Entity Embedding Completion for Wide-Coverage Entity Disambiguation

Daisuke Oba†‡∗ Ikuya Yamada‡⋄ Naoki Yoshinaga§ Masashi Toyoda§
† The University of Tokyo ‡ Studio Ousia ‣ RIKEN AIP
§ Institute of Industrial Science, The University of Tokyo
oba@tkl.iis.u-tokyo.ac.jp ikuya@ousia.jp {ynaga,toyoda}@iis.u-tokyo.ac.jp

Abstract

Entity disambiguation (ED) is typically solved by learning to classify a given mention into one of the entities in the model’s entity vocabulary by referring to their embeddings. However, this approach cannot address mentions of entities that are not covered by the entity vocabulary. Aiming to enhance the applicability of ED models, we propose a method of extending a state-of-the-art ED model by dynamically computing embeddings of out-of-vocabulary entities. Specifically, our method computes embeddings from entity descriptions and mention contexts. Experiments with standard benchmark datasets show that the extended model performs comparable to or better than existing models whose entity embeddings are trained for all candidate entities as well as embedding-free models. We release our source code and model checkpoints at https://github.com/studio-ousia/steel.

1 Introduction

Entity disambiguation (ED) (§ 2) has been studied as a method to organize information about real-world entities in text. Consequently, ED must be capable of handling diverse entities in various domains. Existing state-of-the-art ED models are based on huge pretrained language models (PLMs) (Devlin et al., 2019), and are trained to match a given mention with its candidate entities via embeddings (Broscheit, 2019; Ling et al., 2020; Févry et al., 2020; Yamada et al., 2022). Despite the advances in performance, these embedding-based ED models do not address out-of-vocabulary entities that are absent from the training data. Even worse, since maintaining millions of entity embeddings demands a huge memory space in training, the models are often trained only for candidate entities (§ 2) that appear in testing (Broscheit, 2019; Yamada et al., 2022).1 To address the above problems, researchers have explored ED models without entity embeddings. Cao et al. (2021) solves ED as autoregressive generation, which decodes the entity name token-by-token. Barba et al. (2022) formulates ED as span extraction, which selects a correct entity name from input concatenating mention contexts and candidate entity names. Although these approaches allow a model to handle millions of candidate entities, they still have difficulties in handling newly-emerged entities that are absent from the training data and evolved entities whose attributes have changed over time. Meanwhile, Logeswaran et al. (2019) and Wu et al. (2020) proposed zero-shot methods to handle any candidate entities by reading their descriptions at testing. The performance of these zero-shot methods, however, is inferior to the aforementioned models, probably because these methods represent each entity using short entity-related text instead of massive training examples used in the ED models.

In this paper, we propose an adaptation method that enables entity embedding-based ED models, which are trained for limited entities, to handle the other rare and emerging entities by dynamically predicting their embeddings. In particular, we actually extend the architecture of the state-of-the-art Transformer-based ED model developed in Yamada et al. (2022). We first train this model with an entity vocabulary consisting of only common entities (§ 4.1). Next, we extend the model (§ 4.3) by using our BERT-based encoders that dynamically predict entity embeddings from descriptions and relevant sentences (§ 4.2), thus enabling the model to handle the out-of-vocabulary entities.

1 Reducing the entity vocabulary size is critical to train huge Transformer-based ED models (§ 3), as the entity embeddings can occupy GPU memory during training. For instance, seven million 256-dimensional embeddings for all entities in the English Wikipedia consume approximately 22GB of GPU memory in the training standard with the Adam optimizer.
Through experiments (§ 5) with standard benchmark datasets for ED (i.e., AIDA-CoNLL (Hoffart et al., 2011), MSNBC, AQUAINT, ACE2004, WNED-CWEB, and WNED-WIKI (Guo and Barbosa, 2018)), we verify that our extended model performs competitively to the original state-of-the-art model, whose trained entity embeddings encompass all the candidate entities of the evaluation datasets (§ 5). We then perform an empirical analysis on the nature of our proposed encoders, and on reducing memory costs in the inference time (§ 6).

Our key contributions are:

- Adapting trained entity embedding-based models to any sets of candidate entities, by predicting embeddings of missing entities with neural encoders (§ 4).
- Showing that our method enables a model trained with a limited entity vocabulary to perform comparably with models trained with all candidate entities of the evaluation data (§ 5).
- Presenting an empirical analysis of our predicted embeddings, and the possibility for reducing memory costs during inference (§ 6).

2 Entity Disambiguation Task

Given a knowledge base (KB) such as Wikipedia, and a text with spans of mentions, the ED model assigns each mention span to an entity in the KB. The task is similar to word sense disambiguation.

Since the number of entities can be more than millions, during the evaluation, the ED models additionally assume a small list of candidate entities for each mention, using a mention-entity dictionary derived from hyperlinks in Wikipedia. Following existing embedding-based ED models (e.g., Yamada et al. (2022)), we use the standard entity candidate set, KB-YAGO (Ganea and Hofmann, 2017).

3 Related Work

Several studies have proposed knowledge-intensive Transformer-based ED models that have entity embeddings as parameters, such as weights of the classifier (Broscheit, 2019; Ling et al., 2020; Févry et al., 2020; Yamada et al., 2022). Despite the state-of-the-art in the benchmark, they cannot handle out-of-vocabulary entities and entities of changed properties, since these embeddings are typically trained via ED task with labeled corpora. In addition, as argued by Barba et al. (2022), it is hard to integrate embeddings of all ever-increasing entities into the huge Transformer-based models, especially to maintain their gradient values during training. Ling et al. (2020) therefore demands expensive TPU clusters, and some employ transductive settings to train embeddings for a small subset of entities containing all candidate entities used in the evaluation (Broscheit, 2019; Yamada et al., 2022).

Before Transformer becomes the dominant architecture for the ED task, knowledge-intensive ED models were developed (Yamada et al., 2016; Ganea and Hofmann, 2017; Le and Titov, 2018). Since these models require relatively few parameters to be learned (e.g., several linear transformation matrices (Ganea and Hofmann, 2017)), loading and training embeddings of entire entities may not be a problem compared to the Transformer-based models. However, they still do not address entities absent in or updated from the training phase.

Zero-shot systems, which represent entities as descriptions or contexts rather than embeddings, can convey textual information pertaining to novel or changing entities. However, a single entity description, used by Logeswaran et al. (2019) and Wu et al. (2020), has less information compared to embeddings learned from many examples. FitzGerald et al. (2021) embeds each mention context as keys to retrieve associated entities from the target mention contexts (query). However, they also suffer from less informative contexts per key embedding. Generation- or extraction-based approaches can reduce computational costs by not integrating entity-level knowledge such as entity embeddings; Cao et al. (2021) generates tokens of entity titles and Barba et al. (2022) selects the correct title from the concatenated titles of candidate entities. However, while they can generate or extract any entities, they cannot properly model novel and evolving entities since they are trained with labeled data based on a specific KB. Moreover, the performance still falls short of entity embedding-based models.

Along with the fact that Transformer-based models performed well on the other entity-related tasks, such as question answering, by using entity embeddings (Zhang et al., 2019; Yamada et al., 2020; Févry et al., 2020; de Jong et al., 2021), we can infer the importance of explicitly modeling knowledge as the parameters to optimize the task performance. This study focuses on this powerful entity embedding-based models.

\(^2\)Lower Recall@1 than that of Wu et al. (2020)
4 STEEL

In this section, we present our adaptation method, STEEL, which complements STatic and Encoder-based entity embeddings for Entity Linking. While our method is applicable to a variety of entity embedding-based methods, in this paper, we apply it to the architecture of the state-of-the-art ED model developed in Yamada et al. (2022).

We begin by training the base ED model, whose entity embeddings comprise only a reasonable number of common entities $V_c$ (§ 4.1). We then estimate the embeddings for other entities $\notin V_c$ using our proposed encoders, and extend the model (§ 4.2). Hereafter, $V_w$ and $V_e$ denote the word and entity vocabularies of the base model. $H$ and $D$ denote the size of the hidden states and of the embeddings.

4.1 Base ED Model

Input Representations Our base ED model is adopted from Yamada et al. (2022), an extension of BERT (Devlin et al., 2019). This model takes a concatenation of word token sequence and entity token sequence as an input, and entity tokens of target mentions are masked (Figure 1). Each entity token corresponds to an entity in the word token sequence (e.g., dark_side stands for word tokens of the, dark, and side). Each token is represented by summing the following three embeddings:

- **Token Embeddings** correspond to each word $w_i \in V_w$ and entity $e_i \in V_e$. We denote the word token embeddings as $A \in \mathbb{R}^{|V_w| \times H}$. We denote entity token embeddings as $B \in \mathbb{R}^{|V_e| \times D}$ and $U \in \mathbb{R}^{D \times H}$, to reduce computational cost where $D \ll H$, following Yamada et al. (2020).

- **Position Embeddings** correspond to the positions of tokens. We represent words and entities at position $i$ in the word sequence by $D_i \in \mathbb{R}^H$ and $E_i \in \mathbb{R}^H$. For entities spanning multiple word tokens (e.g., dark_side), we average their position embeddings.

Training We first build and train an ED model that includes entity embeddings $B \in \mathbb{R}^{|V_e| \times D}$ for only the top-K frequent entities $V_e$ in Wikipedia articles as trainable parameters. The model performs entity disambiguation in the manner of masked language modeling (Devlin et al., 2019).

Using hyperlinks in Wikipedia articles as entity annotations, we mask each mention token with a certain probability, and optimize the model to estimate the entity $e \in V_e$ from the BERT’s output $h_e \in \mathbb{R}^H$ corresponding to [MASK]:

$$m_e = \text{layernorm}(\text{gelu}(W_f h_e + b_f)) \quad (1)$$

$$\hat{y} = \text{softmax}(B m_e) \quad (2)$$

where $W_f \in \mathbb{R}^{D \times H}$ and $b_f \in \mathbb{R}^D$ are trainable parameters, gelu (Hendrycks and Gimpel, 2016)
is the activation function, and layernorm (Lei Ba et al., 2016) is the layer normalizer. Compared to the model of Yamada et al. (2022), we eliminate a bias term in Eq. 2, as STEEL extends only the entity embedding $B$. We maximize the log-likelihood of the estimation $\hat{y}$.

### 4.2 Our Encoders for Embedding Prediction

We propose two BERT-based encoders (Figure 1) to predict the embeddings of out-of-vocabulary entities $e_i \notin V_e$. Each encoder follows the same architecture as our base ED model which is an extension of BERT, while removing the linear transformation head to estimate entities in Eq. 1 and 2. Encoders use entity descriptions and mention contexts (sentences referring to the entity) as inputs respectively, which are language resources easily obtained from a KB. Each of them corresponds to connotative and denotative definitions of entities.

**Description Encoder:** The encoder predicts entity embeddings from descriptions. We first convert each description into an input representation in the same way as § 4.1 (i.e., word tokens and entity tokens). We then obtain the BERT output $h_{CLS} \in \mathbb{R}^H$ corresponding to [CLS], and compute the embedding of the entity $e_i$ as follows:

$$\hat{e}_{DESC} = W_{DESC} h_{CLS} + b_{DESC}$$

where $W_{DESC} \in \mathbb{R}^{D \times H}$ and $b_{DESC} \in \mathbb{R}^D$ are the trainable parameters.

**Hyperlink Encoder:** The encoder predicts embedding for entity $e_i$ from the sentences with hyperlinks of $e_i$, which we hereafter refer to as mention contexts. We first obtain mention contexts referring to $e_i$ from KB, and convert them to input representations in the same way as § 4.1 (i.e., word token sequences and entity token sequences). We then replace the entity token corresponding to $e_i$ with [MASK]. By using $N$ mention contexts for $e_i$, we obtain $N$ outputs from the BERT corresponding to each [MASK] (i.e., $h_{MASK1} \ldots h_{MASKN}$), and we average them as the predicted embeddings for $e_i$ in order to embed multiple contexts:

$$\hat{h}_{MASK} = \frac{1}{N} \sum_{n=1}^{N} h_{MASK_n}$$

$$\hat{e}_{HL} = W_{HL} \hat{h}_{MASK} + b_{HL}$$

where $W_{HL} \in \mathbb{R}^{D \times H}$ and $b_{HL} \in \mathbb{R}^D$ are trainable parameters.

**Encoder Training:** We have trained the above two encoders as follows. We first initialize parameters of the two encoders using the trained parameters of the base ED model (§ 4.1), since these encoders adopt a similar architecture and input representations as ED model. As the supervision to train the encoders, we utilize embeddings $B$ for the common entities $V_e$, which are integrated in and jointly trained with our base model (§ 4.1). During training the encoders, we freeze $B$. With this approach, the predicted embeddings are placed in the same space as that of the base ED model. We minimize the mean squared error between the trained embedding $B_i$ for the entity $e_i \in V_e$, and the corresponding predicted embedding. To reduce training costs, we optimize each encoder separately.

### 4.3 Entity Embedding Completion

We predict embeddings for the out-of-vocabulary entities $V_e^{oov} \notin V_e$ using our encoders, and complement the entity embedding matrix $B$.

**Embedding Prediction and Ensemble:** We first predict the embeddings of out-of-vocabulary entities $V_e^{oov}$ using each of our neural encoders. We then simply average the embeddings predicted by our encoders (Eq. 3 and 5), to leverage different language resources:

$$\hat{e}_i = \frac{1}{2}(\hat{e}_{DESC}_i + \hat{e}_{HL}_i)$$

**Completion:** We extend the trained entity embedding matrix $B$ using the predicted embeddings of entities $V_e^{oov}$, as illustrated in Figure 1. Accordingly, we rearrange Eq. 2 as follows:

$$\hat{y} = \text{softmax}(B \circ m_e)$$

where $B^o \in \mathbb{R}^{|V_e \cup V_e^{oov}| \times D}$ denotes extended entity embedding matrix. Thus, the trained model can address out-of-vocabulary entities. In this paper, we focus on adding new embeddings for out-of-vocabulary entities. However, it is also capable of replacing the trained embeddings in $B$ of entities whose meaning has changed due to updates of the associated KB such as Wikipedia.

**Inference:** The model performs inferences sequentially in several steps. We first mask all entity tokens. The model estimates the entities in each step (Eq. 7), and then replaces the [MASK] of the highest probability with the estimated entity token. We repeat this process until all [MASK] tokens have been disambiguated.
4.4 Implementation Details

**Base ED Model:** To ensure computational efficiency and improve the reproducibility on a common computational environment, we implement an ED model based on the BERT\_BASE (Devlin et al., 2019), whereas Yamada et al. (2022) exploited BERT\_LARGE. As the training corpus, we use the December 20th, 2018 version of English Wikipedia articles consisting of over three billion words and ten million entity annotations. We set the entity vocabulary size of our base ED model to $|V_e| = 500K$ consisting of the most frequent entities in the articles. We split the articles into the sets of up to 512 words, and extract up to 128 hyperlinks for each set. We then tokenize them using BERT’s tokenizer with the vocabulary $V_e$ consisting of 30K tokens. We initialize all parameters common with BERT using the pre-trained parameters, and set other parameters randomly. We train the model in two steps; we first train the model with the BERT’s parameters fixed for 10 epochs, and we then update entire parameters for 10 epochs. Additional details are described in Appendix A. We designate this model as YAMADA\_BASE, which will be extended by STEEL.

**Encoders for Embedding Prediction:** We initialize the parameters by using those of our trained base model, YAMADA\_BASE. As the training corpus, we exploit the descriptions and mention contexts corresponding to $V_e$ from the same Wikipedia articles. As the entity description, we extract the first 512 tokens and up to 128 hyperlinks contained in them, from the corresponding entity’s page. As the mention context, we extract the sentences with hyperlinks pointing to the corresponding entity. We here use OpenNLP sentence tokenizer\(^3\) to detect sentences. As an extracted mention context, we include both or either of the before and after sentences altogether for exploiting longer contexts per mention. We exploit up to eight mention contexts per entity (i.e., $N = 8$ in Eq. 4). Further details are described in Appendix B.

5 Experiments

We conduct experiments on entity disambiguation with the goal of verifying that the performance of YAMADA\_BASE with STEEL is comparable to that of state-of-the-art models, whose entity vocabulary contains all candidate entities in the evaluation as well as to that of embedding-free models.

5.1 Settings

**Evaluation Datasets and Metric:** We follow the settings specified in previous studies (Ganea and Hofmann, 2017; Le and Titov, 2018; Yamada et al., 2022). Specifically, we use benchmark datasets; AIDA-CoNLL (Hoffart et al., 2011), MSNBC, AQUAINT, ACE2004, WNED-CWEB, and WNED-WIKI (Guo and Barbosa, 2018). For entity candidate sets, we use KB+YAGO (Ganea and Hofmann, 2017), and select top-30 candidates per mention based on the associated prior probability. All entity candidates used in these evaluation data consist of 140K entities (hereafter, $V_e^{all}$), $52.7\%$ of candidate entities $V_e^{all}$ are not included in the entity vocabulary $V_e$ of our base ED model. Moreover, $13.3\%$ of gold entities are not included in $V_e$, which will never be solved without STEEL’s extension. We report In-KB accuracy for AIDA-CoNLL and Micro-F1 for the others.

**Fine-tuning for AIDA-CoNLL:** In the evaluation with AIDA-CoNLL, we conduct fine-tuning using the attached training set (AIDA-train) as the baselines did, while freezing the extended entity embedding matrix $B^e$. We prepare inputs in the same way as in the training step (§ 4.1), and mask the entity tokens with a probability of 90% following Yamada et al. (2022). We optimize the model by maximizing the log-likelihood of the prediction in Eq. 7. The best batch size, learning rate, and epochs are searched from the same space as Devlin et al. (2019) based on the development set (AIDA-testa). Further details are described in Appendix C.

**Baselines:** We employ embedding-based models (Ganea and Hofmann, 2017; Le and Titov, 2018; Broscheit, 2019; Févry et al., 2020; Yamada et al., 2022), whose entity vocabulary covers all the candidate entities $V_e^{all}$ in their training phase. In addition, we train a full-coverage version of our base ED model whose entity vocabulary $V_e$ is equal to $V_e^{all}$, which we refer to as YAMADA\_BASE. This full-coverage model can be regarded as the smaller version of Yamada et al. (2022) without bias terms in Eq. 3. As the baselines trained without entity embeddings, we employ the generation-based (Cao et al., 2021) and extraction-based (Barba et al., 2022) models.

5.2 Results

Table 1 shows the results for AIDA-CoNLL. First, by applying STEEL to YAMADA\_BASE, we achieve comparable performance against the full-coverage

---

\(^3\)https://opennlp.apache.org
Table 1: In-KB accuracy on AIDA-CoNLL. Models are fine-tuned unless otherwise stated. The figures of systems marked with $\ast$ are quoted from existing studies. Entity candidate sets used in the system marked with $\dagger$ are limited to those included in the 500k entities $V_e$.

<table>
<thead>
<tr>
<th>Systems</th>
<th>CoNLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/ entity embeddings:</td>
<td></td>
</tr>
<tr>
<td>Ganea and Hofmann (2017)$\ast$</td>
<td>92.2</td>
</tr>
<tr>
<td>Le and Titov (2018)$\ast$</td>
<td>93.1</td>
</tr>
<tr>
<td>Broscheit (2019)$\ast$</td>
<td>87.9</td>
</tr>
<tr>
<td>Févry et al. (2020)$\ast$</td>
<td>92.5</td>
</tr>
<tr>
<td>Yamada et al. (2022) $\ast_{\text{LARGE}}$</td>
<td>95.0</td>
</tr>
<tr>
<td>YAMADA$\text{BASE}$</td>
<td>94.7</td>
</tr>
<tr>
<td>w/o entity embeddings:</td>
<td></td>
</tr>
<tr>
<td>Cao et al. (2021)$\ast$ (generation-based)</td>
<td>93.3</td>
</tr>
<tr>
<td>Barba et al. (2022)$\ast$ (extraction-based)</td>
<td>92.6</td>
</tr>
<tr>
<td>YAMADA$\text{500K}_\text{BASE} + \text{STEEL}$</td>
<td>94.3</td>
</tr>
<tr>
<td>YAMADA$\text{BASE} + \text{STEEL}$ w/o fine-tuning</td>
<td>91.8</td>
</tr>
<tr>
<td>YAMADA$\text{500K}_\text{BASE} \dagger$</td>
<td>90.8</td>
</tr>
</tbody>
</table>

Table 3 shows the accuracy on AIDA-CoNLL compared to the reported score\(^4\) of the state-of-the-art zero-shot system (Wu et al., 2020), which addresses ED by comparing the mention contexts and candidate entities’ descriptions without using entity embeddings. Note that it is not a fair comparison as their candidate sets are not identical to KB+YAGO, since their model uses their own candidate generator (i.e., bi-encoder). However, the gold recall of their candidate set 97% is comparable to KB+YAGO’s 98%. Even taking into account the difference in entity candidates, our model yielded higher performance with more than 10% margin. This is probably because we solve ED by using entity embeddings predicted from descriptions or mention contexts only for $V_e^{\text{oov}}$ and by using high-quality entity embeddings optimized via ED task with massive examples for the other frequent entities. Entity embedding-based models advanced the state-of-the-arts in the past studies, but such models are generally not good at handling rare and emerging entities as discussed in § 1 and § 3. Our approach solves these critical weaknesses while inheriting the strengths of the conventional entity embedding-based models.

Overall, the above observations indicate that STEEL obtains entity embeddings of comparable quality to that of state-of-the-art system (Yamada et al., 2022), and successfully integrates them to the model to perform ED on out-of-vocabulary entities.

### 6 Analysis

In this section, we provide a detailed analysis of our proposed adaptation method, STEEL.

#### 6.1 Ablation Study

We investigate the impact of ensembling two types of predicted entity embeddings, as derived from the description and hyperlink encoders (§ 4.2). By

\(^4\)https://github.com/facebookresearch/BLINK
Table 2: Micro-F1 on the standard evaluation datasets, with the exception of AIDA-CoNLL. The figures of systems marked with ∗ are quoted from existing studies. Entity vocabulary sets used by the system marked with ♠ are limited to those included in 500K frequent entities $V_e$. avg. indicates the averaged score over all datasets.

Table 3: Accuracy on the AIDA-CoNLL. The figures of systems marked with ∗ are quoted from their official website.

Table 4: Averaged Micro-F1 on all datasets when only one of the proposed encoders is ablated.

6.2 Entity Disambiguation Performance for Mentions of Out-of-Vocabulary Entities

We investigate how well our extended model disambiguates the mentions of out-of-vocabulary entities $V^{\text{oov}}_e$, whose embeddings are predicted and integrated into the model by STEEL. We report precision regarding those mentions over the six datasets. We also report the performance of the simple baseline, which selects the entity with the highest prior probability $\hat{p}(\text{entity}|\text{mention})$ (Ganea and Hofmann, 2017).

Table 5 ($V^{\text{oov}}_e$ column) shows the results. Our model correctly handles approximately 80% mentions of the out-of-vocabulary entities, whereas the prior probability baseline, which exploits surface information, solves less than 70% of the mentions. Moreover, the performance gap with oracle model YAMADA$^{\text{BASE}}$, which optimizes the embeddings of all candidate entities via ED task, is only 1.2 pt. As for the comparison to the state-of-the-art zero-shot system, Wu et al. (2020), we achieved huge performance improvements. Again, we here state that Wu et al. (2020) use candidate sets not identical to KB+YAGO as described in § 5.2.

The performance experienced a more significant drop (-2.2 pt.) when ablating the hyperlink encoder. It indicates the worth of mention contexts to embed out-of-vocabulary entities. This trend differs from that of the ablation results of overall mentions in Table 4. It indicates that the utilization of only mention contexts has a side effect of inhibiting the inference of mentions for in-vocabulary entities $V_e$.
6.3 Effects of the Number of Hyperlink Sentences Fed into Hyperlink Encoder

In this section, we analyzed the impact of the number of mention contexts to be fed into hyperlink encoder on the quality of predicted embeddings. In the settings of our study, hyperlink encoder uses up to eight mention contexts per entity. Therefore, the quality of predicted embeddings can be partially determined by the number of mention contexts used. Here, we extract the subsets of mentions of out-of-vocabulary entities, who have \( \leq 2 \) and \( \geq 7 \) mention contexts available for the hyperlink encoder, and compute the precision for each mention subset.

\( \# \) contexts column in Table 5 shows the results on each mention subset. For mentions of relatively frequent entities (\( \geq 7 \)), the use of predicted embeddings from hyperlink encoder leads to more correct inferences. Conversely, for mentions of rare entities (\( \leq 2 \)), the description encoder performs more effectively. This likely because entity descriptions corresponding to connotative meanings are a frequency-independent resource. Future research directions include the selection of different types of predicted embeddings based on the volume of available resources, and weighted averaging of two types of embeddings.

Table 5: \( V_e^{\text{oov}} \) column indicates precision for the mentions of the out-of-vocabulary entities. \# contexts column shows precision for the subset of those mentions, which are distinguished by the number of mention contexts (max. 8) used to encode the corresponding gold entity.

<table>
<thead>
<tr>
<th>Method</th>
<th>( V_e^{\text{oov}} # ) contexts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \leq 2 )</td>
</tr>
<tr>
<td>YAMADA\textsubscript{BASE} + STEEL</td>
<td>79.0</td>
</tr>
<tr>
<td>w/o Description Encoder</td>
<td>79.7</td>
</tr>
<tr>
<td>w/o Hyperlink Encoder</td>
<td>76.8</td>
</tr>
</tbody>
</table>

Table 6: Precision on all datasets for the mentions of vocabulary entities, who have frequencies of embeddings based on the volume of available resources, and weighted averaging of two types of embeddings.

<table>
<thead>
<tr>
<th>Method</th>
<th>( V_e^{\text{oov}, \leq 8} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>YAMADA\textsubscript{BASE} + STEEL</td>
<td>76.3</td>
</tr>
<tr>
<td>YAMADA\textsubscript{BASE}</td>
<td>75.3</td>
</tr>
</tbody>
</table>

Table 6 shows that if only the same number of mention contexts are available, prediction by STEEL yields better quality entity embeddings than learning them through the ED task. It suggests that it is not always practical to apply optimization of entity embeddings via ED task when their resources are scarce, whereas STEEL makes relatively effective use of limited resources. In practice, STEEL can utilize more mention contexts than the trained base model as KB updates after the model training. The gain therefore will become larger.

6.4 On Whether Embeddings of Rare Entities Should be Learned or Predicted

We measured the performance on the mention subset of out-of-vocabulary entities that have eight or fewer mentions on Wikipedia articles (say, \( V_e^{\text{oov}, \leq 8} \)). In our experimental settings (§ 5.1), up to eight mentions per entity are extracted and used for predicting embeddings for out-of-vocabulary entities, even if there are more than eight mentions available on Wikipedia. By focusing on \( V_e^{\text{oov}, \leq 8} \), the number of mention contexts used to optimize embeddings by YAMADA\textsubscript{BASE} via ED task and to predict embeddings by STEEL are both the same.

Table 6 shows that if only the same number of mention contexts are available, prediction by STEEL yields better quality entity embeddings than learning them through the ED task. It suggests that it is not always practical to apply optimization of entity embeddings via ED task when their resources are scarce, whereas STEEL makes relatively effective use of limited resources. In practice, STEEL can utilize more mention contexts than the trained base model as KB updates after the model training. The gain therefore will become larger.

6.5 Qualitative Analysis of Predicted Entity Embeddings

Using STEEL, we predict each embedding of entire entities in English Wikipedia articles except the most frequent 500K entities in \( e \in V_e \), and perform k-nearest neighbors-search among the trained and predicted embeddings, separately.

Table 7 shows that each predicted embedding is close to embeddings of relevant entities. For instance, the embedding of a station in Denver, Perry_station, is located near embeddings of other stations in Denver. An high-end-audio company, Pass_Labs, is close to a sound-related term, Passband, and a semiconductor maker, Intersil. Among the infrequent entities, it is close to more directly related audio device-makers such as Valve_Amplification_Company.

6.6 Quantization of Entity Embeddings

In this section, we quantized entity embeddings in the extended model (YAMADA\textsubscript{BASE} + STEEL) to reduce the memory cost at inference time. This experiment was motivated by the argument in Barba et al. (2022) that embedding-based ED models are flawed because entire entity embeddings need to be stored in memory at runtime. Our method, which is
Table 7: Nearest neighbors of entity embeddings predicted by our encoder, among entities in \( V_e \) whose embeddings are contained in our base model, as well as other entities whose embeddings are dynamically predicted.

<table>
<thead>
<tr>
<th>Entity Title</th>
<th>Among the most frequent 500K entities</th>
<th>Among the other entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perry_station</td>
<td>RTD_bus_and_rail_services, Denver_Union_Station, U.S._Route_287_in_Colorado</td>
<td>Colorado_station, Knox_station, Evans_station_(RTD), 25th_%26_Welton_station</td>
</tr>
<tr>
<td>Word_embedding</td>
<td>Word-sense_disambiguation, WordNet, Automatic_summarization</td>
<td>Sentence_embedding, Word-sense_induction, Font_embedding</td>
</tr>
<tr>
<td>Pass_Labs</td>
<td>Passband, Intersil, Keysight, TEAC_Corporation, Patch_panel</td>
<td>Valve_Amplification_Company, Nelson_Pass, Class-T_amplifier</td>
</tr>
</tbody>
</table>

Table 8: Micro-F1 of our extended model (w/o fine-tuning) when quantizing its entity embeddings, as well as the required memory to store embeddings for entire English Wikipedia titles (~7 million) during inference.

<table>
<thead>
<tr>
<th>Format</th>
<th>Size</th>
<th>CoNLL</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original: FP32</td>
<td>7.73GB</td>
<td>91.8</td>
<td>89.8</td>
</tr>
<tr>
<td>INT8</td>
<td>1.93GB</td>
<td>91.9</td>
<td>89.8</td>
</tr>
<tr>
<td>INT6</td>
<td>1.45GB</td>
<td>91.8</td>
<td>89.9</td>
</tr>
<tr>
<td>INT4</td>
<td>0.97GB</td>
<td>91.7</td>
<td>89.6</td>
</tr>
</tbody>
</table>

7 Conclusions

In this study, we proposed an adaptation method, STEEL, which makes trained entity embedding-based ED models applicable to candidate entities that are not integrated into the model (§ 4). Our methods extend trained models by predicting entity embeddings of other rare or emerging entities by encoding the corresponding descriptions or relevant sentences. In the experiments with standard benchmark datasets (§ 5), our model performed competitive or better than existing entity embedding-based models trained for all candidate entities used in the evaluation, and also outperformed embedding-free models. We also conducted analyses concerning the predicted embeddings (§ 6.1 to 6.5), and evaluated the capability of reducing memory costs required during inference via quantization (§ 6.6).

Our methods, in addition to out-of-vocabulary entities, can also handle evolving entities whose properties have changed after training the model, even though they were included in the entity vocabulary set during training. This capability is vital in ED since a KB to be linked can be updated intermittently. For example, Wikipedia articles before and after “Joe Biden” have become U.S. president are very different. His embedding trained using the former periods of articles may not be optimal for processing mentions in the newer articles. The performance of replacing trained embeddings with new ones should be evaluated in the future.

As a base model to which our method is applied, we adopted Yamada et al. (2022) since it is the state-of-the-art and its source code is publicly available. We release our implementation at https://github.com/studio-ousia/steel for the other models to enjoy our method in the future.
Limitations

The proposed method works for any rare or emerging entities only when relevant descriptions or sentences are available, as is the case with most knowledge bases such as Wikipedia, Fandom\(^6\), and UMLS (Bodenreider, 2004). If these resources are not available, for example, we can use entity-entity relations collected by knowledge graphs alternatively. We may also utilize the standard entity titles (e.g., Manchester_Airport) as descriptions or as search queries to retrieve possible mention contexts from the web.

The proposed method can handle entities in languages other than English; however, when handling several multilingual entities simultaneously, it is unknown whether it can successfully absorb differences in word order and volume of resources across languages.

Acknowledgement

We thank the anonymous reviewers for their attentive reading of our manuscript and author response, and for their valuable comments and suggestions. This work was partially supported by JSPS KAKENHI Grant Number 21H03494, and by JST CREST Grant Number JPMJCR19A4 including AIP challenge program, Japan.

References


\(^6\)https://www.fandom.com


Ikuya Yamada, Koki Washio, Hiroyuki Shindo, and Yuji Matsumoto. 2022. Global entity disambiguation with BERT. In Proceedings of the 2022 Conference of the North American Chapter of the Association for

<table>
<thead>
<tr>
<th>number of hidden layers</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden size $H$</td>
<td>768</td>
</tr>
<tr>
<td>entity embedding size $D$</td>
<td>256</td>
</tr>
<tr>
<td>mask probability of entity tokens</td>
<td>0.3</td>
</tr>
<tr>
<td>mask probability of word tokens</td>
<td>0</td>
</tr>
<tr>
<td>attention heads</td>
<td>12</td>
</tr>
<tr>
<td>attention head size</td>
<td>64</td>
</tr>
<tr>
<td>activation function</td>
<td>gelu</td>
</tr>
<tr>
<td>maximum word length</td>
<td>512</td>
</tr>
<tr>
<td>batch size</td>
<td>2048</td>
</tr>
<tr>
<td>total epochs</td>
<td>20</td>
</tr>
<tr>
<td>learning rate (1st 10 epochs)</td>
<td>5e-4</td>
</tr>
<tr>
<td>learning rate decay (1st 10 epochs)</td>
<td>linear</td>
</tr>
<tr>
<td>warmup steps (1st 10 epochs)</td>
<td>2500</td>
</tr>
<tr>
<td>learning rate</td>
<td>5e-5</td>
</tr>
<tr>
<td>learning rate decay</td>
<td>linear</td>
</tr>
<tr>
<td>warmup steps</td>
<td>2500</td>
</tr>
<tr>
<td>dropout</td>
<td>0.1</td>
</tr>
<tr>
<td>weight decay</td>
<td>0.01</td>
</tr>
<tr>
<td>gradient clipping</td>
<td>1000</td>
</tr>
<tr>
<td>adam $\beta_1$</td>
<td>0.9</td>
</tr>
<tr>
<td>adam $\beta_2$</td>
<td>0.999</td>
</tr>
<tr>
<td>adam $\epsilon$</td>
<td>1e-6</td>
</tr>
</tbody>
</table>

Table 9: Hyper-parameters used for training our base ED model.


**Appendix**

**A Details of Training the Base ED model**

We implement our base ED model based with PyTorch (v.1.10.1) (Paszke et al., 2019) and Hugging Face’s Transformers (v4.15.0) (Wolf et al., 2019). We exploit the base-uncased version of BERT.³ We optimize the model using AdamW optimizer. Training takes about 10 days using eight Tesla V100 GPUs. Table 9 describes our hyper-parameters.

³https://huggingface.co/docs/transformers/v4.15.0/en/model_doc/bert#overview
number of hidden layers | 12
hidden size $H$ | 768
entity embedding size $D$ | 256
attention heads | 12
attention head size | 64
activation function | gelu
maximum word length | 512
batch size | 256
learning rate | 1e-4
learning rate decay | linear
warmup steps (description-based) | 500
warmup steps (hyperlink-based) | 1000
dropout | 0.1
weight decay | 0.01
gradient clipping | 1000
adam $\beta_1$ | 0.9
adam $\beta_2$ | 0.999
adam $\epsilon$ | 1e-6

Table 10: Hyper-parameters used for training our encoders.

### B Details of Training the Encoders

We implement the encoders with PyTorch (v.1.10.1) and Hugging Face’s Transformers (v4.15.0). We exploit the same version of BERT as in Appendix A. Training of each of the description- and hyperlink-based encoders takes about 7 hours and 17 hours, using a single Quadro RTX 5000 GPU. We optimize the model using AdamW optimizer. We follow most of the hyper-parameters adopted in the base ED model, with the exception of the learning rate. By actually extending the base model with the trained encoder, and by checking its performance on the development set of AIDA-CoNLL (AIDA-testa), we search the best learning rate from $\{1e-4, 2e-5, 5e-5\}$ for each encoder separately. We describe our hyper-parameters in Table 10.

### C Details of Fine-tuning for the evaluation on AIDA-CoNLL dataset

Following the settings of training our base ED model (Appendix A), we split the datasets into sequences consisting of up to 512 word tokens and of up to 128 entity tokens corresponding hyperlinks in the word tokens. We conduct fine-tuning with PyTorch and Trainer provided by Hugging Face. We use a single Quadro RTX 5000 GPU. We search hyper-parameters from the same space as that of Devlin et al. (2019) based on the development set.

Table 11 describes our hyper-parameters used in conducting fine-tuning on the CoNLL dataset.

(AIDA-testa):
- batch size: $\{16, 32\}$
- learning rate: $\{2e-5, 3e-5, 5e-5\}$
- epochs: $\{2, 3, 4\}$

Table 11: Hyper-parameters used for fine-tuning our extended model on AIDA-CoNLL.