RDF2Vec Light – A Lightweight Approach for Knowledge Graph Embeddings

Jan Philipp Portisch[1,2] [0000−0001−5420−0663], Michael Hladik[2] [000−0002−2204−3138], and Heiko Paulheim[1] [0000−0003−4386−8195]

[1] Data and Web Science Group, University of Mannheim, Germany
{jan, heiko}@informatik.uni-mannheim.de
[2] SAP SE Product Engineering Financial Services, Walldorf, Germany
{jan.portisch, michael.hladik}@sap.com

Abstract Knowledge graph embedding approaches represent nodes and edges of graphs as mathematical vectors. Current approaches focus on embedding complete knowledge graphs, i.e. all nodes and edges. This leads to very high computational requirements on large graphs such as *DBpedia* or *Wikidata*. However, for most downstream application scenarios, only a small subset of concepts is of actual interest. In this paper, we present *RDF2Vec Light*, a lightweight embedding approach based on *RDF2Vec* which generates vectors for only a subset of entities. To that end, *RDF2Vec Light* only traverses and processes a subgraph of the knowledge graph. Our method allows the application of embeddings of very large knowledge graphs in scenarios where such embeddings were not possible before due to a significantly lower runtime and significantly reduced hardware requirements.

Keywords: RDF2Vec · knowledge graph embeddings · knowledge graphs · data mining · scalability · resource efficient embeddings

1 Introduction

Public knowledge graphs (KGs), such as *DBpedia* or *Wikidata*, provide deep background knowledge that can be exploited for downstream tasks such as question-answering or recommender systems [3]. *KG embeddings* (KGEs) represent vertices and, depending on the approach, also edges of a KG as numeric vectors. This representation is easily consumable by most algorithms and can be exploited in downstream tasks. Advantages of KGEs, once they have been trained, include simple applicability, fast run time, good performance on multiple tasks, and reusability in downstream applications. On the downside, KGEs produce very large models[3], and are very expensive to train and re-train in the case of evolving knowledge bases. For very large knowledge graphs, such as Wikidata, computing a complete embedding typically takes up to a day or longer [2].

In this paper, we address the scalability aspect of knowledge graph embeddings: Our novel approach, *RDF2Vec Light*, allows to train partial, task-specific models with

[3] For example, the 200 dimensional DBpedia RDF2Vec embedding model available at *KGvec2go* [5] requires more than 10GB of disk storage.

```
Data: G = (V,E): RDF Graph, V_I: vertices of interest, d: walk depth, n: number of walks
Result: W_G: Set of walks
1  W_G = ∅
2  for vertex v ∈ V_I do
3      for 1 to n do
4          add v to w
5          pred = getIngoingEdges(v)
6          succ = getOutgoingEdges(v)
7          while w.length() < d do
8              cand = pred ∪ succ
9              elem = pickRandomElementFrom(cand)
10             if elem ∈ pred then
11                 add elem at the beginning of w
12                 pred = getIngoingEdges(elem)
13             end
14             else
15                 add elem at the end of w
16                 succ = getOutgoingEdges(elem)
17             end
18         end
19         add w to W_G
20     end
21 end
```

Algorithm 1: Walk generation algorithm for *RDF2Vec Light*.

only a fraction of the computation requirements compared to other embedding approaches, while retaining a high performance on multiple tasks. The resulting models contain *only vectors for entities of interest*. Internally, *RDF2Vec Light* only traverses a subset of the underlying knowledge graph which leads to processing times that are much shorter than the original RDF2Vec approach which always processes an entire knowledge graph. Moreover, the resulting models are much smaller.[4]

2 RDF2Vec Light

RDF2Vec is based on performing random walks on a graph [6]. The underlying idea of *RDF2Vec Light* embeddings is to generate only local walks for entities of interest given a predefined task. After the walk generation has been completed, the training of vectors can be performed like in the original approach.

Rather than *starting* random walks at all entities of interest, it is randomly decided for each depth-iteration whether to go backwards, i.e. to one of the node's predecessors, or forwards, i.e. to the node's successors (line 9 of Algorithm 1). As a result, the entities of interest can be at the beginning, at the end, or in the middle of a walk which better captures the context of the entity. This generation process is

[4] *RDF2Vec Light* models are typically only a few kilobytes in size, compared to multiple gigabytes of disk space required to persist classic embedding models.

Table 1. Classification (accuracy) and regression (RMSE) results with RDF2Vec Classic and *RDF2Vec Light*. The best classic and light results are highlighted.

	Cities		Movies		Albums		AAUP		Forbes	
Strategy	SVM	LR	SVM	LR	SVM	LR	SVM	LR	SVM	LR
Light_500_4_CBOW_50	52.56	19.23	73.31	19.75	72.44	12.52	61.93	68.35	60.79	34.62
Classic_500_4_CBOW_50	49.36	16.95	55.25	22.77	51.70	14.06	55.33	70.59	57.28	36.64
Light_500_4_CBOW_100	71.78	21.16	73.50	19.90	72.43	12.35	63.21	**65.85**	60.81	34.96
Classic_500_4_CBOW_100	49.36	22.15	58.21	22.94	57.44	14.17	55.00	73.33	57.36	42.32
Light_500_4_CBOW_200	71.61	54.86	73.93	19.60	73.34	12.50	61.74	67.65	59.11	35.97
Classic_500_4_CBOW_200	49.36	99.73	58.79	23.54	59.18	14.24	56.83	80.29	57.57	45.76
Light_500_4_SG_50	**75.90**	**19.39**	74.15	19.34	76.49	12.00	**65.46**	67.66	**61.56**	34.58
Classic_500_4_SG_50	**80.57**	**12.95**	72.81	19.89	76.42	11.80	**68.04**	64.85	61.08	**34.89**
Light_500_4_SG_100	73.99	20.89	**74.89**	**19.21**	**76.98**	**11.89**	64.54	66.59	61.38	**34.48**
Classic_500_4_SG_100	79.01	15.26	72.72	**19.61**	**76.51**	**11.57**	64.72	**65.50**	60.42	35.26
Light_500_4_SG_200	73.81	44.38	74.58	19.45	76.35	12.16	62.83	70.13	60.26	36.73
Classic_500_4_SG_200	77.06	28.34	**73.85**	19.71	75.66	11.92	66.74	67.96	**61.82**	36.93

described in Algorithm 1. The RDF2Vec method as well as the *RDF2Vec Light* extension have been implemented in Java and Python.[5] The implementation can handle various RDF formats such as n-triples, RDF/XML, Turtle, or HDT [1]. In addition, a REST API has been implemented and is provided on `http://www.kgvec2go.org`.

3 Evaluation

In order to evaluate the approach presented in this paper, the classification and regression experiments, as well as the entity and document relatedness experiments of Ristoski et al. [6] have been repeated. The evaluation follows the setup defined in [4].

Six classic and six light embedding spaces have been trained each with the following parameters held constant: $window\ size = 5$, $negative\ samples = 25$. The parameters that were changed are the generation mode ($cbow$ and sg) as well as the dimension of the embedding space (50, 100, 200). All walks have been generated with 500 walks per entity and a depth of 4. For the evaluation, the DBpedia knowledge graph as of 2016-10[6] has been used.

For the classification and regression tasks, we follow the same setup as in the original RDF2vec paper [7]: For the classification tasks, four classifiers have been evaluated: *Naïve Bayes*, *C4.5* (decision tree algorithm), *k-NN* with $k = 3$, and *Support Vector Machines (SVM)* with $C \in \{10^{-3}, 10^{-2}, 0.1, 1, 10, 10^2, 10^3\}$ where the best C is chosen. A 10-fold cross validation has been used to calculate the performance statistics. For the regression tasks, three approaches have been evaluated: *linear regression*, *k-NN*, and *M5rules*. For the sake of brevity, we only report results for the best performing approaches (SVM and LR).[7]

[5] `https://github.com/dwslab/jRDF2Vec`

[6] `https://wiki.dbpedia.org/downloads-2016-10`

[7] The complete result tables are available at `http://www.rdf2vec.org/rdf2vec_light`

Table 2. Results on the document relatedness task (LP50), reporting the harmonic mean of Pearson correlation and Spearman rank correlation, and on entity relatedness (KORE), using cosine similarity. The best value of each comparison group is highlighted in bold. The overall best value is additionally underlined.

Strategy	LP50	KORE	Strategy	LP50	KORE
Light_500_4_CBOW_50	0.3871	0.3343	Light_500_4_SG_50	0.3421	0.4767
Classic_500_4_CBOW_50	0.2235	0.2982	Classic_500_4_SG_50	0.4400	0.5068
Light_500_4_CBOW_100	0.3741	0.3179	Light_500_4_SG_100	0.3310	0.4281
Classic_500_4_CBOW_100	0.2291	0.3028	Classic_500_4_SG_100	0.4507	0.5288
Light_500_4_CBOW_200	0.3809	0.3474	Light_500_4_SG_200	0.3278	0.4045
Classic_500_4_CBOW_200	0.2374	0.3178	Classic_500_4_SG_200	0.4371	0.5348

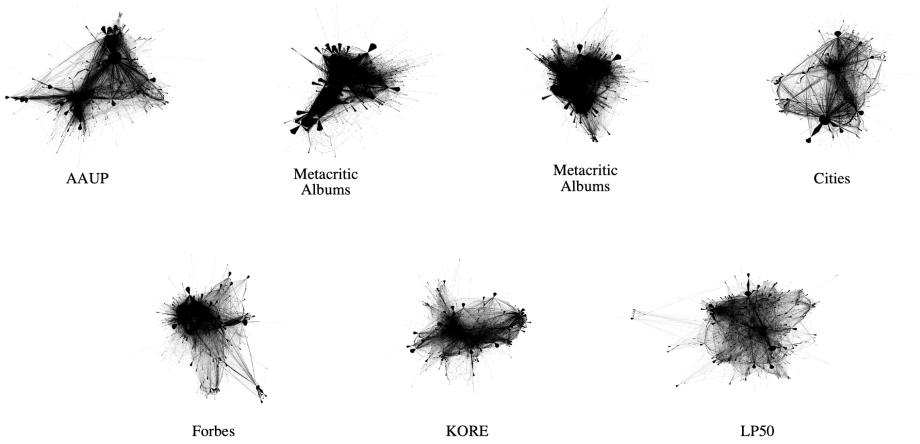

Figure 1. Depiction of the graphs which were assembled using the generated walks.

In the results tables, *strategy* refers to the configuration with which the embeddings have been obtained. The structure can be read as follows: *<mode>_<number_of_walks_per_entity>_<walk_depth>_<training_mode>_<dimension>* where *mode* is either *Light* or *Classic*. For example, *Light_500_4_CBOW_100* refers to *RDF2Vec Light* embeddings with 500 walks per entity, a walk depth of 4, CBOW configuration, and an embedding space dimensionality of 100.

For classification and regression, we can observe that except for the cities dataset, the difference between the two approaches is rather marginal. For entity and document relatedness, the results are less conclusive. Here, we see that the RDF2vec light approach is en par with the classic approach for the CBOW variant, but the results are reversed when looking at the SG variant, which also yields the best results globally.

In order to analyze those results more deeply, and to distinguish the cases where RDF2vec Light is en par with classic RDF2vec from those where it is clearly inferior, we looked at the linkage degree of the entities at hand, as well the homogeneity of the entities of interest.

For the linkage degree, we can observe that a higher degree of the entities of interest leads to a worse performance of RDF2vec Light. This can be seen in the inferior performance of RDF2vec Light for the cities datasets in classification and regression. Cities are among the most strongly interlinked entities in DBpedia [3]. At the same time, the document and entity similarity datasets contain a larger number of strongly interlinked head entities.

While for classification and regression problems, the set of entities is rather homogeneous (i.e., all are cities, albums, etc.), the homogeneity is lower for the document and entity relatedness, where the entities of interest are scattered across many classes. Both degree and homogeneity contribute to the density of the considered subgraphs, as depicted in Fig. 1. From the plots, we can observe a correlation of *RDF2Vec Light* performance and the density of the graph spanned by the random walks – the more dense the graph (i.e., the less head entities there are and the more homogeneous the entity set at hand), the better the performance of *RDF2Vec Light*.

The runtime of *RDF2Vec Light* is linear in the number of entities of interest. On commodity hardware, the runtime is roughly 1 minute per 10 nodes. In comparison, training RDF2Vec on the full DBpedia graph takes a few days.

4 Conclusion and Outlook

In this paper, we presented *RDF2Vec Light*, an approach for learning latent representations of knowledge graph entities that requires only a fraction of the computing power compared to other embedding approaches. Rather than embedding the whole knowledge graph, *RDF2Vec Light* trains vectors for only few entities of interest and their context. For this approach, the walk generation algorithm has been adapted to better represent the context of the entities. Our experiments show that the results achieved with *RDF2Vec Light* are comparable to those obtained with the standard RDF2Vec, while requiring only a fraction of the runtime.

References

1. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). Web Semantics **19**, 22–41 (2013)
2. Han, X., Cao, S., Xin, L., Lin, Y., Liu, Z., Sun, M., Li, J.: OpenKE: An open toolkit for knowledge embedding. In: Proceedings of EMNLP (2018)
3. Heist, N., Hertling, S., Ringler, D., Paulheim, H.: Knowledge graphs on the web–an overview. In: Tiddi, I., Lécué, F., Hitzler, P. (eds.) Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges, pp. 3–22. IOS Press (2020)
4. Pellegrino, M.A., Cochez, M., Garofalo, M., Ristoski, P.: A configurable evaluation framework for node embedding techniques. In: The Semantic Web: ESWC 2019 Satellite Events. pp. 156–160 (2019)
5. Portisch, J., Hladik, M., Paulheim, H.: KGvec2go - knowledge graph embeddings as a service. LREC (2020)
6. Ristoski, P., Rosati, J., Noia, T.D., Leone, R.D., Paulheim, H.: RDF2Vec: RDF graph embeddings and their applications. Semantic Web **10**(4), 721–752 (2019)
7. Ristoski, P., de Vries, G.K.D., Paulheim, H.: A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In: ISWC. pp. 186–194 (2016)