One-Shot Relational Learning for Knowledge Graphs

Wenhan Xiong[†], **Mo Yu**[*], **Shiyu Chang**[*], **Xiaoxiao Guo**[*], **William Yang Wang**[†]
[†] University of California, Santa Barbara
[*] IBM Research
{xwhan, william}@cs.ucsb.edu, yum@us.ibm.com, {shiyu.chang, xiaoxiao.guo}@ibm.com

Abstract

Knowledge graphs (KGs) are the key components of various natural language processing applications. To further expand KGs' coverage, previous studies on knowledge graph completion usually require a large number of training instances for each relation. However, we observe that long-tail relations are actually more common in KGs and those newly added relations often do not have many known triples for training. In this work, we aim at predicting new facts under a challenging setting where only one training instance is available. We propose a one-shot relational learning framework, which utilizes the knowledge extracted by embedding models and learns a matching metric by considering both the learned embeddings and one-hop graph structures. Empirically, our model yields considerable performance improvements over existing embedding models, and also eliminates the need of re-training the embedding models when dealing with newly added relations.[1]

1 Introduction

Large-scale knowledge graphs (Suchanek et al., 2007; Vrandečić and Krötzsch, 2014; Bollacker et al., 2008; Auer et al., 2007; Carlson et al., 2010) represent every piece of information as binary relationships between entities, usually in the form of triples *i.e. (subject, predicate, object)*. This kind of structured knowledge is essential for many downstream applications such as Question Answering and Semantic Web.

Despite KGs' large scale, they are known to be highly incomplete (Min et al., 2013). To automatically complete KGs, extensive research efforts (Nickel et al., 2011; Bordes et al., 2013; Yang et al., 2014; Trouillon et al., 2016; Lao and Cohen, 2010; Neelakantan et al., 2015; Xiong et al., 2017; Das et al., 2017; Chen et al., 2018) have been made to build relational learning models that could infer missing triples by learning from existing ones. These methods explore the statistical information of triples or path patterns to infer new facts of existing relations; and have achieved considerable performance on various public datasets.

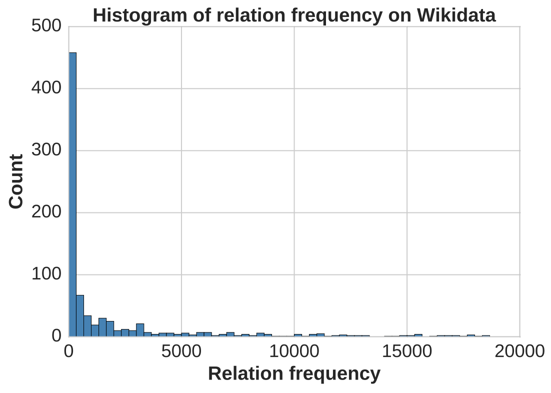

Figure 1: The histogram of relation frequencies in Wikidata. There are a large portion of relations that only have a few triples.

However, those datasets (*e.g.* FB15k, WN18) used by previous models mostly only cover common relations in KGs. For more practical scenarios, we believe the desired KG completion models should handle two key properties of KGs. First, as shown in Figure 1, a large portion of KG relations are actually long-tail. In other words, they have very few instances. But intuitively, the fewer training triples that one relation has, the more KG completion techniques could be of use. Therefore, it is crucial for models to be able to complete relations with limited numbers of triples. However, existing research usually assumes the availability of sufficient training triples for all relations, which

[1]Code and datasets could be found at `https://github.com/xwhan/One-shot-Relational-Learning`.

limits their usefulness on sparse long-tail relations.

Second, to capture up-to-date knowledge, real-world KGs are often dynamic and evolving at any given moment. New relations will be added whenever new knowledge is acquired. If a model can predict new triples given only a small number of examples, a large amount of human effort could be spared. However, to predict target relations, previous methods usually rely on well-learned representations of these relations. In the dynamic scenario, the representations of new relations cannot be sufficiently trained given limited training instances, thus the ability to adapt to new relations is also limited for current models.

In contrast to previous methods, we propose a model that depends only on the entity embeddings and local graph structures. Our model aims at learning a matching metric that can be used to discover more similar triples given one reference triple. The learnable metric model is based on a permutation-invariant network that effectively encodes the one-hop neighbors of entities, and also a recurrent neural network that allows multi-step matching. Once trained, the model will be able to make predictions about any relation while existing methods usually require fine-tuning to adapt to new relations. With two newly constructed datasets, we show that our model can achieve consistent improvement over various embedding models on the one-shot link prediction task.

In summary, our contributions are three-fold:

- We are the first to consider the long-tail relations in the link prediction task and formulate the problem as few-shot relational learning;
- We propose an effective one-shot learning framework for relational data, which achieves better performance than various embedding-based methods;
- We also present two newly constructed datasets for the task of one-shot knowledge graph completion.

2 Related Work

Embedding Models for Relational Learning Various models have been developed to model relational KGs in continous vector space and to automatically infer missing links. RESCAL (Nickel et al., 2011) is one of the earlier work that models the relationship using tensor operations. Bordes et al. (2013) proposed to model relationships in the 1-D vector space. Following this line of research, more advanced models such as DistMult (Yang et al., 2014), ComplEx (Trouillon et al., 2016) and ConvE (Dettmers et al., 2017) have been proposed. These embedding-based models usually assume enough training instances for all relations and entities and do not pay attention to those sparse symbols. More recently, several models (Shi and Weninger, 2017; Xie et al., 2016) have been proposed to handle unseen entities by leveraging text descriptions. In contrast to these approaches, our model deals with long-tail or newly added relations and focuses on one-shot relational learning without any external information, such as text descriptions of entities or relations.

Few-Shot Learning Recent deep learning based few-shot learning approaches fall into two main categories: (1) *metric based approaches* (Koch, 2015; Vinyals et al., 2016; Snell et al., 2017; Yu et al., 2018), which try to learn generalizable metrics and the corresponding matching functions from a set of training tasks. Most methods in this class adopt the general matching framework proposed in deep siamese network (Koch, 2015). One example is the Matching Networks (Vinyals et al., 2016), which make predictions by comparing the input example with a small labeled support set; (2) *meta-learner based approaches* (Ravi and Larochelle, 2017; Munkhdalai and Yu, 2017; Finn et al., 2017; Li et al., 2017), which aim to learn the optimization of model parameters (by either outputting the parameter updates or directly predicting the model parameters) given the gradients on few-shot examples. One example is the LSTM-based meta-learner (Ravi and Larochelle, 2017), which learns the step size for each dimension of the stochastic gradients. Besides the above categories, there are also some other styles of few-shot learning algorithms, *e.g.* Bayesian Program Induction (Lake et al., 2015), which represents concepts as simple programs that best explain observed examples under a Bayesian criterion.

Previous few-shot learning research mainly focuses on vision and imitation learning (Duan et al., 2017) domains. In the language domain, Yu et al. (2018) proposed a multi-metric based approach for text classification. To the best of our knowledge, this work is the first research on few-shot learning for knowledge graphs.

3 Background

3.1 Problem Formulation

Knowledge graphs $\mathcal{G}$ are represented as a collection of triples $\{(h, r, t)\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where $\mathcal{E}$ and $\mathcal{R}$ are the entity set and relation set. The task of knowledge graph completion is to either predict unseen relations r between two existing entities: $(h, ?, t)$ or predict the tail entity t given the head entity and the query relation: $(h, r, ?)$. As our purpose is to infer unseen facts for newly added or existing long-tail relations, we focus on the latter case. In contrast to previous work that usually assumes enough triples for the query relation are available for training, this work studies the case where only one training triple is available. To be more specific, the goal is to rank the true tail entity t_{true} higher than other candidate entities $t \in \mathcal{C}_{h,r}$, given only an example triple (h_0, r, t_0). The candidates set is constructed using the entity type constraint (Toutanova et al., 2015). It is also worth noting that when we predict new facts of the relation r, we only consider a closed set of entities, *i.e.* no unseen entities during testing. For open-world settings where new entities might appear during testing, external information such as text descriptions about these entities are usually required and we leave this to future work.

3.2 One-Shot Learning Settings

This section describes the settings for the training and evaluation of our one-shot learning model.

The goal of our work is to learn a metric that could be used to predict new facts with one-shot examples. Following the standard one-shot learning settings (Vinyals et al., 2016; Ravi and Larochelle, 2017), we assume access to a set of training tasks. In our problem, each training task corresponds to a KG relations $r \in \mathcal{R}$, and has its own training/testing triples: $T_r = \{D_r^{train}, D_r^{test}\}$. This task set is often denoted as the *meta-training* set, $\mathbb{T}_{meta-train}$.

To imitate the one-shot prediction at evaluation time, there is only one triple (h_0, r, t_0) in each D_r^{train}. The $D_r^{test} = \{(h_i, r, t_i, \mathcal{C}_{h_i,r})\}$ consists of the testing triples of r with ground-truth tail entities t_i for each query (h_i, r), and the corresponding tail entity candidates $\mathcal{C}_{h_i,r} = \{t_{ij}\}$ where each t_{ij} is an entity in $\mathcal{G}$. The metric model can thus be tested on this set by ranking the candidate set $\mathcal{C}_{h_i,r}$ given the test query (h_i, r) and the labeled triple in D_r^{train}. We denote an arbitrary ranking-loss function as $\ell_\theta(h_i, r, t_i|\mathcal{C}_{h_i,r}, D_r^{train})$, where θ represents the parameters of our metric model. This loss function indicates how well the metric model works on tuple $(h_i, r, t_i, \mathcal{C}_{h_i,r})$ while observing only one-shot data from D_r^{train}. The objective of training the metric model, *i.e.* the meta-training objective, thus becomes:

$$\min_\theta \mathbb{E}_{T_r}\left[\sum_{(h_i,r,t_i,\mathcal{C}_{h_i,r})\in D_r^{test}} \frac{\ell_\theta(h_i, r, t_i|\mathcal{C}_{h_i,r}, D_r^{train})}{|D_r^{test}|}\right], \tag{1}$$

where T_r is sampled from the meta-training set $\mathbb{T}_{meta-train}$, and $|D_r^{test}|$ denotes the number of tuples in D_r^{test}.

Once trained, we can use the model to make predictions on new relations $r' \in \mathcal{R}'$, which is called the *meta-testing* step in literature. These meta-testing relations are unseen from meta-training, *i.e.* $\mathcal{R}' \cap \mathcal{R} = \phi$. Each meta-testing relation r' also has its own one-shot training data $D_{r'}^{train}$ and testing data $D_{r'}^{test}$, defined in the same way as in meta-training. These meta-testing relations form a meta-test set $\mathbb{T}_{meta-test}$.

Moreover, we leave out a subset of relations in $\mathbb{T}_{meta-train}$ as the meta-validation set $\mathbb{T}_{meta-validation}$. Because of the assumption of one-shot learning, the meta-testing relations do not have validation sets like in the traditional machine learning setting. Otherwise, the metric model will actually see more than one-shot labeled data during meta-testing, thus the one-shot assumption is violated.

Finally, we assume that the method has access to a *background knowledge graph* $\mathcal{G}'$, which is a subset of $\mathcal{G}$ with all the relations from $\mathbb{T}_{meta-train}$, $\mathbb{T}_{meta-validation}$ and $\mathbb{T}_{meta-test}$ removed.

4 Model

In this section, we describe the proposed model for similarity metric learning and also the corresponding loss function ℓ we use to train our model.

The core of our proposed model is a similarity function $\mathcal{M}((h, t), (h', t')|\mathcal{G}')$. Thus for any query relation r, as long as there is one known fact (h_0, r, t_0), the model could predict the likelihood of testing triples $\{(h_i, r, t_{ij})|t_{ij} \in \mathcal{C}_{h_i,r}\}$, based on the matching score between each (h_i, t_{ij}) and (h_0, t_0). The implementation of the above matching function involves two sub-problems: (1) the representations of entity pairs; and (2) the comparison function between two entity-pair representations. Our overall model, as shown in Figure 2,

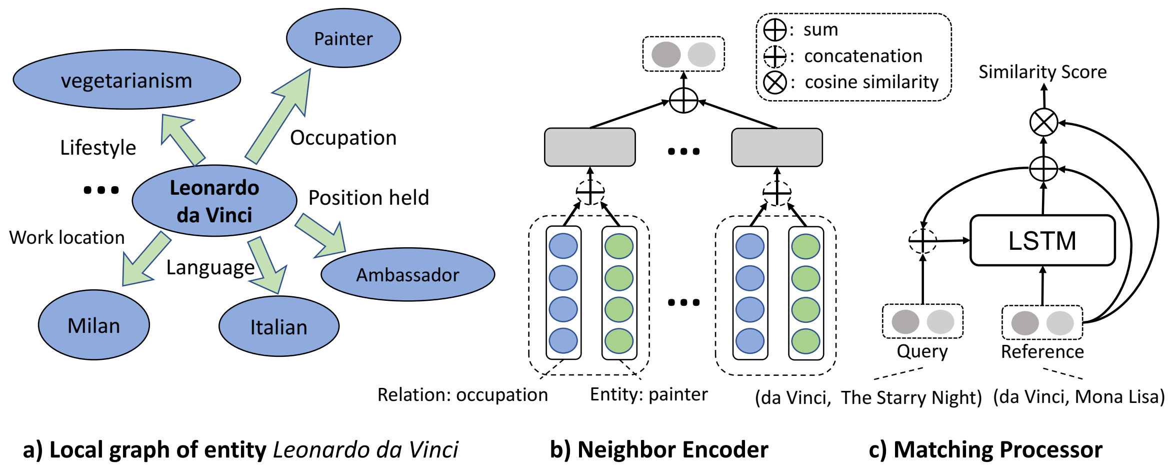

Figure 2: a) and b): Our **neighbor encoder** operating on entity *Leonardo da Vinci*; c): The **matching processor**.

deals with the above two problems with two major components respectively:

• **Neighbor encoder** (Figure 2b), aims at utilizing the local graph structure to better represent entities. In this way, the model can leverage more information that KG provides for every entity within an entity pair.

• **Matching processor** (Figure 2c), takes the vector representations of any two entity pairs from the neighbor encoder; then performs multi-step matching between two entity-pairs and outputs a scalar as the similarity score.

4.1 Neighbor Encoder

This module is designed to enhance the representation of each entity with its local connections in knowledge graph.

Although the entity embeddings from KG embedding models (Bordes et al., 2013; Yang et al., 2014) already have relational information encoded, previous work (Neelakantan et al., 2015; Lin et al., 2015a; Xiong et al., 2017) showed that explicitly modeling the structural patterns, such as paths, is usually beneficial for relationship prediction. In view of this, we propose to use a neighbor encoder to incorporate graph structures into our metric-learning model. In order to benefit from the structural information while maintaining the efficiency to easily scale to real-world large-scale KGs, our neighbor encoder only considers entities' local connections, *i.e.* the one-hop neighbors.

For any given entity e, its local connections form a set of (*relation, entity*) tuples. As shown in Figure 2a, for the entity *Leonardo da Vinci*, one of such tuples is (*occupation, painter*). We refer this neighbor set as as $\mathcal{N}_e = \{(r_k, e_k)|(e, r_k, e_k) \in \mathcal{G}'\}$. The purpose of our neighbor encoder is to encode $\mathcal{N}_e$ and output a vector as the latent representation of e. Because this is a problem of encoding sets with varying sizes, we hope the encoding function can be (1) invariant to permutations and also (2) insensitive to the size of the neighbor set. Inspired by the results from (Zaheer et al., 2017), we use the following function f that satisfies the above properties:

$$f(\mathcal{N}_e) = \sigma(\frac{1}{|\mathcal{N}_e|} \sum_{(r_k,e_k)\in\mathcal{N}_e} C_{r_k,e_k}). \qquad (2)$$

where C_{r_k,e_k} is the feature representation of a relation-entity pair (r_k, e_k) and σ is the activation function. In this paper we set $\sigma = \tanh$ which achieves the best performance on $\mathbb{T}_{meta-validation}$.

To encode every tuple $(r_k, e_k) \in \mathcal{N}_e$ into C_{r_k,e_k}, we first use an embedding layer **emb** with dimension d (which can be pre-trained using existing embedding-based models) to get the vector representations of r_k and e_k:

$$v_{r_k} = \mathbf{emb}(r_k), v_{e_k} = \mathbf{emb}(e_k)$$

Dropout (Srivastava et al., 2014) is applied here to the vectors v_{r_k}, v_{e_k} to achieve better generalization. We then apply a feed-forward layer to encode the interaction within this tuple:

$$C_{r_k,e_k} = W_c(v_{r_k} \oplus v_{e_k}) + b_c, \qquad (3)$$

where $W_c \in R^{d\times 2d}, b_c \in R^d$ are parameters to be learned and $\oplus$ denotes concatenation.

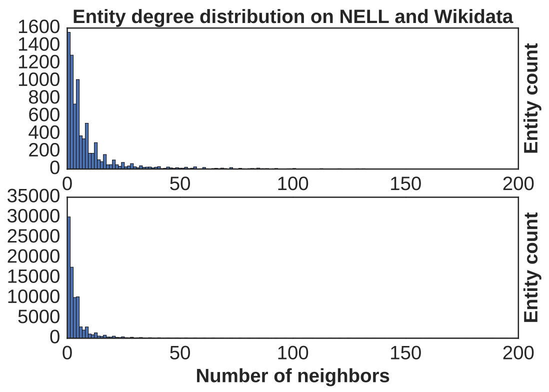

Figure 3: The distribution of entities' degrees (numbers of neighbors) on our two datasets. Since we work on closed-set of entities, we draw the figure by considering the intersection between entities in our background knowledge $\mathcal{G}'$ and the entities appearing in $\mathbb{T}_{meta-train}$, $\mathbb{T}_{meta-validation}$ or $\mathbb{T}_{meta-test}$. Note that all triples in $\mathbb{T}_{meta-train}$, $\mathbb{T}_{meta-validation}$ or $\mathbb{T}_{meta-test}$ are removed from $\mathcal{G}'$. Upper: NELL; Lower: Wikidata.

To enable batching during training, we manually specify the maximum number of neighbors and use all-zero vectors as "dummy" neighbors. Although different entities have different degrees (number of neighbors), the degree distribution is usually very concentrated, as shown in Figure 3. We can easily find a proper bound as the maximum number of neighbors to batch groups of entities.

The neighbor encoder module we propose here is similar to the Relational Graph Convolutional Networks (Schlichtkrull et al., 2017) in the sense that we also use the shared kernel $\{W_c, b_c\}$ to encode the neighbors of different entities. But unlike their model that operates on the whole graph and performs multiple steps of information propagation, we only encode the local graphs of the entities and perform one-step propagation. This enables us to easily apply our model to large-scale KGs such as Wikidata. Besides, their model also does not operate on pre-trained graph embeddings. We leave the investigation of other graph encoding strategies, e.g. (Xu et al., 2018; Song et al., 2018), to future work.

4.2 Matching Processor

Given the neighbor encoder module, now we discuss how we can do effective similarity matching based on our recurrent matching processor. By applying $f(\mathcal{N}_e)$ to the reference entity pair (h_0, t_0) and any query entity pair (h_i, t_{ij}), we get two

Algorithm 1 One-shot Training

```
 1: Input:
 2: a) Meta-training task set $\mathbb{T}_{meta-training}$;
 3: b) Pre-trained KG embeddings (excluding relation in $\mathbb{T}_{meta-training}$);
 4: c) Initial parameters $\theta$ of the metric model;
 5: for epoch = 0:M-1 do
 6:     Shuffle the tasks in $\mathcal{T}_{meta-learning}$
 7:     for $T_r$ in $\mathcal{T}_{meta-learning}$ do
 8:         Sample one triple as the reference
 9:         Sample a batch $B^+$ of query triples
10:         Pollute the tail entity of query triples to get $B^-$
11:         Calculate the matching scores for triple in $B^+$ and $B^-$
12:         Calculate the batch loss $\mathcal{L} = \sum_B \ell$
13:         Update $\theta$ using gradient $g \propto \nabla\mathcal{L}$
14:     end for
15: end for
```

neighbor vectors for each: $[f(\mathcal{N}_{h_0}); f(\mathcal{N}_{t_0})]$ and $[f(\mathcal{N}_{h_i}); f(\mathcal{N}_{t_{ij}})]$. To get a similarity score that can be used to rank (h_i, t_{ij}) among other candidates, we can simply concatenate the $f(\mathcal{N}_h)$ and $f(\mathcal{N}_t)$ in each pair to form a single pair representation vector, and calculate the cosine similarity between pairs. However, this simple metric model turns out to be too shallow and does not give good performance. To enlarge our model's capacity, we leverage a LSTM-based (Hochreiter and Schmidhuber, 1997) recurrent "processing" block (Vinyals et al., 2015, 2016) to perform multi-step matching. Every process step is defined as follows:

$$\begin{aligned} h'_{k+1}, c_{k+1} &= LSTM(q, [h_k \oplus s, c_k]) \\ h_{k+1} &= h'_{k+1} + q \\ score_{k+1} &= \frac{h_{k+1} \odot s}{\|h_{k+1}\| \, \|s\|}, \end{aligned} \tag{4}$$

where $LSTM(x, [h, c])$ is a standard LSTM cell with input x, hidden state h and cell state c, and $s = f(\mathcal{N}_{h_0}) \oplus f(\mathcal{N}_{t_0}), q = f(\mathcal{N}_{h_i}) \oplus f(\mathcal{N}_{t_{ij}})$ are the concatenated neighbor vectors of the reference pair and query pair. After K processing steps[2], we use $score_K$ as the final similarity score between the query and support entity pair. For every query $(h_i, r, ?)$, by comparing (h_i, t_{ij}) with (h_0, t_0), we can get the ranking scores for every $t_{ij} \in \mathcal{C}_{h_i,r}$.

4.3 Loss Function and Training

For a query relation r and its reference/training triple (h_0, r, t_0), we collect a group of positive (true) query triples $\{(h_i, r, t_i^+)|(h_i, r, t_i^+) \in \mathcal{G}\}$ and construct another group negative (false) query

[2] K is a hyperparameter to be tuned.

triples $\{(h_i, r, t_i^-)|(h_i, r, t_i^-) \notin \mathcal{G}\}$ by polluting the tail entities. Following previous embedding-based models, we use a hinge loss function to optimize our model:

$$\ell_\theta = max(0, \gamma + score_\theta^- - score_\theta^+), \quad (5)$$

where $score_\theta^+$ and $score_\theta^-$ are scalars calculated by comparing the query triple $(h_i, r, t_i^+/t_i^-)$ with the reference triple (h_0, r, t_0) using our metric model, and the margin γ is a hyperparameter to be tuned. For every training episode, we first sample one task/relation T_r from the meta-training set $\mathbb{T}_{meta-training}$. Then from all the known triples in T_r, we sample one triple as the reference/training triple D_r^{train} and a batch of other triples as the positive query/test triples D_r^{test}. The detail of the training process is shown in Algorithm 1. Our experiments are discussed in the next section.

5 Experiments

5.1 Datasets

Dataset	# Ent.	# R.	# Triples	# Tasks
NELL-One	68,545	358	181,109	67
Wiki-One	4,838,244	822	5,859,240	183

Table 1: Statistics of the Datasets. # Ent. denotes the number of unique entities and # R. denotes the number of all relations. # Tasks denotes the number of relations we use as one-shot tasks.

Existing benchmarks for knowledge graph completion, such as FB15k-237 (Toutanova et al., 2015) and YAGO3-10 (Mahdisoltani et al., 2013) are all small subsets of real-world KGs. These datasets consider the same set of relations during training and testing and often include sufficient training triples for every relation. To construct datasets for one-shot learning, we go back to the original KGs and select those relations that do not have too many triples as one-shot task relations. We refer the rest of the relations as background relations, since their triples provide important background knowledge for us to match entity pairs.

Our first dataset is based on NELL (Mitchell et al., 2018), a system that continuously collects structured knowledge by reading webs. We take the latest dump and remove those inverse relations. We select the relations with less than 500 but more than 50 triples[3] as one-shot tasks. To show that our model is able to operate on large-scale KGs, we follow the similar process to build another larger dataset based on Wikidata (Vrandečić and Krötzsch, 2014). The dataset statistics are shown in Table 1. Note that the Wiki-One dataset is an order of magnitude larger than any other benchmark datasets in terms of the numbers of entities and triples. For NELL-One, we use 51/5/11 task relations for training/validation/testing. For Wiki-One, the division ratio is 133:16:34.

[3]We want to have enough triples for evaluation.

5.2 Implementation Details

In our experiments, we consider the following embedding-based methods: RESCAL (Nickel et al., 2011), TransE (Bordes et al., 2013), DistMult (Yang et al., 2014) and ComplEx (Trouillon et al., 2016). For TransE, we use the code released by Lin et al. (2015b). For the other models, we have tried the code released by Trouillon et al. (2016) but it gives much worse results than TransE on our datasets. Thus we use our own implementations based on PyTorch (Paszke et al., 2017) for comparison. When evaluating existing embedding models, during training, we use not only the triples of background relations but also all the triples of the training relations and the one-shot training triple of those validation/test relations. However, since the proposed metric model does not require the embeddings of query relations, we only include the triples of the background relations for embedding training. As TransE and DistMult use 1-D vectors to represent entities and relations, they can be directly used in our natching model. While for RESCAL, since it uses matrices to represent relations, we employ mean-pooling over these matrices to get 1-D embeddings. For the ComplEx model, we use the concatenation of the real part and imaginary part. The hyperparameters of our model are tuned on the validation task set and can be found in the appendix.

Apart from the above embedding models, a more recent method (Dettmers et al., 2017) applies convolution to model relationships and achieves the best performance on several benchmarks. For every query $(h, r, ?)$, their model enumerates the whole entity set to get positive and negative triples for training. We find that this training paradigm takes lots of computational resources when dealing with large entity sets and cannot scale to real-world KGs such as Wikidata[4] that have millions

[4]On a GPU card with 12GB memory, we fail to run their ConvE model on Wiki-One with batch size 1.

	NELL-One				Wiki-One			
Model	MRR	Hits@10	Hits@5	Hits@1	MRR	Hits@10	Hits@5	Hits@1
RESCAL	.071/.140	.100/.229	.082/.186	.048/.089	.119/.072	.167/.082	.132/.062	.093/.051
TransE	.082/.093	.177/.192	.126/.141	.032/.043	.023/.035	.036/.052	.029/.043	.015/.025
DistMult	.075/.102	.128/.177	.093/.126	.045/.066	.042/.048	.086/.101	.055/.070	.017/.019
ComplEx	.072/.131	.128/.223	.041/.086	.041/.086	.079/.069	.148/.121	.106/.092	.046/.040
GMatching (RESCAL)	.144/**.188**	.277/.305	.216/.243	.087/**.133**	.113/.139	.330/.305	.180/.228	.033/.061
GMatching (TransE)	**.168**/.171	.293/.255	**.239**/.210	**.103**/.122	.167/.219	.349/.328	.289/.269	.083/.163
GMatching (DistMult)	.119/.171	.238/.301	.183/.221	.054/.114	.190/**.222**	**.384/.340**	**.291**/.271	.114/**.164**
GMatching (ComplEx)	.132/.185	**.308/.313**	.232/**.260**	.049/.119	**.201**/.200	.350/.336	.231/**.272**	**.141**/.120
GMatching (Random)	.083/.151	.211/.252	.135/.186	.024/.103	.174/.198	.309/.299	.222/.260	.121/.133

Table 2: Link prediction results on validation/test relations. KG embeddings baselines are shown at the top of the table and our one-shot learning (GMatching) results are shown at the bottom. **Bold** numbers denote the best results on meta-validation/meta-test. Underline numbers denote the model selection results from all KG embeddings baselines, or from all one-shot methods, i.e. selecting the method with the best validation score and reporting the corresponding test score.

of entities. For the scalability concern, our experiments only consider models that use negative sampling for training.

5.3 Results

The main results of our methods are shown in Table 2. We denote our method as "GMatching" since our model is trained to match local graph patterns. We use mean reciprocal rank (MRR) and Hits@K to evaluate different models. We can see that our method produces consistent improvements over various embedding models on these one-shot relations. The improvements are even more substantial on the larger Wiki-One dataset. To investigate the learning power of our model, we also try to train our metric model with randomly initialized embeddings. Surprisingly, although the results are worse than the metric models with pre-trained embeddings, they are still superior to the baseline embedding models. This suggests that, by incorporating the neighbor entities into our model, the embeddings of many relations and entities actually get updated in an effective way and provide useful information for our model to make predictions on test data.

It is worth noting that once trained, our model can be used to predict any newly added relations without fine-tuning, while existing models usually need to be re-trained to handle those newly added symbols. On a large real-world KG, this re-training process can be slow and highly computational expensive.

Remark on Model Selection Given the existence of various KG embedding models, one interesting experiment is to incorporate model selection into hyper-parameter tuning and choose the best validation model for testing.

If we think about comparing KG embedding and metric learning as two approaches, the results from the model selection process can then be used as the "final" measurement for comparison. For example, the baseline KG embedding achieves best MRR on Wiki-One with RESCAL (11.9%), so we report the corresponding testing MRR (7.2%) as the final model selection result for KG embedding approach. In this way, at the top half of Table 2, we select the best KG embedding method according to the validation performance. The results are highlighted with underlines. Similarly, we select the best metric learning approach at the bottom.

Our metric-based method outperforms KG embedding by a large margin from this perspective as well. Taking MRR as an example, the selected metric model achieves 17.1% on NELL-One and 20.0% on Wiki-One; while the results of KG embedding are 9.3% and 7.2%. The improvement is 7.8% and 12.8% respectively.

5.4 Analysis on Neighbor-Encoder

As our model leverages entities' local graph structures by encoding the neighbors, here we try to investigate the effect of the neighbor set by restricting the maximum number of neighbors. If the size of the true neighbor set is larger than the maximum limit, the neighbors are then selected by random sampling. Figure 4 shows the learning curves of different settings. These curves are based on the Hits@10 calculated on the validation set. We see that encoding more neighbors

		MRR		Hits@10	
Relations	# Candidates	GMatching	ComplEx	GMatching	ComplEx
sportsGameSport	123	0.424	**0.466**(0.139*)	**1.000**	0.479(0.200*)
athleteInjuredHisBodypart	299	0.025	0.026(**0.330***)	0.015	0.059(**0.444***)
animalSuchAsInvertebrate	786	0.447	0.333(**0.555***)	0.626	0.587(**0.783***)
automobilemakerDealersInCountry	1084	**0.453**	0.245(0.396*)	**0.821**	0.453(0.500*)
sportSchoolIncountry	2100	**0.534**	0.324(0.294*)	**0.745**	0.529(0.571*)
politicianEndorsesPolitician	2160	0.093	0.026(**0.194***)	0.226	0.047(**0.357***)
agriculturalProductFromCountry	2222	**0.120**	0.029(0.042*)	**0.288**	0.058(0.086*)
producedBy	3174	0.085	0.040(**0.165***)	0.179	0.075(**0.241***)
automobilemakerDealersInCity	5716	0.026	0.024(**0.041***)	0.040	0.051(**0.174***)
teamCoach	10569	0.017	0.065(**0.376***)	0.024	0.079(**0.547***)
geopoliticalLocationOfPerson	11618	0.028	0.016(**0.284***)	0.035	0.035(**0.447***)

Table 3: Results decomposed over different relations. "*" denotes the results with standard training settings and "# Candidates" denotes the size of candidate entity set.

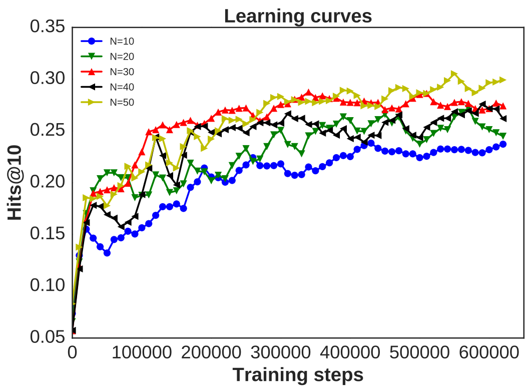

Figure 4: The learning curves on NELL-one. Every run uses different number of neighbors. The y-axis is Hits@10 calculated on all the validation relations.

Configuration	Hits@10
Full Model with ComplEx	**.308/.313**
w/o Matching Processor	.266/.269
w/o Neighbor Encoder	.248/.296
w/o Scaling Factor	.229/.219

Table 4: Ablation on different components.

for every entity generally leads to better performance. We also observe that the model that encodes 40 neighbors in maximum actually yields worse performance than the model that only encodes 30 neighbors. We think the potential reason is that for some entity pairs, there are some local connections that are irrelevant and provide noisy information to the model.

5.5 Ablation Studies

We conduct ablation studies using the model that achieves the best Hits@10 on the NELL-One dataset. The results are shown in Table 4. We use Hits@10 on validation and test set for comparison, as the hyperparameters are selected using this evaluation metric. We can see that both the matching processor[5] and the neighbor encoder play important roles in our model. Another important observation is that the scaling factor $1/\mathcal{N}_e$ turns out to be very essential for the neighbor encoder. Without scaling, the neighbor encoder actually gives worse results compared to the simple embedding-based matching.

5.6 Performance on Different Relations

When testing various models, we observe that the results on different relations are actually of high variance. Table 3 shows the decomposed results on NELL-One generated by our best metric model (GMatching-ComplEx) and its corresponding embedding method. For reference, we also report the embedding model's performance under standard training settings where 75% of the triples (instead of only one) are used for training and the rest are used for testing. We can see that relations with smaller candidate sets are generally easier and our model could even perform better than the embedding model trained under standard settings. For some relations such as *athleteInjuredHisBodypart*, their involved entities have very few connections in KG. It is as expected that one-shot learning on these kinds of relations is quite challenging. Those relations with lots of (>3000) candidates are challenging for all models. Even for embedding model with more training triples, the performance on some relations is still very limited. This suggests that the knowledge graph completion task is still far from being solved.

[5]Matching without Matching Processor is equivalent to matching using simple cosine similarity.

6 Conclusion

This paper introduces a one-shot relational learning framework that could be used to predict new facts of long-tail relations in KGs. Our model leverages the local graph structure of entities and learns a differentiable metric to match entity pairs. In contrast to existing methods that usually need finetuning to adapt to new relations, our trained model can be directly used to predict any unseen relation and also achieves much better performance in the one-shot setting. Our future work might consider incorporating external text data and also enhancing our model to make better use of multiple training examples in the few-shot learning case.

Acknowledgments

This research is supported by an IBM Faculty Award. We also thank the anonymous reviewers for their useful feedback.

References

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM.

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

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. 2010. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3. Atlanta.

Wenhu Chen, Wenhan Xiong, Xifeng Yan, and William Wang. 2018. Variational knowledge graph reasoning. *arXiv preprint arXiv:1803.06581.*

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851.*

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2017. Convolutional 2d knowledge graph embeddings. *arXiv preprint arXiv:1707.01476.*

Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. 2017. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400.*

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

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

Gregory Koch. 2015. *Siamese neural networks for one-shot image recognition.* Ph.D. thesis, University of Toronto.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Ni Lao and William W Cohen. 2010. Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 81(1):53–67.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. 2017. Meta-sgd: Learning to learn quickly for few shot learning. *arXiv preprint arXiv:1707.09835.*

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. 2015a. Modeling relation paths for representation learning of knowledge bases. *arXiv preprint arXiv:1506.00379.*

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015b. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, volume 15, pages 2181–2187.

Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. 2013. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*.

Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. 2013. Distant supervision for relation extraction with an incomplete knowledge base. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 777–782.

Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, B Yang, J Betteridge, A Carlson, B Dalvi, M Gardner, B Kisiel, et al. 2018. Never-ending learning. *Communications of the ACM*, 61(5):103–115.

Tsendsuren Munkhdalai and Hong Yu. 2017. Meta networks. *arXiv preprint arXiv:1703.00837*.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. 2015. Compositional vector space models for knowledge base inference. In *2015 aaai spring symposium series*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816.

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.

Sachin Ravi and Hugo Larochelle. 2017. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, volume 1, page 6.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2017. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*.

Baoxu Shi and Tim Weninger. 2017. Open-world knowledge graph completion. *arXiv preprint arXiv:1711.03438*.

Jake Snell, Kevin Swersky, and Richard S Zemel. 2017. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. *arXiv preprint arXiv:1805.02473*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509.

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

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2015. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. 2016. Representation learning of knowledge graphs with entity descriptions. In *AAAI*, pages 2659–2665.

Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*.

Kun Xu, Lingfei Wu, Zhiguo Wang, and Vadim Sheinin. 2018. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. 2018. Diverse few-shot text classification with multiple metrics. *arXiv preprint arXiv:1805.07513*.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3394–3404.

A Hyperparameters

For the NELL dataset, we set embedding size as 100. For Wikidata, we set the embedding size as 50 for faster training with millions of triples. The embeddings are trained for 1,000 epochs. The other hyperparamters are tuned using the Hits@10 metric[6] on the validation tasks. For matching steps, the optimal setting is 2 for NELL-One and 4 for Wiki-One. For the number of neighbors, we find that the maximum limit 50 works the best for both datasets. For parameter updates, we use Adam (Kingma and Ba, 2014) with the initial learning rate 0.001 and we half the learning rate after 200k update steps. The margin used in our loss function is 5.0. The dimension of LSTM's hidden size is 200.

B Few-Shot Experiments

Metrics	GMatching		ComplEx	
	1-shot	5-shot	1-shot	5-shot
MRR	.132/.185	.178/.201	.072/.131	.113/.200
Hits@10	.308/.313	.307/.311	.128/.223	.221/.325
Hits@5	.232/.260	.241/.264	.041/.086	.160/.269
Hits@1	.049/.119	.109/.143	.041/.086	.113/.133

Table 5: 5-shot experiments on NELL-One.

[6]The percentage of correct answer ranks within top10.