Entropy-Based Subword Mining with an Application to Word Embeddings

Ahmed El-Kishky[1], **Frank Xu**[2], **Aston Zhang**[3], **Stephen Macke**[1], **Jiawei Han**[1]
The University of Illinois at Urbana Champaign
Shanghai Jiao Tong University
Amazon Inc.
{elkishk2,smacke, hanj}@illinois.edu[1], frankxu@sjtu.edu.cn[2], astonz@amazon.com[3]

Abstract

Recent literature has shown a wide variety of benefits to mapping traditional one-hot representations of words and phrases to lower-dimensional real-valued vectors known as word embeddings. Traditionally, most word embedding algorithms treat each word as the finest meaningful semantic granularity and perform embedding by learning distinct embedding vectors for each word. Contrary to this line of thought, technical domains such as scientific and medical literature compose words from subword structures such as prefixes, suffixes, and root-words as well as compound words. Treating individual words as the finest-granularity unit discards meaningful shared semantic structure between words sharing substructures. This not only leads to poor embeddings for text corpora that have long-tail distributions, but also heuristic methods for handling out-of-vocabulary words. In this paper we propose SubwordMine, an entropy-based subword mining algorithm that is fast, unsupervised, and fully data-driven. We show that this allows for great cross-domain performance in identifying semantically meaningful subwords. We then investigate utilizing the mined subwords within the FastText embedding model and compare performance of the learned representations in a downstream language modeling task.

1 Introduction

In recent years, distributed continuous word representations have become a popular tool for providing a low-dimensional, alternative representation to traditional one-hot bag of words (Rumelhart et al., 1988; Elman, 1990). These word-embedding vectors are typically a real-valued vector of dimensionality much smaller than the vocabulary size of a corpus. In addition to computational efficiency of working with low-dimensional representations, distributed representations have

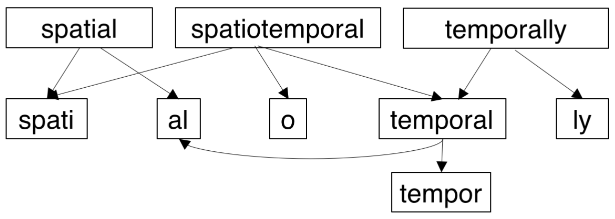


Figure 1: Hierarchical segmentation of words *spatial, spatiotemporal, temporally* into subwords.

been shown to capture syntactic and semantic regularities and have been shown to boost the performance in tasks such as text classification, sequential classification, sentiment analysis, and machine translation (Mikolov et al., 2013c; Joulin et al., 2017; Huang et al., 2015; Tang et al., 2014; Zou et al., 2013). Many different methods have been proposed to derive these continuous representations from large, unlabeled, text corpora (Collobert and Weston, 2008; Mikolov et al., 2013a,b).

While distributed continuous representations have helped push the state-of-the-art in a variety of NLP tasks, because most text corpora have long-tail distributions, embeddings in the long-tail are often of poor quality due to infrequency. This is even worse for out-of-vocabulary words which are all given the same constant embedding vector because no information is available to infer a meaningful representation. To address this deficiency, we propose to mine smaller subword structures that form the base syntactic unit and leverage the discovered subwords for generating better-quality word embeddings. As seen in Figure 1, morphologically-rich words often contain semantically meaningful subwords that are shared among many words. Understanding the semantic meaning of these subwords can be used to infer the meaning of words that contain them. With this motivation, we propose SubwordMine, an algorithm for mining semantically-meaningful subwords from corpus vocabulary. We utilize the mined subwords as the base-unit for embedding and combine them to construct a word's distributed vector representation. The resultant word embeddings are robust to data-sparsity due to word infrequency and can be constructed on many out-of-vocabulary words.

We state and analyze the problem in Section 2, followed by our proposed solution in Section 3 where we present the key components of our solution, subword mining and subword-based word embeddings. In Section 4, we review the related work. Then we evaluate the proposed solution in Section 5, conclude in Section 6, and present future directions in Section 7.

2 SubwordMine Framework

We formalize and analyze the task of extracting subword structure and propose a framework for entropy-based subword mining.

2.1 Preliminaries

The input is a corpus W, consisting of $|W|$ words: $W = w_1, \ldots, w_{|W|}$. From this corpus, we construct a vocabulary of unique words, V, of size $|V|$ such that $\forall w \in W, w \in V$. In addition, the v_{th} word is a sequence of $|v|$ characters: $c_{v,i}, i = 1, \ldots, |v|$. For convenience we index all the unique characters that compose the input vocabulary with C characters and $c_{v,i} = x, x \in \{1, \ldots, C\}$ means that the i_{th} character in v_{th} word is the x_{th} character in the character vocabulary. Given an input corpus consisting of a word sequence and a vocabulary list of unique words, our goal is to segment the vocabulary list to identify human-interpretable and semantically meaningful subwords, then utilize these subwords for parameter sharing when learning distributed word representations from the corpus.

Definition 1 (Subword Formalization) *We formally define subwords and other necessary notation and terminology as follows:*

- *A* subword *is a sequence of contiguous characters:* s=$\{c_{v,i}, ...c_{v,i+n}\}$ $n > 0$
- *A* partition *over* v_{th} *word is a sequence of subwords:* $\mathcal{G}_v$ =$(s_{v,1}, \ldots, s_{v,G_v})$ $G_v \geq 1$ *s.t. the concatenation of the subword instances is the original word.*

In Definition 1 we formalize a subword and the resultant partition from segmenting a vocabulary word into subwords. In addition we outline the desired properties of the resultant subword as well as the mining and embedding framework as follows:

- The subwords extracted are semantically-meaningful and human-interpretable.
- Utilizing these subwords improves word embeddings.
- The overall method is computationally efficient.
- The number of subwords generated is comparable to the vocabulary size.

2.2 SubwordMine Framework

To extract subwords that satisfy our desired requirements, we propose a framework that can be divided into two sequential steps: 1) subword pattern mining 2) subword segmentation. Our process for transforming each word in the input vocabulary word to a high-quality 'bag-of-subwords' involves creating a subword vocabulary, and then using these subwords to hierarchically segment each word in the vocabulary. By applying an information-theoretic metric to detect candidate subword boundaries, we identify candidate subwords within each vocabulary word. From this candidate pool, we then apply an unsupervised dynamic programming segmentation algorithm to select a subset of these subwords that best segment the word. After inducing a partition on each word, we can recursively segment each subword to an arbitrary level of subword granularity. The resultant subwords from the hierarchical segmentation can then be used for word embedding.

The goal of frequent subword pattern mining is to collect aggregate statistics on subword patterns for use in the word segmentation algorithm. For each character-ngram that appears more than once in the vocabulary, there is the potential for parameter sharing via that candidate subword. Additionally the frequency counts of these subwords will be used for entropy-boundary computation to identify potential subword candidates. These candidates are inputted to the word-segmentation algorithm that attempts to apply Occam's Razor by selecting subwords that maximally cover each word using the fewest number of subwords. Each subword can then be recursively segmented into further subwords.

3 Methodology

We present a subword mining algorithm that, given an input vocabulary list V, segments each vocabulary word into, non-overlapping, character-ngrams. Our method is purely data-driven relying on character co-occurence statistics allowing for good cross-domain performance on a variety of scientific datasets. Additionally the method operates directly on an input vocabulary list, forgoing any corpus-level statistics. This allows for more scalable subword extraction as passes over large corpora are unnecessary. From a high-level perspective, the subword segmentation algorithm can be decomposed into the following steps:

1. Mine candidate subword counts and compute relevant co-occurence statistics.
2. Apply SubwordMine to segment each word.
3. Recurse for finer-grained subword segmentation.

We apply an entropy-based scoring function to identify subword boundaries: generating candidate subwords. Given a collection of subwords, the next step in the framework is to apply a dynamic-programming algorithm to segment each word into subwords. The framework proceeds to recursively segment each subword in the segmentation. We will discuss these steps in greater detail in the next subsections.

3.1 Subword Vocabulary Generation

Our segmentation of words into subwords relies on the idea of subword compositionality. That is, the input vocabulary can be constructed by composing subwords drawn from a smaller subword vocabulary. As such we introduce a two-step approach for creating the initial subword vocabulary: prefix and suffix generation followed by root-word generation.

Prefix & Suffix Generation

The first step in creating the subword vocabulary is to generate a set of high-quality prefixes and suffixes. The method is based on the principle that a high-quality prefix or suffix can be measured by a high level of unpredictability in transition to longer substrings from the current substring state. For example, following the prefix "pr", most prefixes transition to the character "e" with high probability forming a prefix "pre". On the other

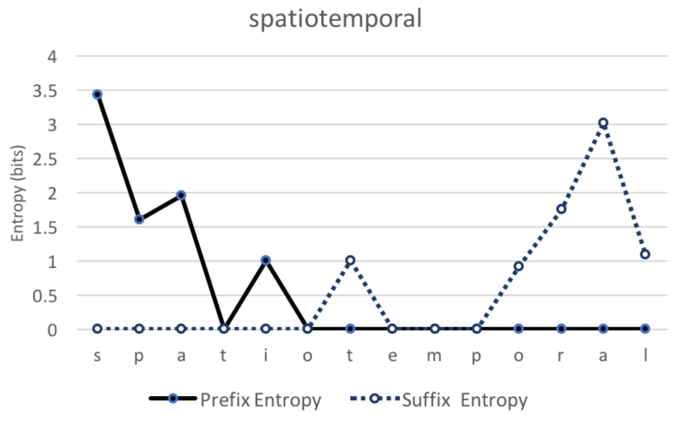


Figure 2: Entropy for candidate prefixes and suffixes in the word "spatiotemporal" from a DBLP titles dataset.

hand, transitioning from "pre" to a longer prefix is not as predictable as a large number of words contain the prefix "pre" followed by a variety of root words starting with different characters. We identify these high-unpredictability boundaries using the concept of information entropy to score the predictability of each prefix or suffix boundary (Shannon, 2001).

Let v be a word consisting of $|v|$ characters and s_i be a prefix of w ending at the i_{th} character of w. For each candidate prefix boundary i for $i \in [1 \ldots |v|]$, the information entropy of the prefix is computed as follows:

$$\mathcal{E}(i) = -\sum_{j=1}^{C} \mathsf{P}(s_i \oplus c_j | s_i) \times \log_2 \mathsf{P}(s_i \oplus c_j | s_i) \tag{1}$$

Where $\oplus$ denotes the binary concatenation of two subwords and the transitional probabilities between a prefix and the prefix with the next character appended is estimated by:

$$\mathsf{P}(s_i \oplus c_j | s_i) = \frac{f(s_i \oplus c_j)}{f(s_i)} \tag{2}$$

and $f(s_i)$ denotes the frequency of a prefix s_i in the input vocabulary list. The entropy of suffixes can, without loss of generality, be similarly computed by reversing each word in the vocabulary and treating each suffix as a prefix.

The information entropy of each possible prefix and suffix in the vocabulary is computed in linear time using a prefix tree data structure to store counts over prefixes. Given entropy scores for each prefix and suffix, scores are computed for each candidate split point in each word. Under the entropy scoring of prefixes and suffixes, we identify *local maxima* in entropy as candidate boundaries for prefixes and suffixes. That is entropy of prefixes of one-character shorter and one-character longer should be lower than a candidate prefix boundary. This is intuitive as under our principle of compositionality assumption, complex words are formed by concatenating subword structures. As such, given an incomplete subword, the next character can easily be predicted, but given a complete subword, any number of new subwords can be concatenated to the completed subword increasing the unpredictability and thus entropy. These high-entropy positions thus serve as a strong indicator of subword boundaries. As seen in Figure 2, for the word "spatiotemporal", candidate prefixes and suffixes are found at boundaries exhibit a local maxima in entropy. For "spatiotemporal", candidate prefixes are "spa" and "spati" while candidate suffixes include "al" and "temporal".

Root Word Generation

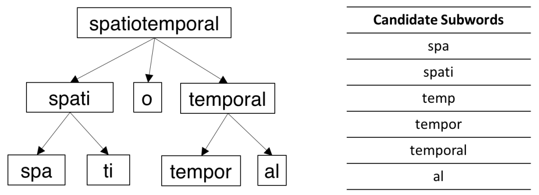
While utilizing entropy-scoring, it is possible to detect subword structures that occur at the beginning or end of a word, often many words contain subword structure between prefixes and suffixes. For each prefix and suffix candidate identified in a word, it is possible to generate candidate root word by stemming the word and removing prefixes and suffixes. This creates a high-quality pool of rootwords to be used in conjunction with prefixes and suffixes for segmenting the vocabulary.

Example 1 (Root Extraction) *Removing prefixes and suffixes yields candidate root words.*

[pre] + *<u>authenticat</u>* + ***[ion]***

The characters grouped together by ***[]*** *are prefixes and suffixes. When removed, the remaining underlined character-sequence represent candidate root words.*

As seen in Example 1, when stripping the possible prefixes and suffixes of a word, the remaining character sequence is considered a candidate root word. We apply some filtering conditions for each candidate root to test the viability as a shareable root. These include: 1) a minimum support of two within the vocabulary 2) the entropy boundary of each rootword must be non-zero. Additionally, for each word in the vocabulary, after stripping prefixes and suffixes, the candidate root words that meet the root word constraints are extracted and added to the subword vocabulary.



Candidate Subwords
spa
spati
temp
tempor
temporal
al

(a) Parsimonious Segmentation (b) Candidate Subwords

Figure 3: Segmentation of the word "spatiotemporal" using disjoint interval covering.

3.2 Parsimonious Subword Segmentation

In the previous section, we introduce an unsupervised method of subword generation based on an entropy-based predictability metric for boundary detection. In this subsection we introduce an unsupervised segmentation algorithm that, utilizing a given subword vocabulary, segments a word into subwords. Our algorithm first identifies candidate subwords from the subword vocabulary within a word, then selects a subset of these candidate subwords that best segment the word. The main insight behind the unsupervised segmentation is a per-word implementation of Occam's Razor. That is, according to the preference for parsimonious hypotheses, we posit that each word is composed of the *fewest number of subwords that maximally cover the word.*

Example 2 (Parsimonious Segmentation) *According to parsimonious segmentation, candidate segmentations are scored based on word coverage and number of subwords used for coverage.*

Segmentation	# Subwords	Coverage
[spa] + *tio*+ ***[temporal]***	*2*	*11*
[spati] + *o* + ***[temporal]***	*2*	*13*
[spati] + *o* + ***[tempor]*** + ***[al]***	*3*	*13*
...		
[spa] + *tio* ***[tempor]*** + ***[al]***	*3*	*11*

The highlighted row displays the maximally parsimonious subword segmentation.

As seen in Figure 3, for each word a set of subwords present in the target word are identified and recursive segmentation is performed to separate the word into subwords. Example 2 demonstrates how the candidates are used to segment the target word under the *parsimony criterion*. Subsets of non-overlapping candidate subwords are used to segment and the most parsimonious segmentation is selected. Because there are a $\mathcal{O}(2^{|A_v|})$ number of possible subsets of candidate subwords, direct enumeration of each segmentation quickly proves intractable for even a modest number of candidate subwords. To identify the most parsimonious segmentation, we abstract out our parsimonious subword segmentation task into a general problem we dub *Disjoint Interval Covering* and demonstrate that this problem can be solved via dynamic programming in linear time.

We formalize the disjoint interval covering problem as follows:

Definition 2 (Disjoint Interval Covering) *Given an input $N \in \mathbb{N}$ and a set A of pairs $(a, b) : a, b \in \{1 \dots N\} \times \{1 \dots N\}$ and $a < b$, find the smallest subset $B \subseteq A$ such that $|\bigcup_{x \in B} x|$ is maximized, $|B|$ is minimized, and $\forall x, y \in B : x \neq y \Rightarrow x \cap y = \varnothing$.*

As seen in Definition 2, the input is a set of pairs A and a positive integer N. Within the segmentation perspective, these refer to position index boundary pairs for candidate subwords and the word length. Given these inputs, the objective is to select a minimum subset of disjoint subwords whose length maximize coverage of the word.

$$F(j) = \max_0 \min_1 \left\{ \begin{array}{ll} (0,0), & j < 1 \\ F(j{-}1), & j \geq 1 \\ \max_0 \min_1\limits_{(i,j) \in B} \{F(i{-}1)_0 + (j{-}i{+}1), F(i{-}1)_1{+}1\}, & j \geq 1 \end{array} \right\} \tag{3}$$

We define a recurrence to the disjoint interval covering problem in Equation 3. This recurrence posits that the segmentation that maximally covers the word is either the solution for the current word minus the ending character, or the max-covering, min-subword solution utilizing all subwords that have a right boundary index equal to the index of the end of the word. With proper memoization, it is evident that for a word of size $|v|$, there are $|v|$ subproblems to solve. In addition, because each interval's right boundary corresponds to the word size, each interval is iterated over a constant number of times. As such, for word v, the total, memoized complexity of this segmentation is $\mathcal{O}(v + |A_v|)$ where A_v indicates the pre-segmentation subwords that are substrings of word v.

Algorithm 1: DP Parsimonious Segmentation (DP)

```
Input: Word $v$, Subword Intervals $A_v$
Output: Optimal segmentation $S$
n[0] ← 0; c[0] ← 0; p[0] ← null;
for $j := 1$ to $N_v$ do
    num ← n[j-1]; cov←c[j-1]; pair ← p[j-1];
    for $(i, j) \in A_v$ do
        cov′ ← c[i-1]+(j-i+1)
        num′ ← n[$i-1$]+1
        if cov′ > cov then
            cov ← cov′; num ← num′;
            pair ← $(i, j)$;
        end
        if cov′=cov ∧ num′<num then
            num ← num′; pair ← $(i, j)$
        end
    end
    n[j] ← num; c[j] ← cov; p[j] ← pair;
end
return p
```

Algorithm 1 presents the subword segmentation algorithm. The algorithm takes as input a word and a collection of intervals corresponding to index boundaries of candidate subwords within the word. It then proceeds to select a set of intervals that maximally cover the word while utilizing the fewest number of intervals. Solutions to subproblems are memoized as to avoid repeated computation.

3.3 Hierarchical Subword Segmentation

In Subsection 3.1 we introduced the concept of utilizing high-entropy boundaries to create a subword vocabulary, and in Subsection 3.2 we introduce an algorithm for segmenting words into subwords based on the principle of parsimonious disjoint interval covering. In this subsection we demonstrate a high-level overview on how applying these two methods can be used to hierarchically segment words into multi-granular subwords.

Following the steps from Subsection 3.1, an initial subword vocabulary is created. Within the vocabulary, we differentiate between prefixes, suffixes, and root words. As seen in Algorithm 2, Line 2, each subword found in the input word is mapped to an interval indicating its boundary indices within the word with the condition that prefix intervals must start at the beginning of the word, suffix intervals must terminate at the end of the word, and root word intervals can be located at any position within the word. In addition, the complete word is not included (to ensure the word segments to smaller subwords). The algo-

Algorithm 2: Segmentation Algorithm (SEGMENT)

```
Input: Word $v$, Subword Vocabulary SW
Output: Set of subwords of $v$
1  output ← $\{v\}$
2  $A_v \leftarrow \{(i,j)$ for $v_i \ldots v_j \in SW$ and j-i $\neq |v|\}$
3  if $A_v = \varnothing$ then
4      return output
5  end
6  segmented ← DP(w, $A_v$)
7  for subword ∈ segmented do
8      output ∪ SEGMENT(subword, SW)
9  end
10 return output
```

rithm terminates if the word cannot be further segmented. Otherwise, the word is segmented with the dynamic programming parsimonious segmentation algorithm. Each subword is then treated as a word and recursively segmented by the algorithm, and the collection of all subwords from segmentation are outputted.

3.4 Word Embedding

To efficiently utilize our mined subwords to improve upon word embeddings, we modify the FastText model for word embeddings to use our extracted subwords (Bojanowski et al., 2016).

FastText utilizes the skip-gram objective with negative sampling yielding the following objective (for simplicity, $\ell(x) = \log(1 + \exp(-x))$):

$$\sum_{x=1}^{W} \Big[\sum_{c \in \mathcal{C}_x} \ell(s(w_x, w_c)) + \sum_{t \in \mathcal{N}_{x,c}} \ell(-s(w_x, t)) \Big] \tag{4}$$

The scoring function is then adapted to incorporate subword information as follows:

$$s(w_x, w_c) = \sum_{p \in w_x} \mathbf{z}_p^\intercal \mathbf{v}_c \tag{5}$$

which equates to a simple summation over subword embedding vectors.

4 Related Works

There have been many attempts at automatic substructure extraction from words. These techniques generally fall into one of three families: scoring based on segment predictability, identifying subwords based on discovering similar and dissimilar word parts, and optimization methods.

In morphological analysis of relating phonemes to morphemes, segment predictability has been suggested as a potential identifying characteristic for detecting subword structure. An early quantitative metric proposed was the number of different variations of subwords following a subword sequence whereby a high number of variations indicates a subword boundary (Harris, 1970). While this work provided influential insight into useful metrics for subword-detection, the main objective was developing a scoring function for identifying candidate subwords, not segmentation. Following this line of work, many methods have extended the variation boundary approach to identify frequent morphemes from text corpora (Hafer and Weiss, 1974). A similar method adopts the metric to identify frequent affixes (Déjean, 1998). Both these methods seek to identify a small subset of high-quality, high frequency subwords from each corpus, prioritizing precision over recall. Other methods propose slight variations to the predictability metric such as drops in transitional probabilities (Saffran et al., 1996).

Deviating from predictability-based methods, several subword detection methods have been proposed for detecting subwords by comparing words and identifying similar and dissimilar parts. One such method performs alignment from the left and right edge of words (Neuvel and Fulop, 2002) identifying common subwords. Another method adds words to a trie in correct order and reverse order to identify leading and trailing frequent subwords (Schone and Jurafsky, 2001). Unfortunately both these methods can only identify prefix and suffix subwords, ignoring many internal subwords. Unlike these methods, our subword segmentation is position insensitive and can identify subwords that occur in any position in a word.

Opting for an optimization over a scoring perspective, a variety of methods have been proposed. One such method models segmentation through the minimum description length principle (Creutz and Lagus, 2002). This method attempts to minimize both the vocabulary while maintaining the likelihood of the corpus data.This method was successfully applied to languages such as Turkish (Sak et al., 2010). Unfortunately, unlike methods that take the vocabulary as input, these family of optimization methods must make several passes over the corpus. This not only adds significant runtime and may discourage use as a preprocessing step before embedding, but can also be intractable for large text corpora. Other methods apply a maximum likelihood approach to identifying subwords (also called wordpieces) and has been successfully applied to a variety of NLP tasks (Wu et al., 2016; Schuster and Nakajima, 2012). And similarly the byte-pair compression algorithm has been used to identify subwords for neural machine translation tasks (Sennrich et al., 2015). Both these methods construct a fixed-size subword vocabulary to construct each word as a sequence of subwords.

Many attempts have been proposed to address data-sparsity when learning distributed word representations. These methods posit that individual words have semantically meaningful attributes that are shared among other words allowing for parameter sharing between vocabulary words. One such method proposes a factored neural language model where words are represented as a set of features including subword information (Alexandrescu and Kirchhoff, 2006). Another method attempts to incorporate morphological information into the word embeddings by adding morphological similarity features into a neural network along with the context features (Cui et al., 2015). This method while similarly motivated, does not leverage subword structure but instead utilizes the embeddings of "morphologically similar" words in the embedding process. While this may seem appealing, identifying morphologically similar words can be an expensive process as it requires a search over the entire vocabulary which may be prohibitive during on-the-fly computation out-of-word vocabulary. In addition, this method may miss important morphological cues such as negation subwords. When extended to use subword information, this model assumes subwords are already provided, this requires manual identification of subwords which can be an expensive human-powered task, especially in domain-specific settings or new languages (Qiu et al., 2014). Along similar motivation, a method has been proposed where given an input of morphologically annotated data, log-bilinear models are trained to jointly predict context words and its morphological tag (Cotterell and Schütze, 2015). Despite displaying superior embedding performance on German corpora, this method once again requires human-labeling for tagging words. This limits applicability to domain-specific corpora and new languages where labeled data is scarce or expensive to obtain. The method that is is closest to our approach is the extension of FastText enriched with subword information (Bojanowski et al., 2017). This method extends the standard skip-gram model but utilizes character-ngram subword embeddings for parameter sharing. The major differences between SubwordMine and this method is that FastText embedding utilizes all character n-grams of user-specified lengths for subword embedding and performs a simple sum over their representations while SubwordMine performs unsupervised segmentation and applies a novel attention mechanism to combine the subword representations into word representations. Finally, utilizing subword information was shown to improve performance in machine translation (Sennrich et al., 2016).

Another spectrum of approaches address word sparsity through the use of characters as the base unit for embedding. Some approaches treat each word as a sequence of characters. and apply recurrent neural networks to the task of language modeling (Bojanowski et al., 2015; Sutskever et al., 2011). Other related models apply convolutional neural networks directly on characters (Kim et al., 2016).

5 Experimental Results

We introduce the datasets used and methods for comparison. We then describe our evaluations for both subword extraction and for word embedding performance.

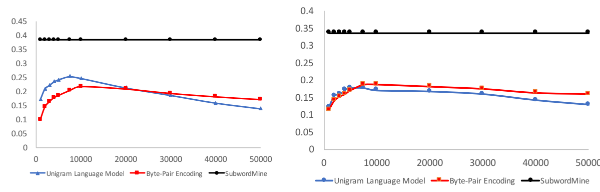
5.1 Datasets and methods for comparison

Datasets

We use the following three datasets for evaluation purpose:

- **DBLP Abstracts**. Computer science abstracts containing 529K abstracts, 186K unique words, and 39M tokens.
- **DBLP Title**. Titles of computer science papers published in 20 conferences containing 44K titles, 5.5K unique words, and 351K tokens.
- **PubMed Abstracts**. Abstracts of research papers obtained from from PubMed Central containing 421K abstracts, 334K unique words and 5.8M tokens.

For baseline comparison methods to our proposed SubwordMine algorithm we utilize a unigram language model segmentation of 'wordpieces' and byte-pair encoding segmentation as



(a) PubMed Abstracts (b) DBLP Abstracts

Figure 4: Accuracy in extracting Greek and Latin root words while varying subword vocabulary size.

Model	Extraction Accuracy	
	PubMed	DBLP
Byte-Pair Encoding	0.2180	0.1881
Unigram LM	0.2535	0.1782
SubwordMine	**0.3831**	**0.3363**

Table 1: Accuracy in automatically extracting Greek and Latin root words.

described in the related works. For comparable methods for embedding we utilize FastText, a proposed variation of the Skip-Gram objective that utilize subword information, and modify FastText to use a variety of subword segmentations.

5.2 Subword Extraction Accuracy

To evaluate the effectiveness of our proposed unsupervised segmentation algorithm at extracting semantically meaningful subwords, we collect a list of approximately three-thousand English words and their Greek or Latin roots. For each segmentor trained on each dataset, we test to see if the segmentation correctly extracts the ground truth root.

As seen in Figure 4, the accuracy of extracting the root words varies with the vocabulary size for both BPE and the Unigram LM method. As seen in Table 2, for the optimal vocabulary size for both methods, we see SubwordMine still outperforms both methods.

5.3 Perplexity

We investigate the benefits of using semantically meaningful subwords for parameter-sharing during when learning word embeddings. For each set subword-enriched embedding vectors, we learn a language model and evaluate its quality by computing the language model perplexity on a DBLP title dataset. The model used for language modeling is an LSTM variant of a recurrent neural network with two hidden layers and 600 hidden units per layer and regularized with dropout with 0.2 probability. The RNNs are unrolled for 35 steps and the batch size is set to 20. Parameters are learned using Adagrad with a gradient clipping of 1. Each language model instance trained on a training set partition consisting of 80% of the DBLP data and evaluation of perplexity was computed for each model on an independent test set consisting of 10% of the data after selecting the best performing iteration of the model on the remaining validation set.

Model	Perplexity	
	Untuned	Tuned
SkipGram	378.45	245.01
BPE-FastText	356.29	210.59
ULM-FastText	324.07	220.73
FastText	370.68	212.88
SubwordMine	**320.65**	**207.84**

Table 2: Test perplexity on the language modeling task for DBLP titles dataset. Evaluation is performed with fixed pre-trained embeddings, and embedding tuning.

The results are summarized in Table 2. Because our implementation performs minimal data cleaning and does not drop infrequent or out-of-vocabulary words, we expect the resulting perplexity should be relatively higher than cleaned-datasets but directly comparable among the differing methods (Bojanowski et al., 2016).

For the LSTM model, we observe that across all sub-word enriched embeddings perform better in language modeling over traditional skip-gram. Additionally, for both the untuned and tuned settings, SubwordMine segmentations improve test-perplexity over all other subword extraction method including original FastText's enumeration of all possible subwords. This is likely due to the sheer number of enumerated subwords and subword embeddings generated by FastText which may be more difficult to learn.

6 Conclusions

In this paper, we propose a computationally efficient method of segmenting vocabulary lists into semantically meaningful subwords. We demonstrate experimentally that utilizing the subwords in word embeddings in scientific domain corpora improves embedding quality as measured by a downstream language modeling task.

7 Future Works

Currently SubwordMine applies unsupervised segmentation. While this has be shown to yield high-quality segmentations, one natural extension is to incorporate human-labeling and perform supervised segmentation. Another area of work is to utilize subword structures in a variety of sequential modeling tasks which could improve tasks such as entity typing, relation extraction, and machine translation where substructures can provide valuable signals.

References

Andrei Alexandrescu and Katrin Kirchhoff. 2006. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 1–4. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

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

Piotr Bojanowski, Armand Joulin, and Tomas Mikolov. 2015. Alternative structures for character-level rnns. *Prof. ICLR*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Ryan Cotterell and Hinrich Schütze. 2015. Morphological word-embeddings. In *HLT-NAACL*, pages 1287–1292.

Mathias Creutz and Krista Lagus. 2002. Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning-Volume 6*, pages 21–30. Association for Computational Linguistics.

Qing Cui, Bin Gao, Jiang Bian, Siyu Qiu, Hanjun Dai, and Tie-Yan Liu. 2015. Knet: A general framework for learning word embedding using morphological knowledge. *ACM Transactions on Information Systems (TOIS)*, 34(1):4.

Hervé Déjean. 1998. Morphemes as necessary concept for structures discovery from untagged corpora. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, pages 295–298. Association for Computational Linguistics.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Margaret A Hafer and Stephen F Weiss. 1974. Word segmentation by letter successor varieties. *Information storage and retrieval*, 10(11-12):371–385.

Zellig S Harris. 1970. From phoneme to morpheme. In *Papers in Structural and Transformational Linguistics*, pages 32–67. Springer.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Armand Joulin, Edouard Grave, and Piotr Bojanowski Tomas Mikolov. 2017. Bag of tricks for efficient text classification. *EACL 2017*, page 427.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *AAAI*, pages 2741–2749.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013c. Linguistic regularities in continuous space word representations. In *hlt-Naacl*, volume 13, pages 746–751.

Sylvain Neuvel and Sean A Fulop. 2002. Unsupervised learning of morphology without morphemes. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning-Volume 6*, pages 31–40. Association for Computational Linguistics.

Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Co-learning of word representations and morpheme representations. In *COLING*, pages 141–150.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. 1988. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Jenny R Saffran, Elissa L Newport, and Richard N Aslin. 1996. Word segmentation: The role of distributional cues. *Journal of memory and language*, 35(4):606–621.

Haşim Sak, Murat Saraclar, and Tunga Güngör. 2010. Morphology-based and sub-word language modeling for turkish speech recognition. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 5402–5405. IEEE.

Patrick Schone and Daniel Jurafsky. 2001. Knowledge-free induction of inflectional morphologies. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–9. Association for Computational Linguistics.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 5149–5152. IEEE.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. *Proc. ACL*.

Claude E Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. 2014. Learning sentiment-specific word embedding for twitter sentiment classification. In *ACL (1)*, pages 1555–1565.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Will Y Zou, Richard Socher, Daniel Cer, and Christopher D Manning. 2013. Bilingual word embeddings for phrase-based machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1393–1398.