Online Updates of Knowledge Graph Embedding

Luo Fei[1], Tianxing Wu[2], and Arijit Khan[1]

[1] Nanyang Technological University, Singapore
{fei.luo, arijit.khan}@ntu.edu.sg
[2] Southeast University, China
tianxingwu@seu.edu.cn

Abstract. Complex networks can be modeled as knowledge graphs (KGs) with nodes and edges denoting entities and relations among those entities, respectively. A knowledge graph embedding assigns to each node and edge in a KG a low-dimensional semantic vector such that the original structure and relations in the KG are approximately preserved in these learned semantic vectors. KG embeddings support downstream applications such as KG completion, classification, entity resolution, link prediction, question answering, and recommendation. In the real world, KGs are dynamic and evolve over time. State-of-the-art KG embedding models deal with static KGs. To support dynamic updates (even local), they must be retrained on the whole KG from scratch, which is inefficient. To this end, we propose a new context-aware Online Updates of Knowledge Graph Embedding (OUKE) method, which supports embedding updates in an online manner. OUKE learns two different vectors for each node and edge, i.e., knowledge embedding and context embedding. This strategy effectively limits the impacts of a local update in a smaller region, so that OUKE is able to efficiently update the KG embedding. Experiments on the link prediction in dynamic KGs demonstrate both effectiveness and efficiency of our solution.

Keywords: Knowledge graphs, embedding, dynamic updates.

1 Introduction

Knowledge graph is a data model for complex networks to manage large-scale and real-world facts [14, 9]. Examples include DBpedia [18], YAGO [13], Freebase [4], NELL [22], personalized health knowledge graphs [11], etc., where a node represents an entity, and an edge denotes a relationship between two entities. Knowledge graph embedding [32, 3] is increasingly becoming popular, which aims to represent each relation and entity in a knowledge graph $\mathcal{G}$ as a d-dimensional vector, such that the original structure and relations in $\mathcal{G}$ are approximately preserved in this semantic space. KG embeddings are used in downstream applications, e.g., link prediction [27, 36, 31], entity classification [37], question answering [14, 33], KG completion [6], and recommender systems [38].

In the real world, KGs are dynamic and evolve over time [26, 19]. DBpedia extracts the update stream of Wikipedia each day to keep the KG up-to-date

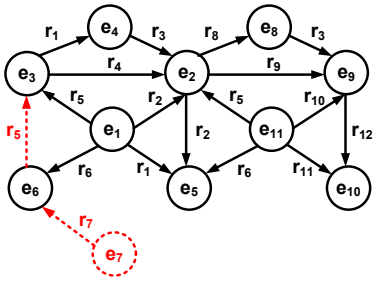


Fig. 1. Dynamic updates in a KG: dashed nodes and edges denote updates.

[12]. IMDB provides daily dumps of movies, TV series, actors, directors, among others, as well as their relationships [1]. Amazon product KG is updated quite frequently because there is a large number of new products everyday [9]. Recently, IBM's COVID-19 knowledge graph can ingest 100 000 PDF pages per day [2]. However, existing models [24, 5, 34, 36, 30, 10, 7, 3] embed static KGs. To support dynamic updates in a KG, these models must be retrained on the whole KG from scratch, which is inefficient, and is also impracticable when the KG has higher update frequency (e.g., once per day). To this end, we study the novel problem of efficiently updating a KG embedding in an online manner.

Consider the KG in Figure 1, the updates are denoted by dashed nodes and edges. Assume that we have embedded this KG using TransE [5], which should hold the translation relation: $\boldsymbol{h} + \boldsymbol{r} \approx \boldsymbol{t}$ (vectors are represented as bold characters) for each triple (h, r, t), where h is a head entity, r is a relation, and t is a tail entity. For instance, the triple (*Leonardo da Vinci*, *creator*, *Mona Lisa*) denotes that *Leonardo da Vinci* is the *creator* of the *Mona Lisa*. After adding a new triple (e_6, r_5, e_3), where e_6, e_3 are existing entities and r_5 is an existing relation in the earlier version of the KG, we now need to satisfy $\boldsymbol{e_6} + \boldsymbol{r_5} \approx \boldsymbol{e_5}$. No matter for which element in (e_6, r_5, e_3) we decide to update its vector, it will break the translation relations $\boldsymbol{h} + \boldsymbol{r} \approx \boldsymbol{t}$ for other triples containing our selected element, thereby creating a cascade of updates on the embedding of the entire KG. In summary, when a KG has local updates with addition and deletion of triples, if we revise the vectors of some entities and relations due to such updates, these revisions may cascade in the entire KG via connections among entities and relations, which is expensive.

When local updates occur in a knowledge graph, in the context of KG embedding, can we limit the impacts of such updates in certain regions, and not in the entire KG? To this end, we design a novel, context-aware Online Updates of Knowledge Graph Embedding (OUKE) approach, that answers this question affirmatively. We assign two different vectors to each entity and relation. When an entity (or a relation) denotes itself, we use a vector, called the *knowledge embedding*. When it denotes a part of the context of other entities (or relations), we use another vector, referred to as the *contextual element embedding*. In the neighborhood of an entity (or a relation), contextual element embeddings are aggregated to form the *contextual subgraph embedding* via a Relational Graph Convolution Network (R-GCN) [25]. We construct the *joint embedding* of each entity ($\boldsymbol{h}^\star$ or $\boldsymbol{t}^\star$) and relation ($\boldsymbol{r}^\star$) by combining the knowledge embedding (i.e.,

$\boldsymbol{h}^k$, $\boldsymbol{t}^k$, or $\boldsymbol{r}^k$) and the contextual subgraph embedding (i.e., $\boldsymbol{sg}(h)$, $\boldsymbol{sg}(t)$, or $\boldsymbol{sg}(r)$) via a gate strategy. Finally, we employ the joint embedding of each entity or relation to hold the translation relation: $\boldsymbol{h}^\star+\boldsymbol{r}^\star\approx\boldsymbol{t}^\star$.

We next propose an online learning algorithm to incrementally update the KG embedding. **(1)** Following the inductive learning, we keep all learnt parameters in R-GCNs and the gate strategy unaffected. **(2)** Contextual element embeddings of existing entities and relations also remain the same. **(3)** After a KG update, for many entities and relations, their contexts remain unchanged, so their contextual subgraph embeddings would remain uninterrupted. Thus, with existing knowledge embeddings of such entities and relations, corresponding triples would satisfy: $\boldsymbol{h}^\star+\boldsymbol{r}^\star\approx\boldsymbol{t}^\star$. Hence, we also keep the knowledge embeddings of existing entities and relations unchanged so long as their contexts are unchanged. **(4)** What shall we do with an existing entity or relation having changed context? Notice that its contextual subgraph embedding, a combination of context element embeddings of its neighboring entities or relations, computed by the R-GCN, will change to reflect this update. We next relearn the knowledge embeddings of existing entities and relations with changed contexts, and in that process we adjust both their knowledge embeddings and joint embeddings, with the aim that the joint embeddings, after such update, still approximately satisfy the translations in the modified graph. **(5)** In addition, we also learn knowledge embeddings and contextual element embeddings of emerging entities and relations. In this way, our algorithm greatly reduces the number of triples which need to be retrained while preserving $\boldsymbol{h}^\star+\boldsymbol{r}^\star\approx\boldsymbol{t}^\star$ on the whole KG. This enables online learning with higher efficiency.

Example 1. In Figure 1, after adding triples (e_7, r_7, e_6) and (e_6, r_5, e_3) into KG $\mathcal{G}$, we have an emerging entity e_7, an emerging relation r_7, four existing relations with changed contexts r_1, r_4, r_5, r_6, and two existing entities with changed contexts e_3 and e_6. Based on our online learning, we retrain only nine triples having $e_3, e_6, e_7, r_1, r_4, r_5, r_6$, and r_7; i.e., (e_3, r_1, e_4), (e_3, r_4, e_2), (e_1, r_5, e_3), (e_1, r_6, e_6), (e_6, r_5, e_3), (e_1, r_1, e_5), (e_7, r_7, e_6), (e_{11}, r_5, e_2), and (e_{11}, r_6, e_5) and not all eighteen triples in the updated version of $\mathcal{G}$. This demonstrates the efficiency improvement due to our method even with such a small KG. In particular, we learn knowledge embeddings, contextual element embeddings, and joint embeddings of the emerging entity e_7 and the emerging relation r_7. For existing relations with changed contexts (r_1, r_4, r_5, r_6) and existing entities with changed contexts (e_3 and e_6), their contextual element embeddings remain the same, but the contextual subgraph embeddings are updated via R-GCN, due to changed contexts. We also learn their updated knowledge embeddings and joint embeddings.

We empirically evaluate our method, OUKE (Online Updates of Knowledge Graph Embedding) on link prediction over dynamic KGs. Compared to static KG embedding models, OUKE has comparable effectiveness in different evaluation metrics, and better efficiency in online learning since the static models must be retrained on the entire KG.

Related Work. Static KG embedding can be translation-based (e.g., TransE [5], TransH [34], and TransR [20]), via compositions of head-tail entity pairs with

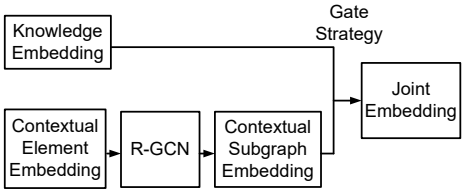


Fig. 2. Architecture of OUKE: Learning from Scratch.

their relations (e.g., RESCAL [24], DisMult [36], ComplEx [30]), as well as neural network-based (e.g., R-GCN [25] and ConvE [7]). GAKE [10] simultaneously models triples and structural contexts in embedding learning on static KGs.

Temporal KG embedding has been considered in [28, 8, 21] over multiple given snapshots of a KG, by incorporating time in the entity-relation space, to better perform link prediction, time prediction, and future fact prediction. In other words, they only conduct offline embedding learning with multiple given KG snapshots, but when faced with KG updates, they also need to be retrained on the whole KG, so unlike ours, they cannot embed dynamic KGs with high efficiency. To the best of our knowledge, puTransE [27] is the only existing model supporting online embedding learning for dynamic KGs. puTransE learns embeddings of entities and relations from local parts of a KG, so it avoids retraining on the entire KG, but cannot preserve the global structural information of the KG in the learnt embedding. When compared with puTransE [27], our proposed method, OUKE generally outperforms it in both effectiveness and efficiency.

Several dynamic graph embedding methods are also developed, e.g., [29, 39]. They incrementally compute the embeddings of new nodes and update existing node embeddings after a graph update. However, when we need to learn edge (i.e., relation) embeddings and consider various semantic correlations among nodes and edges in a dynamic KG embedding, these models cannot be applied.

2 Problem Formulation

We categorize the problem of learning KG embedding in a dynamic scenario as two sub-problems: Learning from scratch and online learning. Let a KG $\mathcal{G}^{\mathcal{T}} = \{(h, r, t)\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where $\{(h, r, t)\}$ denotes a set of triples, h is a head entity, r is a relation, t is a tail entity, e.g., $\{BMW, product, Germany\}$ is a triple in a KG indicating that *BMW* is *produced* in *Germany*, $\mathcal{E}$ and $\mathcal{R}$ are the sets of all entities and relations in $\mathcal{G}$, respectively, and $\mathcal{T}$ is the current time step. We assume that each entity e in a KG has at least one type [33, 23], denoted by $T(e)$, e.g., *BMW* is of type *Automobiles* and *Germany* is of type *Country*.

Problem 1. **Learning from Scratch.** KG $\mathcal{G}^{\mathcal{T}}$ is given as an input to our method at time step $\mathcal{T}$. The outputs are vector representations of entities and relations. Each entity or relation has two vectors: knowledge embedding and context element embedding.

At time step $\mathcal{T}+1$, $\mathcal{G}^{\mathcal{T}}$ becomes $\mathcal{G}^{\mathcal{T}+1}$ with updates including addition and deletion of triples, even having emerging entities and relations. We define *online learning* as follows:

Problem 2. **Online Learning.** The inputs are KG $\mathcal{G}^{\mathcal{T}+1}$ at time step $\mathcal{T}+1$, KG $\mathcal{G}^{\mathcal{T}}$ at time step $\mathcal{T}$, and earlier embedding at time step $\mathcal{T}$. Our method OUKE outputs updated and new vectors for entities and relations at time step $\mathcal{T}+1$.

3 Embedding Learning from Scratch

The architecture of learning from scratch in OUKE is presented in Figure 2. We assign two different vectors to each entity or a relation: *knowledge embedding* and *contextual element embedding*.

Learning from scratch involves two phases: (1) **context encoding** models the context of each entity or relation as a (multi)graph, and then utilizes a relational graph convolutional network (R-GCN) [25] to encode such contexts; (2) **embedding learning** applies a gate strategy to aggregate knowledge and context vectors, and then defines a loss function based on translation for training.

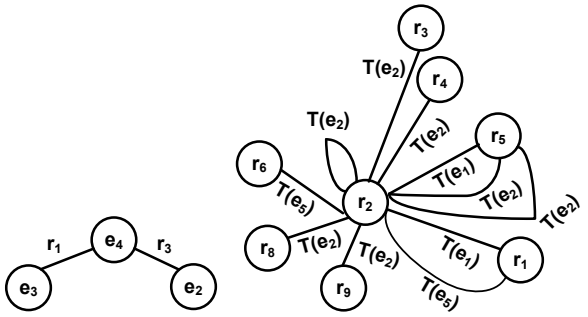
Context Encoding. We define the context of each entity as an undirected subgraph consisting of its neighbor entities (connected via some relations) and itself. For efficiency of OUKE, we only consider one-hop neighbor entities. In our experiments, when we consider more distant neighbors besides one-hop ones, it consumes much more time for model training, but OUKE's accuracy in link prediction does not significantly improve. This is because the further away entities are from each other, the less relevance they have [15, 17, 16], and less relevant neighbors may also introduce noise, in addition to useful information. Considering these trade-offs and based on our experimental results, we choose one-hop neighbor entities to build the context of each entity.

Different from entities, each relation occurs many times in a KG. We define the context of each relation as an undirected multi-graph consisting of all its neighbor relations (connected via entities having different types) and itself.

Example 2. In Figure 3, we show the contexts of entity e_4 and relation r_2, respectively, from the example KG in Figure 1. The context of e_4 consists of itself, and its neighboring entities, e_3 and e_2, connected via relations r_1 and r_3, respectively. So, the context of an entity forms a subgraph of the input KG. The context of r_2 consists of itself and other neighboring relations: r_1, r_3, r_4, r_5, r_6, r_8, and r_9. They are connected via entities of specific types, $T(e)$ denotes the type of entity e. For example, in *IMDB* the entity types could be actors, movies, directors, etc. Since each relation occurs many times in a KG, the same pair of relations can be connected via multiple entity types. Therefore, the context of a relation could be a multi-graph.

Since contexts of entities and relations are (multi)graphs having edge relations, the problem of context encoding is converted to relational subgraph encoding via R-GCN as follows.

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in N_i^r}\frac{1}{|N_i^r|}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right) \tag{1}$$



(a) Context of entity e_4 (b) Context of relation r_2

Fig. 3. Context of entity/ relation. $T(e)$ denotes the type of entity e.

Here, $h_i^{(l)}$ is the hidden state of node v_i (from the context graph) at the l-th layer. N_i^r denotes the set of neighbor indices of node i under relation $r \in R$. Equation 1 aggregates transformed feature vectors of neighboring nodes and itself in a relation-specific manner and through a normalized sum, and then passes through an element-wise activation function $\sigma = \text{ReLU}$. We employ two R-GCNs, one for entities and another for relations. We consider one hidden layer in both R-GCNs since one hidden layer achieves better results and a good trade-off between accuracy and efficiency based on our experiments.

Joint Embedding. We aggregate the contextual subgraph embeddings of entities and relations with their knowledge embeddings via a gate strategy [35], and form the joint embedding $\boldsymbol{o}^\star$ of each object in the KG.

$$\boldsymbol{o}^\star = \boldsymbol{g} \odot \boldsymbol{o}^k + (\boldsymbol{1} - \boldsymbol{g}) \odot \boldsymbol{sg}(o) \tag{2}$$

where $\odot$ means element-wise multiplication, o is an entity or a relation, $\boldsymbol{o}^k$ is its knowledge embedding, $\boldsymbol{sg}(o)$ is the vector representation of the context of o, $\boldsymbol{g} = logistic(\tilde{\boldsymbol{g}})$ ensures that the value of each element in the gate vector $\boldsymbol{g}$ is in $[0, 1]$, and $\tilde{\boldsymbol{g}} \in \mathbb{R}^d$ is a parameter vector. Note that all entities share a $\boldsymbol{g}$ denoted as $\boldsymbol{g}^e$, and all relations share another $\boldsymbol{g}$ denoted as $\boldsymbol{g}^r$.

Embedding Learning. Given a triple (h, r, t), we define a score function based on a translation operation as follows:

$$f(h, r, t) = \|\boldsymbol{h}^\star + \boldsymbol{r}^\star - \boldsymbol{t}^\star\|_2^2 \tag{3}$$

where $\boldsymbol{h}^\star$, $\boldsymbol{r}^\star$ and $\boldsymbol{t}^\star$ are computed by Equation 2.

For training, we define a margin-based loss function:

$$\mathcal{L} = \sum_{(h,r,t)\in S} \sum_{(h',r,t')\in S'} \max(0, f(h, r, t) + \gamma - f(h', r, t')) \tag{4}$$

where γ is the margin, S is the set of correct triples, and S' is the set of incorrect triples. Since a KG only contains correct triples, we corrupt them by replacing head entities or tail entities to build S'. The replacement process follows the Bernoulli sampling method [34]. During training, the knowledge embeddings and context embeddings of all entities and relations are initialized following the

uniform distribution $U(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ [5], where k is the number of dimensions for embeddings. All parameters including embeddings are updated using the Adam optimizer in each minibatch.

4 Online Embedding Updates

KGs are updated over time with addition and deletion of triples [26, 19, 12, 9]. KG embeddings should also be updated accordingly in an online manner. Following the inductive learning, we keep all the learnt parameters in R-GCNs and the gate strategy unchanged. Contextual element embeddings of existing entities and relations also remain the same. After a KG update, for many entities and relations, their contexts remain unaffected, so their contextual subgraph embeddings would remain uninterrupted. Thus, with existing knowledge embeddings of such entities and relations, corresponding triples would satisfy: $\boldsymbol{h}^\star + \boldsymbol{r}^\star \approx \boldsymbol{t}^\star$. Hence, we also keep the knowledge embeddings of existing entities and relations unchanged so long as their contexts are unchanged.

For existing entities and relations with changed contexts, we do the following. Recall that the joint embedding of each entity or relation must approximately satisfy the translation relations in our model. This joint embedding depends on both the knowledge embedding and the contextual subgraph embedding. Consider an entity t that has changed context, so the vector representation of t's context (i.e., $\boldsymbol{sg}(t)$, a combination of context element embeddings of its neighbor entities, computed by the R-GCN) would change to reflect this update. We next adjust the knowledge embedding of t (i.e., $\boldsymbol{t}^k$) to update its joint embedding $\boldsymbol{t}^\star$, so that this updated $\boldsymbol{t}^\star$ approximately satisfies the translations in the modified graph. In practise, we find that the modifications happened in the joint embeddings are generally small due to local updates in a KG, which explains why our method is effective in approximately preserving the translations in the modified KG. In future, it would be interesting to analyze the errors due to approximately maintaining the translations in our online updates.

Finally, we also learn knowledge embeddings and contextual element embeddings of emerging entities and relations. In summary, we only need to compute the knowledge embeddings and contextual element embeddings of emerging entities and relations, as well as the knowledge embeddings of existing entities and relations with changed contexts, so that the joint embeddings of existing entities and relations with changed contexts also approximately satisfy the translations in the modified graph. With above strategies, the impacts of a KG update will be limited to a certain region, especially it will not affect the triples where the contexts of entities and relations are unchanged, which greatly improves efficiency. This also shows the benefit of having two separate vectors for every entity or relation in OUKE: knowledge embedding and context embedding.

Space Complexity. Let us denote by e, e_t, and r the total number of entities, entity types, and relations in the input KG. Assume each minibatch, on average, consists of e_b entities and r_b relations. We define the size of the adjacency matrix in the R-GCN for entities as $e_b \times e_b$ and that in the R-GCN for relations as $r_b \times r_b$ (as training happens in minibatches).

Table 1. Characteristics of datasets.

Datasets	#Entities (Avg.)	#Edges (Avg.)	#Relations (Avg.)	#Add Triples (Avg.)	#Del Triples (Avg.)	#Train (Avg.)	#Valid	#Test
YAGO-3SP	27 009	130 757	37	950	150	124 757	3 000	3 000
IMDB-3SP	169 146	524 296	14	9 181	521	518 296	3 000	3 000

Suppose the dimension of the embedding space is k, the R-GCNs for entities and relations have l_e and l_r hidden layers, respectively, thus we have $\mathcal{O}((l_e \cdot r + l_r \cdot e_t)k \times k)$ weight matrices. In the gate strategy, all entities, relations, and weights also correspond to k-dimensional parameter vectors. In addition, each entity and each relation has two vector representations, so we totally have $(2e + 2r)k$-dimensional vectors to represent entities and relations. In summary, the space complexity of OUKE is $\mathcal{O}(e_b^2 + r_b^2 + (l_e \cdot r + l_r \cdot e_t)k^2 + (e + r)k)$. As we discussed in §3, the number of hidden layers $l_e = l_r = 1$ in our implementation.

Time Complexity. For learning from scratch and online learning, we analyze their time complexities of updating parameters when given a triple pair $\{(h, r, t), (h', r, t')\}$ in a minibatch.

Since we adopt the negative sampling strategy proposed in [34], in the given triple pair, if $h \neq h'$, then $t = t'$; if $h = h'$, then $t \neq t'$. When given a triple pair in a minibatch, updating knowledge embeddings and context subgraph embeddings of three entities and a relation requires $\mathcal{O}(8k)$, where k is the dimension of the embedding space. Besides, updating the parameters in two R-GCNs and the gate strategy requires $\mathcal{O}((l_e \cdot r + l_r \cdot e_b)k^2)$ and $\mathcal{O}(2k)$, respectively. Since learning from scratch needs to update all of the above parameters, the total time complexity of updating parameters for given a triple pair $\{(h, r, t), (h', r, t')\}$ is $O(10k + (l_e \cdot r + l_r \cdot e_b)k^2)$.

In online learning, all parameters in two R-GCNs and the gate strategy are unchanged, and we only update the knowledge embeddings and context subgraph embeddings of emerging entities and relations, as well as the knowledge embeddings of existing entities and relations with changed contexts. Thus, the total time complexity is $O(\mu k)$ $(1 \leq \mu \leq 8)$, which reflects the efficiency of online learning. Besides, online learning also has much fewer triples to train when comparing with learning from scratch (as we demonstrated in Example 1), which is another reason for efficiency improvement.

5 Experimental Results

We conduct experiments to demonstrate the effectiveness and efficiency of OUKE in link prediction over evolving KGs. We employ PyTorch v1.1 deep learning library to implement OUKE and competitors. We perform experiments on a single machine with 256GB, 2.2GHz Intel(R) Xeon(R) CPU E5-2698 v4 processor. Our GPU platform is Tesla V100 (16GB VRAM) with CUDA 9.0.

5.1 Experimental Setup

Datasets. We use two datasets from real-world KGs, having three snapshots. We split each snapshot into a training set, a validation set, and a test set. The

Table 2. Accuracy results on link prediction. Lower values of MR and higher values of Hits@K (defined in §5.1) indicate better accuracy.

		YAGO-3SP		IMDB-3SP	
		MR	Hits@10	MR	Hits@10
Snapshot1	GAKE	2984	0.237	5798	0.213
	R-GCN	416	0.225	2646	0.112
	TransE	710	**0.311**	**2408**	0.352
	puTransE	938	0.262	3518	0.188
	OUKE-LFS	**511**	0.296	2987	**0.349**
Snapshot2	GAKE	3012	0.218	5542	0.218
	R-GCN	391	0.237	2482	0.123
	TransE	745	**0.314**	**2314**	0.340
	puTransE	897	0.259	3506	0.182
	OUKE-LFS	**554**	0.304	3018	**0.364**
	OUKE-OL	604	0.292	3133	0.329
Snapshot3	GAKE	2873	0.220	5623	0.219
	R-GCN	397	0.270	2871	0.052
	TransE	703	**0.325**	**2575**	0.351
	puTransE	1082	0.247	3522	0.187
	OUKE-LFS	**537**	0.305	3091	**0.361**
	OUKE-OL	599	0.281	3299	0.334

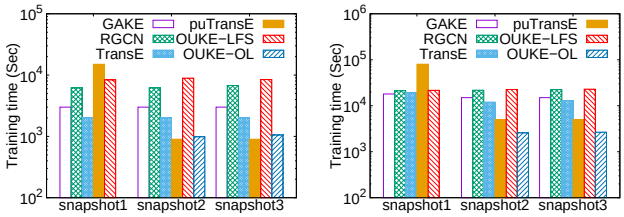
three snapshots share the same validation set and test set, in which triples are unchanged in these snapshots.

(1) YAGO-3SP. YAGO (http://yago-knowledge.org/) is a large-scale encyclopedic KG constructed from Wikipedia, WordNet, and GeoNames. Different versions of YAGO were published at various times. We extract subsets of YAGO2.5, YAGO3, and YAGO3.1 as three snapshots of our dataset YAGO-3SP.

(2) IMDB-3SP. The Internet Movie Database (IMDB) is a KG consisting of movies, TV series, actors, directors, etc., and their relations. IMDB provides daily dumps (https://datasets.imdbws.com/), and we download them from January 22 to January 24 in 2019 as three snapshots.

In Table 1, for each dataset we recorded: 1) the average numbers of entities (#Entities (Avg.)), edges (#Edges (Avg.)), and relations (#Relations (Avg.)) in different snapshots, respectively; 2) the average numbers of added triples (#Add Triples (Avg.)) and deleted triples (#Del Triples (Avg.)) between snapshots, respectively; 3) the average number of triples in the training sets (#Train (Avg.)) of different snapshots, the number of triples in the validation set (#Validate), and the number of triples in the test set (#Test).

Competitors. (1) puTransE [27] is the only existing model supporting online embedding learning for dynamic KGs. puTransE learns embeddings of entities and relations from local parts of a KG, so it avoids retraining on the entire KG, but cannot preserve the global structural information of the KG in the learnt embedding. **(2) TransE** [5] is a static KG embedding method using translation operations on entities and relations. **(3) GAKE** [10] simultaneously models triples and structural contexts in embedding learning on static KGs. **(4) R-**



(a) YAGO-3SP (b) IMDB-3SP

Fig. 4. Efficiency results on link prediction.

GCN [25] learns embedding considering multiple relations between entities in a static KG. Methods (2)-(4) learn embeddings only from scratch over static KGs.

Accuracy metrics for link prediction. For each triple (h, r, t) in the test set, we replace the head entity h (or tail entity t) with each entity e in the snapshot to construct a triple (e, r, t) (or (h, r, e)), and rank all e based on the score calculated by the scoring function. If a constructed triple occurs in the training set, then the corresponding entity e will not participate in the ranking process, because training data cannot be used in testing. Based on such ranking results, we can get the rank of the original correct entity in each test triple, and we use the following evaluation metrics: **(1) Mean Rank (MR)**: the average rank of all head entities and tail entities in test triples. **(2) Hits@K**: the proportion of the ranks not larger than K for all head entities and tail entities in test triples. Lower values of MR and higher values of Hits@K indicate better accuracy.

5.2 Accuracy and Efficiency Results

Our empirical results in Table 2 indicate that our learning from scratch approach (OUKE-LFS) produces the best or second-best MR and Hits@10 scores over our datasets. Our online learning method (OUKE-OL) is quite competitive as well, however its accuracy is lower than OUKE-LFS because OUKE-OL retrains on a limited number of triples during online learning. On the other hand, OUKE-OL is faster. For instance, over IMDB-3SP, OUKE-OL is 2-3 orders of magnitude faster than static embedding methods, and up to an order of magnitude faster than puTransE (Figure 4). Considering accuracy and efficiency trade-offs, we find OUKE-OL to be the most suitable for online updates of dynamic KG embedding.

6 Conclusions

We presented a context-aware dynamic knowledge graph (KG) embedding method OUKE, which not only learns embeddings from scratch, but also supports online embedding updates. Compared with state-of-the-art static and dynamic KG embedding techniques on dynamic datasets, OUKE has comparable effectiveness and much better efficiency in online learning. In future, it would be interesting to boost the accuracy of OUKE by improving its context embedding phase, analyzing its robustness with respect to repeated and batch updates, and deploying OUKE in more downstream applications such as KG-based question answering.

Acknowledgement

Arijit Khan is supported by MOE Tier1 and Tier2 grants RG117/19, MOE2019-T2-2-042, and a Delta Corporate Lab Grant SLE-RP8.

References

1. Source for IMDB dataset. https://www.imdb.com/interfaces/.
2. Use Deep Search to Explore the COVID-19 Corpus. https://www.research.ibm.com/covid19/deep-search/.
3. M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, M. Galkin, S. Sharifzadeh, A. Fischer, V. Tresp, and J. Lehmann. 2020. Bringing Light Into the Dark: A Large-scale Evaluation of Knowledge Graph Embedding Models Under a Unified Framework. CoRR abs/2006.13365 (2020).
4. K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In SIGMOD.
5. A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. 2013. Translating Embeddings for Modeling Multi-Relational Data. In NIPS.
6. X. Chen, M. Chen, C. Fan, A. Uppunda, Y. Sun, and C. Zaniolo. 2020. Multilingual Knowledge Graph Completion via Ensemble Knowledge Transfer. In EMNLP (Findings).
7. T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In AAAI.
8. S. S. Dasgupta, S. N. Ray, and P. Talukdar. 2018. HyTE: Hyperplane-based Temporally aware Knowledge Graph Embedding. In EMNLP.
9. X. L. Dong. 2018. Challenges and Innovations in Building a Product Knowledge Graph. In KDD.
10. J. Feng, M. Huang, Y. Yang, and X. Zhu. 2016. GAKE: Graph Aware Knowledge Embedding. In COLING.
11. A. Gyrard, M. Gaur, K. Thirunarayan, A. P. Sheth, and S. Shekarpour. 2018. Personalized Health Knowledge Graph. In CKGSemStats@ISWC.
12. S. Hellmann, C. Stadler, J. Lehmann, and S. Auer. 2009. DBpedia Live Extraction. In OTM Conferences.
13. J. Hoffart, F. M Suchanek, K. Berberich, and G. Weikum. 2013. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. Artificial Intelligence 194 (2013), 28–61.
14. X. Huang, J. Zhang, D. Li, and P. Li. 2019. Knowledge Graph Embedding Based Question Answering. In WSDM.
15. J. Jin, J. Luo, S. Khemmarat, and L. Gao. 2017. Querying Web-Scale Knowledge Graphs Through Effective Pruning of Search Space. IEEE Trans. Parallel Distributed Syst. 28(8): 2342-2356 (2017).
16. T. N. Kipf and M. Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In ICLR.
17. A. Khan, Y. Wu, C. C Aggarwal, and X. Yan. 2013. NeMa: Fast Graph Search with Label Similarity. PVLDB 6, 3 (2013), 181–192.
18. J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. 2015. DBpedia - A Large-Scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web 6, 2 (2015), 167–195.

19. X. Lin, H. Li, H. Xin, Z. Li, and L. Chen. 2020. KBPearl: A Knowledge Base Population System Supported by Joint Entity and Relation Linking. PVLDB 13, 7 (2020), 1035–1049.
20. Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion, in AAAI.
21. S. Liao, S. Liang, Z. Meng, and Q. Zhang. 2021. Learning Dynamic Embeddings for Temporal Knowledge Graphs. In WSDM.
22. T. M. Mitchell, W. W. Cohen, E. R. Hruschka Jr., P. P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. D. Mishra, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. A. Platanios, A. Ritter, M. Samadi, B. Settles, R. C. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. 2018. Never-Ending Learning. Commun. ACM 61, 5 (2018), 103–115.
23. N. Nakashole, T. Tylenda, and G. Weikum. 2013. Fine-grained Semantic Typing of Emerging Entities. In ACL.
24. M. Nickel, V. Tresp, and H.-P. Kriegel. 2011. A Three-Way Model for Collective Learning on Multi-Relational Data. In ICML.
25. M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M.Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In ESWC.
26. J. Shin, S. Wu, F. Wang, C. D. Sa, C. Zhang, and C. Ré. 2015. Incremental Knowledge Base Construction Using DeepDive. PVLDB 8, 11 (2015), 1310– 1321.
27. Y. Tay, A. T. Luu, and S. C. Hui. 2017. Non-Parametric Estimation of Multiple Embeddings for Link Prediction on Dynamic Knowledge Graphs. In AAAI.
28. R. Trivedi, H. Dai, Y. Wang, and L. Song. 2017. Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs. In ICML.
29. R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In ICLR.
30. T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In ICML.
31. M. Wang, L. Qiu, and X. Wang. 2021. A Survey on Knowledge Graph Embeddings for Link Prediction. Symmetry 13, 3 (2021), 485.
32. Q.Wang, Z.Mao, B.Wang, and L. Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. IEEE Transactions on Knowledge and Data Engineering 29, 12 (2017), 2724–2743.
33. Y. Wang, A. Khan, T. Wu, J. Jin, and H. Yan. 2020. Semantic Guided and Response Times Bounded Top-k Similarity Search over Knowledge Graphs. In ICDE.
34. Z. Wang, J. Zhang, J. Feng, and Z. Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In AAAI.
35. J. Xu, X. Qiu, K. Chen, and X. Huang. 2017. Knowledge Graph Representation with Jointly Structural and Textual Encoding. In IJCAI.
36. B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In ICLR.
37. Y. Zhao, A. Zhang, R. Xie, K. Liu, and X. Wang. 2020. Connecting Embeddings for Knowledge Graph Entity Typing. In ACL.
38. F. Zhang, N. Jing Yuan, D. Lian, X. Xie, and W.-Y. Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In KDD.
39. D. Zhu, P. Cui, Z. Zhang, J. Pei, and W. Zhu. 2018. High-Order Proximity Preserved Embedding for Dynamic Networks. IEEE Transactions on Knowledge and Data Engineering 30, 11 (2018), 2134–2144.