Combining RDF Graph Data and Embedding Models for an Augmented Knowledge Graph

Andriy Nikolov
metaphacts GmbH
Walldorf, Germany
an@metaphacts.com

Peter Haase
metaphacts GmbH
Walldorf, Germany
ph@metaphacts.com

Daniel M. Herzig
metaphacts GmbH
Walldorf, Germany
dh@metaphacts.com

Johannes Trame
metaphacts GmbH
Walldorf, Germany
jt@metaphacts.com

Artem Kozlov
metaphacts GmbH
Walldorf, Germany
ak@metaphacts.com

ABSTRACT

Vector embedding models have recently become popular for encoding both structured and unstructured data. In the context of knowledge graphs such models often serve as additional evidence supporting various tasks related to the knowledge base population: e.g., information extraction or link prediction to expand the original dataset. However, the embedding models themselves are often not used directly alongside structured data: they merely serve as additional evidence for structured knowledge extraction. In the *metaphactory* knowledge graph management platform, we use federated hybrid SPARQL queries for combining explicit information stated in the graph, implicit information from the associated embedding models, and information extracted using vector embeddings in a transparent way for the end user. In this paper we show how we integrated RDF data with vector space models to construct an augmented knowledge graph to be used in customer applications.

KEYWORDS

knowledge graph, word embeddings, graph embeddings, SPARQL federation

ACM Reference Format:
Andriy Nikolov, Peter Haase, Daniel M. Herzig, Johannes Trame, and Artem Kozlov. 2018. Combining RDF Graph Data and Embedding Models for an Augmented Knowledge Graph. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3184558.3191527

1 INTRODUCTION

Representing information from semantic web datasets and free text using vector embeddings provides a powerful tool helping to infer implicit relations between data entities. These models can be used directly to predict new links in a knowledge graph or serve as input data for machine learning algorithms which perform knowledge graph population from unstructured sources (e.g., free text). They provide an alternative representation view for knowledge graph data: continuous multi-dimensional vectors as opposed to a directed graph. For this reason, vector space embedding models are normally used for offline tasks separately from the actual data: they merely provide input to other algorithms, which in turn have their results materialized, stored, and exploited in the graph form.

There exist many use case scenarios where information provided by the embedding models serves as a valuable addition to the knowledge graph itself: e.g., to retrieve the most similar entities to provide suggestions to the user. It requires the ability to access and query the graph and the vector space models in a uniform way. Such an ability would allow providing the best available answer to given user queries: e.g., returning exact answers stored in the original graph in an explicit way as well as adding uncertain results inferred from an embedding model or extracted from external sources using machine learning. Moreover, different types of embedding models such as embeddings for entities and relations extracted from the graph and word2vec models learned from natural language text can complement each other to improve the performance on relevant knowledge graph population tasks [8].

The main motivation for this work comes from our experience with the *metaphactory* knowledge graph management platform[1], which is used in a variety of application domains (e.g., cultural heritage, life sciences, pharmaceutics, and IoT infrastructure). In this paper we describe our approach to enable combined usage of the original RDF graph data, implicit relations encoded by word and graph embedding models, and additional knowledge extracted with the help of embeddings by transparent querying using federated SPARQL queries. This allows the platform to support building end-user knowledge graph management applications that make use of both explicit RDF data and associated vector space models in a transparent way. We call such integrated expanded data source an *augmented knowledge graph*.

2 AUGMENTED KNOWLEDGE GRAPH CONSTRUCTION

Augmented knowledge graphs (Figure 1) includes three types of data sources:

- The original core knowledge graph expressed in RDF
- Embedding models describing entities and relations from the original knowledge graph and expressed as sets of vectors

[1] http://www.metaphactory.com/

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3191527

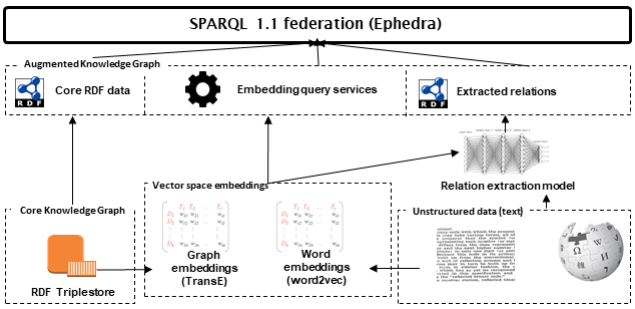

Figure 1: Augmented knowledge graph: components and access.

- Additional statements extracted from unstructured sources (e.g., free text) using the information from the knowledge graph and embedding models as evidence

The core knowledge graph represents the reference dataset consisting of two parts: the ontological schema and instance-level data. These data are stored in the triple store and accessed directly using the SPARQL 1.1 query language. Moreover, the core knowledge graph serves as a source of evidence for training machine learning models to extend it with additional data.

Graph embedding models aim at encoding information contained in a knowledge graph in a continuous vector space model that would preserve to the maximal extent information about the statements contained in the original graph and can be utilized to infer new links. Graph embedding models such as TransE [1], TransR [3], or HolE [6] produce embedding vectors for each entity and relation in the graph. Combining the vectors for two entities (knowledge graph instances) E_1 and E_2 and a relation R provides a degree of confidence that the knowledge graph contains a true statement $R(E_1, E_2)$. Close distance between two entity embedding vectors points to a degree of semantic similarity, i.e., that two entities participate in the same kinds of relations.

In addition to embeddings learned directly from the graph, *word embeddings* produced from a related text corpus (e.g., using a popular *word2vec* algorithm [5]) are utilized as well. Unlike the graph embeddings, the word2vec model contains vectors corresponding to the words of the natural language as opposed to ontological instances. This requires an additional step of mapping the instances from the knowledge graph to word2vec vectors. Such mapping can be realized in two ways:

- Learning the word2vec model from an already annotated corpus. If knowledge graph entities are explicitly referenced in the text (e.g., as in Wikipedia), the trained model will contain vector entries for entities alongside words. If these entities are contained in the core knowledge graph, the model can be used directly, otherwise an additional instance matching step is performed.
- Label-based mapping. For RDF instances not directly represented among pre-trained word embeddings, their labels can be used to position the entities in the word vector space.

Structured RDF graph model and corresponding embedding models complement each other. The former contains explicit and reliable statements about entities. Embedding models can be used to extract two additional kinds of statements:

- Statements involving relations contained in the graph schema. For example, if the graph contains a relation *literaryGenre*, which is present only for some instances of type *Book*, possible missing values for the property can be provided by the graph embedding model.
- "Fuzzy" relations not present in the schema. For instance, embedding models can be used to compute semantic similarity between instances. This allows posting requests such as "who is the most similar to Rembrandt?" although the similarity relation is not explicitly defined in the graph. Furthermore, vector space proximity can be used to compute non-standard aggregation functions: e.g., to find the most similar instance to a group of other instances.

Due to the amount of possible questions which can be posted to the vector space models, it makes sense to compute such information on demand rather than materialize as RDF and store inside the knowledge graph.

Finally, the graph itself as well as associated embedding models can be used as evidence to extract additional relevant statements from relevant unstructured sources: e.g., text and/or images.

3 COMBINING EMBEDDINGS FOR RELATION EXTRACTION

Trained vector space embeddings can be used as input data for information extraction algorithms to extend the knowledge graph using information from unstructured data sources (primarily, text). In particular, relation extraction from text is a long-studied research direction, which benefited in recent years from the development of network-based algorithms. Embedding vectors serve as input for convolutional or recurrent neural network algorithms that make a decision on whether a particular sentence or phrase describes a specific ontological relation between a pair of entities. Usually, such algorithms rely on the word embedding models learned from a text corpus. We extended a state-of-the-art model by combining the outputs learned from a text sequence with the embedding vectors learned from a knowledge graph. As the baseline for our approach, we adapted a bidirectional GRU network algorithm with sentence-level attention [2], which in turn is based on the combination of ideas described in [9] and [4].

One training example in the training setup represents a single sentence mentioning two entities. The input of the network is constructed by encoding each token in the sentence. Each input vector is generated by concatenating the word2vec embedding of the token and the position embedding (positions of the token in the sentence relative to the entities h and t). These vector representations of the tokens are fed to the neural network (Figure 2) consisting of several layers.

Bi-directional GRU layer. The layer includes two sets of GRU units [2] which process the incoming sequence in two different directions: forward and backward. The outputs of each set at each step constitute vectors $\overleftarrow{h}$ and $\overrightarrow{h}$, which are added to produce the

[2] https://github.com/thunlp/TensorFlow-NRE

combined output of the layer: $\boldsymbol{h}_i = \overleftarrow{h_i} \oplus \overrightarrow{h_i}$. The output of the layer is a matrix H formed by vectors $\boldsymbol{h}_1, \cdots, \boldsymbol{h}_n$.

Attention layer The word attention layer post-processes the output H of the bi-directional GRU layer to produce a single output vector. The word-level attention layer combines the outputs produced by the recurrent layer using a weighted sum and produces a single sentence embedding vector for each training example.

$$M = tanh(H)$$
$$\alpha = softmax(w^T M)$$
$$h^* = tanh(H\alpha^T),$$

where h^* is the output of the hidden word-level attention layer.

A further (optional) step involves applying sentence-level attention proposed by [4]. Although a pair of entities E_1 and E_2, for which a relation R holds, occurs in a sentence, it is not always relevant for extracting this relation: e.g., a sentence "Vladimir Putin visited St Petersburg in Feb 2017" cannot serve as a supporting evidence for a correct statement *bornIn(Vladimir Putin, St Petersburg)*. To address these, different training instances for the relation R themselves get different weights, further adjusting the output value h^*.

Word-based classification output The output of the word-based classification of the input sentence S is produced by making a linear transformation of h^* and applying the softmax function.

$$\hat{p}(y|S) = softmax(W^{(S)}h^* + b^{(S)})$$

$$\hat{y} = \arg\max_y(\hat{p}(y|S))$$

Combining with entity embedding vectors To improve the quality of the classification, we further extended the network to include the embedding vectors learned from the knowledge graph data as additional evidence. An additional layer combines the output vector produced by the word-based classification model with TransE vectors. The input of the layer is formed by concatenating the vectors $\{w^{out}, e_1, e_2, \boldsymbol{d}\}$, where w^{out} is formed by the output of the word-based classifier, e_1 and e_2 are TransE embedding vectors of entities E_1 and E_2, and the vector $\boldsymbol{d}$ contains the vector space distances for each relation R_i:

$$d_i = ||e_1 + r_i - e_2||$$

.

Results returned by the output classification layer are materialized as RDF triples and stored in a triple store.

4 EXPERIMENTS

We conducted experiments to investigate the added value of TransE graph embeddings using a subset of Wikidata (containing information about persons and their relations) and a text corpus containing Wikidata abstracts. For experiments, we re-trained the TransE embeddings to exclude relations contained in the test set. We compared the $F1$ measure obtained for the relation extraction task by the original model using only word2vec embeddings with the extended model utilizing both word2vec and TransE embeddings (Table 1).

While the TransE embeddings themselves do not achieve high accuracy in the link prediction task for the very precise factual information (F1 measure between 0.36 and 0.65), they provide added value in refining the output of the text extraction model leading to improved accuracy in all cases. Their main impact was in filtering out spurious candidate pairs.

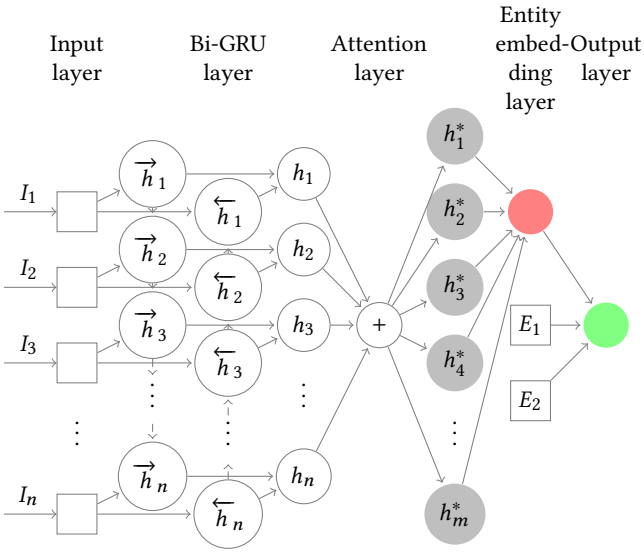

Figure 2: Neural network used for relation extraction using combined word2vec and TransE vectors.

Wikidata relation	Text + word2vec embeddings			Combined model		
	p	r	F1	p	r	F1
P22 (father)	0.90	0.90	0.90	0.91	0.93	**0.92**
P40 (child)	0.78	0.85	0.81	0.88	0.83	**0.85**
P26 (spouse)	0.89	0.75	0.81	0.89	0.77	**0.83**
P3373 (sibling)	0.85	0.78	0.81	0.92	0.78	**0.85**

Table 1: Precision, recall, and F1 measure for the relation extraction task obtained using only word embeddings vs a combination of word and graph embeddings.

RDF statements extracted from text, which do not have 100% precision are stored separately from the main knowledge graph in a special repository.

5 AUGMENTED KNOWLEDGE GRAPH QUERYING

Different components of the augmented knowledge graph are expressed in different formats: RDF triples and embedding vectors. In order to achieve seamless integration of graph and vector space models, these mutually complementary components must be queried in the same way. To this end, we extend the standard SPARQL 1.1 federation mechanism for hybrid queries. The Ephedra query federation architecture [7] allows querying external compute services as SPARQL 1.1 federation members. This is achieved by building wrappers that translate SPARQL 1.1 SERVICE clauses into service requests and then bind results returned by the service to the output variables.

Thus, in order to be able to query vector embedding models using SPARQL 1.1, the embedding wrapper service must transform information expressed in embedding vectors into RDF triples. As discussed in section 2, vector embeddings can be used to retrieve two kinds of statements: additional instantiations of the object

properties from the knowledge graph schema and similarity relations based on the proximity between instances in the vector space. The first kind of statements is trivial to obtain from the graph embeddings such as TransE: given two elements of a triple, one can calculate the expected embedding vector for the third one and retrieve the instances with the shortest distance to this expected one. For example, to retrieve the expected object value for a property R of a subject instance E_s, one would need to compute the position of the expected value using the corresponding embedding vectors r and e_1: $\hat{e}_o = e_1 + r$ and then return the instance E_o such that $e_o = \arg\min_i(||e_i - \hat{e}_o||)$.

To incorporate additional relations based on similarity, we defined an artificial predicate *similarTo*, which returns the most similar objects to a given subject instance: given E_s, return E_o such that $e_o = \arg\min_i(||e_i - e_s||)$. Additionally, similarity is defined as an aggregation function returning entities which are close to a group of other entities. For this, the service first computes a centroid vector $e_c = avg(e_k)$ for the set of input entities E_k and then returns E_o such that $e_o = \arg\min_i(||e_i - e_c||)$.

Unlike the TransE graph embedding model, embeddings trained from text using word2vec do not contain relation embedding vectors explicitly. To compute these, we need to involve information from the main knowledge graph. In order to compute an embedding vector r in the word2vec vector embedding space for an object property R, we first need to select all pairs (E_i^h, E_i^t) from the knowledge graph such that the relation $R(E_i^h, E_i^t)$ holds for them. Then, the embedding vector r for the relation R can be calculated as $r = avg(e_i^t - e_i^h)$.

In this way, we implemented the entity retrieval service that for a given pair (E_s, R) or (R, E_o) is able to return the most likely value for E_o or E_s respectively and exposed it as a REST API. For this service, we built an Ephedra wrapper which is able to transform a SPARQL 1.1 SERVICE clause into a REST API call to the entity retrieval service and then convert retrieved results into SPARQL variable bindings. The SPARQL service accepts the following kinds of input patterns:

- *:subjectURI :propertyURI ?objectVariable*
- *?subjectValue :propertyURI :objectURI*
- *?variable emb:limit "K"*, where "K" is a constant denoting the expected number of the most fitting results

With this approach, we are able to formulate SPARQL queries returning results over vector space models: e.g., "retrieve a list of entities most similar to *Rembrandt*" or "retrieve the most relevant theme of *War and Peace*". These results can be joined with information explicitly contained in the core knowledge graph as RDF using hybrid federated queries. Such queries can potentially include more than one SERVICE clause. For example, the following query retrieves 10 books most similar to "War and Peace" (according to the word2vec embedding model) and additionally retrieves information about the genres of these books, both stored explicitly in Wikidata and guessed using the TransE graph embeddings.

```
SELECT ?book ?genre WHERE {
  SERVICE metaphacts:wikidataWord2Vec {
     wd:Q161531 emb:similarTo ?book . # War and Peace
     ?book emb:limit "10" .
  }
  {
    ?book wdt:P136 ?genre . # Property:genre
  } UNION {
    SERVICE metaphacts:transE {
      ?book wdt:P136 ?genre .
      ?genre emb:limit "5" .
    }
  }
}
```

6 CONCLUSION AND OUTLOOK

In this paper we described an architecture enabling a unified access to the complementary information contained in the knowledge graph and pre-trained embedding vectors. Combining the alternative views over data helps to answer more complex queries over graph data including fuzzy information. Moreover, exploiting different types of vector embeddings for the task of knowledge graph population allows the performance of the relation extraction procedure to be improved.

In our future work we plan to explore two directions. First, we want to focus further on the use of combined embedding models for the task of link discovery with external datasets to enable smooth inclusion of additional datasets into the augmented knowledge graph. Second, we aim at exploiting the augmented knowledge graph model to assist the user with data authoring tasks: e.g, by providing intelligent autosuggestions and pre-filling the initial values in the data editing forms.

Acknowledgements

This work has been supported by the Eurostars project DIESEL (E!9367) and by the German BMWI Project GEISER (project no. 01MD16014).

REFERENCES

[1] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS 2013*. 2787–2795.
[2] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP 2014, Doha, Qatar*. 1724–1734.
[3] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *AAAI 2015*. 2181–2187.
[4] Yankai Lin, Shiqi Shen, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. 2016. Neural Relation Extraction with Selective Attention over Instances. In *ACL 2016*.
[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.
[6] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. 2016. Holographic Embeddings of Knowledge Graphs. In *AAAI 2016, Phoenix, Arizona, USA*. 1955–1961.
[7] Andriy Nikolov, Peter Haase, Johannes Trame, and Artem Kozlov. 2017. Ephedra: Efficiently Combining RDF Data and Services Using SPARQL Federation. In *KESW 2017*. 246–262.
[8] Steffen Thoma, Achim Rettinger, and Fabian Both. 2017. Towards Holistic Concept Representations: Embedding Relational Knowledge, Visual Attributes, and Distributional Word Semantics. In *ISWC 2017, Vienna, Austria*. 694–710.
[9] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. In *ACL 2016, Berlin, Germany*.