Parsimonious Morpheme Segmentation with an Application to Enriching Word Embeddings

[1]Ahmed El-Kishky, [2]Frank Xu, [3]Aston Zhang, [1]Jiawei Han
[1]The University of Illinois at Urbana-Champaign, [2]Carnegie Mellon University, [3]Amazon AI
{elkishk2,hanj}@illinois.edu, frankxu@cmu.edu, astonz@amazon.com

***Abstract*—Traditionally, many text-mining tasks treat individual word-tokens as the finest meaningful semantic granularity. However, in many languages and specialized corpora, words are composed by concatenating semantically meaningful subword structures. Word-level analysis cannot leverage the semantic information present in such subword structures. With regard to word embedding techniques, this leads to not only poor embeddings for infrequent words in long-tailed text corpora but also weak capabilities for handling out-of-vocabulary words. In this paper we propose MorphMine for unsupervised morpheme segmentation. MorphMine applies a parsimony criterion to hierarchically segment words into the fewest number of morphemes at each level of the hierarchy. This leads to longer shared morphemes at each level of segmentation. Experiments show that MorphMine segments words in a variety of languages into human-verified morphemes. Additionally, we experimentally demonstrate that utilizing MorphMine morphemes to enrich word embeddings consistently improves embedding quality on a variety of of embedding evaluations and a downstream language modeling task.**

I. Introduction

Decomposing individual words into finer-granularity morphemes is a necessary step for automatically preprocessing concatenative vocabularies where the number of unique word forms is very large. While linguistic approaches can be used to tackle such segmentation, such rule-based approaches are often tailored to specific languages or domains. As such, data-driven, unsupervised methods that forgo linguistic knowledge have been studied [7], [18]. Typically, these methods focus on segmenting words by applying a probabilistic model or compression algorithms to a full text corpus. The resultant morphemes from these methods have been primarily shown to improve neural machine translation [26], [40].

One natural application to utilize these semantically meaningful morphemes is distributed word representation. There are many advantages to using distributed continuous word representations as an alternative to one-hot bag of words [12], [34] since this leads to a dimensionality much smaller than the vocabulary size of a corpus. It has been shown that working with low-dimensional representations not only demonstrates computational efficiency, but also captures syntactic and semantic regularities while boosting the performance in text classification, sequential classification, sentiment analysis, and machine translation [21], [22], [31], [43], [45]. As such, many methods have been developed to learn these word representations from large, unlabeled text corpora [5], [29], [30].

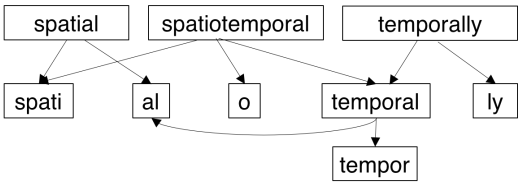


Fig. 1: Hierarchical segmentation of words.

Despite many advances, unsupervised learning of distributed representations can struggle in learning adequate vectors for infrequent words. This problem is ubiquitous because most text corpora demonstrate long-tail distributions in relation to word frequency, with often $40\% - 60\%$ of words in a vocabulary appearing just once in a corpus [25]. Naturally, many methods fail to produce meaningful embeddings for unseen (out-of-vocabulary) words. Using morphemes for parameter sharing not only bolster training data for infrequent words but also allow for constructing meaningful word embeddings for unseen words.

What differentiates our method from others is *extracting morphemes at multiple granularity*. As seen in Figure 1, morphologically-rich words share semantically-meaningful morphemes. Larger morphemes carry more semantic meaning, but are often infrequent within the vocabulary and discarded in favor of more frequent finer-grained morphemes in other methods. Yet, including both fine and coarse-grained morphemes, can better semantically tie the meanings of words that share them. With this motivation, we propose MorphMine, a continuation on preliminary work [11]. We formalize the novel methodology by framing the morpheme segmentation as entropy-boundary identification and segmentation with a parsimony criterion. We introduce a global resegmentation to refine and improve the segmentation after the initial segmentation. Finally, we evaluate our method on a variety of datasets and tasks in multiple language and demonstrate how multi-granular morphemes can be used for enriching word embeddings for robustness to data-sparsity.

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

II. 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 vth 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 $c_{v,i} = x, where\ x \in \{1, \ldots, C\}$ means that the ith character in v^{th} word is the xth 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 morphemes, then utilize these morphemes for parameter sharing when learning distributed word representations from the corpus.

Definition 1 (Morpheme Formalization):

- A *morpheme* is a sequence of characters: $m = \{c_{v,i}, ..., c_{v,i+n}\}\ \ where\ n > 0$
- A *partition* over vocabulary word v is a sequence of morphemes: $\mathcal{G}_v = (m_{v,1}, \ldots, m_{v,G_v})\ \ where\ G_v \geq 1$ s.t. the concatenation of the morphemes is the original word.

In Definition 1 we formalize a morpheme and the resultant partition from segmenting a word into morphemes. In addition we outline the desired properties of the framework as follows:

1) extracts semantically meaningful, human-interpretable at multiple granularity
2) the method is general and applies to words on a variety of languages
3) enriching word morphemes improves word embeddings
4) the overall method is computationally efficient

A. The MorphMine Framework

At a high-level, our proposed framework can be summarized into two sequential steps: (1) mining candidate morpheme patterns and character co-occurence statistics, and (2) performing word segmentation into finer-grained morphemes. In step one, by applying an information-theoretic metric to detect candidate morpheme boundaries, we identify candidate morphemes within each vocabulary word. These morphemes are propagated to other words and pruned to ensure high-quality. For step two, from this candidate pool, we then apply an unsupervised dynamic programming segmentation algorithm to select a subset of these morphemes that best segment each word. Segmentation and partition induction further prune away low-quality morpheme candidates leaving a high-quality morpheme vocabulary. After inducing a partition on each word, we can recursively segment each morpheme to finer granularity. Applying this two-step process maps each word in the input vocabulary to a set of high-quality morphemes. The resultant morphemes from the hierarchical segmentation can then be used for downstream NLP and text analysis tasks.

The main objective in morpheme pattern mining is to collect aggregate statistics on morpheme patterns that can be used to score and reason about the quality of candidate morphemes. These statistics are then used in the word segmentation algorithm. For each character n-gram that appears more than once in the vocabulary, there is a potential for parameter sharing via the candidate morpheme as it appears in multiple vocabulary words. Additionally the frequency counts of these morphemes will be used for entropy-boundary computation to identify and score potential morpheme candidates. These candidates are input to the word-segmentation algorithm that attempts to apply Occam's Razor by positing that using the fewest morphemes in the segmentation best segments each word [16]. This process is then applied recursively to each morpheme to obtain finer-grained morphemes.

By inducing a partition over each vocabulary word, we effectively transform each word into a bag-of-morphemes. These morphemes can be shared among other words within the vocabulary and model the belief that words that share morphemes, share semantic meaning. This is done by individually embedding each morpheme; these morpheme embeddings are then combined to form the final word embedding. Because a word embedding is constructed from the embeddings of constituent morphemes, words that share constituent morphemes will be *partially* constructed from similar morpheme embeddings.

We expound upon our morpheme-mining algorithm and its evaluation in a popular embedding framework in Section III.

III. Methodology

Given an input vocabulary list V, MorphMine segments each word into non-overlapping character n-grams (morphemes). Our method is non-parametric, hierarchical and data-driven, allowing for good cross-domain performance without incorporating domain-specific knowledge or linguistic rulesets. The entire morpheme segmentation can be performed as an easy preprocessing step to the vocabulary for downstream text-related tasks. To learn a morpheme vocabulary and segment an input vocabulary, MorphMine performs the following steps: (1) mine morpheme pattern counts and compute entropy statistics, (2) apply parsimonious segmentation to identify the best *locally-consistent* segmentation, (3) recompute morpheme counts after segmentation to ensure *global-consistency* and maximize parameter sharing of morphemes, and (4) re-segment using refined morpheme vocabulary counts.

We apply an entropy-based scoring function to identify morpheme boundaries: generating candidate morpheme vocabulary. Given this collection of morphemes and their counts, the next step is to apply a dynamic-programming algorithm to segment each word into high-quality morphemes. For each word, the parsimonious segmentation identifies the *most-likely* segmentation using the *fewest number of morphemes*. This step discards a large number of lower quality candidates morphemes from our vocabulary and allows for a more-accurate estimate of morpheme counts. Using the refined vocabulary, we can then re-segment and improve the overall quality of segmentation. The re-segmentation biases towards selecting locally-consistent segmentations that globally-optimize for

morpheme parameter sharing. That is, the resegmentation favors morphemes used in the segmentation of other words in the vocabulary. Finally, the resultant collection of morphemes for each word can be utilized to enrich word embeddings.

A. Morpheme Vocabulary Generation

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

1) Prefix & Suffix Generation.: We posit that prefixes and suffixes can be identified through the concept of transition predictability.

Definition 2 (Transition Predictability): Transition predictability is a quantification of being able to predict the next character in a word given a prefix.

Previous works have attempted to quantify Definition 2, by using number of character choices following a prefix in a vocabulary [9], [17], [19], [35]. For example, many words begin with the prefix, "*pre*" such as, *prepaid*, *preview*, *presoak*, etc. Given the large number of words with the prefix "pre", the transition from "pr" to "pre" predictable, but "pre" to a longer prefix is not as predictable as many words have "pre" followed by a variety of root words.

Unfortunately, using raw counts to identify high-unpredictability boundaries for prefixes does not generalize to large vocabularies and different languages as the character count is arbitrary. As such we propose a metric on the normalized distribution of character choices: information entropy [41]. Let v be a word consisting of $|v|$ characters and m_i be a prefix of v ending at the i_{th} character of v. For each candidate prefix boundary i for $i \in [1 \dots |v|]$, the prefix transition unpredictability can be quantified with information entropy. As the transition between a prefix and longer prefixes can be modeled as a multinomial of support size C, the character vocabulary, we use the multinomial distribution entropy:

$$\begin{aligned} H(X) = &-\log(n!) - n\sum_{j=1}^{C} p_j \log(p_j) + \\ &\sum_{j=1}^{C}\sum_{x_j=0}^{n} \binom{n}{x_j} p_j^{x_j}(1-p_j)^{n-x_j}\log(x_j!) \\ = &-\sum_{j=1}^{C} p_j \log(p_j), \text{ when n=1} \end{aligned}$$

This is the entropy of a multinomial over support C for n independent trials each of which leads to a success for exactly one of the C characters. Because we consider a single trial, $n = 1$, we simplify it into entropy of the categorical distribution. For the prefix m_i:

$$H(m_i) = -\sum_{j=1}^{C} \mathsf{P}(m_i \oplus c_j | m_i) \times \log_2 \mathsf{P}(m_i \oplus c_j | m_i),$$

where $\oplus$ denotes the binary string concatenation of two strings and the transitional prefix probability is estimated as:

$$\mathsf{P}(m_i \oplus c_j | m_i) = \frac{f(m_i \oplus c_j)}{f(m_i)}$$

and $f(m_i)$ denotes the frequency of a prefix m_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.

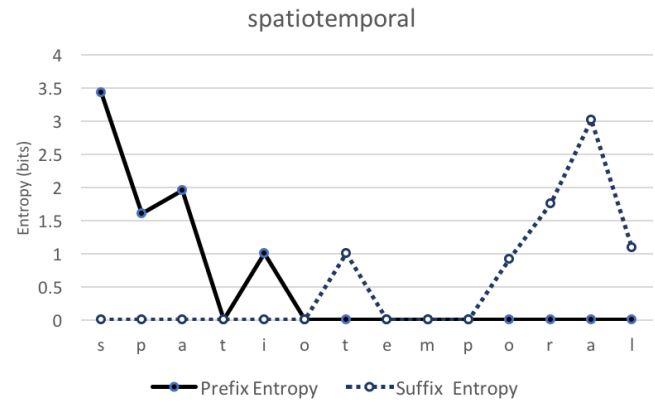
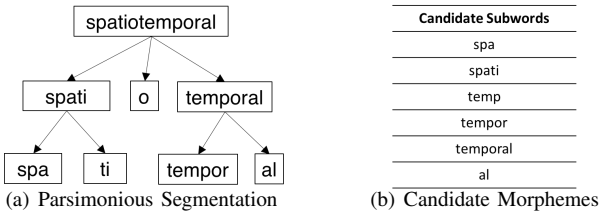


Fig. 2: Prefix and suffix transition entropy.

The information entropy of each possible prefix and suffix in the vocabulary is computed in *linear time* with relation to unique vocabulary size 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 a prefix 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 morpheme structures. As such, given an incomplete morpheme, the next character can easily be predicted, but given a complete morpheme, any number of new morphemes can be concatenated to the completed morpheme increasing the unpredictability and thus entropy. These high-entropy positions thus serve as a strong indicator of morpheme 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".



(a) Parsimonious Segmentation (b) Candidate Morphemes

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

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

Example 1 (Root Extraction): Removing prefixes and suffixes yields candidate roots.

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

[pre] + <u>authentication</u>

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 combinations of 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, and (2) the minimum root length of four. Additionally, for each word in the vocabulary, after stripping prefixes and suffixes, the candidate root words that meet the constraints are added to the morpheme vocabulary.

B. Parsimonious Morpheme Segmentation

After generating a morpheme vocabulary using entropy-based predictability metric for boundary detection, we segment words into morphemes, utilizing this morpheme vocabulary. The algorithm first identifies candidate morphemes from the morpheme vocabulary within a word, then selects a subset of these candidate morphemes that best segment the word. The main insight 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 morphemes that maximally cover the word*.

As seen in Figure 3, morphemes present in the target word are identified and recursive segmentation is performed to segment the word into morphemes. Example 2 demonstrates how the candidates are used to segment the target word under the *parsimony criterion*.

Example 2 (Parsimonious Segmentation): Segmentations are scored based on word coverage and the number of morphemes.

Segmentation	***# Morphs***	***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 morpheme segmentation.

Subsets of non-overlapping candidate morphemes are used in segmentation, and the most parsimonious segmentation is selected. Because the possible subsets of candidate morphemes form a power set, direct enumeration of each segmentation quickly proves computationally slow for even a modest number of candidate morphemes. To identify the most parsimonious segmentation, we abstract our parsimonious morpheme 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 3 (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 3, 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 morphemes and the word length. Given these inputs, the objective is to select a minimum subset of disjoint morphemes that maximally cover the word. That is, select a set of disjoint morpheme whose combined length is as close as possible to the word length.

$$F(j) = \max_{0} \min_{1} \left\{ \begin{array}{ll} (0,0), & j < 1 \\ F(j-1), & j \geq 1 \\ \max_{0} \min_{1} \{F(i-1)_0 + (j-i+1), F(i-1)_1 + 1\}, & j \geq 1 \\ \quad (i,j) \in A & \end{array} \right\} \tag{1}$$

We define a recurrence to the disjoint interval covering problem in Equation 1. 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-morpheme solution utilizing all morphemes 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 morphemes that are substrings of word v, making our overall framework of linear complexity – $\mathcal{O}(V)$.

Algorithm 1 presents the morpheme segmentation algorithm. The algorithm takes as input a word and a collection of intervals corresponding to index boundaries of candidate

Algorithm 1: DP Parsimonious Segmentation (DP)

Input: Word v, morpheme 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) ∈ 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
```

morphemes 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. While the algorithm returns a memoization list of best segmentations that terminate at each index, proper backstracking can construct all possible parsimonious segmentations. In the next subsection we demonstrate how to select among equally-parsimonious segmentations.

1) Maximum Likelihood Scoring: While Algorithm 1 identifies the most parsimonious segmentation, the algorithm often returns many segmentations with equal parsimony. As such, after applying Algorithm 1, maximum likelihood is used to select *the most likely segmentation* among these candidate segmentations.

Given the previous counts of candidate morphemes obtained, it is simple to compute the most likely segmentation among the candidate set of parsimonious segmentations given an independence assumption. Given a segmentation (partition of morphemes) over word v, $\mathcal{G}_v$, one can calculate the likelihood over the partition:

$$\mathcal{L}(\mathcal{G}_v) = \prod_{m \in \mathcal{G}_v} \mathsf{P}(m) \propto \prod_{m \in \mathcal{G}_v} f(m)$$

The independence assumption yields and discarding the normalization yields a simple product over each morpheme count $f(m)$ in the partition. This follows as all parsimonious partitions have the same number of morphemes and as such, the normalization constants for the probabilities should be the same for all parsimonious segments. One additional important constraint we place on our most-likely partition is that, during training and learning of the morpheme vocabulary, the most-likely partition cannot have any morphemes that occur only once in the vocabulary. That is: $\forall m \in \mathcal{G}_v : f(m) > 1$. This ensures that each learned morpheme is shared at least with another word. As seen in Example 3, the largest product of counts is selected as the best segmentation. By applying the restriction that all morphemes must be shared at least once, MorphMine filters poor segmentations such as "*incompletenes + s*" where the morpheme "*incompletenes*" only appears once in the vocabulary.

Example 3 (Most-likely Segmentation): Most likely segmentation from candidates.

Segmentation	$f(m_1)$	$f(m_2)$	*Likelihood Score*
[incompletenes] + [s]	1	2072	2072
[incomplete] + [ness]	4	115	660
[in] + [completeness]	659	4	2636
[incomp] + [leteness]	4	2	8

The highlighted row displays the most likely segmentation.

The mostly likely segmentation "*in + completeness*" is selected and in further steps, "completness" will be recursively decomposed into smaller morphemes "complet" and "ness" parsimoniously.

C. Local Segmentation.

Subsection III-A introduced the concept of utilizing high-entropy boundaries to create a morpheme vocabulary, and Subsection III-B introduced an algorithm for segmenting words into morphemes based on the principle of parsimonious disjoint interval covering and tie-breaking with maximum likelihood. In this subsection we demonstrate a high-level overview on how to apply these two methods to hierarchically segment words into multi-granular morphemes.

Algorithm 2: Segmentation Algorithm (SEGMENT)

Input: Word v, morpheme Vocabulary SW
Output: Set of morphemes of v

```
1  output ← {v}
2  A_v ← {(i, j) for v_i ... v_j ∈ SW and j-i ≠ |v|}
3  if A_v = ∅ then
4      return output
5  end
6  segmented ← DP(w, A_v)
7  for morpheme ∈ segmented do
8      output ∪ SEGMENT(morpheme, SW)
9  end
10 return output
```

Following the steps from Subsection III-A, an initial morpheme vocabulary is created. Within the vocabulary, we differentiate between prefixes, suffixes, and root words. As seen in Algorithm 2, Line 2, each morpheme 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 morphemes). The algorithm terminates if the word cannot be further segmented. Otherwise, the word is segmented with the dynamic programming parsimonious segmentation algorithm.

Each morpheme is then treated as a word and recursively segmented; the collection of all morphemes from segmentation are output.

D. Global Resegmentation

The parsimonious segmentation selects the most-likely locally-consistent segmentation of a word. Yet because each word is segmented independently, a morpheme that is present in two different words may not be selected because parsimonious segmentation is performed on both words independently. To address this, after performing one segmentation we utilize the resultant segmentation to refine the morpheme counts and prune infrequent morphemes from the morpheme vocabulary. Using this refined morpheme vocabulary and a more accurate morpheme count estimation, each word is re-segmented. The resultant segmentation is not only performed with a smaller morpheme vocabulary, but also favors the morphemes that other words have selected in their own parsimonious segmentations, creating a global consistency for the overall vocabulary segmentation.

Example 4 (Re-segmentation with refined counts.): After one pass through the vocabulary and segmenting with parsimonious segmentation. Morpheme counts are re-computed using the resultant segmentations.

Segmentation	***Counts***$_1$	***ML***$_1$	***Counts***$_2$	***ML***$_2$
[**bit**] + [**emporal**]	5,6	**30**	3,1	3
[**bi**] + [**temporal**]	4,6	24	3,5	**15**

While initial counts and scores (Counts$_1$, ML$_1$) determine the locally optimal segmentation. After recomputing the morphemes post initial segmentation, refined counts and scores are computed (Counts$_2$, ML$_2$). The white row displays the initial best segmentation, while the grey row shows the best segmentation after refining morpheme counts.

As seen in Example 4, in the first segmentation, the locally-consistent parsimonious segmentation favors the incorrectly segmented "*bit + emporal*" with likelihood 30 over "*bi + temporal*" with likelihood 24 as the likelihood is higher for the former. After one round of segmentation, it is apparent that the morpheme "*emporal*" was only selected once out of the possible six occurrences, while "*temporal*" was selected five out of six word segmentations. Resegmentation with these refined counts helps choose the correct segmentation "*bi + temporal*" with likelihood score 15 over 3. With this refined segmentation, these morphemes can be used in the morpheme-enriched word embedding learning.

E. Morpheme-Enriched Word Embedding

To efficiently utilize our mined morphemes to improve upon word embeddings, we modify the FastText model for word embeddings to use our extracted morphemes [3] to enrich *infrequent or out-of-vocabulary* words. As explained in the FastText paper, it is often the longest subword that captures the most semantic meaning. As such, we take each and every node in our word segmentation representing morphemes at every granularity and directly input the morphemes extracted from this layer to enrich each word in the vocabulary.

We begin with a brief review of FastText, and then demonstrate integrating morphemes in place of the standard FastText enumerated subwords. First, we note that FastText utilizes the skip-gram objective with negative sampling yielding the following objective (for simplicity, $\ell(x) = \log(1+\exp(-x))$):

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

where w_x is the xth word in the corpus, $\mathcal{C}_x$ denotes the set of context words within a window of word w_x, and $\mathcal{N}_{x,c}$ denotes the set of negative examples sampled from the vocabulary. The scoring function is then adapted to incorporate morpheme information as $s(w_x, w_c) = \sum_{m \in w_x} \mathbf{z}_m^{\intercal} \mathbf{v}_c$ where each $\mathbf{z}_m$ denotes a morpheme embedding vector and the scoring function is a summation over morpheme embedding vectors in a dot-product with the context word vector. While FastText incorporates all contiguous substrings of lengths three to seven as morphemes in the scoring function, we posit that many of these morphemes are semantically not meaningful and, as such, degrade the overall quality of the learned embeddings. We claim that directly incorporating meaningful morphemes extracted by MorphMine for each word and summing over each morpheme's embedding results in higher quality distributed representations.

IV. Related Work

In morphological analysis, predictability has been suggested for detecting morpheme structure. An early quantitative metric proposed was the number of different variations of morphemes following a morpheme sequence whereby a high number of variations indicates a morpheme boundary [19]. While this work provided influential insight into useful metrics for morpheme-detection, the main objective was developing a scoring function for identifying candidate morphemes, not segmentation. Following this line of work, were methods to identify frequent morphemes and affixes [9], [17], [35]. These methods identify a high-precision but low-recall subset of morphemes. Similarity measures have been proposed for detecting affixes by comparing words and identifying similar and dissimilar parts. These methods utilize a variety of techniques including edge-alignment, adding words and their reverse to tries [32], [38]. Unfortunately, these methods can only identify prefix and suffix morphemes, ignoring morphemes. One model segments words by applying the minimum description length principle to minimize the vocabulary while maintaining the likelihood of the corpus data [7], [37]. Other fixed-vocabulary methods apply a unigram language model approach to identifying morphemes (also called wordpieces) and has been successfully applied to a variety of NLP tasks [39], [44]. Similarly, the byte-pair compression algorithm has been used to identify morphemes for neural machine translation tasks [40].

To address data-sparsity when learning word embeddings, some methods apply a factored neural language model where words are represented as a set of features including morpheme information [1]. Other methods add morphological similarity features into a neural network along with the context features [8], [33]. Other methods take morphologically annotated data and train log-bilinear models to jointly predict context words and morphological tags [6]. The method we utilize for our embeddings is FastText [3]. While FastText utilizes all the possible character n-grams up to certain length for enrichment, we only utilize high-quality morphemes in MorphMine. Finally, many methods have utilized characters as the base unit for embedding. Some approaches treat each word as a sequence of characters and apply RNNs or convolutional networks [4], [23], [42].

V. Experimental Results

We introduce the datasets used and methods for comparison. We then evaluate our method on a morpheme segmentation task, a variety of embedding tasks, and a downstream language modeling task.

Datasets

- **English, German, and Turkish Vocabularies and Segmentations**. This dataset consists of three vocabulary lists in English, German, and Turkish with 156K, 290K and 90K unique vocabulary words, respectively. Each list is accompanied by approximately 1500 ground-truth segmentations consisting of a vocabulary word and its segmentation into constituent morphemes. These ground-truth segmentations were annotated as part of the MorphoChallenge [27].
- **English, German, and Turkish Wikipedia Corpora**. This dataset consists of three subsets of Wikipedia for English, German, and Turkish Wikipedia and consisting of 116M, 162M, and 52M tokens. These corpora are used for training unsupervised word embeddings and for training a language model.
- **English, German, and Turkish Word Similarity Pairs**. This dataset consists of collections of annotated word-similarity pairs in three languages. For English, we evaluate on the WS-353 data, a collection of 353 pairs of English words that have been assigned similarity ratings by human annotators, SimLex, a collection of 999 word pairs annotated via Amazon Mechanical Turk, and finally the Stanford Rare Words similarity set (RW) consisting of 2034 rare word pairs. [15], [20], [28]. For German, we operate on canonical translations of the the WS-353 and SimLex datasets [2]. For Turkish we evaluate on the AnlamVer word similarity dataset consisting of 500 word-pairs annotated by 12 human annotators [13].
- **English, German, and Turkish Word Analogies**. Collections of annotated word analogies in three languages. For English, we evaluate on the Google analogy dataset consisting of 19544 analogy question pairs where $8,869$ are semantic and $10,675$ syntactic (i.e. morphological) questions. [29]. For German, we operate on the German translation of the English Google analogy dataset [24]. For Turkish, counterparts of the Google analogy question set was created and contains over 2K analogy tasks.

The vocabulary lists and gold-standard segmentations are used to evaluate each method's ability to extract human-verified morphemes in an unsupervised manner. The human-curated word analogies and word similarity pairs help verify the effect of incorporating various morphemes in the unsupervised word embedding process. Finally, the Wikipedia corpora subsets are used to train the morpheme-enriched word embeddings and evaluate the benefit of morpheme enrichment on a downstream language modeling task.

Baselines

As a baseline for segmentation, we utilize a unigram language model segmentation of "word-pieces" and byte-pair encoding segmentation as described in the related work [39], [44]. We also compare against a state-of-the-art unsupervised morpheme segmentation tool Morfessor [7]. Finally, we compare against a variant of MorphMine that forgoes global consistency whereby each word is re-segmented after recomputing morpheme counts after the initial segmentation.

For baseline embedding methods, we utilize FastText, a proposed variation of the Skip-Gram objective that utilize subword-level information, and modify FastText to incorporate each method's segmentations to enrich word embedding. We enrich FastText with each of the morpheme segmentation baselines to compare against MorphMine enriched emebeddings. With no morpheme enrichment, FastText formulation means that it reduces to Word2Vec which we also compare against.

A. Subword Extraction Accuracy

We evaluate each morpheme segmentation algorithm at identifying human-annotated segmentations in three languages: English, Turkish, and German. We report precision, recall and F1 scores for each method. When evaluating, true-positives are indicated with a valid exact match between the extracted morpheme and the gold-standard.

In Table I, we report the performance of each segmentor at successfully extracting human-annotated morphemes. As both BytePair Encoding and Unigram-LM require a morpheme vocabulary size parameter, for these methods, we perform a parameter sweep and report results from the highest performing run. Across all three languages, variants of MorphMine outperform with respect to F1 score. Further analysis shows this is primarily due to a higher recall. In comparison to MorphMine without global refinement, we see that implementing global refinement generally improves performance as seen in English and German and in the case of Turkish, performance between the MorphMine variants were overall comparable.

B. Word Similarity Task

We evaluate the embeddings on a word similarity task. The ground truth data consists of pairs of words and a human-annotated similarity score averaged across all human evaluations. The scores are computed via the cosine similarity between each word's vector representation and results are

Dataset	English			German			Turkish		
Method	P	R	F1	P	R	F1	P	R	F1
BPE	0.5527	0.3989	0.4634	0.5637	0.4131	0.4768	0.7626	0.2808	0.4104
ULM	0.7473	0.5992	0.6651	0.5827	0.5040	0.5405	**0.8731**	0.3216	0.4701
Morfessor	0.7537	0.6513	0.6987	**0.6803**	0.5616	0.6153	0.69104	0.3710	0.4828
MorphMine-NoRefine	0.8255	0.6503	0.7275	0.5717	**0.7520**	0.6399	0.5894	0.5024	**0.5424**
MorphMine	**0.8345**	**0.6977**	**0.7600**	0.6014	0.7373	**0.6624**	0.5341	**0.5497**	0.5417

TABLE I: Morpheme Segmentation Performance.

quantified through Spearman's rank correlation coefficient between the gold standard and the cosine similarity score. To evaluate performance of the morpheme-based embeddings to infer OOV words, we evaluate similarity on an English rare-words similarity dataset.

As seen in Table II, subword-based methods that utilize morpheme and subword level information outperform SkipGram that forgoes any. Additionally, methods that discriminately generate these morphemes outperform FastText that indiscriminately generate all subwords. Finally, while most subword enriched embeddings perform well on word similarity, MorphMine shines particularly in the similarity task on rare words where it outperforms all baselines. This is likely because MorphMine generates morphemes at multiple granularity which is more likely semantically link a rare word to a frequent word via a semantically-meaningful morpheme. This performance gap is even higher for out-of-vocabulary words were MorphMine significantly outperforms all other baselines at the word similarity task.

C. Word Analogy Task

We next evaluate on a word analogy task of the form "A is to B" as "C is to D", where D is predicted from the vocabulary based on its embedding vector. We use analogy datasets used in previous literature for English, German, and Turkish embedding evaluation [24], [29], [36].

Dataset	English		German		Turkish
Method	**Sem**	**Syn**	**Sem**	**Syn**	**Sem+Syn**
SkipGram	**68**	65	**63**	46	41
BPE	65	68	61	50	43
ULM	67	70	62	51	43
FastText	52	75	59	**53**	43
Morfessor	64	75	61	52	44
MorphMine	67	**78**	61	**53**	**47**

TABLE III: Word analogies.

As seen in Table III, embeddings that utilize subword information perform better at syntactic analogies than SkipGram word embeddings without subword information. This does not extend to semantic analogies whereby utilizing subword-information seems to cause a deterioration in performance. This is intuitive as words without valid morphemes learn noisy embeddings when false morphemes are identified and used to enrich their representation. This is seen in the performance gap between FastText and SkipGram on semantic analogies whereby FastText's large number indiscriminate subwords degrades the quality of the final embedding. This degradation is mitigated by utilizing more-refined morpheme methods such as BytePair Encoding, Unigram-LM, Morfessor, and MorphMine. Overall, embeddings enriched with MorphMine morphemes demonstrate superior syntactic performance to all baselines while demonstrating comparable semantic performance to SkipGram. This supports our intuition that utilizing more subwords is useful, but only when they are of high quality; indiscriminately generating all enumerations of subword degrades quality.

D. Language Modeling Perplexity

As recent embedding evaluations have stressed the importance of evaluating embeddings not only on artificial tasks such as word similarities but also on downstream tasks, we evaluate on a downstream language modeling task [14]. We generate a language model with embedding vectors from the Wikipedia corpora and then evaluate by computing the perplexity on a held-out portion of the corpus *unseen in both the embedding phase and modeling phase*. We use an LSTM with two hidden layers, 600 hidden units per layer regularized with dropout with 0.2 probability, unrolled for 35 steps, and 20 batch size. Parameters are learned using Adagrad with a gradient clipping of 1 for 10 epochs. Each instance is trained on 80% of the data with a 10% test and 10% validation set.

Dataset	English	German	Turkish
Method	**Perplexity**	**Perplexity**	**Perplexity**
SkipGram	159	381	996
BPE	157	375	955
ULM	157	372	952
FastText	158	376	972
Morfessor	155	370	948
MorphMine	**154**	**367**	**940**

TABLE IV: Language modeling task

The results are summarized in Table IV. Because experiments have minimal data cleaning do not drop infrequent or OOV words, the resulting perplexity is relatively higher than cleaned-datasets but directly comparable among the differing methods [3]. We observe that across all segmentation-based morpheme-enriched embeddings perform better in language modeling over traditional skip-gram. In contrast, FastText's indiscriminate enumeration of all possible morphemes appears to perform much poorer in this task. Finally, MorphMine outperforms the other morpheme enriched baselines. This may be due to MorphMine utilizing morphemes of mutliple granularity which closely capture semantic meaning of rare and OOV words at the largest granularity.

Method	English				German		Turkish
Dataset	**WS-353**	**SimLex**	**RW-Frequent**	**RW-OOV**	**WS-353**	**SimLex**	**AnlamVer**
SkipGram	0.72	**0.28**	0.36	–	0.58	0.26	0.45
BPE	0.72	**0.28**	0.41	0.33	0.59	**0.28**	0.47
ULM	**0.74**	**0.28**	0.41	0.35	0.60	**0.28**	0.47
FastText	0.70	0.26	0.34	0.32	0.58	0.26	0.46
Morfessor	**0.74**	**0.28**	0.44	0.35	0.60	**0.28**	0.48
MorphMine	**0.74**	**0.28**	**0.46**	**0.42**	**0.62**	**0.28**	**0.49**

TABLE II: Multilingual word similarity.

Word	BPE	ULM	Morfessor	MorphMine
vandalism	van + dal + ism	van + dal + ism	van + dal + ism	vandal + ism
truncate	trun + cate	trun + cate	truncate	truncat + e
truncated	trun + cat + ed	trun + cat + ed	truncat + ed	truncat + ed
truncating	trun + cat + ing	trun + cat + ing	truncat + ing	truncat + ing
troubleshooting	trouble + shoot + ing	trouble + shoot + ing	trouble + shoot + ing	troubleshoot + ing + trouble + shoot

TABLE V: Select segmentations from different subword segmentation algorithms.

E. Segmentation Case Study

In Table V, we present hand-selected segmentations. Unlike other methods, MorphMine identifies large morphemes shared among words in the vocabulary in addition to the more frequent smaller morphemes. For example, the words *"truncate", "truncated", "truncating"* all share a common root, but all methods except for MorphMine are reluctant to identify *"truncat"* as a valid morpheme by removing 'e' from *truncate*. As such, all other methods fail semantically link these three words. Additionally, for '*vandalism*', most methods attempt to recognize "*van*" as a morpheme as it is a valid word, while MorphMine's parsimony criterion merges this into "*vandal*", which although not present in the vocabulary, is a valid word. Finally, given words such as *"troubleshooting"*, MorphMine's segmentation at multiple granularities captures "*troubleshoot*", which all other methods further decompose, losing much semantic meaning.

F. Scalability

From a high-level perspective, MorphMine consists of two separate steps: (1) mining and learning a high-quality candidate morpheme set from an input vocabulary and (2) utilizing the learned model to segment each word into morphemes. We can empirically estimate the expected runtime of each step of MorphMine by analyzing runtime as a function of input size.

As seen in Figure 4, mining the morpheme vocabulary appears to grow linearly with vocabulary size. We verify this by computing the coefficient of determination, R^2 to show how well a linear function fits the data. Morpheme mining and segmentation regressions yielded an R^2 of 0.989 and 0.991 respectively. This strongly suggests a linear relationship between input vocabulary size and runtime. As empirically Heap-Herdan's law has shown that vocabulary grows sublinearly in relation to corpus size, these results indicate that performing MorphMine segmentation on an input vocabulary as a preprocessing step adds negligible computational overhead [10].

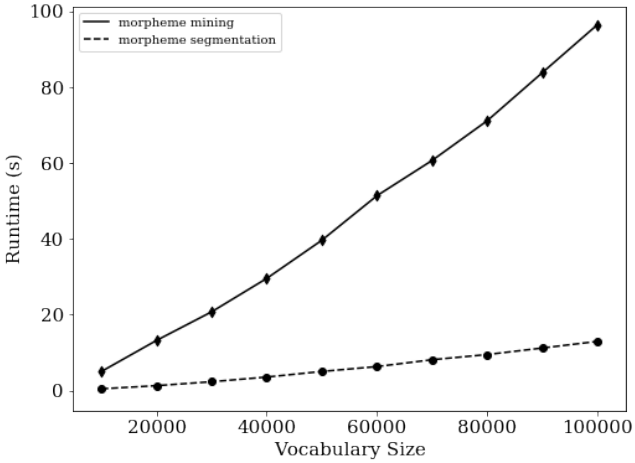


Fig. 4: Decomposition of morpheme segmentation algorithm into unsupervised morpheme mining then vocabulary segmentation.

VI. Conclusions

In this study, we propose a pattern-mining method of segmenting vocabulary into smaller morphemes and demonstrate experimentally on three languages that the method recovers ground-truth morphemes beyond state-of-the-art. By integrating the morphemes in a popular subword-enriched embedding algorithm, we verify that semantically-meaningful morphemes at multiple granularity can benefit word embeddings as evidenced through superior performance on a word analogy and word similarity task. This is especially true for inferring embeddings for infrequent or out-of-vocabulary words. Finally, we demonstrate that enriching embeddings with high-quality morphemes improves language modeling as evidenced through better held-out perplexity on a language modeling task.

VII. Acknowledgements

Research was sponsored in part by U.S. Army Research Lab. under Cooperative Agreement No. W911NF-09-2-0053 (NSCTA), DARPA under Agreements No. W911NF-17-C-0099 and FA8750-19-2-1004, National Science Foundation IIS 16-18481, IIS 17-04532, and IIS-17-41317, DTRA HDTRA11810026, and grant 1U54GM114838 awarded by NIGMS through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (www.bd2k.nih.gov). Any opinions, findings, and conclusions or recommendations expressed in this document are those of the author(s) and should not be interpreted as the views of any U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] Andrei Alexandrescu and Katrin Kirchhoff. Factored neural language models. In *NAACL-HLT*, pages 1–4. ACL, 2006.

[2] Siamak Barzegar, Brian Davis, Manel Zarrouk, Siegfried Handschuh, and André Freitas. Semr-11: A multi-lingual gold-standard for semantic similarity and relatedness for eleven languages. In *LREC*, 2018.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *TACL*, 2017.

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

[5] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, pages 160–167. ACM, 2008.

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

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

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

[9] Hervé Déjean. Morphemes as necessary concept for structures discovery from untagged corpora. In *NeMLaP/CoNLL*, pages 295–298. ACL, 1998.

[10] Leo Egghe. Untangling herdan's law and heaps' law: Mathematical and informetric arguments. *Journal of the American Society for Information Science and Technology*, 58(5):702–709, 2007.

[11] Ahmed El-Kishky, Frank Xu, Aston Zhang, Stephen Macke, and Jiawei Han. Entropy-based subword mining with an application to word embeddings. In *SCLeM*, pages 12–21, 2018.

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

[13] Gökhan Ercan and Olcay Taner Yıldız. Anlamver: Semantic model evaluation dataset for turkish-word similarity and relatedness. In *COLING*, pages 3819–3836, 2018.

[14] Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. Problems with evaluation of word embeddings using word similarity tasks. In *RepEval*, pages 30–35, 2016.

[15] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. *ACM Transactions on information systems*, 20(1):116–131, 2002.

[16] Hugh G Gauch. *Scientific method in practice*. Cambridge University Press, 2003.

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

[18] Harald Hammarström and Lars Borin. Unsupervised learning of morphology. *Computational Linguistics*, 37(2):309–350, 2011.

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

[20] Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2015.

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

[22] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *EACL*, pages 427–431, 2017.

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

[24] Maximilian Köper, Christian Scheible, and Sabine Schulte im Walde. Multilingual reliability and" semantic" structure of continuous word spaces. In *IWCS*, pages 40–45, 2015.

[25] András Kornai. *Mathematical linguistics*. Springer Science & Business Media, 2007.

[26] Taku Kudo. 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, 2018.

[27] Mikko Kurimo, Sami Virpioja, Ville Turunen, and Krista Lagus. Morpho challenge competition 2005–2010: evaluations and results. In *ACL Special Interest Group on Computational Morphology and Phonology*, pages 87–95. ACL, 2010.

[28] Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113, 2013.

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

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[31] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *NAACL-HLT*, pages 746–751, 2013.

[32] Sylvain Neuvel and Sean A Fulop. Unsupervised learning of morphology without morphemes. In *Workshop on Morphological and phonological learning-Volume 6*, pages 31–40. ACL, 2002.

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

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

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

[36] Gürkan Sahin. Classification of turkish semantic relation pairs using different sources. *International Journal of Computer Engineering and Information Technology*, 8(10):196, 2016.

[37] Haşim Sak, Murat Saraclar, and Tunga Güngör. Morphology-based and sub-word language modeling for turkish speech recognition. In *ICASSP*, pages 5402–5405. IEEE, 2010.

[38] Patrick Schone and Daniel Jurafsky. Knowledge-free induction of inflectional morphologies. In *NAACL*, pages 1–9. ACL, 2001.

[39] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *ICASSP*, pages 5149–5152. IEEE, 2012.

[40] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, pages 1715–1725, 2016.

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

[42] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *ICML*, pages 1017–1024, 2011.

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

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

[45] Will Y Zou, Richard Socher, Daniel Cer, and Christopher D Manning. Bilingual word embeddings for phrase-based machine translation. In *EMNLP*, pages 1393–1398, 2013.