Towards Binarization of Knowledge Graph Embeddings for Node Classification

Vitor Faria de Souza, Heiko Paulheim**

University of Mannheim, Data and Web Science Group, Germany

Abstract

Knowledge Graph Embeddings (KGEs) are dense representations of entities and relations of a knowledge graph (KG) in a continuous vector space. In this paper, we present a preliminary analysis showing that KGEs can be substantially compressed by using only binary instead of continuous features, replacing costly floating point storage by a more space-efficient bitwise storage, while retaining the representational power.

Keywords

Knowledge Graph Embedding, Node Classification, Binarization, Compression

1. Introduction

Knowledge Graph Embeddings (KGEs) are dense representations of entities and relations in knowledge graphs, where each entity and relation is represented by a vector in a continuous vector space. They are used for various in knowledge graph tasks, such as link prediction, entity classification, for helping in knowledge-graph related tasks such as entity linking and disambiguation, but also as knowledge representations in other downstream tasks, such as recommender systems. [1, 2]

While the scalability of KGE methods has been identified as a challenge [3, 4], most papers looking into scalability take the angle of scaling up the *training process*, i.e., being able to learn a representation for a (large) knowledge graph with reasonable time and memory usage. What has been overlooked so far is the *size of the resulting embedding*, which is also an important consideration for downstream tasks. Using KGEs in downstream applications requires loading and processing those within the application, which may be hindered by the sheer size of the embedding, especially when computing resources are limited.

To illustrate the issue of KGE size, we use the well-known knowledge graph DBpedia [5], whose latest release contain 7.62 million entities.[1] Both the most common implementation of word2vec [6] in the gensim library [7], which is used by RDF2vec [8], as well as the PyTorch library [9], which underlies embedding frameworks like DGL-KE [4] and PyKEEN [10], use 32

Posters, Demos, and Industry Tracks at ISWC 2024, November 13–15, 2024, Baltimore, USA

*Corresponding author.

heiko.paulheim@uni-mannheim.de (H. Paulheim)

http://www.heikopaulheim.com/ (H. Paulheim)

0000-0003-4386-8195 (H. Paulheim)

[1] https://www.dbpedia.org/blog/dbpedia-snapshot-2022-12-release/

bit floating point values for representing embeddings by default.[2,3] Embedding DBpedia with 200 dimensions would thus yield a 6GB large KGE model.[4]

In this paper, we propose to use *binary* instead of *continuous* embeddings, i.e., embeddings that only use the values 0 and 1 for each dimension. By doing so, the size of embeddings can be reduced by a factor of 32 (using only 1 bit per entity and dimension instead of 32). In the example above, we could thus store a 200 dimensional binary embedding using less than 200MB.

The proposed approach is not limited to a single KGE method. Instead, we propose to binarize an already trained KGE. In experiments with the DLCC node classification benchmark, eleven different embedding methods, and different binarization methods, we show that the loss in accuracy of using binary embeddings is rather small, but allows for a drastic reduction of storage size. Especially for embeddings that are already computed for popular large-scale knowledge graphs [11], binarization can reduce the data volume and downstream processing load.

2. Approach

In this paper, we utilize two approaches for binarizing knowledge graph embeddings: a simple dimension-based baseline, and an autoencoder-based approach adapted from [12].

2.1. Baseline – Dimension-wise Binarization

Given that $vec(e)$ is the embedding vector of an entity e, we first compute the average vector $\overline{vec}$ as

$$\overline{vec}_i = \underset{\text{all entities e}}{avg} vec(e)_i, \tag{1}$$

where $vec(e)_i$ is the i-th element of $vec(e)$. With that average vector, we can obtain a binary vector $bvec$ from $vec(e)$ as

$$bvec_i(e) = \begin{cases} 0 & vec_i(e) < \overline{vec}_i \\ 1 & vec_i(e) \geq \overline{vec}_i \end{cases} \tag{2}$$

In other words: every floating point value lower than the average for that dimension is represented by a 0, all others are represented by a 1.

2.2. Autoencoder-based Binarization

Autoencoders are, in their simplest form, three-layer neural networks trained for dimensionality reduction [13]. Between an input and an output layer, they have a (smaller) code layer. They are trained by presenting instances of a dataset both as input and output (i.e., the input and output being identical), and thereby learning to minimize the reconstruction error.

[12] propose to use a heavyside step function as an activation function $\sigma_E : \mathbb{R} \rightarrow \{0, 1\}$, which only outputs the values 0 and 1. Specifically, they use of the Heaviside step function h,

[2] https://groups.google.com/g/gensim/c/JSSenT7Hhlc/m/FTEynSwmAQAJ
[3] https://pytorch.org/docs/stable/notes/numerical_accuracy.html
[4] 200 dimensions × 7,620,000 entities × 32 Bit

Table 1
Average file sizes and compression factors relative to original-200 of the 33 .VEC files and 55 .TXT vector files, after entity filtering.

Embedding	Compact .VEC files		Full vector .TXT files	
	Average file size (MB)	Average compression factor	Average file size (MB)	Average compression factor
original-200	-	-	1607.5	1
avgbin-200	-	-	174.5	9
auto-128	32.9	**49**	112.6	**14**
auto-256	47.8	34	207.4	8
auto-512	78.3	21	396.6	4

defined as

$$h(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3}$$

With that, the output of $E(x)$ is a binary vector. For further optimizing computations, they propose to align the dimensionality of the code layer to CPU register widths, such as 64, 128, and 256 bits. In this paper, we use their implementation[5] for binarizing knowledge graph embeddings.

3. Evaluation

In order to evaluate the quality of binarized embeddings, we use the DBpedia portion of DLCC [14]. For our experiments, we use the DBpedia embeddings also used in the paper [1], which are available online[6]. As embedding methods, we use four flavors of RDF2Vec [15, 16] and two flavors (Norms L1 and L2) of TransE [17] embeddings are exploited, as well as ComplEx [18], DistMult [19], RESCAL [20], RotatE [21], and TransR [22].

Table 1 shows the average file sizes achieved with the different binarization methods. The standard storage method uses an uncompressed TXT variant (i.e., storing the value ”0” or ”1” in one byte), which can yield a compression factor up to 14. The .VEC format, which is supported by the autoencoder implementation used in this paper [12], actually stores each eight single embedding values in one byte (i.e., multiple embedding dimensions are stored in one long integer). With this method, a compression factor of almost 50 can be achieved, leading to an embedding originally stored in 1.6 GB now only occupying 33 MB.

DLCC has three sizes of problems for each task; here, we only present the results for largest problem size (5,000 examples per task). The full results are available online[7]; the results for the smaller groups (500 and 50 examples, resp.) are comparable to those on the largest task group.

Table 2 shows the results on the DLCC entity classification dataset. The average losses in accuracy are not very large, with the the baseline performing better than the smaller autoencoders. This makes us assume that the original embeddings capture the required information for the tasks at hand already at a very coarse level, i.e., it is sufficient to know that a vector value for an entity at a particulary dimension is high or low, but it is not necessary to know *how* low it actually is.

[5]https://github.com/tca19/near-lossless-binarization
[6]https://data.dws.informatik.uni-mannheim.de/kgvec2go/dbpedia/2021-09/
[7]https://github.com/vitor-faria/kgembeddings-binarization

Table 2
Results on the DLCC gold standard (5,000 examples only, macro average accuracy and relative difference to results with the original embedding)

	Original	Baseline	Δ	Auto-128	Δ	Auto-256	Δ	Auto-512	Δ	Avg. Δ
RDF2vec CBOW	0.745	0.729	-0.021	0.689	-0.075	0.702	-0.057	0.707	-0.050	-0.051
RDF2vec CBOW OA	0.841	0.831	-0.013	0.805	-0.043	0.807	-0.041	0.818	-0.028	-0.031
RDF2vec SG	0.857	0.831	-0.031	0.818	-0.046	0.835	-0.026	0.844	-0.015	-0.030
RDF2veC SG OA	0.881	0.847	-0.038	0.829	-0.059	0.841	-0.046	0.858	-0.027	-0.042
ComplEx	0.833	0.752	-0.097	0.757	-0.091	0.777	-0.067	0.781	-0.063	-0.080
DistMult	0.825	0.728	-0.118	0.739	-0.105	0.750	-0.091	0.762	-0.077	-0.098
RESCAL	0.869	0.830	-0.046	0.836	-0.038	0.833	-0.041	0.838	-0.035	-0.040
RotatE	0.744	0.701	-0.058	0.692	-0.070	0.689	-0.075	0.706	-0.052	-0.064
TransE L1	0.824	0.785	-0.047	0.766	-0.070	0.773	-0.061	0.786	-0.046	-0.056
TransE L2	0.889	0.861	-0.032	0.861	-0.032	0.853	-0.040	0.868	-0.024	-0.032
TransR	0.825	0.779	-0.055	0.775	-0.061	0.775	-0.060	0.789	-0.043	-0.055
Avg. Δ			-0.051		-0.063		-0.055		-0.042	

Moreover, we observe the largest losses for ComplEx and DistMult, which are both tensor factorization methods, whereas RDF2vec SG has the smallest loss, compared to the uncompressed variant. Therefore, we conclude that binarization is not equally effective for each model, but that there are differences by model family.

Furthermore, we can observe that the binarization has a smaller impact than the choice of the embedding variant: even the binarized versions of RDF2vec SG OA yield superior results to the original, i.e., non binarized embeddings of most other embedding methods.

4. Conclusion and Outlook

In this paper, we have shown two strategies for reducing the size of a KGE model, one simple baseline using dimension-wise binarization, and an approach based on neural autoencoders. A promising preliminary evaluation on the DLCC entity classification benchmark shows that the size of embeddings can be drastically reduced at a comparatively low loss in downstream classification performance.

While the focus of this paper has been on optimizing the storage, we have not yet looked into possible gains in *computation time* for downstream processing, i.e., at inference time. For example, [12] have proposed processing binary vectors with logical bitwise operators, which could also yield significant performance gains over floating point operations. This is a promising, but yet underexplored area. Moreover, a comparison of binarizing classically trained embeddings with directly trained binary embeddings would be intriguing, especially for cases where no pre-trained embeddings exist.

So far, we have only looked into tasks involving single entities, as defined in the gEval and DLCC benchmarks. The transfer to another popular usage of knowledge graph embeddings, i.e., link prediction and triple scoring, has not yet been investigated. Except for the B-CP approach [23], there are, to the best of our knowledge, no methods for exploiting binary knowledge graph embeddings for link prediction. In particular, since B-CP directly learns the embeddings, there are no studies on binarizing existing embeddings for link prediction.

In summary, the results have shown that binary knowledge graph embeddings can be an appealing alternative to the widely used floating point representations, allowing a drastic reduction of storage space while, at the same time, leading to only marginal loss in downstream performance.

References

[1] J. Portisch, N. Heist, H. Paulheim, Knowledge graph embedding for data mining vs. knowledge graph embedding for link prediction – two sides of the same coin?, Semantic Web 13 (2022) 1–24. doi:10.3233/SW-212892.

[2] Q. Wang, Z. Mao, B. Wang, L. Guo, Knowledge graph embedding: A survey of approaches and applications, IEEE transactions on knowledge and data engineering 29 (2017) 2724–2743.

[3] R. Biswas, L.-A. Kaffee, M. Cochez, S. Dumbrava, T. E. Jendal, M. Lissandrini, V. Lopez, E. L. Mencía, H. Paulheim, H. Sack, et al., Knowledge graph embeddings: open challenges and opportunities, Transactions on Graph Data and Knowledge 1 (2023) 4–1.

[4] D. Zheng, X. Song, C. Ma, Z. Tan, Z. Ye, J. Dong, H. Xiong, Z. Zhang, G. Karypis, Dgl-ke: Training knowledge graph embeddings at scale, in: Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, 2020, pp. 739–748.

[5] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, C. Bizer, DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia, Semantic Web 6 (2015) 167–195. URL: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-140134. doi:10.3233/SW-140134.

[6] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space, 2013. URL: http://arxiv.org/abs/1301.3781. doi:10.48550/arXiv.1301.3781, arXiv:1301.3781 [cs].

[7] R. Řehůřek, P. Sojka, et al., Gensim—statistical semantics in python, Retrieved from genism. org (2011).

[8] H. Paulheim, P. Ristoski, J. Portisch, Embedding Knowledge Graphs with RDF2vec, Springer Nature, 2023.

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems 32 (2019).

[10] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp, J. Lehmann, Pykeen 1.0: a python library for training and evaluating knowledge graph embeddings, Journal of Machine Learning Research 22 (2021) 1–6.

[11] J. Portisch, M. Hladik, H. Paulheim, KGvec2go – Knowledge Graph Embeddings as a Service, 2020. URL: http://arxiv.org/abs/2003.05809. doi:10.48550/arXiv.2003.05809, arXiv:2003.05809 [cs].

[12] J. Tissier, C. Gravier, A. Habrard, Near-lossless Binarization of Word Embeddings, Proceedings of the AAAI Conference on Artificial Intelligence 33 (2019) 7104–7111. URL: http://arxiv.org/abs/1803.09065. doi:10.1609/aaai.v33i01.33017104, arXiv:1803.09065 [cs].

[13] Y. Wang, H. Yao, S. Zhao, Auto-encoder based dimensionality reduction, Neurocomputing 184 (2016) 232–242.

[14] J. Portisch, H. Paulheim, The DLCC Node Classification Benchmark for Analyzing Knowledge Graph Embeddings, 2022. URL: http://arxiv.org/abs/2207.06014, arXiv:2207.06014

[cs].
[15] J. Portisch, H. Paulheim, Putting RDF2vec in Order, 2021. URL: http://arxiv.org/abs/2108.05280. doi:10.48550/arXiv.2108.05280, arXiv:2108.05280 [cs].
[16] P. Ristoski, J. Rosati, T. Di Noia, R. De Leone, H. Paulheim, RDF2Vec: RDF graph embeddings and their applications, Semantic Web 10 (2019) 721–752. URL: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-180317. doi:10.3233/SW-180317.
[17] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating Embeddings for Modeling Multi-relational Data 2013 (2013).
[18] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, G. Bouchard, Complex embeddings for simple link prediction, in: International conference on machine learning, PMLR, 2016, pp. 2071–2080.
[19] B. Yang, W.-t. Yih, X. He, J. Gao, L. Deng, Embedding Entities and Relations for Learning and Inference in Knowledge Bases, 2014. URL: https://arxiv.org/abs/1412.6575v4.
[20] M. Nickel, V. Tresp, P. Kröger, A Three-Way Model for Collective Learning on Multi-Relational Data., 2011, pp. 809–816.
[21] Z. Sun, Z.-H. Deng, J.-Y. Nie, J. Tang, RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space, 2019. URL: http://arxiv.org/abs/1902.10197. doi:10.48550/arXiv.1902.10197, arXiv:1902.10197 [cs, stat].
[22] Y. Lin, Z. Liu, M. Sun, Y. Liu, X. Zhu, Learning Entity and Relation Embeddings for Knowledge Graph Completion, Proceedings of AAAI 29 (2015) 2181–2187. doi:10.1609/aaai.v29i1.9491.
[23] K. Hayashi, K. Kishimoto, M. Shimbo, Binarized embeddings for fast, space-efficient knowledge graph completion, IEEE Transactions on Knowledge and Data Engineering 35 (2021) 141–153.