Out-of-Sample Representation Learning for Knowledge Graphs

Marjan Albooyeh, Rishab Goel, and Seyed Mehran Kazemi
Borealis AI, Montreal, Canada
{marjan.albooyeh, rishab.goal, mehran.kazemi}@borealisai.com

Abstract

Many important problems can be formulated as reasoning in knowledge graphs. Representation learning has proved extremely effective for *transductive reasoning*, in which one needs to make new predictions for already observed entities. This is true for both attributed graphs (where each entity has an initial feature vector) and non-attributed graphs (where the only initial information derives from known relations with other entities). For *out-of-sample reasoning*, where one needs to make predictions for entities that were unseen at training time, much prior work considers attributed graph. However, this problem is surprisingly under-explored for non-attributed graphs. In this paper, we study the *out-of-sample representation learning* problem for non-attributed knowledge graphs, create benchmark datasets for this task, develop several models and baselines, and provide empirical analyses and comparisons of the proposed models and baselines.

1 Introduction

Multi-relational graphs are a prevalent form of graphs where each edge has a label and a direction associated with it. Many prediction problems can be formulated as reasoning within a multi-relational graph. For example, Figure 1 depicts a job recommendation system that has been formulated in these terms. A notable example of multi-relational graphs is knowledge graphs (KGs) with several applications in natural language processing and information retrieval including search, question answering and commonsense reasoning. Much prior work has considered transductive KG reasoning in which predictions are made at test time for only those entities that were observed during training. These are known as *in-sample* entities. In Figure 1, predicting if A_1 is expert in S_2 is an example of transductive reasoning.

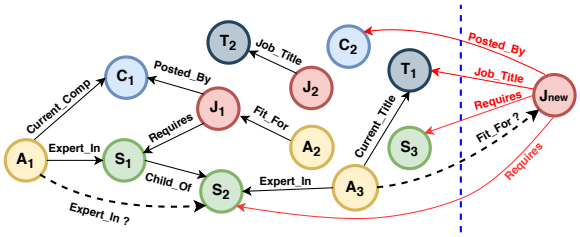


Figure 1: An example of a multi-relational graph for a job recommendation system is presented on the left side of the dashed blue line where the vertices A_i, C_i, S_i, J_i and T_i represent applicants, companies, skills, job postings, and titles respectively. Predicting whether A_1 is expert in S_2 is an example of transductive reasoning. J_{new} represents an out-of-sample entity that has not been observed during training. Predicting whether A_3 is a good fit for J_{new} based on the relations of J_{new} observed during test time (red arrows) is an example of out-of-sample reasoning.

Conversely, we consider *out-of-sample* KG reasoning. We make predictions for previously unseen or *out-of-sample* entities based on their relations with the in-sample entities. This is more challenging than transductive reasoning as it requires generalizing to unseen entities. In Figure 1, predicting whether A_3 is a good fit for the previously unseen job posting J_{new} given J_{new}'s relations with in-sample entities (observed at test time) is an example of out-of-sample reasoning.

Representation learning has proved effective for reasoning in KGs (Nickel et al., 2016; Hamilton et al., 2017b; Kazemi et al., 2020). It has been extensively studied for transductive reasoning in *attributed* graphs (where each entity has an initial feature vector) and *non-attributed* KGs (where the only initial information derives from known relations with other entities) as well as *simple graphs* (in which there is only a single relation). One prominent family of work is based on extensions of the convolution operator to non-Euclidean domains

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 2657–2666

(Kipf and Welling, 2017; Defferrard et al., 2016; Hammond et al., 2011; Schlichtkrull et al., 2018). A second family models relations as translations (or rotations) from subject to object entities (Bordes et al., 2013; Ji et al., 2015; Nguyen et al., 2016; Sun et al., 2019). A third approach represents the facts in a KG as a 3rd order tensor and factorizes this tensor to produce entity and relation embeddings (Yang et al., 2015; Trouillon et al., 2016; Kazemi and Poole, 2018; Zhang et al., 2019).

Out-of-sample representation learning has also been extensively studied for attributed KGs (Xie et al., 2016; Zhao et al., 2017) and attributed simple graphs (Yang et al., 2016; Hamilton et al., 2017a; Veličković et al., 2018; Chen et al., 2018). However, for non-attributed KGs, it remains under-explored. The main challenge of out-of-sample representation learning for non-attributed KGs is that an entity representation must be learned using only the relations the entity participates in. Ma et al. (2018) develop such a model for non-attributed simple graphs but extending their work to KGs is not straightforward. Out-of-sample representation learning in non-attributed graphs is an important problem for high-throughput production systems, as it is not tractable to adapt the transductive approaches and use additional rounds of gradient descent to incorporate new entities at test time.

The contributions of this work are as follows: 1) we formally define out-of-sample representation learning for KGs, 2) we create benchmark datasets for this problem, 3) we propose several baselines, 4) we extend current transductive KG representation learning approaches by developing new training algorithms that can support the incorporation of out-of-sample entities at test time via aggregation functions to compute representations, and 5) we provide a thorough experimental comparison of the baselines and the proposed approaches.

2 Background and Notation

Lower-case letters denote scalars, bold lower-case letters denote vectors, and bold upper-case letters denote matrices. For a vector $\boldsymbol{z} \in \mathbb{R}^d$, we represent by $\boldsymbol{z}[i]$ ($n \leq d$) the i^{th} element of $\boldsymbol{z}$ and by $||\boldsymbol{z}||$ the Euclidean norm of $\boldsymbol{z}$. For $\boldsymbol{z_1}, \boldsymbol{z_2} \in \mathbb{R}^d$, we let $\boldsymbol{z}_1 \odot \boldsymbol{z}_2 \in \mathbb{R}^d$ represent the element-wise (Hadamard) product of the two vectors. For $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k \in \mathbb{R}^d$, we let $\langle \boldsymbol{z}_1, \ldots, \boldsymbol{z}_k \rangle = \sum_{i=1}^{d} (\boldsymbol{z}_1[i] * \cdots * \boldsymbol{z}_k[i])$ represent the sum of the element-wise product of the elements of the k vectors.

Let $\mathcal{V}$ and $\mathcal{R}$ represent a set of entities and relations respectively. We represent a **triple** as $(\mathsf{v}, \mathsf{r}, \mathsf{u})$, where $\mathsf{v} \in \mathcal{V}$ is the *head* (or subject), $\mathsf{r} \in \mathcal{R}$ is the *relation*, and $\mathsf{u} \in \mathcal{V}$ is the *tail* (or object) of the triple. Let ζ represent the set of all triples on entities $\mathcal{V}$ and relations $\mathcal{R}$ that are facts (e.g., $(\mathsf{Montreal}, \mathsf{LocatedIn}, \mathsf{Canada})$). A (non-attributed) **knowledge graph (KG)** $\mathcal{G} \subset \zeta$ is a subset of ζ. Hereafter, whenever we refer to a KG, we assume a non-attributed KG.

Transductive KG Reasoning: In transductive KG reasoning, a model is learned for a KG $\mathcal{G}$ with entities $\mathcal{V}$ and relations $\mathcal{R}$ such that the model can make predictions about any triple $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ where $\mathsf{v}, \mathsf{u} \in \mathcal{V}$ are both in-sample entities and $\mathsf{r} \in \mathcal{R}$.

KG embedding models map entities and relations to hidden representations known as *embeddings* and define a function ϕ from the embeddings of the entities and the relation in a triple to a score corresponding to the degree of belief the model has for the relation holding between the entities. Typically, the embeddings can be formulated as two matrices $\boldsymbol{Z}_{ent} \in \mathbb{R}^{|\mathcal{V}| \times d_{ent}}$ and $\boldsymbol{Z}_{rel} \in \mathbb{R}^{|\mathcal{R}| \times d_{rel}}$ where each row of $\boldsymbol{Z}_{ent}$ corresponds to the embedding for an entity, each row of $\boldsymbol{Z}_{rel}$ corresponds to the embedding for a relation, and d_{ent} and d_{rel} represent entity and relation embedding sizes. One can look up the embedding for a particular entity v by multiplying the transpose of $\boldsymbol{Z}_{ent}$ to the one-hot encoding of v and for a particular relation r by multiplying the transpose of $\boldsymbol{Z}_{rel}$ to the one-hot encoding of r. A large number of approaches define $\boldsymbol{Z}_{ent}$ and $\boldsymbol{Z}_{rel}$ as matrices with directly learnable parameters. Other approaches define encoders that produce these two matrices typically through several rounds of message passing among entities.

Algorithm 1 outlines one epoch of training for learning the embeddings as well as the parameters of the ϕ function. The training is performed using stochastic gradient descent with mini-batches. For each batch (line 2), the `nextBatch` function extracts a set of positive triples from the KG and creates n negative triples per positive triple by corrupting the positive triple according to the procedure introduced in (Bordes et al., 2013). n is known as the *negative ratio*. For each triple $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ in the batch, the embeddings for v, r and u are looked up and the score for the triple is computed according to ϕ. Then the embeddings and the parameters of ϕ are updated based on the predicted scores, the labels of the triples, and a loss function $\mathcal{L}$.

Algorithm 1 Transductive Training (one epoch)

Inputs n : negative ratio, $\mathcal{L}$: loss function

for $batch = 1$ **to** $numBatches$ **do**
  $triples, labels \leftarrow$ `nextBatch` $(batch, n)$
  $scores \leftarrow []$
  for $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ **in** $triples$ **do**
    $\boldsymbol{z}_{\mathsf{v}} \leftarrow \texttt{lookup}(\mathsf{v}, \boldsymbol{Z}_{ent})$
    $\boldsymbol{z}_{\mathsf{r}} \leftarrow \texttt{lookup}(\mathsf{r}, \boldsymbol{Z}_{rel})$
    $\boldsymbol{z}_{\mathsf{u}} \leftarrow \texttt{lookup}(\mathsf{u}, \boldsymbol{Z}_{\mathsf{ent}})$
    $scores$.`append`$(\phi(\boldsymbol{z}_{\mathsf{v}}, \boldsymbol{z}_{\mathsf{r}}, \boldsymbol{z}_{\mathsf{u}}))$
  end for
  `updateParams`$(\mathcal{L}, scores, labels)$
end for

Different models have been proposed in the literature by mainly changing the score function. Note that some models may break the vector embeddings into multiple pieces and reshape each piece before using it in the score function. In this paper, we focus primarily on DistMult, a simple yet effective model for transductive KG embedding. However, many of the ideas we develop in this paper are general and can be applied to other models as well.

DistMult (Yang et al., 2015): In DistMult, $\boldsymbol{Z}_{ent} \in \mathbb{R}^{|\mathcal{V}| \times d}$ and $\boldsymbol{Z}_{rel} \in \mathbb{R}^{|\mathcal{R}| \times d}$. For a triple $(\mathsf{v}, \mathsf{r}, \mathsf{u})$, let $\boldsymbol{z}_{\mathsf{v}}, \boldsymbol{z}_{\mathsf{r}}, \boldsymbol{z}_{\mathsf{u}} \in \mathbb{R}^d$ represent the embeddings for v, r and u respectively where each embedding is obtained by looking up the $\boldsymbol{Z}_{ent}$ and $\boldsymbol{Z}_{rel}$ matrices. DistMult defines the score for the triple as $\phi(\boldsymbol{z}_{\mathsf{v}}, \boldsymbol{z}_{\mathsf{r}}, \boldsymbol{z}_{\mathsf{u}}) = \langle \boldsymbol{z}_{\mathsf{v}}, \boldsymbol{z}_{\mathsf{r}}, \boldsymbol{z}_{\mathsf{u}} \rangle$, i.e. the sum of the element-wise product of the head, relation, and tail embeddings.

Loss function: We use the L2 regularized negative log-likelihood which has proved effective in several works (Trouillon et al., 2016; Kazemi and Poole, 2018). The loss $\mathcal{L}(\Theta)$ for a single batch of labeled triples is defined as follows:

$$\sum_{((\mathsf{v},\mathsf{r},\mathsf{u}),l) \in batch} \texttt{softplus}(-l \cdot \phi(\mathsf{v}, \mathsf{r}, \mathsf{u})) + \lambda ||\Theta||_2^2 \tag{1}$$

where Θ represents the parameters of the model, $\texttt{softplus}(x) = \texttt{log}(1 + \texttt{exp}(x))$, $l \in \{-1, 1\}$ represents the label of the triple in the batch, and λ represents the L2 regularization hyperparameter.

3 Out-of-Sample KG Reasoning

We define out-of-sample reasoning for KGs as:

Definition 1. Out-of-sample reasoning for KGs is the problem of training a model on a KG $\mathcal{G}$ with entities $\mathcal{V}$ and relations $\mathcal{R}$ such that at the test time, the model can be used for making predictions about any out-of-sample entity $\mathsf{v} \notin \mathcal{V}$ given $\mathcal{G}_{\mathsf{v}} = \{(\mathsf{v}, \mathsf{r}, \mathsf{u}) : \mathsf{u} \in \mathcal{V}, \mathsf{r} \in \mathcal{R}\} \cup \{(\mathsf{u}, \mathsf{r}, \mathsf{v}) : \mathsf{u} \in \mathcal{V}, \mathsf{r} \in \mathcal{R}\}$ corresponding to the relations between v and in-sample entities.

According to the definition, $\mathcal{G}_{\mathsf{v}}$ is observed only at the test time and so during training, the model does not observe any triples involving v. To develop a representation learning model for out-of-sample reasoning in KGs, one needs to learn i) embeddings for the in-sample entities in $\mathcal{V}$ and the relations in $\mathcal{R}$, ii) a function ϕ from triples to scores, and iii) a function from $\mathcal{G}_{\mathsf{v}}$ and the in-sample entity and relation embeddings to an embedding for v that can be used to make further predictions about v.

One possible way of extending transductive models such as DistMult to the out-of-sample domain is by following the standard training procedure outlined in Algorithm 1 and then defining an aggregation function with no learnable parameters which, at inference time, provides an embedding for an out-of-sample entity v based on the embeddings of the entities and relations in $\mathcal{G}_{\mathsf{v}}$. A simple aggregation function, for instance, can be the average of the embeddings for entities $\{\mathsf{u} : \exists \mathsf{r}\ s.t.\ (\mathsf{v}, \mathsf{r}, \mathsf{u}) \in \mathcal{G}_{\mathsf{v}}\ or\ (\mathsf{u}, \mathsf{r}, \mathsf{v}) \in \mathcal{G}_{\mathsf{v}}\}$ (i.e. all entities that have a relation with v). Such a procedure, however, introduces an inconsistency between training and testing as the training is done irrespective of the aggregation function and with the objective of performing well on a transductive task whereas the model is tested on an out-of-sample task.

3.1 Proposed Training Procedure

To make the training procedure resemble what is expected of the model at the test time and make it aware of the aggregation function being used, we propose a new training algorithm that guides the learning procedure towards learning entity and relation embeddings that better match the aggregation function. A general training procedure for out-of-sample representation learning is proposed in Algorithm 2. For each triple $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ in the batch, first we lookup the embedding for r. Then with probability $\frac{\psi}{2}$, where $0 \leq \psi \leq 1$ is a hyperparameter, we consider v to be out-of-sample and u to be in-sample. In this case, for v we use an aggregate function that computes the embedding for v based on the triples involving v except for $(\mathsf{v}, \mathsf{r}, \mathsf{u})$, and for u we simply lookup its embedding. Also with probability $\frac{\psi}{2}$, we consider u to be out-of-sample

Algorithm 2 Out-of-Sample Training (one epoch)

Inputs n : negative ratio, $\mathcal{L}$: loss function, ψ : see Section 3.1

for $batch = 1$ **to** $numBatches$ **do**
  $triples, labels \leftarrow \texttt{nextBatch}(batch, n)$
  $scores \leftarrow []$
  for $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ **in** $triples$ **do**
    $\boldsymbol{z}_\mathsf{r} \leftarrow \texttt{lookup}(\mathsf{r}, \boldsymbol{Z}_\mathsf{r})$
    $rand \leftarrow \texttt{random}()$
    if $rand < \frac{\psi}{2}$ **then**
      $\boldsymbol{z}_\mathsf{v} \leftarrow \texttt{aggregate}(\mathsf{v}, \boldsymbol{Z}_{rels}, \boldsymbol{Z}_{ent})$
      $\boldsymbol{z}_\mathsf{u} \leftarrow \texttt{lookup}(\mathsf{u}, \boldsymbol{Z}_{ent})$
    else if $\frac{\psi}{2} < rand < \psi$ **then**
      $\boldsymbol{z}_\mathsf{v} \leftarrow \texttt{lookup}(\mathsf{v}, \boldsymbol{Z}_{ent})$
      $\boldsymbol{z}_\mathsf{u} \leftarrow \texttt{aggregate}(\mathsf{u}, \boldsymbol{Z}_{rel}, \boldsymbol{Z}_{ent})$
    else
      $\boldsymbol{z}_\mathsf{v} \leftarrow \texttt{lookup}(\mathsf{v}, \boldsymbol{Z}_{ent})$
      $\boldsymbol{z}_\mathsf{u} \leftarrow \texttt{lookup}(\mathsf{u}, \boldsymbol{Z}_{ent})$
    end if
    $scores.\texttt{append}(\phi(\boldsymbol{z}_\mathsf{v}, \boldsymbol{z}_\mathsf{r}, \boldsymbol{z}_\mathsf{u}))$
  end for
  $\texttt{updateParams}(\mathcal{L}, scores, labels)$
end for

and v to be in-sample and follow a similar procedure. Finally, with probability $1 - \psi$, we follow the standard training procedure by looking up the embedding for both entities. Having the embeddings for v, r and u, we use a score function (e.g., DistMult) to compute the score for this triple being true. Finally, we update the embeddings (and the parameters of the aggregate and ϕ functions if they have any) according to the scores, labels, and a loss function $\mathcal{L}$. Note that when $\psi = 0$, Algorithm 2 reduces to Algorithm 1. Note that Algorithm 2 is generic and can be used with any KG embedding model.

By using Algorithm 2, one can develop different models for out-of-sample representation learning by choosing different ϕ and `aggregate` functions. We propose two aggregate functions that extend DistMult to out-of-sample domains.

3.2 Proposed Models

oDistMult-ERAvg: Let v be an entity for which we need to compute an embedding using aggregation and $\mathcal{G}_\mathsf{v}$ be the triples involving v. According to the score function of DistMult, for each triple $(\mathsf{v}, \mathsf{r}, \mathsf{u}) \in \mathcal{G}_\mathsf{v}$ (and similarly for each triple $(\mathsf{u}, \mathsf{r}, \mathsf{v}) \in \mathcal{G}_\mathsf{v}$), we want $\langle \boldsymbol{z}_\mathsf{v}, \boldsymbol{z}_\mathsf{r}, \boldsymbol{z}_\mathsf{u} \rangle$ to be high where $\boldsymbol{z}_\mathsf{v}$, $\boldsymbol{z}_\mathsf{r}$ and $\boldsymbol{z}_\mathsf{u}$ represent the embedding of v, r and u respectively. The score can be written as $\langle \boldsymbol{z}_\mathsf{v}, \boldsymbol{z}_\mathsf{r}, \boldsymbol{z}_\mathsf{u} \rangle = \boldsymbol{z}_\mathsf{v} \cdot (\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u})$ where $\cdot$ represents dot product. Since $\boldsymbol{z}_\mathsf{v} \cdot (\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u}) = ||\boldsymbol{z}_\mathsf{v}||\ ||\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u}||\ \cos(\boldsymbol{z}_\mathsf{v}, \boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u})$, one possible choice to ensure a high value for $\langle \boldsymbol{z}_\mathsf{v}, \boldsymbol{z}_\mathsf{r}, \boldsymbol{z}_\mathsf{u} \rangle$ is by choosing $\boldsymbol{z}_\mathsf{v}$ to be the vector $\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u}$ so that the angle θ between the two vectors becomes 0 (and consequently, $\cos(\theta) = 1$). Since there may be multiple triples in $\mathcal{G}_\mathsf{v}$, we average these vectors and define $\boldsymbol{z}_\mathsf{v} = \texttt{aggregate}(\mathsf{v})$ as follows:

$$\boldsymbol{z}_\mathsf{v} = \frac{1}{|\mathcal{G}_\mathsf{v}|}\Big(\sum_{(\mathsf{v},\mathsf{r},\mathsf{u}) \in \mathcal{G}_\mathsf{v}} \boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u} + \sum_{(\mathsf{u},\mathsf{r},\mathsf{v}) \in \mathcal{G}_\mathsf{v}} \boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u}\Big) \quad (2)$$

where $|\mathcal{G}_\mathsf{v}|$ represents the number of triples in $\mathcal{G}_\mathsf{v}$.

oDistMult-LS: An alternative to the averaging strategy in Equation (2) is to find $\boldsymbol{z}_\mathsf{v}$ as the solution to a least squares problem to ensure the score for the triples in $\mathcal{G}_\mathsf{v}$ are maximized. One way to achieve this goal is by solving a (potentially underdetermined) system of linear equations where there exists one equation of the form $\frac{\boldsymbol{z}_\mathsf{v} \cdot (\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u})}{||\boldsymbol{z}_\mathsf{v}||\ ||\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u}||} = 1$ for each triple $(\mathsf{v}, \mathsf{r}, \mathsf{u}) \in \mathcal{G}_\mathsf{v}$ (and similarly for each triple $(\mathsf{u}, \mathsf{r}, \mathsf{v}) \in \mathcal{G}_\mathsf{v}$). The presence of $||\boldsymbol{z}_\mathsf{v}||$ in the denominator makes finding an analytical solution difficult. We note that $||\boldsymbol{z}_\mathsf{v}||$ only affects the magnitude of the scores and not their ranking, so instead we consider the following equation:

$$\frac{\boldsymbol{z}_\mathsf{v} \cdot (\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u})}{||\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u}||} = 1 \quad (3)$$

Considering a matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{G}_\mathsf{v}| \times d}$ (recall that d is the embedding dimension) such that $\boldsymbol{A}[i] = \boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u}$ where r and u are the relation and entity involved in the i-th triple in $\mathcal{G}_\mathsf{v}$ and a vector $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{G}_\mathsf{v}|}$ such that $\boldsymbol{b}[i] = ||\boldsymbol{z}_\mathsf{r} \odot \boldsymbol{z}_\mathsf{u}||$, we compute $\boldsymbol{z}_\mathsf{v} = \texttt{aggregate}(\mathsf{v})$ analytically as follows:

$$\boldsymbol{z}_\mathsf{v} = (\boldsymbol{A}^T\boldsymbol{A} + \lambda \boldsymbol{I})^{-1}\boldsymbol{A}^T\boldsymbol{b} \quad (4)$$

where $\boldsymbol{I} \in \mathbb{R}^{d \times d}$ is an identity matrix and λ is a hyperparameter corresponding to L2 regularization which ensures the system has a unique solution.

While we proposed the aggregation functions for DistMult, note that they can be easily extended to other models such as SimplE, ComplEx, and QuatE that have 2, 4 and 8 $\langle ., ., . \rangle$ terms respectively.

3.3 Time Complexity

We analyze the time complexity of the proposed algorithms for finding the embedding of an out-of-sample entity v. Let us assume that $|\mathcal{G}_\mathsf{v}| = N$

Dataset	In-sample entities ($\|\mathcal{V}\|$)	Out-of-sample entities	$\|\mathcal{R}\|$	Train triples	Validation queries	Test queries
oWN18RR	32270	validation: 2848, test: 2848	11	60608	12760	12440
oFB15k-237	11579	validation: 1395, test: 1396	234	193490	44601	54082

Table 1: Statistics on oWN18RR and oFB15k-237.

and the embedding dimension is d. Finding the embedding for v in oDistMult-ERAvg has a time complexity of $O(Nd)$ as it requires computing N Hadamard products and then averaging the resulting vectors both having a time complexity of $O(Nd)$.

For oDistMult-LS, to create the matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$ one needs to compute N Hadamard products and find the norm of N vectors respectively. The time complexity of this step is $O(Nd)$. The size of the matrix $\boldsymbol{A}$ is $N \times d$ so computing $\boldsymbol{A}^T\boldsymbol{A}$ has a time complexity of $O(Nd^2)$, the matrix inversion has a time complexity of $O(d^3)$ and the product of the resulting inverted matrix into $\boldsymbol{A}^T$ also has a time complexity of $O(Nd^2)$. Therefore, the overall time complexity is $O(Nd^2 + d^3)$. Unless the degree size of the KG is quite large, one can expect d to be larger than N and so the time complexity becomes $O(d^3)$.

4 Datasets

We created datasets for out-of-sample representation learning over KGs using WN18RR (Dettmers et al., 2018) and FB15k-237 (Toutanova and Chen, 2015), two standard datasets for KG completion. WN18RR is a subset of Wordnet (Miller, 1995) and FB15k-237 is a subset of Freebase (Bollacker et al., 2008). We call the two datasets oWN18RR and oFB15k-237 respectively, where "o" in the beginning of the name stands for "out-of-sample". The statistics for these datasets can be found in Table 1.

We outline the steps we took for creating the datasets.

1. We merge the train, validation, and test triples from the original dataset into a single set.

2. From the entities appearing in at least 2 triples, we randomly select 20% to be candidates for the out-of-sample entities; other entities are in-sample entities. We avoid having entities appearing in only 1 triple as out-of-sample entities because, during test time, we select one triple as query and need other triples for learning a representation for the out-of-sample entity.

3. Triples containing two out-of-sample entities are removed, triples with one out-of-sample entity are considered as test triples and other triples are considered as train triples.

4. In step 3, it is possible that some entities selected to be in-sample appear in no training triples. This can happen whenever an in-sample entity only appears in triples involving an out-of-sample entity. A similar situation can occur for some relations as well (i.e. some relations only appearing in the test set). We remove such entities and relations and the triples they appear in from the dataset.

5. After doing the above steps, if the number of triples for an out-of-sample entity is less than 2, we remove that entity from the test set.

6. We randomly select half of the out-of-sample entities and the triples they appear in as the validation set and the other half as the test set.

5 Experiments and results

To measure the performance of different models, for any out-of-sample entity v in the test set with triples $\mathcal{G}_\mathsf{v}$, we create $|\mathcal{G}_\mathsf{v}|$ queries where in the i-th query, we use our learned model to compute an embedding for v given all except the i-th triple in $\mathcal{G}_\mathsf{v}$ and use that embedding to make a prediction about the i-th triple. Figure 2 represents statistics on the number of triples used to compute the embedding of the out-of-sample entities in the test set for both oWN18RR and oFB15k-237. If the i-th triple is of the form $(\mathsf{v}, \mathsf{r}, \mathsf{u})$, then we create the query $(\mathsf{v}, \mathsf{r}, ?)$ and find the ranking our model assigns to u (the correct answer to the query) among entities $\mathsf{u}' \in \mathcal{V}$ such that $(\mathsf{v}, \mathsf{r}, \mathsf{u}') \notin \mathcal{G}_\mathsf{v}$ (the $(\mathsf{v}, \mathsf{r}, \mathsf{u}') \notin \mathcal{G}_\mathsf{v}$ constraint is known as the *filtered* setting). We follow a similar procedure for the case where the i-th triple is of the form $(\mathsf{u}, \mathsf{r}, \mathsf{v})$. Let $\kappa_{(\mathsf{v},\mathsf{r},?),\mathsf{u}}$ represent

		oWN18RR				oFB15k-237			
		MRR	Hit@			MRR	Hit@		
Model	Training	Filtered	1	3	10	Filtered	1	3	10
Popularity	Algorithm 1	0.0094	0.0030	0.0076	0.0215	0.0320	0.0168	0.0322	0.0581
OOV	Algorithm 1	0.0004	0.0000	0.0001	0.0002	0.0002	0.0000	0.0000	0.0001
RGCN-D	Algorithm 1	0.0178	0.0072	0.0166	0.0352	0.1683	0.0974	0.1848	0.3056
DistMult-EAvg	Algorithm 1	0.0446	0.0248	0.0469	0.0841	0.0813	0.0525	0.0973	0.1327
DistMult-ERAvg	Algorithm 1	0.3048	0.2468	0.3331	0.4159	0.2456	0.1615	0.2769	0.4082
DistMult-LS	Algorithm 1	0.3514	0.2840	0.3911	0.4756	0.2073	0.1395	0.2264	0.3375
DistMult-LS-U	Algorithm 1	0.3238	0.2458	0.3693	0.4717	0.1674	0.1099	0.1858	0.2732
oDistMult-EAvg	Algorithm 2	0.2239	0.1315	0.2724	0.3897	0.1765	0.0724	0.2076	0.4012
oDistMult-ERAvg	Algorithm 2	0.3904	0.3460	0.4125	0.4725	**0.2557**	**0.1698**	**0.2885**	**0.4201**
oDistMult-LS	Algorithm 2	**0.4093**	**0.3643**	**0.4371**	**0.4892**	0.2126	0.1232	0.2404	0.3954

Table 2: Results on oWN18RR and oFB15k-237. Best results are in bold.

the rank of u for query $(\mathsf{v}, \mathsf{r}, ?)$. We report filtered *mean reciprocal rank (MRR)* computed as:

$$\frac{1}{\sum_{\mathsf{v}\in Test}|\mathcal{G}_\mathsf{v}|}\sum_{\mathsf{v}\in Test}\Big(\sum_{(\mathsf{v},\mathsf{r},\mathsf{u})\in\mathcal{G}_\mathsf{v}}\frac{1}{\kappa_{(\mathsf{v},\mathsf{r},?),\mathsf{u}}}+\sum_{(\mathsf{u},\mathsf{r},\mathsf{v})\in\mathcal{G}_\mathsf{v}}\frac{1}{\kappa_{(?,\mathsf{r},\mathsf{v}),\mathsf{u}}}\Big) \quad (5)$$

and filtered Hit@k (for $k \in \{1, 3, 10\}$) defined as:

$$\frac{1}{\sum_{\mathsf{v}\in Test}|\mathcal{G}_\mathsf{v}|}\sum_{\mathsf{v}\in Test}\Big(\sum_{(\mathsf{v},\mathsf{r},\mathsf{u})\in\mathcal{G}_\mathsf{v}}\mathbb{1}_{\kappa_{(\mathsf{v},\mathsf{r},?),\mathsf{u}}\le k}+\sum_{(\mathsf{u},\mathsf{r},\mathsf{v})\in\mathcal{G}_\mathsf{v}}\mathbb{1}_{\kappa_{(?,\mathsf{r},\mathsf{v}),\mathsf{u}}\le k}\Big) \quad (6)$$

where $\mathbb{1}_{condition}$ is 1 if the condition holds and 0 otherwise.

5.1 Baselines

We develop several baselines for out-of-sample representation learning over KGs.

Popularity: In this baseline, we rank the in-sample entities based on the number of times they appear in the triples of the training set. We break ties randomly. At the test time, we use this ranking as our answer to all queries.

OOV: This baseline is inspired by the way a word embedding is computed for out-of-vocabulary (OOV) words (i.e. words unseen during training) in some works in the natural language processing literature. After training, we compute the average embedding of all in-sample entities and use it as the embedding for out-of-sample entities.

RGCN-D: Graph convolutional networks (GCNs) have proved effective for inductive and out-of-sample learning when initial entity features are available. When such features are not available, Hamilton et al. (2017a) propose to use node degrees as initial entity features. Since we work with multi-relational graphs, we initialize entity features as vectors of size $2|\mathcal{R}|$ where the i-th and $|\mathcal{R}|+i$-th elements (for $i < |\mathcal{R}|$) represent the number of incoming and outgoing edges with relation type r_i respectively. We use RGCN (Schlichtkrull et al., 2018) as the GCN.

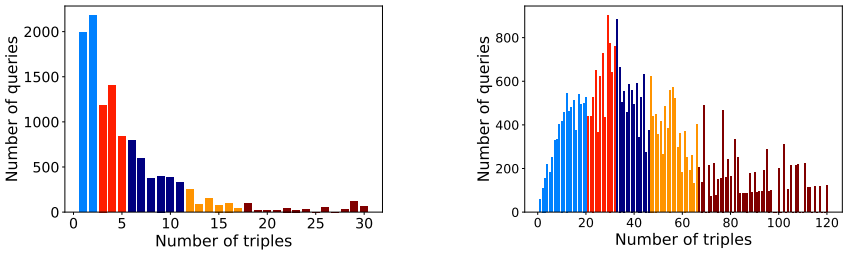
oDistMult-EAvg: Similar to the first baseline in (Ma et al., 2018), we create a simpler version of oDistMult-ERAvg by defining the embedding for an unseen entity v as the average of the embeddings of the entities that are related to v. More formally, this baseline defines $\boldsymbol{z}_\mathsf{v} = \texttt{aggregate}(\mathsf{v}) = \frac{1}{|\mathcal{G}_\mathsf{v}|}(\sum_{(\mathsf{v},\mathsf{r},\mathsf{u})\in\mathcal{G}_\mathsf{v}}\boldsymbol{z}_\mathsf{u}+\sum_{(\mathsf{u},\mathsf{r},\mathsf{v})\in\mathcal{G}_\mathsf{v}}\boldsymbol{z}_\mathsf{u})$.

DistMult-EAvg, DistMult-ERAvg, DistMult-LS: Corresponding to variants of oDistMult-EAvg, oDistMult-ERAvg and oDistMult-LS where instead of using Algorithm 2 for training, the standard training in Algorithm 1 is used.

DistMult-LS-U: As an ablation study, we also include an unnormalized version of DistMult-LS where we change Equation (3) to $\boldsymbol{z}_\mathsf{v}\cdot(\boldsymbol{z}_\mathsf{r}\odot\boldsymbol{z}_\mathsf{u})=1$ (in other words, setting the elements of $\boldsymbol{b}$ in Equation (4) to 1).

5.2 Implementation Details

For RGCN-D, we used the implementation in the deep graph library (DGL). We implemented other models and baselines in PyTorch (Paszke et al., 2017) and used the AdaGrad optimizer (Duchi et al., 2011). We selected the hyperparameters



(a) (b)

Figure 2: The two figures provide statistics on the test sets of (a) oWN18RR and (b) oFB15k-237. They show the number of test queries (on the y-axis) for which the embedding of the out-of-sample entity is computed based on k triples (e.g., for almost 2000 queries in oWN18RR, the embedding of the out-of-sample entity is learned based on only 1 triple). Since the number of samples for many of the larger values of k is 0, to make the plots visually appealing, we restricted the x-axis to $k \leq 30$ for oWN18RR and $k \leq 120$ for oFB15k-237 and did not include in the diagrams the few cases where k was larger. The colors show the bins used for the experiment in Figure 3(b, c).

corresponding to learning rate and L2 regularization (λ) via a grid search over $\{0.1, 0.01\}$ and $\{0.1, 0.01, 0.001, 0.0001\}$ respectively validating the models every 100 epochs and selecting the best hyperparameters and epoch based on validation filtered MRR. We set the negative ratio to 1 and the embedding dimension to 200. When using Algorithm 2 for training, we set ψ to 0.5 unless stated otherwise. The code and datasets are available at https://github.com/BorealisAI/OOS-KGE.

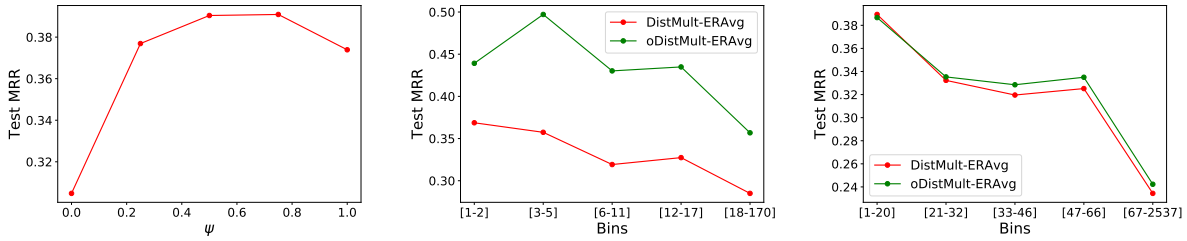
5.3 Results

According to the results on oWN18RR and oFB15k-237 reported in Table 2, in almost all cases, using Algorithm 2 for training as opposed to Algorithm 1 results in a boost of performance. Recall that the models whose names start with an "o" use Algorithm 2 and the models without "o" correspond to the variants where Algorithm 1 is used instead. On oWN18RR, for instance, oDistMult-ERAvg and oDistMult-LS achieve 28% and 16% improvement in terms of filtered MRR compared to DistMult-ERAvg and DistMult-LS respectively. The margins of improvements on oFB15k-237 are smaller as oFB15k-237 is generally a more challenging dataset compared to oWN18RR and it is more difficult to make progress on. We believe the reason for the observed boost when using Algorithm 2 is mainly because the train and test procedures become more consistent compared to when Algorithm 1 is used.

Furthermore, it can be observed that the proposed oDistMult-ERAvg and oDistMult-LS models outperform the other baselines. We believe the reason for the poor performance of RGCN-D on oWN18RR is because the out-of-sample entities have few neighbors (see Figure 2(a)) and the degree information (used as initial features) is not discriminative enough[1]. Between the two proposed models, the winner is dataset-dependant with oDistMult-LS performing slightly better on oWN18RR and oDistMult-ERAvg showing better performance on oFB15k-237. DistMult-LS also outperforms DistMult-LS-U shedding light on the importance of the normalization in Equation (3).

Selecting ψ: For the results in Table 2, we set the value of ψ to 0.5 (see Algorithm 2 for the usage of ψ). Here, we explore different values for ψ to see how it affects the performance. Figure 3(a) shows the test MRR of oDistMult-ERAvg on oWN18RR for different values of ψ. When $\psi = 0$ (corresponding to using the standard transductive training algorithm presented in Algorithm 1), the performance is poor. As soon as ψ becomes greater than zero, we observe a substantial boost in performance. The performance keeps increasing as ψ increases until reaching a plateau and then it goes down when $\psi = 1$ corresponding to a training procedure where for each triple, one entity is always treated as out-of-sample. We repeated the experiment with other models and on other datasets and observed similar behavior. We believe one reason why we observe a better performance for $0 < \psi < 1$ compared to $\psi = 1$ is that when $0 < \psi < 1$, the model is encouraged to learn embeddings that do well for both transductive and out-of-sample prediction

[1] We tried a variant of RGCN without self-loops (similar to the model in (Hamaguchi et al., 2017)) but obtained similar results as RGCN-D.



(a) (b) (c)

Figure 3: (a) The test MRR of oDistMult-ERAvg on oWN18RR for different values of ψ (introduced in Algorithm 2). (b) and (c) Test MRR of DistMult-ERAvg and oDistMult-ERAvg on oWN18RR and oFB15k-237 for different bins (the bins are presented in Figure 2).

tasks with the transductive task acting as an auxiliary task (and possibly as a regularizer) helping the embeddings capture more information.

Neighbor-size effect: Out-of-sample entities appear in a different number of triples. Figure 2 shows statistics for oWN18RR and oFB15k-237 on the number of triples used to learn the embedding for the out-of-sample entity in each query in the test set. To test how this number affects the models, we divided our test queries into 5 bins of (approximately) equal size as shown by the bar colors in Figure 2 and measured the test MRR on each bin. According to the results for oDistMult-ERAvg and DistMult-ERAvg, presented in Figure 3(b,c), oDistMult-ERAvg almost consistently outperforms DistMult-ERAvg on all (except one) bins. For both models, as the number of triples from which we learn the embedding for out-of-sample entities increases, the performance deteriorates, highlighting a shortcoming of our averaging strategy used for aggregation. Future work can look into other aggregation functions (e.g., attention-based averaging).

In-sample performance: To measure how training with Algorithm 2 affects model performance for in-sample (aka transductive) link prediction, we compared DistMult and oDistMult-ERAvg on the original splits of WN18AM, the cleaned version of WN18RR (Hajimoradlou and Kazemi, 2020). For this experiment, we used Adam optimizer (Kingma and Ba, 2014) and added a dropout of 0.5 after the Hadamard product of the embeddings (before taking the sum of the features) in DistMult. We tuned both learning rate and weight decay from the set $\{0.0001, 0.001, 0.01, 0.1\}$. The results in Table 3 indicate that training with our proposed algorithm does not deteriorate the performance for in-sample link prediction.

Model	MRR	Hit@1	Hit@3	Hit@10
DistMult	0.4498	0.4179	0.4614	0.5099
oDistMult-ERAvg	0.4483	0.4072	0.4711	0.5210

Table 3: In-sample link prediction results on a cleaned version of WN18RR named WN18AM (for details, see (Hajimoradlou and Kazemi, 2020)). Although oDistMult-ERAvg has been trained for out-of-sample reasoning, its performance on in-sample reasoning is almost as good as DistMult.

6 Conclusion

We studied out-of-sample representation learning for non-attributed multi-relational graphs - a problem that is surprisingly poorly studied. We created two benchmarks for this task and outlined the procedure we followed for creating these datasets to facilitate the creation of more datasets in the future. We also developed several baselines, a new training algorithm, and two aggregation models for out-of-sample representation learning. Future work includes developing new training strategies, testing other aggregation functions, combining the aggregation functions with other transductive models, extending out-of-sample reasoning to temporal KG completion and knowledge hypergraph completion (e.g., extending the proposed training algorithm and aggregation functions to the temporal or hypergraph versions of DistMult or SimplE (Goel et al., 2020; Fatemi et al., 2019)) transferring the knowledge learned over one graph to a new graph with new entities (similar to (Muhan Zhang, 2020; Teru and Hamilton, 2019)), studying the similarities and differences between out-of-sample representation learning and out-of-vocabulary word embedding, and testing the proposed models on relational domains other than knowledge graphs.

References

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

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NeurIPS*, pages 2787–2795.

Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pages 3844–3852.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *AAAI*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*.

Bahare Fatemi, Perouz Taslakian, David Vazquez, and David Poole. 2019. Knowledge hypergraphs: Prediction beyond binary relations. In *IJCAI*.

Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. 2020. Diachronic embedding for temporal knowledge graph completion. In *AAAI*.

Ainaz Hajimoradlou and Seyed Mehran Kazemi. 2020. Stay positive: Knowledge graph embedding without negative sampling. In *ICML Workshop on Graph Representation Learning and Beyond*.

Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. 2017. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *arXiv preprint arXiv:1706.05674*.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017a. Inductive representation learning on large graphs. In *NeurIPS*.

William L Hamilton, Rex Ying, and Jure Leskovec. 2017b. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40(3):52–74.

David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *ACL (1)*, pages 687–696.

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

Seyed Mehran Kazemi and David Poole. 2018. SimplE embedding for link prediction in knowledge graphs. In *NeurIPS*, pages 4289–4300.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Jianxin Ma, Peng Cui, and Wenwu Zhu. 2018. DepthLGP: learning embeddings of out-of-sample nodes in dynamic networks. In *AAAI*.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Yixin Chen Muhan Zhang. 2020. Inductive matrix completion based on graph neural networks. In *ICLR*.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016. Stranse: a novel embedding model of entities and relationships in knowledge bases. In *NAACL-HLT*.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. RotatE: Knowledge graph embedding by relational rotation in complex space. In *ICLR*.

Komal K Teru and William L Hamilton. 2019. Inductive relation prediction on knowledge graphs. *arXiv preprint arXiv:1911.06962*.

Kristina Toutanova and Danqi Chen. 2015. Observed versus latent features for knowledge base and text inference. In *Continuous Vector Space Models and their Compositionality*, pages 57–66.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *ICML*, pages 2071–2080.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. 2016. Representation learning of knowledge graphs with entity descriptions. In *AAAI*.

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

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*.

Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. 2019. Quaternion knowledge graph embedding. In *NeurIPS*.

Yu Zhao, Sheng Gao, Patrick Gallinari, and Jun Guo. 2017. Zero-shot embedding for unseen entities in knowledge graph. *IEICE Transactions on Information and Systems*, 100(7):1440–1447.