RESEARCH PAPER

Learning to Complete Knowledge Graphs with Deep Sequential Models

Lingbing Guo, Qingheng Zhang, Wei Hu†, Zequn Sun & Yuzhong Qu

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

Keywords: Knowledge graph; entity prediction; triple prediction; recurrent neural network

Citation: L. Guo, Q. Zhang, W. Hu, Z. Sun, & Y. Qu. Learning to complete knowledge graphs with deep sequential models. Data Intelligence 1(2019), 224-243. doi: 10.1162/dint_a_00016
Received: December 19, 2018; Revised: April 22, 2019; Accepted: May 5, 2019

ABSTRACT

Knowledge graph (KG) completion aims at filling the missing facts in a KG, where a fact is typically represented as a triple in the form of (*head*, *relation*, *tail*). Traditional KG completion methods compel two-thirds of a triple provided (e.g., head and relation) to predict the remaining one. In this paper, we propose a new method that extends multi-layer recurrent neural networks (RNNs) to model triples in a KG as sequences. It obtains state-of-the-art performance on the common entity prediction task, i.e., giving head (or tail) and relation to predict the tail (or the head), using two benchmark data sets. Furthermore, the deep sequential characteristic of our method enables it to predict the relations given head (or tail) only, and even predict the whole triples. Our experiments on these two new KG completion tasks demonstrate that our method achieves superior performance compared with several alternative methods.

1. INTRODUCTION

Knowledge graphs (KGs), such as DBpedia [1] and Freebase [2], often use triples, in the form of (*h*, *r*, *t*), to record billions of real-world facts, where *h*, *t* denote entities and *r* denotes a relation between *h* and *t*. Despite the enormous effort put into KG creation and maintenance, current KGs are still far from complete. KG completion is proposed to deal with this problem. Previous methods focus on a general task called entity prediction (also known as link prediction) [3, 4], which requires one to complete a triple in a KG by predicting *t* given (*h*, *r*, ?) or predicting *h* given (?, *r*, *t*). The left-most part of Figure 1 shows a commonly-used model for tail entity prediction. Input $\boldsymbol{h}$, $\boldsymbol{r}$ is firstly projected by some vectors or matrices, where the

† Corresponding author: Wei Hu (Email: whu@nju.edu.cn; ORCID: 0000-0003-3635-6335).

bold letters denote the corresponding embeddings of h and r, and then combined to a continuous representation $\boldsymbol{o}_t$ for predicting t.

Although existing methods have shown good performance on entity prediction, in real world they may still be inadequate to complete a KG. Let us assume that there is a method which can effectively complete an entity h given a relation r explicitly. But if we do not provide any relations, this method would be incompetent to enrich h, since it is incapable of knowing which relation should be used to complete this entity. An alternative way is to iterate over all relations in the KG, but such process is time-consuming and error-prone. We argue that this may prevent the existing methods from being useful in the real world.

The recurrent neural network (RNN) is a neural sequence model, which has achieved state-of-the-art performance on many natural language processing (NLP) tasks, such as language modeling, speech recognition and machine translation [5, 6]. A triple in a KG can be considered to be a sequence of length 3, which inspires us to use RNNs to model KGs. However, we are still challenged by the following two obstacles: (i) triples are not natural language. They model complex structures of KGs with a fixed expression (h, r, t). Such short sequences may be insufficient to provide adequate context for prediction, while it is time-consuming and difficult to construct valuable long sequences from a huge number of paths in KGs; and (ii) relations and entities are elements of two different types, and they appear in triples in a fixed order. It is over-simplified to treat them as the same type.

In this paper, we propose a new method, which employs RNNs to model triples in a KG as sequences. We first design **BSKG**, a basic sequential model for KGs, as an initial version to illustrate our method (Figure 1). We denote an RNN cell by c, which receives its previous hidden state and the current element as input to predict the next. The cell in the entity layer processes entity embeddings like $\boldsymbol{h}$, while the cell in the relation layer processes relation embeddings like $\boldsymbol{r}$. In BSKG, we use one cell to sequentially process all input elements, and thus $\boldsymbol{h}$, $\boldsymbol{r}$ are fed to the same cell c to obtain their respective output. Then, we use $\boldsymbol{o}_r$ to predict relations of h and $\boldsymbol{o}_t$ to predict tails of $h \rightarrow r$.

However, BSKG lacks effectiveness to model complex structures, because it only uses a single RNN cell to process all input sequences. To improve the performance on complex KGs, a few existing methods like TransH [7] and TransR [8] suggest projecting entities by some vectors or matrices before combination. Different from them, we do not choose to extend BSKG by adding such a layer, since it requires creation of variables for each relation, which consumes considerable resources and may also make the models hard to converge. Instead, we propose **DSKG** in this paper, a deep sequential model that extends multi-layer RNNs to model KGs. DSKG can smoothly model complex structures, since each cell in DSKG does not need to output an explicit prediction result, but gives its own comprehension by adding or removing information from the hidden state and conveys this processed hidden state to its next cell. Therefore, the original input is continually refined by each cell, and the complex features can be fluently processed.

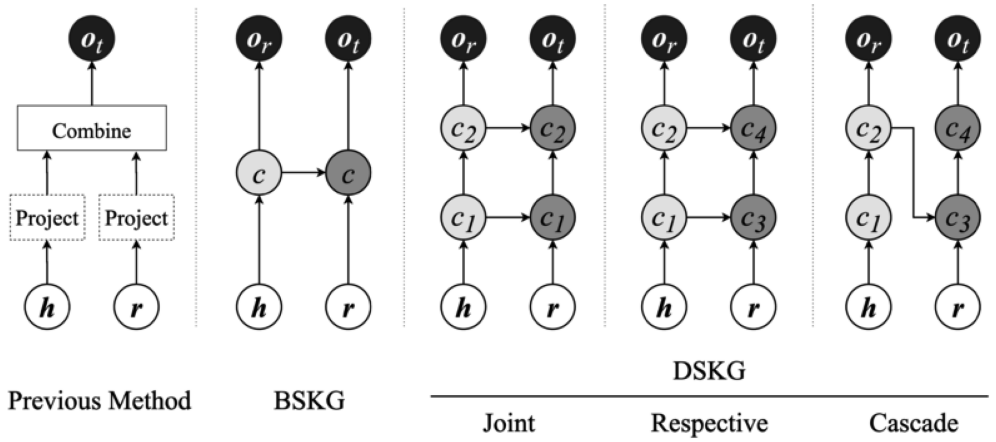


Figure 1. Different models for tail entity prediction. Note: White and black circles denote input and output, respectively. For BSKG and DSKG, light gray circles denote cells in the entity layer and dark gray circles denote cells in the relation layer.

Specifically, we design three different strategies to integrate the RNN cells in DSKG. The right-most part of Figure 1 shows their two-layer examples.

- The **Joint** strategy is the most prevalent way used in the NLP area. RNN cells are reused in different layers like BSKG, but the output hidden state of each cell is not only recurrently fed to itself next time, but also sequentially conveyed to its next cell.
- The **Respective** strategy uses independent RNN cells for the entity layer and the relation layer, that is to say, c_1, c_2, c_3 and c_4 are all different. We want this strategy to achieve better performance when relations are diverse and complex.
- The **Cascade** strategy also uses different RNN cells for the entity layer and the relation layer. Instead of parallel connecting the two layers, we decide to link the end cell of the entity layer to the first cell of the relation layer. As a result, each cell in this strategy can concentrate more on its current work.

The main contributions of this paper are listed as follows:

- We introduce a new method for KG completion, which extends multi-layer RNNs to model triples in a KG as sequences of length 3. Three distinct strategies are proposed to integrate RNN cells and show their different characteristics in our experiments.
- We design two new KG completion tasks, namely relation prediction and triple prediction, as complements to the entity prediction task. The relation prediction task aims to predict relations only given a head (or tail) entity as input.[①] The triple prediction task is to predict the whole triples only given a head entity.

[①] Note that the relation prediction task is different from the task to predict relations given a head entity and a tail entity. The latter one is sometimes called relationship prediction.

- Our experimental results show that our method achieves state-of-the-art performance for entity prediction on the benchmark data sets based on Freebase and WordNet. It also achieves promising results on the new relation prediction and triple prediction data sets.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 describes the details of our method. Section 4 presents the experimental results. Finally, we conclude this paper in Section 5.

2. RELATED WORK

We divide existing models into two sub-areas: translational models and non-translational models. We summarize several representative models in Table 1. Our models are also added for comparison.

Table 1. Comparison of existing models and ours.

Models	Preprepared data	Energy functions	Notation explanations
TransE		$\|\|\boldsymbol{h}+\boldsymbol{r}-\boldsymbol{t}\|\|_{L^*}$	
TransH		$\|\|(\boldsymbol{h}-\boldsymbol{w}_r^T\boldsymbol{h}\boldsymbol{w}_r)+\boldsymbol{r}-(\boldsymbol{t}-\boldsymbol{w}_r^T\boldsymbol{t}\boldsymbol{w}_r)\|\|_{L^*}$	$\boldsymbol{w}_r$: relation-specific normalization vector
TransR	TransE embeddings	$\|\|\boldsymbol{W}_r\boldsymbol{h}+\boldsymbol{r}-\boldsymbol{W}_r\boldsymbol{t}\|\|_{L^*}$	$\boldsymbol{W}_r$: relation-specific matrix
STransE	TransE embeddings	$\|\|\boldsymbol{W}_{r,1}\boldsymbol{h}+\boldsymbol{r}-\boldsymbol{W}_{r,2}\boldsymbol{t}\|\|_{L^*}$	$W_{r,1}$, $W_{r,2}$: relation-specific matrices
PTransE	TransE embeddings; paths	$\boldsymbol{h}+\boldsymbol{r}-\boldsymbol{t}+\frac{1}{\|P(h,t)\|}\sum_{\boldsymbol{p}\in P(h,t)}(\boldsymbol{p}-\boldsymbol{r})$	$P(h, t)$: path set of $(h, \ldots, t)$
NTN	Word embeddings	$\boldsymbol{u}_r^T\tanh(\boldsymbol{h}^T\boldsymbol{M}_r\boldsymbol{t}+\boldsymbol{W}_{r,1}\boldsymbol{h}+\boldsymbol{W}_{r,2}\boldsymbol{t}+\boldsymbol{b}_r)$	$\boldsymbol{u}_r$, $\boldsymbol{M}_r$, $\boldsymbol{W}_{r,1}$, $\boldsymbol{W}_{r,2}$, $\boldsymbol{b}_r$: relation-specific variables
DISTMULT		$\boldsymbol{h}^T\boldsymbol{W}_r\boldsymbol{t}$	$\boldsymbol{W}_r$: relation-specific diagonal matrix
NLFeat	Node and link features	$\boldsymbol{\Phi}_{i,j,k}^T\boldsymbol{\Theta}$	$\boldsymbol{\Phi}_{i,j,k}^T$: feature vector; $\boldsymbol{\Theta}$: weight vector
ConvE		$g\left(\text{vec}\left(g\left(\text{concat}\left(\boldsymbol{h},\boldsymbol{r}\right)*\Omega\right)\right)\boldsymbol{W}\right)\cdot\boldsymbol{t}$	g: activation function; Ω convolutional layer
ConvKB		$\text{concat}\left(g\left(\left[\boldsymbol{h},\boldsymbol{r},\boldsymbol{t}\right]*\Omega\right)\right)\cdot\boldsymbol{w}$	g: activation function; Ω convolutional layer
BSKG		$L_{r(h)}+L_{t(h,r)}$	$c_1^h\rightarrow\{c_1^r\}$
DSKG (joint)			$c_1^h\rightarrow\{c_2^h,c_1^r\},c_2^h\rightarrow\{c_2^r\},c_1^r\rightarrow\{c_2^r\}$
DSKG (respective)			$c_1^h\rightarrow\{c_2^h,c_3^r\},c_2^h\rightarrow\{c_4^r\},c_3^r\rightarrow\{c_4^r\}$
DSKG (cascade)			$c_1^h\rightarrow\{c_2^h\},c_2^h\rightarrow\{c_3^r\},c_3^r\rightarrow\{c_4^r\}$

2.1 Translational Models

Perhaps, TransE [4] is the most well-known embedding model for KG completion. It represents entities and relations as k-dimensional vectors in a unified space, and models a triple (h, r, t) as $\boldsymbol{h} + \boldsymbol{r} \approx \boldsymbol{t}$. TransE works well for one-to-one relationship, but it fails to model more complex (e.g., one-to-many) relationships. TransH [7] resolves this problem by regarding each relation r as a vector on a hyperplane whose normalization vector is $\boldsymbol{w}_r$. It projects entity embeddings $\boldsymbol{h}$, $\boldsymbol{t}$ to this hyperplane by $\boldsymbol{w}_r$, and uses the same energy function as TransE for training. TransR [8] uses relation-specific matrices to project entities. It creates a project matrix $\boldsymbol{W}_r$ for each relation r and projects $\boldsymbol{h}$, $\boldsymbol{t}$ by $\boldsymbol{W}_r$. TransR also employs the same energy function for training.

To extend the above models, STransE [9] learns two project matrices $\boldsymbol{W}_{r,1}$, $\boldsymbol{W}_{r,2}$ for each relation r, where $\boldsymbol{W}_{r,1}$ is used for projecting $\boldsymbol{h}$ and $\boldsymbol{W}_{r,2}$ is for projecting $\boldsymbol{t}$. TranSparse [10] is similar to STransE, but it uses a more complex method to project $\boldsymbol{h}$, $\boldsymbol{t}$. PTransE [11] is also a TransE-like model. It uses additional path information for training. For example, if there exist two triples (e_1, r_1, e_2), (e_2, r_2, e_3), which can be regarded as a path in a KG, and another triple (e_1, r_x, e_3) holds simultaneously, then the path $e_1 \rightarrow r_1 \rightarrow e_2 \rightarrow r_2 \rightarrow e_3$ is a valuable path recorded as (e_1, r_1, r_2, e_3). However, preparing desirable paths needs to iterate over all possible paths, and thus this process may consume quadratic resources. We argue that PTransE and many path-based models may be inefficient to model large KGs.

All the aforementioned models choose to minimize an energy function that is used in or similar to TransE. Moreover, except TransE and TransH, the remaining models require pre-trained entity and relation embeddings from TransE as initial input, which increases the training expense and also blurs their actual performance.

2.2 Non-translational Models

There also exist a number of models that are very different from the translational models. A neural tensor network (NTN) [12] defines a different loss function and creates many variables for each relation. This makes NTN unsuitable for KGs having plenty of relations. Furthermore, NTN requires word embeddings as initial input, which severely increases the training expense. DISTMULT [13] is as simple as TransE, but it employs a completely different energy function. More specifically, it is based on the Bilinear model [14] and represents each relation as a diagonal matrix. Node+LinkFeat (abbr. NLFeat) [15] can also be regarded as a path-based model like PTransE, but it only needs to extract paths of length 1 for constructing node and link features. For example, if there exists a triple (e_i, r_k, e_j) in a KG, and (e_i, r', e_j) holds at the same time, then a binary feature is constructed as $\mathbf{1}(r \,\&\, r_k)$. Although using paths of length 1 to construct features is much easier than using longer paths, it still consumes considerable resources for large KGs.

Recently, the deep neural networks have received much attention in KG completion. Instead of using simple algebraic operations, the deep neural models stack a group of different neural layers to model complex patterns in KGs. R-GCN [16] leverages the conventional graph convolutional networks [17] and extends it to multi-relational data like KGs. ConvE [18] applies convolutional neural networks (CNNs) [19] for KG completion. ConvKB [20] is also based on CNNs but implemented with a novel fashion.

Other methods like [21, 22] use extra data that cannot be extracted from the original training data, such as text corpora or entity descriptions. Due to the focus of this paper, we do not consider them currently.

3. BASIC AND DEEP SEQUENTIAL MODELS

In this section, we first describe our basic sequential model BSKG. Then, we present our deep sequential model DSKG with three different integrating strategies. Finally, two optimization methods useful for accelerating convergence and preventing over-fitting are introduced.

3.1 Basic Sequential Model

We start with the basic sequential model, which has only one single RNN cell. Let T, E, R be the sets of triples, entities and relations in a KG, respectively. Given a triple $(h, r, t) \in T$, we represent h, $t \in E$ and $r \in R$ all as k-dimensional vectors $\boldsymbol{h}$, $\boldsymbol{t}$, $\boldsymbol{r}$, and use the tail entity prediction task for example to write the basic RNN layer as follows:

$$\begin{aligned} \boldsymbol{s}_h &= c(\boldsymbol{h}, \boldsymbol{s}_0) \\ \boldsymbol{s}_r &= c(\boldsymbol{r}, \boldsymbol{s}_h), \end{aligned} \tag{1}$$

where c denotes an RNN cell, which receives its previous hidden state and the current element as input, and outputs the processed hidden state. $\boldsymbol{s}_h$, $\boldsymbol{s}_r$ denote the processed hidden states of input $\boldsymbol{h}$, $\boldsymbol{r}$, respectively. $\boldsymbol{s}_0$ denotes the zero hidden state for initialization.

There are a variety of candidate RNN cells that can be used to implement our model, e.g., the gated recurrent unit (GRU) cell [23] or the long short-term memory (LSTM) cell [24]. For simplicity, we do not discuss the details here, but use

$$\begin{aligned} \boldsymbol{o}_h &= f(\boldsymbol{s}_h, c) \\ \boldsymbol{o}_r &= f(\boldsymbol{s}_r, c), \end{aligned} \tag{2}$$

to abstractly represent the operations for calculating the output of RNN cells.

Then, we can respectively use $\boldsymbol{o}_h$, $\boldsymbol{o}_r$ to predict the relations of h and the tail entities of $h \to r$ as follows:

$$\begin{aligned} \boldsymbol{p}_r &= \boldsymbol{W}_1 \boldsymbol{o}_h + \boldsymbol{b}_1 \\ \boldsymbol{p}_t &= \boldsymbol{W}_2 \boldsymbol{o}_r + \boldsymbol{b}_2, \end{aligned} \tag{3}$$

where $\boldsymbol{W}_1$ denotes the weight matrix of relation prediction, which has a shape of $|R| \times k$. $\boldsymbol{b}_1$ denotes the bias vector. $\boldsymbol{W}_2$, $\boldsymbol{b}_2$ are defined similarly. Therefore, $\boldsymbol{p}_r$, $\boldsymbol{p}_t$ can be directly regarded as the unscaled probabilities for relation prediction and tail entity prediction, respectively.

We respectively calculate the sampled softmax cross-entropy relation loss L_r and entity loss L_t [25] for the unscaled probabilities p_r, p_t as follows:

$$\begin{aligned} L_r &= -\sum_{i}^{|\boldsymbol{p}_r|} y_i \log(n_i), \\ &= -\sum_{i}^{|\boldsymbol{p}_r|} y_i \log\left(\frac{\exp(p_i)}{\sum_{\tilde{r}\in\{r\}\cup N_r}\exp(p_{\tilde{r}})}\right) \\ &= -\log\left(\frac{\exp(p_r)}{\sum_{\tilde{r}\in\{r\}\cup N_r}\exp(p_{\tilde{r}})}\right), \\ L_t &= -\log\left(\frac{\exp(p_t)}{\sum_{\tilde{t}\in\{t\}\cup N_t}\exp(p_{\tilde{t}})}\right), \end{aligned} \tag{4}$$

where n_i denotes the normalized probability of the i-th relation with softmax function. y_i is the label for n_i. Due to the fact that the r-th relation (the t-th entity) is the only correct one in this case, we can remove the zero components in the sum. N_r, N_t denote the sets of sampled negative labels for relation prediction and entity prediction, respectively.

There may exist multiple correct labels for both relation prediction and entity prediction, so in this paper we only use the current input triple to provide the correct labels for both relation prediction and entity prediction. For example, let (h_0, r_0, t_0) be the current training triple and $t_1, \ldots, t_m$ be other correct labels for $(h_0, r_0, ?)$, as these triples (h_0, r_0, t_i), $i = 1, \ldots, m$ also exist in the training data. But we only consider t_0 as the exclusively right label and avoid adding it during sampling negative labels. We propose this method since it makes training much faster and does not need to prepare the label information for multi-label classification. This is also because the number of possible negative labels is much larger than that of correct ones.

Finally, we can jointly minimize both two losses L_r, L_t, or just minimize L_t, as the final loss for training:

$$\begin{aligned} L_j &= L_r + L_t \\ L_e &= L_t, \end{aligned} \tag{5}$$

where L_j denotes the joint loss, and L_e denotes the entity-only loss. Our experimental results show that minimizing L_j performs better on the entity prediction task, and also makes our models capable of predicting relations and triples.

3.2 Deep Sequential Model

RNN can be considered to be a deep neural network with indefinite layers, since it can recurrently process input in arbitrary times. However, only using a single RNN cell to process all information may be inefficient for KGs. Multi-layer RNNs have shown promising performance on modeling complex hierarchical

architectures in the NLP area [26], and KGs happen to have such architectures. Therefore, we propose DSKG, which uses multi-layer RNNs to model complex KGs. Three different integrating strategies, namely **Joint**, **Respective** and **Cascade**, are designed to integrate the RNN cells in DSKG. Figure 1 illustrates a two-layer version of our DSKG model with the three strategies. We describe them in detail below:

3.3 Joint

This strategy uses two distinct RNN cells c_1, c_2 to process both entities and relations. The output hidden state of each cell needs to be fed to itself next time. Note that c_1's output hidden state is also conveyed to c_2. By doing this, we do not need c_1 to give an explicit result for each input, but enable it to convey its own comprehension to c_2. c_2 performs prediction based on its previous hidden state and c_1's processed hidden state, which include both sequential and hierarchical information. We formalize this process as follows:

$$\begin{aligned} \boldsymbol{s}_h^1 &= c_1(\boldsymbol{h}, \boldsymbol{s}_0) \\ \boldsymbol{s}_h^2 &= c_2\left(\boldsymbol{s}_h^1, \boldsymbol{s}_0\right), \end{aligned} \tag{6}$$

$$\begin{aligned} \boldsymbol{s}_r^1 &= \boldsymbol{c}_1(\boldsymbol{r}, \boldsymbol{s}_h^1) \\ \boldsymbol{s}_r^2 &= \boldsymbol{c}_2(\boldsymbol{s}_r^1, \boldsymbol{s}_h^2). \end{aligned} \tag{7}$$

We still use Equation (2) to calculate the output of RNN cells, and get two output pairs $\left(\boldsymbol{o}_h^1, \boldsymbol{o}_h^2\right)$ and $\left(\boldsymbol{o}_r^1, \boldsymbol{o}_r^2\right)$. For each pair, we can either combine its two output by weights or just use its last output to predict relations or tail entities. The former one has an advantage of modeling long hierarchical sequences, such as multiple brace-matching; while the latter one usually performs better when the sequences are short [26]. In this paper, we only use the output of the last cells for prediction, since triples in KGs are short.

3.4 Respective

Because entities and relations have very different characteristics, we believe that building multi-layer RNNs for them respectively may help our model capture very complex structures. Based on this intuition, we provide the relation layer with independent RNN cells. As shown in Figure 1, we still use c_1, c_2 to process input $\boldsymbol{h}$, but assign independent RNN cells c_3, c_4 to process $\boldsymbol{r}$:

$$\begin{aligned} \boldsymbol{s}_r^1 &= c_3(\boldsymbol{r}, \boldsymbol{s}_h^1) \\ \boldsymbol{s}_r^2 &= c_4(\boldsymbol{s}_r^1, \boldsymbol{s}_h^2). \end{aligned} \tag{8}$$

This is the only difference compared with Equation (7). Our experiments show that this strategy improves the performance on completing KGs with more complex structures. For example, FB15K [4] is considered to be a complex data set since it has more than 1,000 different relations, while WN18 [4] only has 18 types of relations.

3.5 Cascade

We propose this strategy because we believe that longer sequential RNN cells may model knowledge structures better, and the cells in the entity layer or the relation layer can concentrate more on its own current work. For comparison, c_1, c_2 in **Respective** pass their output hidden states to c_3, c_4, respectively. Differently, **Cascade** only feeds c_2's output hidden state to c_3. So, **Cascade** cuts down the conflicts between the entity layer and the relation layer. We believe that this can help improve performance on triple prediction. We formalize this strategy as follows:

$$\begin{aligned} \boldsymbol{s}_r^1 &= c_3(\boldsymbol{r}, \boldsymbol{s}_h^2) \\ \boldsymbol{s}_r^2 &= c_4(\boldsymbol{s}_r^1, \boldsymbol{s}_0). \end{aligned} \tag{9}$$

However, $\boldsymbol{h}$ needs to be sequentially conveyed four times to obtain the final output. Such long sequence may help our model process $\boldsymbol{h}$'s features, but may also lose some information of $\boldsymbol{h}$ during conveying.

In addition to the above three integrating strategies, we also design a simple variant, called **Entity-only**, for comparing with **Joint**. The only difference between them is that **Entity-only** only optimizes the entity-only loss, which is more like previous entity prediction models.

3.6 Batch Normalization and Dropout

To accelerate convergence and prevent over-fitting, we also employ two optimization methods in our models.

3.7 Batch Normalization

Batch normalization is widely used to alleviate the impact of improperly-initialized neural networks [27]. It enforces the input of next layers to a uniform Gaussian distribution. In our models, the batch normalization layers are placed before and after both the entity layer and the relation layer.

3.8 Dropout

Dropout is a simple but effective method to prevent over-fitting [28]. It is implemented by keeping a neuron active with probability p_D during training. In our models, we add the dropout layers before and after each RNN cell.

4. EXPERIMENTS

We implemented our method with TensorFlow, and conducted three different experiments, namely entity prediction, relation prediction and triple prediction, to evaluate it. In this section, we first introduce the data sets and experiment settings. Then, we describe the procedure of each experiment and report the corresponding results.

By carrying out these experiments, we want to answer the following three questions:

1) Can our method achieve state-of-the-art performance on some benchmark data sets?
2) How does our method perform on the new KG completion tasks such as relation prediction and triple prediction?
3) What are the strengths and weaknesses of each integrating strategy in our deep sequential model?

4.1 Data Sets and Experiment Settings

In all our experiments, we chose FB15K and WN18 as our data sets, which were proposed in [4] and used by a large number of previous methods. FB15K has 1,345 different relations, while WN18 contains 18 distinct relations. The detailed statistical data of these two data sets are listed in Table 2.

Table 2. Statistics of the experimental data sets.

	FB15K	WN18
# Entities	14,951	40,943
# Relations	1,345	18
# Training triples	483,142	141,442
# Validation triples	50,000	5,000
# Testing triples	59,071	5,000

For each data set, we used the Adam [29] optimizer to train one model for all the evaluation tasks and stopped training when the entity prediction result on the validation data is optimized. Thus, the relation prediction and triple prediction results may not be optimal. For both FB15K and WN18, we set the parameters as follows: learning rate $\lambda = 0.001$, embedding dimension $k = 512$, negative sample number $n_s = 512$, batch size $n_B = 2{,}048$, and the keep-probability for dropout layers $p_D = 0.5$. We employed the LSTM cells in all our models and used two such cells in DSKG (joint), four in DSKG (respective) and DSKG (cascade), just as shown in Figure 1. Adding more cells for DSKG may improve the performance, but would cause slower training speed. Also, our experiments would demonstrate the effectiveness of the two-layer DSKG model.

Additionally, we added the reversed relations in the training data. This strategy is helpful to model relation pairs reversed to each other. There are many methods using reversed relations, such as PTransE [11] and ConvE [15]. We directly used the results reported in their papers. Specifically, for each triple (h, r, t) in the training data, we constructed a reversed triple (t, r^-, h) and added it into the training data. Adding the reversed relations also enabled our models to predict head and tail entities in an integrated fashion, which means that our models can predict tail entities with input $(h, r, ?)$, and predict head entities with $(t, r^-, ?)$ simultaneously. The reversed relations provide a convenient way to evaluate the head prediction. Also, they enhance the connectivity of KGs. Adding them can slightly improve the performance, while significantly boost the convergence speed.

4.2 Entity Prediction

Entity prediction aims to predict h (or t) given an incomplete triple (?, r, t) (or (h, r, ?)). We evaluated our models with the same method used in [4]. For each triple (h, r, t) in the testing data, we constructed two incomplete triples (h, r, ?), (t, r^-, ?) for tail prediction and head prediction, respectively.

Following [4] and many others, two evaluation metrics were used: the mean rank of correct entities (MR) and the percentage of correct entities in ranked top-10 (Hits@10). Besides, an incomplete triple like (h, r, ?) may have multiple correct tails. Entity t in the current testing triple (h, r, t) is only one of the correct. Thus, we also employed the filtered mean rank (FMR) and filtered Hits@10 (FHits@10) [4], which removed all the correct entities except t during ranking.

The experimental results on FB15K and WN18 are shown in Table 3. We can observe that:

1) DSKG outperformed the other models on FB15K and also achieved superior performance on WN18 for Hits@10 and FHits@10.
2) Compared with BSKG, DSKG significantly improved the performance of FHits@10 on FB15K, which has a more complex structure than WN18.
3) DSKG (joint) outperformed DSKG (entity-only) on both FB15K and WN18, which proved that jointly optimizing relation prediction and entity prediction is helpful for predicting entities.
4) The three strategies in DSKG performed similarly on FB15K, but diverged for MR and FMR on WN18. For example, DSKG (cascade) achieved a better FHits10 than DSKG (joint) on WN18, but it performed worse on MR and FMR. We argue that WN18 only has 5,000 triples for testing, but it has about 40,000 different entities. So, if a model fails on one triple in the testing data, its MR and FMR would drop 8.0. However, for FB15K, its MR and FMR would only drop 0.25 at most.
5) DSKG (respective) outperformed DSKG (joint) on FB15K, which may show that using distinct RNN cells for the entity layer and the relation layer is helpful for modeling complex KGs.

Table 4 shows the detailed entity prediction results on FB15K. We separated results by relationship categories, e.g., 1:1 denotes the one-to-one relationship and 1:M denotes one-to-many. We can observe that BSKG and DSKG outperformed the others on the complex relationship categories (i.e., 1:M, M:1 and M:N). The reasons are: (1) Our models are probability-based rather than margin-based, so they would not suffer from the problem of modeling the complex relationships in the margin-based models; (2) RNN cells can properly model triples that have complex relationships, since they can transfer and model entities or relations alternately.

Table 3. Entity prediction results.

Models	FB15K				WN18			
	Hits@10	FHits@10	MR	FMR	Hits@10	FHits@10	MR	FMR
SE	28.8	39.8	273	162	68.5	80.5	1,011	985
NTN	-	41.4	-	-	-	66.1	-	-
TransE	34.9	47.1	243	125	75.4	89.2	263	251
TransH	45.7	64.4	211	84	75.4	86.7	318	303
TransD	49.4	77.3	194	67	79.6	92.5	224	212
TransR	43.8	65.5	198	77	79.8	92.0	232	219
CTransR	48.4	70.2	198	75	78.9	92.3	231	218
PTransE	51.8	84.6	200	54	-	-	-	-
DISTMULT	-	57.7	-	-	-	94.2	-	-
NLFeat	-	87.0	-	-	-	94.3	-	-
STransE	51.6	79.7	219	69	80.9	93.4	217	206
ConvE	-	83.1	-	51	-	93.5	-	374
BSKG	53.1	86.5	189	37	81.7	95.0	252	236
DSKG (entity-only)	53.7	89.5	192	40	82.5	95.1	248	233
DSKG (joint)	53.8	89.8	188	38	82.5	95.2	217	203
DSKG (respective)	53.9	89.9	188	36	81.4	95.2	354	338
DSKG (cascade)	54.1	89.3	183	36	81.5	95.3	370	353

Note: "-" indicates the result unreported in literature. Models are ordered according to their publishing years.

Table 4. Detailed entity prediction results on FB15K by relationship categories.

Models	Head prediction (FHits@10)					Tail prediction (FHits@10)				
	1:1	1:*M*	*M*:1	*M*:*N*	Overall	1:1	1:*M*	*M*:1	*M*:*N*	Overall
SE	35.6	62.6	17.2	37.5	36.7	34.9	14.6	68.3	41.3	42.8
TransE	43.7	65.7	18.2	47.2	44.6	43.7	19.7	66.7	50.0	49.7
TransH	66.8	87.6	28.7	64.5	61.4	65.5	39.8	83.3	67.2	67.1
TransD	86.1	95.5	39.8	78.5	74.5	85.4	50.6	94.4	81.2	80.5
TransR	78.8	89.2	34.1	69.2	66.0	79.2	37.4	90.4	72.1	71.8
CTransR	81.5	89.0	34.7	71.2	67.6	80.8	38.6	90.1	73.8	73.1
PTransE	91.0	92.8	60.9	83.8	81.4	91.2	74.0	88.9	86.4	85.7
STransE	82.8	94.2	50.4	80.1	77.1	82.4	56.9	93.4	83.1	82.3
BSKG	87.8	96.7	64.1	86.4	84.1	87.8	72.3	95.9	89.6	89.0
DSKG (entity-only)	89.2	97.0	67.7	89.6	87.1	89.7	80.5	96.0	92.7	92.0
DSKG (joint)	88.3	97.0	69.5	89.2	87.3	88.7	82.2	96.1	92.7	92.2
DSKG (respective)	89.7	97.1	69.6	89.9	87.5	88.8	82.2	96.2	92.9	92.3
DSKG (cascade)	89.4	97.2	68.3	89.2	86.9	89.7	80.9	96.1	92.3	91.8

4.3 Relation Prediction

Relation prediction aims to predict relations given a head (or tail) entity only. For each triple (h, r, t) in the same testing data as used in the entity prediction experiment, we took $\boldsymbol{h}$ as input for predicting r, and $\boldsymbol{t}$ for r^-. Note that the testing data used here can be redundant. For example, (h_1, r_1, t_1) and (h_1, r_1, t_2) are two different triples for entity prediction, while relation prediction only considers $\boldsymbol{h}_1$, $\boldsymbol{r}_1$. We did not remove the redundant testing triple (h_1, r_1, t_2) in the relation prediction and triple prediction experiments, since the occurrence of (h_1, r_1) can be regarded as its weight, reflecting its prevalence and importance in a KG.

Since there are no previous models designed for this task, we proposed a specific relation prediction model for comparison, which has two fully-connected layers. Each layer in this model has a weight matrix and a bias vector, and uses ReLU as the activating function. Note that DSKG (entity-only) may not suit this experiment task, because it does not minimize the relation prediction loss during training. We used the same evaluation metrics in this experiment.

The relation prediction results are shown in Table 5. For FB15K, we can observe the following:

1) Predicting forward relations was more accurate than predicting backward relations for FHits@10 and FMR, due to the facts that KGs are usually constructed by humans, and (h, r, t) is a more natural representation than (t, r^-, h).
2) DSKG outperformed the fully-connected two-layer model, which verified that jointly optimizing relation prediction and entity prediction can also help predict relations.
3) Both BSKG and DSKG achieved an FMR of 1.5 on predicting forward relations, which indicated that our models predicted the correct forward relations of an entity with a very high probability on average. We also evaluated our models on WN18. Because this data set has only 18 distinct relations, it is not surprising that all the models achieved similar performance. Still, DSKG showed a slight advantage for Hits@10 and FHits@10. It is worth noting that our models can also predict r given both h and t using the same method as in [11] (i.e., the aforementioned relationship prediction task).

Table 5. Relation prediction results.

Data sets	Models	Forward ($h \rightarrow$)				Backward ($t \rightarrow$)			
		Hits@10	FHits@10	MR	FMR	Hits@10	FHits@10	MR	FMR
FB15K	Fully-connected	79.8	98.7	6.4	1.8	88.7	90.6	5.0	4.6
	BSKG	80.2	99.1	6.0	1.5	89.4	91.3	4.6	4.1
	DSKG (joint)	82.6	99.1	5.7	1.5	90.3	92.2	4.4	4.0
	DSKG (respective)	83.5	99.0	5.8	1.7	90.5	92.2	4.6	4.2
	DSKG (cascade)	83.2	99.2	5.6	1.5	90.8	92.4	4.4	4.0
WN18	Fully-connected	99.7	99.7	2.0	1.1	99.5	99.6	2.1	1.5
	BSKG	99.6	99.7	2.0	1.2	99.6	99.8	1.9	1.4
	DSKG (joint)	99.8	99.9	2.0	1.2	99.8	99.9	1.9	1.4
	DSKG (respective)	99.8	99.9	2.7	1.5	99.8	99.9	2.8	1.8
	DSKG (cascade)	99.8	99.9	2.1	1.1	99.8	99.9	2.1	1.4

4.4 Triple Prediction

DSKG is capable of not only predicting entities and relations, but also predicting the whole triples given an entity. So, triple prediction can be considered to be a more integrated task for KG completion. For each triple (h, r, t) in the same testing data as used in the entity prediction experiment, we only used $\boldsymbol{h}$ as input to predict the triples about h. Each model was first asked to predict the relation r_p of h with the highest probability, and then used this relation to predict the most probable tail t_p. As a result, we got one output triple (h, r_p, t_p) for each input $\boldsymbol{h}$. Note that the models in this experiment only predicted triples in the forward direction, because: (i) we have shown that the backward relation prediction results on FMR were worse than the forward relation prediction results; and (ii) predicting forward triples of an entity is more natural to KG modeling.

We evaluated the output triples with the following method. Let S_R denote the set of all triples in a KG, S_O denote the set of output triples, S_p denote the union of testing and validation triples. S_R and S_p are referred to as the representation and prediction triple sets, respectively. We calculate the correct representation number as $n_R = |S_R \cap S_O|$, and the correct prediction number as $n_p = |S_p \cap S_O|$.

So, the representation precision and the prediction precision are calculated as follows:

$$\begin{aligned} \text{Representation precision} &= \frac{n_R}{|S_O|} \\ \text{Prediction precision} &= \frac{n_P}{(|S_O| - n_R + n_P)}. \end{aligned} \tag{10}$$

We slightly modified the evaluation procedures of TransE, TransR and ConvE for comparison, since they are unable to predict relations. We provided relations for them using four different providers, i.e., DSKG (joint), DSKG (respective), DSKG (cascade) and correct relations.

The experimental results are shown in Table 6. We can observe the following:

1) DSKG outperformed TransE and TransR, especially on WN18, even though providing the correct relations for them. This may be caused by the fact that both TransE and TransR are margin-based, and thus comparing distances for prediction may mislead them.
2) ConvE achieved similar performance compared with BSKG, but performed slightly weaker than DSKG.
3) Even we did not provide the correct relations, and DSKG still achieved good results, especially DSKG (cascade), which outperformed all the others on both FB15K and WN18.
4) For all the integrating strategies in DSKG, using their own providers to provide relations performed even better for Representation precision, which is probably because correct relations are actually extracted from testing data, while their own providers preferred to give more relations that have already been trained.

Table 6. Triple prediction results.

Relation providers	Models	FB15K		WN18	
		Representation precision	Prediction precision	Representation precision	Prediction precision
DSKG	TransE (joint)	80.9	30.6	45.9	5.0
	TransE (respective)	81.6	32.9	10.6	0.7
	TransE (cascade)	80.8	31.2	30.6	1.8
	TransR (joint)	85.4	23.4	71.2	15.1
	TransR (respective)	87.4	29.3	55.8	15.6
	TransR (cascade)	85.7	24.4	71.8	15.5
	ConvE (joint)	95.7	71.8	91.1	56.2
	ConvE (respective)	95.8	72.6	82.8	49.7
	ConvE (cascade)	95.8	69.3	92.7	65.0
	BSKG (joint)	97.2	69.1	95.7	69.9
	BSKG (respective)	97.0	70.0	81.0	35.6
	BSKG (cascade)	97.2	69.6	93.6	56.7
	DSKG (joint)	98.5	70.7	96.1	72.5
	DSKG (respective)	98.3	73.4	92.4	58.5
	DSKG (cascade)	98.5	73.4	99.1	89.8
Correct relations	TransE	75.3	49.3	38.7	20.5
	TransR	79.4	48.3	64.7	49.0
	ConvE	94.8	80.4	87.3	85.0
	BSKG	92.1	75.2	89.5	82.1
	DSKG (joint)	95.2	82.8	90.2	83.1
	DSKG (respective)	95.2	82.9	96.7	94.1
	DSKG (cascade)	95.0	82.2	96.8	94.3

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new KG completion method that extends multi-layer RNNs to model triples in KGs as sequences. Our experimental results on FB15K and WN18 showed that our method achieved superior performance not only on the benchmark entity prediction task, but also on two new KG completion tasks to predict relations and triples given one single entity. For future work, we plan to explore the following three directions:

- Integrating the attention mechanism [30] in our method. While the attention mechanism has shown its power in the NLP area, applying it to KG completion has not been well studied. In future, we would like to extend our method with this mechanism to improve its inference ability.
- Using a provided KG to complete another KG. Recently, several methods start to leverage extra textual data for improving KG completion. However, textual data like entity descriptions and text corpora are written in natural language. Due to the ambiguity and heterogeneity of textual data, they may bring mistakes in prediction. Therefore, we think that taking existing KGs as another kind of extra data may improve performance.
- Embedding KGs for entity alignment. We also want to consider the problem of learning KG embeddings for entity alignment. Particularly, we look forward to learning embeddings of different KGs in a unified space and employing RNNs to explore neighboring context for entity alignment.

AUTHOR CONTRIBUTIONS

L. Guo (lbguo.nju@gmail.com) designed the model. Q. Zhang (qhzhang.nju@gmail.com) conducted main experiments. W. Hu (whu@nju.edu.cn, corresponding author) assembled the manuscript. Z. Sun (zqsun.nju@gmail.com) and Y. Qu (yzqu@nju.edu.cn) revised the whole paper.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China under Grant No. 61872172.

REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, & Z.G. Ives. DBpedia: A nucleus for a web of open data. In: KAberer et al. (eds) The Semantic Web. Berlin: Springer, 2007, pp. 722–735. doi: 10.1007/978-3-540-76298-0_52.

[2] K.D. Bollacker, C. Evans, P. Paritosh, T. Sturge, & J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, 2008, pp. 1247–1250. doi: 10.1145/1376616.1376746.

[3] A. Bordes, J. Weston, R. Collobert, & Y. Bengio. Learning structured embeddings of knowledge bases. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence, AAAI, 2011, pp. 301–306. Available at: https://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3659/3898.

[4] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, & O. Yakhnenko. Translating embeddings for modeling multi-relational data. In Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS), NIPS Foundation, 2013, pp. 2787–2795. Available at: http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf.

[5] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, & G.E. Hinton. Grammar as a foreign language. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, MIT Press, 2015, pp. 2773–2781. Available at: http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language.pdf.

[6] R. Józefowicz, O. Vinyals, M. Schuster, N. Shazeer, & Y. Wu. Exploring the limits of language modeling. In: Proceedings of the 4th International Conference on Learning Representations, ICLR, 2016.

[7] Z. Wang, J. Zhang, J. Feng, & Z. Chen. Knowledge graph embedding by translating on hyperplanes. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI, 2014, pp. 1112–1119. Available at: https://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531/8546.

[8] Y. Lin, Z. Liu, M. Sun, Y. Liu, & X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence, AAAI, 2015, pp. 2181–2187. doi: 10.1016/j.procs.2017.05.045.

[9] D.Q. Nguyen, K. Sirts, L. Qu, & M. Johnson. STransE: A novel embedding model of entities and relationships in knowledge bases. In: Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics, ACM, 2016, pp. 460–466. doi: 10.18653/v1/N16-1054.

[10] G. Ji, K. Liu, S. He, & J. Zhao. Knowledge graph completion with adaptive sparse transfer matrix. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI, 2016, pp. 985–991. Available at: https://aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11982/11693.

[11] Y. Lin, Z. Liu, H.-B. Luan, M. Sun, S. Rao, & S. Liu. Modeling relation paths for representation learning of knowledge bases. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, ACL, 2015, pp. 705–714. doi: 10.18653/v1/D15-1082.

[12] R. Socher, D. Chen, C. Manning, D. Chen, & A. Ng. Reasoning with neural tensor networks for knowledge base completion. In: Proceedings of the 26th International Conference on Neural Information Processing Systems, MIT Press, 2013, pp. 926–934. doi: 10.1109/ICICIP.2013.6568119.

[13] B. Yang, S.W. Yih, X. He, J. Gao, & L. Deng. Embedding entities and relations for learning and inference in knowledge bases. In: Proceedings of the 3rd International Conference on Learning Representations, ICLR, 2015. Available at: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICLR2015_updated.pdf.

[14] M. Nickel, V. Tresp, & H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In: L. Getoor, & T. Scheffer (eds.) Proceedings of the 28th International Conference on Machine Learning (ICML-11). Bellevue, Washington: ACM, pp. 809--816.

[15] K. Toutanova, & D. Chen. Observed versus latent features for knowledge base and text inference. In: Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality, 2015, pp. 57–66. doi: 10.18653/v1/w15-4007.

[16] M.S. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov, & M. Welling. Modeling relational data with graph convolutional networks. In: A. Gangemi et al. (eds.) The Semantic Web. ISWC 2018. Cham, Switzerland: Springer, 2018, pp. 593–607. doi: 10.1007/978-3-319-93417-4_38.

[17] D.K. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R.G_omez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, & R.P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, MIT Press, 2015, pp. 2224–2232. Available at: http://dl.acm.org/citation.cfm?id=2969488.

[18] T. Dettmers, P. Minervini, P. Stenetorp, & S. Riedel. Convolutional 2d knowledge graph embeddings. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI, 2018, pp. 1811–1818. Available at: https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366.

[19] Y. LeCun, L. Bottou, Y. Bengio, & P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11)(1998), 2278–2324. doi: 10.1109/5.726791.

[20] D.Q. Nguyen, T.D. Nguyen, D.Q. Nguyen, & D.Q. Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In: Proceedings of the 17th Annual Conference of the North American Chapter of the Association for Computational Linguistics, ACL, 2018, pp. 327–333. doi: 10.18653/v1/N18-2053.

[21] R. Xie, Z. Liu, J. Jia, H. Luan, & M. Sun. Representation learning of knowledge graphs with entity descriptions. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI, 2016, pp. 2659–2665. Available at: tps://aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12216/12004.

[22] H. Xiao, M. Huang, L. Meng, & X. Zhu. SSP: Semantic space projection for knowledge graph embedding with text descriptions. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI, 2017, pp. 3104–3110.

[23] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, & Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 2014, pp. 1724–1734. doi: 10.3115/v1/D14-1179.

[24] S. Hochreiter, & J. Schmidhuber. Long short-term memory. Neural Computation, 1997, pp. 1735–1780. doi: 10.1162/neco.1997.9.8.1735.

[25] S. Jean, K. Cho, R. Memisevic, & Y. Bengio. On using very large target vocabulary for neural machine translation. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, ACL, 2015, pp. 1–10.

[26] M. Hermans, & B. Schrauwen. Training and analyzing deep recurrent neural networks. In: Proceedings of the 26th International Conference on Neural Information Processing Systems, MIT Press, 2013, pp. 190–198. Available at: http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf.

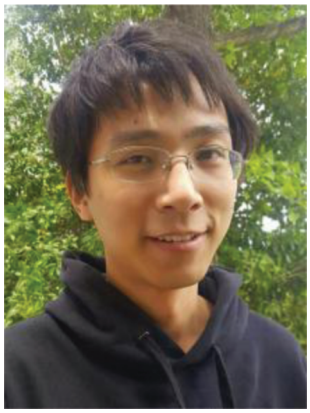
[27] S. Ioffe, & C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning, JMLR, 2015, pp. 448–456. Available at: https://dl.acm.org/citation.cfm?id=3045167%22.

[28] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, & R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 2014, pp. 1929–1958. Available at: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

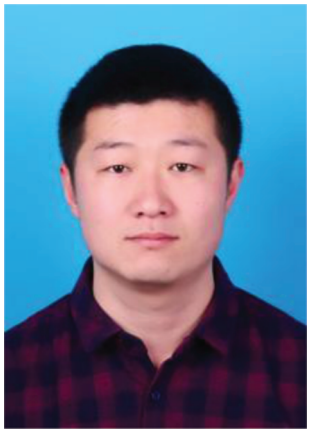
[29] D.P. Kingma, & J. Ba. Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations, ICLR, 2015.

[30] D. Bahdanau, K. Cho, & Y. Bengio. Neural machine translation by jointly learning to align and translate. In: Proceedings of the 3rd International Conference on Learning Representations, ICLR, 2015.

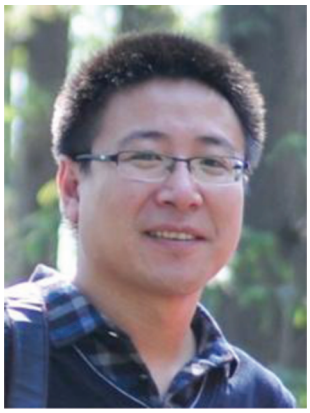
AUTHOR BIOGRAPHY



Lingbing Guo is currently a M.S. student in Department of Computer Science and Technology, Nanjing University, China. He received his B.S. degree in computer science and technology in 2016 from Henan University, China. His research interest is knowledge graph completion.

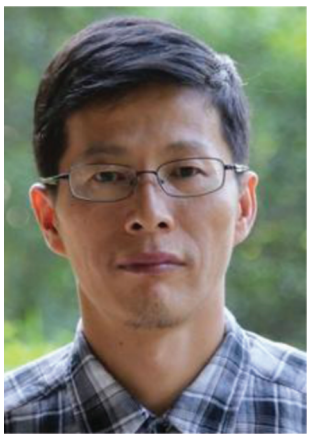


Qingheng Zhang is currently a M.S. student in Department of Computer Science and Technology, Nanjing University, China. He received his B.S. degree in computer science and technology in 2017 from Hohai University, China. His research interest is knowledge graph embedding.



Wei Hu is currently an associate professor in State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, China. He received his PhD degree in computer software and theory in 2009, and his B.S. degree in computer science and technology in 2005, both from Southeast University, China. His main research interests include knowledge graph, data integration and intelligent software.

Zequn Sun is currently a PhD student in Department of Computer Science and Technology, Nanjing University, China. He received his B.S. degree in computer science and technology in 2016 from Hohai University, China. His research interest is entity alignment.



Yuzhong Qu received his B.S. and M.S. degrees in mathematics from Fudan University, China, in 1985 and 1988, respectively, and got his Ph.D. degree in computer software from Nanjing University, China, in 1995. He is currently a full professor in State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, China. His research interests include Semantic Web, question answering and novel software technology for the Web.