Journal of Machine Learning Research 22 (2021) 1-6 Submitted 7/20; Revised 2/21; Published 3/21

PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings

Mehdi Ali* MEHDI.ALI@CS.UNI-BONN.DE
Smart Data Analytics Group, University of Bonn & Fraunhofer IAIS

Max Berrendorf* BERRENDORF@DBS.IFI.LMU.DE
Ludwig-Maximilians-Universität München

Charles Tapley Hoyt* CHARLES.HOYT@ENVEDATX.COM
Enveda Biosciences

Laurent Vermue* LAUVE@DTU.DK
Technical University of Denmark

Sahand Sharifzadeh SHARIFZADEH@DBS.IFI.LMU.DE
Ludwig-Maximilians-Universität München

Volker Tresp VOLKER.TRESP@SIEMENS.COM
Ludwig-Maximilians-Universität München & Siemens AG

Jens Lehmann JENS.LEHMANN@CS.UNI-BONN.DE
Smart Data Analytics Group, University of Bonn & Fraunhofer IAIS

Editor: Antti Honkela

Abstract

Recently, knowledge graph embeddings (KGEs) have received significant attention, and several software libraries have been developed for training and evaluation. While each of them addresses specific needs, we report on a community effort to a re-design and re-implementation of PyKEEN, one of the early KGE libraries. PyKEEN 1.0 enables users to compose knowledge graph embedding models based on a wide range of interaction models, training approaches, loss functions, and permits the explicit modeling of inverse relations. It allows users to measure each component's influence individually on the model's performance. Besides, an automatic memory optimization has been realized in order to optimally exploit the provided hardware. Through the integration of Optuna, extensive hyper-parameter optimization (HPO) functionalities are provided.

Keywords: Knowledge Graphs, Knowledge Graph Embeddings, Relational Learning

1. Introduction

Knowledge graphs (KGs) encode knowledge as a set of triples $\mathcal{K} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ where $\mathcal{E}$ denotes the set of entities and $\mathcal{R}$ the set of relations. Knowledge graph embedding models (KGEMs) learn representations for entities and relations of KGs in vector spaces while preserving the graph structure. The learned embeddings can support machine learning tasks such as entity clustering, link prediction, entity disambiguation, as well as downstream tasks such

*Equal contribution.

as question answering and item recommendation (Nickel et al., 2015; Wang et al., 2017; Ruffinelli et al., 2020; Kazemi et al., 2020).

Most publications of KGEMs are accompanied by reference implementations, but they are seldomly written for reusability or maintained. Existing software packages that provide implementations for different KGEMs usually lack composability: model architectures (or interaction models), training approaches, loss functions, and the usage of explicit inverse relations cannot arbitrarily be combined. The full composability of KGEMs is fundamental for assessing their performance because it allows the assessment of individual components and not solely the sum of differences in published approaches (Ruffinelli et al., 2020). In most previous libraries, only limited functionalities are provided, e.g., a small number of KGEMs are supported, or functionalities such as hyper-parameter optimization (HPO) are missing. For instance, in PyKEEN (Ali et al., 2019a,b), one of the early software packages for KGEMs, models can only be trained under the stochastic local closed-world approach, the evaluation procedure was too slow for larger KGs, and it was designed to be mainly used through a command-line interface rather than programmatically, in order to facilitate its usage for non-experts. This motivated the development of a reusable software package comprising several KGEMs and related methodologies that is entirely configurable.

Here, we present PyKEEN (Python KnowlEdge EmbeddiNgs) 1.0, a community effort in which PyKEEN has been re-designed and re-implemented from scratch to overcome the mentioned limitations, to make models entirely configurable, and to extend it with more interaction models and other components.

2. System Description

In PyKEEN 1.0, a KGEM is considered as a composition of four components that can flexibly be combined: an interaction model (or model architecture), a loss function, a training approach, and the usage of inverse relations. PyKEEN 1.0 currently supports 23 interaction models, seven loss functions, four regularizers, two training approaches, HPO, six evaluation metrics, and 21 built-in benchmarking datasets. It can readily import additional datasets that have been pre-stratified into train/test/evaluation and generate appropriate splits for unstratified datasets. Additionally, we implemented an automatic memory optimization that ensures that the available memory is best utilized.

Composable KGEMs To ensure the composability of KGEMs, the interaction models, loss functions, and training approaches are separated from each other and implemented as independent submodules, whereas the modeling of inverse relations is handled by the interaction models. Our modules can be arbitrarily replaced because we ensured through inheritance that all interaction models, loss functions, and training approaches follow unified APIs, which are defined by *pykeen.model.Model*, *pykeen.loss.Loss*, and *pykeen.training.TrainingLoop*. Currently, we provide implementations of 23 interaction models, the most common loss functions used for training KGEMs including the *binary-cross entropy*, *cross entropy*, *mean square error*, *negative-sampling self-adversarial loss*, and the *softplus loss*, as well as the *local closed-world assumption* (also referred as *KvsAll*) and the *stochastic local closed-world assumption* training approach (also refereed as *NegSamp*) (Ruffinelli et al., 2020). In PyKEEN, each interaction model can be trained based on both approaches. To enable users to investigate the effect of explicitly modeling

inverse relations (Lacroix et al., 2018; Kazemi and Poole, 2018) on the model's performance, each model can be trained with explicit inverse relations in PyKEEN 1.0, i.e., for each relation $r \in \mathcal{R}$ an inverse relation r_{inv} is introduced, and the task of predicting the head entity of a (r, t)-pair becomes the task of predicting the tail entity of the corresponding inverse pair (t, r_{inv}).

To facilitate the composition of KGE models for non-experts, we provide the pykeen.pipeline.pipeline() functions, which provides a high-level entry point into the functionalities of PyKEEN. Users define the components to be used, and the pipeline ensures the correct composition of the KGEM and the correct composition of the training and evaluation workflow.

Evaluation KGEMs are usually evaluated on the task of link prediction. Given (h, r) (or (r, t)), all possible entities $\mathcal{E}$ are considered as tail (or head) and ranked according to the KGEMs interaction model. The individual ranks are commonly aggregated to mean rank, mean reciprocal rank, and hits@k. However, these metrics have been realized differently throughout the literature based on different definitions of the rank, leading to difficulties in reproducibility and comparability (Sun et al., 2019). The three most common rank definitions are the *average rank*, *optimistic rank*, and *pessimistic rank*. In PyKEEN 1.0, we explicitly compute the aggregation metrics for all common rank definitions, *average*, *optimistic*, and *pessimistic*, allowing inspection of differences between them. This can help to reveal cases where the model predicts exactly equal scores for many different triples, which is usually an undesired behavior. In addition, we support the recently proposed *adjusted mean rank* (Berrendorf et al., 2020), which allows the comparison of results across differently sized datasets, as well as offering an interface to use all metrics implemented in scikit-learn (Pedregosa et al., 2011), including AUC-PR and AUC-ROC.

Automatic Memory Optimization Allowing high computational throughput, while ensuring that the available hardware memory is not exceeded during training and evaluation, requires the knowledge of the maximum possible training and evaluation batch size for the current model configuration. However, determining the training and evaluation batch sizes is a tedious process, and not feasible when a large set of heterogeneous experiments are run. Therefore, we implemented an automatic memory optimization step that computes the maximum possible training and evaluation batch sizes for the current model configuration and available hardware before the actual experiment starts. If the user-provided batch size is too large for the used hardware, the automatic memory optimization determines the maximum sub-batch size for the training.

Extensibility Because we defined a uniform API for each interaction model, any new model can be integrated by following the API of the existing models (*pykeen.models*). Similarly, the remaining components, e.g., regularizers, and negative samplers follow a unified API, so that new modules can be smoothly integrated.

Community Standards PyKEEN 1.0 relies on several community-oriented tools to ensure it is accessible, reusable, reproducible, and maintainable. It is implemented for Python 3.7+ using the PyTorch package. It comes with a suite of thorough unit tests that are automated with PyTest, Tox, run in a continuous integration setting on GitHub Actions, and are tracked over time using `codecov.io`. Code quality is ensured with flake8 and careful

Library	AMO	Models	HPO	ES	Evaluation Metrics	Set TA	Set Inv. Rels.	Set Loss Fct.	MGS	DTR
AmpliGraph (Costabello et al., 2019)	-	6	✓	✓	3	-	✓	✓	-	-
DGL-KE (Zheng et al., 2020)	-	6	-	-	3	-	-	✓	✓	✓
GraphVite (Zhu et al., 2019)	-	6	-	-	4	-	-	-	✓	-
LibKGE (Broscheit et al., 2020)	-	10	✓	✓	3*	✓	✓	✓	-	-
OpenKE (Han et al., 2018)	-	11	-	-	3	-	-	✓	-	-
PyTorch-BigGraph (Lerer et al., 2019)	-	4	-	-	4	-	-	✓	✓	✓
Pykg2vec (Yu et al., 2019)	-	18	✓	✓	2	-	-	-	-	-
PyKEEN (Ali et al., 2019b)	-	10	✓	-	2	-	-	-	-	-
PyKEEN 1.0	✓	**23**	✓	✓	**6***	✓	✓	✓	-	-

Table 1: An overview of the functionalities (determined July 2020) of PyKEEN 1.0 and similar libraries. **AMO** refers to automatic memory optimization, **ES** to early stopping, ***** indicates that ranking metrics are computed for different definitions of the rank, **Set TA** refers to interchanging the training approach, **Set Inv. Rels.** to the explicit modeling of inverse relations, **MGS** to multi-GPU support, i.e., training a single model across several GPUs, and **DTR** to distributed training.

application of the GitHub Flow development workflow. Documentation is quality checked by doc8, built with Sphinx, and hosted on `ReadTheDocs.org`.

3. Comparison to Related Software

Table 1 depicts the most popular KGE frameworks and their features. It shows that PyKEEN 1.0, in comparison with related software packages, emphasizes on both, full composability of KGEMs and extensive functionalities, i.e., a large number of supported interaction models, and extensive evaluation (several metrics are supported) and HPO functionalities. Concerning the evaluation metrics, PyKEEN and LibKGE are the only libraries that compute the ranking metrics (i.e., *mean rank* and *hits@k*) for different definitions of the rank, which ensures that undesired cases are detected in which the model predicts equal scores for many triples. Finally, PyKEEN 1.0 is the only library that performs an automatic memory optimization that ensures that the memory is not exceeded during training and evaluation. GraphVite, DGL-KE, and PyTorch-BibGraph focus on scalability, i.e., they provide support for multi-GPU/CPU or/and distributed training, but focus less on compositionality and extensibility. For instance, PyTorch-BigGraph supports only a small number of interaction models that follow specific computation blocks.

4. Availability and Maintenance

PyKEEN 1.0 is publicly available under the MIT License at `https://github.com/pykeen/pykeen`, and is distributed through the Python Package Index. It will be maintained by the core developer team that is supported by the Smart Data Analytics research group (University of Bonn), Fraunhofer IAIS, Munich Center for Machine Learning (MCML),

Siemens, and the Technical University of Denmark (section for Cognitive Systems and section for Statistics and Data Analysis). The project is funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and Grant No. 01IS18050D (project MLWin) as well as the Innovation Fund Denmark with the Danish Center for Big Data Analytics driven Innovation (DABAI) which ensures the maintenance of the project in the next years.

References

Mehdi Ali, Charles Tapley Hoyt, Daniel Domingo-Fernández, Jens Lehmann, and Hajira Jabeen. Biokeen: a library for learning and evaluating biological knowledge graph embeddings. *Bioinformatics*, 35(18):3538–3540, 2019a.

Mehdi Ali, Hajira Jabeen, Charles Tapley Hoyt, and Jens Lehmann. The keen universe. In *International Semantic Web Conference*, pages 3–18. Springer, 2019b.

Max Berrendorf, Evgeniy Faerman, Laurent Vermue, and Volker Tresp. Interpretable and fair comparison of link prediction or entity alignment methods with adjusted mean rank. In *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'20)*. IEEE, 2020.

Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. Libkge - A knowledge graph embedding library for reproducible research. In *EMNLP (Demos)*, pages 165–174. Association for Computational Linguistics, 2020.

Luca Costabello, Sumit Pai, Chan Le Van, Rory McGrath, Nicholas McCarthy, and Pedro Tabacof. AmpliGraph: a Library for Representation Learning on Knowledge Graphs, March 2019. URL `https://doi.org/10.5281/zenodo.2595043`.

Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. Openke: An open toolkit for knowledge embedding. In *Proceedings of EMNLP*, 2018.

Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems*, pages 4284–4295, 2018.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. *arXiv preprint arXiv:1806.07297*, 2018.

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You {can} teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2020.

Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. *arXiv preprint arXiv:1911.03903*, 2019.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

Shih Yuan Yu, Sujit Rokka Chhetri, Arquimedes Canedo, Palash Goyal, and Mohammad Abdullah Al Faruque. Pykg2vec: A python library for knowledge graph embedding. *arXiv preprint arXiv:1906.04239*, 2019.

Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. Dgl-ke: Training knowledge graph embeddings at scale. *arXiv preprint arXiv:2004.08532*, 2020.

Zhaocheng Zhu, Shizhen Xu, Jian Tang, and Meng Qu. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In *The World Wide Web Conference*, pages 2494–2504, 2019.