Subword-based Compact Reconstruction of Word Embeddings

Shota Sasaki[1,2], **Jun Suzuki**[2,1], **Kentaro Inui**[2,1]
[1]RIKEN AIP, [2]Tohoku University
{sasaki.shota, jun.suzuki, inui}@ecei.tohoku.ac.jp

Abstract

The idea of *subword-based* word embeddings has been proposed in the literature, mainly for solving the out-of-vocabulary (OOV) word problem observed in standard *word-based* word embeddings. In this paper, we propose a method of reconstructing pre-trained word embeddings using subword information that can effectively represent a large number of subword embeddings in a considerably small fixed space. The key techniques of our method are twofold: memory-shared embeddings and a variant of the key-value-query self-attention mechanism. Our experiments show that our reconstructed subword-based embeddings can successfully imitate well-trained word embeddings in a small fixed space while preventing quality degradation across several linguistic benchmark datasets, and can simultaneously predict effective embeddings of OOV words. We also demonstrate the effectiveness of our reconstruction method when we apply them to downstream tasks[1].

1 Introduction

Pre-trained word embeddings (or embedding vectors), especially those trained on a vast amount of text data, such as the Common Crawl (CC) corpus[2], are now considered as highly beneficial, fundamental language resources. Typical examples of large, well-trained word embeddings are those trained on the CC corpus with 600 billion tokens by fastText (Bojanowski et al., 2017) and with 840 billion tokens by GloVe (Pennington et al., 2014), which we refer to as `fastText.600B`[3] and `GloVe.840B`[4], respectively. In fact, we often leverage such word embeddings to further improve the task performance of many natural language processing (NLP) tasks, such as constituency parsing (Suzuki et al., 2018; Gómez-Rodríguez and Vilares, 2018), discourse parsing (Yu et al., 2018), semantic parsing (Groschwitz et al., 2018; Dong and Lapata, 2018), and semantic role labeling (Strubell et al., 2018).

Despite their significant impact on the NLP community, well-trained word embeddings still have several disadvantages. In this paper, we focus on two issues surrounding well-trained word embeddings: i) the massive memory requirement and ii) the inapplicability of out-of-vocabulary (OOV) words. It is crucial to address such issues, especially when applying them to real-world open systems. The total number of embeddings (i.e., the total memory requirement of such word embeddings) often becomes unacceptably large, especially in limited-memory environments, including GPUs, since the vocabulary size is more than 2 million words, which require at least 2 gigabytes (GB) of memory for storage.

One possible solution is to merely discard (less important) words from the vocabulary, which can straightforwardly reduce the memory requirement. However, such a naive method can cause another well-known drawback regarding the inapplicability of OOV words. The applicability of OOV words is highly desirable in real systems since input words can be uncontrollably diverse. Therefore, there is a trade-off between the number of embedding vectors and the applicability of OOV words; thus, our goal is to investigate and develop a method that simultaneously has less memory requirement and high applicability of OOV words, which are both desirable properties for word embeddings in real-world open systems.

Recently, methods that leverage subword information have been proposed and have become popular for overcoming the OOV word issue. Con-

[1]Our code and reconstructed subword-based word embeddings trained from `GloVe.840B` and `fastText.600B` are available: https://github.com/losyer/compact_reconstruction

[2]http://commoncrawl.org

[3]https://fasttext.cc/docs/en/english-vectors.html

[4]https://nlp.stanford.edu/projects/glove/

ceptually, the subword-based approach can cover all the words that can be constructed by a combination of subwords. Thus, the subword-based approach can greatly mitigate (or solve) the OOV word issue. We extend this approach to simultaneously enabling a reduction in the total number of embedding vectors through the reconstruction of word embeddings by subwords. The key techniques of our approach are twofold: memory-shared embeddings and a variant of the key-value-query (KVQ) self-attention mechanism (Vaswani et al., 2017). That is, our approach reconstructs well-trained word embeddings using a limited number of embedding vectors that are shared by all the subwords with an effective weighting calculated by the self-attention mechanism.

In our experiments, we show that our reconstructed subword-based embeddings can successfully imitate well-trained word embeddings, such as `fastText.600B` and `GloVe.840B`, in a small fixed space while preventing quality degradation across several linguistic benchmark datasets from word similarity and analogy tasks. We also demonstrate the effectiveness of our reconstructed embeddings for representing the embeddings of OOV words. Lastly, we confirm the performance of our reconstructed embeddings on several downstream tasks from the named entity recognition task and the textual entailment task.

2 Related Work

The OOV word issue is one of the widely discussed topics in word embedding research, which several researches have recently attempted to solve. For example, methods that leverage subword information, such as character N-grams (including character unigrams) (Bojanowski et al., 2017; Pinter et al., 2017; Zhao et al., 2018) and morphological features (Luong et al., 2013), have recently been discussed as means of constructing word embeddings that consider the applicability of OOV words. Moreover, Pilehvar and Collier (2017) have proposed a method called `SemLand`, which induces OOV word embeddings by leveraging external resources. Bahdanau et al. (2017) and Herbelot and Baroni (2017) have also proposed methods that estimate OOV word embeddings using an additional LSTM and leveraging a small additional dataset, respectively.

Among them, the study most closely related to ours is that of Zhao et al. (2018). Their basic idea is to reconstruct each pre-trained word embedding using a bag-of-character N-grams. We refer to their method as '`BoS`'. The motivation for reconstructing pre-trained word embeddings and utilizing character N-grams in our approach is substantially the same, however, an essential difference from `BoS` is that we additionally consider jointly reducing the total number of embedding vectors.

Another study that shares the same motivation and goal is that of Pinter et al. (2017). Their method, referred to as `MIMICK`, utilizes only character information instead of character N-grams by mixing it with more sophisticated neural networks, i.e., LSTM (Hochreiter and Schmidhuber, 1997). `MIMICK` can produce a more compact model than the original word embeddings. The important difference between their method and ours is that our method only consists of the subword embeddings, whereas their method consists of the character embeddings and several transformation matrices for calculating LSTMs. We compare their method with ours in our experiments and empirically show the effectiveness of our approach.

Moreover, Bojanowski et al. (2017) have proposed a method called `fastText`, which also incorporates character N-gram embeddings in addition to word embeddings. However, they did not explicitly prove the effectiveness of OOV word embeddings. Thus, it is still unclear how well the combination of character N-grams can reconstruct appropriate embeddings for OOV words. In addition, their method trains word embeddings from a corpus, which is not a reconstruction setting we discuss in this paper. Therefore, their method is orthogonal to ours.

We often aim to reduce memory consumption of word embeddings in the real-world since they require relatively large memory. Suzuki and Nagata (2016) proposed a parameter reduction method for word embeddings by using machine learning techniques. Our method can also be interpreted as a kind of parameter reduction method based on the subword features. However, their method only considers the model shrinkage, and does not utilize any subword information nor consider the OOV issue.

To summarize, none of the previous studies have attempted to simultaneously achieve a smaller number of embedding vectors and higher applicability of OOV words. Thus, in this paper, we report the first attempt to investigate how we can simultaneously achieve them.

Additionally, deep contextualized pre-trained language models, such as ELMo (Peters et al., 2018), have recently been proposed as alternatives to the pre-trained word embeddings to further improve task performances. However, ELMo still takes advantage of `Glove.840B` to achieve its state-of-the-art performance. This fact implies that we can still combine the pre-trained word embeddings with strong pre-trained language models; thus, the importance of word embeddings in the literature remains unchanged even though stronger pre-trained models have been established.

3 Reconstruction of Word Embeddings Using Subwords

In this section, we explain a formal task definition that we tackle in this paper.

3.1 Preliminaries

Notation rules: In this paper, we use the following notation rules unless otherwise specified. First, a lower-case bold letter, e.g., $\boldsymbol{v}$ and $\boldsymbol{e}$, represents a column vector, and an upper-case bold letter, e.g., $\boldsymbol{V}$ and $\boldsymbol{E}$, represents a matrix. Then, $\|\boldsymbol{v}\|_p$ represents L_p-norm of the given vector $\boldsymbol{v}$. Next, let a lower-case letter, e.g., z or i, be a scalar variable or index, and an upper-case letter, e.g., C or H, indicate a scalar but hyper-parameter during the training. Here, we introduce the notation $\boldsymbol{V}[i]$ to represent the i-th column vector in the matrix $\boldsymbol{V}$ to simplify the representation. Moreover, an upper-case letter in a calligraphy form, e.g., $\mathcal{W}$ and $\mathcal{S}$, denotes a set, and the absolute value of a set, such as $|\mathcal{W}|$ and $|\mathcal{S}|$, indicates the number of instances in the corresponding set. Finally, the Greek letter, such as Φ and η, indicates a function.

Words and their embedding: Let $\mathcal{W}$ be a vocabulary, i.e., a set of words. Let $\zeta(\cdot)$ represent a mapping function from a word to the corresponding index of the word, namely,

$$\zeta(\cdot) : \mathcal{W} \to \mathcal{I}_w \;\text{ where }\; \mathcal{I}_w = \{1, \dots, |\mathcal{W}|\}. \quad (1)$$

In this paper, we always assume that $\zeta(\cdot)$ is a bijective function; thus, each word has its own unique index between 1 and $|\mathcal{W}|$. This also implies that the relation $|\mathcal{W}| = |\mathcal{I}_w|$ always holds.

Let $\boldsymbol{e}_w$ be a D-dimensional embedding vector for the word $w \in \mathcal{W}$, and let $\boldsymbol{E}$ denote an embedding matrix for all words in $\mathcal{W}$, where $\boldsymbol{E} = \mathbb{R}^{D\times|\mathcal{I}_w|}$. Then, we assume that the following relation always holds between $\boldsymbol{e}_w$ and $\boldsymbol{E}$:

$$\boldsymbol{e}_w = \boldsymbol{E}[z_e] \quad \text{where} \quad z_e = \zeta(w). \quad (2)$$

Therefore, the i-th column vector in the matrix $\boldsymbol{E}$ represents the word embedding of the corresponding word w that satisfies $i = \zeta(w)$.

Subwords and their embedding: Let $\mathcal{S}$ be a vocabulary for all pre-defined subwords obtained from the words in $\mathcal{W}$. Let $\eta_v(\cdot)$ represent a mapping function from a subword to the corresponding index of the subword, that is,

$$\eta_v(\cdot) : \mathcal{S} \to \mathcal{I}_s \;\text{ where }\; \mathcal{I}_s = \{1, \dots, |\mathcal{S}|\}. \quad (3)$$

Similar to ζ in Eq. 1, $\eta_v(\cdot)$ is generally defined as a bijective function[5]. In this case, each subword has its own unique index between 1 and $|\mathcal{S}|$, and the relation $|\mathcal{S}| = |\mathcal{I}_s|$ always holds.

Here, we introduce $\boldsymbol{v}_s$ as a D-dimensional vector for the subword $s \in \mathcal{S}$ and $\boldsymbol{V}$ as an embedding matrix for all subwords in $\mathcal{S}$, where $\boldsymbol{V} = \mathbb{R}^{D\times|\mathcal{I}_s|}$. Then, we also assume the following relation between $\boldsymbol{v}_s$ and $\boldsymbol{V}$:

$$\boldsymbol{v}_s = \boldsymbol{V}[z_v] \quad \text{where} \quad z_v = \eta_v(s). \quad (4)$$

Therefore, the j-th column vector in the matrix $\boldsymbol{V}$ represents the subword embedding of the corresponding subword s that satisfies $j = \eta_v(s)$.

Word to subword mapping: Additionally, we introduce a (abstract) function $\phi(\cdot)$ that maps a word $w \in \mathcal{W}$ to a list of subwords contained in the word w. We can define $\phi(\cdot)$ in detail from several choices. For example, if we define $\phi(\cdot)$ to extract all the character bi-grams appearing in a given word and $w =$ 'higher', then we obtain a list of total seven distinct subword indices of '⟨w⟩h', 'hi', 'ig', 'gh', 'he', 'er', 'r⟨/w⟩' as the return value of $\phi(w)$, where '⟨w⟩' and '⟨/w⟩' are special characters that represent the beginning and end of a word, respectively.

3.2 Task definition

Conceptually, we aim to reconstruct all the embeddings in $\boldsymbol{E}$ using $\boldsymbol{V}$ and a pre-defined subword mixing function $\tau(\cdot)$. Formally, our reconstruction problem is represented as a minimization problem of the following form:

$$\hat{\boldsymbol{V}} = \underset{\boldsymbol{V}}{\arg\min} \left\{ \Psi(\boldsymbol{E}, \boldsymbol{V}, \tau) \right\}, \quad (5)$$

[5] We redefine η_v as a surjective function in Section 5.2.

where $\Psi(\cdot)$ is a loss function used to calculate the total reconstruction loss between $\boldsymbol{E}$ and $\boldsymbol{V}$.

As a brief summary, our goal is to find $\hat{\boldsymbol{V}}$ at which the loss function $\Psi(\cdot)$ is minimized from the machine learning perspective. Note that the previous study, i.e., `BoS`, also utilized the above formulation for the reconstruction problem. Moreover, `MIMICK` can also be considered to utilize this formulation if $\boldsymbol{V}$ consists of all the single characters.

Subword mixing function $\tau(\cdot)$: The role of the function $\tau(\cdot)$ is to calculate an alternative embedding of the word embedding $\boldsymbol{e}_w$ using a list of subwords contained in the given word w. One of the most popular definitions of $\tau(\cdot)$ is to simply sum-up all the obtained subwords as follows:

$$\tau_{\mathtt{sum}}(\boldsymbol{V}, w) = \sum_{s \in \phi(w)} \boldsymbol{v}_s. \tag{6}$$

In fact, a subword mixing function of this form was utilized in the previous studies, such as `fastText` and `BoS`.

Loss function $\Psi(\cdot)$: First, to improve readability, we introduce $\hat{\boldsymbol{v}}_w$ as a short notation of $\tau(\boldsymbol{V}, w)$, namely $\hat{\boldsymbol{v}}_w = \tau(\boldsymbol{V}, w)$. There are also several possible choices for the definition of the loss function $\Psi(\cdot)$. Here, we consider utilizing a squared loss function $\Psi_{\mathtt{lsq}}$, which can be written as the summation of the squared losses over an individual embedding vector $\boldsymbol{e}_w$:

$$\Psi(\boldsymbol{E}, \boldsymbol{V}, \tau) = \sum_{w \in \mathcal{W}} C_w \left\| \boldsymbol{e}_w - \hat{\boldsymbol{v}}_w \right\|_2^2, \tag{7}$$

where C_w is a weight factor for each word. Intuitively, $\Psi(\cdot)$ is used to calculate the weighted sum of the L_2-norm distances between the reference vector $\boldsymbol{e}_w$ and a vector calculated by the subword mixing function $\tau(\cdot)$.

4 Reconstruction with Model Shrinkage

In this section, we briefly explains the background that we still need to consider the number of embedding vectors in the subword-based approach. Table 1 shows the statistics of the total number of embedding vectors and the total memory requirement in several different settings. As shown in row `(a)`, the original `fastText.600B` word embeddings consist of 2 million words, which require 2.2 GB to store them since each word has a $D = 300$ dimensional vector. If we consider using all the character N-grams obtained from all the words (row `(e)`), then surprisingly, the memory requirement becomes approximately 25GB, which is too large for practical use. Therefore, it is crucial to technically reduce the memory requirement.

ID	Setting		# of vecs	mem. (GB)
`(a)`	original `fastText.600B`		2.0 M	2.2 GB
`(b)`	char N-gram	$N = 1, 2, 3$	0.2 M	0.3 GB
`(c)`		$N = 3, 4, 5, 6$	6.2 M	7.1 GB
`(d)`		$N = 1$ to 6	6.3 M	7.2 GB
`(e)`		$N = 1$ to ∞	21.8 M	24.9 GB

Table 1: Statistics for each setting: The columns '# of vecs' and 'mem.' represent the number of embedding vectors and the memory requirement, respectively. M denotes one million, and we consider that a real value requires 4 bytes of storage in the calculation of the memory requirement.

A practical approach is to partially take advantage of a certain range of smaller N-grams, such as $N = 1$ to 3 (row `(b)`) or $N = 3$ to 6 (row `(c)`). However, smaller subword settings, such as row `(b)`, might markedly degrade the performance from the original word embeddings. Therefore, it is necessary to discover a better balance between the memory requirement (or the total number of embedding vectors) and performance.

5 Modifications to Improve Performance

In this section, we describe several modifications to simultaneously achieve the purpose of smaller number of embedding vectors but higher performance with the applicability of OOV words. To do so, we incorporate several techniques in the baseline word embedding reconstruction approach explained in Section 3. Roughly speaking, we enhance the mapping function $\eta_v(\cdot)$ and the subword mixing function $\tau(\cdot)$.

5.1 Frequent subwords: Modification of $\eta_v(\cdot)$

We take advantage of the top-F frequent subwords that can be counted from the words in vocabulary $\mathcal{W}$ as a subword vocabulary instead of all possible subwords $\mathcal{S}$. Let $\mathcal{S}_F$ represent the set of the top-F frequent subwords, where $\mathcal{S}_F \subseteq \mathcal{S}$. Then, we define a new mapping function $\eta_{v,F}(\cdot)$ as follows:

$$\eta_{v,F}(\cdot) : \mathcal{S}_F \to \mathcal{I}_{s,F} \text{ where } \mathcal{I}_{s,F} = \{1, \ldots, |\mathcal{S}_F|\}. \tag{8}$$

5.2 Memory sharing: Modification of $\eta_v(\cdot)$

In the baseline method, we assumed that the mapping function $\eta_v(\cdot)$ is a bijective function as de-

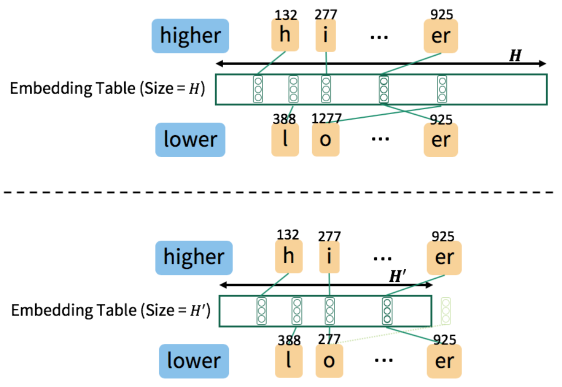

Figure 1: Intuitive idea of our memory-shared embeddings by hashing. Here, we assume $H' < H$.

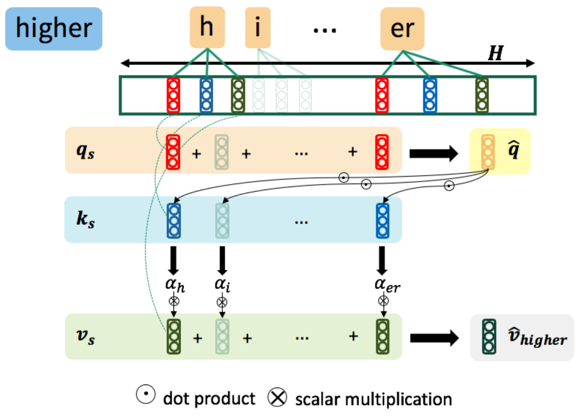

Figure 2: Illustration of how our KVQ self-attention operation calculates each word embedding.

scribed in Section 3.1. Again, this means that each subword has its own unique subword index. Here, we modify $\eta_v(\cdot)$ as a surjective function by introducing the following new mapping function $\eta_{v,H}(\cdot)$ as a replacement of $\eta_v(\cdot)$ in Eq. 3:

$$\eta_{v,H}(\cdot) : \mathcal{S} \rightarrow \mathcal{I}_{s,H} \text{ where } \mathcal{I}_{s,H} = \{1, \ldots, H\}. \tag{9}$$

Here, we assume $H < |\mathcal{S}|$. This mapping function $\eta_{v,H}(\cdot)$ implies that each subword is mapped to an index, but the index is not unique and may be shared between other subwords. Therefore, the number of subword embeddings can also be reduced to H by sharing the embeddings; thus, the subword embedding matrix $\boldsymbol{V}$ can also be reduced from $\boldsymbol{V} = \mathbb{R}^{D\times|\mathcal{S}|}$ to $\boldsymbol{V} = \mathbb{R}^{D\times H}$. Here, if we assume the relation $H \ll |\mathcal{S}|$, then we can greatly reduce the total embedding size.

There are several possible choices for the definition of the mapping function $\eta_{v,H}(\cdot)$. We select a simple hash function for $\eta_{v,H}(\cdot)$. This means that subword embeddings are randomly shared over the subwords. One large merit of using simple hash functions is that we require no external mapping structure of every (subword, subword index) pair, which also matches our goal of reducing the memory requirement in actual use cases. Figure 1 illustrates an intuitive idea of memory-shared embeddings by hashing.

5.3 Combination of $\eta_{v,F}(\cdot)$ and $\eta_{v,H}(\cdot)$

Also $\eta_{v,F}(\cdot)$ and $\eta_{v,H}(\cdot)$ can be combined step by step. First, we reduce the subword vocabulary $\mathcal{S}$ to top-F frequent subwords $\mathcal{S}_F$ as described in Section 5.1. Second, we apply our memory sharing method to only $\mathcal{S}_F$ in contrast to applying it to $\mathcal{S}$ in Section 5.2. Here, we define a new mapping function $\eta_{v,FH}(\cdot)$ as follows:

$$\eta_{v,FH}(\cdot) : \mathcal{S}_{\mathcal{F}} \rightarrow \mathcal{I}_{s,H} \text{ where } \mathcal{I}_{s,H} = \{1, \ldots, H\}. \tag{10}$$

5.4 Attention operation: Modification of $\tau(\cdot)$

Previous researches such as `fastText` and `BoS` treat $\tau(\boldsymbol{V}, w)$ as a summation of all subword embeddings described by Eq. 6. However, the summation is less expressive and it may lack capability in a memory-sharing setting since subwords share their embeddings randomly. One possible improvement is to handle the importance of each subword based on a given word during the calculation of $\Phi(\boldsymbol{V}, w)$.

A simple approach to deal with this phenomenon is to incorporate a "context-dependent" weighting factor for each subword in a given word. Thus, we consider the following subword mixing function $\tau_{\mathtt{kvq}}(\boldsymbol{V}, w)$ as:

$$\tau_{\mathtt{kvq}}(\boldsymbol{V}, w) = \sum_{s \in \phi(w)} a_{s,w} \boldsymbol{v}_s \tag{11}$$

where $a_{s,w}$ represents a context-dependent weighting factor of the subword s, where the "context" here means all the subwords obtained from word w.

To calculate $a_{s,w}$, we first introduce $\boldsymbol{k}_s$ and $\boldsymbol{q}_s$, which are similarly defined to $\boldsymbol{v}_s$ in Eq. 4, namely, $\boldsymbol{k}_s = \boldsymbol{V}[z_k]$, where $z_k = \eta_k(s)$ and $\boldsymbol{q}_s = \boldsymbol{V}[z_q]$, where $z_q = \eta_q(s)$. Similar to $\eta_v(\cdot)$ in Eq. 3, $\eta_k(\cdot)$ and $\eta_q(\cdot)$ are two distinct mapping functions that map a given subword s into a subword index. Then, we introduce a key-value-query (KVQ) self-attention operation inspired by

abbre.	data size	number of OOV data `fastText.600B`	`GloVe.840B`
Word similarity estimation (**WordSim**)			
MEN	3,000	0	0
M&C	30	0	0
MTurk	287	0	0
RW	2,034	37	36
R&G	65	0	0
SCWS	2,003	2	2
SLex	998	0	0
WSR	252	0	0
WSS	203	0	0
Word analogy estimation (**Analogy**)			
GL	19,544	0	0
MSYN	8,000	1000	1000

Table 2: Evaluation datasets used in our experiments. MEM (Bruni et al., 2014), M&C (Miller and Charles, 1991), MTurk (Radinsky et al., 2011), RW (Luong et al., 2013), R&G (Rubenstein and Goodenough, 1965), SCWS (Huang et al., 2012), SLex (Hill et al., 2014), WSR and WSS (Agirre et al., 2009), GL (Mikolov et al., 2013a), and MSYN (Mikolov et al., 2013b).

method	hyper-parameters		$\lvert\mathcal{W}\rvert$	$\lvert\mathcal{S}\rvert$	size (GB)
`fastText.600B`			2M	–	2.23GB
`SUM-F`	$F = 0.5$M	-	2M	0.5M	0.59GB
`SUM-H`	-	$H = 0.5$M	2M	21.8M	0.59GB
`KVQ-H`	-	$H = 0.5$M	2M	21.8M	0.59GB
`SUM-FH`	$F = 1.0$M	$H = 0.5$M	2M	1.0M	0.59GB
`KVQ-FH`	$F = 1.0$M	$H = 0.5$M	2M	1.0M	0.59GB
`SUM-F`	$F = 0.2$M	-	2M	0.2M	0.23GB
`SUM-H`	-	$H = 0.2$M	2M	21.8M	0.23GB
`KVQ-H`	-	$H = 0.2$M	2M	21.8M	0.23GB
`SUM-FH`	$F = 1.0$M	$H = 0.2$M	2M	1.0M	0.23GB
`KVQ-FH`	$F = 1.0$M	$H = 0.2$M	2M	1.0M	0.23GB

Table 3: Statistics for our methods.

`Transformer` (Vaswani et al., 2017), that is,

$$a_{s,w} = \frac{\exp(Z\hat{\boldsymbol{q}} \cdot \boldsymbol{k}_s)}{\sum_{s' \in \phi(w)} \exp(Z\hat{\boldsymbol{q}} \cdot \boldsymbol{k}_{s'})}, \quad (12)$$

where Z is a scaling hyper-parameter. and $\hat{\boldsymbol{q}} = \sum_{s \in \phi(w)} \boldsymbol{q}_s$. Figure 2 illustrates how our KVQ self-attention operation calculates each word embedding.

6 Experiments

6.1 Evaluation of model shrinkage

This section describes our experiments for evaluating the performance of the model shrinkage.

6.1.1 Settings

Evaluation data: Table 2 shows a summary of the evaluation datasets used in our experiments. We conducted experiments on well-studied linguistic benchmark datasets, i.e., nine for word similarity (**WordSim**) tasks and two for word analogy (**Analogy**) tasks. In this evaluation, we discarded data in the evaluation datasets if at least one of the words in the data was an OOV word. Note that this is the standard evaluation criterion used in the previous studies. By following this criterion, we investigate the effectiveness in terms of model shrinkage since we can fairly compare the performance with the original (word-based) word embeddings.

Pre-trained word embeddings: For the reconstruction target, we selected `fastText.600B`[6]. Note that it achieved the state-of-the-art performance on the WordSim and Analogy datasets (Bojanowski et al., 2017). The hyper-parameters $D = 300$ and $|\mathcal{W}| = 2$M were automatically obtained from the properties of `fastText.600B`.

Hyper-parameters for training: We took advantage of a N-grams' range of $N = 3$ to 30. We adopted Adam (Kingma and Ba, 2014) as our optimization algorithm to minimize Eq. 5. We set the following hyper-parameters for Adam: $\alpha = 0.0001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$. We leveraged a mini-batch training, whose size was 200, and trained each method for 300 epochs. For C_w in Eq. 7, we utilized the occurrence information calculated from a large external corpus. Then, we set $Z = \sqrt{D}$ for all experiments.

Comparison: We compared the following five distinct settings of subword-based reconstruction of word embeddings.

1. **SUM-F**: Select $\eta_{v,F}(\cdot)$ in Eq. 8 (Section 5.1) for the subword mapping function and $\tau_{\texttt{sum}}(\cdot)$ in Eq. 6 for the subword mixing function.
2. **SUM-H**: As in the first setting but substitute $\eta_{v,F}(\cdot)$ with $\eta_{v,H}(\cdot)$ in Eq. 9 (Section 5.2).
3. **KVQ-H**: As in the second setting but substitute $\tau_{\texttt{sum}}(\cdot)$ in Eq. 6 with $\tau_{\texttt{kvq}}(\boldsymbol{V}, w)$ in Eq. 11 (Section 5.4).
4. **SUM-FH**: As in the second setting but substitute $\eta_{v,H}(\cdot)$ with $\eta_{v,FH}(\cdot)$ in Eq. 10 (Section 5.3).
5. **KVQ-FH**: As in the third setting but substitute $\eta_{v,H}(\cdot)$ with $\eta_{v,FH}(\cdot)$ in Eq. 10 (Section 5.3).

6.1.2 Results

Figures 3 show the performance/model size (or performance/number of embedding vectors)

[6]Additionally, we also conducted the same experiments using `GloVe.840B` instead of `fastText.600B`. See Appendix A for the results.

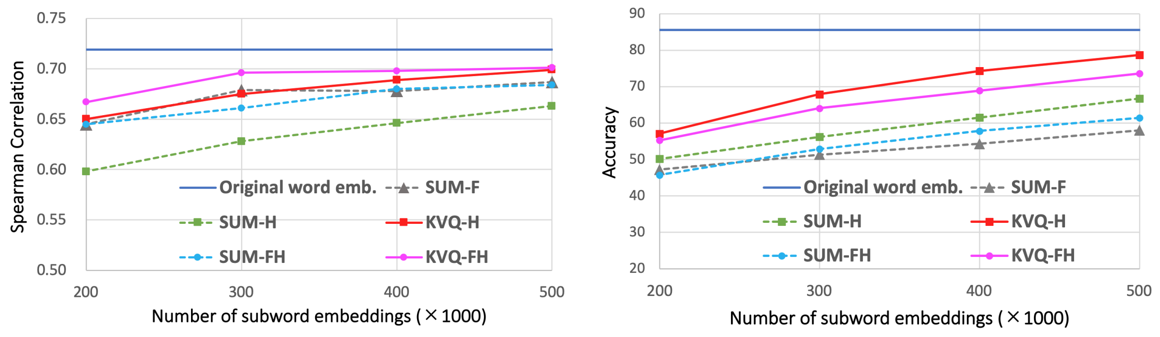

Figure 3: Performance/model size curves for WordSim (left) and Analogy (right). The x-axis represents the number of subword embeddings. The y-axis represents the performance evaluated by macro-average of Spearman's rho (left) and micro-average accuracy (right), respectively.

			WordSim										Analogy		
method	hyper-parameters		MEN	MC	MTurk	RW	R&G	SCWS	Slex	WSR	WSS	Macro	GL	MSYN	Micro
`fastText.600B`			.815	.850	.735	.572	.871	.684	.471	.640	.835	.719	84.9	87.8	85.6
`SUM-F`	$F = 0.5$M	-	.768	.829	.746	.566	.817	.668	.402	.578	.806	.687	53.3	71.2	58.0
`SUM-H`	-	$H = 0.5$M	.740	.824	.708	.533	.811	.652	.367	.536	.797	.663	61.9	79.9	66.7
`KVQ-H`	-	$H = 0.5$M	789	**.848**	.734	.562	**.836**	**.687**	.423	.597	.811	.699	**76.4**	**85.2**	**78.7**
`SUM-FH`	$F = 1.0$M	$H = 0.5$M	.772	.802	.743	**.569**	.813	.664	.397	.578	.816	.684	56.4	75.4	61.4
`KVQ-FH`	$F = 1.0$M	$H = 0.5$M	**.799**	.812	**.749**	.564	.816	.684	**.425**	**.630**	**.826**	**.701**	70.9	81.2	73.6
`SUM-F`	$F = 0.2$M	-	.731	.809	.710	.526	.777	.642	.369	.482	.762	.645	40.3	66.2	47.2
`SUM-H`	-	$H = 0.2$M	.660	.762	.671	.476	.756	.600	.310	.414	.732	.598	42.9	70.4	50.2
`KVQ-H`	-	$H = 0.2$M	**.773**	**.820**	.694	.523	.793	.652	.360	.500	.778	.650	**50.8**	**74.6**	**61.9**
`SUM-FH`	$F = 1.0$M	$H = 0.2$M	.719	.801	**.734**	.525	.798	.635	.334	.490	.767	.645	39.3	63.9	45.8
`KVQ-FH`	$F = 1.0$M	$H = 0.2$M	.754	.805	.731	**.542**	**.800**	**.667**	**.385**	**.523**	**.797**	**.667**	49.1	72.7	55.3

Table 4: Results of model shrinkage experiments by reconstructing the `fastText.600B` embeddings. Each dataset in WordSim and Analogy was evaluated by Spearman's rho and accuracy, respectively. 'Macro' and 'Micro' represent the macro-average of Spearman's rho over all WordSim datasets and the micro-average of accuracy over all Analogy datasets.

curves for WordSim and Analogy, respectively. Each plot is the macro-average of Spearman's rho (WordSim) or micro-average of accuracy (Analogy) of all evaluation datasets. Moreover, Table 3 shows the statistics of each setting and Table 4 show a summary of the detailed results for all datasets. Overall, we observed a consistent tendency that the methods using KVQ obtained the best performance compared with the methods using SUM. Notably, `KVQ-H` significantly outperformed `SUM-F`, `SUM-H` and `SUM-FH` by more than 10 points in terms of the micro-average accuracy of all Analogy datasets when the model size was 0.5M. The difference between `KVQ-H` and `SUM-H` shows that the approach of memory-shared embeddings is particularly useful when we combined it with the KVQ operation. Moreover, in some cases, we observed that `KVQ-H` achieved the performance of the original word embeddings `fastText.600B` when $H = 0.5$M. This means that `KVQ-H` with $H = 0.5$M with successfully reduced the model size nearly fourfold compared with the original word embeddings while maintaining the original performance.

6.1.3 Analysis

Time efficiency of SUM and KVQ: We investigated the difference in time efficiency between SUM and KVQ. We calculated the time computing word embeddings from subword embeddings using each operation. SUM and KVQ took 5.7×10^{-6} and 1.6×10^{-5} seconds per word, respectively, i.e., KVQ took 2.8 times longer than SUM. However, the calculation speed per word is sufficiently high to be negligible in the real applications since other operations such as calculating deep neural networks may take much longer.

6.2 Experiments of OOV word embeddings

This section describes our experiments for evaluating the performance of OOV word embeddings.

6.2.1 Settings

Evaluation data: We used the identical nine WordSim datasets used in Section 6.1.

method	hyper-parameters		Macro
Random	-	-	.110
SUM-F	$F = 0.5$M	-	**.640**
SUM-H	-	$H = 0.5$M	.609
KVQ-FH	-	$H = 0.5$M	.598
SUM-FH	$F = 1.0$M	$H = 0.5$M	.626
KVQ-FH	$F = 1.0$M	$H = 0.5$M	.636
SUM-F	$F = 0.2$M	-	.611
SUM-H	-	$H = 0.2$M	.552
KVQ-H	-	$H = 0.2$M	.566
SUM-FH	$F = 1.0$M	$H = 0.2$M	.609
KVQ-FH	$F = 1.0$M	$H = 0.2$M	**.614**

Table 5: Results of (synthetic) OOV word experiments on WordSim by reconstructing the fastText.600B embeddings. The performance was evaluated by Spearman's rho.

method	hyper-parameters		$\|\mathcal{W}\|$	$\|\mathcal{S}\|$	RW
Random	-	-	0.16M	-	.452
MIMICK	-	-	0.16M	<1K	.201
BoS	-	-	0.16M	0.53M	.46*
SUM-F	$F = 0.04$M	-	0.16M	0.04M	.513
SUM-H	-	$H = 0.04$M	0.16M	2.03M	.485
KVQ-H	-	$H = 0.04$M	0.16M	2.03M	.509
SUM-FH	$F = 0.50$M	$H = 0.04$M	0.16M	0.50M	.488
KVQ-FH	$F = 0.50$M	$H = 0.04$M	0.16M	0.50M	**.522**
fastText	-	-	0.16M	0.53M	.48*

Table 6: Results of OOV experiments on the Stanford Rare Word dataset. * indicates the values reported by Zhao et al. (2018). Note that fastText learned subword embeddings from an English Wikipedia dump since this method is not a reconstruction method.

Preparation of training data: As we showed in Table 2, the numbers of OOV problems for fastText.600B and GloVe.840B are indeed very small. This is because their vocabulary sizes exceeds 2 million words, and the words contained in the evaluation datasets tend to be 'non-rare words' in general. Therefore, it is difficult to precisely evaluate the effectiveness of the estimation of OOV word embeddings.

To overcome this difficulty, we artificially made our reconstruction problem much more difficult, namely, we discarded the words contained in the evaluation datasets from the vocabulary $\mathcal{W}$ for training. This means that all the problems in the evaluation datasets now became OOV problems. In other words, the number of OOV data in Table 2 in this setting always matches to the evaluation data size, such as 2034 for RW.

Other settings: We used the same experimental settings as used in Section 6.1 unless otherwise specified. For example, we used fastText.600B as the reconstruction target and the same training hyper-parameters.

6.2.2 Results

Table 5 shows the results of the (synthetic) OOV word experiments. First, we observed that the performance of the Random baseline's was nearly equal to zero across all the datasets. This means that there is no correlation between the Random and human-annotated scores. Importantly, the performances of KVQ-FH, SUM-FH and SUM-F were significantly improved by 44-53 points from that of Random. This result indicates that KVQ-FH, SUM-FH and SUM-F successfully predicted the OOV word embeddings.

However, we also observed no significant difference between KVQ-FH, SUM-FH and SUM-F for both the $H = 0.5$M and $H = 0.2$M settings.

6.2.3 Comparison with previous studies

As we discussed in Section 2, several closely related methods have also tackled to solve the OOV word issue, such as MIMICK and BoS. We aim to directly compare our approach with these methods to investigate whether it can outperform them. However, these methods (or available authors' codes) do not work on the large-vocabulary settings employed in Sections 6.1 and 6.2. Thus, as an alternative, we strictly followed the experimental settings described in (Zhao et al., 2018) and compared the performance under fair conditions.

Evaluation data: In their evaluation setting, they evaluated the OOV word performance over RW shown in Table 2. They included all the words appearing in RW as the evaluation data, in contrast to discarding the OOV data as in Section 6.1.

Pre-trained word embeddings: The target word embeddings for the reconstruction were the embeddings trained on Google News with 100 billion tokens[7] that were pre-cleaned by (Zhao et al., 2018). The resultant embeddings consist of 0.16M lower-cased word embeddings.

Comparison: We compared our approach with the following related methods:

1. Random: the performance when we used random vectors for OOV words.
2. MIMICK[8] (Pinter et al., 2017).
3. BoS[9] (Zhao et al., 2018).

Other settings: The shared memory size H was set to 0.04M since the vocabulary of this setting

[7]https://code.google.com/archive/p/word2vec/
[8]https://github.com/yuvalpinter/Mimick
[9]https://github.com/jmzhao

method	hyper-parameters		K	size (GB)	F1
fastText.600B	-	-	20	2.23GB	90.3
KVQ-FH	F = 1.0M	H = 0.5M	100	0.59GB	**90.4**
KVQ-FH	F = 1.0M	H = 0.2M	100	0.23GB	89.3
GloVe.840B	-	-	10	2.45GB	**90.8**
KVQ-FH	F = 1.0M	H = 0.5M	100	0.59GB	90.6
KVQ-FH	F = 1.0M	H = 0.2M	50	0.23GB	90.2

Table 7: Results of the NER experiments on the CoNLL-2003 dataset.

method	hyper-parameters		K	size (GB)	Acc
fastText.600B	-	-	10	2.23GB	87.8
KVQ-FH	F = 1.0M	H = 0.5M	20	0.59GB	**88.0**
KVQ-FH	F = 1.0M	H = 0.2M	10	0.23GB	87.6
GloVe.840B	-	-	1	2.45GB	**88.3**
KVQ-FH	F = 1.0M	H = 0.5M	10	0.59GB	87.8
KVQ-FH	F = 1.0M	H = 0.2M	20	0.23GB	87.6

Table 8: Results of the TE experiments on the SNLI dataset.

was relatively very small compared with that in our experiments, i.e., 2M vs 0.16M.

Results: Table 6 shows a comparison with the related methods. All our reconstruction methods outperformed BoS, which was the previous state-of-the-art method, with substantial improvements by 2-6 points. Moreover, KVQ-FH achieved the best performance in this comparison.

6.3 Evaluation on downstream tasks

To investigate the effectiveness of our reconstucted embeddings in downstream tasks, we evaluated them in the named entity recognition (NER) and the textual entailment (TE) tasks.

6.3.1 Settings

Evaluation data: We used the CoNLL 2003 dataset (Tjong Kim Sang and De Meulder, 2003) for an NER experiment and the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015) for a TE experiment.

Other settings: We used fastText.600B and GloVe.840B as the target word embeddings for the reconstruction. For our reconstruction embeddings, we calculated the embeddings of all the words in the datasets, thus there exist no OOV words when using our methods.

We used AllenNLP[10] to train base NER and TE models. We basically used the provided hyper-parameter values in their repository for both training and testing. Additionally, we added one hyper-parameter K to re-scale embeddings (i.e., multiply all the elements in the embeddings by K) since we learned that the re-scaling may significantly affect the overall performance of downstream tasks in certain situation. We search K from $[1, 5, 10, 20, 50, 100]$ on the validation set of each dataset.

[10] https://allennlp.org/

6.3.2 Results

Tables 7 and 8 show the comparison between the original (large) embeddings and our reconstructed (small) embeddings. When $H = 0.5$M, the performances of our reconstructed embeddings are equivalent to or even better than the ones of the original embeddings. One might be surprised the improved results by KVQ-FH since the model sizes of KVQ-FH were relatively very small comparing with the original embeddings. However, this may be a reasonable observation since our method additionally offered the embeddings of OOV words that cannot be handled by the original embeddings. Moreover, even when $H = 0.2$M, i.e. the model size was approximately ten times smaller, the degradation was less than 1.0, which is considered to be sufficiently acceptable for real-world systems if we can significantly reduce the model (system) size.

7 Conclusion

We discussed and investigated an approach that reconstructs subword-based word embeddings in a reduced memory space. We demonstrated that memory-shared embeddings with the KVQ self-attention operation significantly outperformed the conventional summation-based approach, such as BoS. Moreover, our best setting successfully reduced the number of embedding vectors to approximately ten times smaller than that of the original word embeddings while maintaining an acceptable performance loss on the downstream tasks. We also confirmed the effectiveness of our approach in terms of the applicability of OOV words. We believe that our reconstructed subword-based word embeddings can be better alternatives of fastText.600B and GloVe.840B because they require less memory requirement and have high applicability of OOV words.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 15H01702 and JST CREST Grant Number JPMJCR1513, Japan.

References

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. 2009. A Study on Similarity and Relatedness Using Distributional and WordNet-based Approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 19–27.

Dzmitry Bahdanau, Tom Bosc, Stanislaw Jastrzebski, Edward Grefenstette, Pascal Vincent, and Yoshua Bengio. 2017. Learning to compute word embeddings on the fly. *arXiv preprint arXiv:1706.00286*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 632–642.

Elia Bruni, Nam Khanh Tran, and Marco Baroni. 2014. Multimodal Distributional Semantics. *J. Artif. Int. Res.*, 49(1):1–47.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 731–742.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1314–1324.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. Amr dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1831–1841.

Aurélie Herbelot and Marco Baroni. 2017. High-risk learning: acquiring new word vectors from tiny data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 304–309.

Felix Hill, Roi Reichart, and Anna Korhonen. 2014. SimLex-999: Evaluating Semantic Models with (Genuine) Similarity Estimation. *arXiv preprint arXiv:1408.3456*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Eric H. Huang, Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2012. Improving Word Representations via Global Context and Multiple Word Prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 873–882.

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

Thang Luong, Richard Socher, and Christopher Manning. 2013. Better Word Representations with Recursive Neural Networks for Morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning (CoNLL)*, pages 104–113.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013b. Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 746–751.

George A. Miller and Walter G. Charles. 1991. Contextual Correlates of Semantic Similarity. *Language & Cognitive Processes*, 6(1):1–28.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Represqentation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 2227–2237.

Mohammad Taher Pilehvar and Nigel Collier. 2017. Inducing embeddings for rare and unseen words by leveraging lexical resources. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 388–393.

Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. 2017. Mimicking word embeddings using subword rnns. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 102–112.

Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. 2011. A Word at a Time: Computing Word Relatedness Using Temporal Semantic Analysis. In *Proceedings of the 20th International Conference on World Wide Web (WWW)*, pages 337–346.

Herbert Rubenstein and John B. Goodenough. 1965. Contextual Correlates of Synonymy. *Commun. ACM*, 8(10):627–633.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5027–5038.

Jun Suzuki and Masaaki Nagata. 2016. Learning compact neural word embeddings by parameter space sharing. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2046–2052.

Jun Suzuki, Sho Takase, Hidetaka Kamigaito, Makoto Morishita, and Masaaki Nagata. 2018. An empirical study of building a strong baseline for constituency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 612–618.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning (CoNLL)*, pages 142–147.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008.

Nan Yu, Meishan Zhang, and Guohong Fu. 2018. Transition-based neural rst parsing with implicit syntax features. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 559–570.

Jinman Zhao, Sidharth Mudgal, and Yingyu Liang. 2018. Generalizing word embeddings using bag of subwords. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 601–606.