Jamo Pair Encoding: Subcharacter Representation-based Extreme Korean Vocabulary Compression for Efficient Subword Tokenization

Sangwhan Moon†‡, Naoaki Okazaki†
Tokyo Institute of Technology†‡, Odd Concepts Inc.‡,
sangwhan@iki.fi, okazaki@c.titech.ac.jp

Abstract
In the context of multilingual language model pre-training, vocabulary size for languages with a broad set of potential characters is an unsolved problem. We propose two algorithms applicable in any unsupervised multilingual pre-training task, increasing the elasticity of budget required for building the vocabulary in Byte-Pair Encoding inspired tokenizers, significantly reducing the cost of supporting Korean in a multilingual model.

Keywords: tokenization, vocabulary compaction, sub-character representations, out-of-vocabulary mitigation

1. Background
With the introduction of large-scale language model pre-training in the domain of natural language processing, the domain has seen significant advances in the performance of downstream tasks using transfer learning on pre-trained models (Howard and Ruder, 2018; Devlin et al., 2018) when compared to conventional per-task models. As a part of this trend, it has also become common to perform this form of pre-training against multiple languages when training a single model. For these multilingual pre-training cases, state-of-the-art methods have relied on subword based tokenizers, such as Byte-Pair Encoding (BPE) (Sennrich et al., 2016) or SentencePiece (Kudo and Richardson, 2018) as a robust mechanism to mitigate the out-of-vocabulary (OOV) problem at a tokenizer level, by having a fallback to a character level vocabulary. Not only have these methods shown to be robust against OOV compared to standard lexicon-based tokenization methods, but they also have benefited from a computational cost perspective as it reduces the size of the input and output layer.

While these methods have shown significant improvements in alphabetic languages such as English and other Western European languages that use a Latin alphabet, these methods have limitations when applied to languages that have a large and diverse character level vocabulary, such as Chinese, Japanese, and Korean (CJK) languages.

In this paper, we describe the challenges of subword tokenization when applied against CJK. We discuss the difference of Korean compared to other CJK languages, how to take advantage of the difference which Korean has when a subword tokenizer is used, and finally, propose a subword tokenizer-agnostic method, which allows the tokenizer to take advantage of Korean specific properties.

2. Problem Definition
CJK languages, due to the strong linguistic dependency of borrowed words from Chinese as part of their vocabulary, have a much more extensive range of characters needed to express the language compared to other alphabetic (e.g., Latin) languages. This reflects directly on the vocabulary budget requirements needed for an algorithm, which builds a subword vocabulary on character pairs such as BPE. Roughly, the minimum size of the subword vocabulary can be approximated as \( |V| \approx 2|V_c| \), where \( V \) is the minimal subword vocabulary, and \( V_c \) is the character level vocabulary.

Since languages such as Japanese require at least 2000 characters to express everyday text, in a multilingual training setup, one must make a tradeoff. One can reduce the average surface of each subword for these character vocabulary intensive languages, or increase the vocabulary size. The former trades off the performance and representational power of the model, and the latter has a computational cost. Similar problems also apply to Chinese, as it shares a significant portion of the character level vocabulary. However, this also allows some level of sharing, which reduces the final budget needed in the vocabulary.

Korean is an outlier in the CJK family, which linguistically has a shared vocabulary in terms of roots, but uses an entirely different character representation. A straightforward approach would be to share the character level vocabulary between CJK languages, as it was possible between Chinese and Japanese. However, this, unfortunately, is not a straightforward operation, as Hangul (the Korean writing system) is phonetic, unlike the other two examples. This means that while the lexicon may have the exact same roots, the phonetic transcription is challenging to do an inverse transform algorithmically. This requires comprehension of the context to select the most likely candidate, which would be analogous to a quasi-masked language modeling task.

3. Related Work and Background
The fundamental idea of characters is not new; in the past, many character-level approaches have been proposed in the form of task-specific architectures. There are also sub-character level methods analogous to our method, all of which we discuss in the language-specific sections below.

3.1. Non-Korean Languages
A study on a limited subset of Brahmic languages (Ding et al., 2018) proposes a method which can be used to reduce the vocabulary budget needed for all languages by generalizing, simplifying, then aligning multiple language alphabets together. This is applicable when the writing systems have genealogical relations that allow this form of
alignment. Previous works (He et al., 2018; Shi et al., 2015; Sun et al., 2014; Yin et al., 2016) demonstrate the potential of sub-character based methods in the context Chinese and Japanese of CJK languages through radical based decomposition.

### 3.2. Korean

Korean, as shown in figure 1 builds on a small phonetic alphabet, and uses a combination of the consonants and vowels called Jamo as a building block, and use a combination of those when composing each character. Following the notation in (Stratos, 2017), given this Jamo alphabet \( J \), where the following table can explain \( |J| = 51 \), and each possible role when forming a complete character.

The Jamo alphabet \( J \) is a union defined as \( J = J_h \cup J_v \cup J_t \), where \( J_h \) is a Choseong (head consonant), \( J_v \) is Jungseong (vowel), and \( J_t \) is Jongseong (tail consonant). Therefore, the first example illustrated in figure 1 can be explained as \( \exists \in J_h \), \( \exists \in J_v \), and \( \exists \in J_t \). Note that \( J_t \) can be omitted, in which case it corresponds to \( \langle \text{nil} \rangle \in J_t \).

We attempt to address this limitation in our work. This is analogous to how subword tokenization methods have brought to the field guarantees of lossless encoding and decoding, which was not possible with conventional lossy encoding methods such as lemmatization, stemming, and other normalization methods.

| Level | Jamo (Subcharacters) |
|-------|----------------------|
| \( J_h \), Choseong | ㅏ ㅐ ㅑ ㅒ ㅓ ㅔ ㅕ ㅖ |
| \( J_v \), Jungseong | ㅗ ㅘ ㅙ ㅚ ㅛ ㅜ ㅝ ㅞ ㅟ ㅠ ㅡ ㅢ ㅣ |
| \( J_t \), Jongseong | ㄱ ㄲ ㄳ ㄴ ㄵ ㄶ ㄷ ㄹ ㄺ ㄻ |

Table 1: Hangul Jamo sub-characters, along with their respective positional roles for composition.

Exploiting these characteristics is not a new idea, and have been explored in the context of an end-to-end architecture in (Stratos, 2017), as a method for learning subword embeddings in (Park et al., 2018), and for classification in (Cho et al., 2019).

One significant contribution of our method is the guarantees of round trip consistency. Previous work, which we discussed above, also discuss sub-character (Jamo) based methods, but the evaluation was limited to tasks that do not require generation, and reconstruction was not discussed.

Figure 1: Transformation process and Hangul Jamo sub-character composition. In the real world, Hangul to Chinese almost always has a 1:n mapping.

![Figure 1](image1.png)

Figure 2: In this example, the standard form 하다 (to do) can be conjugated into many forms. The first two Jamo correspond to a common morpheme, which through agglutination becomes different conjugations.

### 4. Method

The core motivation of our method is to remove the Unicode bottleneck we have described in the previous section, expose the alphabet composition characteristics and the agglutinative nature of Korean to the subword tokenizers. We also introduce a hard round-trip requirement, which guarantees lossless reconstruction of the transformed text back to the original input while not introducing any unexpected side-effects when training with other languages.

Our method uses special unused characters in the Hangul Unicode page as hints for the processors to operate. Tokenizers will treat these invisible characters the same as a standard character in the same Unicode page. This is crucial for tokenizer implementations that treat pairs from different Unicode pages as non-mergeable.

The method itself is implemented and provided as two modules, with two different algorithms. The two modules are the pre-processor, which performs the Jamo decomposition, and the post-processor, which reconstructs it back to a more human-readable form. The post-processor is only needed for generative tasks when the output will be presented to a human. The pre-processor needs to be run at least once for each input.

We propose two different algorithms: a simple method which aligns the output to a character grid of 3, and a complex method which does not have alignment and instead relies on an automaton to reconstruct the transformed text. Both methods prefix orphaned Jamo (e.g., ㅋㅋ, which roughly means "laugh out loud"), which is extremely common in internet corpora, with a post-processor hint. These methods will be referenced as aligned and automaton in later parts of our paper.
The two methods have different characteristics. Aligned can be reconstructed with an extremely simple post-processor, and has much higher guarantees for reconstruction. The automaton requires a significantly more complex state-machine based post-processor for reconstruction, which operates in the same way an Input Method Processor (IME) does.

4.1. General Decomposition

Decomposition exploits the Unicode level properties, which Korean text has, with similar properties but slightly different from NF KD normalization. The difference is mainly for reconstruction simplicity and reliability when dealing with non-deterministic output, such as what would come out of a model that has not fully converged. Decomposition involves arithmetic operations against the integer Unicode codepoint for each character. Given the integer Unicode codepoint \( c_i \) for character \( c \), and the constants \( k_1 = 44032 \), \( k_2 = 588 \), and \( k_3 = 28 \), the following formula explains the decomposition:

\[
\begin{align*}
   c_i' &= c_i - k_1 \\
   i_h &= \frac{c_i'}{k_2} \\
   i_v &= \frac{c_i' - (k_2 \cdot i_h)}{k_3} \\
   i_t &= (c_i' - (k_2 \cdot i_h)) - k_3 \cdot i_v
\end{align*}
\]

The constants correspond to offset information for each part in the global Unicode table and page for Korean. The computed \( i_h, i_v \), and \( i_t \) correspond to the index of the Jamo in table 1 for \( \mathcal{J}_h, \mathcal{J}_v \), and \( \mathcal{J}_t \) respectively.

Orphaned Jamo is prepended with a special character \( \text{U}+\text{115F} \) (Hangul Choseong Filler). During reconstruction, if the post-processor sees this character, it will treat it as a look-ahead hint and ignore it, and not attempt to reconstruct a full character in the next iteration. Each orphan Jamo character is prefixed with this hint.

This operation is performed for \( c \in C \) where \( c \) is an individual character, and \( C \) is the corpus if the \( c \) is a codepoint that is in a Korean codepage. For non-Korean, the \( c \) is left intact.

4.2. Aligned Processing

4.2.1. Aligned Decomposition

In the pre-processor of the aligned method, we replace cases of <nil> in \( \mathcal{J}_{h,v} \) to a special filler character \( \text{U}+\text{11FF} \) (Hangul Jongseong Ssangnieun) in the Unicode Jamo page to make the output friendlier against algorithms which avoid merging character pairs in different code pages. This particular placeholder was chosen as it is impossible for a user to input this through a standard IME.

This ensures that when the post-processor sees a Choseong (head consonant), it can perform a read-ahead for two more characters and perform a reconstruction.

! Normalization Form Compatibility Decomposition.

| Input | 감가 | 감가 | 감가 |
|-------|-----|-----|-----|
| Output | 감가 | 감가 | 감가 |

Table 2: An example of the decomposition in aligned mode. <o> denotes orphan hints, and <f> denotes filler.

4.2.2. Aligned Reconstruction

In this algorithm, the post-processor is implemented such that it is an inverse transform of the previous decomposition process to derive the original \( c_i \). The post-processor reads each \( c \in C \) from text \( C \), and in the event \( c \) is in the Choseong set \( \mathcal{J}_o \), it reads ahead for two more characters, which are expected to correspond to Jamos within \( \mathcal{J}_h \) and \( \mathcal{J}_v \) respectively. Given these characters \( c_h, c_v, \) and \( c_t \) we look up the index \( i_h, i_v \), and \( i_t \) such that \( c_h = \mathcal{J}_{h,i_h}, c_v = \mathcal{J}_{v,i_v}, \) and \( c_t = \mathcal{J}_{t,i_t} \) with an exception where \( i_t = 0 \) if \( c_t = \text{U}+\text{11FF} \), the filler character.

Given \( i_h, i_v, i_t \), the inverse transform can be done with the following formula.

\[
c_i = k_1 + (i_h \cdot k_2 + i_v \cdot k_3 + i_t)
\]

The computed \( c_i \) is the reconstructed codepoint of the original character before decomposition. While simplicity is the strength of this method, it does not fully expose the agglutinative nature of the underlying language. This is mainly caused by the filler character acting as a merge bottleneck, so the vocabulary training results in fitting towards a complete character boundary for cases where the bottleneck happens. This results in unmerged shared morphemes, and an example is illustrated in figure 3.

| Standard Form | 부르다 | 부르다 |
|--------------|-------|-------|
| Declarative Present Informal Low | 브르다 | 브르다 |
| Correct Shared Surface | 브르다 | 브르다 |
| Actual Shared Surface | 브르다 | 브르다 |

Figure 3: An example of the filler character interfering with a shared morpheme surface merge. The filler character in this example is <f>, which results in an incorrect surface, as seen in the fourth example.

4.3. Automaton (Unaligned) Processing

4.3.1. Automaton Decomposition

The decomposition process in the automaton based algorithm is the same as aligned, as explained in 1. One minor difference is that the filler character can be omitted, as the state machine in the post-processing step can automatically transition the state to a finished character even there is no filler. An example illustrating the difference is in table 3.
### 4.3.2. Automaton Reconstruction

In this case, as the Jamo output is not aligned to the character boundary, reconstruction must be done with a stateful parser algorithm. We use a simplified version of what is commonly used in an IME for Korean input for reconstruction; the main difference is that this assumes vowel and consonant combinations are already pre-combined. This assumption holds as long as the decomposed text was processed through the pre-processing as we defined, as \( i_v \) and \( i_c \) computed earlier reference pre-combined characters in \( J_v \) and \( J_c \). We will not be discussing every corner case in this paper, but at a high level, the state machine uses six registers, and has three possible states as illustrated in figure 4 - initial, interim, and end.

Initial state is triggered when the current character \( c \) satisfies the condition \( c \in J_c \). Interim corresponds to any intermediate state where parts of the registers have been filled in but are not ready to emit a complete character. End generally happens after all registers have been filled, and a character is emitted. In this state machine, the registers can contain values such that \( R_1 \in J_v, R_2 \in J_c, R_3 \in J_v, \) and \( R_4 \in J_c \). The algorithm in which the state machine runs is illustrated in the appendix section, as algorithm 1.

Generally, this state machine will attempt to compose complete characters when all registers R1 to R3 are filled, or if R1 and R2 are filled, and a termination condition occurs. This can happen if R2 is filled and a vowel comes in, as unlike a true IME input, our method has the vowels pre-combined. As the vowel slot has already been saturated, the values of the current registers will be emitted to a character, and the registers will be reset. (This condition is illustrated in the branch of our pseudocode in 1, where the internal state is \( S = \text{INTERIM}, c \in J_c, R_2 \neq 0 \).) One other special case is the carryover operation, which is the only time R4 is used. This is used when R1 to R3 has been filled, but another consonant comes in. In this case, \( c \) is assigned to R4, and the state is set to the initial state.

In this algorithm, emit() is a bridge to the \( R_4 \), and the state is set to the initial state. This function combines two separate consonants (e.g. \( \text{\textasciitilde}{\textstyle\text{\textasciitilde}{}} \)) in \( J_c \) to a combined consonant (e.g. \( \text{\textasciitilde}{\textstyle\text{\textasciitilde}{}} \)) in \( J_c \). This is done through a lookup table of valid combined consonant characters in \( J_c \).

### 4.4. Expectations

As a result of introducing either of these methods, we expect to see improvements in terms of robustness against out-of-vocabulary, flexibility in target vocabulary size, and sub-word tokens, which better represent morphemes or common patterns one can observe in an agglutinative language when the Unicode character boundary bottleneck is removed. We will confirm our hypotheses in the experiment section.

### 5. Experiments

As this work builds on the foundation of unsupervised, sub-word tokenized large corpus pre-training in a multilingual context, as without these methods, the problem we are attempting to address would have never surfaced in the first place. The most significant contribution to this particular approach is BERT (Devlin et al., 2018), which also has a multilingual pre-trained model (bert-base-multilingual-cased) readily available. Along with that, we evaluate against XLM’s (Lample and Conneau, 2019) 100 language model (xlm-mlm-100-1280), and two publicly available pre-trained BERT models: KoBERT and BERT Korean (BERTKr)². We will be using only the tokenizer portions from these models as comparisons.

| Model   | Langs | All | Kor | Average |
|---------|-------|-----|-----|---------|
| BERT    | 104   | 119547 | 1568 | 5.39 / 1.35 |
| XLM     | 100   | 200000 | 2581 | 6.61 / 1.37 |
| KoBERT  | 1     | 8002   | 7378 | 2.70 / 1.67 |
| BERTKr  | 1     | 32006  | 11607 | 3.04 / 2.58 |

Table 4: Langs is the count of languages the model supports. All and Kor is the total vocabulary size followed by the size of the Korean side of the vocabulary. Average is average subword length for the whole vocabulary, followed by the Korean portion.

We compare with and without our methods on the SentencePiece(Kudo and Richardson, 2018) tokenizer, trained against 500,000 randomly sampled sentences from the kosentences Wikipedia dataset, and 50,000 sentences from the Naver Sentiment Movie Dataset’s test set to represent user written content. This corpus was then used to train SentencePiece models with vocabulary sizes of 350, 1K, 2.5K, 5K, and 10K. Sentences longer than the maximum designated sentence length were discarded during training. The hyperparameters for training the tokenizer have been noted in the appendix section.

### 5.1. Datasets

For our experiments, we will be using the following datasets to evaluate the effects of our method. As our primary interest is on the tokenization aspect, we will only be using the corpus to validate the method against domain shift. The rationale behind this is that each underlying model has different performance characteristics, along with the pre-train corpus not being part of the release. For these reasons, we focus mainly on evaluating differences on a tokenization level.

For vocabulary training, we will be using multiple subsets of different sizes from the Wikipedia part of the kosentences³ corpus. As our primary focus is on improving transfer

² [https://github.com/SKTBrain/KoBERT](https://github.com/SKTBrain/KoBERT)
³ [https://github.com/yeontaek/BERT-Korean-Model](https://github.com/yeontaek/BERT-Korean-Model)
⁴ [https://github.com/cynthia/kosentes](https://github.com/cynthia/kosentes)
learning, we follow the same protocol of learning the vocabulary at the pre-train stage and using the vocabulary as-is for downstream tasks. For these reasons, the tokenization performance will be evaluated against the whole corpora in the datasets below.

5.1.1. Naver Sentiment Movie Corpus
The Naver Sentiment Movie Corpus\(^4\) is a sentiment analysis corpus, containing 200,000 user review comments, and a corresponding label which denotes the sentiment. We only used the review comment text from this dataset’s training split, as we used the test set during vocabulary pre-train.

5.1.2. KorQuAD 1.0
KorQuAD\(^5\) is a Korean adaptation of the same reading comprehension task defined as the popular SQuAD (Rajpurkar et al., 2016) dataset. The task involves answering a question given a passage of text, consists of 10,645 passages, and 66,181 questions. We performed the evaluation on all of the passages, questions, and answers.

5.1.3. Text Mining Dataset
The Text Mining Dataset\(^6\) contains Korean text data from two sources; one from Naver news, and the other from Naver movie reviews. The news corpus contains 1,661,786 sentences from article text, and 2,655,847 comments. The movie review corpus contains 3,280,685 review comments.

5.1.4. Namuwiki
The kosentences dataset we used for training the vocabulary above contains text from two sources, one of which is from Wikipedia, and the other from Namuwiki. The latter is closer to casual writing and contains significant amounts of paraphrased non-Korean text. We intentionally do pre-training only on the Wikipedia part of the dataset to use this as an evaluation dataset.

As this dataset contains over 27 million sentences, we sampled 1 million random sentences from this corpus and used it for evaluation.

\(^4\) https://github.com/e9t/nsmc
\(^5\) https://korquad.github.io/
\(^6\) https://github.com/lovit/textmining_dataset

5.2. Results

5.2.1. Out-of-Vocabulary Robustness
The most substantial contribution from this work is the OOV mitigation properties, as it reduces the minimum required character level budget to represent every possible standard case (excluding the three extension blocks) from an excessively large 22,852 character level subwords\(^7\) to a theoretical lower boundary of 102 character level subwords. As observed in the OOV analysis in table 5 and vocabulary analysis in table 6, Our method shows significant improvements in mitigating OOV on equal or even smaller vocabulary sizes compared to the baselines, even with a smaller total vocabulary size allocated for Korean subwords. As we were unable to find good metrics for comparing OOV robustness, we were required to define it ourselves. The four evaluations are computed as follows:

- **OC**: Count of all OOV tokens.
- **OR**: Count of all sentences with OOV tokens, divided by the number of sentences. This is denoted in the percentile scale (x100).
- **LC**: The sum of all OOV token surfaces, divided by the amount of original surface.
- **ML**: Mean length of all OOV token surfaces.

In a vocabulary size of 10K (of which Korean uses less than 25% of the budget), we can consistently observe that the OOV rate is at an utterly negligible amount of less than 0.2%. Our method shows to be active in multiple domains, unlike the behavior we see in the baseline models. Comparatively, the vocabulary was trained with a smaller dataset compared to the baselines. Even with this constraint, due to the decomposition reducing the character level vocabulary, we see improvements even when considering rare combinations for \(V'_v\) and \(V'_t\), which generally requires large corpora when training with composite characters.

5.2.2. Elasticity of Vocabulary Budget
As noted above, we have reduced the minimum character level budget required to represent every possible case down

\(^7\) Unicode Hangul Syllables and Jamo combined is 11,426 characters, which means the budget requirement is usually twice that in the vocabulary for most subword tokenizers.
| Model   | Size | NSMC          |          | TM News-Body     |          | TM News-Comments       |
|---------|------|---------------|----------|------------------|----------|------------------------|
|         |      | OC | OR | LC | ML | OC | OR | LC | ML | OC | OR | LC | ML |
| BERT    | 120K | 81603 | 30.38 | 6.10 | 5.80 | 151175 | 8.26 | 0.50 | 4.17 | 1351933 | 35.83 | 5.60 | 6.58 |
| XLM     | 200K | 259266 | 62.78 | 4.53 | 2.06 | 2627434 | 61.86 | 2.62 | 2.92 | 4239320 | 67.39 | 3.92 | 2.45 |
| KoBERT  | 5K   | 38480 | 14.98 | 0.96 | 1.84 | 30724 | 1.77 | 0.03 | 1.23 | 574058 | 17.00 | 0.81 | 2.00 |
| BertKR  | 32K  | 63729 | 24.93 | 5.68 | 6.52 | 119017 | 6.46 | 0.34 | 3.58 | 1159335 | 31.36 | 4.55 | 6.11 |
|         |      | SP |          |      |      |      |      |      |      |      |      |      |      |
|         |      | 1K | 101738 | 35.12 | 1.98 | 1.61 | 759562 | 30.70 | 0.78 | 1.76 | 2414089 | 51.92 | 2.45 | 1.99 |
|         |      | 2.5K | 18412 | 8.10 | 0.35 | 1.23 | 79231 | 4.60 | 0.08 | 1.23 | 408751 | 13.07 | 0.41 | 1.32 |
|         |      | 5K | 10096 | 4.50 | 0.19 | 1.19 | 30950 | 1.89 | 0.03 | 1.16 | 263799 | 8.77 | 0.26 | 1.25 |
|         |      | 10K | 662 | 0.31 | 0.01 | 1.15 | 659 | 0.04 | 0.00 | 1.08 | 24005 | 0.86 | 0.02 | 1.13 |
|         |      | SP+Al |      |      |      |      |      |      |      |      |      |      |      |
|         |      | 1K | 48282 | 18.40 | 1.03 | 1.91 | 164518 | 9.89 | 0.00 | 1.00 | 359798 | 10.98 | 1.01 | 1.79 |
|         |      | 2.5K | 23912 | 9.51 | 0.52 | 2.20 | 60189 | 3.56 | 0.00 | 1.00 | 221439 | 7.16 | 0.08 | 1.70 |
|         |      | 5K | 3868 | 4.15 | 0.10 | 2.19 | 1708 | 0.11 | 0.00 | 1.00 | 4546 | 0.17 | 0.00 | 1.21 |
|         |      | 10K | 1376 | 0.50 | 0.04 | 2.22 | 422 | 0.03 | 0.00 | 1.00 | 1524 | 0.06 | 0.00 | 1.14 |
|         |      | SP+Au |      |      |      |      |      |      |      |      |      |      |      |
|         |      | 1K | 23955 | 9.53 | 0.53 | 2.20 | 60189 | 3.56 | 0.00 | 1.02 | 221438 | 7.16 | 0.08 | 1.68 |
|         |      | 2.5K | 19016 | 7.78 | 0.48 | 2.22 | 30468 | 1.84 | 0.00 | 1.02 | 153298 | 5.23 | 0.04 | 1.43 |
|         |      | 5K | 2163 | 0.78 | 0.05 | 2.19 | 1708 | 0.11 | 0.00 | 1.00 | 4546 | 0.17 | 0.00 | 1.21 |
|         |      | 10K | 832 | 0.29 | 0.02 | 2.50 | 422 | 0.03 | 0.00 | 1.00 | 1524 | 0.06 | 0.00 | 1.14 |
|         |      | SP+Al |      |      |      |      |      |      |      |      |      |      |      |
|         |      | 1K | 23 | 0.01 | 0.00 | 1.55 | 1 | 0.00 | 0.00 | 1.00 | 18 | 0.00 | 0.00 | 1.00 |
|         |      | 2.5K | 23955 | 9.53 | 0.53 | 2.20 | 60189 | 3.56 | 0.00 | 1.02 | 221438 | 7.16 | 0.08 | 1.68 |
|         |      | 5K | 19016 | 7.78 | 0.48 | 2.22 | 30468 | 1.84 | 0.00 | 1.02 | 153298 | 5.23 | 0.04 | 1.43 |
|         |      | 10K | 832 | 0.29 | 0.02 | 2.50 | 422 | 0.03 | 0.00 | 1.00 | 1524 | 0.06 | 0.00 | 1.14 |
|         |      | SP+Au |      |      |      |      |      |      |      |      |      |      |      |
|         |      | 1K | 23 | 0.01 | 0.00 | 1.55 | 1 | 0.00 | 0.00 | 1.00 | 18 | 0.00 | 0.00 | 1.00 |

Table 5: OC is OOV token count, OR is OOV sentence ratio, LC is lost character percentage (actual percentile, scaled by 100), and ML is mean lost character count across all OOV tokens.

to 102 character level subwords. Following this logic, we can also assume that anything above 102 can be claimed for actual subwords; this means even for a pure Korean corpus. Our method frees up 22,750 slots for actual subwords, instead of having to allocate it to completed characters which are unlikely to be seen.

To confirm that our method increases the elasticity of the vocabulary budget, we trained multiple sizes during the OOV robustness experiments to verify how small the vocabulary can be compacted. We compare the performance differences and characteristics with OOV in the exhaustive experiment table 5. We can see that the ratio of sentences with OOV, even on small vocabularies such as 1K, has comparable robustness against tokenizers with much larger vocabularies. The lower boundary of aligned is 109 subwords, and automaton (unaligned) is 116 subwords. Both boundaries were found by setting the coverage rate set to 0.994. As this is effectively an alphabet sized vocabulary, the number is only useful as a theoretical lower boundary limit, as this vocabulary would not be practically useful.

5.2.3. Subword Surface Length

Longer surface lengths result in shorter sequences, which prevent truncation or omission in training due to architecture limitations and also increase computation time in sequence models. We also see moderate gains in the subword surface length, as shown in table 6.

We see a small surface length jump when introducing align (Al) on top of the SentencePiece baseline, but the gain is not as significant as when using automata (Au), which nearly
doubles the surface length compared to other cases with the same vocabulary budget. Our contribution here is the increase in surface length relative to the allocated vocabulary size, which improves on the baselines.

| Model          | Count | Min | Max | Mean | Std |
|----------------|-------|-----|-----|------|-----|
| BERT           | 1568  | 1   | 5   | 1.32 | 0.62|
| XLM            | 2581  | 1   | 8   | 1.55 | 0.72|
| KoBERT         | 7378  | 1   | 9   | 2.12 | 0.93|
| BERTKr         | 11607 | 1   | 14  | 2.19 | 1.00|
| SP@1K          | 861   | 1   | 3   | 1.02 | 0.13|
| SP@2.5K        | 1902  | 1   | 5   | 1.31 | 0.57|
| SP@5K          | 2903  | 1   | 5   | 1.39 | 0.64|
| SP@10K         | 2748  | 1   | 3   | 1.02 | 0.15|
| SP+Al@350      | 150   | 0.4 | 1.2 | 0.96 | 0.91|
| SP+Al@1K       | 460   | 0.4 | 2   | 1.18 | 0.99|
| SP+Al@2.5K     | 1001  | 0.4 | 3.6 | 1.46 | 1.42|
| SP+Al@5K       | 1463  | 0.4 | 4.8 | 1.59 | 1.67|
| SP+Al@10K      | 1049  | 0.4 | 3.6 | 1.44 | 1.50|
| SP+Au@350      | 272   | 0.4 | 2.8 | 0.92 | 0.95|
| SP+Au@1K       | 845   | 0.4 | 5.2 | 1.32 | 1.57|
| SP+Au@2.5K     | 2125  | 0.4 | 6   | 1.71 | 1.93|
| SP+Au@5K       | 3514  | 0.4 | 6.4 | 1.90 | 2.05|
| SP+Au@10K      | 2466  | 0.4 | 6   | 1.72 | 1.88|

Table 6: Count of Korean subwords, minimum, maximum, average, and standard deviance of the token length for Korean subwords in each model’s vocabulary. For our methods, we have scaled it down by 2.5 for a fair comparison. SP is SentencePiece, and what comes after the @ is the size of the vocabulary. +Al denotes aligned, and +Au denotes automaton (unaligned).

6. Limitations and Future Work

Both methods suffer from one severe limitation; the reconstruction is only possible given that the model output is sane. This is analogous to the Unicode pair reconstruction problem which can be observed in the tokenizer used by GPT-2 (Radford et al., ) or other byte-level approaches(Gillick et al., 2016), as unless the output is an sane byte sequence which can be reconstructed into a Unicode character, the output is not only unusable, it also can break parsing the remainder of the sequence.

While hypothetically a mitigation for these cases can be put in by encoding checks for cases where the post-processor cannot reconstruct into the subword tokens, which in turn can be used as hints to discard erroneous output from the model in a generative task, we have not explored this direction within the scope of this work and leave it as a point for future exploration.

7. Conclusion

To summarize the contributions from our work, we start by defining the commonly overlooked problem in multilingual pre-training when including CJK languages, along with the computational budget, complexities, and inefficiencies associated with that.

We then propose a novel, tokenizer agnostic method, which supplements subword tokenizers in the context of Korean, which can be beneficial when combined with other methods involving pre-training large models. As this pre-processing only needs to be done once per unseen text, and it has no side-effects when used in polyglot corpora, the cost of applying this to any large scale multilingual pre-training is not only negligible, but it is also expected to learn better subwords and completely mitigate out-of-vocabulary issues in all downstream tasks for Korean, while reducing the character level budget needed to represent Korean.

Additionally, we propose a fully reversible method, which enables generative downstream tasks such as (but not limited to) machine translation, text summarization, and language modeling.

Finally, we demonstrate that our method improves the flexibility and performance of state-of-the-art methods while mitigating common OOV problems when training multilingual models. We hope our contributions will enable a cost-effective and straightforward path for including Korean in future multilingual research.

8. Bibliographical References

Chaudhary, A., Zhou, C., Levin, L., Neubig, G., Mortensen, D. R., and Carbonell, J. (2019). Adapting Word Embeddings to New Languages with Morphological and Phonological Subword Representations.

Cho, W. I., Kim, S. M., and Kim, N. S. (2019). Investigating an effective character-level embedding in korean sentence classification. arXiv preprint arXiv:1905.13656.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. oct.

Ding, C., Utiyama, M., and Sumita, E. (2018). Simplified abugidas. In ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers).

Gillick, D., Brunk, C., Vinyals, O., and Subramanya, A. (2016). Multilingual language processing from bytes. In 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2016 - Proceedings of the Conference.

He, H., Wu, L., Yang, X., Yan, H., Gao, Z., Feng, Y., and Townsend, G. (2018). Dual Long Short-Term Memory Networks for Sub-Character Representation Learning. In Advances in Intelligent Systems and Computing.

Howard, J. and Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. jan.

Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In EMNLP 2018 - Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Proceedings.

Lample, G. and Conneau, A. (2019). Cross-lingual Language Model Pretraining. jan.

Park, S., Byun, J., Baek, S., Cho, Y., and Oh, A. (2018). Subword-level word vector representations for Korean. In ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers).
Appendix

The content below was not of adequate length for the main paper.

8.1. Hyperparameters

The following hyperparameters were used to train the SentencePiece model. We used identity for the normalization rule, to prevent Unicode normalization composing them back together. Additionally, the maximum sentence length was set to 8000, and the model type was set to unigram. Character coverage was set proportionally to vocabulary size. We used 0.994 coverage for $|V| = 350$, 0.995 for $|V| = 1000$, 0.997 for $|V| = 2500$, 0.09985 for $|V| = 5000$, and 1.0 for $V = 10000$.

8.2. Automaton Reconstruction Pseudocode

This section describes the algorithm used for reconstruction in our automaton based method.

**Initialize**: $S \leftarrow$ INIT; Reset();

**foreach** $c \in C$, **given that** $c \in J$

**if** $S = \text{INIT}$

**if** $c \in J_v$

**if** $R_4$

emit($R_1, R_2, R_3$);

Reset(); $R_1, R_2 \leftarrow \{R_4, c\}$

else

**if** $R_4$

$R_3 = \text{combine}(R_3, R_4)$;

emit($R_1, R_2, R_3$); Reset();

$R_1 \leftarrow c$; $S \leftarrow \text{INTERIM}$

end

else **if** $S = \text{INTERIM}$

**if** $c \in J_v$

**if** $R_2$

emit($R_1, R_2, R_3$);

Reset(); $S \leftarrow \text{INIT}$;

else

$R_2 \leftarrow c$

else

if $R_2$

$R_3 \leftarrow c$; $S \leftarrow \text{END}$

else

emit($R_1, 0, 0$);

Reset(); $R_1 \leftarrow c$; $S \leftarrow \text{END}$

end

else **if** $S = \text{END}$

**if** $c \in J_v$

emit($R_1, R_2, 0$);

**if** $R_3$

Reset(); $R_1, R_2 \leftarrow \{R_3, c\}$; $S \leftarrow \text{INTERIM}$

else

Reset(); $S \leftarrow \text{INIT}$

else

$l \leftarrow \text{combine}(R_3, c)$;

**if** $R_3 = l$

emit($R_1, R_2, R_3$);

Reset(); $R_1 \leftarrow c$; $S \leftarrow \text{INTERIM}$

else

$R_4 \leftarrow c$; $S \leftarrow \text{INIT}$

end

end

**Algorithm 1**: Simplified Version of the automaton based reconstruction algorithm. $\leftarrow$ is equivalent to a MOV instruction. $l$ is a temporary scratch, which is actually R4 but named differently for clarity.

3497