RDF2vec Embeddings for Updateable Knowledge Graphs – Reuse, don't Retrain!*

Sang Hyu Hahn and Heiko Paulheim[0000−0003−4386−8195]

University of Mannheim, Data and Web Science Group, Germany
heiko.paulheim@uni-mannheim.de

Abstract. Most Knowledge Graph Embeddings, like RDF2vec, are designed to be trained on a fixed knowledge graph (KG). When that KG is updated, they usually need to be retrained from scratch, which takes quite a bit of time. In this paper, we introduce a method of incrementally updating an RDF2vec embedding instead of retraining it. We conduct an experiment using different snapshots of DBpedia, demonstrating that this is a competitive, yet faster method to obtain embedding vectors of an updated knowledge graph, which sometimes even yields better results than retraining from scratch.

Keywords: Updateable Knowledge Graphs · Embeddings · RDF2vec

1 Introduction

Knowledge Graph Embeddings (KGEs) are widely used, e.g., for link prediction in knowledge graphs, or to provide dense representations for other downstream tasks, such as node classification or recommender systems [8]. The vast majority of KGE approaches assume a *static* knowledge graph, i.e., upon an update of that graph, the embeddings have to be re-trained from scratch [1]. At the same time, training KGEs is usually a time and resource consuming process.

In this paper, we consider a widely used KGE technique, i.e., RDF2vec [5], and show how it can be adapted so that existing KGEs can be updated rather than re-trained. We look at four different snapshots of DBpedia [4], and compare embedding models which are trained on an earlier version and updated to a newer one to those freshly trained on the new version. We show that the results with updated KGEs on downstream tasks are en par or even superior, at much lower training efforts. Other than approaches like OUKE [2] and DKGE [9], it can also update embeddings of entities without changes in their direct context, and also supports the addition of new relations, not only new entities.

2 Approach

Our approach builds upon the capability of word2vec to resume training based on new sentences, updating both vectors for existing words, as well as learning

* Acknowledgement: Supported by the state of Baden-Württemberg through bwHPC.

```
Data: W_init an initial set of walks, d: walk depth
Result: W_final: Set of walks
for walk w ∈ W_init do
    while w.length() < d do
        if (random()<0.5) then
            edge = pickRandomFrom(getIngoingEdges(w.first()))
            add edge at beginning of w
        end
        else
            edge = pickRandomFrom(getOutgoingEdges(w.last()))
            add edge at end of w
        end
    end
end
add w to W_final
```

Algorithm 1: Overall walk generation algorithm for generating new walks

vectors for new words.[1] To that end, new walks reflecting the updates in a KG have to be extracted. We pursue three strategies to extract new walks for an updated KG.

All of them are inspired by the algorithm of RDF2vec Light, which extracts walks for specific entities [7]. With $G = (V, E)$ being the KG for which an embedding has already been trained, and $G' = (V', E')$ being the new version of the KG, the three strategies all call algorithm 1 with different sets of walks W_{init}:

Entity-based computes a set of n walks for each new entity. Calls algorithm 1 with W_{init} being the set of (0-hop) walks constituted by $V' \setminus V$.

Edge-based computes a set of n walks for each new edge. Calls algorithm 1 with W_{init} being the set of 1-hop walks constituted by $E' \setminus E$.

Combined computes a set of n walks for each new entity, and for each new edge connecting two existing entities. Calls W_{init} for the union of 0-hop walks constituted by $V' \setminus V$, and 1-hop walks constituted by $\{(e_1, r, e_2) \in E' \setminus E \mid e_1 \notin V' \setminus V \wedge e_2 \notin V' \setminus V\}$.

The last approach is faster than the edge-based approach for new entities, since it generates n walks per new entity in G', while the edge-based approach would generate $n \cdot d$ walks, where d is the degree of the new entity.

3 Experiments

In our experiments, we use four snapshots of DBpedia[2]. All three update scenarios start from the 2019-09 version (4.3M entities, 11.8M triples) and update the

[1] `https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#online-training-resuming-training`

[2] `https://databus.dbpedia.org/dbpedia/mappings/mappingbased-objects`

Table 1. DBpedia versions used in the experiments

Version	#Entities	#Triples	#relations
2019-09	4,256,911	11,799,129	621
2020-12	7,266,101	21,384,769	635
2021-12	7,347,010	22,072,275	634
2022-12	7,958,883	22,791,171	633

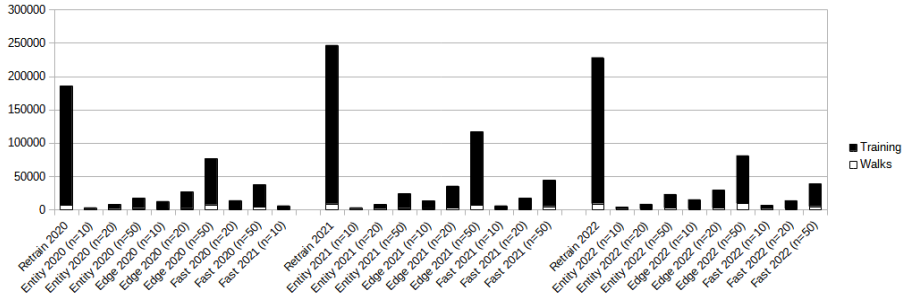

Fig. 1. Depiction of runtimes, split by walk generation (white) and model training/update (black).

embeddings to the 2020-12 (7.3M entities, 21.3M triples, 2021-12 (7.3M entities, 22.0M triples), and 2022-12 (8.0M entities, 22.8M triples) version, respectively. We run experiments with 10, 20, and 50 walks per new entity and/or edge. Statistics about the datasets are depicted in table 1.

We use the gEval benchmark suite [6], which defines different tasks for evaluating and comparing KGEs, and the RDF2vec parameters reported to work best in past works[3].[4] The results are shown in tables 2 and 3. The runtimes[5] for creating the KGEs are depicted in Fig. 1.

4 Summary and Limitations

The results show that the update mechanism can update RDF2vec representations for new versions of a KG, in most of the cases performing at least as good or even better than fully retrained ones, but with less consumption of computational resources. While the fact that updated embeddings may even outperform those retrained from scratch might come as a surprise, it can be explained by the fact that an updated KGE can reflect knowledge encoded in *both* versions of the KG, not just one, and, hence, may even encode more information.

Currently, the approach only supports *addition* of entities and edges. The case where entites and edges are *removed* is not directly addressed by our approach.

[3] Skip-gram, 5 epochs, 200 dimensions, window size=5, depth=8
[4] Code to reproduce experiments: `https://github.com/sanghyu/RDF2vec-Update`
[5] Intel Xeon Gold 6230 (2.1 Ghz), using 8 cores with 300GB RAM

Table 2. Classification (C, accuracy) and regression (R, RMSE) results. The best performing results per DBpedia version are marked in bold; all results outperforming retraining from scratch are marked in grey.

	AAUP		Cities		Forbes		Albums		Movies	
Strategy	C	R	C	R	C	R	C	L	C	R
Baseline 2019	.670	71.965	.771	18.112	.568	40.738	.632	15.167	.733	22.525
Retrain 2020	.708	64.510	.776	14.340	.597	36.414	.659	15.698	.747	19.752
Entity 2020 (n=10)	.687	66.347	.789	14.668	.602	36.483	.682	15.163	.705	21.882
Entity 2020 (n=20)	**.713**	64.154	.772	14.112	.610	37.372	.726	14.154	.726	21.056
Entity 2020 (n=50)	.704	63.464	.757	13.869	.610	37.803	.742	14.369	.731	20.355
Edge 2020 (n=10)	.708	**62.061**	.770	12.595	**.625**	37.345	.760	13.338	**.756**	19.587
Edge 2020 (n=20)	.709	63.925	.791	12.238	.611	36.680	**.776**	13.526	.750	19.403
Edge 2020 (n=50)	.701	64.484	.778	**12.035**	.609	**35.941**	.757	13.368	.750	**19.148**
Comb. 2020 (n=10)	.704	62.572	**.783**	13.863	.621	37.083	.748	13.799	.733	21.143
Comb. 2020 (n=20)	.699	62.586	.778	12.832	.616	36.245	.764	13.158	.738	20.514
Comb. 2020 (n=50)	.693	63.594	.763	12.732	.617	36.414	.773	**12.691**	**.756**	19.748
Retrain 2021	**.715**	65.680	.776	14.255	.617	37.513	.649	14.927	.738	20.219
Entity 2021 (n=10)	.689	67.052	.794	14.137	.602	37.034	.680	15.413	.703	21.993
Entity 2021 (n=20)	.691	65.430	.782	13.931	.604	37.244	.717	14.359	.717	21.220
Entity 2021 (n=50)	.707	63.956	.780	14.140	.609	37.323	.727	14.536	.723	20.553
Edge 2021 (n=10)	.708	**63.069**	.785	13.937	.618	37.220	.752	13.664	**.753**	19.606
Edge 2021 (n=20)	.710	63.659	.786	13.719	.606	37.292	.772	13.198	.746	19.539
Edge 2021 (n=50)	.693	64.388	.787	13.594	.615	**36.057**	**.783**	**12.734**	.747	**19.481**
Comb. 2021 (n=10)	.699	65.351	**.807**	13.943	**.625**	37.597	.752	13.899	.732	21.237
Comb. 2021 (n=20)	.709	64.583	.785	**13.469**	.619	36.580	.770	13.198	.730	20.737
Comb. 2021 (n=50)	**.715**	64.684	.790	13.957	.621	36.263	.771	12.825	.743	19.964
Retrain 2022	.674	66.313	.773	16.639	.617	37.165	.691	14.830	.731	19.808
Entity 2022 (n=10)	.687	66.347	.789	14.668	.602	**36.483**	.682	15.163	.705	21.882
Entity 2022 (n=20)	.704	64.619	.786	14.221	.606	37.160	.727	14.402	.725	21.186
Entity 2022 (n=50)	.703	64.652	.781	13.611	.605	37.347	.722	14.202	.729	20.655
Edge 2022 (n=10)	**.707**	**62.957**	.785	13.043	**.619**	37.235	.742	13.664	**.755**	19.596
Edge 2022 (n=20)	.698	63.931	.783	13.113	.618	37.820	.748	13.284	.754	19.349
Edge 2022 (n=50)	.696	65.469	**.806**	**12.517**	.610	36.866	**.768**	13.256	.749	**19.289**
Comb. 2022 (n=10)	.696	64.327	.777	13.426	.618	37.215	.741	13.850	.729	21.033
Comb. 2022 (n=20)	.701	64.249	.779	13.437	.618	37.272	.765	13.151	.734	20.248
Comb. 2022 (n=50)	.702	64.351	.783	13.133	.616	36.888	.760	**12.954**	.744	20.248

For those cases, a combination with techniques that forget and reconstruct embeddings, as suggested in [3], might be useful.

The fact that, in many cases, using updated KGEs even outperforms using KGEs fully retrained from scratch yields the interesting question whether this should be considered a general training paradigm for RDF2vec on versioned knowledge graphs, as it indicates that incrementally training those embeddings might yield better embeddings than only training on the version for which the embedding is needed.

References

1. Biswas, R., Kaffee, L.A., Cochez, M., Dumbrava, S., Jendal, T.E., Lissandrini, M., Lopez, V., Mencía, E.L., Paulheim, H., Sack, H., et al.: Knowledge graph embeddings: open challenges and opportunities. Transactions on Graph Data and Knowledge **1**(1), 4–1 (2023)
2. Fei, L., Wu, T., Khan, A.: Online updates of knowledge graph embedding. In: Complex Networks & Their Applications. pp. 523–535. Springer (2022)
3. Krause, F.: Dynamic knowledge graph embeddings via local embedding reconstructions. In: European Semantic Web Conference. pp. 215–223. Springer (2022)

Table 3. Clustering (accuracy), Semantic Analogies (accuracy), Document Similarity (Kendall's Tau), and Entity Similarity (Harmonic Mean) results. The best performing results per DBpedia version are marked in bold; all results outperforming retraining from scratch are marked in grey.

	Clustering				Semantic Analogies				Ent.	Doc.
Strategy	cit/cou (2k)	cit/cou	5 cls.	teams	cap-cou	cap-cou (all)	curr	cit-stat	Rel.	Sim.
Baseline 2019	.628	.593	.522	.862	.352	.263	.387	.356	.457	.293
Retrain 2020	.809	.590	.787	.502	.755	.677	.418	.336	**.513**	.346
Entity 2020 (n=10)	.730	.821	.781	**.864**	**.879**	.816	.299	**.485**	.430	.424
Entity 2020 (n=20)	.729	**.843**	.784	.781	.761	**.822**	.399	.446	.451	.405
Entity 2020 (n=50)	.733	.671	.756	.779	.715	.803	.491	.383	.486	.396
Edge 2020 (n=10)	.732	.741	.779	.783	.662	.783	.478	.334	.489	.414
Edge 2020 (n=20)	.756	.708	.782	.782	.642	.749	.510	.267	.508	.423
Edge 2020 (n=50)	**.828**	.636	**.790**	.778	.626	.725	**.528**	.198	.512	.414
Comb. 2020 (n=10)	.757	.739	.681	.862	.773	.789	.382	.469	.474	**.427**
Comb. 2020 (n=20)	.776	.708	.751	.783	.820	.788	.469	.319	.696	.419
Comb. 2020 (n=50)	.819	.663	.765	.780	.810	.793	.519	.350	.509	.420
Retrain 2021	.825	.633	**.772**	.490	.739	.683	.461	.347	**.547**	.425
Entity 2021 (n=10)	.748	**.823**	.719	.858	**.836**	.799	.363	**.466**	.426	.438
Entity 2021 (n=20)	.747	.726	.754	.782	.781	**.805**	.423	.409	.457	.430
Entity 2021 (n=50)	.785	.675	.746	.780	.670	.756	.464	.317	.473	.410
Edge 2021 (n=10)	.774	.736	.756	.787	.674	.740	.462	.218	.504	.444
Edge 2021 (n=20)	.803	.702	.763	.783	.652	.722	.479	.178	.506	.454
Edge 2021 (n=50)	**.864**	.655	.759	.777	.634	.668	**.488**	.140	.506	**.464**
Comb. 2021 (n=10)	.788	.735	.660	**.860**	.767	.764	.452	.363	.489	.451
Comb. 2021 (n=20)	.834	.706	.756	.820	.773	.761	.483	.309	.495	.463
Comb. 2021 (n=50)	.848	.655	.762	.780	.804	.683	.461	.347	.510	**.464**
Retrain 2022	.828	.653	.774	.476	.725	.635	.384	.343	**.532**	.392
Entity 2022 (n=10)	.730	.821	.781	**.864**	**.879**	**.816**	.299	**.485**	.430	.424
Entity 2022 (n=20)	.769	**.833**	.775	.804	.800	.811	.359	.438	.460	.422
Entity 2022 (n=50)	.741	.672	.776	.785	.739	.749	.428	.352	.494	.419
Edge 2022 (n=10)	.778	.736	.780	.794	.709	.743	.428	.295	.515	.433
Edge 2022 (n=20)	.797	.695	.768	.793	.682	.701	.431	.219	.510	.444
Edge 2022 (n=50)	**.849**	.641	**.783**	.780	.702	.692	**.448**	.165	.507	**.461**
Comb. 2022 (n=10)	.731	.738	.756	.860	.848	.801	.383	.435	.502	.429
Comb. 2022 (n=20)	.790	.712	.763	.792	.828	.769	.425	.377	.503	.422
Comb. 2022 (n=50)	.837	.645	.782	.775	.812	.746	.443	.316	.572	.442

4. Lehmann, J., et al.: Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic web **6**(2), 167–195 (2015)
5. Paulheim, H., Ristoski, P., Portisch, J.: Embedding Knowledge Graphs with RDF2vec. Springer Nature (2023)
6. Pellegrino, M.A., et al.: Geval: a modular and extensible evaluation framework for graph embedding techniques. In: ESWC. pp. 565–582. Springer (2020)
7. Portisch, J., Hladik, M., Paulheim, H.: Rdf2vec light–a lightweight approach for knowledge graph embeddings. arXiv preprint arXiv:2009.07659 (2020)
8. Portisch, J. et al.: Knowledge graph embedding for data mining vs. knowledge graph embedding for link prediction–two sides of the same coin? Semantic Web (2022)
9. Wu, T., Khan, A., Yong, M., Qi, G., Wang, M.: Efficiently embedding dynamic knowledge graphs. Knowledge-Based Systems **250**, 109124 (2022)