e_t, r_k, e_j) \in D^+ \}
\]

To understand the difference between this approach and the previous one (where we assumed all valid unknown triples were false), let us consider the example in Figure [III]. The first approach would generate “good” negative triples such as (LeonardNimoy, starredIn, StarWars), (AlecGuinness, starredIn, StarTrek), etc., but also type-consistent but “irrelevant” negative triples such as (BarackObama, starredIn, StarTrek), etc. (We are assuming (for the sake of this example) there is a type Person but not a type Actor.) The second approach (based on perturbation) would not generate negative triples such as (BarackObama, starredIn, StarTrek), since BarackObama does not participate in any starredIn events. This reduces the size of \( D^- \), and encourages it to focus on “plausible” negatives. (An even better method, used in Section [VIII] is to generate the candidate triples from text extraction methods run on the web. Many of these triples will be false, due to extraction errors, but they define a good set of “plausible” negatives.)

Assuming that triples that are missing from the KG are false is known as the closed world assumption. As we stated before, this is not always valid, since the graph is known to be incomplete. A more accurate approach is to make a local-closed world assumption (LCWA) [51] [24]. Using the LCWA is closer to the open world assumption because it only assumes that a KG is locally complete. More precisely, if we have observed any triple for a particular subject-predicate pair \( e_i, r_k \), then we will assume that any non-existing triple \( (e_t, r_k, \cdot) \) is indeed false. (The assumption is valid for functional relations, such as bornIn, but not for set-valued relations, such as starredIn.) However, if we have not observed any triple at all for the pair \( e_i, r_k \), we will assume that all triples \( (e_i, r_k, \cdot) \) are unknown.

E. Pairwise loss training

Given that the negative training examples are not always really negative, an alternative approach to likelihood training is to try to make the probability (or in general, some scoring function) to be larger for true triples than for assumed-to-be-false triples. That is, we can define the following objective function:

\[
\min_{\Theta} \sum_{x^+ \in D^+} \sum_{x^- \in D^-} \mathcal{L}(f(x^+; \Theta), f(x^-; \Theta)) + \lambda \text{reg}(\Theta)
\]  

(5)

where \( \mathcal{L}(f, f') \) is a margin-based ranking loss function such as

\[
\mathcal{L}(f, f') = \max(1 + f - f', 0).
\]  

(6)

This approach has several advantages. First, it does not assume that negative examples are necessarily negative, just that they are “more negative” than the positive ones. Second, it allows the \( f(\cdot) \) function to be any function, not just a probability (but
we do assume that larger $f$ values mean the triple is more likely to be correct.

This kind of objective function is easily optimized by stochastic gradient descent (SGD) [52]: at each iteration, we just sample one positive and one negative example. SGD also scales well to large datasets. However, it can take a long time to converge. On the other hand, as we will see below, some models, when combined with the squared loss objective, can be optimized using alternating least squares (ALS), which is typically much faster.

### F. Statistical properties of knowledge graphs

Knowledge graphs typically adhere to some deterministic rules, such as type constraints and transitivity (e.g., if Leonard Nimoy was born in Boston, and Boston is located in the USA, then we can infer that Leonard Nimoy was born in the USA). However, KGs also typically have various “softer” statistical patterns or regularities, which are not universally true but nevertheless have useful predictive power.

One example of such statistical pattern is known as homophily, that is, the tendency of entities to be related to other entities with similar characteristics. This has been widely observed in various social networks [53] [54]. In our running example, an instance of a homophily pattern might be that Leonard Nimoy is more likely to star in a movie which is made in the USA. For multi-relational data (graphs with more than one kind of link), homophily has also been referred to as autocorrelation [55].

Another statistical pattern is known as block structure. This refers to the property where entities can be divided into distinct groups (blocks), such that all the members of a group have similar relationships to members of other groups [56] [57] [58]. For example, an instance of block structure might be that science fiction actors are more likely to star in science fiction movies, where the group of science fiction actors consists of entities such as Leonard Nimoy and Alec Guinness and the group of science fiction movies of entities such as Star Trek and Star Wars.

Graphs can also exhibit global and long-range statistical dependencies, i.e., dependencies that can span over chains of triples and involve different types of relations. For example, the citizenship of Leonard Nimoy (USA) depends statistically on the city where he was born (Boston), and this dependency involves a path over multiple entities (Leonard Nimoy, Boston, USA) and relations (bornIn, locatedIn, citizenOf). A distinctive feature of relational learning is that it is able to exploit such patterns to create richer and more accurate models of relational domains.

### IV. LATENT FEATURE MODELS

In this section, we assume that the variables $y_{ijk}$ are conditionally independent given a set of global latent features and parameters, as in Equation [1]. We discuss various possible forms for the score function $f(x; \Theta)$ below. What all models have in common is that they explain triples via latent features of entities. For instance, a possible explanation for the fact that Alec Guinness received the Academy Award is that he is a good actor. This explanation uses latent features of entities (being a good actor) to explain observable facts (Guinness receiving the Academy Award). We call these features “latent” because they are not directly observed in the data. One task of all latent feature models is therefore to infer these features automatically from the data.

In the following, we will denote the latent feature representation of an entity $e_i$ by the vector $e_i \in \mathbb{R}^{H_e}$, where $H_e$ denotes the number of latent features in the model. For instance, we could model that Alec Guinness is a good actor and that the Academy Award is a prestigious award via the vectors

$$e_{\text{Guinness}} = \begin{bmatrix} 0.9 \\ 0.2 \end{bmatrix}, \quad e_{\text{AcademyAward}} = \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix}$$

where the component $e_{ij}$ corresponds to the latent feature Good Actor and $e_{ij}$ correspond to Prestigious Award. (Note that, unlike this example, the latent features that are inferred by the following models are typically hard to interpret.)

The key intuition behind relational latent feature models is that the relationships between entities can be derived from interactions of their latent features. However, there are many possible ways to model these interactions, and many ways to derive the existence of a relationship from them. We discuss several possibilities below. See Table III for a summary of the notation.

#### A. RECAL: A bilinear model

RECAL [59] [60] [61] is a relational latent feature model which explains triples via pairwise interactions of latent features. In particular, we model the score of a triple $x_{ijk}$ as

$$f^{\text{RECAL}}_{ijk} := e_i^T W_k e_j = \sum_{a=1}^{H_e} \sum_{b=1}^{H_e} w_{abk} e_{ia} e_{jb} \quad (7)$$

where $W_k \in \mathbb{R}^{H_e \times H_e}$ is a weight matrix whose entries $w_{abk}$ specify how much the latent features $a$ and $b$ interact in the $k$-th relation. We call this a bilinear model, since it captures the interactions between the two entity vectors using multiplicative terms. For instance, we could model the pattern that good actors are likely to receive prestigious awards via a weight matrix such as

$$W_{\text{receivedAward}} = \begin{bmatrix} 0.1 & 0.9 \\ 0.1 & 0.1 \end{bmatrix}.$$ 

In general, we can model block structure patterns via the magnitude of entries in $W_k$, while we can model homophily patterns via the magnitude of its diagonal entries. Anti-correlations in these patterns can be modeled efficiently via negative entries in $W_k$.

Hence, in Equation (7) we compute the score of a triple $x_{ijk}$ via the weighted sum of all pairwise interactions between the latent features of the entities $e_i$ and $e_j$. The parameters of the model are $\Theta = \{\{e_i\}_{i=1}^{N_e}, \{W_k\}_{k=1}^{N_r}\}$. During training we jointly learn the latent representations of entities and how the latent features interact for particular relation types.

In the following, we will discuss further important properties of the model for learning from knowledge graphs.
Relational data

Symbol | Meaning | Size
---|---|---
$N_e$ | Number of entities | $N_e$ x $N_e$ x $N_e$
$N_r$ | Number of relations | $N_r$ x $N_r$ x $N_r$
$N_d$ | Number of training examples | $N_d$
$e_i$ | $i$-th entity in the dataset (e.g., LeonardNimoy) | 
$r_k$ | $k$-th relation in the dataset (e.g., bornIn) | 
$D^+$ | Set of observed positive triples | 
$D^-$ | Set of observed negative triples | 

Probabilistic Knowledge Graphs

Symbol | Meaning | Size
---|---|---
$Y$ | (Partially observed) labels for all triples | $N_e$ x $N_e$ x $N_r$
$F$ | Score for all possible triples | $N_e$ x $N_e$ x $N_r$
$Y_k$ | Slice of $Y$ for relation $r_k$ | $N_e$ x $N_r$
$F_k$ | Slice of $F$ for relation $r_k$ | $N_e$ x $N_e$

Graph and Latent Feature Models

Symbol | Meaning | Size
---|---|---
$\phi_{ijk}$ | Feature vector representation of triple $(e_i, r_k, e_j)$ | 
$w_k$ | Weight vector to derive scores for relation $k$ | 
$\Theta$ | Set of all parameters of the model | 
$\sigma(\cdot)$ | Sigmoid (logistic) function | 

Latent Feature Models

Symbol | Meaning | Size
---|---|---
$H_e$ | Number of latent features for entities | $N_e$
$H_r$ | Number of latent features for relations | $N_r$
$e_i$ | Latent feature representation of entity $e_i$ | $H_e$
$r_k$ | Latent feature representation of relation $r_k$ | $H_r$
$H_a$ | Size of $h_a$ layer | 
$H_b$ | Size of $h_b$ layer | 
$H_c$ | Size of $h_c$ layer | 
$E$ | Entity embedding matrix | $N_e$ x $H_e$
$W_k$ | Bilinear weight matrix for relation $k$ | $H_r$ x $H_e$
$A_k$ | Linear feature map for pairs of entities | $(2H_e) x H_a$
$C$ | Linear feature map for triples | $(2H_e + H_r) x H_e$

Relational learning via shared representations: In equation (7), entities have the same latent representation regardless of whether they occur as subjects or objects in a relationship. Furthermore, they have the same representation over all different relation types. For instance, the $i$-th entity occurs in the triple $x_{ijk}$ as the subject of a relationship of type $k$, while it occurs in the triple $x_{piq}$ as the object of a relationship of type $q$. However, the predictions $f_{ijk} = e_i^T W_k e_j$ and $f_{piq} = e_j^T W_k e_i$ both use the same latent representation $e_i$ of the $i$-th entity. Since all parameters are learned jointly, these shared representations permit to propagate information between triples via the latent representations of entities and the weights of relations. This allows the model to capture global dependencies in the data.

Semantic embeddings: The shared entity representations in RESCAL capture also the similarity of entities in the relational domain, i.e., that entities are similar if they are connected to similar entities via similar relations. For instance, if the representations of $e_i$ and $e_j$ are similar, the predictions $f_{ijk}$ and $f_{ijq}$ will have similar values. In return, entities with many similar observed relationships will have similar latent representations. This property can be exploited for entity resolution and has also enabled large-scale hierarchical clustering on relational data. Moreover, since relational similarity is expressed via the similarity of vectors, the latent representations $e_i$ can act as proxies to give non-relational machine learning algorithms such as $k$-means or kernel methods access to the relational similarity of entities.

Connection to tensor factorization: RESCAL is similar to methods used in recommendation systems and to traditional tensor factorization methods. In matrix notation, Equation (7) can be written compactly as $F_k = E W_k E^T$, where $F_k \in \mathbb{R}^{N_e \times N_r}$ is the matrix holding all scores for the $k$-th relation and the $i$-th row of $E \in \mathbb{R}^{N_e \times H_e}$ holds the latent representation of $e_i$. See Figure 3 for an illustration. In the following, we will use this tensor representation to derive a very efficient algorithm for parameter estimation.

Fitting the model: If we want to compute a full probabilistic model, the parameters of RESCAL can be estimated by minimizing the log-loss using gradient-based methods such as stochastic gradient descent. However, if we don’t require a full probabilistic model, we can use an alternative approach to estimate the parameters very efficiently. First, let us make a closed world assumption, i.e., we assume every value of $y_{ijk}$ is known, and is either 0 or 1. Second, instead of the log-loss, we will use squared loss.

Thus we have the following optimization problem:

$$\min_{E, W_k} \sum_k \| Y_k - EW_k E^T \|^2_F + \lambda_1 \| E \|^2_F + \lambda_2 \sum_k \| W_k \|^2_F.$$

where $\lambda_1, \lambda_2 \geq 0$ control the degree of regularization. The main advantage of Equation (8) is that it can be optimized via alternating least squares (ALS), in which the latent factors $E$ and $W_k$ can be computed via a sequence of very efficient, alternating closed-form updates.

It has been shown analytically that a single update of $E$ and $W_k$ in RESCAL-ALS scales linearly with the number of entities $N_e$, linearly with the number of relations $N_r$, and linearly with the number of observed triples, i.e., the number of non-zero entries in $Y$. In practice, a small number (say 30 to 50) of iterated updates are often sufficient for the algorithm to arrive at stable estimates of the parameters. Given a current estimate of $E$, the updates for each $W_k$ can be computed in parallel to improve the scalability on knowledge graphs with a large number of relations. Furthermore, by exploiting the special tensor structure of RESCAL, we can derive improved updates for RESCAL-ALS that compute the estimates for the parameters with a runtime complexity of $O(H_e^2)$ for a single update (as opposed to a runtime complexity...
of $O(H^2)$ for naive updates) [61] [65]. Hence, for relational domains that can be explained via a moderate number of latent features, RESCAL-ALS is highly scalable and very fast to compute.

**Decoupled Prediction:** In Equation (1), the probability of single relationship is computed via simple matrix-vector products in $O(H^2)$ time. Hence, once the parameters have been estimated, the computational complexity to predict the score of a triple depends only on the number of latent features and is independent of the size of the graph. However, during parameter estimation, the model can capture global dependencies due to the shared latent representations.

**Relational learning results:** RESCAL has been shown to achieve state-of-the-art on a number of relational learning tasks. For instance, [59] showed that RESCAL provides comparable or better relationship prediction results on a number of small benchmark datasets compared to Markov Logic Networks (with structure learning) [66], the Infinite (Hidden) Relational model [67] [68], and Bayesian Clustered Tensor Factorization [69]. Moreover, RESCAL has been used for link prediction on entire knowledge graphs such as YAGO and DBpedia [60] [70]. Aside from link prediction, RESCAL has also successfully been applied to SRL tasks such as entity resolution and link-based clustering. For instance, RESCAL has shown state-of-the-art results in predicting which authors, publications, or publication venues are likely to be identical in publication databases [59] [61]. Furthermore, the semantic embedding of entities computed by RESCAL has been exploited to create taxonomies for uncategorized data via hierarchical clusterings of entities in the embedding space [71].

**B. Other tensor factorization models**

Various other tensor factorization methods have been explored for learning from knowledge graphs and multi-relational data. [2] [23] factorized adjacency tensors using the CP tensor decomposition to analyze the link structure of Web pages and Semantic Web data respectively. [24] applied pairwise interaction tensor factorization [75] to predict triples in knowledge graphs. [76] applied factorization machines to large uni-relational datasets in recommendation settings. [77] proposed a tensor factorization model for knowledge graphs with a very large number of different relations.

It is also possible to use discrete latent factors. [78] proposed Boolean tensor factorization to disambiguate facts extracted with OpenIE methods and applied it to large datasets [79]. In contrast to previously discussed factorizations, Boolean tensor factorizations are discrete models, where adjacency tensors are decomposed into binary factors based on Boolean algebra.

Another approach for learning from knowledge graphs is based on matrix factorization, where, prior to the factorization, the adjacency tensor $\mathbf{Y} \in \mathbb{R}^{N_x \times N_x \times N_e}$ is reshaped into a matrix $\mathbf{Y} \in \mathbb{R}^{N_x^2 \times N_e}$ by associating rows with subject-object pairs $(e_i, e_j)$ and columns with relations $r_k$ (cf. [80] [81]), or into a matrix $\mathbf{Y} \in \mathbb{R}^{N_e \times N_x \times N_x}$ by associating rows with subjects $e_i$ and columns with relation/objects $(r_k, e_j)$ (cf. [82] [83]). Unfortunately, both of these formulations lose information compared to tensor factorization. For instance, if each subject-object pair is modeled via a different latent representation, the information that the relationships $y_{i,j,k}$ and $y_{j,i,k}$ share the same object is lost. It also leads to an increased memory complexity, since a separate latent representation is computed for each pair of entities, requiring $O(N_x^2 H_e + N_e H_x^2)$ parameters (compared to $O(N_e H_e + N_e H_x^2)$ parameters for RESCAL).

**C. Multi-layer perceptrons**

We can interpret RESCAL as creating composite representations of triples and deriving their existence from this representation. In particular, we can rewrite RESCAL as

$$J_{i,j,k}^{\text{RESCAL}} := w_k^T \phi_{i,j}^{\text{RESCAL}}$$

$$\phi_{i,j}^{\text{RESCAL}} := e_j \otimes e_i,$$

where $w_k = \text{vec}(\mathbf{W}_k)$. Equation (9) follows from Equation (7) via the equality $\text{vec}(\mathbf{AXB}) = (\mathbf{B}^T \otimes \mathbf{A}) \text{vec}(\mathbf{X})$. Hence, RESCAL represents pairs of entities $(e_i, e_j)$ via the tensor product of their latent feature representations (Equation (10)) and predicts the existence of the triple $x_{i,j,k}$ from $\phi_{i,j}$ via $w_k$ (Equation (9)). See also Figure [4]. For a further discussion of the tensor product to create composite latent representations please see [84] [85] [86].

Since the tensor product explicitly models all pairwise interactions, RESCAL can require a lot of parameters when the number of latent features are large (each matrix $\mathbf{W}_k$ has $H^2$ entries). This can, for instance, lead to scalability problems on knowledge graphs with a large number of relations.

In the following we will discuss models based on multi-layer perceptrons, which allow us to consider alternative ways to create composite triple representations and allow to use nonlinear functions to predict their existence.

In particular, let us define the following multi-layer perceptron (MLP) model:

$$J_{i,j,k}^{\text{E-MLP}} := w_k^T \mathbf{g}(\mathbf{h}_a)$$

$$\mathbf{h}_a := \mathbf{A}_k \phi_{i,j}^{\text{E-MLP}}$$

$$\phi_{i,j}^{\text{E-MLP}} := [e_i; e_j]$$

where $\mathbf{g}(\mathbf{u}) = [g(u_1), g(u_2), \ldots]$ is the function $g$ applied element-wise to vector $\mathbf{u}$; one often uses the nonlinear function $g(u) = \tanh(u)$. We call this model the E-MLP.

In the E-MLP model, we create a composite representation $\phi_{i,j,k}^{\text{E-MLP}} = [e_i; e_j] \in \mathbb{R}^{2H_4}$ via the concatenation of $e_i$ and $e_j$. However, concatenation alone does not consider any interactions between the latent features of $e_i$ and $e_j$. For this reason, we add a (vector-valued) hidden layer $\mathbf{h}_a$ of size $H_4$, from which the final prediction is derived via $w_k^T \mathbf{g}(\mathbf{h}_a)$. The important difference to tensor-product models like RESCAL is that we learn the interactions of latent features via the matrix $\mathbf{A}_k$ (Equation (12)), while the tensor product considers always all possible interactions between latent features. This adaptive approach can reduce the number of required parameters significantly, especially on datasets with a large number of relations.

One disadvantage of the E-MLP is that it has to define a matrix $\mathbf{A}_k$ for every possible relation, which again requires a
D. Other neural network models

We can also combine traditional MLPs with bilinear models, resulting in what [89] calls a “neural tensor network” (NTN). More precisely, we can define the NTN model as follows:

$$f_{ijk}^{\text{NTN}} := w_k^T \mathbf{g}(h_i; h_k)$$

$$h_i := [h_k^1, \ldots, h_k^{H_k}]$$

$$h_k^i := e_i^T B_k^i e_j,$$  

$$i, j = 1, \ldots, n$$

2The relations edu-start, edu-end, job-start, job-end represent the start and end dates of attending an educational institution and holding a particular job, respectively

Here $\mathbf{B}_k$ is a tensor, where the $\ell$-th slice $\mathbf{B}_k^\ell$ has size $H_c \times H_c$, and there are $H_k$ slices. This models combines the bilinear representation of RESCAL with the additive model of the E-MLP. The NTN model has many more parameters than the MLP or RESCAL (bilinear) models. Indeed, the results in [91] and [24] both show that it tends to overfit, at least on the (relatively small) datasets uses in those papers.

In addition to multiplicative models such as RESCAL, the MLPs, and NTN, models with additive interactions between latent features have been proposed. In particular, latent distance models (also known as latent space models in social network analysis) derive the probability of relationships from the distance between latent representations of entities: entities are likely to be in a relationship if their latent representations are close according to some distance measure. For uni-relational data, [22] proposed this approach first in the context of social networks by modeling the probability of a relationship $x_{ij}$ via the score function $f(e_i, e_j) = -d(e_i, e_j)$ where $d(\cdot, \cdot)$ refers to an arbitrary distance measure such as the Euclidean distance. The structured embedding (SE) model [88] extends this idea to multi-relational data by modeling the score of a triple $x_{ijk}$ as:

$$f_{ijk}^{\text{SE}} := -\|A_k^i e_i - A_k^j e_j\|_1 = -\|h_i\|_1$$

where $A_k^i = [A_k^1; \ldots; A_k^n]$. In Equation (20) the matrices $A_k^i, A_k^j$ transform the global latent feature representations of entities to model relationships specifically for the $k$-th relation. The transformations are learned using the ranking loss in a way such that pairs of entities in existing relationships are closer to each other than entities in non-existing relationships.

To reduce the number of parameters over the SE model, the TransE model [90] translates the latent feature representations via a relation-specific offset instead of transforming them via matrix multiplications. In particular, the score of a triple $x_{ijk}$ is defined as:

$$f_{ijk}^{\text{TransE}} := -d(e_i + r_k, e_j).$$

This model is inspired by the results in [87], who showed that some relationships between words could be computed by their vector difference in the embedding space. As noted in [90], under unit-norm constraints on $e_i, e_j$ and using the squared Euclidean distance, we can rewrite Equation (21) as follows:

$$f_{ijk}^{\text{TransE}} = -(2r_k^T (e_i - e_j) - 2e_i^T e_j + \|r_k\|_2^2)$$
TABLE V
SUMMARY OF THE LATENT FEATURE MODELS, \( h_a, h_b \) AND \( h_c \) ARE HIDDEN LAYERS OF THE NEURAL NETWORK; SEE TEXT FOR DETAILS.

<table>
<thead>
<tr>
<th>Method</th>
<th>( f )</th>
<th>Hidden Feature Maps ( A_k )</th>
<th>Hidden Interactions ( C )</th>
<th>Number of Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Embeddings [88]</td>
<td>( -</td>
<td>h_a</td>
<td>_1 )</td>
<td>( [A_k^+; -A_k^-] )</td>
</tr>
<tr>
<td>E-MLP [89]</td>
<td>( w_k^e g(h_a) )</td>
<td>( [A_k^+; A_k^-] )</td>
<td>-</td>
<td>( N_eH_a + 2N_eH_a + N_eH_e )</td>
</tr>
<tr>
<td>ER-MLP [24]</td>
<td>( w_k^t g(h_a) )</td>
<td>-</td>
<td>( C )</td>
<td>( H_e + N_eH_e + N_eH_e )</td>
</tr>
<tr>
<td>TransE [90]</td>
<td>( -(2h_a - 2h_b +</td>
<td>r_k</td>
<td>^2 )</td>
<td>( [r_k; -r_k] )</td>
</tr>
<tr>
<td>Bilinear (RESCAL) [60]</td>
<td>( w_k^e h_b )</td>
<td>( B_k ) = [I]</td>
<td>-</td>
<td>( N_eH_k^2 + N_eH_e )</td>
</tr>
<tr>
<td>NTN [89]</td>
<td>( w_k^d g([h_a; h_b]) )</td>
<td>( B_k ) = ( {B_l^k } )</td>
<td>-</td>
<td>( \frac{1}{2}N_eH_b + N_e(H_b + H_a) + \frac{1}{2}N_eH_e )</td>
</tr>
</tbody>
</table>

Furthermore, if we assume \( A_k = [r_k; -r_k] \), \( H_b = 1 \) and \( B_k = [I] \), then we can rewrite this model as follows:

\[
f_{ij,k}^{\text{TransE}} = -(2h_a - 2h_b + |r_k|^2). \tag{23}
\]

All of the models we discussed are shown in Figure 5. See Table V for a summary. [91] perform an experimental comparison of these models; they show that the RESCAL model, and especially a diagonal version of it, does best on two different link prediction tasks. (They do not consider ER-MLP in that paper, but this is shown to be better than NTN in [24], presumably due to overfitting. A more detailed experimental comparison, on larger datasets, is left to future work.)

V. GRAPH FEATURE MODELS

In this section, we assume that the existence of an edge can be predicted by extracting features from the observed edges in the graph. For example, due to social conventions, parents of a person are often married, so we could predict the triple (John, marriedTo, Mary) from the existence of the path John \( \xrightarrow{\text{parentOf}} \) Anne \( \xrightarrow{\text{parentOf}} \) Mary, representing a common child. In contrast to latent feature models, this kind of reasoning explains triples via observable variables, i.e., directly from the observed triples in the knowledge graph. We will now discuss some models of this kind.

A. Similarity measures for uni-relational data

Observable graph feature models are widely used for link prediction in graphs that consist only of a single relation, e.g., as in social network analysis (friendships between people), biology (interactions of proteins), and Web mining (hyperlinks between Web sites). The intuition behind these methods is that similar entities are likely to be related (homophily) and that the similarity of entities can be derived from the neighborhood of nodes or from the existence of paths between nodes. For this purpose, various indices have been proposed to measure the similarity of entities, which can be classified into local, global, and quasi-local approaches [93].

Local similarity indices such as Common Neighbors, the Adamic-Adar index [94] or Preferential Attachment [95] derive the similarity of entities from their number of common neighbors or their absolute number of neighbors. Local similarity indices are fast to compute for single relationships and scale well to large knowledge graphs as their computation depends only on the direct neighborhood of the involved entities. However, they can be too localized to capture important patterns in relational data and cannot model long-range or global dependencies.

Global similarity indices such as the Katz index [96] and the Leicht-Holme-Newman index [97] derive the similarity of entities from the ensemble of all paths between entities, while indices like Hitting Time, Commute Time, and PageRank [98] derive the similarity of entities from random walks on the graph. Global similarity indices often provide significantly better predictions than local indices, but are also computationally more expensive [93, 54].

Quasi-local similarity indices like the Local Katz index [54] or Local Random Walks [99] try to balance predictive accuracy and computational complexity by deriving the similarity of entities from paths and random walks of bounded length.

In the following, we will discuss an approach that extends this idea of quasi-local similarity indices for uni-relational networks to learn from large multi-relational knowledge graphs.

B. Path Ranking Algorithm

The Path Ranking Algorithm (PRA) [100, 101] extends the idea of using random walks of bounded lengths for predicting links in multi-relational knowledge graphs. In particular, let \( \pi_L(i, j, k, t) \) denote a path of length \( L \) of the form \( e_1 \xrightarrow{r_1} e_2 \xrightarrow{r_2} \cdots \xrightarrow{r_{L-1}} e_j \), where \( t \) represents the sequence of edge types \( t = (r_1, r_2, \ldots, r_L) \). We also require there to be a direct arc \( e_i \xrightarrow{r_k} e_j \), representing the existence of a relationship of type \( k \) from \( e_i \) to \( e_j \). Let \( \Omega_L(i, j, k) \) represent the set of all such paths of length \( L \), ranging over path types \( t \). (We can discover such
We can then predict the edge probabilities using logistic regression:

$$f_{ijk} = \frac{1}{1 + e^{-\sum w_k \phi_{ijk}}}$$

Interpretability: A useful property of PRA is that its model is easily interpretable. In particular, relation paths can be regarded as bodies of weighted rules — more precisely Horn clauses — where the weight specifies how predictive the body of the rule is for the head. For instance, Table VI shows some relation paths along with their weights that have been learned by PRA in the KV project (see Section VIII) to predict which college a person attended. The first relation path in Table VI can be interpreted as follows: it is likely that a person attended a college if the person drafted the college.

By using a sparsity promoting prior on $w_k$, we can perform feature selection, which is equivalent to rule learning.

Relational learning results: PRA has been shown to outperform the deterministic relational learning method FOIL [102] for link prediction in NELL [101]. It has also been shown to have comparable performance to ER-MLP on link prediction in KV: PRA obtained a result of 0.884 for the area under the ROC curve, as compared to 0.882 for ER-MLP [24].

VI. COMBINING LATENT AND GRAPH FEATURE MODELS

It has been observed experimentally (see, e.g., [24]) that neither state-of-the-art relational latent feature models (RLFMs) nor state-of-the-art graph feature models are superior for learning from knowledge graphs. Instead, the strengths of latent and graph-based models are often complementary, as both families focus on different aspects of relational data:

- Latent feature models are well-suited for modeling global relational patterns via newly introduced latent variables. They are computationally efficient if triples can be explained with a small number of latent variables.
- Graph feature models are well-suited for modeling local and quasi-local graphs patterns. They are computationally efficient if triples can be explained from the neighborhood of entities or from short paths in the graph.

There has also been some theoretical work comparing these two approaches [103]. In particular, it has been shown that tensor factorization can be inefficient when relational data consists of a large number of strongly connected components. Fortunately, such “problematic” relations can often be handled efficiently via graph-based models. A good example is the marriedTo relation: One marriage corresponds to a single strongly connected component, so data with a large number of marriages would be difficult to model with RLFMs. However, predicting marriedTo links via graph-based models is easy: the existence of the triple $(John, marriedTo, Mary)$ can be simply predicted from the existence of $(Mary, marriedTo, John)$ by exploiting the symmetry of the relation. If the $(Mary, marriedTo, John)$ edge is unknown, we can use statistical patterns, such as the existence of shared children.

Combining the strengths of latent and graph-based models is therefore a promising approach to increase the predictive performance of graph models. It typically also speeds up the training. We now discuss some ways of combining these two kinds of models.

A. Additive relational effects model

[103] proposed the additive relational effects (ARE), which is a way to combine RLFMs with observable graph models. In particular, if we combine RESCAL with PRA, we get

$$f_{ijk}^{\text{RESCAL+PRA}} = w_k \phi_{ijk}^{\text{RESCAL}} + w_k \phi_{ijk}^{\text{PRA}}.$$  (26)

ARE models can be trained by alternately optimizing the RESCAL parameters with the PRA parameters. The key benefit is now RESCAL only has to model the “residual errors” that cannot be modelled by the observable graph patterns. This allows the method to use much lower latent dimensionality, which significantly speeds up training time. The resulting combined model also has increased accuracy [103].

B. Other combined models

In addition to ARE, further models have been explored to learn jointly from latent and observable patterns on relational data. [80] combined a latent feature model with an additive term to learn from latent and neighborhood-based information on multi-relational data, as follows:

$$f_{ijk}^{\text{ADD}} = w_k^{(1)} \phi_{ijk}^{\text{SUB}} + w_k^{(2)} \phi_{ijk}^{\text{OBJ}} + w_k^{(3)} \phi_{ijk}^{\text{N}}$$

$$\phi_{ijk}^{\text{N}} = \{ y_{ij,k'} : k' \neq k \}$$  (27)
Here, $\phi_{ij}^{\text{SUB}}$ is the latent representation of entity $e_i$ as a subject and $\phi_{ij}^{\text{OBJ}}$ is the latent representation of entity $e_j$ as an object. Furthermore, [81] considered an additional term

$$f_{ijk}^{\text{ADD}} := w_k \phi_{ij}$$

where $\phi_{ij}^{\text{SUB+OBJ}}$ is a (non-composite) latent feature representation of subject-object pairs. The main idea behind $\phi_{ijk}^{N}$ in Equations (28) and (29) is to model patterns efficiently where the existence of a triple $y_{ijk}$ is predictive of another triple $y_{ij}$ between the same pair of entities (but of a different relation type). For instance, if Leonard Nimoy was born in Boston, it is also likely that he lived in Boston. This dependency between the relation types bornIn and livedIn can be modeled in Equation (29) by assigning a large weight to $w_{\text{bornIn, livedIn}}$.

Note that this kind of neighborhood information can be included in ARE if we set $\phi_{ijk} = \{y_{ijk} : k = 1 : N_i\}$, but we force $w_k^{(2)}$ in Equation (26) to have a zero entry $w_k^{(2)}$, to avoid using $y_{ijk}$ to predict $y_{ij}$.

ARE and the models of [80] and [81] are similar in spirit to the model of [104], which augments SVD (i.e., matrix factorization) of a rating matrix with additive terms to include local neighborhood information. Similarly, factorization machines [105] allow to combine latent and observable patterns, by modeling higher-order interactions between input variables via low-rank factorizations [74].

An alternative way to combine different prediction systems is to fit them separately, and use their outputs as inputs to another “fusion” system. This is called stacking [106]. For instance, [24] used the output of PRA and ER-MLP as scalar features, and learned a final “fusion” layer by training a binary classifier. Stacking has the advantage that it is very flexible in the kinds of models that can be combined. However, it has the disadvantage that the individual models cannot cooperate, and thus any individual model needs to be more complex than in a combined model which is trained jointly. For example, if we fit RESCAL separately from PRA, we will need a larger number of latent features than if we fit them jointly.

VII. MARKOV RANDOM FIELDS

In Markov Random Fields (MRFs) one assumes that the $y_{ijk}$ interact locally, i.e. there is interaction between different $y_{ijk}$ that are close in the graph. Interactions are modeled via potential functions that can be defined in various ways. A common approach is to use “Markov logic” [107], which is a template language for defining potential functions on dependency graphs of arbitrary size (sometimes called a “relational MRF”) via logical formulae.

MRFs are a useful tool for modeling KGs, as shown in [108]. However, they suffer from several computational problems: the difficulty of rule learning, the difficulty of prediction, and the difficulty of estimating the parameters. The rule learning problem has been studied in various papers (see Section V-B and [51][21]), but the general problem is intractable.

The prediction/inference problem (of computing the MAP state estimate) is (in general) also NP-hard. Various approximations to this problem have been proposed in the graphical models literature, such as Gibbs sampling (see e.g., [26][109]).

An interesting recent approach, called Probabilistic Soft Logic (PSL) [110], is based on a continuous relaxation. The resulting system can be scaled to fairly large knowledge bases, as shown in [111].

The parameter estimation problem (which is usually cast as maximum likelihood or MAP estimation), although convex, is in general quite expensive, since it needs to call prediction as a subroutine. Various approximations, such as pseudo likelihood, have been developed (cf., relational dependency networks in [112]). However, these approaches still don’t have the flexibility of pairwise loss minimization.

In summary, although relational MRFs are a useful tool, they are not as easy to scale as the other approaches we consider in this paper.

VIII. THE KNOWLEDGE VAULT: RELATIONAL LEARNING FOR KNOWLEDGE BASE CONSTRUCTION

The Knowledge Vault (KV) [24] is a very large-scale automatically constructed knowledge base, which follows the Freebase schema (KV uses the 4469 most common predicates). It is constructed in three steps. In the first step, content from different Web sources such as texts, tabular data, page structure, and human annotations is extracted from Web sources (the extractors are described in detail in [24]). Second, an SRL model is trained on Freebase to serve as a “prior” for computing the probability of (new) edges. Finally, the confidence in the automatically extracted facts is evaluated using both the extraction scores and the prior SRL model.

The KV system trains both a PRA model and an ER-MLP model to predict links in the knowledge graph. These are combined using stacking, as discussed above. The scores from the fused link-prediction model are then combined with various features derived from the extracted triples, based on the confidence of the extractors, the number of (de-duped) Web pages in which the triples were found, etc. See Figure 6 for an illustration.

We now give a qualitative example of the benefits of combining the prior with the extractors (i.e., the Fusion Layer
in Figure [6]. Suppose the extraction pipeline extracted a triple corresponding to the following relation:\footnote{3}{For clarity of presentation we show a simplified triple. Please see [24] for the actually extracted triples including complex value types (CVT).}

(Barry Richter, attended, University of Wisconsin-Madison).

The extraction confidence for this triple (obtained by fusing multiple extraction techniques) was just 0.14, since it was based on the following two rather indirect statements:\footnote{4}{Source: \url{http://www.legendsofhockey.net/LegendsOfHockey/jsp/SearchPlayer.jsp?player=11371}.

\url{http://host.madison.com/sports/high-school/hockey/numbers-dwinding-for-once-mighty-madison-high-school-hockey-programs/article_95843e00-ec34-11df-9da9-001cc4002e0.html}.}

In the fall of 1989, Richter accepted a scholarship to the University of Wisconsin, where he played for four years and earned numerous individual accolades...

and\footnote{5}{Source: \url{http://www.legendsofhockey.net/LegendsOfHockey/jsp/SearchPlayer.jsp?player=11377}.}

The Polar Caps’ cause has been helped by the impact of knowledgeable coaches such as Andrinda, Bye and former UW teammates Chris Tancill and Barry Richter.

However, we know from Freebase that Barry Richter was born and raised in Madison, WI. This increases our prior belief that he went to school there, resulting in a final fused belief of 0.61.

Combining the prior model (learned using SRL methods) with the information extraction model improved performance significantly, increasing the number of high confidence triples (those with a calibrated probability above 90%) from 100M (based on extractors alone) to 271M (based on extractors plus prior). This is perhaps one of the largest applications of SRL to KBs to date. See [24] for further details.

IX. EXTENSIONS AND FUTURE WORK

A. Non-binary relations

So far we completely focused on binary relations; here we discuss how relations of other cardinalities can be handled.

Unary relations: Unary relations refer to statements on properties of entities, e.g., the height of a person. Such data can naturally be represented by a matrix, in which rows represent entities, and columns represent attributes. \footnote{6}{Source: \url{http://www.ontologylib.com/ontologylib/txt/ontologies/KB-DBpedia-2011/ontologyspace.txt}.}

proposed a joint tensor-matrix factorization approach to learn simultaneously from binary and unary relations via a shared latent representation of entities. In this case, we may also need to modify the likelihood function, so it is Bernoulli for binary edge variables, and Gaussian (say) for numeric features or Possion for count data (see [113]).

Higher-order relations: In knowledge graphs, higher-order relations are typically expressed via multiple binary relations. In Section [11] we expressed the ternary relationship playedCharacterIn(LeonardNimoy, Spock, StarTrek) via two binary relationships (LeonardNimoy, played, Spock) and (Spock, characterIn, StarTrek). However, there are multiple actors who played Spock in different Star Trek movies. To model this without loss of information, we can use auxiliary nodes to identify the respective relationship. For instance, to model the relationship

\[
\text{playedCharacterIn}(\text{LeonardNimoy}, \text{Spock}, \text{StarTrek-1})
\]

where we used the auxiliary entity MovieRole-1 to uniquely identify this particular relationship. In most applications auxiliary entities get an identifier; if not they are referred to as blank nodes. In Freebase auxiliary nodes are called Compound Value Types (CVT). Since higher-order relations involving time and location are relatively common, the YAGO2 project extended the SPO triple format to the (subject, predicate, object, time, location) (SPOTL) format to model temporal and spatial information about relationships explicitly, without transforming them to binary relations [23].

A related issue is that the truth-value of a fact can change over time. For example, Google’s current CEO is Larry Page, but from 2001 to 2011 it was Eric Schmidt. Both facts are correct, but only during the specified time interval. For this reason, Freebase allows some facts to be annotated with beginning and end dates, using CVT (compound value type) constructs, which represent n-ary relations via auxiliary nodes.

In the future, it is planned to extend the KV system to model such temporal facts. However, this is non-trivial, since it is not always easy to infer the duration of a fact from text, since it is not necessarily related to the timestamp of the corresponding source (cf. [114]).

As an alternative to the usage of auxiliary nodes, a set of \(n\)--th-order relations can be represented by a single \(n+1\)--th-order tensor. RESCAL can easily be generalized to higher-order relations and can be solved by higher-order tensor factorization or by neural network models with the corresponding number of entity representations as inputs [113].

B. Hard constraints: types, functional constraints, and others

Imposing hard constraints on the allowed triples in knowledge graphs can be useful. Powerful ontology languages such as the Web Ontology Language (OWL) [115] have been developed, in which complex constraints can be formulated. However, reasoning with ontologies is computationally demanding, and hard constraints are often violated in real-world data [116] [117]. Fortunately, machine learning methods can be robust in the face of contradictory evidence.

Deterministic dependencies: Triples in relations such as subClassOf and isLocatedIn follow clear deterministic dependencies such as transitivity. For example, if Leonard Nimoy was born in Boston, we can conclude that he was born in Massachusetts, that he was born in the USA, that he was born in North America, etc. One way to consider such ontological constraints is to precompute all true triples that can be derived from the constraints and to add them to the knowledge graph prior to learning. The precomputation of triples according to ontological constraints is also called materialization. However, on large knowledge graphs, full materialization can be computationally demanding.
Type constraints: Often relations only make sense when applied to entities of the right type. For example, the domain and the range of marriedTo is limited to entities which are persons. Modelling type constraints explicitly requires complex manual work. An alternative is to learn approximate type constraints by simply considering the observed types of subjects and objects in a relation. The standard RESCAL model has been extended by [20] and [63] to handle type constraints of relations efficiently. As a result, the rank required for a good RESCAL model can be greatly reduced. Furthermore, [81] considered learning latent representations for the argument slots in a relation to learn the correct types from data.

Functional constraints and mutual exclusiveness: Although the methods discussed in Sections IV and V can model long-range and global dependencies between triples, they do not explicitly enforce functional constraints that induce mutual exclusivity between possible values. For instance, a person is born in exactly one city, etc. If one of the these values is observed, then observable graph models can prevent other values from being asserted, but if all the values are unknown, the resulting mutual exclusion constraint can be hard to deal with computationally.

C. Generalizing to new entities and relations

In addition to missing facts, there are many entities that are mentioned on the Web but are currently missing in knowledge graphs like Freebase and YAGO. If new entities or predications are added to a KG, one might want to avoid retraining the model due to runtime considerations. Given the current model and a set of newly observed relationships, latent representations of new entities can be calculated approximately in both tensor factorization models and in neural networks, by finding representations that explain the newly observed relationships relative to the current model. Similarly, it has been shown that the relation-specific weights $W_k$ in the RESCAL model can be calculated efficiently for new relation types given already derived latent representations of entities [118].

D. Querying probabilistic knowledge graphs

RESCAL and KV can be viewed as probabilistic databases (see, e.g., [119] [120]). In the Knowledge Vault, only the probabilities of triples are queried. Some applications might require more complex queries such as: Who is born in Rome and likes someone who is a child of Albert Einstein. It is known that queries involving joins (existentially quantified variables) are expensive to calculate in probabilistic databases ([119]). In [118], it was shown how some queries involving joins can be efficiently handled within the RESCAL framework.

X. Concluding Remarks

We have provided a review of state-of-the-art statistical relational learning (SRL) methods applied to very large knowledge graphs. We have also demonstrated how statistical relational learning can be used in conjunction with machine reading and information extraction methods to automatically build such knowledge repositories. As a result, we have shown how to create a truly massive, machine-interpretable “semantic memory” of facts, which is already empowering many important applications. However, although these KGs are impressive in their size, they still fall short of representing the kind of knowledge that humans possess. Notably missing are representations of “common sense” facts (such as the fact that water is wet, wet things can be slippery, etc.), as well as “procedural” or how-to knowledge (such as how to drive a car, how to send an email, etc.) Representing, learning, and reasoning with this kind of knowledge remains the next frontier for AI and machine learning.

ACKNOWLEDGMENT

Maximilian Nickel acknowledges support by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216. Volker Tresp acknowledges support by the German Federal Ministry for Economic Affairs and Energy, technology program “Smart Data” (grant 01MT14001).

REFERENCES


2003.


