Further Compressing Distilled Language Models via Frequency-aware Partial Sparse Coding of Embeddings

Kohki Tamura[†] **Naoki Yoshianga**[‡] **Masato Neishi**[†*]
[†]The University of Tokyo [‡]Institute of Industrial Science, The University of Tokyo
[†]{tamura-k,neishi}@tkl.iis.u-tokyo.ac.jp [‡]ynaga@iis.u-tokyo.ac.jp

Abstract

Although pre-trained language models (PLMs) are effective for natural language understanding (NLU) tasks, they demand a huge computational resource, thus preventing us from deploying them on edge devices. Researchers have therefore applied compression techniques for neural networks, such as pruning, quantization, and knowledge distillation, to the PLMs. Although these generic techniques can reduce the number of internal parameters of hidden layers in the PLMs, the embedding layers tied to the tokenizer are hard to be compressed, occupying a non-negligible portion of the compressed model. In this study, aiming to further compress PLMs reduced by the generic techniques, we exploit frequency-aware sparse coding to compress the embedding layers of the PLMs fine-tuned to downstream tasks. To minimize the impact of the compression on the accuracy, we retain the embeddings of common tokens as they are and use them to reconstruct embeddings of rare tokens by locally linear mapping. Experimental results on the GLUE and JGLUE benchmarks for language understanding in English and Japanese confirmed that our method can further compress the fine-tuned DistilBERT models while maintaining accuracy.

1 Introduction

Transformer (Vaswani et al., 2017)-based language models (LMs) have been extensively used to solve natural language processing (NLP) tasks via pre-train and fine-tuning (Devlin et al., 2019); the accuracy of the fine-tuned LMs can be improved by scaling up the model and pre-training data sizes (Kaplan et al., 2020). Pre-trained LMs (PLMs) thereby became larger and larger, which prevents us from deploying them on resource-constrained environments. Thus, we cannot leverage powerful PLMs to text with privacy concerns in end-user devices or confidential documents in small businesses.

[*]Currently, he works for Mirai Translate, Inc.

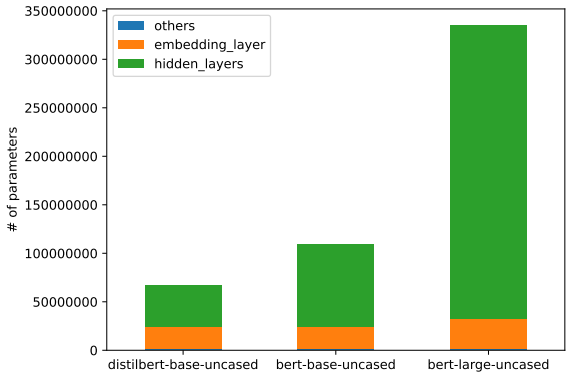

Figure 1: The number of parameters in BERT variants; these PLMs have similar numbers of parameters in the embedding layers, which become more dominant (34.9%) in `distilbert-base-uncased`.

To make PLMs faster and smaller while maintaining the accuracy, researchers have utilized common compression techniques for neural networks (surveyed in Zhu et al. (2023)); the techniques include pruning, quantization, and knowledge distillation, mainly focusing on compressing hidden layers which occupy the largest part in the PLMs with deep Transformer layers (Wan et al., 2024; Zhou et al., 2024). In the distilled PLMs, however, parameters in those other than hidden layers account for a large proportion of the entire parameters (Figure 1), and most of them are accounted for by the embedding layers. For instance, the parameters of the embedding layer account for about 34.9% of `distilbert-base-uncased`[1] (Sanh et al., 2019), whereas they occupy about 21.4% of the original 12-layer `bert-base-uncased`[2] (Devlin et al., 2019). Therefore, we subject the embedding layers to further compression.

In this study, given a PLM fine-tuned to the target downstream task, we propose to compress the

[1]https://huggingface.co/distilbert/distilbert-base-uncased

[2]https://huggingface.co/google-bert/bert-base-uncased

Proceedings of the 28th Conference on Computational Natural Language Learning, pages 388–399

embedding layer of the PLM by using sparse coding of embeddings, which represents embeddings with a sparse linear combination of basis embeddings (Faruqui et al., 2015). The issue here is that the sparse coding introduces approximation errors, or noises, into the fine-tuned embeddings. To reduce the impact of these noises on the PLM's outputs, we perform a frequency-aware partial sparse coding of embeddings; namely, we regard a small number of common token embeddings as basis embeddings to reconstruct the remaining rare token embeddings, as employed in Chen et al. (2016) for recurrent neural network LMs.

Since the embeddings of the recent PLMs will be contextualized through deep Transformer layers and noisy rare token embeddings will be supplemented by intact embeddings of surrounding common tokens, we adopt simple locally-linear embeddings (Roweis and Saul, 2000; Sakuma and Yoshinaga, 2019) to choose a few basis (common token) embeddings for each rare token embedding, thereby enabling sparser coding of embeddings. Each rare token embedding is thereby represented as a weighted linear sum of the nearest neighbor common token embeddings. Finally, we save the weight and the IDs of the common tokens to dynamically reconstruct embeddings during inference.

We applied our method to English and Japanese DistilBERT models fine-tuned to GLUE (Wang et al., 2018) and JGLUE datasets (Kurihara et al., 2022), respectively. We then compared our methods with three baselines; the two of them approximate the same rare token embeddings as our method, by <unk> token embedding in the target PLM and by common basis embeddings induced by principal component analysis, respectively. The other approximates the entire embedding layers using sparse vectors to select vectors to sum up from shared chunks of vectors (additive quantization).

The contributions of this paper are as follows:

- We present a simple, frequency-aware partial sparse coding to compress embedding layers in the PLMs fine-tuned to downstream tasks.
- We confirmed an advantage of our method on distilled LMs in two languages, fine-tuned to various natural language understanding tasks.
- We confirmed the robustness of our frequency-aware sparse coding of embeddings in that the PLM retains the original accuracy even when the reconstruction introduces noises.

2 Proposed Method

The major difficulty in compressing the embeddings of a fine-tuned Transformer-based PLM is that if the compression introduces some approximation (noises) in the embeddings, they will severely affect the latter processing in the deep Transformer layers. Fukuda et al. (2020) confirmed on sentiment classification that the accuracy of BERT decreased greatly (>10%) when one or more words take perturbations mimicking typos.

Motivated by this observation, we adopt partial sparse coding, which reconstructs only a subset of PLM's embeddings whose approximation errors (noises) will not severely affect the PLM's behavior. To reduce the memory footprint, we divert common token embeddings to the candidate of basis embeddings that represent the rare token embeddings.

Our partial sparse coding consists of the following two steps:

Step 1: Splitting vocabularies. We first split the vocabulary of the PLM, $\mathcal{V}$, into two portions, $\mathcal{V}_C$ (**source vocabularies**) and $\mathcal{V}_R = \mathcal{V} - \mathcal{V}_C$ (**target vocabularies**), in which the embeddings of the source vocabularies (**source embeddings** are used as basis embeddings in sparse coding to approximate the embeddings of the target vocabularies (**target embeddings**).

Step 2: Reconstructing target embeddings. We then compute compact representations for the target embeddings, $\mathcal{Y} = \{\boldsymbol{y}_i \in \mathbb{R}^d\}_{i=1}^{|\mathcal{V}_R|}$ (d is the number of embedding dimensions), by approximating them as a weighted linear sum of the source embeddings, $\mathcal{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^{|\mathcal{V}_C|}$. To facilitate the compression, we choose a small subset of size k, $\mathcal{N}_i$, among the source embeddings $\mathcal{X}$ to approximate each target embedding $\boldsymbol{y}_i \in \mathcal{Y}$. We then represent target embedding $\boldsymbol{y}_i$ by compact (sparse vector) representations, $\hat{\boldsymbol{y}}_i = \{(j, \alpha_{ij}) \,|\, \boldsymbol{x_j} \in \mathcal{N}_i\}$, namely, k pairs of embedding ID $j \in \mathcal{N}_i$ and the weight $\alpha_{ij} \in \mathbb{R}$ for the linear summation.

We then reduce the target embeddings $\mathcal{Y}$ by replacing them with their sparse vector representations and dynamically reconstruct the target embedding $\boldsymbol{y}_i$ during the inference by referring to $\hat{\boldsymbol{y}}_i$, thus obtaining a model with a smaller embedding layer. The resulting embedding layer is composed of the original parameters (embeddings) $\mathcal{X}$ for the source

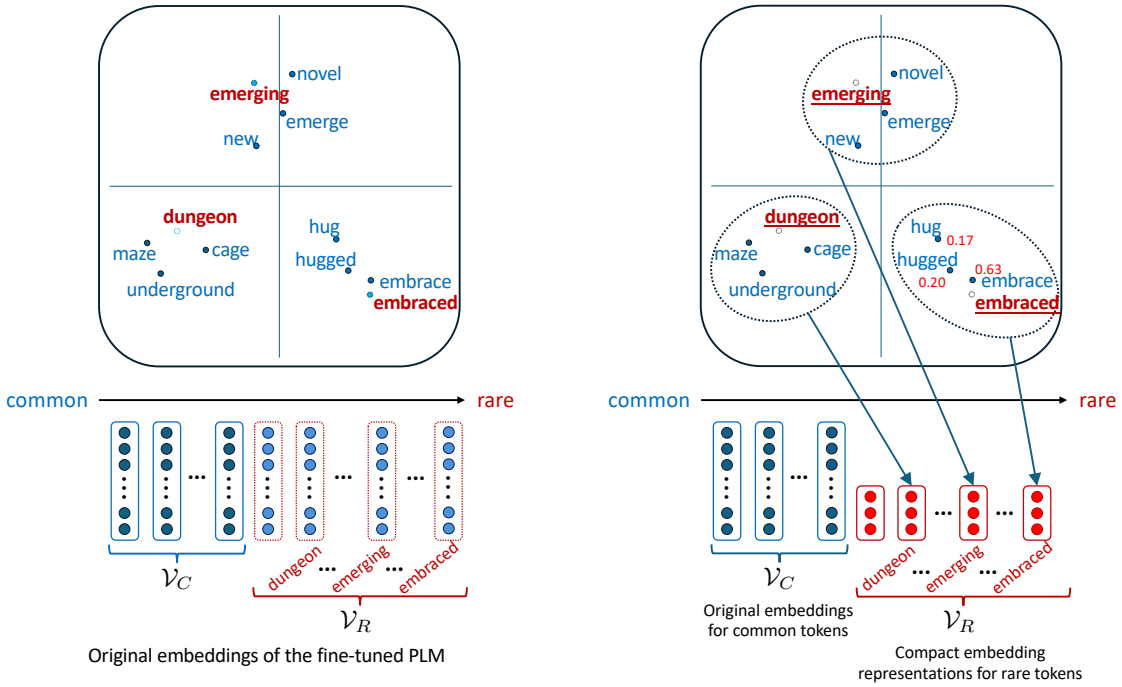

Figure 2: An overview of our frequency-aware partial sparse coding of embeddings. We represent embeddings of rare tokens (*e.g.*, "hugged") with their nearest neighbor embeddings of common tokens.

embeddings and the compact representations for the target embeddings $\mathcal{Y}$. The modified model has $(d - 2k)|\mathcal{V}_R|$ fewer parameters,[3] which greatly reduces the number of parameters in the embedding layer when $k \ll d$ and $|\mathcal{V}_C| \ll |\mathcal{V}_R|$.

2.1 Step 1: Splitting vocabularies

To retain the inference accuracy of the fine-tuned PLMs, we need an effective criterion to split the LM's vocabulary into the source, basis embeddings, and the target embeddings for reconstruction. We thus leverage the frequency of tokens in the training data of the downstream task which are used to fine-tune the target PLM. Specifically, we count the frequency, f_i, for each token, $t_i \in \mathcal{V}$ in the training data which is tokenized with the target PLM's tokenizer. We set the top-n common tokens in $\mathcal{V}$ as $\mathcal{V}_C$ and the others as $\mathcal{V}_R$.

We should mention that a similar approach has been explored by Chen et al. (2016) to compress word embedding layers of recurrent neural network (RNN) LMs. The essential difference is that our method *explicitly* narrows down the candidate basis (common token) embeddings to reconstruct each rare token embedding, whereas Chen et al. (2016) used ℓ_1-regularization in learning a weight matrix for linear combinations to promote the sparseness of the weights implicitly, as described in § 2.2.

[3]Our method requires slightly more parameters (§ 2.2).

2.2 Step 2: Reconstructing target embeddings

To obtain the compact representations of the target embeddings, we want to use only a small subset of size k of the source embeddings to approximate the target embeddings. Because we want to explicitly control the required memory footprint and the PLM has a strong contextualization ability based on the surrounding intact embeddings for common tokens, we adopt a simple method of locally linear mapping (Roweis and Saul, 2000; Sakuma and Yoshinaga, 2019), which selects for each target embedding k nearest neighbor source embeddings for approximation.

In the original locally linear mapping for task-specific multilingual models (Sakuma and Yoshinaga, 2019), the authors first represent target embeddings (*e.g.*, Japanese word embeddings) with a weighted linear sum of top-k nearest neighbor source embeddings (*e.g.*, English word embeddings) in one semantic space (*e.g.*, the semantic space of the PLM), and use these weights to reconstruct target embeddings in another semantic space (*e.g.*, the semantic space of the fine-tuned PLM) to realize a task-specific multilingual model. In our setting, however, since the target semantic space (here, the semantic space of the fine-tuned PLM) also has the target embeddings for the target tokens, in contrast to Sakuma and Yoshinaga (2019), we

do not need to consider two semantic spaces and can compute a linear weighted sum in the semantic space of the fine-tuned PLMs.[4]

In this study, we add a small fix to locally linear mapping to use normalized embeddings instead of raw embeddings, and force estimated embeddings to have the same length as the original just fine-tuned one. First, we normalize $\mathcal{Y}$ and $\mathcal{X}$ to make all embeddings $\boldsymbol{e}$ to be $\|\boldsymbol{e}\| = 1$, and obtain the normalized embeddings as $\mathcal{X}^{\mathrm{n}}$ and $\mathcal{Y}^{\mathrm{n}}$. We set $\mathcal{N}_i$ as tokens with the top k nearest neighbor embeddings in $\mathcal{X}^{\mathrm{n}}$ from each embedding $\boldsymbol{y}_i^{\mathrm{n}}$ in $\mathcal{V}_R$ with cosine similarity, and compute the weights $\boldsymbol{\alpha}_i$ to estimate each $\hat{\boldsymbol{y}}_i^{\mathrm{n}}$ by $\sum_{j \in \mathcal{N}_i} \hat{\alpha}_{ij} \boldsymbol{x}_j^{\mathrm{n}}$. With locally linear mapping, we compute $\hat{\boldsymbol{\alpha}}_i$ which approximate $\boldsymbol{y}_i^{\mathrm{n}}$ the most by weighted linear sum of $\boldsymbol{x}_j^{\mathrm{n}}$ represented as

$$\hat{\boldsymbol{\alpha}}_i = \arg\min_{\boldsymbol{\alpha}_i} \left\| \boldsymbol{y}_i^{\mathrm{n}} - \sum_{j \in \mathcal{N}_i} \alpha_{ij} \boldsymbol{x}_j^{\mathrm{n}} \right\|^2 \quad (1)$$

using Lagrange multiplier from $\boldsymbol{x}_i^{\mathrm{n}}$, $\boldsymbol{y}_i^{\mathrm{n}}$ and a constraint of $\sum_j \alpha_{ij} = 1$ by compute

$$\hat{\alpha}_{ij} = \frac{\sum_l (C_i^{-1})_{jl}}{\sum_j \sum_l (C_i^{-1})_{jl}} \quad (2)$$

under $C_{ijl} = (\boldsymbol{y}_i^{\mathrm{n}} - \boldsymbol{x}_j^{\mathrm{n}}) \cdot (\boldsymbol{y}_i^{\mathrm{n}} - \boldsymbol{x}_l^{\mathrm{n}})$ to estimate whole $\hat{\mathcal{Y}}^{\mathrm{n}}$ ($l \in \mathcal{N}_i$). We finally estimate each $\hat{\boldsymbol{y}}_i$ by adjusting the length of $\hat{\boldsymbol{y}}_i^{\mathrm{n}}$ same as $\boldsymbol{y}_i$ with

$$\hat{\boldsymbol{y}}_i = \|\boldsymbol{y}_i\| \frac{\hat{\boldsymbol{y}}_i^{\mathrm{n}}}{\|\hat{\boldsymbol{y}}_i^{\mathrm{n}}\|} \quad (3)$$

We save necessary parameters to reconstruct $\hat{\boldsymbol{y}}_i$ instead of the original embedding $\boldsymbol{y}_i$. To reconstruct $\hat{\boldsymbol{y}}_i$, we need ID of embeddings in $\mathcal{N}_i$, weights $\boldsymbol{\alpha}_i$ and the length of embedding $\|\boldsymbol{y}_i\|$. We save these $2k + 1$ parameters for every token in $\mathcal{V}_R$ as our compact representation, thus we reduce $(d - (2k + 1))|\mathcal{V}_R|$ parameters. In the inference, we dynamically reconstruct $\hat{\boldsymbol{y}}_i$ upon request when the tokenizer outputs those tokens.

3 Experimental Setup

We evaluate our frequency-aware sparse coding of embeddings on distilled PLMs fine-tuned to NLU tasks in terms of the model size and performance.

[4]Because the target rare token embeddings may not be updated for the target task and are in the semantic space of PLM instead of the fine-tuned PLM), we may be able to obtain better embeddings by computing weights for the summation in the semantic space of the PLM and by using the weights to reconstruct the target embeddings in the semantic space of the fine-tuned PLM. However, our preliminary experiments revealed that the fine-tuning did not change the embeddings much, this did not contribute to the accuracy improvements.

3.1 Datasets

For evaluation, we adopt GLUE (Wang et al., 2018) and JGLUE benchmark (Kurihara et al., 2022) for language understanding tasks in English and Japanese, respectively.

GLUE is a benchmark consisting of nine natural language understanding (NLU) tasks. It contains datasets of acceptability (CoLA), sentiment analysis (SST-2), paraphrase (MRPC, QQP), textual similarity (STS-B), and natural language inference (NLI; MNLI, QNLI, RTE, and WNLI). The sizes of the datasets range from <1k to over 500k. In the experiments, we adopted the common metrics used in the evaluation of BERT (Devlin et al., 2019) and DistilBERT (Sanh et al., 2019); F_1 for MRPC and QQP, Spearman Correlation for STS-B, and accuracy for the others.

Since we experimented on the diverse settings of k and $\mathcal{V}_R$ resulting in plenty of results, it was not possible to upload all of our results to test on the website[5] because of its limitation of submission. Hence, for every task, we used the original validation set as the test set. Instead of the original validation set, we split the train set into 90% and 10% shuffling randomly using a fixed random seed 42 and treated the latter as a validation set. We did not evaluate our method on QQP and WNLI since these tasks have different label distributions between the validation set and the test set, which means that the results of experiments on these datasets may be misleading.[6] In addition, we did not conduct experiments on MNLI due to the computational cost of running experiments on this large dataset.

JGLUE is a benchmark consisting of seven NLU tasks in Japanese. It contains datasets of text classification (MARC-ja and JCoLA), sentence pair classification (JSTS and JNLI), and QA (JSQuAD and JCommonsenseQA). Because the MARC-ja dataset is no longer available at this time, we evaluated our method on the other tasks. We used only Spearman's Correlation for the evaluation of JSTS following STS-B, and accuracy on the other tasks, following Kurihara et al. (2022).

Since the test sets of JGLUE have not been released yet, we employed the same process as we did for GLUE, except for JCoLA (we used the "`validation_out_of_domain`" subset as the test data). We experimented on JCoLA, JSTS, and JNLI datasets of the benchmark.

[5]https://gluebenchmark.com

[6]https://gluebenchmark.com/faq

3.2 PLMs for embedding compression

We applied our method to fine-tuned DistilBERT models whose hidden layer of BERT is compressed by knowledge distillation. Specifically, we experimented on `distilbert-base-uncased`[7] for English and `line-distilbert-base-japanese`[8] for Japanese. In what follows, we report the averages and standard deviations of three fine-tuning trials.

The English PLM has 23M parameters in the embedding layer, which consist of 30,522 token embeddings of 768 dimensions and account for 34.9% of the parameters (67M in total). This PLM employs WordPiece as the tokenizer. In fine-tuning, we trained for three epochs with a learning rate of 2e-5, except for five epochs on MRPC.

The Japanese PLM has 25M parameters of the embedding layer, which consist of 32,768 token embeddings of 768 dimensions and account for 36.6% of the parameters (68M in total). The tokenization of this model is done in two stages; pre-tokenization by MeCab[9] (unidic-lite) and tokenization by unigram LM of SentencePiece (Kudo and Richardson, 2018). In fine-tuning, we trained for four epochs with a learning rate of 5e-5.

3.3 Embedding compression

Threshold for common tokens We initially treat all PLM vocabularies that appear during the fine-tuning as $\mathcal{V}_C$ and the others as $\mathcal{V}_R$. Then, we transfer common tokens in $\mathcal{V}_C$ to $\mathcal{V}_R$ to see the trade-off between the compression rate and the performance. In these settings, the top 50% to 90% of the tokens with the higher frequency remain as $\mathcal{V}_C$, and the others are transferred to the $\mathcal{V}_R$. In the experiments, we compare our method while varying the retention rate of $\mathcal{V}_C$, $\mathrm{r}(\mathcal{V}_C)$, for each task; for example, $\mathrm{r}(\mathcal{V}_C) = 1.0$ means all the tokens that appeared during the fine-tuning are kept in $\mathcal{V}_C$ and $\mathrm{r}(\mathcal{V}_C) = 0.5$ means the half of the tokens that appeared during the fine-tuning are transferred to $\mathcal{V}_R$.

The number of the source embeddings We also compare our method while varying k, the number of the source embeddings used to represent each target embedding, ranging from one to five for each task. We tune k to minimize the inference error on the validation set and report the results of the best-performing k. We will later confirm that the choice of k does not affect the PLM performance, thanks to its strong contextualization capabilities.

3.4 Baselines

We compare our model with three baselines: i) replacing $\mathcal{V}_R$ with **<unk>** token learned by the PLM, ii) Principal Component Analysis (PCA)-based approximation and iii) Additive Quantization.

"unknown" token (<unk>) replaces all of the target tokens in $\mathcal{V}_R$ with a special token <unk> to leverage the unknown token embedding learned by the PLM.

Principal Component Analysis (PCA) uses the bases of the embedding space obtained by PCA as the source embeddings, instead of $\mathcal{V}_C$, to reconstruct $\mathcal{V}_R$. We compute the coordinate in k-dimensional space with this basis for each target token, and we treat the coordinate as the weight like our method. We save the same number of the source embedding, k, to reconstruct $\mathcal{V}_R$, and the coordinate of k dimension for each token in $\mathcal{V}_R$ instead of the original embeddings. We show the results from a single k selected with the same criteria as the proposed method, while ranging k from one to ten.

Additive Quantization (AQ) represents the original embeddings with the sum of basis embeddings which are shared across the target tokens with similar meanings (Babenko and Lempitsky, 2014; Shu and Nakayama, 2017). Although AQ reconstructs the original embeddings by a sum of a small subset of the basis embeddings as in our method, it is designed to reconstruct all embeddings using the independently-learned basis embeddings. We have used the official implementation of Shu's method (Shu and Nakayama, 2017)[10] with hyperparameters of $K = 16$ and $M = 32$.

The former two baselines approximate the same $\mathcal{V}_C$ as ours to see the effectiveness of choosing the source embeddings from the nearest neighbors, while the last baseline compresses the entire set of embeddings to see the impact of frequency-aware partial sparse coding.

[7] https://huggingface.co/distilbert/distilbert-base-uncased

[8] https://huggingface.co/line-corporation/line-distilbert-base-japanese

[9] https://taku910.github.io/mecab/

[10] https://github.com/zomux/neuralcompressor

$\lvert\mathcal{V}_C\rvert$	CoLA 5585	SST-2 11570	MRPC 11561	STS-B 10794	QNLI 26180	RTE 13863	Average
$\mathrm{r}(\mathcal{V}_C) = 1.0$							
<unk>	$37.16_{\pm 1.12}$	$\mathbf{90.18}_{\pm 0.23}$	$\mathbf{87.95}_{\pm 0.39}$	$83.67_{\pm 0.15}$	$86.96_{\pm 0.10}$	$57.76_{\pm 1.42}$	73.95
PCA	$41.79_{\pm 0.48}{}^{k=2}$	$89.72_{\pm 0.05}{}^{k=7}$	$87.72_{\pm 0.54}{}^{k=1}$	$81.08_{\pm 0.41}{}^{k=1}$	$86.97_{\pm 0.09}{}^{k=1}$	$53.19_{\pm 2.20}{}^{k=2}$	73.41
proposed	$\mathbf{41.91}_{\pm 0.20}{}^{k=5}$	$89.87_{\pm 0.40}{}^{k=4}$	$87.75_{\pm 0.47}{}^{k=3}$	$\mathbf{85.45}_{\pm 0.14}{}^{k=1}$	$\mathbf{86.99}_{\pm 0.09}{}^{k=2}$	$\mathbf{57.88}_{\pm 0.53}{}^{k=1}$	**74.98**
$\mathrm{r}(\mathcal{V}_C) = 0.5$							
<unk>	$29.12_{\pm 1.05}$	$89.56_{\pm 0.14}$	$\mathbf{88.18}_{\pm 0.40}$	$81.54_{\pm 0.21}$	$86.16_{\pm 0.20}$	$54.99_{\pm 1.88}$	71.59
PCA	$\mathbf{33.29}_{\pm 0.27}{}^{k=2}$	$89.30_{\pm 0.20}{}^{k=10}$	$86.29_{\pm 0.44}{}^{k=1}$	$75.23_{\pm 0.91}{}^{k=10}$	$83.85_{\pm 0.32}{}^{k=1}$	$53.79_{\pm 1.55}{}^{k=10}$	70.29
proposed	$32.33_{\pm 0.62}{}^{k=5}$	$\mathbf{90.18}_{\pm 0.38}{}^{k=5}$	$87.25_{\pm 0.43}{}^{k=4}$	$\mathbf{82.67}_{\pm 0.23}{}^{k=3}$	$\mathbf{86.19}_{\pm 0.05}{}^{k=1}$	$\mathbf{56.92}_{\pm 0.15}{}^{k=1}$	**72.59**
original	$48.74_{\pm 0.38}$	$90.02_{\pm 0.35}$	$88.75_{\pm 0.12}$	$85.77_{\pm 0.12}$	$87.06_{\pm 0.12}$	$57.40_{\pm 1.84}$	76.29

Table 1: The results of **GLUE** benchmark. The numbers in brackets show the number of the source embeddings, k, chosen by using the validation set.

$\lvert\mathcal{V}_C\rvert$	JCoLA 3558	JSTS 4576	JNLI 4403	Average
$\mathrm{r}(\mathcal{V}_C) = 1.0$				
<unk>	$74.60_{\pm 0.58}$	$84.70_{\pm 0.09}$	$87.69_{\pm 0.36}$	82.33
PCA	$75.33_{\pm 0.00}{}^{k=1}$	$\mathbf{84.73}_{\pm 0.12}{}^{k=8}$	$87.83_{\pm 0.28}{}^{k=2}$	82.63
proposed	$\mathbf{76.50}_{\pm 0.10}{}^{k=2}$	$84.65_{\pm 0.11}{}^{k=1}$	$\mathbf{87.85}_{\pm 0.24}{}^{k=2}$	**83.00**
$\mathrm{r}(\mathcal{V}_C) = 0.5$				
<unk>	$70.61_{\pm 1.55}$	$83.55_{\pm 0.03}$	$86.72_{\pm 0.19}$	80.29
PCA	$\mathbf{75.67}_{\pm 0.49}{}^{k=1}$	$83.46_{\pm 0.13}{}^{k=10}$	$86.52_{\pm 0.25}{}^{k=1}$	81.88
proposed	$\mathbf{75.67}_{\pm 0.47}{}^{k=5}$	$\mathbf{84.30}_{\pm 0.12}{}^{k=3}$	$\mathbf{87.47}_{\pm 0.08}{}^{k=1}$	**82.48**
AQ	$28.81_{\pm 1.48}$	$46.95_{\pm 11.21}$	$-2.34_{\pm 1.14}$	24.47
original	$77.08_{\pm 0.31}$	$84.67_{\pm 0.10}$	$87.96_{\pm 0.29}$	83.24

Table 2: The results of **JGLUE** benchmark. The numbers in brackets show the number of the source embeddings, k, chosen by using the validation set.

4 Results

4.1 Main results

We first compared the results of the proposed method and the three baselines. <unk> and PCA baselines approximate the same target embeddings as ours, under the settings of $\mathrm{r}(\mathcal{V}_C) = 1.0$ and $\mathrm{r}(\mathcal{V}_C) = 0.5$. We also compared to AQ in JGLUE, it approximates the entire embeddings regardless of $\mathrm{r}(\mathcal{V}_C)$.

Tables 1 and 2 show the results on the GLUE and JGLUE benchmark datasets, respectively. From the results, we can observe that our method outperforms the baselines on average and exhibits stable performance across tasks. Our method outperforms the PCA baseline in all tasks except for CoLA of $\mathrm{r}(\mathcal{V}_C) = 0.5$ and JSTS of $\mathrm{r}(\mathcal{V}_C) = 1.0$, thus confirming the importance of target-dependent source (basis) embeddings. Meanwhile, our method slightly underperforms the <unk> baseline in SST-2, MRPC, and JSTS of $\mathrm{r}(\mathcal{V}_C) = 1.0$, and MRPC of $\mathrm{r}(\mathcal{V}_C) = 0.5$. However, the token and sentence coverage by only common tokens are higher in those datasets as we will later confirm in Tables 5 and 6; all the three frequency-aware methods exhibit similar performance to the original model even under $\mathrm{r}(\mathcal{V}_C) = 0.5$. Overall, our method mitigates performance degradation compared to only replacing such tokens with <unk> tokens, especially for $\mathrm{r}(\mathcal{V}_C) = 0.5$ in both languages.

The relationship between performance and $\lvert\mathcal{V}_C\rvert$ Tables 3 and 4 show the results of our method while varying $\mathrm{r}(\mathcal{V}_C)$. From the tables, there is a weak tendency that setting a lower value to $\mathrm{r}(\mathcal{V}_C)$ results in lower performance. Thus, rare tokens weakly affect the PLM's performance, and it is reasonable to compress only rare token embeddings while keeping the original common token embeddings.

We compare the token and sentence coverage by $\mathcal{V}_C$ in the following three GLUE datasets: CoLA, which has the largest performance drop at small $\mathrm{r}(\mathcal{V}_C)$, MRPC, which has a small performance drop despite being the similar training data size to CoLA, and QNLI, which has a small performance drop because more tokens are preserved (§ 4.2) at

$r(\mathcal{V}_C)$	CoLA	SST-2	MRPC	STS-B	QNLI	RTE
0.5	$32.33_{\pm 0.62}{}^{k=5}$	$\mathbf{90.18}_{\pm 0.38}{}^{k=5}$	$87.25_{\pm 0.43}{}^{k=4}$	$82.67_{\pm 0.23}{}^{k=3}$	$86.19_{\pm 0.05}{}^{k=1}$	$56.92_{\pm 0.15}{}^{k=1}$
0.6	$35.27_{\pm 0.32}{}^{k=5}$	$89.68_{\pm 0.29}{}^{k=5}$	$87.80_{\pm 0.42}{}^{k=3}$	$83.80_{\pm 0.17}{}^{k=2}$	$86.27_{\pm 0.12}{}^{k=1}$	$56.80_{\pm 0.30}{}^{k=1}$
0.7	$37.50_{\pm 0.12}{}^{k=4}$	$89.60_{\pm 0.25}{}^{k=5}$	$88.09_{\pm 0.45}{}^{k=5}$	$84.03_{\pm 0.16}{}^{k=2}$	$86.63_{\pm 0.10}{}^{k=1}$	$57.04_{\pm 0.26}{}^{k=1}$
0.8	$39.56_{\pm 0.15}{}^{k=5}$	$89.99_{\pm 0.26}{}^{k=5}$	$\mathbf{88.55}_{\pm 0.91}{}^{k=4}$	$84.29_{\pm 0.15}{}^{k=3}$	$86.86_{\pm 0.21}{}^{k=1}$	$57.76_{\pm 0.26}{}^{k=1}$
0.9	$39.72_{\pm 0.44}{}^{k=5}$	$90.02_{\pm 0.37}{}^{k=3}$	$88.10_{\pm 0.84}{}^{k=5}$	$85.01_{\pm 0.11}{}^{k=1}$	$\mathbf{87.02}_{\pm 0.09}{}^{k=1}$	$\mathbf{58.24}_{\pm 0.39}{}^{k=1}$
1.0	$\mathbf{41.91}_{\pm 0.20}{}^{k=5}$	$89.87_{\pm 0.40}{}^{k=4}$	$87.75_{\pm 0.47}{}^{k=3}$	$\mathbf{85.45}_{\pm 0.14}{}^{k=1}$	$86.99_{\pm 0.09}{}^{k=2}$	$57.88_{\pm 0.53}{}^{k=1}$
original	$48.74_{\pm 0.38}$	$90.02_{\pm 0.35}$	$88.75_{\pm 0.12}$	$85.77_{\pm 0.12}$	$87.06_{\pm 0.12}$	$57.40_{\pm 1.84}$

Table 3: The results of modified models with our method under different $r(\mathcal{V}_C)$ in the **GLUE** benchmark. The numbers in brackets show the number of the source embeddings, k, chosen by using the validation set.

$r(\mathcal{V}_C)$	JCoLA	JSTS	JNLI
0.5	$75.67_{\pm 0.47}$	$84.30_{\pm 0.12}$	$87.47_{\pm 0.08}$
0.6	$76.79_{\pm 0.36}$	$84.50_{\pm 0.14}$	$87.55_{\pm 0.23}$
0.7	$76.93_{\pm 0.10}$	$84.53_{\pm 0.14}$	$87.57_{\pm 0.31}$
0.8	$76.45_{\pm 0.47}$	$\mathbf{84.66}_{\pm 0.14}$	$87.80_{\pm 0.18}$
0.9	$\mathbf{77.13}_{\pm 0.33}$	$\mathbf{84.65}_{\pm 0.13}$	$87.76_{\pm 0.21}$
1.0	$76.50_{\pm 0.10}$	$\mathbf{84.65}_{\pm 0.11}$	$\mathbf{87.85}_{\pm 0.24}$
original	$77.08_{\pm 0.31}$	$84.67_{\pm 0.10}$	$87.96_{\pm 0.29}$

Table 4: The results of modified models with our method under different $r(\mathcal{V}_C)$ in the **JGLUE** benchmark.

$r(\mathcal{V}_C)$	CoLA	MRPC	QNLI
0.5	99.37	96.00	98.27
0.6	99.47	96.66	98.72
0.7	99.55	97.06	99.14
0.8	99.60	97.52	99.52
0.9	99.64	97.92	99.74
1.0	99.69	98.30	99.90

Table 5: Token coverage in the **GLUE** test data by tokens in $\mathcal{V}_C$.

$r(\mathcal{V}_C)$	CoLA	MRPC	QNLI
0.5	56.66	9.31	22.84
0.6	61.38	15.44	33.15
0.7	66.35	18.63	47.34
0.8	68.65	23.53	64.67
0.9	72.00	28.43	79.15
1.0	75.17	34.07	91.10

Table 6: Sentence coverage in the **GLUE** test data only by tokens in $\mathcal{V}_C$. In covered sentences, the model performs exactly the same as the original model.

$r(\mathcal{V}_C)$	GLUE				JGLUE
	CoLA	MRPC	STS-B	QNLI	JSTS
0.5	10.15	19.05	17.71	43.13	10.17
1.0	18.85	36.76	34.16	85.88	16.65

Table 7: The rate of parameters (%) that our method requires compared to the original embedding layer. We also list the result of CoLA and MRPC, for the analysis related to Tables 5 and 6.

$r(\mathcal{V}_C) = 0.5$.

Tables 5 and 6 show the token and sentence coverage by $\mathcal{V}_C$ for these three characteristic datasets, respectively. From the results, $\mathcal{V}_C$ of CoLA has high token and sentence coverage even though $\mathcal{V}_C$ of CoLA is much smaller than QNLI and MRPC (Table 7). CoLA, however, has a large performance drop despite high coverage. We guess that the differences in performance degradation are explained by differences in the information required by the tasks, rather than by the rate of affected sentences.

The overhead to recover rare token embeddings It requires 142 ms to recover rare token embedding ($r(V_C) = 1.0$, $k = 5$) for JCoLA "`validation`" datasets using a server with Intel Xeon 2.40-GHz CPU. This is negligible ($< 5\%$) against the inference time (3010 ms) of the same datasets with the original PLM, which uses an additional NVIDIA P6000 GPU for matrix multiplication.

4.2 Sensitivity to compression rate

Using our method, we can explicitly control the compression rate of embedding by varying $r(\mathcal{V}_C)$. We thus investigate the relation between the compression rate of the fine-tuned PLMs by our method and the PLM's performance (Tables 1 and 2), among three datasets: STS-B and QNLI in GLUE, and JSTS in JGLUE. These datasets have different training data sizes (5.2k examples for STS-B, 94.3k for QNLI, and 11.2k for JSTS in our settings), as shown in Table 7.

We can see that the compression rates of CoLA and JSTS of $r(\mathcal{V}_C) = 1.0$ and MRPC and STS-B of $r(\mathcal{V}_C) = 0.5$ are similar but their performance drops differ greatly, as shown in Tables 1 and 2. This will be because individual tasks require different degrees of information, and we thus need to tune the compression rate depending on the target downstream tasks.

r($\mathcal{V}_C$)	GLUE				JGLUE	
	STS-B		QNLI		JSTS	
	all	(emb.)	all	(emb.)	all	(emb.)
0.5	47.7M	(4.15M)	53.7M	(10.11M)	46.1M	(2.56M)
1.0	51.6M	(8.01M)	63.7M	(20.13M)	47.7M	(4.19M)
orig.	67.0M	(23.44M)	67.0M	(23.44M)	68.7M	(25.17M)

Table 8: The number of parameters in the DistilBERT models with vocabulary compressed by our method.

	CoLA	JCoLA
$r(\lvert\mathcal{V}_C\rvert) = 1.0$		
<unk>	35.55	5.63
$k = 1$	70.56	29.14
$k = 2$	75.60	34.59
$k = 3$	77.51	37.41
$k = 4$	78.53	39.21
$k = 5$	79.16	40.48
AQ ($\mathcal{V}_C \cup \mathcal{V}_R$)	77.89	65.27
AQ ($\mathcal{V}_C$)	70.16	54.73
AQ ($\mathcal{V}_R$)	79.55	66.55

Table 9: Cosine similarity of the approximated embeddings to the original embedding.

4.3 Sensitivity to k

In our method and the PCA baseline, we can obtain a better approximation by increasing the number of the source embeddings, k. In this section, we investigate the relation between the quality of approximation and the PLM's performance.

Table 9 shows the cosine similarity between the original and the reconstructed embeddings in the PLMs fine-tuned to the CoLA and JCoLA datasets. From the table, we can confirm that more similar embeddings to the original can be reconstructed when we increase the number of the source embedding, k, in both CoLA and JCoLA. However, the higher similarity does not always lead to higher performance as lower k are chosen in most datasets in Table 1 and 2. Meanwhile, AQ achieves comparable (CoLA) or much better (JCoLA) similarity for rare tokens but performs poorly (Table 2). These results confirm that the noises in common token embeddings are vital and the PLMs have a strong contextualized ability to guess the meanings of rare tokens from their surrounding contexts, we do not need to care much about tuning k to obtain a better approximation of embeddings.

5 Related Work

In the development of neural network-based NLP, how to embed a sequence of discrete symbols in languages into the continuous space has been an important issue, and various compact representations of embeddings have been explored. In what follows, we first review approaches to compressing word embeddings (§ 5.1). We next introduce finer-grained tokenization than words, which results in compact embedding layers (§ 5.2). We then discuss a method of predicting embeddings of out-of-vocabulary words (§ 5.3).

5.1 Compressing Word Embeddings

Classical approaches to neural language modeling leverage word-level embeddings such as CBoW (Mikolov et al., 2013) and GloVe (Pennington et al., 2014), which are learned via shallow neural networks. Since out-of-vocabulary (OOV) words cause serious issues in word-based embeddings, the embeddings are often trained to cover as many words as possible, which makes embedding layers larger. Hence, researchers have worked to compress word embeddings during or after training neural models.

Matrix (Tensor) factorization decomposes a large matrix (tensor) by a product of low-rank matrices (tensors) and has been used to compress word embeddings (Chen et al., 2018a; Acharya et al., 2019; Winata et al., 2019; Lan et al., 2020; Hrinchuk et al., 2020; Lioutas et al., 2020; Lee et al., 2021; Wang et al., 2023). In particular, ALBERT (Lan et al., 2020) learns to represent the embedding layer of a PLM with a product of two small matrices during pre-training; although the factorized vocabulary reduces the memory footprint, it involves matrix multiplications that slow the inference and is not adopted in other PLMs.

Sparse coding has been thereby explored to address the aforementioned issue in matrix factorization (Faruqui et al., 2015; Chen et al., 2016; Shu and Nakayama, 2017; Chen et al., 2018b; Tissier et al., 2019; Ma et al., 2019; Kim et al., 2020). The sparse coding represents embeddings using a sparse linear combination of basis embeddings; each embedding is represented by a short sparse vector, which has pairs of IDs for basis embeddings and weight (optional). In particular, Chen et al. (2016) adopted frequency-aware partial sparse coding as ours and applied it to embedding layers of RNN-LMs with word tokenization. To choose a small subset of basis (common token) embeddings for each rare token embedding, they used ℓ_1-regularization. However, the sparsity is limited

since ℓ_1 regularization does not directly minimize the number of basis embeddings for reconstruction.

In this study, focusing on the recent subword-based Transformer-based PLMs that have strong contextualization abilities of embeddings, we develop a lightweight method that chooses a fixed number of basis embeddings to represent each rare embedding from its nearest-neighbor common token embeddings and confirms that it attains high sparsity while retaining the original accuracy.

5.2 Finer-grained Tokenization

To address the issue of OOV words, researchers leveraged finer-grained tokenization based on subwords (Sennrich et al., 2016; Kudo, 2018) to back-off embeddings of OOV words to those of subwords (ultimately, characters or bytes). The finer-grained tokenization allows us to reduce the vocabulary size dramatically. Furthermore, to handle massive vocabularies in multilingual models, character- (Clark et al., 2022) and byte-level tokenization (Xue et al., 2022) have been used. These finer-grained tokenizations, however, incur high computational costs, because they heavily rely on the hidden layers of PLMs to recover (sub)word-level representations. Meanwhile, recent large language models (LLMs) are trained with a larger set of subwords; Takase et al. (2024) reported that larger vocabulary contributes to the performance of LLMs. Meanwhile, when we adopt pretrain-and-fine-tune paradigm, we need to stick to the original tokenizer of the PLMs, since it is difficult to obtain a different set of fine-grained vocabularies that replace the existing subword-level vocabularies in the PLMs. We thus need to address large subword vocabularies of PLMs to compress PLMs.

5.3 Predicting OOV Embeddings

As stated in § 5.1, out-of-vocabulary (OOV) words had been a problem in the word-level embeddings, before the subword-based tokenization becomes a de-facto standard in neural text processing via Transformer-based PLMs. Several researchers thus attempted to reconstruct OOV embeddings from subword embeddings (Pinter et al., 2017; Zhao et al., 2018; Sasaki et al., 2019; Fukuda et al., 2020; Chen et al., 2022). Although these methods can compute OOV embeddings from subword embeddings, they usually leverage a neural network to accurately predict OOV embeddings, which not only requires an additional memory footprint but also slows down the inference. In this study, we resort to the strong contextualization abilities of PLMs to handle OOV words, and focus on reconstructing rare token embeddings by abusing common token embeddings, to minimize the space and time cost to compute the embeddings in the inference.

6 Conclusions

We proposed a simple yet effective sparse coding method to compress the embedding layer of a given fine-tuned PLM. We keep common tokens that appear frequently in the fine-tuning data and only compress the embeddings of rare tokens that do not appear in the fine-tuning data. We select only a small subset of the nearest neighbor source (common token) embeddings to approximate the target (rare token) embeddings so that we represent the target embeddings with only a small number of parameters. Our experimental results confirmed that our frequency-aware partial sparse coding can greatly compress the embedding layer while preventing performance degradation. Our method works effectively without carefully choosing the number of the source embedding for compression.

In future work, we will apply a method to select the target tokens for compression from the vocabulary while considering the easiness of reconstruction as well as the frequency. We also plan to apply our method to decoder-only and encoder-decoder LMs, although there are issues as stated in the Limitations section.

Limitations

Since our method discards the original embeddings for rare tokens and dynamically reconstructs those embeddings upon request, the application to decoder-only and encoder-decoder PLMs has some challenges. First, If the PLMs do not adopt the weight tying, which shares the weights of the embedding and softmax layers, then our method is applicable to the embedding layers. If the PLMs adopt the weight tying, a naive application of our method to those PLMs will result in outputs without rare tokens. However, we will be able to generate rare tokens, by remembering neighboring rare tokens for each common token with embeddings; we first choose a common token as the next token, by greedy decoding or some decoding strategy, and then reconstruct the neighboring rare token embeddings to include those rare tokens as the candidates of the next token. We will plan to evaluate this method on recent decoder-only large LMs.

Acknowledgements

This work was partially supported by the special fund of Institute of Industrial Science, The University of Tokyo and by JSPS KAKENHI Grant Number JP21H03494.

References

Anish Acharya, Rahul Goel, Angeliki Metallinou, and Inderjit Dhillon. 2019. Online embedding compression for text classification using low rank matrix factorization. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press.

Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 931–938. IEEE.

Lihu Chen, Gael Varoquaux, and Fabian Suchanek. 2022. Imputing out-of-vocabulary embeddings with LOVE makes LanguageModels robust with little cost. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3488–3504, Dublin, Ireland. Association for Computational Linguistics.

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

Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018b. Learning k-way d-dimensional discrete codes for compact embedding representations. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 854–863. PMLR.

Yunchuan Chen, Lili Mou, Yan Xu, Ge Li, and Zhi Jin. 2016. Compressing neural language models by sparse word representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 226–235, Berlin, Germany. Association for Computational Linguistics.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

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

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A. Smith. 2015. Sparse overcomplete word vector representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1491–1500, Beijing, China. Association for Computational Linguistics.

Nobukazu Fukuda, Naoki Yoshinaga, and Masaru Kitsuregawa. 2020. Robust Backed-off Estimation of Out-of-Vocabulary Embeddings. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4827–4838, Online. Association for Computational Linguistics.

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, Online. Association for Computational Linguistics.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

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

Taku Kudo. 2018. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Kentaro Kurihara, Daisuke Kawahara, and Tomohide Shibata. 2022. JGLUE: Japanese general language understanding evaluation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 2957–2966, Marseille, France. European Language Resources Association.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning

of language representations. In *International Conference on Learning Representations*.

Jong-Ryul Lee, Yong-Ju Lee, and Yong-Hyuk Moon. 2021. Block-wise word embedding compression revisited: Better weighting and structuring. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4379–4388, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Vasileios Lioutas, Ahmad Rashid, Krtin Kumar, Md. Akmal Haidar, and Mehdi Rezagholizadeh. 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, Online. Association for Computational Linguistics.

Yukun Ma, Patrick H. Chen, and Cho-Jui Hsieh. 2019. MulCode: A multiplicative multi-way model for compressing neural language model. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5257–5266, Hong Kong, China. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

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*, pages 102–112, Copenhagen, Denmark. Association for Computational Linguistics.

Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

Jin Sakuma and Naoki Yoshinaga. 2019. Multilingual Model Using Cross-Task Embedding Projection. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 22–32, Hong Kong, China. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *Proceedings Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing*.

Shota Sasaki, Jun Suzuki, and Kentaro Inui. 2019. Subword-based Compact Reconstruction of Word Embeddings. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3498–3508, Minneapolis, Minnesota. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Raphael Shu and Hideki Nakayama. 2017. Compressing word embeddings via deep compositional code learning. In *Proceedings of the sixth International Conference on Learning Representations*.

Sho Takase, Ryokan Ri, Shun Kiyono, and Takuya Kato. 2024. Large vocabulary size improves large language models. *arXiv*, abs/2406.16508.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2019. Near-lossless binarization of word embeddings. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press.

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*, volume 30. Curran Associates, Inc.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. 2024. Efficient large language models: A survey. *Transactions on Machine Learning Research*. Survey Certification.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Haoyu Wang, Ruirui Li, Haoming Jiang, Zhengyang Wang, Xianfeng Tang, Bin Bi, Monica Cheng, Bing Yin, Yaqing Wang, Tuo Zhao, and Jing Gao. 2023. Lighttoken: A task and model-agnostic lightweight token embedding framework for pre-trained language models. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 2302–2313, New York, NY, USA. Association for Computing Machinery.

Genta Indra Winata, Andrea Madotto, Jamin Shin, Elham J Barezi, and Pascale Fung. 2019. On the effectiveness of low-rank matrix factorization for LSTM model compression. In *Proceedings of the 33rd Pacific Asia Conference on Language, Information and Computation*, Hakodate, Japan.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

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*, pages 601–606, Brussels, Belgium. Association for Computational Linguistics.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A survey on efficient inference for large language models. *arXiv*, abs/2404.14294.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2023. A survey on model compression for large language models. *arXiv*, abs/2308.07633.