Block-wise Word Embedding Compression Revisited: Better Weighting and Structuring

Jong-Ryul Lee[1], Yong-Ju Lee[1,2], Yong-Hyuk Moon[1,2]*
[1]Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea
[2]University of Science and Technology (UST), Daejeon, Korea
{jongryul.lee,yongju,yhmoon}@etri.re.kr

Abstract

Word embedding is essential for neural network models for various natural language processing tasks. Since the word embedding usually has a considerable size, in order to deploy a neural network model having it on edge devices, it should be effectively compressed. There was a study for proposing a block-wise low-rank approximation method for word embedding, called *GroupReduce*. Even if their structure is effective, the properties behind the concept of the block-wise word embedding compression were not sufficiently explored. Motivated by this, we improve *GroupReduce* in terms of word weighting and structuring. For word weighting, we propose a simple yet effective method inspired by the term frequency-inverse document frequency method and a novel differentiable method. Based on them, we construct a discriminative word embedding compression algorithm. In the experiments, we demonstrate that the proposed algorithm more effectively finds word weights than competitors in most cases. In addition, we show that the proposed algorithm can act like a framework through successful cooperation with quantization.

1 Introduction

Deep neural networks have had lots of attention due to their great success in many applications. Recently, deep learning is being actively applied to edge devices like a smartphone with important reasons including data privacy and low latency. However, deep neural networks usually have a tremendous number of parameters, so that one does not simply deploy them on such devices having limited resources. In order to resolve this issue, there is a line of research compressing neural networks.

Existing works for neural network compression mainly focus on convolutional layers and fully connected layers. In addition to those layers, there is a special and important layer called word embedding which has a considerable size and is commonly used in natural language processing (NLP) tasks. A word embedding is represented by a matrix where each row vector corresponds to a word, which is used as a vector representation of the word. There are also many existing works for compressing a word embedding layer. Among those works, (Chen et al., 2018) proposed an interesting compression method, named *GroupReduce*, for word embedding which constructs word clusters and conducts low-rank approximation on blocks (sub-embedding matrices) induced by them. They also proposed a low-rank approximation method working with specific weights on words. Even if their structure is simple and effective, the properties behind the concept of the block-wise word embedding compression were not sufficiently explored.

The major contribution of this work is to propose two effective word weighting methods for block-wise word embedding compression and to exploit a non-uniform partitioning method for lightweight embedding structure. Based on them, we construct a Discriminative Block-wise word embedding compression algorithm (*DiscBlock*) which significantly outperforms *GroupReduce*. In addition, we show that it can be cooperated with another compression technique like quantization as a compression framework.

Outline. In this work, we first introduce a block-wise word embedding structure inspired by *GroupReduce* of (Chen et al., 2018). Next, we discuss better word weighting and clustering to build the structure. After that, we conduct extensive experiments to demonstrate the effectiveness of *DiscBlock* with various downstream tasks such as language modeling, machine translation, text classification, and question and answering.

2 Related Work

Word Embedding Compression. Word embedding is a crucial part for natural language process-

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 4379–4388
November 7–11, 2021.

ing, and it requires considerable size. Thus, many approaches have been proposed for compressing it. Several works were proposed for compact representation of word embedding. (Andrews, 2016) proposed a way of exploiting Lloyd's algorithm to get low-bit representations of embedding vectors. (Ling et al., 2016) studied 8-bit representations for word embedding with training. (Hubara et al., 2017) proposed low-bit quantized neural networks for convolutional neural networks and recurrent neural networks. More recent works focus on devising a better structure of word embedding or an optimized way of computing encodings. (Shu and Nakayama, 2018) proposed a method compressing word embedding through compositional discrete codes, which can be trained by gradient descent. (Shi and Yu, 2018) proposed a product quantization-based compression method, which divides an embedding matrix into sub-matrices via k-means clustering. In aspect of word clustering, it is similar to our clustering method, but we do not use embedding vectors as the targets of clustering. Instead, we use real-valued word weights. In a slightly different line of research, (May et al., 2019) proposed an evaluation score for the downstream performance of compressed word embeddings, which is named the eigenspace overlap score. In addition, (May et al., 2019) showed a lower bound of the eigenspace overlap score for a uniform simple quantization-based compression method to explain its empirical effectiveness. We do not use the eigenspace overlap score in this work, but the quantization method will be used in the experiments. (Kim et al., 2020b) proposed a codebook-based compression method supporting word-level adaptive code length. The adaptive code length of a word can be considered as a word importance measure, but the code length should be predefined on domain of very limited size.

Since word embedding is generally represented by a matrix, decomposition-based compression techniques and efficient embedding structures were proposed. (Chen et al., 2018) proposed the block-wise low-rank approximation method for word embedding. (Hrinchuk et al., 2020) devised a way of interpreting an embedding matrix into a 3-dimensional tensor and proposed an embedding structure by decomposing it with tensor-train decomposition. (Panahi et al., 2020) proposed a small-size word embedding structure inspired by quantum entanglement. (Lioutas, 2020) proposed a recent study for word embedding factorization based on distillation. As (Lioutas, 2020) conducted experiments for combining their approach with *GroupReduce*, it can be also applied to our algorithm.

There are lots of existing approaches for word embedding compression, but none of existing approaches deeply study word weights in sense of compressing word embedding.

Decomposition. Since this work is based on low-rank approximation, we also study decomposition-based model compression approaches. (Kim et al., 2016) proposed a low-rank Tucker decomposition on kernel tensors. (Yu et al., 2017) proposed a framework unifying low-rank approximation and pruning of kernel tensors, which assumes that kernels are likely to be low-rank and sparse. (Astrid and Lee, 2018) proposed a canonical polyadic decomposition-based compression method for approximating a convolutional layer. (Ma et al., 2019) proposed a variation of the transformer of (Vaswani et al., 2017) by decomposing multi-linear attention with Block-Term tensor decomposition (De Lathauwer, 2008). Note that the transformer of (Ma et al., 2019) contains word embedding, but it is not compressed in their work.

3 Preliminaries

3.1 Notations

The set of words, called a vocabulary, is denoted by V, and its size is denoted by n. We have a $n \times d$ embedding matrix E corresponding to V where d is the dimension of each word embedding vector and $n > d$. $\log x$ stands for the natural logarithm of x. $diag(x_1, \ldots, x_k) \in \mathbb{R}^{k \times k}$ is the diagonal matrix with the input arguments. For any vector $\mathbf{v}$, v_i denotes the i-th element of $\mathbf{v}$. In addition, when we conduct Singular Value Decomposition (SVD) on a matrix, the diagonal matrix is assumed to be already multiplied to another for simplicity.

3.2 Weighted Singular Value Decomposition

Consider a list of L consisting of the entire words, which is sorted in a certain order. Consider a vector $\mathbf{s}$ such that s_i is the weight assigned to the i-th word in L. Let us define a diagonal matrix $S = diag(\sqrt{s_1}, \sqrt{s_2}, \ldots, \sqrt{s_n})$. Then, (Chen et al., 2018) introduced a rewritten form of the objective function of weighted SVD and how to get the solution as follows.

$$\min_{U \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{d \times n}} \left\| SE - SUV^T \right\|_F^2 . \quad (1)$$

Suppose that we conduct SVD on SE instead of E and the result is $\widehat{U}\widehat{V}^T$. Then, the solution of the weighted SVD on E with weight vector $\mathbf{s}$ is $(U^*, V^*) = \left(S^{-1}\widehat{U}, \widehat{V}^T\right)$.

3.3 Block-wise Low-Rank Approximation

Let us introduce *GroupReduce*, which is the block-wise low-rank approximation for word embedding of (Chen et al., 2018). *GroupReduce* works with a set of multiple groups $\mathcal{G}$ such that the union of all the groups in $\mathcal{G}$ is the entire word set and they are disjoint. For grouping, (Chen et al., 2018) takes a simple approach, which is to sort words in descending order of frequency and partition them to same-size g groups.

For each group G_i in $\mathcal{G}$, we induce the sub-embedding matrix consisting of the embedding vectors of words in G_i, and it is denoted by E_i. In addition, suppose that each word w in G_i is associated with its frequency as a weight. Then, *GroupReduce* computes the weighted SVD of E_i with a certain rank r_i as $U_i \times V_i$. (Chen et al., 2018) set r_i to be $\frac{f_i}{f_c} r$ where f_i is the average frequency of words in G_i, f_c is the average frequency of words in the group with the least frequent words, and r is a user-specific rank to the group with the least frequent words. Finally, *GroupReduce* approximates E as following:

$$\begin{aligned} E &= [E_1, E_2, \ldots, E_g] \text{ (by reordering)} \\ &\approx \left[U_1 (V_1)^T, U_2 (V_2)^T, \ldots, U_g (V_g)^T\right], \end{aligned}$$

where $[A, B]$ is the concatenation with sub-embedding matrices A and B over words.

A Note on Refinement. (Chen et al., 2018) proposed an algorithm of refining this group assignment scheme with consideration of minimizing the total reconstruction error of the weighted SVD. However, it may be failed to get a better assignment, because words in a group with a low rank has great tendency to be moved to another with a high rank. This must be unintended and unhelpful due to the meaning of word weights. Thus, in this work, their refinement algorithm is not used.

4 Proposed Algorithms

DiscBlock uses the same block-wise word embedding structure of (Chen et al., 2018). In addition, given word weights, we assign a rank to each group in the same way of (Chen et al., 2018). The difference between *DiscBlock* and *GroupReduce* are made on word weighting and clustering, which are explained in this section.

4.1 Beyond Frequency: Better Weighting

Even if the concept of the word frequency is quite simple and it is reasonably useful, it is not the best option in many cases. For example, when the word frequency is used as the importance of a word to a document in information retrieval, the importance of unimportant words like 'is' can be overestimated. Since *GroupReduce* uses word frequency as a measure of word importance, it may have a similar problem that unimportant words are overestimated so that they may be falsely included in a high-rank word group. Motivated by this, we propose two different methods for word weighting.

4.1.1 Simple Yet Effective Word Weighting

In order to solve the problem of word frequency in information retrieval, the concept of the Term Frequency and Inverse Document Frequency (TF-IDF) score was introduced. This scoring method determines the importance of a word to a document with consideration of both frequency and the number of documents having it. Inspired by the concept of the TF-IDF score, we define TF-IDF based word importance as follows.

$$\begin{aligned} \text{tf}_{\text{avg}}(w) &= \frac{\alpha}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \frac{f_{w,D}}{\max_{i \in D} f_{i,D}}, \\ \text{idf}(w) &= 1 + \max\left\{\log \frac{|\mathcal{D}|}{|\mathcal{D}_w| + 1}, 0\right\}, \\ \text{tf-idf}_{\text{avg}}(w) &= \text{tf}_{\text{avg}}(w) \times \text{idf}(w) + \epsilon \end{aligned}$$

where α is a user-specific parameter for scaling, ϵ is a small value for avoiding zero, $\mathcal{D}$ is the entire document set, $\mathcal{D}_w$ is a set of documents having word w, and $f_{w,D}$ is the frequency of word w in document D. In this work, α and ϵ are set to be 0.1 and $\frac{1}{|\mathcal{D}|}$, respectively.

Rationale. $\text{tf}_{\text{avg}}(w)$ is a normalized term frequency and $\text{idf}(w)$ is the logarithm of the inverse document frequency of w. Note that for frequent words over many documents like 'is', the inverse document frequency is likely to be low, so that we can avoid to assign such words to a high-rank word group.

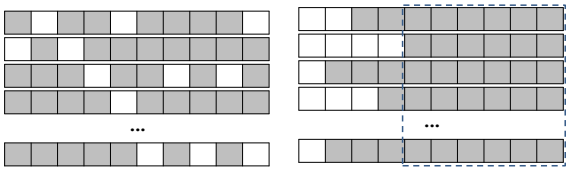
4.1.2 Differentiable Word Weighting

For more effectively achieving word weights than the TF-IDF based method, we devise a trainable (differentiable) word weighting method. Given a trained (target) model M, this method modifies M

and trains it to learn effective word weights. After the training process, trained word weights are used to form the diagonal matrix S and to construct the block-wise low-rank approximation. Note that when training word weights, the other weight parameters in M are not re-trained.

Word Importance through Masking. Consider a word w and its embedding vector $\mathbf{v}_w$ in the embedding matrix of M. Suppose that we assign a number of zeros to $\mathbf{v}_w$ uniformly at random and there is no loss of accuracy for M. Then, we can consider that the reason why $\mathbf{v}_w$ carries many zeros without loss of accuracy is that w is not an important word to M.

Based on this idea, we propose an advanced method for computing word weights based on masks on elements of the embedding matrix. The masks are formulated to effectively cause information bottleneck (low-rankness), which will be helpful for a model to select masks while minimizing task loss in training process. Suppose that we have a positional index p_w on $\mathbf{v}_w$, which is called a pivot and determined by the importance of w. The proposed method gives sparsity to $\mathbf{v}_w$ by replacing values after p_w in $\mathbf{v}_w$ with zeros, so that resulting masks are aligned as depicted in Figure 1(b).



(a) Unaligned Masking (b) Aligned Masking

Figure 1: Masks on elements in the embedding matrix E. Gray cells are masked elements. In (b), all cells in the dotted line are masked.

Then, we claim the following proposition.

Proposition 4.1. *For a masked embedding matrix $\hat{E}$ by the proposed method, if the rank of E is r, the rank of $\hat{E}$ is,*

$$rank(\hat{E}) \leq \min \left\{ \max_{w \in V} p_w, r \right\}.$$

By manipulating pivots, the proposed method can guarantee that the masked embedding matrix is low-rank (i.e., $rank(\hat{E}) < d$).

Making it Trainable. The remaining problem is how to determine pivot p_w for each word w depending on the importance of w in a differentiable way. In order to parameterize the pivot, consider a function $p(x_w)$ where x_w is a trainable parameter such that $0 < x_w \leq 1$ and it is defined as:

$$p(x_w) = \max \{ \lfloor d x_w \rfloor, 1 \}.$$

Then, we formulate a masking function $m : \mathbb{R} \rightarrow \{0, 1\}^d$ as following:

$$m^{(i)}(x_w) = \begin{cases} 1, & \text{if } i \leq p(x_w) \\ 0, & \text{otherwise,} \end{cases}$$

where $m^{(i)}(x_w)$ is the i-th element of $m(x_w)$. It is easy to see that the range of $m(\cdot)$ is the same as the output space of the pivot-based masking. In addition, since x_w is proportional to the number of non-masked elements, which is likely to have positive correlation with the importance of w, we use x_w as the word weight of w in this method.

Note that for $m^{(i)}(x_w)$, if $p(x_w) = i$, $m^{(i)}$ is not differentiable, and otherwise, its derivative is zero. This property leads to the fact that it is hard to train x_w with gradient descent due to the zero derivative issue. (Kim et al., 2020a) addressed a similar problem to this issue by introducing a trainable gate function. We use the same gradient shaping function $\beta : \mathbb{R} \rightarrow \mathbb{R}$ proposed by (Kim et al., 2020a) as following:

$$\beta(x) = \frac{Lx - \lfloor Lx \rfloor}{L},$$

where L is a large positive integer. The trainable masking function $\hat{m} : \mathbb{R} \rightarrow \mathbb{R}^d$ is defined as:

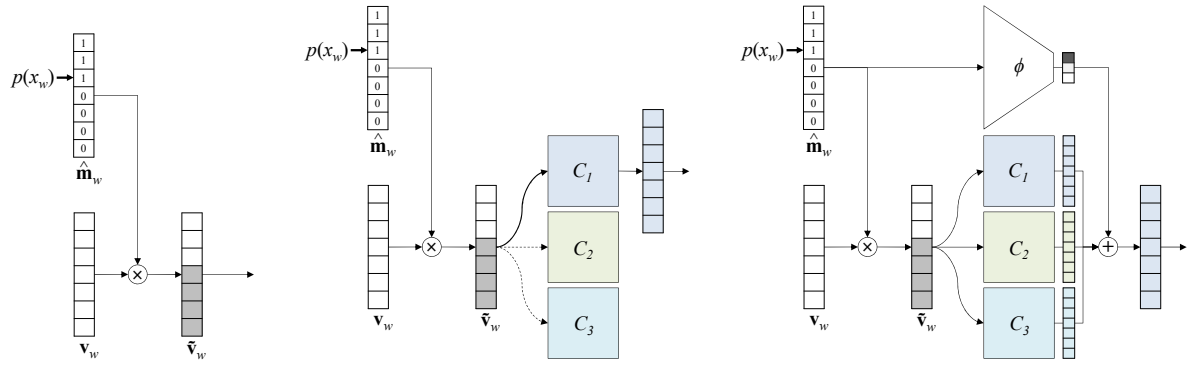
$$\hat{m}^{(i)}(x_w) = m^{(i)}(x_w) + \beta(p(x_w)). \quad (2)$$

It is easy to prove the uniform convergence of $\hat{m}(\cdot)$ to $m(\cdot)$ with a large value of L. The idea of this approach is that $\beta(\cdot)$ has an extremely small value near zero, but its derivative is one where $\beta(\cdot)$ is differentiable, so that we can train x_w.

Learning from Hunger. In order to learn word importance properly in the training process, we need to define an additional loss term. This is because without any regulation, the model must be trained to minimize the number of masked elements. The additional loss based on the sparsity of masked embedding vectors is defined as:

$$\mathcal{L}(\mathbf{x}; \gamma) = \lambda \left\| \overline{\gamma} - (\mathbf{1} - \mathbf{x}) \right\|_2^2, \quad (3)$$

where λ and γ are real-valued user-specific parameters, $\|\cdot\|_2^2$ is the $l2$-norm, $\overline{\gamma} \in \mathbb{R}^n$ is the vector where all the elements are γ, $\mathbf{1} \in \mathbb{R}^n$ is the ones



(a) No Compensation (Identity) (b) Discrete and Uniform Selection (c) Continuous Selection

Figure 2: Types of compensation functions. Gray elements in a masked vector represent masking. $\beta(\cdot)$ is omitted.

vector, and $\mathbf{x} \in \mathbb{R}^n$ is a trainable word weight vector. λ is set up with consideration of the ratio between task loss and $\mathcal{L}(\mathbf{x}; \gamma)$, and γ is defined to control the desired sparsity. This loss function leads the model to learn word importance with limited budget.

One Step Further. If we use masked embedding matrix $\hat{E}$ instead of E, the output distribution of the subsequent layers of M will be changed due to the rank reduction. In order to alleviate the unintended change, we propose a function $c(\cdot)$ which takes the masked word vector and the masking vector as inputs. This is called a compensation function, and it can be formulated in two ways.

Given a masked word vector $\tilde{\mathbf{v}}_w \in \mathbb{R}^d$ in $\hat{E}$, one way for the compensation is to define g linear layers, select one of them depending on what $p(x_w)$ is, and to make $\tilde{\mathbf{v}}_w$ pass through it. For simplicity and efficiency, the selection is uniformly conducted in forward pass. That is, if $\frac{d(i-1)}{g} < p(x_w) \leq \frac{di}{g}$, $\tilde{\mathbf{v}}_w$ will be passed through the i-th linear layer.

The other way is to let the model find effective selection with training. Given a masked word vector $\tilde{\mathbf{v}}_w \in \mathbb{R}^d$ and its masking vector $\hat{\mathbf{m}}_w \in \mathbb{R}^d$ computed by $\hat{m}(\cdot)$, $c(\cdot)$ is formulated as:

$$\phi(\hat{\mathbf{m}}_w) = \sigma\left(W_2 W_1 \hat{\mathbf{m}}_w + \mathbf{b}\right)$$

$$c(\tilde{\mathbf{v}}_w, \hat{\mathbf{m}}_w) = \sum_{i=1}^{g} \phi^{(i)}(\hat{\mathbf{m}}_w)\left(C_i \tilde{\mathbf{v}}_w + \mathbf{b}_i\right),$$

where σ can be the softmax function or Gumbel-Softmax, $W_1 \in \mathbb{R}^{\delta \times d}$, $W_2 \in \mathbb{R}^{g \times \delta}$, $\mathbf{b} \in \mathbb{R}^g$, and $C_i \in \mathbb{R}^{d \times d}$, and $\mathbf{b}_i \in \mathbb{R}^d$. C_i and $\mathbf{b}_i$ are the weight matrix and the bias of the i-th linear layer in $c(\cdot)$, respectively. W_1 and W_2 are weight matrices and $\mathbf{b}$ is a bias vector to determine which linear layer is used. δ is designed to have a lower value than g. W_1 and W_2 are initialized as all-ones matrices while $\mathbf{b}$ and $\mathbf{b}_i$ are initialized as zero-vectors. C_i is initialized as a matrix where diagonal entries are one and off-diagonal entries are zeros.

Including the case of no compensation, the two compensation functions are depicted in Figure 2.

The rationale behind the compensation functions is that we want to compute word weights while mimicking the block-wise low-rank approximation. Consider a sub-embedding matrix $E_i \in \mathcal{G}$ and its low-rank approximation $U_i(V_i)^T$. Since the rank r_i of U_i is smaller than d, we can consider that U_i has word embedding vectors projected to a lower dimensional embedding space. On the other hand, V_i^T can be seen as a linear transformation matrix toward the original dimensional space. Similarly, the compensation function acts like V_i^T, so that it will be trained to alleviate the impact of using masked word vectors to the subsequent layers.

4.2 Clustering

Recall that in *GroupReduce*, words are sequentially partitioned into same-size groups in the sorted list in descending order of frequency. Let us call this the uniform partitioning method. Even if *GroupReduce* implicitly assumes that words in the same group have similar importance, due to the power-law distribution of words in terms of frequency, words which have very different importance can be included in the same group. In addition, the uniform partitioning method hinders *GroupReduce* from achieving high compression ratio. This issue will be discussed in the experiments.

In order to address this problem, one decent option is to use the k-means clustering method instead

Table 1: Used Datasets and Models (OP means the performance of an original model and E. Size means the size of an embedding matrix.)

Dataset	n	d	E. Size	Model	Metric
PTB	10K	650	52MB	79MB	PPL
WikiText2	33K	650	173MB	200MB	PPL
WikiText103	268K	1,500	3.2GB	3.4GB	PPL
SNLI	34K	300	41MB	50MB	Acc.
SST-5	10K	300	23MB	26MB	Acc.
SQuAD	49K	300	59MB	75MB	F1
IWSLT14	25K (de) 18K (en)	620	106MB	243MB	BLEU

of the uniform partitioning method. It is trivial that with a proper word importance function, the k-means clustering method can more effectively collect words in the same group having similar importance than the uniform partitioning method.

5 Experiments

5.1 Implementation Details

Tasks, Datasets, and Models. We conduct extensive experiments to demonstrate the effectiveness of *DiscBlock*. We have following tasks: language modeling, question and answering, text classification, and machine translation.

Since we deal with many different types of datasets and models, we use various open-sourced implementations as follows. Note that we do not make any change over datasets and preprocessing implementations. For language modeling, we use Penn Treebank, WikiText2, and WikiText103 of (Merity et al., 2017) as datasets. We use a 2-layered LSTM model with dropout after the word embedding layer for the encoder. Our implementation for this task comes from the language modeling codebase provided by PyTorch examples[1]. For question and answering, we use SQuAD (Stanford Question Answering Dataset) 1.0 in (Rajpurkar et al., 2016) and the DrQA model proposed in (Chen et al., 2017). Our implementation for handling this dataset is based on the codebase[2] used in (May et al., 2019). For text classification, we use two datasets: SNLI (Stanford Natural Language Inference) in (Bowman et al., 2015) and SST-1 (Stanford Sentiment Treebank) in (Socher et al., 2013). For SNLI, we use a open-sourced codebase[3] providing a bidirectional LSTM. For SST-1, another open-sourced codebase[4] is used with TextCNN in (Kim, 2014). For machine translation, we use the IWSLT14 (International Workshop on Spoken Language Translation 2014) German-to-English dataset in (Cettolo et al., 2015). For this dataset, we use JoeyNMT[5] which is a lightweight framework for machine translation proposed in (Kreutzer et al., 2019). In addition, a recurrent neural network based on GRU with attention is used for this task. The basic statistics of the datasets presents in Table 1. All scores reported in this work come from test sets except SQuAD. For SQuAD, scores are computed from the validation set.

Training. In order to get base models, which have word embedding targets to compress, we use epochs specified in the open-sourced codebases except WikiText103. Due to the huge size of WikiText103, we use 10 epochs for training on it.

The learning rate and the number of epochs for retraining are varied over datasets. Retraining epochs are determined with consideration of total retraining time. If the dataset and the model are not large, the number of epochs for retraining is the same as that for training the base models from scratch. In addition, the learning rate for retraining is scaled to half or 10% of the original rate.

Note that the learning rate for training word weights is also experimentally determined. The number of epochs for training word weights is set to be usually much smaller than that for training the base models.

Trainable Weight Initialization. For the differentiable word weighting method, given word w, x_w is initialized to tf-idf$_{\text{avg}}(w)$. Since the differentiable word weighting method is proposed to get better weights than tf-idf$_{\text{avg}}$, it is a good starting point.

Compression Ratio. In order to fairly evaluate the effectiveness of each comparison method, we control compression ratio to be approximately $50\times$ for IWSLT14 and $20\times$ for the other datasets if there is no mention about compression ratio.

Hyperparameters. We have several hyperparameters for the word weighting methods. For simplicity, δ is 1 and γ is set to 0.95 or 0.99.

λ is determined through multiple experiments and it ranges from 0.5 to 25.0 except SNLI. For SNLI, λ is set to be 0. In this case, the sparsity loss is not helpful to train effective masks.

The number of groups g is experimentally determined to be 5. We conducted experiments for 10 and 20, but both *DiscBlock* and *GroupReduce* show stable performance with $g = 5$.

[1] https://github.com/pytorch/examples
[2] https://github.com/HazyResearch/smallfry
[3] https://github.com/imran3180/pytorch-nli
[4] https://github.com/Doragd/Text-Classification-PyTorch
[5] https://github.com/joeynmt/joeynmt

Table 2: Overall Compression Performance (OP means the performance score of an original model.)

Dataset	OPS	Retrain	SVD		*TensorTrain*		*GroupReduce*		*DiscBlock*-F		*DiscBlock*-T		*DiscBlock*-D	
			Score	Ratio	Score	Ratio	Score	Ratio	Score	Ratio	Score	Ratio	Score	Ratio
PTB	80.8	Before	372.1	20×	-	20×	181.7	19×	156.6	19×	136.4	21×	**125.7**	20×
		After	96.0		141.1		102.4		92.9		92.0		**88.7**	
WikiText2	93.3	Before	1,246.9	21×	-	20×	188.6	20×	172.4	20×	150.6	20×	**139.3**	20×
		After	115.0		150.8		113.0		107.8		104.3		**102.7**	
WikiText103	61.0	Before	1,882.1	20×	-	-	*84.3	20×	122.2	20×	92.6	20×	**75.3**	20×
		After	83.2		-		*76.9		82.5		72.4		**67.6**	
SNLI	82.6	Before	37.4	20×	-	20×	*57.9	19×	56.7	19×	**68.8**	20×	68.5	20×
		After	**81.6**		80.2		*81.0		81.1		80.2		80.0	
SST-5	43.7	Before	40.1	19×	-	20×	40.8	20×	39.3	20×	40.5	20×	**43.3**	20×
		After	43.6		41.0		42.8		42.2		42.8		**44.8**	
SQuAD	74.2	Before	30.1	20×	-	20×	*68.2	20×	66.2	21×	69.7	20×	**71.2**	20×
		After	71.7		72.7		***73.3**		72.2		72.6		73.2	
IWSLT14	30.6	Before	13.2	50×	-	48×	*23.5	50×	13.5	50×	16.3	50×	**24.2**	50×
		After	**29.3**		27.0		*26.9		27.3		24.0		28.8	

Table 3: More Powerful Compression Test ($\approx 50\times$, Retrained)

Methods	PTB	SQuAD	SST-5	IWSLT14
SVD	126.0	72.0	40.2	**29.3**
TensorTrain	161.7	72.2	**40.5**	27.0
GroupReduce	*119.5	*71.5	*36.3	*26.9
DiscBlock-F	145.4	69.8	34.2	27.3
DiscBlock-T	122.8	71.5	39.0	24.0
DiscBlock-D	**112.0**	**72.3**	39.7	28.8

Competitors. We have three competitors: SVD, *TensorTrain*, and *GroupReduce*.

SVD is the truncated singular value decomposition method. The compression ratio of SVD is controlled by manipulating the number of singular values to use.

TensorTrain is the tensor-train decomposed-based method in (Hrinchuk et al., 2020). In (Hrinchuk et al., 2020), *TensorTrain* is computed by training from scratch. Similarly, we train it from scratch, but the other trainable parameters are trained from pretrained values provided by the base model. Note that for each dataset, the learning rate to train *TensorTrain* is experimentally selected from between the learning rate for training the base model and that for retraining.

For *GroupReduce*, we use the refinement of (Chen et al., 2018). *GroupReduce* first constructs uniform partitions and refines them via local search heuristics, but the uniform partition construction sometimes fails due to the power-law frequency distribution of words. In this case, even if the rank assigned to the least frequent partition is 1, the compressed embedding size is much larger than the target compression ratio. For comparing *GroupReduce* with our algorithm even in such a case, we add a base value to each frequency score for smoothing the distribution. The base value is 2^k where k is the minimum value for achieving the target compression ratio. We denote results where this remedy is applied by * as a prefix.

5.2 Results

Let us denote *DiscBlock* with frequency, tf-idf$_{\text{avg}}$, and the differentiable word weighting method by *DiscBlock-F*, *DiscBlock-T*, and *DiscBlock-D*, respectively. The implementation is available at the repository[6].

5.2.1 Overall Performance

The overall compression performance of the comparison methods presents in Table 2. *TensorTrain* is not tested for WikiText103 because it requires too much cost to be trained.

The table presents that *DiscBlock* is much more effective than SVD and *GroupReduce* in most cases. Especially, the gap between *DiscBlock* and the others is remarkable in terms of model performance before retraining. This merit is critical when the base model and the dataset are extremely large.

We also conducted experiments about more powerful compression scenarios. The results are shown in Table 3. Compared to the other competitors, *DiscBlock-D* has the most stable performance even in such scenarios.

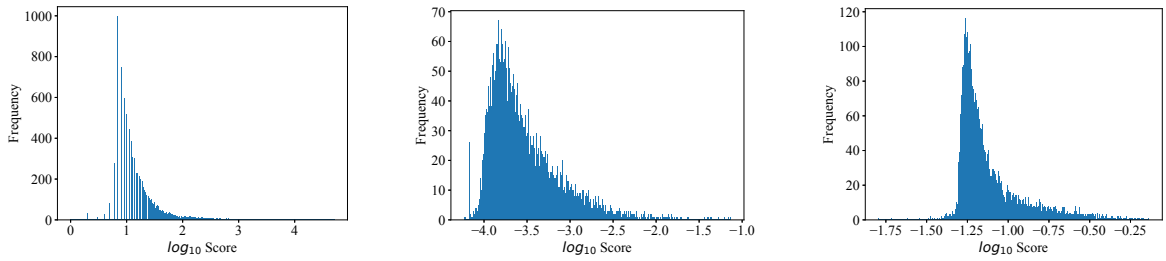
5.2.2 Word Weight Distribution

Figure 3 depicts the distribution of word weights computed by frequency, tf-idf$_{\text{avg}}$, and the differentiable word weighting method. Compared to frequency, tf-idf$_{\text{avg}}$ and the differentiable word weighting method provide more smoothed distributions. Such smoothed distributions have advantage to avoid over-estimating words which are considered important by a word weighting method.

5.2.3 Effectiveness of Compensation Functions

We compared compensation functions for the differentiable word weighting method and the results

[6] https://github.com/etri-edgeai/nn-comp-discblock

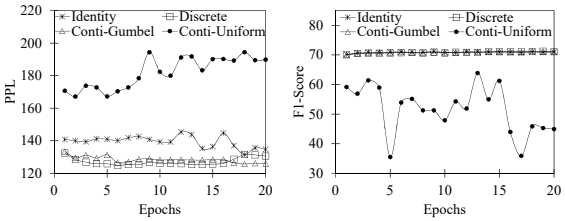


(a) Frequency (b) tf-idf$_{avg}$ (c) Differentiable Weighting

Figure 3: Word Weight Distribution (PTB). The horizontal axis is logarithmically scaled.

Table 4: Test on Compensation Functions for Training Word Weights

Methods	PTB	SQuAD	SST-5	IWSLT14
Before Training	137.4	69.7	40.5	16.3
Identity	**124.8**	71.2	40.1	23.7
Discrete and Uniform	130.0	**71.3**	40.2	23.4
Conti-Gumbel	125.7	71.2	**43.3**	**24.2**
Conti-Softmax	125.8	71.0	41.0	23.4



(a) PTB (b) SQuAD

Figure 4: Change of Compressed Model Performance on Training Word Weights

Table 5: Comparing with Uniform Partitioning (UP) (Retrained)

Methods	PTB	SQuAD	SST-5	IWSLT14
UP-F	105.0	-	43.2	-
UP-T	89.1	72.4	43.5	-
UP-D	91.0	73.2	42.9	-
DiscBlock-F	92.9	72.2	42.2	27.3
DiscBlock-T	92.0	72.6	42.8	24.0
DiscBlock-D	**88.7**	**73.2**	**44.8**	**28.8**

are presented in Table 4. Conti-Gumbel is the continuous selection with Gumbel-Softmax, while Conti-Softmax is that with the softmax function. The results show that Conti-Gumbel is the best except PTB and SQuAD. Even for PTB and SQuAD, Conti-Gumbel achieves almost same performance compared to the best competitors of them. Meanwhile, the discrete method is not that effective compared to Identity. This may be because the discrete method divides words by the uniform selection, which is different from partitions computed by the k-means clustering method.

5.2.4 Why Non-uniform Clustering Matters

The k-means (non-uniform) clustering method is necessary for achieving high-level compression performance. The first reason is shown in Table 5, which contains comparing results in terms of architectural effectiveness. In this table, the uniform partitioning method does not have any result for IWSLT14, because it cannot achieve near $20\times$ compression ratio even if the assigned rank to the least frequent word group is 1. It does not have any result for SQuAD with frequency either due to the power-law frequency distribution of words. In addition, the results imply that the k-means clustering method provides slightly better compressed word embedding structures in many cases.

In Figure 4, Conti-Uniform stands for using the uniform partitioning method to construct word groups with word weights computed by the differentiable weighting method. Compared to results with the k-means clustering method (*DiscBlock*), Conti-Uniform fails to find good word weights.

5.2.5 Another Application: Knowledge Embedding Compression

Knowledge embedding consists of numerical vectors representing entities and relations in a knowledge graph. We conducted a toy experiment for demonstrating the effectiveness of our algorithm to knowledge embedding compression. The dataset used in this experiment is FB15K-237, which consists of 14.5K entities and 237 relations. Since the number of relations is ignorable, we compress only the entity embedding matrix. In the table, H@10 is the proportion of correct entities ranked in the top 10 entities and MRR is the mean reciprocal rank measuring the number of correct predicted triples. We implement this experiment based on the opensource codebase[7].

[7]https://github.com/thunlp/OpenKE

Table 6: Knowledge Embedding Compression for FB15K-237 ($\approx 6\times$)

Models	TransE		TransD	
Metrics	MRR	H@10	MRR	H@10
OP	27.8	46.2	28.3	48.4
SVD	13.1	27.1	11.6	24.1
DiscBlock-F	18.6	31.1	17.1	28.8
DiscBlock-D	**18.8**	**33.9**	**17.3**	**31.1**
SVD (Retrain)	**24.5**	**42.7**	25.1	**43.8**
TensorTrain (Retrain)	22.8	40.1	24.5	42.0
DiscBlock-F (Retrain)	24.4	41.7	**25.6**	43.3
DiscBlock-D (Retrain)	23.8	41.9	25.0	42.8

The results are shown in Table 6. For this task, frequency is the number of occurrences of an entity in triples. tf-idf$_{\text{avg}}$ is not applied because there is no similar concept to a document in this task. Compared to SVD, *DiscBlock-D* is somewhat behind in terms of MRR and H@10 after retraining. However, it still has better performance than the competitors before retraining. This result implies that *DiscBlock-D* well approximates the original embedding matrix, but the block-wise structure is not helpful to achieve high performance via retraining. We believe that this observation can be a good starting point for future research.

5.2.6 Toward a Compression Framework: Cooperating with Quantization

Since we cluster words with their importance, each sub-embedding matrix includes words having similar word importance. That is, given a compression method, which is controllable in terms of compression strength, we can apply it to each sub-embedding matrix according to its average word importance. In this experiment, we use SmallFry in (May et al., 2019), which is a quantization method for word embedding. For each word group G_i, we apply SmallFry to it with assigning the number of bits depending on the average word importance. The number of bits q^* is defined as:

$$q^* = \min\left\{q, \max\left\{1, 2^{\lfloor \log_2 \omega \frac{s}{s_{\max}} q \rfloor}\right\}\right\}, \quad (4)$$

where q is a user-specific parameter for the number of bits, s is the average score of G_i, and $s_{\max}$ is the maximum average score over groups in $\mathcal{G}$. For simplicity, ω and q are set to 1 and 2, respectively.

The result is shown in Table 7 where BlockFry is a method which partitions word groups with a word importance and applies SmallFry to sub-embedding matrices induced by the groups. In the result, BlockFry is more effective than SmallFry in many cases. Especially, in terms of model performance before retraining, the gap between them is considerable.

Table 7: Cooperation with SmallFry (Quantization)

Methods	PTB	SQuAD	SST-5	IWSLT14
Target Ratio	$50\times$	$43\times$	$42\times$	$51\times$
SmallFry	307.9	72.1	41.0	0.5
BlockFry-F	188.9	**72.6**	42.3	25.9
BlockFry-T	159.2	71.0	**42.7**	**26.7**
BlockFry-D	**138.0**	71.8	42.1	23.1
SmallFry (Retrain)	93.1	**74.0**	44.0	**30.4**
BlockFry-F (Retrain)	95.3	73.8	44.4	30.2
BlockFry-T (Retrain)	94.0	73.9	43.7	30.3
BlockFry-D (Retrain)	**90.7**	73.8	**44.5**	30.3

6 Conclusions

The block-wise low-rank approximation of (Chen et al., 2018) is an effective method for word embedding compression. However, its word weighting and partitioning scheme is somewhat simple and there is big room for improvement from it. Motivated by this, we propose a discriminative block-wise word embedding compression algorithm, named *DiscBlock*, based on the two effective word weighting methods and the k-means clustering method. The experimental results show that *DiscBlock* significantly outperforms the competitors including *GroupReduce* in terms of accuracy loss in most cases. In addition, we explore the limitation of *GroupReduce* in terms of compression ratio due to the uniform partition construction. Finally, as a compression framework, we show that *DisckBlock* can cooperate with another compression method to achieve better compression performance than it can achieve alone.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2021-0-00907, Development of Adaptive and Lightweight Edge-Collaborative Analysis Technology for Enabling Proactively Immediate Response and Rapid Learning).

References

Martin Andrews. 2016. Compressing word embeddings. In *Neural Information Processing*, pages 413–422.

Marcella Astrid and Seung-Ik Lee. 2018. Deep compression of convolutional neural networks with low-rank approximation. *ETRI Journal*, 40(4):421–434.

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)*. Association for Computational Linguistics.

M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and Marcello Federico. 2015. Report on the 11 th iwslt evaluation campaign , iwslt 2014.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879.

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. 2018. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In *Advances in Neural Information Processing Systems 31*, pages 10988–10998. Curran Associates, Inc.

Lieven De Lathauwer. 2008. Decompositions of a higher-order tensor in block terms—part ii: Definitions and uniqueness. *SIAM J. Matrix Anal. Appl.*, 30(3):1033–1066.

Oleksii Hrinchuk, Valentin Khrulkov, Leyla Mirvakhabova, Elena Orlova, and Ivan Oseledets. 2020. Tensorized embedding layers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4847–4860.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.*, 18(1):6869–6898.

Jaedeok Kim, Chiyoun Park, Hyun-Joo Jung, and Yoonsuck Choe. 2020a. Plug-in, trainable gate for streamlining arbitrary neural networks. In *AAAI*, pages 4452–4459.

Yeachan Kim, Kang-Min Kim, and SangKeun Lee. 2020b. Adaptive compression of word embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3950–3959.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

Julia Kreutzer, Jasmijn Bastings, and Stefan Riezler. 2019. Joey NMT: A minimalist NMT toolkit for novices. 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): System Demonstrations*, pages 109–114.

Shaoshi Ling, Yangqiu Song, and Dan Roth. 2016. Word embeddings with limited memory. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 387–392.

Ahmad andKumar Krtin andHaidar Md. Akmal andRezagholizadeh Mehdi Lioutas, Vasileios andRashid. 2020. Improving Word Embedding Factorization for Compression Using Distilled Nonlinear Neural Decomposition. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2774–2784.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. 2019. A tensorized transformer for language modeling. In *Advances in Neural Information Processing Systems 32*, pages 2232–2242.

Avner May, Jian Zhang, Tri Dao, and Christopher Ré. 2019. On the downstream performance of compressed word embeddings. In *Advances in Neural Information Processing Systems 32*, pages 11805–11816.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Aliakbar Panahi, Seyran Saeedi, and Tom Arodz. 2020. word2ket: Space-efficient word embeddings inspired by quantum entanglement. In *International Conference on Learning Representations*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Kaiyu Shi and Kai Yu. 2018. Structured word embedding for low memory neural network language model. In *Proc. Interspeech 2018*, pages 1254–1258.

Raphael Shu and Hideki Nakayama. 2018. Compressing word embeddings via deep compositional code learning. In *ICLR*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008.

X. Yu, T. Liu, X. Wang, and D. Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 67–76.