Do We Really Need Subword Prefix For Bert Tokenizer? An Empirical Study

Anonymous ACL submission

Abstract

Subword tokenizers are widely adopted for pretrained language models (PLMs). As they are originally designed for machine translation (MT) tasks, these tokenizers always introduce a subword prefix to distinguish starting and continuing tokens for better decoding. In this paper, we empirically study the necessity of the subword prefix for pretrained language models on natural language understanding (NLU) tasks. Experimental results show that our prefix-free variant, Bert-SPFT, achieves a comparable result with 19% fewer embedding parameters. The further probing task also suggests that the capability to distinguish the subword types is not related to the model performance.

1 Introduction

Tokenization is an essential preprocessing step for many NLP tasks by transforming original text into tokens. To alleviate unknown token problems, subword tokenization methods are proposed and become the main streaming for machine translation tasks(Sennrich et al., 2016). In recent years, subword tokenization methods are also widely adopted in pretrained languages models (PLMs) like Bert(Devlin et al., 2019), Roberta(Liu et al., 2019), GPT-2(Radford et al., 2019) and so on.

The conventional subword tokenization methods are originally designed for machine translation (MT) tasks for which it's necessary to transform the generated tokens into space-separated texts for decoding. A prefix is introduced to these tokenizers, which makes subwords two different types: starting subwords and continuing subwords. We call it the **subword prefix**. For example, the tokenizer of Bert employs the prefix "##" for continuing subwords. For decoding, a space can be added directly in front of the starting subword to ensure the successful decoding of space-separated text.

Although the subword prefix strategy facilitates the decoding of machine translation models, it is unnecessary for natural language understanding (NLU) tasks. We argue that the subword prefix even makes the vocabulary parameter redundant to some extent. With the subword prefix, some words may have both starting form and continuing form in the vocabulary, such as"apple" and "##apple", "layer" and "##layer". Taking *bert-base-uncased*[1] as an example, 8.8% of tokens have two forms at the same time.

We first investigate the performance of the tokenizer without using the subword prefix. In this paper, we use the Bert tokenizer as an example as it is a widely used pretrained language model for NLU tasks. Based on the original Bert tokenizer, we propose a tokenization method, Subword Prefix Free Tokenizer (SPFT), without using the subword prefix. On more than 10 NLU tasks, such as text classification, question answering, and named entity recognition, our tokenizer achieves comparable performance even with 19% less embedding parameters. It validates our hypothesis that the subword prefix is unnecessary for NLU tasks.

We further analyze the segmentation result and find that the Bert-SPFT produces a better segmentation result both statistically and morphologically. We also observe that Bert can easily distinguish the two types of subwords at the very early pretraining stage, which indicates that this ability can not be translated into model performance.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, we are the first to study how the subword prefix affects the PLMs' performance for NLU tasks.
- Experiments on more than 10 NLU tasks demonstrate that Bert-SPFT can obtain a comparable result with the original Bert while having a 19% reduction in vocabulary parameters.

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

- We provide in-depth analysis to learn that Bert-SPFT can achieve better morphological and statistical performance.

- The results of a probing experiment suggest that PLMs can predict the subword prefix at a very early stage. However, this capability can not be translated to the downstream tasks' performance.

2 Related Work

In this section, we first revisit the subword tokenization methods and discuss how tokenization methods affect pretrained language models.

2.1 Subword Tokenization

Gage proposes the Byte-Pair-Encoding(BPE) algorithm as a compression algorithm that merges most often co-occurring pairs into one token. Sennrich et al. first applies BPE on machine translation tasks. Wang et al. applies BPE, not at the character level but at the byte level. GPT-2(Radford et al., 2019) and Roberta(Liu et al., 2019) employs the BBPE as the tokenizer. Schuster and Nakajima proposes the WordPiece for processing Japanese and Korean text. The difference is that WordPiece merges pairs with the likelihood of an n-gram-based language model. BERT(Devlin et al., 2019) uses WordPiece as the tokenizer. Kudo employs a unigram language model(UnigramLM) to transform the text into subwords. Yang et al. employs the UnigramLM method in their tokenizer. Kudo and Richardson provides a popular tokenizer package SentencePiece for both BPE and Unigram LM algorithms.

2.2 How Tokenization Methods affect Downstream Tasks

Gallé claims that the shorter token sequence translates to better results on machine translation tasks. Gowda and May finds that the word frequency of the long-tail words affects the translation results. Bostrom and Durrett claims that UnigramLM gives a better segmentation result than BPE for morphological performance and downstream task results.

3 Experiments on NLU tasks

In this section, we first give a quick look at the Bert tokenizer and our subword prefix-free variant, Bert-SPFT. Then we test our Bert-SPFT on various downstream NLU tasks.

3.1 Bert Tokenizer

The Bert model uses WordPiece(Schuster and Nakajima, 2012) as its tokenizer. It introduces a subword prefix "##" to distinguish the subwords at the beginning (starting subwords) and the others (continuing subwords) when building its vocabulary. Then, based on this vocabulary, it is simplified to a longest-match tokenizer for practical use.

We argue that the subword prefix is unnecessary and it is a waste of vocabulary capacity. We propose a subword prefix-free tokenizer Bert-SPFT based on the Bert tokenizer. The only difference is our Bert-SPFT only uses the subword without prefix in the vocabulary. As Figure1 shows, compared to the Bert tokenizer, our Bert-SPFT tends to preserve the core parts of the words with the fixed vocabulary.

3.2 Experiments on Downstream tasks

In this subsection, we perform extensive experiments to examine how Bert-SPFT works on NLU tasks.

We use the *bert-base-uncased* as the pretrained model. We choose text classification tasks from GLUE (Wang et al., 2018), question answering tasks of SQuADv1.1(Rajpurkar et al., 2016), named entity recognition tasks of CoNLL2003(Tjong Kim Sang and De Meulder, 2003). The experimental details are listed in Appendix A.

As Table 1 shows, our Bert-SPFT only performs slightly worse on the average score with a reduction on embedding parameters. We assume that the Bert-SPFT is not well trained in the pretrained model. We further conduct a continue-pretraining with the Bert-SPFT. After continuing pretraining for 5000 steps, we get comparable results on these tasks. This experimental result demonstrates the effectiveness of our proposed Bert-SPFT on downstream NLU tasks.

4 Analysis

In this section, we carry out a statistical analysis on Bert-SPFT and find that the Bert-SPFT leads to more frequent long-tail tokens. We use a human-annotated segmentation dataset to verify that Bert-SPFT is more consistent with human annotations. Finally, we conduct a probing experiment to verify that although the model can easily distinguish different subwords, this is not related to downstream tasks.

Word: *unchecked*
BERT: *un ##che ##cked*
BERT-SPFT: *un checked*

Word: *misunderstands*
BERT: *mis ##under ##stands*
BERT-SPFT: *mis understands*

Word: *singalongs*
BERT: *sing ##alo ##ng ##s*
BERT-SPFT: *sing along s*

Word: *misidentifications*
BERT: *un ##mis ##ide ##nti ##fication ##s*
BERT-SPFT: *mis identification s*

Word: *microelectromechanical*
BERT: *micro ##ele ##ct ##rom ##ech ##ani ##cal*
BERT-SPFT: *micro electro mechanical*

Word: *unsuspected*
BERT: *un ##sus ##pe ##cted*
BERT-SPFT: *un suspected*

Word: *eyewitness*
BERT: *eye ##wi ##tness*
BERT-SPFT: *eye witness*

Word: *mispronunciation*
BERT: *mis ##pro ##nu ##ncia ##tion*
BERT-SPFT: *mis pronunciation*

Figure 1: Examples of the tokenization results of the original Bert tokenizer and our proposed Bert-SPFT.

	Cola	QNLI	MNLI(m/mm)		MRPC	QQP	STSB	STS2	QA	NER	Avg
Bert	55.53	**91.46**	**84.58**	**84.90**	83.04	**90.96**	88.21	92.84	**88.33**	**94.62**	85.45
-S	55.59	90.94	84.13	84.32	**84.71**	90.88	88.47	92.57	87.89	93.77	85.33
-S + P	**57.06**	91.08	84.37	84.55	83.33	**90.96**	**88.70**	**92.91**	88.02	94.14	**85.51**

Table 1: Experimental results of downstream tasks. Bert is the *bert-base-uncased* model. "**-S**" means using the same pretrained model with our proposed Bert-SPFT. "**-S + P**" means we first conduct a continue pretraining with the Bert-SPFT and *bert-base-uncased* model. QA represents the F1 score of SQuADv1.1. NER represents the F1 score of CoNLL2003. All the results are the average of five runs.

	vocab size	avg token length
Bert	30522	1.1273
-S	24694 (-19%)	1.1451 (+1.59%)
-S + R	30522	1.1042 (-2.05%)

Table 2: The average token length and vocab size of three tokenizers. **Bert** means the *bert-base-uncased* tokenizer. "**-S**" means our proposed Bert-SPFT. "**-S+R**" means our proposed Bert-SPFT with a reconstructed vocabulary".

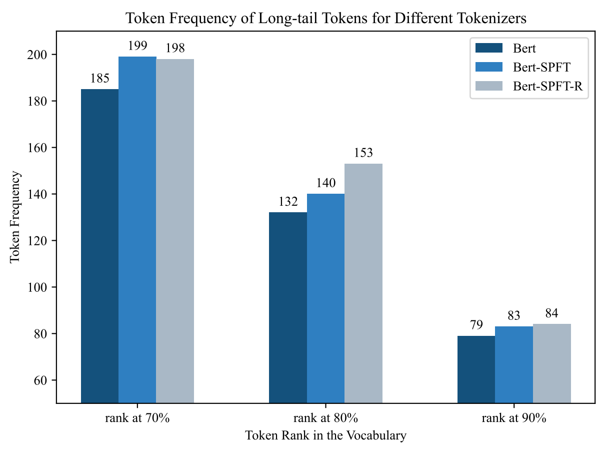

Figure 2: The token frequency of long-tail tokens in the vocabulary of different tokenizers.

4.1 Statistical Analysis

Since our Bert-SPFT only uses the subwords without “##”, the vocabulary size is about 81% of the original Bert tokenizer. For a fair comparison, we also train a WordPiece tokenizer with the same vocabulary size as the original Bert tokenizer.

As Table2 shows, Bert-SPFT tends to lead to a longer token sequence as the vocabulary shrinks by 19%. However, given the same vocab budget, the Bert-SPFT leads to 2% less than the original Bert in token length.

As Figure2 shows, the last 30% of tokens in the vocabulary has a higher token frequency which suggests for both pretrain and downstream tasks, using Bert-SPFT can lead to a more adequate training on long-tail tokens.

4.2 Morphological Experiments

Besides the benefits in average token length and long-tail token frequency, we verify the consistency of Bert-SPFT with human segmentation data.

Motivated by Itzhak and Levy, we use LADEC(Gagné et al., 2019) a large dataset of human-annotated English compounds dataset and measure the overlap of the tokenized result and human annotated labels.

The total amount of the LADEC is 8956. Each compound word has two morphemes. We remove all the plural words and finally get 6017 words.

We define two metrics to match the consistency

	Full Match	Term Match
Bert	60.63%	71.61%
-S	75.93(+25.23)%	81.4(+13.67)%

Table 3: The full match ratio and term match ratio of the original *bert-base-uncased* tokenizer and our proposed Bert-SPFT on the LADEC dataset.

of tokenized results and human labels.

Full Match Ratio means that the number of the tokenized result is consistent with human labels. Given the word “walkabout” with the compounds “walk” and “about”, if the tokenized result is “walk about” or “walkabout”, we consider it a full match.

Term Match Ratio measures how many terms of the tokenized result match the human labels. The denominator is the total number of terms. The numerator is the total matched terms. If a word is matched as a whole, we consider the matched term number to be 2.

As Table 3 shows, our proposed Bert-SPFT outperforms the original Bert tokenizer with a 25.23% increase in full match ratio and a 13.67% increase in term match ratio.

The experimental results verify that the results of Bert-SPFT and human segmentation data are more consistent.

4.3 Probing on subword types

Results on downstream tasks demonstrate the effectiveness of Bert-SPFT, and we would like to further investigate subword prefixes in pretrained models. We conduct further experiments to examine the following research questions.

- **RQ1. Can PLMs distinguish the starting subword and continuing subword?**
- **RQ2. If the PLMs have the ability, is this correlated to the model performance?**

Motivated by previous work(Itzhak and Levy, 2022), we directly extract the embedding matrix from the pretrained *bert-base-uncased* model. We split them with a portion of 80%, 10%, and 10% as train/dev/test datasets.

We build a simple three-layer feed-forward network and train it for 100 epochs.

We find that the model can easily distinguish the starting subwords from the continuing subwords with nearly 100% accuracy which proves the **RQ1**.

To validate the **RQ2**, we train a Bert from scratch and save the checkpoint for every 1000 steps. For each checkpoint, we predict the subword prefix with the embedding matrix and perform MNLI fine-tuning to verify the performance on downstream tasks.

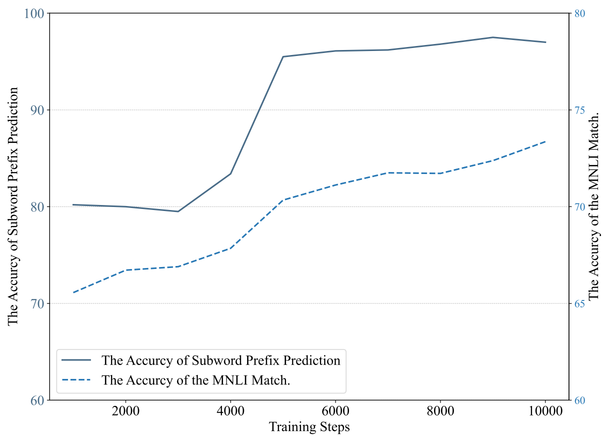

Figure 3: The accuracy of subword prefix prediction and MNLI match score for each training step.

As Figure3 shows, at the 5000 steps of training, the model is capable of distinguishing the two types of subwords with more than 95% accuracy. However, when we use the MNLI score to measure the performance of the model on downstream tasks, we find that the model is still far from being well trained. This probing experiment proves that although PLMs can recognize different subword types, this cannot be translated into benefits for downstream tasks.

5 Conclusion

In this paper, we empirically study the necessity of the subword prefix in tokenizer. Downstream experiments prove our subword prefix-free tokenizer, Bert-SPFT, achieves a comparable result with 19% fewer embedding parameters. From statistical and morphological aspects, we also observe a better segmentation with our Bert-SPFT. We further conduct a probing task that suggests the capability to distinguish the subword types is not related to the model performance.

In future work, we would like to further investigate the Bert-SPFT by pretraining a Bert model from scratch. We believe that pre-training from scratch can better exploit the Bert-SPFT. Besides, we want to study the subword prefix in other tokenizers, such as the byte-level BPE tokenizer of Roberta. The more difficult part is that the tokenizer of Roberta directly models the spaces.

References

Kaj Bostrom and Greg Durrett. 2020. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624, Online. Association for Computational Linguistics.

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

Philip Gage. 1994. A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

Christina L Gagné, Thomas L Spalding, and Daniel Schmidtke. 2019. Ladec: The large database of english compounds. *Behavior research methods*, 51(5):2152–2179.

Matthias Gallé. 2019. Investigating the effectiveness of BPE: The power of shorter sequences. 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 1375–1381, Hong Kong, China. Association for Computational Linguistics.

Thamme Gowda and Jonathan May. 2020. Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964, Online. Association for Computational Linguistics.

Itay Itzhak and Omer Levy. 2022. Models in a spelling bee: Language models implicitly learn the character composition of tokens. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5061–5068, Seattle, United States. 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.

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

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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, Austin, Texas. Association for Computational Linguistics.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE.

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.

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

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.

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2020. Neural machine translation with byte-level subwords. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9154–9160.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

A Appendix

A.1 Details of Downstream Experiments

A.1.1 Implementation Details on Downstream Experiments

We use the same setting for text classification tasks in GLUE. The batch size is 32. The max sequence

	Exact Match	F1
Bert	80.75	88.33
Bert-SPFT	80.21	87.89
Bert-SPFT *	80.34	88.02

Table 4: Exact match and F1 score of different tokenizers on SQuADv1.1.

	Acc	F1	Pre	Rec
Bert	98.92	94.62	94.18	95.07
Bert-SPFT	98.74	93.77	93.49	94.05
Bert-SPFT *	98.91	94.14	93.84	94.44

Table 5: Accuarcy, F1, Precision and Recall score on CoNLL2003.

length is 128. The train epochs are 3. The learning rate is set to $2e-5$.

For SQuADv1.1, we use the batch size of 12. The max sequence length is 384. The stride window size is 128. The learning rate is set to $3e-5$.

All the experiments are implemented with huggingface[2] and performed on an NVIDIA A100. For each experiment, we run 5 times and then calculate the average scores.

A.1.2 SQuADv1.1 Result

As Table4 shows the exact match score and F1 score of different tokenizers on SQuADv1.1.

A.1.3 CoNLL2003 Result

As Table5 shows the accuracy, F1, precision and recall score on CoNLL2003 NER task.

A.2 Details of Morphological Experiments

	Bert	Bert-SPFT
Full Match	520	520
Exact Match	3128	4049
Single Term Match	1322	658

Table 6: Details Matching Counts of the Bert tokenizer and Bert-SPFT in the LADEC dataset.**Full Match** means the given word is in the vocabulary and will not be segmented anymore. **Exact Match** means the segmentation results are consistent with the human annotation. **Single Term Match** means only one term is matched for human annotation and tokenized results.

	Label-0	Label-1
Training Set	18944	4666
Evaluation Set	2336	585
Test Set	2414	577

Table 7: In the Probing experiment, the number of positive and negative examples in the training set, evaluation set, and test set

A.3 Details of Probing Experiments

A.3.1 Statistics for Datasets

A.3.2 Implementation Details of the Probing Model

The size of each feed forward layer is $(768, 32)$, $(32, 16)$ and $(16, 1)$. We use the batch size of 256. We train the model for 100 epochs. The experiments are performed on CPUs.

A.3.3 Implementation Details of the Bert Pre-training

As a training set, we leverage data from the English wiki and book corpus. We take advantage of a 256 train batch size. 128 is the maximum sequence length. Weight decay is set to 0.01 and the learning rate is $1e-4$.

A.4 Limitations

The results in this paper have some limitations. First, we only focus on the WordPiece tokenizer. But our method can also be applied to other tokenizers like the Byte-level BPE tokenizer (Wang et al., 2020) of Roberta(Liu et al., 2019) . In subsequent experiments, we will conduct research on these tokenizers. Second, we only use a pre-trained BERT and then conduct a few steps continue pretraining to verify the effect of our proposed tokenizer on downstream tasks. We think this does not fully demonstrate the effectiveness of our proposed tokenizer. For future work, we would like to train a Bert from scratch and fine-tune downstream tasks to see if there are more improvements.

[2]https://huggingface.co/