Efficient Entity Embedding Construction from Type Knowledge for BERT

Yukun Feng[1]*, **Amir Fayazi**[2], **Abhinav Rastogi**[2], **Manabu Okumura**[1]
[1]Tokyo Institute of Technology
[2]Google Research
{yukun,oku}@lr.pi.titech.ac.jp
{amiraf,abhirast}@google.com

Abstract

Recent work has shown advantages of incorporating knowledge graphs (KGs) into BERT (Devlin et al., 2019) for various NLP tasks. One common way is to feed entity embeddings as an additional input during pre-training. There are two limitations to such a method. First, to train the entity embeddings to include rich information of factual knowledge, it typically requires access to the entire KG. This is challenging for KGs with daily changes (e.g., Wikidata). Second, it requires a large scale pre-training corpus with entity annotations and high computational cost during pre-training. In this work, we efficiently construct entity embeddings only from the type knowledge, that does not require access to the entire KG. Although the entity embeddings contain only local information, they perform very well when combined with context. Furthermore, we show that our entity embeddings, constructed from BERT's input embeddings, can be directly incorporated into the fine-tuning phase without requiring any specialized pre-training. In addition, these entity embeddings can also be constructed on the fly without requiring a large memory footprint to store them. Finally, we propose task-specific models that incorporate our entity embeddings for entity linking, entity typing, and relation classification. Experiments show that our models have comparable or superior performance to existing models while being more resource efficient.

1 Introduction

Many studies have attempted to enhance pre-trained language models with knowledge such as ERNIE (Zhang et al., 2019), KnowBert (Peters et al., 2019), K-ADAPTER (Wang et al., 2020), E-BERT (Poerner et al., 2020), and KEPLER (Wang et al., 2021). Among them, ERNIE, KnowBert, E-BERT, and KEPLER are typical work that do so by incorporating entity embeddings. The entity embeddings are usually trained by methods that model the global graph structure, such as TransE (Bordes et al., 2013a) used in ERNIE and TuckER (Balažević et al., 2019) used in KnowBert. These entity-incorporated pre-trained language models have shown to be powerful on various natural language processing (NLP) tasks, such as entity linking, entity typing, and relation classification.

In this paper, we investigate whether we can construct entity embeddings by considering only local entity features. This is motivated by the observation that the context itself usually provides good information for the right answer. A number of examples are shown in Table 1. Instead of heavily relying on entity embeddings that encode global information, we simply tell the model what these entities are by using local features to help the model infer the answer from the context more easily. For example, if we can know 'Cartí Sugtupu' is a place in the relation classification example in Table 1, the task may be easier. To utilize such information for an entity, we select entity-type knowledge from Wikidata as a local feature for the entity. Specifically, we propose to encode the labels of neighboring nodes of the entity connected through *instance_of* edges in Wikidata. Figure 1 shows an example. These labels can informatively tell the entity type and are usually short, which enables them to be efficiently encoded by simple methods, that we mention later.

One big advantage of utilizing only local features of entities is that we can update our entity embeddings very fast once the knowledge graph (KG) is changed, which is a desirable feature for KGs with rapid updates. We can construct the entity embeddings even on the fly to significantly reduce memory consumption and parameters since a number of tasks (e.g., entity linking) easily involve millions of entities. A disadvantage is that it is hard to infer the answer if large amounts of information are missing. For example, the LAnguage Model Analysis (LAMA) task (Petroni et al., 2019) re-

*Work was extended after the internship with Google.

Findings of the Association for Computational Linguistics: AACL-IJCNLP 2022, pages 1–10
November 20–23, 2022.

quires a [MASK] placeholder in the given sentence "Sullivan was born in Chippewa Falls, Wisconsin in [MASK]" to be filled. The type knowledge may not be able to answer this question. Thus, we do not focus on such tasks. Instead, we apply our method on several typical entity-focused tasks, which were also chosen by related work.

To construct the entity embeddings, we simply average BERT's WordPiece embeddings from the type label of the entity as there are only 2.8 or 2.96 WordPiece tokens on average per label depending on our tasks. Thus, our method is very fast and can be used to construct the entity embeddings on the fly without much cost to save memory and reduce parameters. For example, E-BERT requires six hours to train its entity embeddings, while our method takes only about 1 minute to prepare the entity embeddings for our downstream tasks. The trained entity embeddings of E-BERT take up around 30GB in size[1]. Thus, storing these embeddings requires a large memory footprint, and the size continues to grow linearly if new entities are added. However, our method does not require such extra space for entity embeddings.

For incorporation, previous work incorporates their entity embeddings during both fine-tuning and pre-training (ERNIE and KnowBert). However, pre-training language models is a cumbersome and resource-intensive task. We show simply incorporating our entity embeddings during fine-tuning without any pre-training works well. One reason may be that these entity embeddings are directly constructed through averaging BERT's WordPiece embeddings, so that they look like BERT's WordPiece embeddings, which may be helpful for incorporation for BERT.

Finally, we propose task-specific models to incorporate our entity embeddings[2]. For entity linking, we propose a model that incorporates entity embeddings into the output; for entity typing and relation classification, the proposed model incorporates entity embeddings into the input. We show that our entity embeddings and incorporation method are simple and can achieve comparable or superior performance to existing methods on entity linking, entity typing, and relation classification. The contribution of this work can be summarized as follows:

[1]This size here is from the downloaded embeddings provided by the author.

[2]Our code is available at https://github.com/yukunfeng/efficient_bert_ent_emb

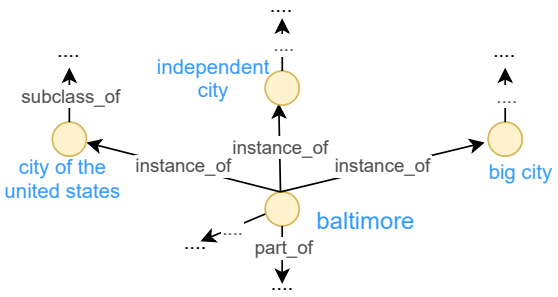

Figure 1: An example of connected entity nodes from Wikidata. The circles are entity nodes with blue texts as their labels. We encode the labels of the neighboring nodes of "baltimore" connected through *instance_of* edges to construct its entity embedding.

- We propose an efficient method to construct entity embeddings that are particularly a good fit for BERT, and they work well without any pre-training step during incorporation.
- Our entity embeddings can be constructed on the fly for BERT. We do not need a large memory footprint to store entity embeddings, which is often required by other work.
- We propose task-specific models to incorporate our entity embeddings for entity linking, entity typing and relation classification.

2 Related Work

ERNIE (Zhang et al., 2019), KnowBert (Peters et al., 2019), E-BERT (Poerner et al., 2020), and our model are all based on Google $\text{BERT}_{\text{BASE}}$ and aim to incorporate entity embeddings into them. The main differences between the models are the methods for constructing entity embeddings and incorporating them.

For entity embeddings, ERNIE uses the one trained on Wikidata by TransE (Bordes et al., 2013b). KnowBert uses TuckER (Balazevic et al., 2019) embeddings, and E-BERT incorporates Wikipedia2Vec entity embeddings (Yamada et al., 2016).These entity embeddings were trained with consideration for a KG structure and have to be trained again if new updates need to be incorporated from KGs, which further requires additional pre-training of ERNIE and KnowBert. When only local features are used to construct the entity embeddings, the aforementioned issues can be avoided. In addition, our entity embeddings are simply obtained by averaging BERT WordPiece embeddings and can be constructed on the fly to save a large memory footprint usually required by

Task	Example	Label
Entity linking	Cricket - **England** beat **Pakistan** by 107 runs in second one-dayer.	England_cricket_team Pakistan_national_cricket_team
Entity typing	GM is a publicly traded company that releases every bit of news **they** have	organization
Relation classification	**Cartí Sugtupu** can be reached by boat from the nearby onshore settlement of Carti and the **Carti Airstrip**.	place_served_by_transport_hub

Table 1: Examples of entity linking, entity typing, and relation classification. The text in bold is the entity of interest. In these examples, we can infer the label from the context.

other work. We found that although our entity embeddings contain only local information, they perform well when combined with context. However, ERNIE, KnowBert or E-BERT are supposed to work better than ours where large amounts of information are missing such as in LAMA task.

For the incorporation, ERNIE and KnowBert both use new encoder layers to feed the entity embeddings, which requires pre-training. In contrast, E-BERT achieves comparable results without pre-training by directly incorporating its entity embeddings into the standard BERT model during task-specific fine-tuning. One proposal from E-BERT is to align the entity and BERT WordPiece embeddings in the same space. To do so, it first trains word and entity embeddings jointly and then learns a linear mapping from word to BERT WordPiece embeddings. The final entity embeddings can be obtained by applying this learned linear mapping so that they look like BERT WordPiece embeddings. This mapping helps improve 4.4 micro F1 score on the test data on entity linking task. To learn this mapping, E-BERT needs to train both word and entity embeddings, which are 30GB in size. Our method for constructing entity embeddings shares the similar spirit, but it is an averaging method from BERT WordPiece embeddings.

K-ADAPTER (Wang et al., 2020) and KEPLER (Wang et al., 2021) are both trained using multi-task learning based on RoBERTa (Liu et al., 2019) in relation classification and knowledge base completion and do not rely on entity embeddings.

Outside the area of incorporating entity embedding into pretrained language model, there are a number of work that propose to use entity types from KGs on various tasks. For example, on entity linking task, some work use entity types together with entity descriptions or entity embedding trained over whole KG (Francis-Landau et al., 2016; Gupta et al., 2017; Gillick et al., 2019; Hou et al., 2020; Tianran et al., 2021). Some work use only entity types on entity linking task (Sun et al., 2015; Le and Titov, 2019; Raiman, 2022). Khosla and Rose (2020) use entity type embeddings for coreference resolution. The main difference between our work with them is that we mainly design our method for constructing entity embedding and our incorporation method for BERT. As introduced before, we simply create entity embeddings from the BERT's internal WordPiece embeddings. When incorporating our entity embeddings into BERT, we also propose a model that makes use of BERT's position embeddings on entity typing and relation classification task (mentioned in Sec. 5.2).

3 Entity Embedding Construction

We take the labels of the neighboring nodes for an entity obtained from Wikidata as local features. Since these labels are usually very short, as shown in Figure 1, we can efficiently obtain label embeddings by averaging WordPiece embeddings in the label. The final entity embeddings are then obtained by averaging the label embeddings. We denote $\mathbf{m}_{ij}$ as the j-th WordPiece embeddings in the i-th label of an entity. The entity embeddings $\mathbf{e}$ are computed as follows:

$$\mathbf{e} = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{N_i}\sum_{j=1}^{N_i}\mathbf{m}_{ij}, \quad (1)$$

where M and N_i are the number of labels and that of WordPiece tokens in the i-th label, respectively. Please note that M and N_i are small in our relation classification task (1.27 and 2.96, respectively, on average). Finally, the generated entity embeddings are updated in the task-specific fine-tuning.

4 Entity Linking

4.1 Task Description

Entity linking (EL) is the task of recognizing named entities and linking them to a knowledge base. In this paper, we focus on an end-to-end EL system that includes detecting the entities and then disambiguating them to the correct entity IDs.

	Train	Dev.	Test
#Tokens	222K	56K	51K
#Gold entities	18454	4778	4778
#Unique generated entities	230K	154K	148K
#Conversion rate	0.8	0.80	0.81

Table 2: Data statistics of AIDA and found unique entities by generator. The conversion rate is the ratio of found entities that we can link to Wikidata.

Following the same setting of E-BERT[3], we use KnowBert's candidate generator to first find all spans that might be potential entities in a sentence. These spans are matched in a precomputed span-entity co-occurrence table (Hoffart et al., 2011) and each span is annotated with linked entity candidate IDs associated with prior probabilities based on frequency. Note that the generator tends to over-generate and most found spans should be rejected according to our observation on the training dataset. Thus, given a span in a sentence, our model needs to learn to reject it or predict the correct one among its candidate IDs in accordance with the context. As with E-BERT, we formulate this task as a classification task where the model needs to classify the given input. The classified labels contain candidate IDs and a rejection label.

4.2 Dataset

We use the AIDA dataset (Hoffart et al., 2011), which was also chosen in related works. The gold named entities in AIDA and spans found by KnowBert's generator are identified with Wikipedia URLs. Due to this reason, we have to convert them to Wikidata IDs to determine the type knowledge of an annotated entity, in which a number are missing during conversion. The statistics of AIDA, found entities by generator, and conversion rates are shown in Table 2.

4.3 Model

Our model is based on $\text{BERT}_{\text{BASE}}$ and the architecture is shown in Figure 2. We describe the incorporation method, modeling, and training hyper-parameters in the following.

4.3.1 Incorporation Method

Given a span from the generator, we denote the embeddings of candidate entities as $\{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_N\}$ and corresponding prior probabilities as $\{p_1, p_2, ..., p_N\}$. The entity embeddings are computed by Eq. 1. Since different candidate entities may have the same type (e.g., the type 'country' may contain different entities), the model cannot distinguish these label embeddings in classification if we simply use the entity embeddings as the label embeddings. Note that this is not an issue when incorporating these entity embeddings into the input, as shown later in our entity typing and relation classification tasks, because the surface forms of entities included in the input can help distinguish between each embedding. Thus, to distinguish these label embeddings, we propose to combine the surface forms of entity candidates, which are still local features, and entity embeddings into label embeddings. The embeddings of surface forms of entities are denoted as $\{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_N\}$. $\mathbf{s}_i$ is simply computed by averaging the WordPiece embeddings in the surface form, which is the same way as computing our entity embeddings. Since large number of entities are involved in this task as shown in Table 2, we compute $\mathbf{s}_i$ and $\mathbf{c}_i$ both on the fly to save memory and reduce the parameters. This means the gradients will come to the WordPiece embeddings during backpropagation. To combine $\mathbf{s}_i$ and $\mathbf{c}_i$, we use a gate to learn to control the weight between $\mathbf{s}_i$ and $\mathbf{c}_i$, and label embedding $\mathbf{l}_i$ is computed as follows:

$$\begin{aligned} g &= \text{sigmoid}(\mathbf{W}\mathbf{c}_i), \\ \mathbf{l}_i &= (1-g) \odot \mathbf{c}_i + g \odot \mathbf{s}_i \end{aligned} \tag{2}$$

$\odot$ is element-wise multiplication and $\mathbf{W} \in \mathbb{R}^{d \times d}$ are trainable parameters where d is a BERT dimension. If $\mathbf{c}_i$ is not found during the aforementioned conversion, we only use $\mathbf{s}_i$.

4.3.2 Modeling

We denote the output vector from the BERT encoder at the position of '[ENT]' as $\mathbf{o}_{\text{ENT}}$. The value of the i-th candidate entity before the softmax function is computed as $\mathbf{l}_i^T \mathbf{o}_{\text{ENT}} + b_i$ where b_i is the bias of the i-th entity candidate. To incorporate the prior probabilities in the classification, we set b_i as $\log p_i$ so that the probability will be p_i if no other information is available (i.e., $\mathbf{l}_i^T \mathbf{o}_{\text{ENT}}$ equals zero). The bias of a rejection label will be learned from the training data. We use the standard cross entropy as our loss function.

4.3.3 Hyper-parameters

Since the dataset is quite small as shown in Table 2, we only train for maximum of four epochs, and the

[3]Our code is based on E-BERT, which is available from https://github.com/NPoe/ebert

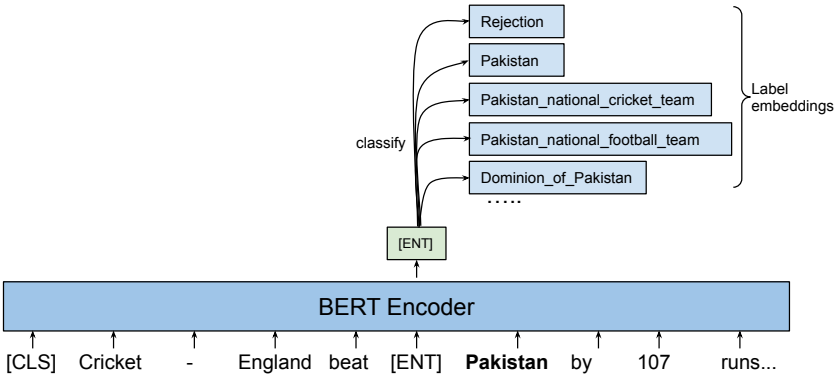

Figure 2: Model architecture for entity linking. The text in bold in the example is the span that the model needs to reject it as a named entity or accept it and link it to the correct entity entry in accordance with the context. A special symbol '[ENT]' is inserted before the span, and the output vector from it will be used for classification.

model with best micro F1 score on the valid dataset is chosen. The batch size is set to 16 and the default AdamW optimizer was used with a linear learning rate scheduler (10% warmup). The learning rate was chosen among {1e-5, 2e-5, 3e-5,} on a valid set.

4.4 Results

The results on the AIDA test set are shown in Table 3. We mainly compare our model with BERT-Random (introduced later), KnowBert and E-BERT as they also focus on incorporating entity embeddings to BERT. Note that we only include end-to-end EL models in this table, and the results are not comparable to ones of disambiguation-only EL models where the golden entity mentions are given.

We used BERT-Random as our baseline, which is the same as our model except that the label embeddings are randomly initialized and trained from scratch. Compared with BERT-Random, our model shows significant improvement, which suggests our proposed label embeddings are effective.

E-BERT incorporates its entity embeddings not only to the output but also to the input. The embedding of its '[ENT]' in the input is computed by averaging all embeddings of candidate entities. We also tried a similar strategy but found no obvious change in our model. Thus, we only focused on the output. In addition, E-BERT uses another strategy that iteratively refines predictions during inference. However, this strategy slows down the inference speed. The results, indicate that the local features work even better than global features used to train entity embeddings in E-BERT. This may suggest that we can utilize local features to construct entity embeddings in tasks where the context already contains a lot of information. Please also note that we can only convert around 80% of Wikipedia URLs to Wikidata IDs, and this may limit the performance of our model. Another advantage is that our label embeddings are constructed on the fly and thus save memory and reduce the number of training parameters. Finally, our model and E-BERT achieved the highest strong micro-F1 and macro-F1 scores among all models, indicating it may be a good way to incorporate knowledge through entity embeddings.

Models	Strong micro-F1	Strong macro-F1
Cao et al. (2021)	83.7	-
Kannan Ravi et al. (2021)	83.1	-
van Hulst et al. (2020)	-	81.3
Broscheit (2019)	79.3	-
Kolitsas et al. (2018)	82.6	82.4
Hoffart et al. (2011)	71.9	72.8
E-BERT (Poerner et al., 2020)	85.0	84.2
KnowBert (Peters et al., 2019)	73.7	-
Our model	**86.3**	**84.4**
BERT-Random	73.3	76.8

Table 3: Results on AIDA test set. BERT-Random use randomly initialized label embeddings trained from scratch.

5 Entity Typing and Relation Classification

5.1 Task Description

The goal of entity typing is to predict the types of a given entity from its context. Note that it is not necessary that the mention of a given entity is a named entity. For example, the type 'they' is labeled as 'organization' as shown in the example of entity

typing in Table 1. The formulation of relation classification is similar with the only difference being that there are two target entities in the sentence. We need to predict the relation of two given target entities together with the context. Thus, the application of our entity embeddings is similar for entity typing and relation classification. We introduce our incorporation method in the following section.

5.2 Incorporation Method

Unlike the EL task where we applied our entity embeddings to the output, we only incorporate entity embeddings to the input for these two tasks. To incorporate the entity embeddings, we propose a method that emphasizes target entities (e.g., in relation classification, there are two entity mentions). Specifically, for all entities, we first sum the embeddings of the entities and the corresponding BERT WordPiece tokens, and then feed them into the BERT model. For target entities, we explicitly insert the entity embeddings into the input of WordPiece token embeddings and make the entity embeddings share the same position embeddings with their corresponding WordPiece token embeddings, as if they are in the same position. Our model architecture is shown in Figure 3. We mathematically describe our method as follows.

We denote the number of WordPiece tokens in a sentence as T, and the i-th WordPiece token embedding, entity embedding, and position embedding as $\mathbf{w}_i$, $\mathbf{e}_i$, and $\mathbf{p}_i$, respectively. As shown in the figure, the entity embedding $\mathbf{e}_i$ is $\mathbf{0}$ if the i-th token is not the start token of an entity. For simplicity, we ignore token type embeddings here, although they are actually used in our model. We first obtain the input $\mathbf{x_i}$ to the BERT encoder by summing the entity embeddings with the other embeddings:

$$\mathbf{x_i} = \mathbf{e}_i + \mathbf{w}_i + \mathbf{p}_i. \quad (3)$$

Since target entities are usually more important than other entities in an entity-centric task, we explicitly insert target entity embeddings that have the same position embeddings as their aligned WordPiece embeddings, as if they are in the same position. For the relation classification task, there are two target entities, and thus the extra inserted inputs are $\mathbf{x_{T+1}}$ and $\mathbf{x_{T+2}}$, which are computed as follows:

$$\begin{aligned} \mathbf{x_{T+1}} &= \mathbf{e}_{k_1} + \mathbf{p}_{k_1}, \\ \mathbf{x_{T+2}} &= \mathbf{e}_{k_2} + \mathbf{p}_{k_2}, \end{aligned} \quad (4)$$

where k_1 and k_2 are the index of the first and second target entities, respectively.

5.3 Experiments

5.3.1 Entity Typing

We chose Open Entity (Choi et al., 2018) to evaluate our model. The dataset has several versions, and we chose the one that has nine general types (e.g., person, location, and object), which is the same as that in previous works. One example from this dataset is shown in Table 1. As previously mentioned, the entity mention in Open Entity is not limited to named entities, and pronoun mentions and common noun expressions are also included. We used a preprocessed version from ERNIE (Zhang et al., 2019). This preprocessed dataset was annotated with mentions of named entities and automatically linked to Wikidata by TAGME (Ferragina and Scaiella, 2010) so that we could find their type knowledge in Wikidata for all entities in Open Entity. We used the same annotated entities as the ones used in ERNIE by keeping the same confidence threshold to filter unreliable entity annotations. The statistics of this dataset are shown in Table 4. Most annotated entities are non-target because the entity mention in Open Entity is not limited to named entities. Our model needs to utilize the context together with the entity annotations to infer the types of the target entity. We can also see the type labels of entities are quite short (only 2.8 word pieces per label), and this may be one reason that our averaging method for constructing entity embeddings works. If the label is long (e.g., becoming a text description), the averaging method might be too simple to encode it. Since the involved entities are not that many, we did not construct the entity embeddings on the fly to speed up the training. That is, the entity embeddings are initialized by Eq. 1 and are updated in the training.

	Train	Dev.	Test
#Instances	2,000	2,000	2,000
#Target entities	122	107	94
#All entities	2573	2511	2603
#Labels per entity	1.56	1.56	1.63
#WordPieces per label	2.8	2.8	2.8

Table 4: Statistics of Open Entity dataset with nine label types. TAGME (Ferragina and Scaiella, 2010) is used to automatically annotate named entities in the dataset.

Our code was adapted from ERNIE,[4] and we

[4] https://github.com/thunlp/ERNIE

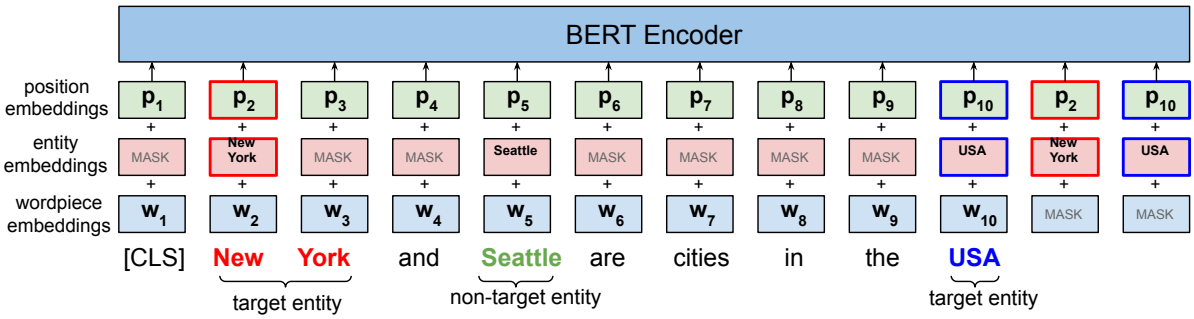

Figure 3: Overall architecture showing a sequence input to the BERT encoder for the relation classification task. The entity embeddings are obtained by encoding the labels of their neighboring nodes, as described in Sec. 3. Note that the entity and position embeddings for the two target entities are copied over to the end of the sequence.

used the same setup as it. For each instance, we used a special symbol to mark the span of a target entity and used the [CLS] vector in the last hidden layer from the BERT encoder for classification. For the hyper-parameters, we basically followed those of ERNIE. The learning rate was set to 2e-5 with the AdamW optimizer and a linear learning rate scheduler (10% warmup). The model was trained for 10 epochs with a batch size of 16. The results are shown in Table 5. Among the models in the BASE size, our model is comparable to or more effective than the related methods. Compared with KnowBert and ERNIE, the construction of our entity embeddings is more efficient and our model does not require pre-training. Further analysis of our model will be in the ablation study.

5.3.2 Relation Classification

We used a preprocessed relation classification dataset from ERNIE (Zhang et al., 2019) to evaluate our model. This dataset is from the FewRel corpus (Han et al., 2018) and was rearranged by Zhang et al. (2019) for the common relation classification setting. One example from this dataset is shown in Table 1. We used FewRel oracle entity IDs, which were also used in ERNIE and E-BERT (Poerner et al., 2020). These oracle entity IDs cover only target entities; there are no annotations for non-target entities. Our model needs to predict the relation of two given target entities with their annotations and context. The statistics of the FewRel dataset are shown in Table 6. Since oracle annotations were used, the statistics of annotated target entities are not shown in the table. Again, we can see the type labels are quite short, which enables them to be encoded with a simple averaging method. Since there are not many entities involved, we take these entity embeddings as parameters and do not construct them on the fly.

As with the entity typing task, special tokens [HD] and [TL] were used to mark the span of a head and tail entity, respectively. The [CLS] vector in the last hidden layer of the BERT encoder was used for relation classification. For the hyper-parameters, we basically followed those of ERNIE. The model is trained for 10 epochs with a batch size of 16. The default AdamW optimizer was used with a linear learning rate scheduler (10% warmup). The learning rate was set to 4e-5, which was chosen among {2e-5, 3e-5, 4e-5, 5e-5} on the valid dataset.

The results are shown in Table 7. ERNIE, E-BERT, and our model can be directly compared with because all the models are based on $\text{BERT}_{\text{BASE}}$ and used the same entity annotations. Our model achieves better results than ERNIE and E-BERT, indicating that our methods are effective while being cost-efficient. However, E-BERT reports that their entity coverage is about 90% (around 10% of entity embeddings are not found in their Wikipedia2Vec embeddings), while the entity coverage in our model and ERNIE is about 96%. This may put E-BERT at a disadvantage.

5.4 Ablation Study

To analyze the gain, we define three components in our model for entity typing and relation classification: *entityEmbs*, defined by Eq. 1, *sum*, defined by Eq. 3, and *insert*, defined by Eq. 4. When *entityEmbs* is not used, the entity embeddings are initialized randomly. The results for cases when independently excluding each component are shown in Table 8. When *entityEmbs* was removed, the performance of our model on two datasets dropped significantly, which indicates our method for con-

	Model	Architecture	P	R	F1
Incorporate KG in pre-training	ERNIE (Zhang et al., 2019)	$\text{BERT}_{\text{BASE}}$	78.42	72.90	75.56
	KnowBERT (Peters et al., 2019)	$\text{BERT}_{\text{BASE}}$	78.60	73.70	76.10
	K-ADAPTER (Wang et al., 2020)	$\text{RoBERTa}_{\text{LARGE}}$	79.30	75.84	77.53
	KEPLER (Wang et al., 2021)	$\text{RoBERTa}_{\text{BASE}}$	77.80	74.60	76.20
Fine-tuning only	$\text{BERT}_{\text{BASE}}$ (our reproduction)	$\text{BERT}_{\text{BASE}}$	79.78	70.90	75.08
	Our model	$\text{BERT}_{\text{BASE}}$	78.53	74.16	76.28

Table 5: Results of our model and related models on the entity typing dataset - Open Entity. Note that only K-ADAPTER is in the LARGE size, and ERNIE, KnowBERT, and K-ADAPTER also require incorporating knowledge during fine-tuning.

	Train	Dev.	Test
#Instances	8,000	16,000	16,000
#Labels per entity	1.27	1.25	1.25
#WordPieces per label	2.96	3.0	3.02

Table 6: Relation classification dataset FewRel with 80 relation types.

Model	P	R	F1
ERNIE (Zhang et al., 2019)	88.49	88.44	88.32
E-BERT (Poerner et al., 2020)	88.51	88.46	88.38
$\text{BERT}_{\text{BASE}}$ (our reproduction)	86.16	86.16	86.16
Our model	**88.93**	**88.93**	**88.93**

Table 7: Relation classification results on FewRel. Only ERNIE incorporates entity embeddings in both pre-training and fine-tuning steps.

Model	Open Entity	FewRel
Our model	76.28	88.93
w/o *entityEmbs*	74.03	84.98
w/o *sum*	75.83	88.81
w/o *insert*	75.62	87.99

Table 8: Ablation study with F1 scores. Each component in our model is excluded independently.

Model		P	R	F-1
Our model		78.53	74.16	76.28
w/o *sum*	*insert*	78.33	73.48	75.83
	insert for all	78.73	72.32	75.39
w/o *insert*	*sum* for only non-target	79.12	72.43	75.62

Table 9: Ablation study on Open Entity dataset.

structing entity embeddings is effective while maintaining cost-efficiency. Once *entityEmbs* was used, we can see that *sum* shows improvement on the two datasets. The performance can be further improved if *insert* was used together with *sum*, which suggests that *sum* does not make full use of the information for target entities, and emphasizing target entities explicitly by *insert* is effective.

To analyze how *insert* and *sum* separately work on target and non-target entities, we conducted another ablation study on Open Entity, and the results are shown in Table 9. Since there are no non-target entity annotations in FewRel, only Open Entity is included. If *insert* was applied for all entities, the performance degraded, which suggests that emphasizing non-target entities is not helpful, and it is more effective to incorporate entity embeddings for target and non-target entities in a different way. When *sum* was applied only to non-target entities without *insert*, its performance was better than that of $\text{BERT}_{\text{BASE}}$, indicating that incorporating the embeddings of non-target entities is useful.

6 Conclusion

In this paper, we proposed to construct entity embeddings using local features instead of training those with consideration of the whole KG for tasks where the context already contains large amounts of information. Utilizing local features to construct the entity embeddings is much faster than the methods mentioned in related work. The local features of an entity used in this paper are the labels of its neighboring nodes connected through *instance_of* edges in Wikidata. Since these labels are usually very short, we can simply encode them by averaging their WordPiece embeddings. The simple averaging method enables us to even construct entity embeddings on the fly without much cost. This is helpful for saving memory and reducing parameters in tasks where minions of entities may be involved. Finally, we proposed task-specific models to incorporate our entity embeddings. Unlike most previous works, our entity embeddings can be directly incorporated during fine-tuning without requiring any specialized pre-training. Our experiments on entity linking, entity typing, and relation classification show that our entity embeddings and incorporation method are simple and effective, and the proposed models have comparable or superior performance to existing models while having the aforementioned advantages.

References

Ivana Balažević, Carl Allen, and Timothy M Hospedales. 2019. Tucker: Tensor factorization for knowledge graph completion. In *Empirical Methods in Natural Language Processing*.

Ivana Balazevic, Carl Allen, and Timothy Hospedales. 2019. TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5185–5194, Hong Kong, China. Association for Computational Linguistics.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013a. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

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

Samuel Broscheit. 2019. Investigating entity knowledge in BERT with simple neural end-to-end entity linking. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 677–685, Hong Kong, China. Association for Computational Linguistics.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. 2021. Autoregressive entity retrieval. In *International Conference on Learning Representations*.

Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-fine entity typing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 87–96, Melbourne, Australia. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Paolo Ferragina and Ugo Scaiella. 2010. Tagme: on-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1625–1628.

Matthew Francis-Landau, Greg Durrett, and Dan Klein. 2016. Capturing semantic similarity for entity linking with convolutional neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1256–1261, San Diego, California. Association for Computational Linguistics.

Daniel Gillick, Sayali Kulkarni, Larry Lansing, Alessandro Presta, Jason Baldridge, Eugene Ie, and Diego Garcia-Olano. 2019. Learning dense representations for entity retrieval. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 528–537, Hong Kong, China. Association for Computational Linguistics.

Nitish Gupta, Sameer Singh, and Dan Roth. 2017. Entity linking via joint encoding of types, descriptions, and context. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2681–2690, Copenhagen, Denmark. Association for Computational Linguistics.

Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2018. FewRel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4803–4809, Brussels, Belgium. Association for Computational Linguistics.

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. 2011. Robust disambiguation of named entities in text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 782–792, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Feng Hou, Ruili Wang, Jun He, and Yi Zhou. 2020. Improving entity linking through semantic reinforced entity embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6843–6848, Online. Association for Computational Linguistics.

Manoj Prabhakar Kannan Ravi, Kuldeep Singh, Isaiah Onando Mulang, Saeedeh Shekarpour, Johannes Hoffart, and Jens Lehmann. 2021. Cholan: A modular approach for neural entity linking on wikipedia and wikidata. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*.

Sopan Khosla and Carolyn Rose. 2020. Using type information to improve entity coreference resolution. In *Proceedings of the First Workshop on Computational Approaches to Discourse*, pages 20–31, Online. Association for Computational Linguistics.

Nikolaos Kolitsas, Octavian-Eugen Ganea, and Thomas Hofmann. 2018. End-to-end neural entity

linking. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 519–529, Brussels, Belgium. Association for Computational Linguistics.

Phong Le and Ivan Titov. 2019. Distant learning for entity linking with automatic noise detection. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4081–4090, Florence, Italy. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Matthew E. Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54, Hong Kong, China. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Nina Poerner, Ulli Waltinger, and Hinrich Schütze. 2020. E-BERT: Efficient-yet-effective entity embeddings for BERT. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 803–818, Online. Association for Computational Linguistics.

Jonathan Raiman. 2022. Deeptype 2: Superhuman entity linking, all you need is type interactions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):8028–8035.

Yaming Sun, Lei Lin, Duyu Tang, Nan Yang, Zhenzhou Ji, and Xiaolong Wang. 2015. Modeling mention, context and entity with neural networks for entity disambiguation. In *Twenty-fourth international joint conference on artificial intelligence*.

Li Tianran, Yang Erguang, Zhang Yujie, Chen Yufeng, and Xu Jinan. 2021. Improving entity linking by encoding type information into entity embeddings. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pages 1087–1095, Huhhot, China. Chinese Information Processing Society of China.

Johannes M. van Hulst, Faegheh Hasibi, Koen Dercksen, Krisztian Balog, and Arjen P. de Vries. 2020. Rel: An entity linker standing on the shoulders of giants. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Cuihong Cao, Daxin Jiang, Ming Zhou, et al. 2020. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv preprint arXiv:2002.01808*.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhiyuan Liu, Juanzi Li, and Jian Tang. 2021. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*.

Ikuya Yamada, Hiroyuki Shindo, Hideaki Takeda, and Yoshiyasu Takefuji. 2016. Joint learning of the embedding of words and entities for named entity disambiguation. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 250–259, Berlin, Germany. Association for Computational Linguistics.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451, Florence, Italy. Association for Computational Linguistics.