OpenKE: An Open Toolkit for Knowledge Embedding

Xu Han[1]*, **Shulin Cao**[2]*, **Xin Lv**[1], **Yankai Lin**[1], **Zhiyuan Liu**[1], **Maosong Sun**[1,3], **Juanzi Li**[1]
[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2]Department of Computer Science and Technology, Beijing Normal University, Beijing, China
[3]Beijing Advanced Innovation Center for Imaging Technology,
Capital Normal University, Beijing, China

Abstract

We release an open-source toolkit for knowledge embedding (OpenKE), which provides a unified framework and various fundamental models to embed knowledge graphs into continuous low-dimensional spaces. OpenKE prioritizes operational efficiency to support quick model validation and large-scale knowledge embedding. Meanwhile, OpenKE maintains enough modularity and extensibility to enable new models to be easily integrated into the framework. Besides the toolkit, the embeddings of some existing large-scale knowledge graphs pre-trained by OpenKE are also available, which can be directly used for relevant applications. The toolkit, technical documentation, and pre-trained embeddings are all released on our website [1].

1 Introduction

People construct various large-scale knowledge graphs (KGs) to organize structural knowledge about the world, such as WordNet (Miller, 1995), Freebase (Bollacker et al., 2008) and Wikidata (Vrandečić and Krötzsch, 2014). These typical KGs are organized in the form of triples (h, r, t), with h and t indicating *head* and *tail* entities and r indicating the relation between h and t, e.g., (*Mark Twain*, `PlaceOfBirth`, *Florida*). Abundant structural information in KGs is widely used to enhance various knowledge-driven applications (e.g. question answering and web search) with the ongoing effective construction of KGs.

Limited by the scale and sparsity of KGs, we have to represent KGs with corresponding distributed representations to utilize knowledge information for specific applications. Therefore, a variety of knowledge embedding (KE) approaches have been proposed to embed both entities and relations in KGs into continuous low-dimensional spaces, such as linear models (Bordes et al., 2011, 2012, 2014), latent factor models (Sutskever et al., 2009; Jenatton et al., 2012; Yang et al., 2015; Liu et al., 2017), neural models (Socher et al., 2013; Dong et al., 2014), matrix factorization models (Nickel et al., 2011, 2012, 2016; Trouillon et al., 2016), and translation models (Bordes et al., 2013; Wang et al., 2014; Lin et al., 2015; Ji et al., 2015). Although these models achieve great results on the benchmark datasets, their existing implementations are scattered and unsystematic. Moreover, the codes for model validation are often time-consuming and their interfaces are also different from each other.

These issues lead to difficulty in further development based on these models, and adopting them for real-world applications. Hence, it becomes important to develop an efficient and effective open-source toolkit for KE, which may benefit both the communities in academia and industry. For this purpose, we develop an open-source KE toolkit and name the toolkit "OpenKE". The toolkit provides a flexible framework and unified interfaces for developing KE models. While taking in some targeted optimization methods, the platform of OpenKE enables KE models to become more efficient and capable of embedding large-scale KGs. The design and optimization features of OpenKE are threefold:

(1) At the data and memory level, the unified framework of OpenKE serves KE models via the underlying management of data and memory. Model developments based on OpenKE no longer require complicated data processing.

(2) At the algorithm level, OpenKE unify the mathematical forms of various specific models to implement them based on the unified framework. We also propose a novel negative sampling strat-

* indicates equal contribution
[1] http://openke.thunlp.org/

Model	Scoring Function	Parameters	Loss Function
RESCAL (Nickel et al., 2011)	$\mathbf{h}^\top \mathbf{M}_r \mathbf{t}$	$\mathbf{M}_r \in \mathbb{R}^{k \times k}, \mathbf{h} \in \mathbb{R}^k, \mathbf{t} \in \mathbb{R}^k$	margin-based loss
TransE (Bordes et al., 2013)	$-\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{L_1/L_2}$	$\mathbf{r} \in \mathbb{R}^k, \mathbf{h} \in \mathbb{R}^k, \mathbf{t} \in \mathbb{R}^k$	margin-based loss
TransH (Wang et al., 2014)	$-\|(\mathbf{h} - \mathbf{w}_r^\top \mathbf{h} \mathbf{w}_r) + \mathbf{r} - (\mathbf{t} - \mathbf{w}_r^\top \mathbf{t} \mathbf{w}_r)\|_{L1/L2}$	$\mathbf{w}_r \in \mathbb{R}^k, \mathbf{r} \in \mathbb{R}^k, \mathbf{h} \in \mathbb{R}^k, \mathbf{t} \in \mathbb{R}^k$	margin-based loss
TransR (Lin et al., 2015)	$-\|\mathbf{M}_r \mathbf{h} + \mathbf{r} - \mathbf{M}_r \mathbf{t}\|_{L_1/L_2}$	$\mathbf{M}_r \in \mathbb{R}^{k_r \times k_e}, \mathbf{r} \in \mathbb{R}^{k_r}, \mathbf{h} \in \mathbb{R}^{k_e}, \mathbf{t} \in \mathbb{R}^{k_e}$	margin-based loss
TransD (Ji et al., 2015)	$-\|(\mathbf{r}_p \mathbf{h}_p^\top + \mathbf{I})\mathbf{h} + \mathbf{r} - (\mathbf{r}_p \mathbf{t}_p^\top + \mathbf{I})\mathbf{t}\|_{L1/L2}$	$\mathbf{r}_p \in \mathbb{R}^{k_r}, \mathbf{h}_p \in \mathbb{R}^{k_e}, \mathbf{t}_p \in \mathbb{R}^{k_e}, \mathbf{I} \in \mathbb{R}^{k_r \times k_e}$, $\mathbf{r} \in \mathbb{R}^{k_r}, \mathbf{h} \in \mathbb{R}^{k_e}, \mathbf{t} \in \mathbb{R}^{k_e}$	margin-based loss
DistMult (Yang et al., 2015)	$< \mathbf{h}, \mathbf{r}, \mathbf{t} >$	$\mathbf{r} \in \mathbb{R}^k, \mathbf{h} \in \mathbb{R}^k, \mathbf{t} \in \mathbb{R}^k$	logistic loss
HolE (Nickel et al., 2016)	$\mathbf{r}^\top \left(\mathcal{F}^{-1}\left(\overline{\mathcal{F}(\mathbf{h})} \odot \mathcal{F}(\mathbf{t})\right)\right)$	$\mathbf{r} \in \mathbb{R}^k, \mathbf{h} \in \mathbb{R}^k, \mathbf{t} \in \mathbb{R}^k$	logistic loss
ComplEx (Trouillon et al., 2016)	$\Re(< \mathbf{h}, \mathbf{r}, \overline{\mathbf{t}} >)$	$\mathbf{r} \in \mathbb{C}^k, \mathbf{h} \in \mathbb{C}^k, \mathbf{t} \in \mathbb{C}^k$	logistic loss

Table 1: The brief introduction of some typical KE models. For most models, k is the dimension of both entities and relations. For some other models, K_e is the dimension of entities and k_r is the dimension of relations. $\mathcal{F}$ denotes the Fourier transform. $\odot$ denotes the element-wise product. $< a, b, c >$ denotes the element-wise multi-linear dot product.

egy instead of the original one to merge arithmetic operations for further acceleration.

(3) At the computational level, OpenKE can separate the overall KG into several parts and adapts KE models for parallel training. Based on the underlying management of data and memory, we also adopt TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2017) to build a convenient platform to run models on GPUs. These interfaces and functions almost cover the usual development habits of users.

Besides the toolkit, we also provide the pre-trained embeddings of some existing large-scale KGs, which can be used directly for other relevant works without repeatedly spending much time embedding KGs. In this paper, we mainly present the architecture design and implementation of OpenKE, as well as the benchmark evaluation results of some typical KE models implemented with OpenKE. Other related resources and details can be found on our website.

2 Background

For a typical KG $\mathcal{G}$, it expresses data as a directed graph $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{T}\}$, where $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{T}$ indicate the sets of entities, relations and facts respectively. Each triple $(h, r, t) \in \mathcal{T}$ indicates there is a relation $r \in \mathcal{R}$ between $h \in \mathcal{E}$ and $t \in \mathcal{E}$. For the entities $h, t \in \mathcal{E}$ and the relation $r \in \mathcal{R}$, we use the bold face $\mathbf{h}, \mathbf{t}, \mathbf{r}$ to indicate their low-dimensional vectors respectively.

For any entity pair $(h, t) \in E \times E$ and any relation $r \in \mathcal{R}$, we can determine whether there is a fact $(h, r, t) \in \mathcal{T}$ via their low-dimensional embeddings learned by KE models. These embeddings greatly facilitate understanding and mining knowledge in KGs. In practice, the KE models define a scoring function $S(h, r, t)$ for each triple (h, r, t). In most cases, there are only true triples in KGs, and KGs are also far from completion. Non-existing triples can be either false or missing. Local closed world assumption (Dong et al., 2014) has been proposed to solve this problem, which requires existing triples to have higher scores than those non-existing ones. Hence, the scoring function $S(h, r, t)$ returns a higher score if (h, r, t) is true, vice versa.

Some KE models formalize a margin-based loss as training objective to learn the embeddings of the entities and relations:

$$\mathcal{L} = \sum_{t \in \mathcal{T}} \sum_{t' \in \mathcal{T}'} \left[\gamma + S(t') - S(t)\right]_+. \tag{1}$$

Here $[x]_+$ indicates keeping the positive part of x and $\gamma > 0$ is a margin. $\mathcal{T}'$ denotes the set of non-existing triples, which is constructed by corrupting entities and relations in existing triples,

$$\mathcal{T}' = \mathcal{E} \times \mathcal{R} \times \mathcal{E} - T. \tag{2}$$

Some other KE models cast the training objective as a classification task. The embeddings of the entities and relations can be learned by minimizing the regularized logistic loss,

$$\mathcal{L} = \sum_{t \in \mathcal{T}} \log(1 + \exp(-S(t))) + \sum_{t' \in \mathcal{T}'} \log(1 + \exp(S(t'))). \tag{3}$$

As mentioned above, there are many different KE models. The main difference among these models is their scoring functions. Hence, we briefly introduce several typical models and their scoring functions in Table 1. These models and their extensions are state-of-the-art and widely introduced in many works. We systematically incorporate all of them into our OpenKE.

3 Design Goals

Before introducing the concrete toolkit implementations, we report the design goals and features of OpenKE, including system encapsulation, operational efficiency, and model extensibility.

3.1 Encapsulation

Developers tend to maximize the reuse of code that others have built to avoid unnecessary redundant development in practice. For KE, its task is fixed, and its experimental settings and model parameters are also similar. However, previous model implementations are scattered and lack of necessary interface encapsulation. Thus, developers have to spend extra time reading obscure open-source code and writing glue code for data processing when they construct models for their applications. In view of this issue, we build a unified underlying platform in OpenKE and encapsulate various data and memory processing which is independent of model implementations. As shown in Figure 1, the system encapsulation makes it easy to train and test KE models, and we just need to set hyperparameters via interfaces of the platform to construct KE models.

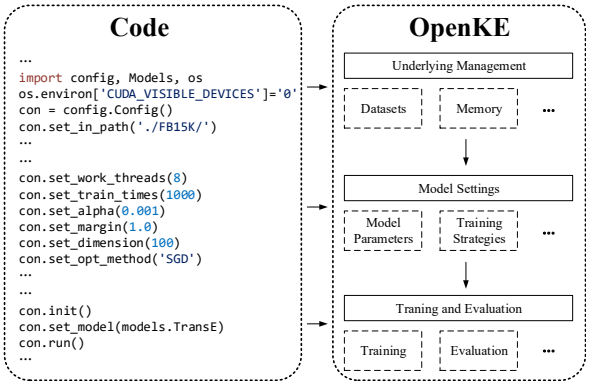

Figure 1: An example for training a KE model (TransE) via OpenKE.

```python
import numpy as np
import tensorflow as tf
from Model import *
class TransE(Model):
	def _calc(self, h, t, r):
		return abs(h + r - t)
	def embedding_def(self):
		config = self.get_config()
		self.ent_embeddings = tf.get_variable('ent_embeddings',
				[config.entTotal, config.hidden_size])
		self.rel_embeddings = tf.get_variable('rel_embeddings',
				[config.relTotal, config.hidden_size])
	def loss_def(self):
		config = self.get_config()
		pos_h, pos_t, pos_r = self.get_positive_instance(in_batch = True)
		neg_h, neg_t, neg_r = self.get_negative_instance(in_batch = True)
		p_h = tf.nn.embedding_lookup(self.ent_embeddings, pos_h)
		p_t = tf.nn.embedding_lookup(self.ent_embeddings, pos_t)
		p_r = tf.nn.embedding_lookup(self.rel_embeddings, pos_r)
		n_h = tf.nn.embedding_lookup(self.ent_embeddings, neg_h)
		n_t = tf.nn.embedding_lookup(self.ent_embeddings, neg_t)
		n_r = tf.nn.embedding_lookup(self.rel_embeddings, neg_r)
		_p_score = self._calc(p_h, p_t, p_r)
		_n_score = self._calc(n_h, n_t, n_r)
		p_score =  tf.reduce_sum(tf.reduce_mean(_p_score, 1, keep_dims = False),
				1, keep_dims = True)
		n_score =  tf.reduce_sum(tf.reduce_mean(_n_score, 1, keep_dims = False),
				1, keep_dims = True)
		self.loss = tf.reduce_sum(tf.maximum(p_score - n_score + config.margin, 0))
```

Figure 2: An example for implementing a KE model (TransE) via OpenKE.

3.2 Efficiency

Previous model implementations focus on model validation and enhancing experimental results rather than improving time and space efficiency. In fact, as real-world KGs can be very large, training efficiency is an important concern. Hence, OpenKE integrates efficient computing power, training methods, and various acceleration strategies to support KE models. We adopt TensorFlow and PyTorch to implement the model training and test modules based on the interfaces of the underlying platform. These machine learning frameworks enable models to be run on GPU, with just few minutes needed for training and testing models on benchmark datasets. In order to train existing large-scale KGs, we also implement lightweight C/C++ versions for quick deployment and multi-threading acceleration of KE models, in which some models (e.g. TransE) can embed more than $100M$ triples in the few hours on ordinary devices.

3.3 Extensibility

In view that different KE models have different design solutions, we fully consider making OpenKE extensible to future variants when designing and implementing OpenKE. For the underlying platform, we encapsulate data processing and memory management, and then provide various data sampling interfaces. For the training modules, we provide enough interfaces for possible training methods. For the construction of KE models, we unify their mathematical forms and encapsulate them into a base class. These framework designs can greatly meet the needs of current and future models, and customized interfaces to meet individual requirements are also available in OpenKE. As shown in Figure 2, all specific models are implemented by inheriting the base class with designing their own scoring functions and loss functions. In addition, models in OpenKE can be placed into the framework of TensorFlow and PyTorch to interact with other machine learning models.

4 Implementations

In this section, we mainly present the implementations of the acceleration modules and the special

Algorithm 1 Parallel Learning

Require: Entity and relation sets $\mathcal{E}$ and $\mathcal{R}$, training triples $\mathcal{T} = \{(h, r, t)\}$.

Initialize all model embeddings and parameters.
for $i \leftarrow 1$ to $epoches$ **do**
  In each thread:
  for $j \leftarrow 1$ to $batches/threads$ **do**
    Sample a positive triple (h, r, t)
    Sample a corrupted triple (h', r', t')
    Compute the loss function $\mathcal{L}$
    Update the gradient $\nabla\mathcal{L}$
  end for
end for
Return all embeddings and parameters

sampling algorithm in OpenKE. OpenKE has been available to the public on GitHub [2] and is open-source under the MIT license. Other Implementation details can be found in the technical documentation of OpenKE on the website.

4.1 GPU Learning

GPUs are widely used in machine learning tasks to speed up model training in recent years. In order to accelerate KE models, we integrate GPU learning mechanisms into our toolkit. We build the GPU learning platform based on TensorFlow and PyTorch. Both TensorFlow and PyTorch are machine learning libraries, which provide effective hardware optimizations and abundant arithmetic operators for convenient model constructions, especially the stable environments for GPU learning. The autograd packages in these libraries also bring additional convenience. TensorFlow and PyTorch enable us to coustruct models with no need for manual back propagation implementations, which further reduces the programming complexity for GPU Learning. We develop necessary encapsulation modules aligning to TensorFlow and PyTorch so that the development and deployment of KE models can be faster and further convenient. Models can be deployed easily on a variety of devices without implementing complicated device setting code, even for multiple GPUs.

4.2 Parallel Learning

Abundant computing resources (e.g Servers with multiple GPUs) do not exist all the time. In fact, we often rely on simple personal computers for model validation. Hence, we enable OpenKE to adapt KE models for parallel learning on CPU besides employing GPU learning, which allow users

[2] http://github.com/thunlp/OpenKE

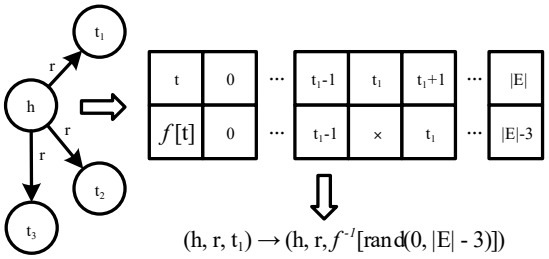

Figure 3: An example for the offset-based negative sampling algorithm.

to make full use of all available computing resources. The parallel learning method is shown in Algorithm 1. The main idea of the parallel learning method is based on the data parallelism mechanism, which divides training triples into several parts and train each part with a corresponding thread. In parallel learning, there are two strategies implemented to update gradients. One of the methods is the lock-free strategy, which means all threads share the unified embedding space and update embeddings directly without synchronized operations. We also implement a central synchronized method. In the central synchronized setting, each thread calculates its own gradient. After summing up the gradients from all threads, the results will be updated to the embeddings and parameters of models then.

4.3 Offset-based Negative Sampling

All KE models learn their parameters by minimizing the margin-based loss function Eq. (1) or the regularized logistic loss Eq. (3). Both of these loss functions need to construct non-existing triples as negative samples. We have empirically found that the corrupted triples have great influence on final performance. Randomly replacing entities or relations with any other ones may make the negative triple set $\mathcal{T}'$ contain some positive triples in $\mathcal{T}$, which would weaken the performance of KE models. The original sampling algorithm will spend much time checking whether generated triples are in T and filtering them out. In OpenKE, we propose an offset-based negative sampling algorithm to generate negative triples. As shown in Figure 3, we renumber all entities with new serial numbers. Each entity's new number is obtained by adding an offset to its original ID, and the offset is the total number of positive entities which have lower IDs. Our algorithm first randomly sample a new number and then map the new number back

to its corresponding entity. This algorithm can directly generate negative triples without any checking. Because the relation set is very small, we still directly replace positive relations for relation corruption.

5 Evaluations

In order to evaluate OpenKE, we implement different KE models with OpenKE, and compare their performance with the results reported in their papers on the link prediction task. Link prediction has been widely used for evaluating KE models, which needs to predict the tail entity when given a triple $(h, r, ?)$ or predict the head entity when given a triple $(?, r, t)$.

Some datasets are usually used as benchmarks for link prediction, such as FB15K and WN18. WN18 is the subset of WordNet; FB15K is the relatively dense subgraph of Freebase. These public datasets are available online [3]. Following previous works, We adopt them in our experiments. We list the statistics of FB15K and WN18 in Table 2, including the number of entities, relations, and facts.

Dataset	Rel	Ent	Train	Valid	Test
FB15K	1,345	14,951	483,142	50,000	59,071
WN18	18	40,943	141,442	5,000	5,000

Table 2: Statistics of FB15K and WN18.

As mentioned above, OpenKE supports models with efficient learning on both CPU and GPU. For CPU, the benchmarks are run on an Intel(R) Core(TM) i7-6700K @ 3.70GHz, with 4 cores and 8 threads. For GPU, the models in both TensorFlow and PyTorch versions are trained by GeForce GTX 1070 (Pascal), with CUDA v.8.0 (driver 384.111) and cuDNN v.6.5. To compare with the previous works and the results reported in their papers, we simply follow the parameter settings used before and traverse all training triples for 1000 rounds. Other detailed parameters and training strategies are shown in our source code. We show these results in Table 3 and Table 4. For further demonstrating the efficiency of OpenKE, we select TransE as a representative and implement it with both OpenKE and KB2E [4], and then compare their training time. KB2E is a widely-used toolkit for KE models on GitHub. These results can be found in Table 5.

[3] https://everest.hds.utc.fr/doku.php?id=en:transe

[4] https://github.com/thunlp/KB2E

Datasets / Models	FB15K TF	FB15K PT	FB15K MT
TransE	75.6(+28.5)	75.4(+28.3)	74.3(+27.2)
TransH	72.8(+14.3)	72.7(+14.2)	74.8(+16.3)
TransR	74.9(+6.2)	75.7(+7.0)	75.6(+6.9)
TransD	74.3(+0.1)	74.2(+0.0)	75.2(+1.0)
RESCAL	49.1(+5.0)	57.2(+13.1)	-
DistMult	73.4(+15.7)	75.4(+17.4)	-
HolE	70.4(−3.5)	-	-
ComplEx	72.3(−11.7)	80.5(−3.5)	-

Table 3: Experimental results of link prediction on FB15K (%).

Datasets / Models	WN18 TF	WN18 PT	WN18 MT
TransE	90.5(+1.3)	90.0(+0.8)	83.3(−5.9)
TransH	94.6(+7.9)	94.4(+7.7)	92.5(+5.8)
TransR	93.8(+1.8)	94.4(+2.4)	94.6(+2.9)
TransD	94.2(+1.7)	94.3(+1.8)	91.9(−0.3)
RESCAL	80.2(+27.4)	80.2(+27.4)	-
DistMult	93.6(−0.6)	93.6(−0.6)	-
HolE	94.4(−0.5)	-	-
ComplEx	94.0(−0.7)	94.0(−0.7)	-

Table 4: Experimental results of link prediction on WN18 (%).

From the results in Table 3, Table 4 and Table 5, we observe that: (1) Models implemented with OpenKE have the comparable accuracies compared to the values reported in the original papers. These results are compatible with our expectations. For some models, their accuracies are slightly higher due to OpenKE. These results indicate our toolkit is effecive. (2) OpenKE significantly accelerates the training process of the models trained both on CPU and GPU. As compared to the model implemented with KB2E, all models in OpenKE achieve more than $10\times$ speedup. These results show that our toolkit is efficient.

The evaluation results indicate that our toolkit significantly handles the time-consuming problem and can support existing models to learn large-scale KGs. In fact, TransE based on OpenKE only spends about 18 hours training the whole Wikidata for 10000 rounds and gets stable embeddings. There are more than $40M$ entities and $100M$ facts in Wikidata. We also evaluate the embeddings learned on the whole Wikidata on the link prediction task. Because the whole Wikidata is quite huge, we emphasize link prediction of Wikidata more on ranking a set of candidate entities rather than requiring one best answer. Hence, we report the proportion of correct entities in top-N ranked entities (Hits@10, Hits@20, Hits@50 and Hits@100) in Table 6. To our best knowledge, this is the first time that adopting KE models to embed an existing large-scale KG. The results shown in Table 6 indicate that OpenKE enables models to

Models	Time (s)
TransE (KB2E, CPU)	7124
TransE (OpenKE, CPU, 1-Thread)	386
TransE (OpenKE, CPU, 2-Thread)	206
TransE (OpenKE, CPU, 4-Thread)	118
TransE (OpenKE, CPU, 8-Thread)	76
TransE (OpenKE, GPU, TensorFlow)	178
TransE (OpenKE, GPU, PyTorch)	266

Table 5: Training time of different implementations of TransE on FB15K.

Metric	Hits@10	Hits@20	Hits@50	Hits@100
Head	29.6	36.2	46.7	56.3
Tail	66.8	75.2	84.9	90.6

Table 6: Experimental results of link prediction on the whole Wikidata.

effectively and efficiently embed large-scale KGs.

6 Conclusion

We propose an efficient open-source toolkit OpenKE for knowledge embedding. OpenKE builds a unified underlying platform to organize data and memory. OpenKE also applies GPU learning and parallel learning to speed up training. We also unify mathematical forms for specific models and encapsulate them to maintain enough modularity and extensibility. The experimental results demonstrate that the models implemented by OpenKE are efficient and effective. In the future, we will incorporate more knwoledge embedding models and maintain the stable embeddings of some large-scale knowledge graphs.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of OSDI*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of KDD*.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2012. Joint learning of words and meaning representations for open-text semantic parsing. In *Proceedings of AISTATS*.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2014. A semantic matching energy function for learning with multi-relational data. *Proceedings of ML*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Proceedings of NIPS*.

Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. 2011. Learning structured embeddings of knowledge bases. In *Proceedings of AAAI*.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of KDD*.

Rodolphe Jenatton, Nicolas L Roux, Antoine Bordes, and Guillaume R Obozinski. 2012. A latent factor model for highly multi-relational data. In *Proceedings of NIPS*.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of ACL*.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of AAAI*.

Hanxiao Liu, Yuexin Wu, and Yiming Yang. 2017. Analogical inference for multi-relational embeddings. In *Proceedings of ICML*.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic embeddings of knowledge graphs. In *Proceedings of AAAI*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of ICML*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2012. Factorizing yago: scalable machine learning for linked data. In *Proceedings of WWW*.

Adam Paszke, Soumith Chintala, Ronan Collobert, Koray Kavukcuoglu, Clement Farabet, Samy Bengio, Iain Melvin, Jason Weston, and Johnny Mariethoz. 2017. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of NIPS*.

Ilya Sutskever, Joshua B Tenenbaum, and Ruslan Salakhutdinov. 2009. Modelling relational data using bayesian clustered tensor factorization. In *Proceedings of NIPS*.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *Proceedings of ICML*.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of AAAI*.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of ICLR*.