Knowledge Transfer for Out-of-Knowledge-Base Entities: A Graph Neural Network Approach

Takuo Hamaguchi,[1] **Hidekazu Oiwa**,[2] **Masashi Shimbo**,[1] and **Yuji Matsumoto**[1]
[1]Nara Institute of Science and Technology, Ikoma, Nara, Japan
[2]Recruit Institute of Technology
{takuo-h,shimbo,matsu}@is.naist.jp, oiwa@recruit.ai

Abstract

Knowledge base completion (KBC) aims to predict missing information in a knowledge base. In this paper, we address the out-of-knowledge-base (OOKB) entity problem in KBC: how to answer queries concerning test entities not observed at training time. Existing embedding-based KBC models assume that all test entities are available at training time, making it unclear how to obtain embeddings for new entities without costly retraining. To solve the OOKB entity problem without retraining, we use graph neural networks (Graph-NNs) to compute the embeddings of OOKB entities, exploiting the limited auxiliary knowledge provided at test time. The experimental results show the effectiveness of our proposed model in the OOKB setting. Additionally, in the standard KBC setting in which OOKB entities are not involved, our model achieves state-of-the-art performance on the WordNet dataset.

1 Introduction

Knowledge bases such as WordNet [Miller, 1995] and Freebase [Bollacker *et al.*, 2008] are used for many applications including information extraction, question answering, and text understanding. These knowledge bases can be viewed as a set of *relation triplets*, i.e., triplets of the form (h, r, t) with an entity h called the *head entity*, a relation r, and an entity t called the *tail entity* [Bordes *et al.*, 2013; Nguyen *et al.*, 2016]. Some examples of relation triplets are (*Philip-K.-Dick*, *write*, *Do-Androids-Dream-of-Electric-Sheep?*) and (*Do-Androids-Dream-of-Electric-Sheep?*, *is-a*, *Science-fiction*). Although a knowledge base contains millions of such triplets, it is known to suffer from incompleteness [Nickel *et al.*, 2016]. Knowledge base completion (KBC) thus aims to predict the information missing in knowledge bases.

In recent years, *embedding-based* KBC models have been successfully applied to large-scale knowledge bases[Wang *et al.*, 2014b; Lin *et al.*, 2015; Ji *et al.*, 2015; Xiao *et al.*, 2016b; Nguyen *et al.*, 2016; Guu *et al.*, 2015]. These models build the distributed representations (or, *vector embeddings*) of entities and relations observed in the training data, and use various vector operations over the embeddings to predict missing relation triplets.

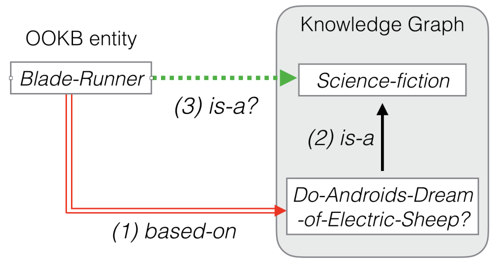

Figure 1: OOKB entity problem. At test time, we receive a new triplet (1) depicted as a red double arrow, which contains an entity "*Blade Runner*" that is not observed in the knowledge graph (shown as the shaded box). The task is to tell whether any other relations hold between *Blade Runner* and the entities in the knowledge graph, by using (1) the new triplet and (2) the existing triplet, depicted by a black arrow. For example, we want to answer the question, "Is Blade Runner science fiction?", i.e., whether (3) the green dashed arrow in the figure should be drawn.

In this paper, we address the *out-of-knowledge-base* (OOKB) entity problem in embedding-based KBC. This problem arises when new entities (OOKB entities) occur in the relation triplets that are given to the system *after* training. As these entities were unknown to the system at training time, the system does not have their embeddings, and hence does not have a means to predict the relations for these entities. The OOKB entity problem thus asks how to perform KBC involving such OOKB entities. Although it can be solved by retraining the embeddings using the added relation triplets containing the OOKB entities, a solution avoiding costly retraining is desirable.

This problem is of practical importance because OOKB entities crop up whenever new entities, such as events and products, are produced, which happens everyday. For example, suppose we find an OOKB entity "*Blade-Runner*" in a new triplet (*Blade-Runner*, *based-on*, *Do-Androids-Dream-of-Electric-Sheep?*). We want to infer more facts (viz. triplets) about *Blade-Runner* from the knowledge we already have, and answer questions such as "Is Blade Runner science fiction?" If the knowledge base contains a triplet (*Do-Androids-Dream-of-Electric-Sheep?*, *is-a*, *Science-fiction*), it should help us estimate that the answer is yes. Figure 1 illustrates this example schematically.

There have been some attempts to obtain the embeddings for OOKB entities using external resources [Wang *et al.*, 2014a; Fang *et al.*, 2016; Zhong *et al.*, 2015]. Although these approaches may be useful, they require additional computation over large resources, which may not be always feasible. By contrast, we pursue a KBC model that exploits existing triplets in the knowledge base, without relying on external resources. Indeed, the *Blade Runner* example above suggests the possibility of inferring new facts about OOKB entities without the help of external resources.

To solve the OOKB entity problem, we apply *graph neural networks* (Graph-NNs) [Scarselli *et al.*, 2009; Li *et al.*, 2015] to a *knowledge graph*, which is a graph obtained by regarding entities as nodes and triplets as edges. A Graph-NN is a neural network architecture defined on a graph structure and composed of two models called the *propagation model* and *output model*. The propagation model manages how information propagates between nodes in the graph. In the propagation model, we first obtain the embedding vectors of the neighborhood of a given node (entity) e and then convert these vectors into a representation vector of e using a pooling function such as the average. In other words, each node is embedded as a vector in continuous space, which is also used to calculate the vectors of the neighborhood nodes. This mechanism enables the vector for an OOKB entity to be composed from its neighborhood vectors at test time. The output model defines the task-oriented objective function over node vectors. This allows us to use an existing embedding-based KBC model as the output model. In this paper, we use TransE [Bordes *et al.*, 2013] as the output model, but we can adopt other embedding-based KBC methods.

Our main contributions can be summarized as follows:

- We propose a new formulation of the problem of OOKB entities in KBC.
- We propose a Graph-NN suitable for the KBC task with OOKB entities.
- We verify the effectiveness of our models in both the standard and the "OOKB" entity settings.

2 OOKB Entity Problem in Knowledge Base Completion

2.1 Knowledge Graph

Let $\mathcal{E}$ be a set of *entities* and $\mathcal{R}$ be a set of *relations*. Define the *fact*, or *relation triplet*, to be a triplet of form (h, r, t) where $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$. Let $\mathcal{G}_{\text{gold}} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ be the set of *gold* facts, i.e., the set of all relation triplets that hold for pairs of entities in $\mathcal{E}$ and relations in $\mathcal{R}$. If a triplet is in $\mathcal{G}_{\text{gold}}$, we say it is a *positive triplet*; otherwise, it is a *negative triplet*. The goal of knowledge completion is to identify $\mathcal{G}_{\text{gold}}$, when only its proper subset, or an "incomplete" *knowledge base*, $\mathcal{G} \subset \mathcal{G}_{\text{gold}}$ is accessible.

A knowledge base $\mathcal{G}$ is often called *knowledge graph* because each triplet in $\mathcal{G}$ can be regarded as a (labeled) edge in a graph; i.e., the entities in a triplet correspond to the end nodes and the relation gives the label of the edge.

2.2 KBC: Triplet Classification

Triplet classification is a typical KBC task introduced by [Socher *et al.*, 2013] [1] and has since been a standard benchmark for KBC methods [Wang *et al.*, 2014b; Ji *et al.*, 2015; Xiao *et al.*, 2016a; Yoon *et al.*, 2016; Nguyen *et al.*, 2016].

In this task, existing knowledge base $\mathcal{G}$ is assumed to be incomplete, in the sense that some of triplets that must be present in $\mathcal{G}$ are missing; i.e., $\mathcal{G} \neq \mathcal{G}_{\text{gold}}$.

Let $\mathcal{H} = (\mathcal{E} \times \mathcal{R} \times \mathcal{E}) \backslash \mathcal{G}$ be the set of triplets not present in $\mathcal{G}$. Because $\mathcal{G}$ is incomplete, two cases are possible for each triplet $x \in \mathcal{H}$; either x is a positive triplet (i.e., $x \in \mathcal{G}_{\text{gold}}$), or x is a negative triple (i.e., $x \notin \mathcal{G}_{\text{gold}}$). For the former case, x is not in $\mathcal{G}$ only because of the incompleteness, and the knowledge base must be updated to contain x. We thus encounter the problem of determining which of the above two possible cases each triplet not present in $\mathcal{G}$ falls into. This problem is called *triplet classification*.

Viewed as a machine learning problem, triplet classification is a classifier induction task in which $\mathcal{E}$ and $\mathcal{R}$ are given, and knowledge base $\mathcal{G}$ forms the training set (with only positive examples), with $\mathcal{H}$ being the test set. The set $\mathcal{H}$ can be divided into the set of positive test examples $\mathcal{H} \cap \mathcal{G}_{\text{gold}} = \mathcal{G}_{\text{gold}} \backslash \mathcal{G}$ and the set of negative test examples $\mathcal{H} \backslash \mathcal{G}_{\text{gold}}$.

In the standard triplet classification, $\mathcal{E}$ and $\mathcal{R}$ are limited to the entities and relations that appear in $\mathcal{G}$. That is, $\mathcal{E} = \mathcal{E}(\mathcal{G})$ and $\mathcal{R} = \mathcal{R}(\mathcal{G})$, where $\mathcal{E}(\mathcal{G}) = \{h \mid (h, r, t) \in \mathcal{G}\} \cup \{t \mid (h, r, t) \in \mathcal{G}\}$ and $\mathcal{R}(\mathcal{G}) = \{r \mid (h, r, t) \in \mathcal{G}\}$ denote the entities and relations appearing in $\mathcal{G}$, respectively.

2.3 OOKB Entity Problem

We now introduce a new task in KBC, called the OOKB entity problem.

In addition to the knowledge base $\mathcal{G}$ observed at training time, new triplets $\mathcal{G}_{\text{aux}}$ are given at test time, with $\mathcal{E}(\mathcal{G}_{\text{aux}}) \not\subseteq \mathcal{E}(\mathcal{G})$ and $\mathcal{R}(\mathcal{G}_{\text{aux}}) \subseteq \mathcal{R}(\mathcal{G})$. Thus, $\mathcal{G}_{\text{aux}}$ contains new entities $\mathcal{E}_{\text{OOKB}} = \mathcal{E}(\mathcal{G}_{\text{aux}}) \backslash \mathcal{E}(\mathcal{G})$, but no new relations are involved. We call $\mathcal{E}_{\text{OOKB}}$ *OOKB entities*. It is assumed that every triplet in $\mathcal{G}_{\text{aux}}$ contains exactly one OOKB entity from $\mathcal{E}_{\text{OOKB}}$ and one entity from $\mathcal{E}(\mathcal{G})$; that is, the additional triplets $\mathcal{G}_{\text{aux}}$ represent edges bridging $\mathcal{E}(\mathcal{G})$ and $\mathcal{E}_{\text{OOKB}}$ in the combined knowledge graph $\mathcal{G} \cup \mathcal{G}_{\text{aux}}$. In this setting, $\mathcal{E} = \mathcal{E}(\mathcal{G}) \cup \mathcal{E}_{\text{OOKB}} \neq \mathcal{E}(\mathcal{G})$, and the task is to correctly identify missing relation triplets that involve the OOKB entities $\mathcal{E}_{\text{OOKB}}$. Because the embeddings for these entities are missing, they must be computed from those for entities in $\mathcal{G}$. In other words, we want to design a model by which the information we already have in $\mathcal{G}$ can be transferred to OOKB entities $\mathcal{E}_{\text{OOKB}}$, with the help of the added knowledge $\mathcal{G}_{\text{aux}}$.

3 Proposed Model

3.1 Graph-NNs

Graph-NNs are neural networks defined on a graph structure. Although there exist graph-NNs that encode an entire graph into a vector [Cao *et al.*, 2016; Defferrard *et al.*, 2016], here

[1] A similar task had been around for general graphs under the name of *link prediction*.

we focus on the one that provides the means to encode nodes and edges into vectors, as this is more suitable for KBC.

According to [Scarselli *et al.*, 2009; Li *et al.*, 2015], a graph-NN consists of two models, the propagation model and the output model. The propagation model determines how to propagate information between nodes in a graph. The output model defines an objective function according to given tasks using vector-represented nodes and edges. In this paper, we modify the propagation model to be suitable for knowledge graphs. For the output model, we use the embedding-based KBC model TransE [Bordes *et al.*, 2013].

3.2 Propagation Model on a Knowledge Graph

Let $\mathcal{G}$ be a knowledge graph, $e \in \mathcal{E}(\mathcal{G})$ be an entity, and $\mathbf{v}_e \in \mathbb{R}^d$ be the d-dimensional representation vector of e. Li et al. define the propagation model by the following equation [Li *et al.*, 2015]:

$$\mathbf{v}_e = \sum_{(h,r,e)\in\mathcal{N}_{\text{head}}(e)} T_{\text{head}}(\mathbf{v}_h; h, r, e) + \sum_{(e,r,t)\in\mathcal{N}_{\text{tail}}(e)} T_{\text{tail}}(\mathbf{v}_t; e, r, t), \quad (1)$$

where head neighborhood $\mathcal{N}_{\text{head}}$ and tail neighborhood $\mathcal{N}_{\text{tail}}$ are $\mathcal{N}_{\text{head}}(e) = \{(h,r,e) \mid (h,r,e) \in \mathcal{G}\}$ and $\mathcal{N}_{\text{tail}}(e) = \{(e,r,t) \mid (e,r,t) \in \mathcal{G}\}$ in a knowledge graph $\mathcal{G}$, respectively. In addition, $T_{\text{head}}, T_{\text{tail}} : \mathbb{R}^d \times \mathcal{E}(\mathcal{G}) \times \mathcal{R}(\mathcal{G}) \times \mathcal{E}(\mathcal{G}) \to \mathbb{R}^d$ are called *transition functions* [Li *et al.*, 2015] and used to transform the vector of a neighbor node for its incorporation in the current vector $\mathbf{v}_e$, depending on the property of the edge between them.

We generalize the propagation function using the following *pooling function* P:

$$S_{\text{head}}(e) = \{T_{\text{head}}(\mathbf{v}_h; h, r, e) \mid (h,r,e) \in \mathcal{N}_h(e)\}, \quad (2)$$
$$S_{\text{tail}}(e) = \{T_{\text{tail}}(\mathbf{v}_t; e, r, t) \mid (e,r,t) \in \mathcal{N}_t(e)\}, \quad (3)$$
$$\mathbf{v}_e = P(S_{\text{head}}(e) \cup S_{\text{tail}}(e)), \quad (4)$$

Here, $S_{\text{head}}(e)$ contains the representation vectors of neighborhood $\mathcal{N}_{\text{head}}(e)$, and $S_{\text{tail}}(e)$ contains those for $\mathcal{N}_{\text{tail}}(e)$. The difference between Eq. (1) and ours (Eqs. (2)–(4)) is the use of the pooling function in place of the summation. The candidates for functions T_{head}, T_{tail}, and P are described below.

Transition Function The aim of transition function T (including both T_{head} and T_{tail}) is to modify the vector of a neighbor node to reflect the relations between the current node and the neighbor. The examples of the transition function are listed here:

$$T(\mathbf{v}) = \mathbf{v}, \qquad \text{(identity)}$$
$$T(\mathbf{v}) = \tanh(\boldsymbol{A}\mathbf{v}), \qquad \text{(single } \tanh \text{ layer)}$$
$$T(\mathbf{v}) = \mathrm{ReLU}(\boldsymbol{A}\mathbf{v}), \qquad \text{(single ReLU layer)}$$

where $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ is a matrix of model parameters and $\tanh$ and ReLU are elementwise hyperbolic tangent and rectified linear unit functions. In addition, we can use other neural network techniques, such as batch-normalization [Ioffe and Szegedy, 2015], residual connection [He *et al.*, 2016], and long short term memory [Li *et al.*, 2015].

We can also make the transition function dependent on the relation between the current node (entity) and the neighbor, such as in the following:

$$T_{\text{head}}(\mathbf{v}_h; h, r, e) = \tanh(\boldsymbol{A}^{\text{head}}_{(h,r,e)}\mathbf{v}_h),$$
$$T_{\text{tail}}(\mathbf{v}_t; e, r, t) = \tanh(\boldsymbol{A}^{\text{tail}}_{(e,r,t)}\mathbf{v}_t).$$

Note that the parameter matrices are now defined individually for each combination of node e, the current neighbors (h or t), and the relation r between them.

In the experiments of Section 4, we use the following transition functions:

$$T_{\text{head}}(\mathbf{v}_h; h, r, e) = \mathrm{ReLU}(\mathrm{BN}(\boldsymbol{A}^{\text{head}}_r\mathbf{v}_h)), \quad (5)$$
$$T_{\text{tail}}(\mathbf{v}_t; e, r, t) = \mathrm{ReLU}(\mathrm{BN}(\boldsymbol{A}^{\text{tail}}_r\mathbf{v}_t)), \quad (6)$$

where BN indicates batch normalization [Ioffe and Szegedy, 2015].

Pooling Function Pooling function P is a function that maps a set of vectors to a vector, i.e., $P : 2^{\mathbb{R}^d} \to \mathbb{R}^d$. Its objective is to extract shared aspects from a set of vectors. For $S = \{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^N$, some simple pooling functions are as follows:

$$P(S) = \sum_{i=1}^N \mathbf{x}_i, \qquad \text{(sum pooling)}$$
$$P(S) = \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i, \qquad \text{(average pooling)}$$
$$P(S) = \max(\{\mathbf{x}_i\}_{i=1}^N), \qquad \text{(max pooling)}$$

where $\max$ is the elementwise max function. Sum pooling was used in [Scarselli *et al.*, 2009; Li *et al.*, 2015]; see also Eq. (1).

Stacking and Unrolling Graph Neural Networks As explained above, the propagation models decide how to propagate information from a node to its neighborhood. Applying this propagation model repeatedly, we can broadcast information of a node to farther nodes, i.e., each node can receive further information. Broadcasting can be implemented in one of two ways: stacking or unrolling.

The unrolled Graph-NN is discussed in [Scarselli *et al.*, 2009; Li *et al.*, 2015]. In the unrolled Graph-NN, the propagation model uses the same model parameters in every propagation. The propagation procedures are the same as described in Eqs. (2)–(4).

The stacked Graph-NN is constructed in a similar manner to the well-known stacking technique [Vincent *et al.*, 2010]. In particular, the propagation process in the stacked Graph-NN uses different model parameters depending on time step n. The transition function at each time step n, indicated by superscript n, is as follows.

$$\mathbf{v}_e^{(n)} = \begin{cases} \mathbf{v}_e, & \text{if } n = 0, \\ P(S_{\text{head}}^{(n-1)}(e) \cup S_{\text{tail}}^{(n-1)}(e)), & \text{otherwise,} \end{cases}$$

where

$$S^{(n)}_{\text{head}}(e) = \{T^{(n)}_{\text{head}}(\mathbf{v}^{(n)}_h; h, r, e) \mid (h, r, e) \in \mathcal{N}_{\text{head}}(e)\}$$
$$S^{(n)}_{\text{tail}}(e) = \{T^{(n)}_{\text{tail}}(\mathbf{v}^{(n)}_t; e, r, t) \mid (e, r, t) \in \mathcal{N}_{\text{tail}}(e)\}.$$

where $T^{(n)}_{\text{head}}$ and $T^{(n)}_{\text{tail}}$ are transition functions depending on head/tail and time.

3.3 Output Model: Score and Objective Functions

We use a TransE-based objective function as the output model. TransE [Bordes *et al.*, 2013] is one of the basic embedding-based models for KBC, and we use it for its simplicity and ease of training. Notice however that our architecture is not limited to TransE, and adopting other embedding-based models for the output model is equally straightforward.

Below, we explain the score function of TransE and its commonly used *pairwise-margin* objective functions. We then describe the modified objective function we used in our experiments, called the *absolute-margin* objective.

Score Function The (implausibility) score function f evaluates the implausibility of a triplet (h, r, t); smaller scores indicate that the triplet is more likely to hold. In TransE, the score function is defined by $f(h, r, t) = \|\mathbf{v}_h + \mathbf{v}_r - \mathbf{v}_t\|$, where $\mathbf{v}_h$, $\mathbf{v}_r$, and $\mathbf{v}_t$ are the embedding vectors of the head, relation, and tail, respectively. This score function states that for a positive triplet (h, r, t), the sum of the head and relation vectors $\mathbf{v}_h + \mathbf{v}_r$ must be close to the tail vector $\mathbf{v}_t$, i.e., $\mathbf{v}_h + \mathbf{v}_r \sim \mathbf{v}_t$. This score function is modified and extended in [Wang *et al.*, 2014b; Lin *et al.*, 2015; Ji *et al.*, 2015; Xiao *et al.*, 2016b]. As mentioned earlier, all these models can be used as our output model.

Pairwise-Margin Objective Function The objective (loss) function defines the quantity to be minimized through optimization. The following *pairwise-margin* objective function is commonly used with KBC methods including TransE [Bordes *et al.*, 2013; Xiao *et al.*, 2016b]:

$$\mathcal{L} = \sum_{i=1}^{N} [\tau + f(h_i, r_i, t_i) - f(h'_i, r_i, t'_i)]_+ \tag{7}$$

where $[x]_+$ is the hinge function $[x]_+ = \max(0, x)$ and scalar $\tau \in \mathbb{R}$ is a threshold (called *margin*), with (h_i, r_i, t_i) denoting a positive triplet and (h'_i, r_i, t'_i) denoting a negative triplet. This objective function requires score $f(h'_i, r_i, t'_i)$ to be greater than score $f(h_i, r_i, t_i)$ by at least τ. If the difference is smaller than τ, then the optimization changes the parameters to meet the requirement. In contrast, if the difference is greater than τ, the parameters are not updated.

The pairwise-margin objective thus pays attention to the difference in scores between positive-negative triplet pairs.

Absolute-Margin Objective Function Instead of the pairwise-margin objective, in this paper, we employ the following objective function, which we call the *absolute-margin* objective.

$$\mathcal{L} = \sum_{i=1}^{N} f(h_i, r_i, t_i) + [\tau - f(h'_i, r_i, t'_i)]_+ \tag{8}$$

Table 1: Specifications of the triplet classification datasets. half of the validation and test sets are negative triplets, and these are included in the numbers of validation triplets and test triplets.

	WordNet11	Freebase13
Relations	11	13
Entities	38,696	75,043
Training triplets	112,581	316,232
Validation triplets	5,218	11,816
Test triplets	21,088	47,466

Table 2: Result of the standard KBC experiment (without OOKB entities). The figures represent accuracy. Except for the proposed method, they are obtained from the respective papers. Bold and underlined figures are the best and second best scores for each dataset, respectively.

Method	WordNet11	Freebase13
NTN [Socher *et al.*, 2013]	70.4	87.1
TransE [Bordes *et al.*, 2013]	75.9	81.5
TransH [Wang *et al.*, 2014b]	78.8	83.3
TransR [Lin *et al.*, 2015]	85.9	82.5
TransD [Ji *et al.*, 2015]	86.4	**89.1**
TransE-COMP [Guu *et al.*, 2015]	80.3	87.6
TranSparse [Ji *et al.*, 2016]	86.8	88.2
ManifoldE [Xiao *et al.*, 2016a]	<u>87.5</u>	87.3
TransG [Xiao *et al.*, 2016b]	87.4	87.3
lppTransD [Yoon *et al.*, 2016]	86.2	<u>88.6</u>
NMM [Nguyen *et al.*, 2016]	86.8	<u>88.6</u>
Proposed method	**87.8**	81.6

where τ is a hyperparameter, again called the *margin*. This objective function considers positive and negative triplets separately in the first and the second terms, not jointly as in the pairwise-margin objective. The scores for the positive triplets will be optimized towards zero, whereas the scores of the negative triplets are going to be at least τ. This objective function not only is easy to optimize, but also obtained good results in our preliminary experiments. We thus used this objective function for the experiments in Section 4.

4 Experiments

4.1 Implementation and Hyperparameters

We implemented our models using the neural network library Chainer (http://chainer.org/). The code and dataset are available at https://github.com/takuo-h/GNN-for-OOKB. All networks were trained by stochastic gradient descent with backpropagation; specifically, we used the Adam optimization method [Kingma and Ba, 2014]. The step size of Adam was $\alpha_1/(\alpha_2 \cdot k + 1.0)$, where k indicates the number of epochs performed, $\alpha_1 = 0.01$, and $\alpha_2 = 0.0001$. The mini-batch size was $5,000$ and the number of training epochs was 300 in every experiment. Moreover, the dimension of the embedding space was 200 in the standard triplet classification and 100 in other settings.

In the preliminary experiments, we tried several activation functions and pooling functions, and found the following hyperparameter settings on account of both computational time and performance. We used Eqs. (5)–(6) as transision functions in both the standard and OOKB settings. As the pooling

Table 3: Number of entities and triplets in the the OOKB datasets. The numbers of triplets include negative triplets.

	Head			Tail			Both		
	1,000	3,000	5,000	1,000	3,000	5,000	1,000	3,000	5,000
Training triplets	108,197	99,963	92,309	96,968	78,763	67,774	93,364	71,097	57,601
Validation triplets	4,613	4,184	3,845	3,999	3,122	2,601	3,799	2,759	2,166
OOKB entities	348	1,034	1,744	942	2,627	4,011	1,238	3,319	4,963
Test triplets	994	2,969	4,919	986	2,880	4,603	960	2,708	4,196
Auxiliary entities	2,474	6,791	10,784	8,191	16,193	20,345	9,899	19,218	23,792
Auxiliary triplets	4,352	12,376	19,625	15,277	31,770	40,584	18,638	38,285	48,425

Table 4: Results of the OOKB experiment: accuracy of the simple baseline and proposed models. Bold and underlined figures are respectively the best and second best scores for each dataset.

		Head			Tail			Both		
Method	Pooling	1,000	3,000	5,000	1,000	3,000	5,000	1,000	3,000	5,000
Baseline	sum	54.6	52.5	52.0	53.7	53.0	52.8	54.0	52.7	53.2
	max	58.1	56.3	56.4	55.2	54.2	55.3	56.8	56.8	56.4
	avg	63.0	60.2	61.1	63.8	<u>63.9</u>	<u>63.0</u>	65.3	<u>63.9</u>	<u>64.8</u>
Proposed	sum	70.2	62.6	59.6	64.6	56.5	55.0	59.5	55.2	54.2
	max	<u>80.3</u>	<u>75.4</u>	<u>72.7</u>	<u>74.8</u>	63.1	58.7	<u>68.0</u>	59.5	56.5
	avg	**87.3**	**84.3**	**83.3**	**84.0**	**75.2**	**69.2**	**83.0**	**73.3**	**68.2**

function, we used the max pooling function in the standard triplet classification, and tried three pooling functions, max, sum, average, in the OOKB setting. The results of the preliminary experiments were reflected in our selection of the absolute-margin objective function over the pairwise-margin objective function as well as the margin value $\tau = 300$ in the absolute-margin objective function (Eq. (8)). The absolute-margin objective function converged faster than the pairwise-margin objective function. Because the task is a binary classification of triplets into positive (i.e., the relations that must be present in the knowledge base) and negative triplets (those that must not), we determined the threshold value for output scores between these classes using the validation data.

To deal with the limited available computational resources (e.g., GPU memory), we sampled the neighbor entities randomly when an entity has too many of them. Indeed, there were some entities that appeared in a large number of the triplets, and thus had many neighbors; when the neighborhood size exceeded 64, we randomly chose 64 entities from the neighbors.

4.2 Standard Triplet Classification

We compared our model with the previous KBC models in the standard setting, in which no OOKB entities are involved.

Datasets We used WordNet11 and Freebase13 [Socher *et al.*, 2013] for evaluation. The data files were downloaded from http://cs.stanford.edu/people/danqi/. These datasets are subsets of two popular knowledge graphs, WordNet [Miller, 1995] and Freebase [Bollacker *et al.*, 2008]. The specifications on these datasets are shown in Table 1. Both datasets contain training, validation, and test sets. The validation and test sets include positive and negative triplets. In contrast, the training set does not contain negative triplets. As usual with the case in which negative triplets are not available, *corrupted* triplets are generated from positive triplets are used as a substitute for negative triplets. From a positive triplet (h, r, t) in knowledge base $\mathcal{G}$, a corrupted triplet is generated by substituting a random entity sampled from $\mathcal{E}(\mathcal{G})$ for h or t. Specifically, to generate corrupted triplets, we used the "Bernoulli" trick, a technique also used in [Wang *et al.*, 2014b; Lin *et al.*, 2015; Ji *et al.*, 2015; Ji *et al.*, 2016].

Result The results are shown in Table 2. Our model showed state-of-the-art performance on the WordNet11 dataset. On the Freebase13 dataset, it did not perform as well as the state-of-the-art KBC methods, although it was slightly better than TransE on which our model was built on.

4.3 OOKB Entity Experiment

Datasets We processed the WordNet11 dataset to construct several datasets for our OOKB entity experiment.

In total, nine datasets were constructed with different numbers and positions of OOKB entities sampled from the test set. The process consists of two steps: choosing OOKB entities and filtering and splitting triplets.

1. *Choosing OOKB entities.* To choose the OOKB entities, we first selected $N = 1,000$, $3,000$, and $5,000$ triplets from the WordNet11 test file.

   For each of these three sets, we chose the initial candidates for the OOKB entities (denoted by $\mathcal{I}$) in three different ways (thereby yielding nine datasets in total); these settings are called Head, Tail, and Both. In the Head setting, all head entities in the N triplets are regarded as candidate OOKB entities. The Tail setting is similar, but with the tail entities regarded as candidates. In the Both setting, all entities appearing as either a head or tail are the candidates.

The final OOKB entities are the entities $e \in \mathcal{I}$ that appear in a triplet (e, r, e') or (e', r, e) in the WordNet11 training set, with $e' \notin \mathcal{I}$. Note that the entities $h, t \notin \mathcal{I}$ are contained in the knowledge bases we already have and not in the OOKB entities. This last process filters out candidate OOKB entities that do not have any connection with the training entities.

2. *Filtering and splitting triplets.* Using the selected OOKB entities, the original training dataset was split into the training dataset and the auxiliary datasets for the OOKB entity problem. That is, triplets that did not contain the OOKB entities were placed in the OOKB training set, and triplets containing one OOKB entity and one non-OOKB entity were placed in the auxiliary set. Triplets that contained two OOKB entities were discarded.

   For the test triplets, we used the same first N triplets in the WordNet11 test file that we used in Step 1, with the exception that the triplets that did not contain any OOKB entities were removed. For the validation triplets, we simply removed the triplets containing OOKB entities from the WordNet11 validation set.

The details of the generated OOKB datasets are shown in Table 3. We denote each of the nine datasets by {Head, Tail, Both}-{1,000, 3,000, 5,000}, respectively, where the first part represents the position of OOKB entities and the second part represents the number of triplets used for generating the OOKB entities.

Result Using the nine datasets generated from WordNet11, we verified the effectiveness of our proposed model.

We used the following simple method as the baseline in this experiment. Given an OOKB entity u, we first obtained the embedding vectors of the neighborhood (determined by the triplets in the auxiliary knowledge) using TransE, and then converted these vectors into the representation vector of u using a pooling function: sum, max, or average. Note that because all the neighborhood entities of u are in the training knowledge base, their vectors can be computed using standard KBC methods. We followed the original paper [Bordes *et al.*, 2013] of TransE for the hyperparameter and other settings.

The results are shown in Table 4. The column labeled "pooling" indicates which pooling function was used. As the table shows, our model outperforms the baselines considerably. In particular, Graph-NN with average pooling outperforms the other methods on all datasets. Graph-NN with max pooling also shows good accuracy, but in several settings, such as Tail-3000, it was outperformed by the baseline model with average pooling.

4.4 Stacking and Unrolling Graph-NNs

Stacking and unrolling are important techniques because they enable us to broadcast node information to distant nodes. In this experiment, we illustrate the effect of stacking and unrolling in the standard triplet classification task. The dataset is WordNet11, used for standard triplet classification.

Table 5: Accuracy of stacked and unrolled Graph-NNs. The depth indicates the number of times the propagation model is iteratively applied.

Depth	Stacking	Unrolling
1	87.8	87.8
2	87.5	87.2
3	87.1	86.7
4	87.0	87.0

Accuracy Comparison Table 5 shows the performance of the stacked and unrolled Graph-NNs. The parameter "depth" indicates how many times the propagation model is iteratively applied. Note that when depth $= 1$, the two models reduce to the vanilla Graph-NN, for which the result was 87.8%, as also shown in Table 2. These results imply that the stacking and unrolling techniques do not improve performance. We believe this is because of the power of the embedding models, i.e., we can embed information about distant nodes into a continuous space.

5 Conclusion

In this paper, we proposed a new KBC task in which entities unobserved at training time are involved. For this task, we proposed a Graph-NN tailored to KBC with OOKB entities. We conducted two triplet classification tasks to verify the effectiveness of our proposed model. In the OOKB entity problem, our model outperformed the baselines considerably. Our model also showed state-of-the-art performance on WordNet11 in the standard KBC setting.

Acknowledgments

We thank the anonymous reviewers for valuable comments. This work was partially supported by JSPS Kakenhi Grant 15H02749.

References

[Bollacker *et al.*, 2008] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250, 2008.

[Bordes *et al.*, 2013] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems 26*, pages 2787–2795, 2013.

[Cao *et al.*, 2016] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016.

[Defferrard *et al.*, 2016] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29*, pages 3844–3852, 2016.

[Fang *et al.*, 2016] Wei Fang, Jianwen Zhang, Dilin Wang, Zheng Chen, and Ming Li. Entity disambiguation by knowledge and text jointly embedding. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 260–269, 2016.

[Guu *et al.*, 2015] Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 318–327, 2015.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[Ji *et al.*, 2015] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, volume 1: Long Papers, pages 687–696, 2015.

[Ji *et al.*, 2016] Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. Knowledge graph completion with adaptive sparse transfer matrix. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016.

[Kingma and Ba, 2014] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[Li *et al.*, 2015] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. *CoRR*, abs/1511.05493, 2015.

[Lin *et al.*, 2015] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

[Miller, 1995] George A. Miller. Wordnet: A lexical database for English. *Communication of the ACM*, 38(11):39–41, 1995.

[Nguyen *et al.*, 2016] Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. Neighborhood mixture model for knowledge base completion. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 40–50, 2016.

[Nickel *et al.*, 2016] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104:11–33, 2016.

[Scarselli *et al.*, 2009] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20:61–80, 2009.

[Socher *et al.*, 2013] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934, 2013.

[Vincent *et al.*, 2010] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[Wang *et al.*, 2014a] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph and text jointly embedding. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1591–1601, 2014.

[Wang *et al.*, 2014b] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 2014.

[Xiao *et al.*, 2016a] Han Xiao, Minlie Huang, and Xiaoyan Zhu. From one point to a manifold: Knowledge graph embedding for precise link prediction. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 1315–1321, 2016.

[Xiao *et al.*, 2016b] Han Xiao, Minlie Huang, and Xiaoyan Zhu. TransG: A generative model for knowledge graph embedding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1: Long Papers, pages 2316–2325, 2016.

[Yoon *et al.*, 2016] Hee-Geun Yoon, Hyun-Je Song, Seong-Bae Park, and Se-Young Park. A translation-based knowledge graph embedding preserving logical property of relations. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 907–916, 2016.

[Zhong *et al.*, 2015] Huaping Zhong, Jianwen Zhang, Zhen Wang, Hai Wan, and Zheng Chen. Aligning knowledge and text embeddings by entity descriptions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 267–272, 2015.