Fine-grained Entity Typing for Relation-Sparsity Entities

No Author Given

No Institute Given

Abstract. This paper works on fine-grained entity typing without using external knowledge for Knowledge Graphs (KGs). Aiming at identifying the semantic type of an entity, this task has been studied predominantly in KGs. Provided with dense enough relations among entities, the existing mainstream KG embedding based approaches could achieve great performance on the task. However, many entities are sparse in their relations with other entities in KGs, which fails the existing KG embedding models in fine-grained entity typing. In this paper, we propose a novel KG embedding model for relation-sparsity entities in KGs. In our model, we map all attributes and types into the same vector sapce, where attributes could be granted with different weights according to an employed attention mechanism, while attribute values could be trained as bias vecotrs from attribute vectors pointing to type vectors. Based on this KG embedding model, we perform entity typing from coarse-grained level to more fine-grained level hierarchically. Besides, we also propose ways to utilize zero-shot attribute values that never appear in the training set. Our experiments performed on real-world KGs show that our approach is superior to the most advanced models in most cases.

Keywords: Fine-grained Entity Typing · Knowledge Graph Embedding · Knowledge Graph

1 Introduction

Type information of entities is very important in Knowledge Graphs (KGs). Unfortunately, many entities' type information is usually missing even in some well-known KGs such as Yago [20] and DBPedia [1]. To complete the missing type information in KGs, the task of **entity typing** [15] is proposed, aiming at identifying the semantic type (e.g., Artist) of an entity (e.g., Leonardo da Vinci) in KGs.

While traditional entity typing approaches only focus on assigning entities with a small set of coarse-grained types including Person, Organization, Location and Others [19], **fine-grained entity typing** assigns more specific types to entities, which could form a type-path in the type hierarchy in KGs [18]. As the example shown in Fig. 1, "Leonardo da Vinci" is associated with a type-path *"thing/person/artist/painter"*. Apparently, fine-grained types (e.g., *Painter* and *Artist*) make more sense in data mining than coarse-grained types (e.g., *Person*)

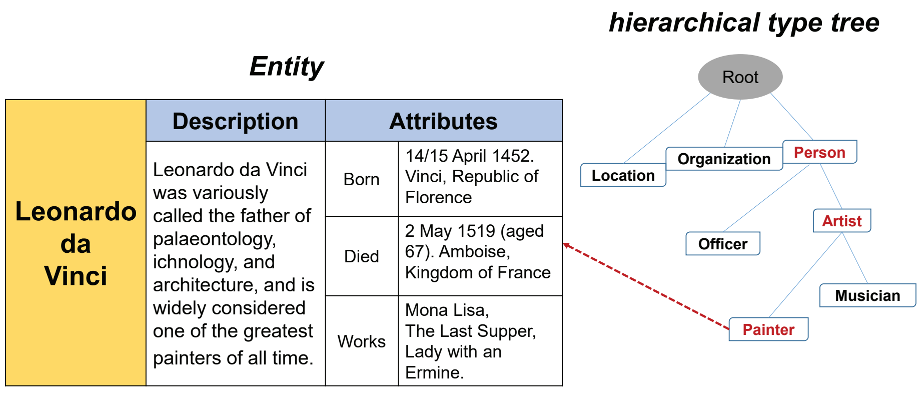
Entity
hierarchical type tree
Leonardo da Vinci
Description
Attributes
Leonardo da Vinci was variously called the father of palaeontology, ichnology, and architecture, and is widely considered one of the greatest painters of all time.
Born
14/15 April 1452. Vinci, Republic of Florence
Died
2 May 1519 (aged 67). Amboise, Kingdom of France
Works
Mona Lisa, The Last Supper, Lady with an Ermine.
Root
Location
Organization
Person
Officer
Artist
Musician
Painter

Fig. 1. An Example of Fine-grained Entity Typing in KG

since they provide us with more specific semantic information [22]. Therefore, the more fine-grained the types are, the more instrumental they would be in many of the KG-based tasks, such as knowledge base completion [5], entity linking [12], relation extraction [13], and question answering [25].

Plenty of work has been done on fine-grained entity typing. While traditional information extraction based approaches focus on extracting type information for entities from external text resource [3, 14, 26], more and more recent work tend to infer missing entity types for entities based on KGs' internal information. The existing work on KG-based fine-grained entity typing mainly relies on KG embedding for entity typing, i.e., the entities are first embedded based on information in KGs including relations, continuous attribute values and descriptions etc., and then classified into different semantic types according to their embedded results. However, although the TransE [2] and its variants [10, 11, 21] are widely applied to many KG-relevant applications, they are helpless to those entities having sparse relations with other entities. Some work also infers missing types for entities according to the embedding results of entities based on their text descriptions [16]. But text descriptions are not always in high-quality. Recent work inputs the relations of entities and continuous attributes into a multi-layer perceptron to train the representation of entities for entity typing and achieves state-of-the-art results [8]. However, without dense enough relations among entities for embedding learning, they are difficult to achieve desirable results. In practice, there are a large proportion of entities having very sparse relations with other entities in KGs. As listed in Table 1, there are more than 20% entities having no more than 3 relational triples with other entities on DBpedia, CN-DBpedia, and Yogo3. On the other hand, the average number of triples for such entities are not that small, which can also be observed in Table 1. While

Table 1. Percentage of Entities having Sparse Relations (1, 2, or 3) with Other Entities, and the Avarage Number of Triples with them on DBpedia, CN-DBpedia and Yago3

	0 Relation		1 Relation		2 Relation		3 Relation	
DBpedia	5%	3.5	2%	3.9	7%	5.1	12%	6.3
CN-DBpedia	11%	4.3	6%	4.7	5%	5.3	9%	6.1
Yogo3	7%	3.8	3%	4.4	6%	5.6	10%	6.4

some of these triples are just attribute triples describing an attribute and corresponding attribute value of an entity, other triples are "unlinked" relational triples, which have their tail entity mentions unlinked to their corresponding KG entities.

To address fine-grained entity typing for relation-sparsity entities in KGs, this paper proposes a novel KG embedding model based on attributes and attribute values of entities. Particularly, this KG embedding model maps all the attributes and types into the same vector space in a TransE-like way, where "unlinked" relational triples are also taken as attribute triples. In this model, attributes are granted with different weights according to a selective attention mechanism [7], while attribute values could be trained as bias vectors from attribute vectors to type vectors. Based on this KG embedding model, we perform entity typing from coarse-grained level to more fine-grained level hierarchically. Besides, it is common to meet zero-shot attribute values that never appear in the training set in the entity typing process. To handle these special cases, we also design a similarity measurement to find a set of closest attribute values to denote the zero-shot one, such that the robustness of our model could be further improved.

We summarize our contributions as follows:

- We propose a new KG embedding model based on attributes and attribute values, which is particularly designed for fine-grained entity typing to relation-sparsity entities in KGs.
- Based on this embedding model, we then propose to perform entity typing from coarse-grained level to more fine-grained level hierarchically.
- We design an algorithm to handle the entities with untrained (zero-shot) triple tails to ensure the robustness of our model.

We use two datasets from real-world KGs for experimental study. Our experiments performed on these two KGs show that our approach is superior to the most advanced models in most cases.

Roadmap. The rest of the paper is organized as follows: We cover the related work in Sec. 2, and then formulate the problem in Sec. 3. After present our approach in Sec. 4, we report our empirical study in Sec. 5. We finally conclude in Sec. 6.

2 Related Work

Entity typing is a long-standing problem in the Knowledge Graph (KG) construction research field. In this section, we first introduce the traditional solutions using external textual semantic information, and then cover the mainstream methods on entity typing within KGs.

2.1 Entity Typing with Texts

The goal of entity typing is to give entities more specific types after they have been recognized from text. As the number of types and the complexity of the problem increases, researchers try many ways to organize hierarchical information of types [26]. In recent work, Choi et al. [3] constructs an ultra-fine-grained dataset with 10,201 types at the most fine granularity. Based on the same dataset, Federico et al. [14] map all types onto a sphere space and train a transpose matrix to obtain the types of entities. This work achieve the state-of-art results in entity typing with texts. However, the information used for entity typing is usually about the sentences themselves, so the performance of entity typing in fine granularity is still unsatisfactory.

2.2 Entity Typing in KGs

The classification of entities KG becomes a classical problem that refers to KG completion. In KG completion, KG embedding is often used to solve such problems. For example, TransE [2], which is the basic of all KG embedding methods, trains vector expressions of entities and relationships based on relations between entities. In TransE, relations are trained as transitions from head entities to tail entities, which can be expressed as $\mathbf{h} + \mathbf{r} = \mathbf{t}$. Since then, various KG embedding methods focus on how to obtain a better representation of KG based on the relations between entities, such as TransR [11], PTrans [10], TransH [21] and so on.

Another kind of methods is to obtain the low-dimensional vector representations of entities semantics and then use the vectors to classify the entities. These methods are based on various kinds information in KGs rather than just relational triples. For example, Neelakantan and Chang [16] generate feature vectors from the description of entities in KG. Xu et al. [22] adopt a multi-labelled hierarchical classification method to assign Chinese entities of DBpedia types according to attributes and category information. In recent work, Jin et al. [8] comprehensively consider the relationship between entities and continuous attribute values, and combine them with a multi-layer perceptron to obtain the semantic vectors of entities.

The above methods consider entity typing in a complete KG. Nevertheless, there are many relation-sparsity entities in KGs during the actual construction process. These relation-sparsity entities tend to have many triples with unsplit and unlinked tails, so that the above methods may fail to address such triples. Our approach is designed to solve the problem of entity typing in this case. We take full use of discontinuous attribute values which mostly come from unsplit and unlinked tails to ensure that the relation information in such tails is not lost.

3 Problem Formulation

A typical KG consists of a number of facts, usually in the form of *triples* denoted by *(head, predicate, tail)*, where *head* is the *subject entity* and *tail* is either the *object entity* or an *attribute value* of the subject entity. We call a triple as a *relational triple* if the *object* of the triple is an entity and the *predicate* denotes the relation between the two entities. And we call a triple as an *attribute triple* if the *predicate* denotes an attribute of the entity [6].

Given a KG with a hierachical type tree such as the one shown in Fig. 1, the task of fine-grained entity typing aims at finding fine-grained semantic types for entities with missing type information in the KG, w.r.t. the given hierachical type tree. More formally, we give the relevant definitions with fine-grained entity typing task as follows.

Definition 1. (Knowledge Graph). *Knowledge graph* $KG = \{E, RT, AT\}$ *is defined as a set of entities* E*, their relation triples* RT *and their attribute triples* AT*.*

Definition 2. (Hierarchical Type Tree). *Hierarchical type tree organizes types in the form of a tree which provides hypernym-hyponym relations between types. Formally, Hierarchical type tree* $Ttr = \{TS, TR\}$ *is the set of types* TS *and the relations between types* TR*.*

Definition 3. (Fine-grained Entity Typing). *Given a knowledge graph* $KG = \{E, RT, AT\}$ *and a hierarchical type tree* $Ttr = \{TS, TR\}$*,* ***Fine-grained Entity Typing*** *aims to find a path* $\{t_1, t_2 ... t_n\}$ *in* Ttr *for each entity in* E*, where* t_i *is the hypernym of* t_{i-1}*.*

Example 1. As shown in Fig. 1, **hierarchical type tree** is a tree which reflects hypernym-hyponym relations between types, while **knowledge graph** is a collection which contains entities like "Leonardo da Vinci" and their attributes and relations between each other. The task of **fine-grained entity typing** is to find a type path in **hierarchical type tree** for each entity in **knowledge graph**, such as *"thing/person/artist/painter"* for *"Leonardo da Vinci"*.

4 Our Approach

The architecture of our approach is given in Fig. 2. Our model mainly consists of two modules: embedding layer and replacing layer. We obtain vector representations of triples at the embedding layer which needs labeled entities and a hierarchical type tree Ttr as input. The role of the replacing layer is to classify the unlabeled entities based on the embedded results and handle the entities with untrained (zero-shot) triple tails. In the following, we briefly introduce how to build a proper hierarchical type tree and then present the embedding layer and replacing layer respectively.

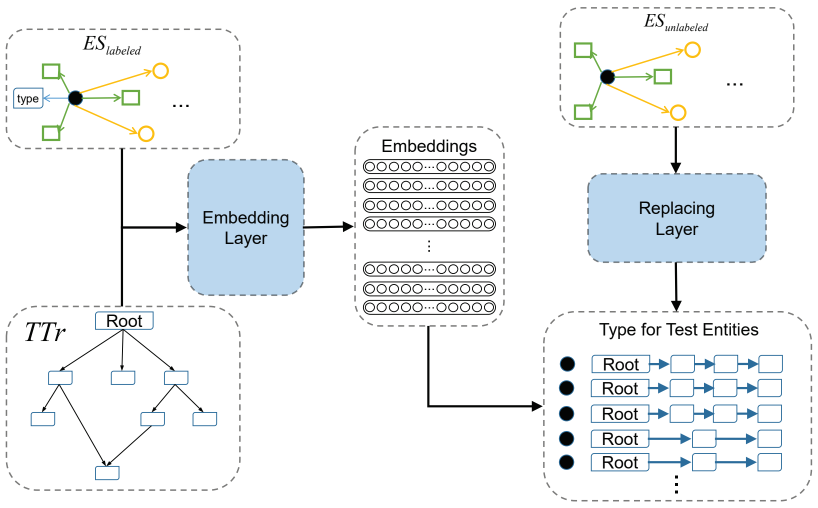

Fig. 2. Architecture of Our Approach

– **Hierarchical Type Tree:** Types are naturally hierarchical, we start with the coarse-grained typing of entities, and then we gradually get more fine-grained types. To this end, we construct a tree that can reflect the hypernym-hyponym relationship of types at the semantic level which can be used to carry out hierarchical classification.
Given a set of types denoted as TS, for each pair of type combination $< t_1, t_2 >$ in TS, we calculate the possibility that t_1, t_2 have hypernym-hyponym relationship according to the entity set of types t_1 and t_2:

$$P_{hyp}(t_1, t_2) = \sqrt{\frac{|ES(t_1) \cap ES(t_2)|}{|ES(t_1)|} \times (1 - \frac{|ES(t_1) \cap ES(t_2)|}{|ES(t_2)|})} \tag{1}$$

where $ES(t)$ is the set of entities of type t. This formula is proposed by Lenci et al. [9], which calculates confidence based on the coincidence between the sets of entities. If $P_{hyp}(t_1, t_2) > \theta$, we consider that t_1 is the hypernym of t_2, where θ is threshold value. The set of hypernym-hyponym relations between types is denoted as TR which is formulated as follows:

$$TR = \{< t_1, t_2 > | t_1, t_2 \in Ts \ \& \ P_{hyp}(t_1, t_2) > \theta\} \tag{2}$$

To build a good hierarchical type tree, we also need to remove some redundant relationships to ensure that the children of each node are at the same layer of the tree. For example, if $<$ "person", "artist" $>, <$ "artist", "painter" $>$ and $<$ "person", "painter" $> \in TR$, we remove $<$ "person", "painter" $>$ from TR so that the children of node "person" in Ttr are at the same layer, which means they are of the similar granularity.

– **Embedding Layer:** The main task of the embedding layer is to construct classifiers and train the vector representation of elements in triples and types.

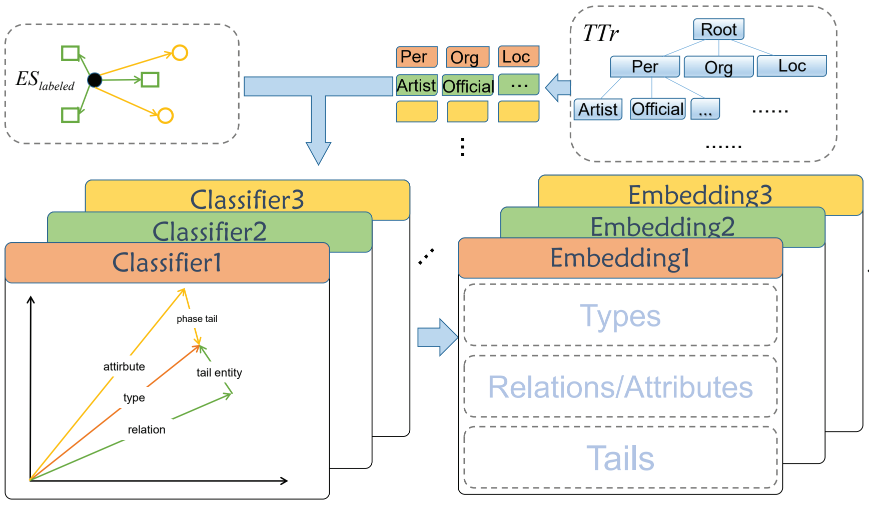

Fig. 3. Architecture of Embedding Layer

To obtain the type path of the entity, we build several classifiers to identify types of different granularities. Also, we propose to fully use discontinuous attribute values, which are trained as bias vector in this layer. We give more details in Sec. 4.1.

– **Replacing Layer:** The replacing layer will be triggered when unlabeled entities are classified by embedding results. This is because we often encounter entities with untrained (zero-shot) triples which have no corresponding vector representation. So we use replacing layer to handle such entities to ensure the robustness of our model. More details could be found in Sec. 4.2.

4.1 Embedding Layer

The architecture of the embedded layer is shown in Fig. 3. There are two main inputs to the embedded layer, one is the set of labeled entities ($ES_{labeled}$), and the other is the hierarchical type tree Ttr. In Ttr, the coarser the type granularity is, the closer it is to the root node. There are three coarse-grained types: "person" ("Per"), "organization" ("Org"), and "location" ("Loc"). We start by training a classifier to classify these three types, and we denote it as $classifier_1$. Then three classifiers are trained to classify the sub-types of "Per", "Org" and "Loc" respectively. For example, if "Per" has five sub-types including artist, officials..., we train a classifier to classify entities which has been classified by $classifier_1$ as "Per" into five more fine-grained types.

In each classifier, we first learn the low-dimensional vector representation of each triple's predicate and tail in embedding layer. After that, the representation of each entity is calculated by the representation of its triples and should be as close to the representation of entity's type as possible. The validity of the method of obtaining types by triples' information has been proved by multiple experiments [8]. But all of these approaches only consider predicate information in triples, and they can only work when the tails of triples are entities which have been linked. However, relation-sparsity entities have a small number of such triple tails.

As shown in Table 1, most relation-sparsity entities also have rich triples that contain many undiscovered relations. These relations are mainly derived from the tails of unsplit and unlinked triples, or from phrase tails that have not yet been refined, and they will be treated as discontinuous attribute values. For example, (“Leonardo da Vinci” , “born” , “14/15 Arilll 1452.Republic of Florence”) can be spilt into an attribute triple (“Leonardo da Vinci” , “birthday” , “14/15 Arilll 1452”) and a relation triple (“Leonardo da Vinci” , “birthplace” , “Republic of Florence”). The goal of our model is to make full use of discontinuous attribute values in fine-grained entity typing.

1) Embedding with Predicates in Triples. In triples, predicates are usually relation names or attribute names. These predicates themselves can reflect the type information of the corresponding head entity. For example, “Leonardo Da Vinci” has an attribute: “bron”, which obviously tends to be the attribute of entities whose type is “Per”. In previous work, SDType [17] proposed a method to calculate the type probability distribution of entities based on their triples' predicates, which indicates that the predicates of entities can indeed reflect the type information of entities. In our approach, we map entities' attribute/relation name and type into the same vector space. Vector representations of entities can be obtained by weighted summation of their attribute/relation vectors and should be as close as possible to the entity's type vector. For an entity in training set, the embedded target can be represented as follows when considering only the average weighting:

$$\frac{\sum_{i=0}^{|PS|} \overrightarrow{p_i}}{|PS|} = \overrightarrow{type} \tag{3}$$

where PS means the set of predicates in entity's triples and $p_i \in PS$, $type$ means entity's type. In the vector space that satisfies the above formula, the more a predicate P is monopolized by a type T, the closer the $\overrightarrow{P}$ is to the $\overrightarrow{T}$. A predicate P being monopolized by a type T means if an entity's triples have predicate P, the entity's type is mostly likely to T. As shown in Fig. 6, predicate “born” exists in triples of entities whose type is “Per” in most cases, so $\overrightarrow{born}$ is close to $\overrightarrow{Per}$ but far from $\overrightarrow{Loc}$ and $\overrightarrow{Org}$.

2) Embedding with Triple Tails. If we just use predicates of triples to train the entity representation, it does work well at coarse granularity. However, as the granularity increases, it becomes difficult to identify the more accuratet type of an entity due to insufficient information. For example, “Leonardo Da Vinci” has a predicate “work”, which can help us know he is a “Per”. However, both entities of type “musician” and entities of type “painter” have predicate “work”. We could hardly know whether “Leonardo Da Vinci” is a “painter” or a “musician”

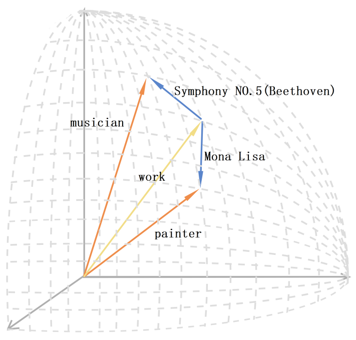

Fig. 4. Example of Bias Vector

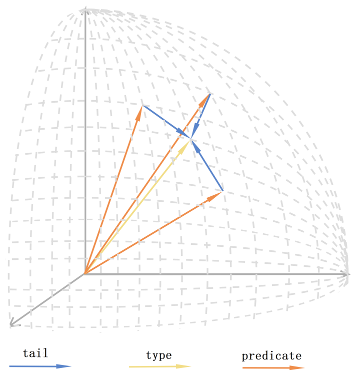

Fig. 5. Training Target

simply by the predicate "work", while the object of predicate "work" can help us identify him as a "painter".

So we consider training the tails of triples as bias vectors pointing to more specific types, as shown in Fig. 4, the vector of predicate "work" is close both to the vector of type "musician" and the vector of type "painter". Apparently, when the object of "work" is a painting like "Mona Lisa," the entity is more likely to be a "painter", and when the object of "work" is a music like "Symphony No. 5 (Beethoven)" the entity is more likely to be a "musician". So we train the tails like "Mona Lisa" and "Symphony No. 5 (Beethoven)" to be bias vectors pointing to more specific types. When considering the tails of triples, we can obtain the type vector of each entity from its predicate vector plus the vector of the corresponding triple tail. The training target and the example are shown in Fig. 5 and Fig. 4 respectively. The formal expression is as follows:

$$\frac{\sum_{i=0}^{|PS|}(\overrightarrow{p_i}+\overrightarrow{t_i})}{|PS|}=type \tag{4}$$

where t_i is the corresponding triple tail of predicate p_i. For continuous attribute values, we train them after discretizing them by clustering.

3) Getting Weights for Predicates. Each predicate is supposed to have a different weight when judging the type. For example, "born" is a common predicate that "Per", "Org", and "Loc" can have, whereas "alias" is usually only reserved for "Per". This is reflected in low-dimensional vector space where $\overrightarrow{born}$ is very close to $\overrightarrow{Per}$ and $\overrightarrow{alias}$ is not particularly close to the vector of any type. Fig. 6 and Fig. 7 illustrate this phenomenon. Clearly "born" is more significant and should have a higher weight. Based on this idea, we use a selective attention mechanism to obtain the weight of the predicate, and the training objective is

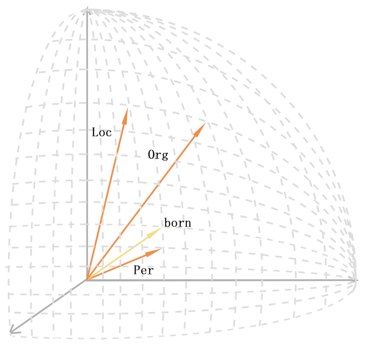

Fig. 6. “born”

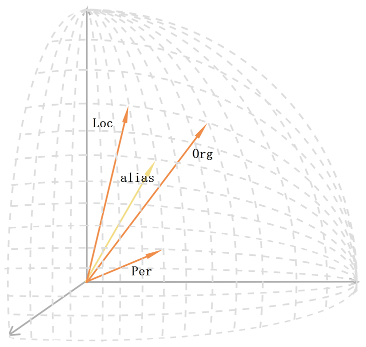

Fig. 7. “alias”

defined as follows:

$$\frac{\sum_{i=0}^{|PS|} e^{Weight(p_i)}(\overrightarrow{p_i} + \overrightarrow{t_i})}{\sum_{i=0}^{|PS|} e^{Weight(p_i)}} = \overrightarrow{type} \tag{5}$$

The weights of predicates, denoted by $Weight(p_i, TS)$, defines the variance of the distances between $\overrightarrow{p_i}$ and the vectors of each class.

$$Weight(p_i, TS) = \frac{\sum_{j=0}^{|TS|} \left(\left\| \overrightarrow{p_i} - \overrightarrow{type_j} \right\|_2 - Avg_{dis}(p_i, TS) \right)^2}{|TS|} \tag{6}$$

where TS means the set of types that the current classifier needs to distinguish and $type_j \in TS$. $Avg_{dis}(p_i, TS)$ is the average distance between $\overrightarrow{p_i}$ and the vector of each type in TS:

$$Avg_{dis}(p_i, TS) = \frac{\sum_{j=0}^{|TS|} \left\| \overrightarrow{p_i} - \overrightarrow{type_j} \right\|_2}{|TS|} \tag{7}$$

During the process of training, negative samples are obtained by replacing the types of positive samples. For example, the negative sample of (“Leonardo da Vinci”,“painter”) can be (“Leonardo da Vinci”,“musician”). The loss function for each positive sample is defined as follows:

$$l_{(e,type)} = \left\| \frac{\sum_{i=0}^{|PS|} e^{Weight(p_i,TS)}(\overrightarrow{p_i} + \overrightarrow{t_i})}{\sum_{i=0}^{|PS|} e^{Weight(p_i,TS)}} - \overrightarrow{type} \right\|_2 \tag{8}$$

For each positive sample $(e, type)$ and the corresponding negative sample $(e, type')$, where $type'$ is the error label after replacing. We use hinge loss to get the loss of each sample. The hinge loss is defined as:

$$l_{hinge} = max(0, l_(e, type) - l_(e, type') + \xi) \tag{9}$$

where ξ is the fixed margin. Finally, our loss function is defined as:

$$L = \sum_{e \in ES} max(0, l_(e, type) - l_(e, type') + \xi) \tag{10}$$

where ES is the entity set in training.

4.2 Replacing Layer

In the real world, there are an infinite number of possible objects corresponding to predicates in triples, and even the triple tails with the same semantic meaning may have different but similar expressions. The training set cannot contain all possible triple tails. In fact, we often encounter untrained tails during testing, so we design a replacing layer to replace such tails with the closest trained tails. We mainly adopt three kinds of similarity: gensim [1] (Bag of Words), longest common sub-sequence (LCS) and BERT-wwm [4]. The gensim similarity is defined as follows:

$$Sim_{gensim}(s_1, s_2) = \frac{|Bow(s_1) \cap Bow(s_2)|^2}{|Bow(s_1)| \times |Bow(s_2)|} \tag{11}$$

where $Bow(s)$ means the "Bag of Words" of s. s_1, s_2 are two phrases used to judge similarity. And the longest common sub-sequence (LCS) similarity is as follows:

$$Sim_{LCS}(s_1, s_2) = \frac{len(LCS(s_1, s_2))^2}{len(s_1) \times len(s_2)} \tag{12}$$

where $LCS(s_1, s_2)$ means the longest common sub-sequence of s_1 and s_2 and $len(\dots)$ means the number of characters in the phrase.

Here we use BERT-wwm [4] to get the vector presentations of phrases and then calculate the similarity by cosine function.

Based on the above three similarities, we define the final similarity as follows:

$$Sim = Sim_{gensim} + \lambda_1 \cdot Sim_{LCS} + \lambda_2 \cdot Sim_{BERT-wwm} \tag{13}$$

λ_1 and λ_2 are weight parameters. The above formula expresses the semantic similarity between phrases, triple tails with similar semantics should have similar representation in the vector space in our model. So we use formula 13 to replace untrained triple tails with tails that are already trained.

5 Experiments and Analysis

In this section, we first introduce our datasets and the metrics we use for evaluation in Sec. 5.1 and then introduce the methods we compare with in Sec. 5.2. Finally, we present the experimental results and analysis in Sec. 5.3.

[1] https://pypi.org/project/gensim/

Table 2. Statistics of the Datasets

	Entities	Types	Predicates	Rel.triples	Attr.triples	
					Continuous	Discontinuous
CN-DBpedia	198,546	175	2185	178,554	515,862	1,270,694
DBpeadia	300,000	214	1426	5,243,230	849,387	0

5.1 Datasets and Experimental Setup

Datasets: We collect our data from CN-DBpedia [2] and DBpedia [3]. The data in CN-DBpedia is mainly collected from the inforbox if BaiduBaike which contains many unsplit and unlinked relation triples. So, a large number of relation-sparsity entities exist in this dataset, which is specifically used to test the performance of our method on relation-sparsity entities. The DBpedia dataset is proposed by Jin et al. [8], which is extracted from DBpedia. In this dataset, each entity has rich relationships and continuous attribute values. As we can see in Table 2, both datasets have about 200 fine-grained types, and the second dataset has a much larger proportion of relational triples than the first one.

Metris: As for the evaluation metrics, we use Micro-averaged F1 (Mi-F1) and Macro-averaged F1 (Ma-F1), which have been used in many fine-grained typing systems [18, 23, 24].

Parameter Settings: In our experiments, all predicate vectors are normalized: $\|\overrightarrow{p_i}\|_2 = 1$, and the length of bias vector is positioned to 1/6 of the length of predicate vector: $\left\|\overrightarrow{t_i}\right\|_2 = 1/6$. And we set fixed margin in loss function $\xi = 1$. In replacing layer, both λ_1 and λ_2 are chosen among{0.2, 0.4, 0.6, 0.8}, we found $\lambda_1 = 0.4$ and $\lambda_2 = 0.8$ achieve best performance.

5.2 Approaches for Comparison

In this section, we briefly introduce five comparative methods including *TransE* [2], *SDType* [17] *APE* [8], our proposed *Baseline Model* and *Final Model.*

- The *TransE* method aims to learn representations of all entities and relations in KGs. In *TransE*, the relation r in each relation triple: (h,r,t) can be considered as a translating from the head entity to the tail entity. By constantly adjusting the vector representation of h, r and t, we wish (h+r) is equal to t as much as possible, that is, h+r=t. This method is mainly used for prediction of tail entities, but vector representation of entities can also be used for entity typing. So, we input the entity representation obtained by *TransE* into a linear classifier for entity typing. For CN-DBpedia, due to the lack of relations, we link some tail entities to ensure that *TransE* could train normally.
- The *SDType* method is a heuristic model which counts on the distribution of head entities' types and tail entities' types for each predicate, respectively.

[2] http://kw.fudan.edu.cn/cndbpedia/intro/
[3] https://wiki.dbpedia.org

Table 3. The Overall of the Comparsion Results

Approaches	CN-DBpedia		DBpeadia	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1
TransE	0.375	0.261	0.519	0.512
SDType	0.558	0.447	0.577	0.569
APE	0.732	0.597	0.657	0.649
Baseline1	0.638	0.493	0.628	0.621
Baseline2	0.723	0.517	0.654	0.646
TransCate	**0.761**	**0.563**	**0.662**	**0.651**

For unlabeled entity, SDType calculates the probability that it is of each type based on these distribution.

- The *APE* method inputs the one-hot encoding of the entity's owned predicates and the vector of the continuous attribute values into a multi-layer perceptron. After that it uses a softmax layer to get the entity's type at the last layer of the network.
- The *Baseline1* considers only the predicates themself without the triple tails information, and each predicate has an average weight when the weighted sum is taken.
- The *Baseline2* does not replace untrained (zero-shot) triple tails by replacing layer but assign vectors of such tails to the average of vectors of trained tails owned by their corresponding predicates.
- The *Final Model* (*TransCate*) adds the triple tails information to the loss function and uses the selective attention mechanism to obtain the predicates' weight. Besides, replacing layer is introduced to handle the untrained triple tails.

5.3 Experimental Results

1) Overall Comparsion Results. We assign 40% of the entities in each dataset to test the performance of the entity typing. Each dataset is subjected to multiple experiments to get the average result except SDType as it relies on probability distribution and the effect of multiple experiments remained unchanged. As can be seen from Table 3, our method has obtained the desired performance, with a 3% improvement over the best methods of the past in CN-DBpedia since CN-DBpedia contains a large number of relation-sparsity entities. For DBpedia, our method only has tiny improvement. It is worth mentioning that TransE's performance is poor on both datasets. We notice that its performance is close to SDType, but much lower than other methods in most cases. The main reason for this is that TransE is designed for tail entity prediction rather than entity typing.

Analysis: In CN-DBpedia, the performance of TransE is much lower than that of other methods due to the sparse relations, and the performance of the baseline1 is lower than that of APE because it only uses predicate information of triples, while APE also use the information of continuous attribute values in triples. When our method uses triple tails information, the results are the

Table 4. The Effectiveness of Tails and Attention

	3 types		23 types		149 types	
	Mi-F1	Ma-F1	Mi-F1	Ma-F1	Mi-F1	Ma-F1
Predicates	0.982	0.961	0.829	0.737	0.638	0.493
P+Tails	0.986	0.970	0.887	0.804	0.744	0.545
P+T+Attentions	0.991	0.983	0.904	0.826	0.761	0.563

best among all the comparison methods. In DBpedia, as the relations become denser, the performance of other methods approaches that of ours. The use of the replacing layer improves performance in CN-DBpedia more than it does in DBpedia because the test set of CN-DBpedia has more untrained triple tails.
2) The Effectiveness of Tails and Attention. We also evaluate the results for different granularity types when our approach use predicates information in triples, tails information and attention mechanisms. Table 4 shows the experimental results on CN-DBpedia. We observe that even simple typing with predicates can achieve good results when judging only three types: person, organization and location. However, the improvement brought by triple tails and attention mechanism becomes larger and larger as the granularity increases.

6 Conclusions and Future Work

In this paper, we propose a new KG embedding model based on attributes and attribute values, which is particularly designed for fine-grained entity typing to relation-sparsity entities in KGs. Based on this KG embedding model, we perform entity typing from coarse-grained level to more fine-grained level hierarchically. We also design an algorithm to handle the entities with untrained (zero-shot) triple tails to ensure the robustness of our model. Our experiments performed on two real-world KGs show that our approach is superior to the most advanced models in most cases.

Future work looks forward to finding a better replacing algorithm for the untrained triple tails. In addition, we would like to consider the description text of the entity as an important information for better entity typing performance to triple-sparsity entities.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: The semantic web, pp. 722–735. Springer (2007)
2. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in neural information processing systems. pp. 2787–2795 (2013)
3. Choi, E., Levy, O., Choi, Y., Zettlemoyer, L.: Ultra-fine entity typing. arXiv preprint arXiv:1807.04905 (2018)
4. Cui, Y., Che, W., Liu, T., Qin, B., Yang, Z., Wang, S., Hu, G.: Pre-training with whole word masking for chinese bert

5. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 601–610. ACM (2014)
6. He, F., Li, Z., Qiang, Y., Liu, A., Liu, G., Zhao, P., Zhao, L., Zhang, M., Chen, Z.: Unsupervised entity alignment using attribute triples and relation triples. In: International Conference on Database Systems for Advanced Applications. pp. 367–382. Springer (2019)
7. Jiang, T., Liu, M., Qin, B., Liu, T.: Attribute acquisition in ontology based on representation learning of hierarchical classes and attributes. arXiv preprint arXiv:1903.03282 (2019)
8. Jin, H., Hou, L., Li, J., Dong, T.: Attributed and predictive entity embedding for fine-grained entity typing in knowledge bases. In: Proceedings of the 27th International Conference on Computational Linguistics. pp. 282–292 (2018)
9. Lenci, A., Benotto, G.: Identifying hypernyms in distributional semantic spaces. In: * SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012). pp. 75–79 (2012)
10. Lin, Y., Liu, Z., Luan, H., Sun, M., Rao, S., Liu, S.: Modeling relation paths for representation learning of knowledge bases. arXiv preprint arXiv:1506.00379 (2015)
11. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: AAAI. vol. 15, pp. 2181–2187 (2015)
12. Ling, X., Singh, S., Weld, D.S.: Design challenges for entity linking. Transactions of the Association for Computational Linguistics 3, 315–328 (2015)
13. Liu, Y., Liu, K., Xu, L., Zhao, J., et al.: Exploring fine-grained entity type constraints for distantly supervised relation extraction (2014)
14. López, F., Heinzerling, B., Strube, M.: Fine-grained entity typing in hyperbolic space
15. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Lingvisticæ Investigationes 30(1), 3–26
16. Neelakantan, A., Chang, M.W.: Inferring missing entity type instances for knowledge base completion: New dataset and methods. arXiv preprint arXiv:1504.06658 (2015)
17. Paulheim, H., Bizer, C.: Type inference on noisy rdf data. In: International semantic web conference. pp. 510–525. Springer (2013)
18. Ren, X., He, W., Qu, M., Huang, L., Ji, H., Han, J.: Afet: Automatic fine-grained entity typing by hierarchical partial-label embedding. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 1369–1378 (2016)
19. Sang, E.F., De Meulder, F.: Introduction to the conll-2003 shared task: Language-independent named entity recognition. arXiv preprint cs/0306050 (2003)
20. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: Proceedings of the 16th international conference on World Wide Web. pp. 697–706. ACM (2007)
21. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: AAAI. vol. 14, pp. 1112–1119 (2014)
22. Xu, B., Zhang, Y., Liang, J., Xiao, Y., Hwang, S.w., Wang, W.: Cross-lingual type inference. In: International Conference on Database Systems for Advanced Applications. pp. 447–462. Springer (2016)

23. Yaghoobzadeh, Y., Schütze, H.: Multi-level representations for fine-grained typing of knowledge base entities
24. Yaghoobzadeh, Y., Tze, H.S.: Corpus-level fine-grained entity typing using contextual information
25. Yahya, M., Berberich, K., Elbassuoni, S., Weikum, G.: Robust question answering over the web of linked data. In: Proceedings of the 22nd ACM international conference on Conference on information & knowledge management. pp. 1107–1116. ACM (2013)
26. Yavuz, S., Gur, I., Su, Y., Srivatsa, M., Yan, X.: Improving semantic parsing via answer type inference. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 149–159 (2016)