Word segmentation using grouped semantic networks

by

Misha Schwartz

A thesis submitted in conformity with the requirements
for the degree of Masters
Graduate Department of Computer Science
University of Toronto

Abstract

Word segmentation using grouped semantic networks

Misha Schwartz
Masters
Graduate Department of Computer Science
University of Toronto
2018

This paper introduces the WuGS (Word segmentation using Grouped Semantic networks) algorithm for training word segmentation models that use parallel translated corpora to infer semantic and morphological similarities between words. These similarities are used to break down the words of a language into morphemes, the minimally semantically significant component parts of a language. The WuGS algorithm builds models in the form of grouped semantic networks which allows for the classification of words according to shared root morphemes in a way that respects allomorphy. We evaluate these models on three languages and show that these models perform on par with two existing baseline models and that parallel corpora are a good source of semantic data for this task. We also show that a reliance on semantic data results in an overly conservative approach to morphological segmentation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The words in this sentence are not indivisible semantic units of language. Just as sentences can be divided into words, words can also be divided into smaller meaningful units called morphemes. It is relatively easy to pick out individual words from a written sentence due to separators such as white-space and punctuation, which make it easy to determine where one word stops and another begins. However, this is not necessarily the case with morphemes. A morpheme boundary could be at the edge of a word but it could equally be somewhere in the middle. This makes the task of morphological segmentation a difficult and by no means straightforward task.

The goal of this paper is to describe an algorithm that trains and constructs language models that can be used for morphological segmentation. Models can theoretically be constructed for any language, assuming training data is available. The model itself is a semantic network with words represented as vertices and semantic similarity between words represented as edges. Vertices are grouped according to both their semantic and orthographic similarities in order to train and refine the model. These groups allow the model to incorporate concepts such as root words, affix sequences, and allomorphy without the need for complex language specific rules. This is not the first algorithm to employ both semantic and orthographic information to achieve this task, but it is unique in its use of translational equivalence in order to estimate semantic relatedness instead of more statistics-heavy approaches such as measuring cosine similarity between word vectors. It is also unique in its use of groups within a semantic network as a way of modeling morphological segmentations.

Morphological segmentation algorithms typically fall into two categories, unsupervised and semi-supervised. Unsupervised approaches generate models from unlabeled data (Harris, 1970; Sirts and Goldwater, 2013) whereas semi-supervised models use a minimal amount of labeled data combined with unlabeled data to generate models (Ko-

honen et al., 2010; Ruokolainen et al., 2014). Semi-supervised models typically are used in cases where there is limited language data available and there are some available experts to manually label a subset of the available data (Ruokolainen et al., 2016). But what if there is limited language data available and no one to manually label data? The algorithm presented in this paper considers the case where minimal data is available but it is in a specific format: parallel translated texts.

1.1 Morphological Segmentation

If you are reading this paper than by definition you can interpret at least one natural language, in this case English. This is probably something that comes naturally and does not require the active analysis it once did. However, the process of extracting meaning from speech or written text is by no means simple and involves many complex, interacting processes (see: the entire field of linguistics, cognitive science). One of these processes involves analyzing and understanding the structure of individual words. In order to analyze the structure of words, we must first determine how to differentiate one sub-word component from another. The process of splitting a word up into its component parts is called morphological segmentation and these components are called morphemes.

For the moment let us define a word as a string of graphemes surrounded by either whitespace or punctuation. There are many ways to split this word up into parts, assuming that each grapheme is an indivisible component of the word and that all parts must contain at least one letter. In fact there are 2^{n-1} possible ways to split a word containing n graphemes[1]. However, we are only interested in segmentations where each sub-word component has intrinsic semantic content that can be combined to construct the meaning of the larger word. For example, the word *papers* can be divided into the morphemes *paper* and *s* which each have the meaning of 'a thin sheet of dried wood pulp' and 'multiple/plural' respectively. By combining these two sub-word components, we interpret the meaning 'multiple thin sheets of dried wood pulp'; a reasonable definition for the word *papers*. It should be noted that neither of the two segments can be further segmented in this way (there are no meanings of *pap* and *er* for example that combine to give us the meaning of *paper*). Morphemes therefore, are defined as meaningful units of language that cannot be segmented further (Haspelmath, 2002). The main goal of this paper is to present an algorithm that can construct a language model that in turn can can be used to split any word in a given language into its component morphemes. Essentially, the model should be able to detect morpheme boundaries that are not explicitly marked

[1] 2^{n-1} is the number of ordered partitions of n

with a space character[2] or other explicit boundary marker.

1.1.1 Types of morphemes

There are many ways to define, and describe the interactions between morphemes (Haspelmath, 2002), but the distinction between root morphemes and affixes is most important for this paper. Roots tend to be free, non-productive morphemes whose meaning is sometimes analyzed as contributing the 'core' meaning to a word. Affixes on the other hand are typically bound, productive morphemes that contribute additional meaning to the word it belongs to. For example, *paper* would be considered a root, whereas *s* would be an affix.

A free morpheme is one that can be either part of a larger word or may appear as a word in its own right, whereas a bound morpheme is one that must be part of a larger word. In our previous example *paper* is a free morpheme since it can appear in a sentence as its own word, but *s* will never appear by itself (except in some specific, contrived examples). With this in mind, we can look for evidence of free morphemes by identifying instances where a word appears within a larger word. For example, *paper* is a free morpheme because it can occur independently but also as part of the larger word *papers*

Morphological researchers have long modeled the process of word creation as starting from a root morpheme, onto which other morphemes are sequentially added, adding to the overall meaning of the word at each step (Selkirk, 1982). Morphemes which are used most often in this process of word formation are considered productive, whereas those that are rarely used are considered non-productive. The morpheme *s* is very productive since it can be added to the majority of nouns in the English language, whereas there are very few root morphemes onto which we add the morpheme *paper* to construct a new word. We can therefore look for evidence of productive morphemes by identifying strings that occur much more frequently and attach to a wide variety of roots.

Most languages have syntactic rules which govern the order in which words can combine to form sentences. The same is true for the way in which morphemes can combine to form words. In English for example, we can combine *paper* and *s* as *papers* but never as *spaper*, or *unimaginable* as *ableimaginun*. Often these rules describe where a given morpheme will occur relative to the root; the *s* in *papers* for example always occurs after the root *paper*. English speakers are probably most familiar with the concept of prefixes (morphemes that occur before the root) and suffixes (morphemes that occur after the

[2] in morphology: space is not the final frontier.

root). However, there are many other ways a morpheme can appear relative to the root. For example, infixes are inserted within a root (*abso-fucking-lutely*) and circumfixes appear both before and after (*em-bold-en*). Collectively, these morpheme types are called affix morphemes, in contrast to root morphemes. It should also be noted that languages typically prefer certain types of affixes to others. English words for example contain more suffixes than other affix types, whereas Berber words favour prefixes instead (Kossmann, 1997; Dryer and Haspelmath, 2013).

To complicate matters further, a single morpheme can take on many forms. Consider the plural morpheme *s* that appears in the word *papers* and the plural morpheme *es* that appears in the word *beaches*. These two morphemes clearly convey the same meaning and they are orthographically similar enough that most English speakers would recognize that they should be analyzed as different forms of the same underlying morpheme. Multiple forms of the same morpheme are called allomorphs and must be accounted for in any morphological segmentation model.

1.1.2 Comparative morphology

By limiting our definition of a word to a string of characters separated by whitespace or punctuation we are already showing our bias towards certain types of language. Orthographies (written language) allow us to approximate and represent spoken language in a written form. However, not all orthographies are created equal. On the one hand, we have phonemic orthographies which represent each phoneme (speech sound) with a single unique grapheme (written character). On the other hand, we have logographic orthographies where each character represents a concept or phrase with no relation to the sound of a given word. In between these two extremes there are other types of writing system such as partial or defective phonemic orthographies, syllabic orthographies, and abjads. By assuming that orthography is a good proxy for pronunciation we are already creating a bias for languages with phonemic orthographies. This bias is addressed here by by allowing the algorithm to be modified to accommodate languages with less phonemic orthographies.

A further distinction can be made between languages with higher or lower morpheme-to-word ratios. Isolating languages have a low morpheme-to-word ratio; there is whitespace or punctuation between almost every written morpheme. Synthetic languages on the other hand have a high morpheme-to-word ratio; morphemes are typically written with very little whitespace. The higher the ratio, the harder it is to detect boundaries between morphemes, because there are more boundaries to detect. Morphological seg-

mentation for an isolating language is almost trivial since the orthography does all the segmentation for you. On the other hand, morphological segmentation is more difficult for synthetic languages but it is more worthwhile since it allows you to gain much more information about the morphological structure of a languages than is initially apparent.

The actual meaning of certain morphemes and the way they appear relative to each other is something else that differs between languages. For example, one language may describe grammatical gender using one morpheme and number using another, whereas a different language may describe both gender and number using a single morpheme. This makes translation complicated, since what can be described with a few morphemes in one language may require many more to describe the same concept in another.

Even if two languages do use the same number of morphemes to describe a concept, the way those morphemes are arranged can also differ in terms of ordering and degree of isolation. Just as syntactic rules describe the order of words in a sentence, morpho-syntactic rules may also describe the order of morphemes in a word. For example, one language may always mark gender with a prefix, another with an infix, another as an entirely separate word, and another may not mark gender at all. This variation often makes it difficult to describe a one-to-one mapping between morphemes of different languages but it does provide a framework to detect morphemes by comparing their representation between two languages.

This may mean that if we are using this technique of language comparison to detect morphemes, the result will likely vary depending on the languages being compared. In other words, the morphemes detected by comparing language A to language B may be different than those detected by comparing language A to language C. This assumption will be explored further in chapter 4.

1.2 Motivation

The original motivation for developing this algorithm came from the observation that many language-agnostic morphological segmentation algorithms over-segment words. In other words, they are typically a little too eager to break a word into its (assumed) component parts. For example, an algorithm may be trained to recognize that the string *ing* is a common suffix in English. The training data may contain words like: *making*, *cooperating*, *opening*, etc. which prompts the algorithm to train a model that predicts all instances of *ing* at the right edge of a word are morphemes. Then, when the model is given a word like *bring*, it may incorrectly predict that it should be segmented as *br-ing*. The algorithm may have even determined that *br* is a valid root since it observes

contrasting words like *bred*, *brine* and recognizes that *ed* and *ine* are also English suffixes.

One way to prevent this over-segmentation problem is to provide more context about specific words (like *bring*) that help the algorithm learn not only what a possible segmentation could look like, but which words should be considered for segmentation in the first place. This context could be provided explicitly, but that defeats one of the main purposes of language-agnostic algorithms — not having to rely on language specific data. A better solution is to try and infer this context from existing data. The algorithm presented here attempts to infer a semantic context for each word in question and to use that context to avoid this type of over-segmentation. To go back to the previous example, it should recognize that *bring*, *bred*, and *brine* should not be considered to have a common root because they are not semantically related.

Using semantic data in order to improve and refine morphological segmentation models is not a new idea (Narasimhan et al., 2015; Schone and Jurafsky, 2000). But the technique typically used to extract semantic data is to analyze very large corpora in order to extract word vectors. This is a great approach for languages where such large corpora are readily available. However there are many languages where the amount of data required to use these techniques is simply not available; or at least not available in a form that is ready to be analyzed. In order to overcome this hurdle for low-resource languages, this algorithm instead looks at translated corpora. We assume that the necessary semantic relationships between words can be extracted from translated corpora with less raw data than monolingual corpora. This is because the translations themselves provide a lot of semantic information, so we don't only have to rely on context clues to determine semantic relatedness. This gives us the ability to extract the same sort of semantic similarity from less, but more structured training data. Consider the difference that the discovery of the Rosetta stone made in the study of Egyptian hieroglyphics — even short translated texts can make all the difference.

Another observation of existing language-agnostic models shows that their design prioritizes languages with relatively linear, simple morphemic structures and those with phonemic orthographies. In order to level the playing field, the algorithm presented here does not assume a single basic morphemic structure or orthographic-to-phonemic mapping for all languages. The algorithm presented here allows for some flexibility in terms of how each of these components of a given language are parsed and analyzed — not so much that an advanced knowledge of the morphological structures for a given language are required, but enough so that the algorithm does not inherently perform better for one specific group of languages.

Chapter 2

Related Work

The task of unsupervised morphological segmentation has been a common subject of study for computational linguists for more than half a century (Harris, 1955). Most recent papers make a distinction between morphological segmentation and morphological analysis. The former is concerned with determining morpheme boundaries based solely on the immediate orthographic or phonemic context. The latter takes a more syntactic approach, which incorporates the idea of hierarchical syntactic structures and morpheme categories into the model (Goldsmith, 2010). In this chapter we will look at several examples of each type with a focus on the way that each chooses to represent morphemes in the models they build. For a more in-depth review of the various morphological segmentation methods see Goldsmith (2010).

Harris (1955)

Some of the earliest morphological segmentation algorithms approached the problem from the perspective of determining the most likely morpheme boundary given the immediate linear context. In his 1955 paper, Harris considered the problem in terms of splitting a sentence into morphemes by calculating peaks in "successor frequency" and defining morpheme boundaries as falling at these peaks. This technique is driven by the observation that the types of phonemes that can occur at a given point in an utterance are dependent on the phonemes that come directly before it. However, this chain of phoneme dependency is weaker when a phoneme is morpheme-initial. This means that if it is relatively difficult to predict the next phoneme in a sequence, there is a higher chance that the next phoneme will begin a new morpheme.

Harris gives the example utterance "*he's quicker*" which can be represented phonetically as /hiyzkwɪkəɹ/ and segmented into morphemes as /hiy/-/z/-/kwɪk/-/əɹ/. He

calculates the successor frequency (SF) of each sequence of phonemes (starting from the first phoneme in each utterance) by counting the number of phoneme types that follow each sequence in his corpus. For example, he found nine phonemes in his corpus that follow /h/ ($SF_{/h/} = 9$), fourteen that follow /hi/ ($SF_{/hi/} = 14$), and so on. Morpheme boundaries were then calculated to be placed directly after a peak in SF. For example, if $SF_{/hiy/} = 29$ but $SF_{/hiyz/} = 11$, then a boundary was placed at /hiy/-/z/. Several variations on this technique of determining morpheme boundaries are also discussed including: calculating SF based on the following phonemes (instead of the preceding) and calculating SF based on n-grams (as opposed to individual phonemes).

The main drawback to this method is that it requires the input data to already be segmented into short two- or three-word utterances. This essentially means that the input data must already be pre-segmented to a certain extent before this technique can work properly. Also, this method assumes that each utterance is represented as a string of phonemes instead of orthographically, which means that an initial tokenizing step (splitting the input into words based on punctuation and whitespace) would be much more difficult. Interestingly, Harris does go into some detail about the drawbacks of using an overly complex phonemic alphabet to represent each utterance.

Harris's technique introduces the idea of detecting morphemes according to the distributional properties of groups of phonemes. This inspired many later approaches to this problem such as Harris's later work (Harris, 1970) and the morfessor family of algorithms which are discussed below.

Creutz and Lagus (2007)

The original morfessor algorithm (Creutz and Lagus, 2007) attempts to segment words into their component morphemes by asking the question: what is the most efficient way to store the representation of these words in memory? This approach uses the Minimum Description Length (MDL) principle (Jorma, 1998) in order to construct a model that tries to minimize both the model's lexicon (the morphemes described by the model) and grammar (the "rules" dictating the usage of each morpheme). The grammar of a morfessor model is a Hidden Markov Model (HMM) describing transition probabilities between categories of morphemes in the lexicon and the emission probabilities of morphemes given these categories. Segmentation is performed by finding the most likely string of morphemes that can be chained together to form a word, given the probabilities described in the HMM.

The morfessor algorithm described above is the most simplified form; other adaptations of this algorithm attempt to improve on this baseline by incorporating other

information about the morphemes themselves. This information includes morpheme frequency, length, and left-right perplexity — a measure of the likelihood that a morpheme occurs near the right or left edge of a word. The left-right perplexity measure can be used to categorize morphemes into distributional categories such as prefix, suffix, and root categories. This information is used to inform certain heuristics used in these adapted morfessor models in order to determine whether a segmentation of a given word is a good one. For example, one algorithm (*morfessor-baseline-length*) enforces the heuristic that a string may only be split into two segments if at least one of the segments already exists in the model's lexicon. Another adaptation (*categories-ML*) first builds a regular morfessor model but then categorizes certain members of the lexicon as "non-morphemes" and attempts to remove them from the model by merging these non-morphemes with adjacent segments.

In the description of the morfessor algorithm, Creutz and Lagus explain that they recognize that some nuances will necessarily be lost when attempting to create a morphological segmentation algorithm purely from a list of unannotated words. This lack of nuance is mostly reflected in the tendency for this model to split most words into too many segments (over-segmentation). This model also does not incorporate the concept of allomorphy at all. Adaptations to the baseline morfessor algorithm attempt to improve on it by introducing additional linguistically motivate heuristics. A side-by-side evaluation of several of these adaptations on the baseline seems to show that no single algorithm is best for all languages. The best model for Finnish (a highly synthetic language) is not necessarily the best model for English (a more isolating language). For example, the *morfessor-baseline-length* had the best precision scores when tested against a Finnish corpus but the worst against the English corpus. This demonstrates that it may not be possible to create a single best morphological segmentation algorithm for all languages without allowing for significant language specific adaptations (Bender, 2009).

Cotterell et al. (2016a)

Originally, the task of morphological segmentation only considered splitting a word into a flat structure. However, Cotterell et al. argue that the composition of morphemes in words is similar to the syntactic structure of sentences and therefore should be modelled using a hierarchical tree structure. This allows for a more in-depth analysis of the segmentation of a given word. A tree structure is able to show the order in which morphemes can be joined to create a full word as well as the dependency structure between each of these component morphemes.

This concept is not new and is actually explored in an adaptation to the morfessor

algorithm (Creutz and Lagus, 2005). However Cotterell et al. expand on this method by introducing an algorithm that not only produces segments in a tree structure but also incorporates the idea of allomorphy by learning how certain morphemes change the orthography of the morpheme they attach to. For example, the suffix *ly* changes the word *untestable* when added (the result is *untestably* not *untestablely*). Their models are able to reconstruct the original form of a word from the version modified after combining it with an affix. This allows the model to produce segmentations that lend themselves to a more in-depth morphological analysis since both underlying and surface forms of morphemes are represented.

Cotterell et al. evaluate their algorithm by showing that it outperforms their previous model which produces a flat structured output (Cotterell et al., 2016b). However, they only provide an evaluation based on a single language (presumably English) so it is difficult to say whether these results are generalizable to other languages.

This paper also introduces a morphological tree-bank, created by applying their model to 7000+ English words. This is the first morphological tree-bank of its kind and may be useful for developing future supervised morphological segmentation algorithms that want to train on more structured data.

Sirts and Goldwater (2013); Eskander et al. (2016)

All the algorithms discussed so far have been unsupervised, meaning the inputs to these algorithms do not require the data to be pre-segmented (manually or otherwise). Sirts and Goldwater demonstrate a semi-supervised algorithm that generates an adaptor grammar (AG) from a large amount of unsegmented data and a small amount of segmented data. They show that starting their model off with a little bit of segmented data does significantly improve the performance of the model.

An AG is a probabilistic context-free grammar (PCFG) which also includes an "adaptor" which modifies the probabilities of generating sub-trees according to the frequency that these sub-trees occur in the training data. Sirts and Goldwater use the small amount of segmented training data to tune the adaptor probabilities and to prune some non-terminals in the PCFG which can be shown to be inconsistent with the segmented data. They also use their pre-segmented training data in their AG-select model training technique. This technique involves first generating multiple possible parse trees for the words in the training set and then categorizing these possible parses based on the shape of the parse tree (number of branches, depth, etc.). Then they can use their pre-segmented training data to rank the categories of parse trees, not just the individual parse trees for a given word. This allows their model to generalize and perhaps provide better segmen-

tations for words with more uncommon constructions that would not have been parsed properly otherwise.

Results from their experiments show that their semi-supervised algorithms are competitive when compared to other segmentation algorithms. However, the most interesting claim made in this paper is that their AG-select model can be initially trained on data in one language and then adapted for use with another language by adapting it with a small amount of segmented data. This technique is explored further by Eskander et al. (2016), who describe a technique for using AG models to perform morphological segmentation for a previously unseen language using AG models trained on multiple other languages. They train multiple models for several languages and select the top performing model according to the average F-measure score across all languages. The model that does the best on average across all known languages is assumed to be the best for the unknown language. The results of their experiment show that models selected in this way perform as well as or better than morfessor but worse than an AG model trained specifically for that language.

Schone and Jurafsky (2000); Narasimhan et al. (2015)

The algorithm introduced by Schone and Jurafsky is the first to explicitly incoporate semantic information. Their goal is to reduce the probability that their model will attempt to segment a mono-morphemic word simply because the word looks like it could be segmented into morphemes that occur with a high frequency in the training data. They give the example of the English word *ally* which should not be parsed as *all-y*. In order to solve this over-segmentation problem they propose incorporating semantic information about each word into the model. This would allow the model to rule out the *all-y* parse if it has sufficient evidence that *all* and *ally* are not semantically related.

Schone and Jurafsky present an algorithm that takes not only words as input but also semantic vectors for each of these words. The algorithm first generates several possible segmentations by inserting the words into a trie structure and hypothesizing that morpheme boundaries occur where the trie branches. Segmentations for a given word are only considered valid if the segmentation implies that two words should be related (because they share a common root morpheme) and the semantic relation can be shown using the vectors for each of these words. For example, a parse that supposes that *ally* contains the root *all* could be eliminated if we can show that *ally* and *all* are not semantically related. Two words are considered sufficiently semantically related if their cosine similarity is greater than the average cosine similarity between a word and each word in the corpus. Schone and Jurafsky's evaluation of this algorithm shows that

it performs similarly to an adaptation of the morfessor algorithm and they conclude that incorporating semantic data may be a useful supplement to existing algorithms.

Narasimhan et al. expand on Schone and Jurafsky's work by also incorporating semantic data into their MorphoChains algorithm. They take a similar approach by first finding multiple possible segmentations for words and then eliminating possibilities that conflict with the semantic data. They attempt to model words by detecting root morphemes and then showing how affix morphemes can be iteratively added in "chains" to form new words. For example, the root word *play* can take the suffix *ful* to form the word *playful* which is described by the chain *play*→*playful* — chains are described as parent-child pairs with the parent on the left and the child on the right. The MorphoChain models are log-linear models that predict these parent child pairs from a list of candidates (all possible segmentations of a given word made by splitting the word exactly once). The features used to train these log-linear models include a cosine similarity measure between each parent and child. By including this as a feature, the models are able to incorporate semantic similarity in the hopes of eliminating incorrect segmentations.

Narasimhan et al. evaluate their method against several baselines and show that their model does in fact typically perform better in terms of overall F-measure score. However, they also show that their models have a problem of under-segmenting words in English and Turkish, and over-segmenting words in Arabic. This is likely due to their model's over-reliance on the semantic data. This tendency is discussed further in the following chapters.

Chapter 3

Model

This chapter introduces an algorithm for Word segmentation using Grouped Semantic networks (WuGS). By using translated texts as training data, the WuGS algorithm generates not one but two morphological segmentation models at a time. These models come in the form of grouped semantic networks that are constructed using multilingual parallel translated texts as training data. First, the translations are analyzed in order to extract semantic relationships between individual words in each of the translated languages. Then, the orthography of semantically related words is compared in order to determine the most likely morphological segmentation of each word in the training data. By categorizing each predicted morpheme according to its productiveness and whether it is free or bound, these models should be able to predict possible segmentations even for previously unseen words. Also, by accounting for allomorphy and patterns in semantically related groups of words, the models should avoid over-segmentation. The details of how this fine balance is achieved are presented below.

3.1 Input Data

The input data used to construct the morphological segmentation models comes in the form of multilingual parallel translated texts. These texts are sentence aligned such that the n^{th} sentence in text A, is a translation of the n^{th} sentence in text B. For example, an excerpt from some English/French translations of subtitles from the movie Big Fish extracted from the OpenSubtitles corpus (Tiedemann, 2012) are as follows:

I saw an iceberg once.
J'en ai vu un, un jour

They were hauling it down to Texas for drinking water.
On le halait jusqu'au Texas pour avoir de l'eau potable.

They didn't count on there being an elephant frozen inside.
On ne s'attendait pas y trouver un éléphant congel.

A single piece of text containing these sort of pairings for multiple sentences is typically called a bitext (or a multitext when more than two languages are represented). The languages represented in bitexts are often differentiated as the source and the target. The source is the text in its original language, either as originally written or a transcribed version of the original speech. The target is a human-translated version of the source. However, it is not uncommon for a bitext to contain texts that were actually both translated from a third text entirely.

One important thing to notice about the bitext example above is that the translations are sentence to sentence and not word to word. Also, the translations for each sentence are often in the context of a larger text. For example, the first French sentence in the example above could just have easily been translated as "I saw one once", with no reference to icebergs at all. This is because the French translator is making use of an anaphoric reference to the iceberg, introduced previously in the text[1]. This means that a given word may be translated multiple ways even within the same text. This variety of possible translations can be exploited to help our translation model learn semantic similarities between words as will be demonstrated below.

3.2 Training

In order to train morphological segmentation models the algorithm executes the following steps. First it takes the sentence aligned input data and finds the n most likely word-to-word translations for each word in each language. Then it uses these word translations to create a network where vertices are individual words that are connected by an edge if they are semantically similar. Semantic similarity is deduced from the word-to-word translations. The vertices are also grouped based on whether they share a common root morpheme. A group therefore contains words that have a root morpheme in common but may differ in terms of which affixes are connected to that root.

By looking at the co-occurrence patterns between these affixes we can begin to get an idea of which affixes co-occur most frequently. After this initial grouping, a node may belong to multiple groups or none at all. The subsequent steps attempt to minimize

[1]The previous line in the film is "You know about icebergs, Dad?"

the number of nodes that a single word belongs to while still respecting the affix co-occurrence patterns observed in the data. The last step involves recognizing cases where our algorithm has incorrectly parsed a string of morphemes as a single morpheme and splitting up these strings in our models. Each of these steps is described in detail below.

Word-to-word translation

For the purpose of the WuGS algorithm, we consider two words to be semantically similar if they are translated similarly. For example, the words *house*, *houses*, *home* and *homes* are considered semantically similar if we can demonstrate that they can all be translated as '*maison*' and/or '*maisons*' in French. However, our input data aligns translated texts at the sentence level, not the word level. In order to proceed, we must extract word-to-word translations from our input data for both languages. The WuGS algorithm does not introduce a novel way of doing word alignment. Instead we rely on existing word alignment algorithms to do this initial data processing. There are many such algorithms available and most behave similarly: they take a sentence-aligned corpus as input and try to determine what is the most likely translation for a given word in a given sentence using an iterative expectation-maximization approach. The experiments described in chapter 4 all use Percy Liang's aligner (Liang et al., 2006) but in reality, any word alignment model can be used.

At this point we should have a mapping from each word in language A to all possible translations of that word in language B, as determined by the word alignment algorithm. For example, the top five translations for the French word *linguistique* translated into English (according to the cross-em aligner trained on the Europarl v7 corpus) are: *linguistic* (0.408), *linguistics* (0.176), *language* (0.069), *linguistically* (0.046), *word* (0.043)[2]. The aligner also predicts a long tail of very low-probability translations which we could choose to ignore by only considering translations with a translation probability above a certain threshold if desired. Figure 3.1 below shows the typical distribution of translation probabilities for a given word.

The semantic network

A semantic network is a graph where each word is a vertex and edges connect vertices that are semantically related. We want to create one semantic network for each of our translated languages, using the translation probabilities discovered in the previous word alignment step. The output of the word alignments can be displayed as a bipartite graph

[2]Translation probabilities are shown in brackets.

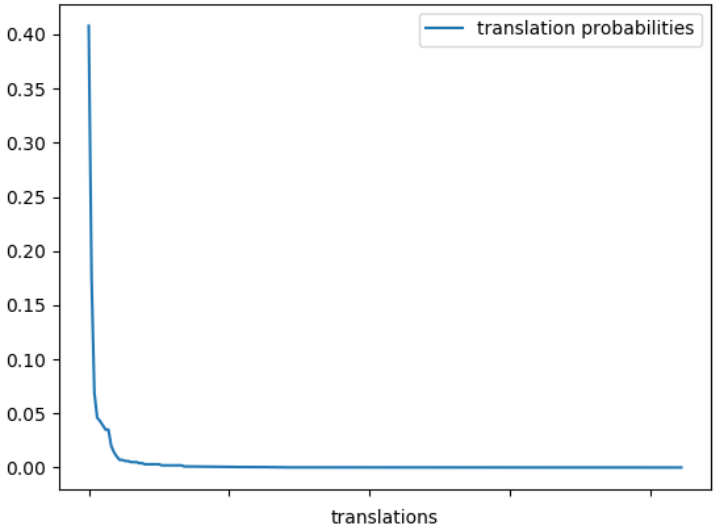


Figure 3.1: Translation probability distribution of the word '*linguistique*' translated to English

with unidirectional edges. These edges are present between a word w_a in language a and word w_b in language b if our word alignment output predicts that w_a can be translated as w_b. Note that our model may predict that w_a can translated as w_b but w_b cannot be translated as w_a; hence the unidirectional nature of the edges. For example, a partial bipartite graph showing these relationships might look like figure 3.2.

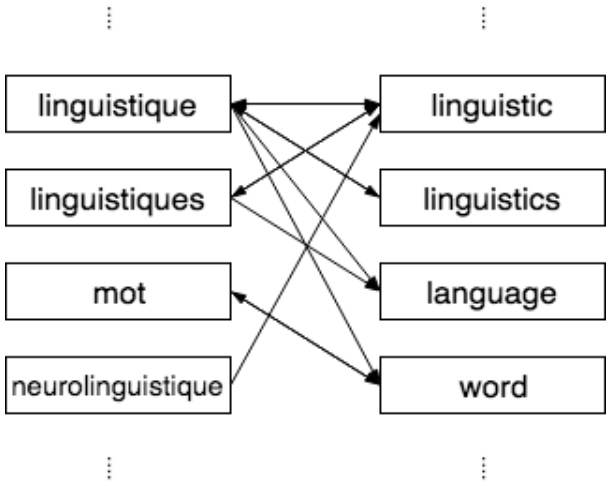


Figure 3.2: Partial bipartite graph showing English-French word translations

In order to transform this bipartite graph into two semantic network graphs (one for

each language), we can connect all words in a given language that share a translation. In other words we can create a semantic network for language A by removing all nodes in our bipartite graph that correspond to words in language B and collapsing some or all of the edges. Then we do the same for language B by removing the nodes corresponding to words in language A. An algorithm demonstrating this step is shown in pseudo-code in algorithm 1. The input variable `EdgesA` is an array of triples corresponding to a word in language A, a translation of that word in language B and the translation probability between those words. `EdgesB` is the same but from language B to language A. Helper function definitions can be found in appendix A.

Algorithm 1 From word translation to semantic network

```
FUNCTION create_semantic_network(edges_a, edges_b, min_prob)
    hash_b = edges_to_hash(edges_b, min_prob) # see helper functions
    network = Hash() # a new hash table
    FOR edge IN edges_a
        word, translation, prob = edge
        IF prob >= min_prob AND hash_b.has_key(translation)
            IF NOT network.has_key(word)
                network[word] = []
            FOR word_2 IN hash_b[translation]
                network[word] = network[word] + [word_2]
    RETURN network

min_prob = 0.2 # for example
SemNetA = create_semantic_network(EdgesA, EdgesB, min_prob)
SemNetB = create_semantic_network(EdgesB, EdgesA, min_prob)
```

There are several options regarding which types of semantic relationships we want to consider. There are two types of relationships between words of the same language demonstrated in figure 3.2; those where both words in language A can be translated as the same word in langauge B (symmetric translation), and those where one word in language A can be translated into a word in language B which in turn can be translated back as another word in language A (asymmetric translation). For example, the French words *linguistique* and *linguistiques* demonstrate a symmetric translation relationship in figure 3.2 because they both can be translated as the English word *language*. However, the English words *linguistic* and *language* demonstrate an asymmetric translation because *linguistic* can be translated into French as *linguistiques* which in turn can be translated back to English as *language* but the relationship does not go the other way.

We have the option of constructing our semantic networks considering only symmetric,

asymmetric, or both types of translation. Algorithm 1 considers asymmetric relations but could be adapted to consider symmetric relations instead (see algorithm 10 in appendix A). Figures 3.3 and 3.4 show the difference between semantic networks built with each of the two strategies outlined above.

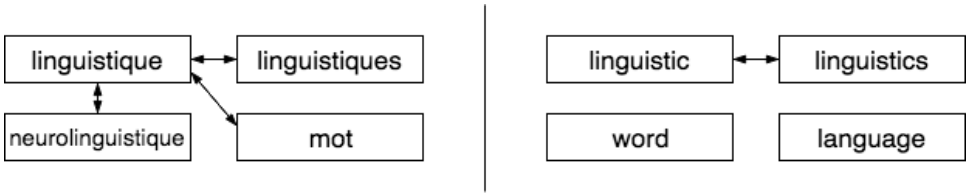


Figure 3.3: Semantic network derived from figure 3.2 using symmetric translations

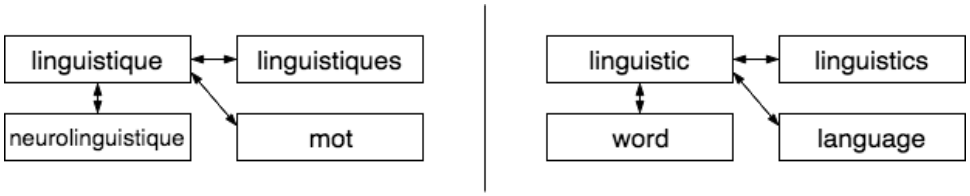


Figure 3.4: Semantic network derived from figure 3.2 using asymmetric translations

At first glance it may seem that we should always use both strategies since doing so will always provide us with a semantic network with more connections at this point. However, we must consider the fact that the word alignment algorithms we use are not perfect and may introduce some error into our models by misaligning words. It is possible that these misalignments are realized more as one strategy compared to the other which means that we might in fact get better results by choosing only one strategy.

Network pruning and grouping

The semantic networks constructed in the previous section connect words that have some degree of translational equivalence, which is a measure of semantic similarity. However, just because words are semantically similar does not necessarily mean that they also share a root morpheme. In the example semantic network shown in figure 3.4, the words *linguistic* and *language* are shown to be semantically related even though they do not share a common root morpheme[3]. In order to determine which words are both

[3]It should be noted that this paper takes a synchronic approach to language analysis, meaning that we do not consider historical roots of words as valid interpretations of a given word's root morpheme. This is the approach taken by most modern morphological segmentation corpora such as the Celex morphology corpus (Baayen et al., 1995). In this example it means that *linguistic* and *language* should not be analyzed as having the same historical root morpheme: either the Latin *lingua* or the Proto-Indo-European **dnghu-* both meaning tongue (Quiles et al., 2012)

semantically related and share a common root morpheme, the next step is to define a string similarity measurement and threshold and to remove the edges between any two words that are not sufficiently similar. One solution is to measure string similarity using a longest common substring (LCS) algorithm and to remove edges between words that do not have an LCS of some minimum number of characters. This would preserve the edge between *linguistic* and *linguistics* (an LCS of 10 characters: *linguistic*) but remove the edge between *linguistic* and *language* (an LCS of 3 characters: *ngu*).

By pruning edges of the semantic networks using a string similarity measurement, we are not only reducing the number of edges in the graph but also identifying sub-networks defined by a shared LCS. These subnetworks are called groups and each group has a label corresponding to the shared LCS of its members. For example, the words *make*, *maker*, *remakes* are semantically related and the LCS of these words is the string *make*; therefore we can create a group for these words with the label *make*. In many cases, the group label should end up corresponding to the root morpheme that all the words in the group have in common. This is similar to a technique used in Cotterell et al. (2016a) to reconstruct the underlying form of morphemes.

At this point, a word may belong to more than one group. For example, in the *make* group described above, all of the words are also semantically related to the word *making* which does not contain the substring *make* — the LCS between these words including *making* is simply *mak*. This means that words like *make*, *maker*, and *remakes* belong to both the *make* group and the *mak* group. This example illustrates the case where there are several allomorphs of a given root morpheme. Another case where a word may belong to multiple groups occurs in the case of compounds. The word *shoemaker* is semantically related to words like *shoe* and *shoes* as well as words like *make* and *making*. This means that it is likely to be put into the groups: *make*, *mak*, and *shoe*. These groupings are illustrated in figure 3.5.

The next goal of the WuGS algorithm is to determine which group is most appropriate for each word. This is done by examining patterns of affix morphemes within groups and maximizing occurrence probabilities of these affixes.

Building the model

After grouping words according to potential shared root morphemes, the next step is to determine which of these potential shared roots is the correct root for a given word. This can be done by first segmenting a word according to its predicted roots, seeing which affixes are predicted by this segmentation and then ranking each potential root according to the occurrence probabilities of the affixes. For example, consider the words

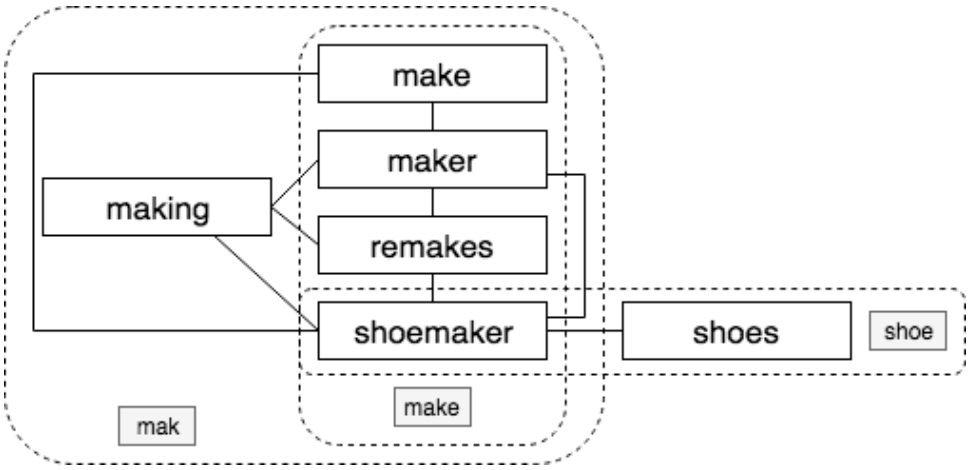


Figure 3.5: Example semantic network with multiple groupings

precognition, *recognition*, and *cognition*, all of which are semantically related. Based on these relations, the word *precognition* could either have the root *recognition* (with the prefix *p*) or *cognition* (with the prefix *pre*). By showing that *pre* is a much more probable prefix than *p* we predict that *cognition* is a better candidate as the root morpheme of *precognition* than *recognition*.

The probability of a given affix is determined by collecting co-occurrence data of predicted affixes already present in the model. For example, the words *remakes* and *maker* are in both the *make* and *mak* groups which means they can be segmented as either *re-make-s* and *∅-make-r*[4] or *re-mak-es* and *∅-mak-er*. This can be analyzed as two counts of *re* and *∅* co-occurring, one count of *es* and *er* co-occurring and one count of *s* and *r* co-occurring. This data collection algorithm is described in algorithm 2. The group's input is a hash table mapping a potential root morpheme to a list of words. For example, `groups['mak']` may contain the words *make*, *making*, etc.

After collecting basic co-occurrence statistics, we can begin to eliminate low-probability roots based on several heuristics. These heuristics are based on the assumptions that affixes are productive morphemes and that they co-occur predictably. In other words, the morphemes we identify as affixes should occur with a much higher frequency than those we identify as roots. For example, we may see the root morpheme *make* (or *mak*) occur several times in a given corpus in words like *make*, *maker*, *remakes*. But the suffix *er* will occur much more frequently as a productive affix in words like *maker*, *baker*, *producer*, etc. As a more concrete example, consider the Celex English Morphology Lemma (EML) corpus (Baayen et al., 1995) which contains 24 examples of words with the root *mak* and

[4]The *∅* character is used to show the absence of an affix, not an example of null-derivation (Haspelmath, 2002).

Algorithm 2 co-occurence data

```
FUNCTION co_occurrence_data(groups)
    data = Hash()
    FOR root IN groups
        affixes = get_affixes(groups[root], root)
        FOR aff IN affixes
            IF NOT data.has_key(aff)
                data[aff] = []
            data[aff] = data[aff] + [root]
    RETURN data
data = co-occurrence_data(groups)

# the number of co-occurrences between two affixes
# can be calculated with the following function
FUNCTION co_occurrences(aff_a, aff_b, data)
    shared_roots = data[aff_a] INTERSECTS data[aff_b]
    RETURN shared_roots.length
```

1426 examples of words with the affix *er*. Affixes are also assumed to co-occur predictably because of the tendency for languages to use affixes to mark the syntactic category of a given word (Haspelmath, 2002). For example, in English, both the suffixes *ing* and *ed* are used to mark tense on verbs. Therefore, if we have evidence of a word containing an *ing* suffix, we can expect to also find a word with the same root morpheme with an *ed* suffix (eg: *baked*/*baking*, *feared*/*fearing*)[5].

Possible roots for a given word are determined by the groups that word currently belongs to. To eliminate a possible root, we can simply remove a word from a group. For example, the word *precognition* may be removed from the group defined by the root *recognition* if we determine that *recognition* is not a likely root. With all this is mind, the following steps can be used to eliminate low-probability roots for specific words:

1. eliminate non-productive affixes.
2. redefine group labels according to their members.
3. account for allomorphy.

Eliminate non-productive affixes

After the data collection step described in algorithm 2, if we inspect the counts for each affix a familiar pattern emerges. There are several affixes that occur many times but there

[5]While the WuGS algorithm does not identify irregular forms directly (e.g. *fell*/*falling*) they may be identified by their conspicuous absence. This is discussed further in section 5.

is a long tail of affixes that occur very few times, maybe only once. Since we predict that affixes are productive morphemes we can choose to remove words from a group that implies the existence of a very low-frequency affix. For example, if the word *carbonation* has the potential root *bon* (it belongs to the *bon* group), then this implies the existence of two affixes: the prefix *car* and the suffix *ation*. Although the suffix may occur quite frequently, the prefix occurs very infrequently. Perhaps this is the only example of *car* as a prefix in the entire corpus. If a single affix implied by a given root occurs with a low enough frequency, we can remove that word from the group. An implementation of this step is shown in algorithm 3.

Algorithm 3 remove low-frequency affixes

```
FUNCTION remove_low_freq_affixes(groups, min_occurrences)
    data = co-occurrence_data(groups)
    FOR root IN groups
        words = groups[root]
        FOR word IN words
            affixes = get_affixes([word], root)
            FOR affix IN affixes
                IF data[affixes].length < min_occurrences
                    groups[root].remove(word)
                    BREAK
```

Algorithm 3 can actually be applied iteratively until there are no more words to remove from groups. This is because words are removed on the evidence of a single low-frequency affix even though there may be several affixes implied by the word and its root (the affixes returned by calling the function `get_affixes([word], root)`). Given the *carbonation* example above, if we have set the threshold to `min_occurrences = 2` and there are two examples of the affix *ation* and one example of the affix *car*, then after the first pass there will be no examples of *car* and a single example of *ation*. This then reduces the number of occurrences of *ation* below the the threshold which means that on the second pass, the other word/root combination that implies *ation* would also be eliminated. The advantage of iterative application of this step is that it is able to remove low-frequency affixes that initially occur at rates above the `min_occurrences` threshold.

Redefine group labels

After this first elimination step the group definitions themselves must be re-evaluated. This is because the label for each group was defined as the string of characters that each of the members of that group had in common. After removing some members of

the group, the string in common may have changed. For example, if a group label is defined as the longest common substring (LCS) of the words in the group consisting of *prepare*, *prepared*, and *paris*, then the group label should be *par*. But if the word *paris* is removed from this group in the previous step since *is* is determined to be a sufficiently low-frequency affix, then the new group consists of *prepare* and *prepared* only. At this point, the group label should be changed to *prepare* since that is now the LCS of the words in the group.

Account for allomorphy

It should be noted at this point that the algorithm has not taken allomorphy into account. This means that the model currently contains overlapping groups that should really be combined. For example, the word *maker* would likely belong to both the *make* and *mak* groups. If we consider that both of these group labels are allomorphs of the same root this implies that *er* and *r* are also allomorphs of the same suffix.

The evidence for allomorphy comes from two sources in our data. If there are two groups whose members intersect, they are potential candidates for root allomorphy. If the affixes implied by the two roots for words belonging to the intersection of the two groups are also candidates for allomorphy then the two groups can be merged.

Affixes are candidates for allomorphy if they regularly co-occur in overlapping groups and rarely co-occur in the same group. For example, the affix *er* is implied by the presence of the word *maker* in the group *mak* and the affix *r* is implied by the same word in the group *make*. Because the word *maker* appears in both groups this is evidence that both groups overlap. However, the two affixes *er* and *r* are not both implied by words in a single group. In other words, there is no word *makr* in the *mak* group and no word *makeer* in the *make* group.

After merging, the group labels are changed to reflect both allomorphs by bracketing the extra characters. In the example above, the label of the group created by merging the *make* and *mak* groups would be *mak(e)*. Subsequently, the word *maker* in this group now is analyzed as having the root *mak(e)* with the affix *(e)r*. Adjacent characters in brackets should be interpreted as overlapping, as opposed to consecutive.

3.3 Putting the model to work

Now that the model has been finalized, it is ready to be used to segment words. There are two steps to word segmentation, which involve first discovering all roots present in a word and then segmenting the word into roots and affixes. Where multiple possible

segmentations exist for a given word, they can be ordered based on the degree of semantic relatedness between the word being segmented and the words in the group predicted by the segmentation. The degree of semantic relatedness is calculated by counting the number of edges between the word in question and the other words in the group.

Segmenting compounds

Up to this point, the algorithm has considered words that contain a single root. However this is not always the case; a word may contain multiple roots as well as affixes, and the algorithm described here must be able to account for these words. Just like before, we can use both semantic and orthographic clues to determine which words contain multiple roots; which we will refer to generally as compound words (or compounds). Compounds are defined here as being words that belong — or used to belong before one of the pruning steps — to multiple groups where the label for at least one of the groups is also a word that the compound is semantically related to. For example, the word *houseboats* belonged (before pruning) to two groups whose labels are *house* and *boat*, and there is an edge in the semantic network connecting the vertex for *houseboats* to the vertex for *boat*.

The reason why the criteria includes groups that a word *used to* belong to is because of the various pruning steps described in previous sections. For example, algorithm 3 describes a method for removing all words from groups that imply non-productive affixes. By applying this algorithm, the word *houseboats* would likely be removed from the group *house* because the proposed suffix *boats* is not productive. Similarly it would likely be removed from the group *boat* since *house* is not a productive prefix. Luckily, the semantic network is implemented in a way which makes it easy to query which groups a word *used to* belong to as well as which groups it *currently* belongs to.

Using these criteria, we can determine whether a given word is a compound and determine where the morpheme boundary between the two root words lies. The boundary may not be totally clear if the two roots of the word are not adjacent, as is the case with a word like *cupsful* (a plural of *cupful*) which contains the plural affix *s* after the root *cup*. In cases with non-adjacent roots, the algorithm will analyze the middle section as an infix if there is evidence for it as an affix either following the left-most root or preceding the right-most root. In the *cupsful* example, we should see if there is a word *cups* with the root *cup* or the word *sful* with the root *ful*. If neither are present in the model, we can conclude that the word should not be analyzed as a compound.

This method can also be adapted slightly to accommodate words with more than two roots by iteratively applying the method described above until no more compounds are found. For example, the word *crossbowman* may belong to both the groups *bow* and *man*

which allows us to initially determine that it can be split into *crossbow-man*. The *man* segment is a root and so cannot be segmented further. However, the *crossbow* segment contains the root *bow* as well as the remainder *cross*. This remainder should first be analyzed to see if it is a productive affix on the root (can we analyze *cross* as a prefix of *bow*?). If not, then *crossbow* should be analyzed as a second compound.

It may also be the case that *crossbowman* also belongs to the group *cross*. In this case we may be able to determine from the outset that *cross-bow-man* is a possible way of segmenting this word as a triple compound.

Segmenting affixes and accounting for affix sequences

The affixes described by this model so far are discovered by taking a word, removing its root, seeing what is left, and determining whether what is left is a productive affix. However, this process is not sufficient if there are several sequential affixes. Lets take the classic example of the word *antidisestablishmentarianism*. The root of this word is *establish* but our model will have eliminated the group *establish* because the potential affixes *antidis* and *mentarianism* are not productive. The proper analysis is to treat these potential affixes not as a single morpheme but as a sequence of morphemes (*anti* + *dis* and *ment* + *arian* + *ism*).

Evidence for affix sequences comes from the fact that many group labels are actually words themselves. For example, the word *creationism* may belong to the group *creation*, but *creation* itself is also a word in the network which belongs to the group *creat(e)*. This indicates that *creation* is not a root in itself (even though it is the label of a group) but is a root combined with an affix. In this case, the correct segmentation for *creationism* is *creat(e)+ion+ism*.

By analyzing each group label to see whether it could itself be segmented further, the model can account for these affix sequences. The recursive algorithm 4 demonstrates how a word can be segmented even if there are multiple possible affix sequences.

Algorithm 4 possible segmentations

```
FUNCTION possible_segmentations(groups, word)
    segs = []
    FOR group IN get_groups(groups, word)
        affixes = get_affixes([word], group)
        IF is_word(group) # returns true if the argument is a word in groups
            affixes = affixes + possible_segmentations(groups, group)
        segs.append(affixes)
    RETURN segs  # in the actual implementation there is a step that sorts this as we
```

However, there is a major flaw in algorithm 4 that becomes obvious if we reconsider the *antidisestablishmentarianism* example. In order to determine that *establish* is the root of this word, all intermediate words must be present in the network. To take a simpler example, we cannot determine that *creat(e)* is the root of *creationism* if the intermediate word *creation* is not present. One solution would be to adapt the algorithm to recognize that *ionism* can be decomposed into two productive affixes present in the model (*ion* and *ism*) and just assume that this is the correct segmentation if we can show that *creationism* once belonged to the *creat(e)* group. The drawback of this is that it introduces too many opportunities for over-segmentation based on minimal evidence, something WuGS is designed to avoid. The solution is to introduce the concept of potential words that can be verified or eliminated during online training.

Potential words are words that do not exist in the model, but are predicted to exist in the language in question based on '*holes*' in the model. An example of a '*hole*' is exactly like the *creation* example above. During segmentation, the model may predict that there exists an intermediate word — in our example, the intermediate word is *creation* in between the word *creationism* and the root *creat(e)* — by determining that an affix that has been determined to not be productive enough is actually composed of multiple productive affixes. Potential words are inserted into the model as if they were actual words but given a '*potential*' label. Words with a '*potential*' label are ignored by the segmentation algorithms. The '*potential*' label may be removed if evidence of its existence is provided as the model is updated.

Updating the model and guided training

Because the model is essentially just a semantic network, it can be updated after training by adding or removing vertices or edges from the network. Adding a vertex is the equivalent of learning a new word, adding an edge is the equivalent of learning that two words are semantically related. Simply adding to the network will not change the segmentations the model predicts. In order to do that the newly added words must be sorted into groups by predicting their roots. The roots can be predicted for the newly added words by applying the same grouping algorithm to the sub-network containing the newly added vertices and any other adjacent vertices (those that are connected to at least one new vertex by an edge). If a word is added that already exists in the model as a potential word, then the 'potential' label can be removed. However, any edges that were predicted for the potential word that are not also confirmed by new data must remain as potential edges.

The existence of potential vertices and edges in the graph means that the model can

guide its own learning by forming hypotheses and then seeking out data to confirm or invalidate them. As a very basic example, it is possible to interactively train the model by iterating over all potential vertices and edges and asking a speaker of the language either: "Is X a word?" where X is a potential vertex or "Is the word X related to the word Y" where there is a potential edge between vertices X and Y. It may even be possible to conduct semi-guided training that uses additional unstructured input data in a focused way, essentially using the initial model to bootstrap multiple iterations of unsupervised learning. These techniques will be discussed in more detail in chapter 5.

Manual configuration

It should be clear by now that the morphology of any language can be complex and many factors can go into determining where boundaries between morphemes lie. It may be possible to create an algorithm that performs equally well for all languages out of the box, but if that algorithm exists, the WuGS algorithm as described is unlikely to be it. The simple fact that WuGS uses orthographic data means that it is biased in favour of languages with more phonemic writing systems. In order to mitigate these biases, WuGS provides the opportunity to be customized based on limited linguistic knowledge. These customizations can either be applied as phonemic regularization rules or as root detection algorithms.

Phonemic regularization rules essentially make an orthography more phonemic. For example, a grapheme to phoneme (g2p) algorithm may translate a written word to its phonological representation. For example, the word *laughter* can be represented phonologically as /læftɹ/. This makes sure that both *au* and *gh* are treated as indivisible units of speech, which eliminates the possibility that the model will consider segmentations like *la-ug-hter*. Phonemic regularization rules can be applied to each word as a preprocessing step, essentially converting the input data before it is seen by the algorithm. Additionally, they can be applied to individual morphemes after each step of the algorithm.

Root detection algorithms describe the process by which we can analyze multiple words in order to discover their shared root morpheme. One possible root detection algorithm discussed above is to take the longest common substring (LCS) of two words, but that is not the only option. Some languages have root morphemes composed of non-adjacent characters, in which case using a longest common subsequence algorithm may be more appropriate[6]. Or, you may decide that you don't want to consider a certain set

[6]The difference between these two is that the longest common substring considers only adjacent characters, whereas the longest common subsequence considers non-adjacent characters as well. The longest common substring of the words *wake* and *woke* is *ke* whereas the longest common subsequence

of characters — like vowels for example — when detecting roots. These are just a few examples but possibilities are endless, as long as an algorithm can compare two strings and return what they have in common, it can be used to build these models.

3.3.1 Output

After the model described above is constructed, it can now be used to actually perform morpheme segmentation. If the input word is already present in the model (i.e. it exists as a vertex in the network), the segmentation procedure is exactly as described in algorithm 4 after first determining whether a word is a compound and splitting it into its component parts if so. If multiple possible segmentations are present for a given word, they can be ordered according to the degree of semantic relatedness between the input word and the other words with the same root implied at each segmentation step. For example, imagine the word *unimaginable* belongs to two groups with labels *unimagin(e)* and *imagin(e)*. In order to determine whether this word should be initially segmented as *unimagin(e)-able* or *un-imagin(e)-able*, we can count how many edges there are between the words in the *unimagin(e)* group and *unimaginable* and the words in the *imagin(e)* group and *unimaginable*. The segmentation created based on the group with more connected edges is then ordered first.

If a word is not present in the model, a best guess is made as to where to insert the word in the model and the same segmentation procedure is performed. If the input word is present as a potential word then that word is treated as non-potential for the purposes of its segmentation. On the other hand, if a word is not present in the model at all, it is inserted into the model by choosing a group and performing the same segmentation procedure as if that word belonged to that group. The group detection procedure is described in algorithm 5 below where `groups` is a list of all group labels, *word* is the input word, and `root_detection_function` is a function that performs the root detection algorithm described in section 3.3.

Algorithm 5 possible groups

```
FUNCTION possible_groups(groups, word, root_detection_function)
    possibilities = []
    FOR group IN groups
        IF group == root_detection_function(word, group)
            possibilities.append(group)
    RETURN possibilities
```

is *wke*.

If we are also provided some context for the word not present in the model, it is possible to limit the `groups` argument to just those groups that contain at least one word that is semantically related to the unknown input word. For example, if the `root_detection_function` used for a given model finds the longest common subsequence between two words, it might predict that the input word *homeowner* belongs to the group *meow* which contains the words *meow*, *meowing* and *meows*. However, if we can show that *homeowner* is not semantically related to any of these words, we can eliminate the *meow* group as a possibility.

Additionally, the number of possibilities can be limited based on the affixes that are implied by the input word belonging to a given group. This is the same initial pruning procedure that was used initially to construct the model where low-frequency (non-productive) affixes were eliminated early on. In the *homeowner* example, the affixes implied by its inclusion in the *meow* group are *ho* and *ner*, neither of which are likely to be high-frequency affixes. If it can be shown by inspecting the other affixes in the model that *ho* or *ner* are not productive, we can safely eliminate *meow* as a possible root.

If there are no possible groups found, then there is one other possibility; the input word might be a compound. The same procedure for detecting compounds as described in 3.3 can be applied. If there has been semantic information provided for the input word then those semantic relationships are used. Otherwise the word is assumed to be semantically related to all other words.

A high-level description of the segmentation procedure is shown below in algorithm 6.

Algorithm 6 segment

```
FUNCTION segment(word, model)
    IF word IN model
        IF word.is_a_compound()
            part_a, part_b = split_compound()
            segmentation = segment(part_a).combine_segmentations(segment(part_b))
        ELSE
            segmentation = possible_segmentations(model, word)
    ELSE IF potential(word) IN model # is the word a potential word in the model
        depotentialize(word, model)  # remove the potential label from the word
        segmentation = segment(word, model)
        repotentialize(word, model)  # add a potential label to the word
    ELSE
        insert_unknown_word_in_model(word, model)
        segmentation = segment(word, model)
        remove_from_model(word, model)
    RETURN segmentation
```

Chapter 4

Evaluation

The Word segmentation using Grouped Semantic networks (WuGs) algorithm described in chapter 3 is evaluated here by comparing segmentations generated by the model to a gold-standard corpus containing morphological segmentations. These comparisons can be made in several ways, each resulting in an evaluation that reveals something specific about the accuracy of the models being tested. The evaluations presented below will test both morphological precision and recall as well as boundary precision and recall. Each of these concepts and the means of evaluating them are explained in further detail below.

Since the algorithm described here trains models using parallel corpora, models can also be evaluated by comparing the target language it has been trained with. For example, an English model can be trained using an English-French corpus, an English-Arabic corpus, etc. By comparing results between these models, we can investigate how the target language affects the accuracy of a model.

Baseline models and settings

In order to provide a frame of reference, the results from evaluating two other models will also be shown as a baseline comparison whenever possible. The baseline models are generated using the morfessor2.0 algorithm (Virpioja et al., 2013) and the MorphoChain algorithm (Narasimhan et al., 2015). The morfessor2.0 algorithm is a very popular morphological segmentation algorithm that can be trained using raw text data. It is a python based implementation of the original morfessor algorithm (Creutz and Lagus, 2007). The MorphoChain algorithm on the other hand is trained using both raw text data and word embeddings. This makes it an interesting baseline to compare to since, like the algorithm described in this paper, it also uses semantic data. The word embeddings for MorphoChains were generated by extracting vectors from the training

corpus using word2vec software (Mikolov et al., 2013).

For each algorithm compared here, the basic settings were used and no attempt was made to manipulate the settings in order to get better performance for a given model. This was done in order prevent bias in favour of one model over another. It is assumed that for both the morfessor 2.0 and the MorphoChain algorithms, the default settings are intended as the recommended settings for using each algorithm "out of the box"[1]. For the algorithm described in this paper, the base settings are as follows:

- words that have a likelihood of being translational equivalents less than 10% are not considered;
- both symmetric and asymmetric translations are used;
- a root must be at least two characters long;
- affixes that occur less than five times in the model are removed.

For one final experiment involving the conversion of words to their phonological representation during training, the grapheme to phoneme model epitran (Mortensen et al., 2018) was used during the segmentation step. This final experiment tests the assumption that a phonological representation of a word results in better segmentations.

Gold-standard corpus

The gold-standard corpus used in these evaluations is the Celex2 corpus which contains morphological segmentations for English (en), German (de), and Dutch (nl). All of the models used in this evaluation were trained on the TED Talks 2013 parallel corpus (Tiedemann, 2012) which was selected because it contained corpora for each of the three gold-standard languages as well as several other languages with non-Latin orthographies such as Arabic and Chinese. It was also chosen because the content of the corpus itself covered a large variety of topics, meaning there would be a lot of semantic diversity in order to test how the models perform outside of a narrow semantic domain.

[1] Also note that while the morfessor algorithm is incredibly easy to use, it was very difficult to get the MorphoChain algorithm to produce a model trained on any corpus other than their minimal example. For this reason, the MorphoChain results shown here are from a slightly modified algorithm. I tried to make as few changes possible in order to maintain the integrity of the code. However it should be noted that since I am not the author of this algorithm, I cannot be certain that my changes did not affect the results.

4.1 Comparing Models

4.1.1 Morphological precision and recall

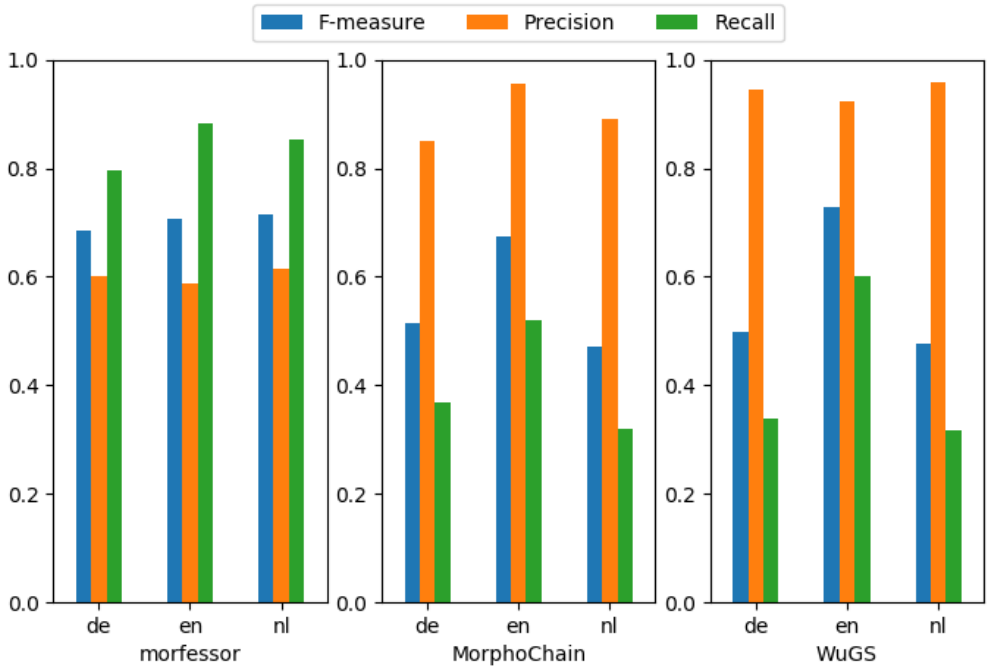
Morphological precision and recall are calculated by comparing the morphemes generated from the output of a given model (test morphemes) to the gold-standard morphemes described in the Celex2 corpus (gold morphemes). Precision is calculated as the number of test morphemes that also appear as gold morphemes for a given word, divided by the total number of words. Conversely, recall is calculated as the number of gold morphemes that also appear as test morphemes for a given word, divided by the total number of words. Both of these scores are calculated using the EMMA2 (Evaluation Metric for Morphological Analysis 2) software (Kurimo et al., 2009), a more efficient version of the original EMMA algorithm (Spiegler and Monson, 2010). EMMA2 also reports an F-measure which is calculated as $2pr/(p+r)$ where p is the precision score and r is the recall score. The F-measure is used to merge the precision and recall scores into a single value which can be used to directly compare the results of each model.

Figure 4.1 shows these three scores for the WuGS and the two baseline models across all three languages. Both the WuGS model and the MorphoChain model show quite high precision scores but relatively low recall scores whereas the morfessor model is more consistent. A high precision score with a low recall score indicates that the segmentations that the model is making are accurate but that there are many cases where the model is not making segmentations where it should. This is evidence of under-segmentation in both the WuGS and the MorphoChain model. On the other hand, the morfessor model seems to make more incorrect segmentations (lower precision) but makes more correct segmentations more often (higher recall). This is evidence of over-segmentation by the morfessor model.

One of the stated goals behind developing the WuGS algorithm was to reduce the effects of over-segmentation by ensuring that segmentations were made only with sufficient semantic and orthographic evidence. The results shown here seem to indicate that this approach was too conservative. In other words, WuGS has improved precision at the expense of recall. This also seems to be a flaw in the MorphoChain algorithm, which also requires semantic evidence. On the other hand, the morfessor algorithm is not conservative enough and that is reflected in its relatively low precision score.

By comparing the performance between languages, it seems that all models seem to perform better for English compared to German or Dutch. This may be because there are slightly fewer segmentations to make in English. The average number of morphemes per word in the English gold-standard corpus is 1.7 whereas for Dutch and German it is

Figure 4.1: Precision, recall, and F-measure reported by EMMA2



(See table B.1 for complete data)

closer to 1.95. There may be other factors at play as well, these will be discussed further in chapter 5.

Boundary precision and recall

The Boundary Precision and Recall (BPR) metric (Kurimo et al., 2009) calculates similar precision, recall, and F-measure scores, but instead of looking at the morphemes described by a given segmentation of a word, BPR is concerned with the boundary points between these morphemes. A boundary point exists wherever there is a morpheme boundary between two letters in a word. For example, the segmentation *pre-meditat-ed* for the word *premeditated* implies morpheme boundaries after the third and tenth letters. Precision from BPR can be calculated by checking how many of the predicted boundary points are correct (i.e., how many match the gold reference) and recall can be calculated by checking how many of the boundary points in the gold reference were predicted.

The advantage of comparing the segmentation boundaries as compared to the morphemes themselves is that it also evaluates the order of morphemes. For example, morpheme precision tests whether a given predicted morpheme appears *anywhere* in the gold-standard segmentation whereas a boundary only matches if the segmentation was

made between the correct letters.

Regardless, this distinction seems to have made very little difference compared to the EMMA2 analysis. This may be because all three languages being evaluated do not allow for significant reordering of morphemes within words and they also do not use a lot of reduplication (Dryer and Haspelmath, 2013). We would expect the EMMA2 and BPR results to be drastically different only if the morpho-syntactic rules of the language being evaluated were flexible enough to allow for free ordering of morphemes or if the gold-standard that is being compared against presents words segmented into morphemes using the longest forms of all allomorphs. For example, if the gold-standard describes the segmentation of the word *making* as *make-ing* then the BPR may consider the boundaries to be wrong even for a correct parse of *mak-ing*. In order to prevent this and to get the most accurate results, the Celex2 corpora were pre-processed to ensure that all segmentations contained the exact same number of characters as the original word.

The table in figure 4.2 also shows the recall scores for a simple model that returns the word being segmented[2]. In other words, it always predicts that there are no segmentations to be made for the input word. This shows that the WuGS outperforms this simple baseline by 7–10% which shows that it performs better than a naive guess.

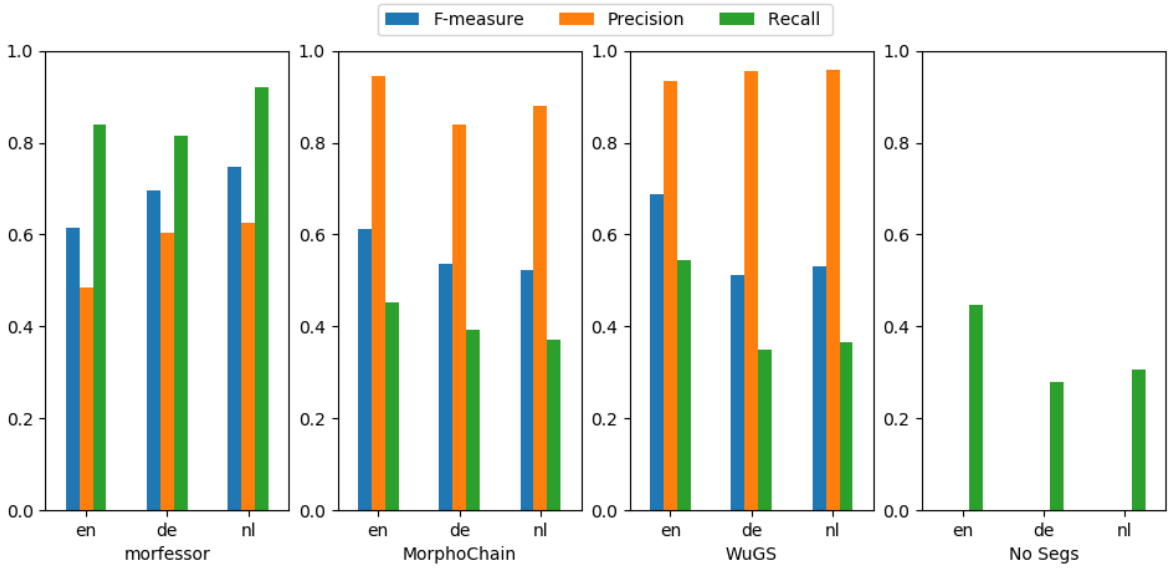
Since the BPR and EMMA2 analysis results are so similar, let us focus on just BPR for the moment. We have already seen that the results for each model differ depending on the source language. By controlling for source language, we can also see how the model behaves when being asked to segment words that occur in the training data compared to previously unseen words. This was tested by splitting the gold-standard into the set of words that were in the training model, and the set of words that were not in the training model, and analyzing these sets with the BPR algorithm. Figure 4.3 shows the results of this test.

As might be expected, the F-measure is higher for all models for the 'in-training' data. This is because a model is likely to have more available evidence for a word it was trained on. For the models that are trained using semantic evidence, that evidence is only available for previously seen words. For all other words the semantic relatedness must either be guessed at or an analysis must be made without it. See chapter 5 for a discussion of incorporating semantic information at the testing stage.

One thing to note is that the percentage of 'in-training' words out of all words in the test set varies from language to language. For English, 38% of words are 'in-training' but

[2]Precision and F-measure are not shown, since precision will trivially always be 1.0 and the F-measure is calculated from this trivial precision score.

Figure 4.2: Precision, recall, and F-measure reported by BPR



(See table B.2 for complete data)

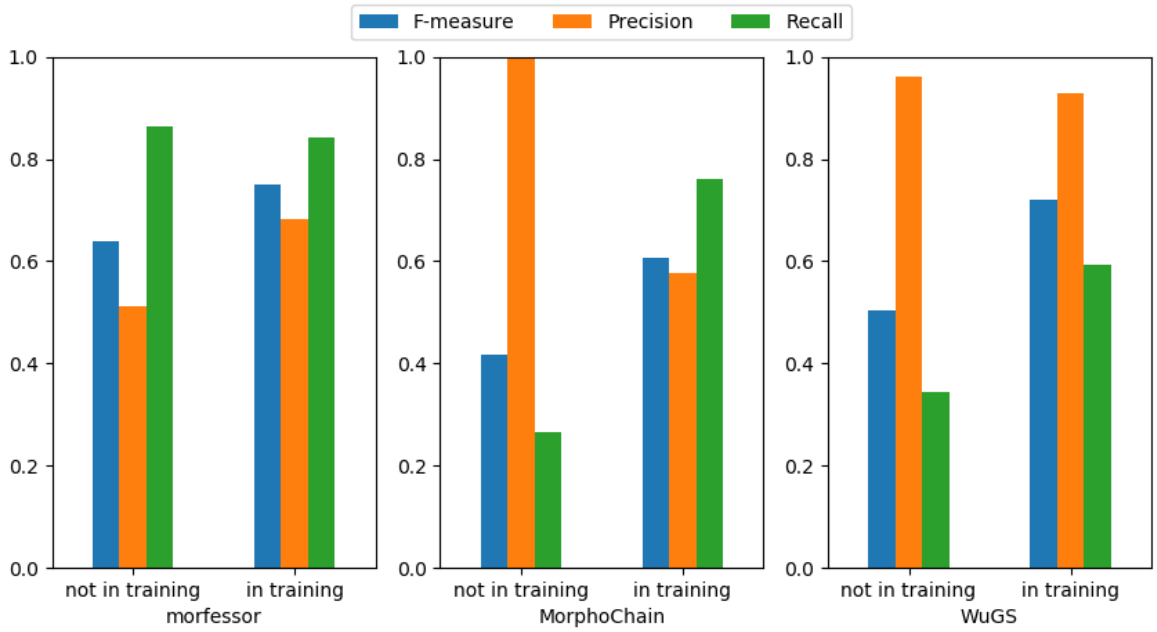
for German it is 30% and for Dutch it is 19%. Interestingly, this did not seem to have a major effect on the results as compared to the overall scores shown here. For all the data broken down by language see table B.7 in appendix B.

Another way to break down the test data is by the number of morphemes in the gold-standard. We have already hypothesized that the algorithms that rely on semantic information (WuGS and MorphoChain) typically under-segment whereas the morfessor algorithm seems to over-segment. If this is true we should see the results for the morfessor models improve as the number of morphemes increases and we should see an opposite trend for the other two.

Figure 4.4 shows the result of this experiment. The gold-standard corpus was divided according to how many morphemes are in each word. For example, the scores in the '2' column were evaluated on words that can be segmented in to exactly 2 morphemes according to the gold-standard. Both the WuGS and the MorphoChain models have very good precision scores throughout but the recall scores start off low and decrease as the number of morphemes increase. The morfessor model on the other hand starts with relatively low precision and recall scores[3] and they both increase as the number

[3]Note that the recall score will always be 1.0 when there is exactly one morpheme in the gold-standard. This is due to a quirk in how the BPR algorithm treats words with a single morpheme. Since there are no morpheme boundaries to analyze when there is only a single morpheme per word, BPR does not calculate any mismatched boundaries and assigns a perfect recall score.

Figure 4.3: Precision, recall, and F-measure reported by BPR for previously seen and novel words



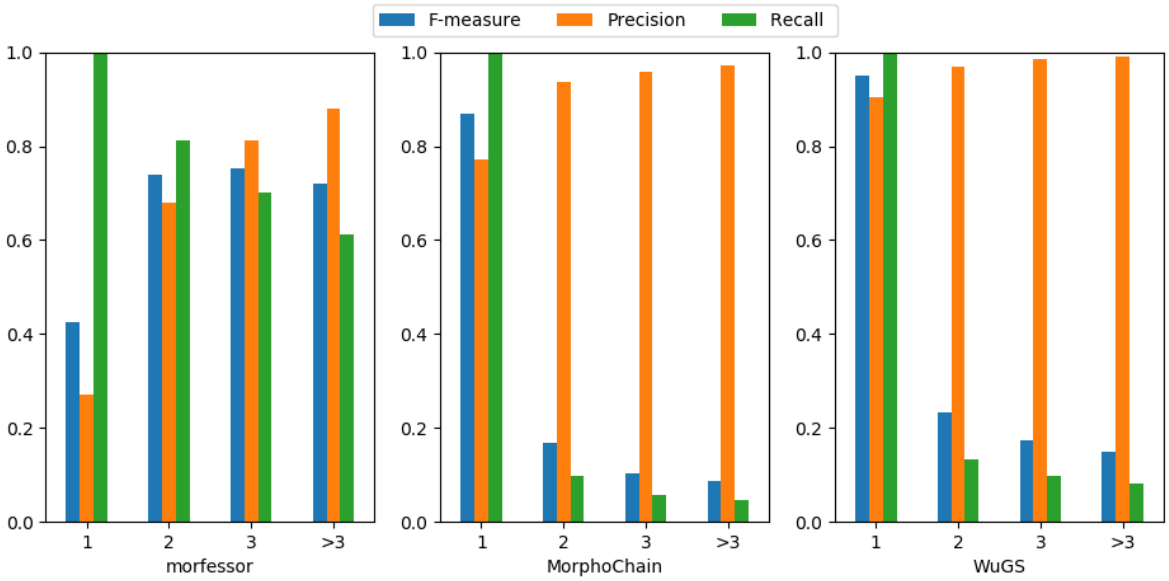
(See table B.3 for complete data)

of morphemes increases. The fact that the precision rates remain high for both WuGS and MorphoChain reaffirms that the observation that when these models do predict segmentations they are usually the correct ones.

4.2 Orthography vs. phonology

Chapter 1 discusses the fact that unsupervised morphological segmentation models may be able to produce better results for languages with more phonemic orthographies. Testing this proved to be a challenge, since simply showing that a given model performs better for a language with a more phonemic orthography is not conclusive, since many other factors can influence that result, given that two entirely different languages are being compared. Therefore, in order to test this, two models were trained for the same language: English. One model was trained with orthographic input data (the input data was not changed) and one with phonological input data. The phonological input was obtained by converting all the words in the English half of a translated parallel corpus to their phonological representation using epitran (Mortensen et al., 2018), a multilingual grapheme-to-phoneme converter.

Figure 4.4: Precision, recall, and F-measure reported by BPR by morphs per word



(See table B.4 for complete data)

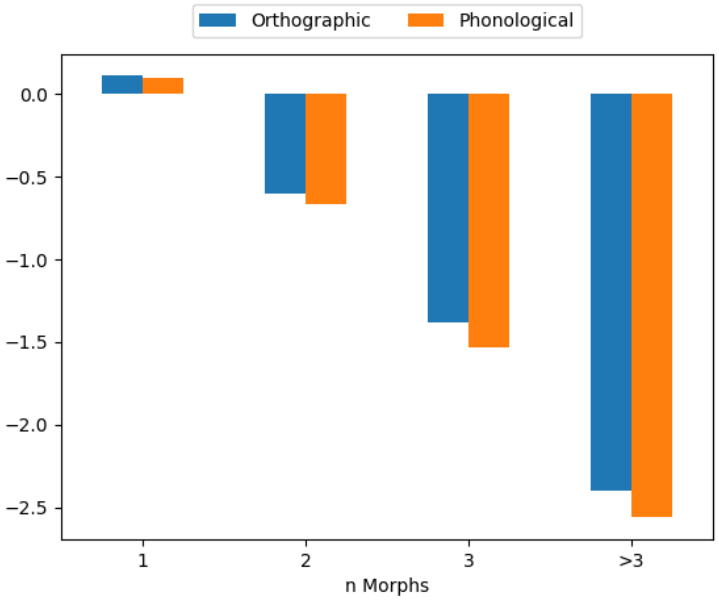
The second challenge arose because there was no gold-standard corpus containing phonological representations of morphological segmentations for English. Because of this, it was not possible to test the models in the typical way by comparing their segmentations to a gold-standard. However it is still possible to run a limited test on the phonological model by testing whether it correctly predicts the number of morphemes per word. This analysis is not as precise as the EMMA2 or BPR analyses but can be used as a rough guide to compare the performance of the orthographic or the phonological model relative to each other.

Figure 4.5 shows the results of an experiment which calculated the average difference between the number of morphemes in the (orthographic) gold-standard for a given word and the number of morphemes predicted by each model for the same word. Figure 4.5 shows the results binned according to the number of morphemes in the gold-standard segmentation. The values are predicted as follows for each model:

$$\frac{\sum_{n=1}^{|gold\_segs|} |gold\_segs_n| - |model\_segs_n|}{|gold\_segs|}$$

Where *gold_segs* is the set of gold-standard segmentations, *model_segs* is the set of segmentations for the model being tested, and where $gold\_seg_n$ and $model\_seg_n$ refer to the segmentations for the same word. A positive value indicates that the model predicted

Figure 4.5: Average morpheme count delta from gold-standard for orthographic and phonological training data



(See table B.5 for complete data)

more segmentations on average than the gold-standard and a negative value indicates that the model predicted fewer.

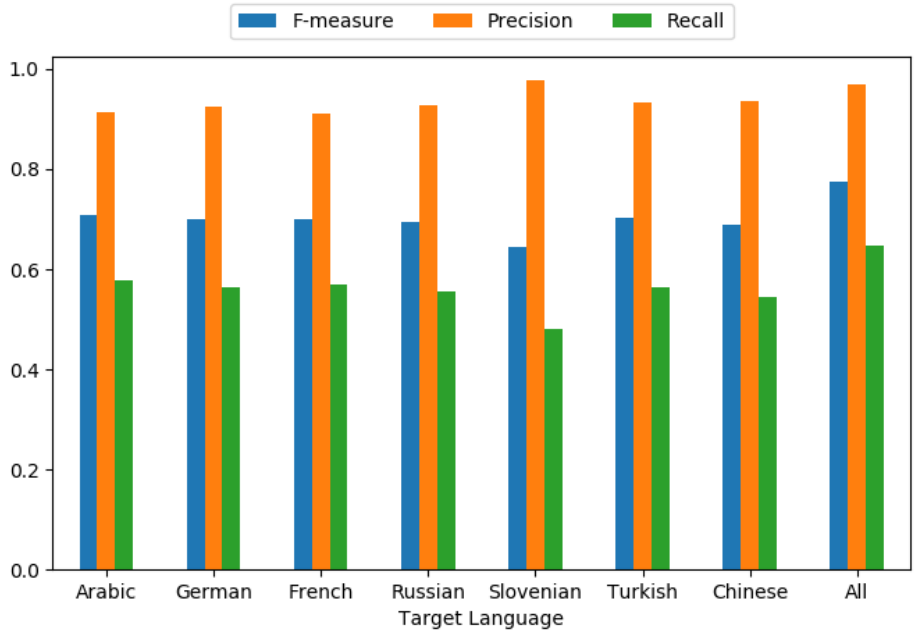
As we saw in previous experiments, both models under-segment in most cases. For example, both models predict that there are almost 1.5 fewer morphemes per word on average than the gold-standard when the gold-standard shows that there should be three morphemes.

Interestingly, the actual results from the orthographic and phonological models are very similar. The phonological model performs slightly better when the number of morphemes per word should be exactly one and slightly worse in all other contexts. This result shows that the assumption that a phonological representation will inevitably result in a more accurate model is not necessarily true. Possible reasons for why this assumption does not hold are discussed in chapter 5.

4.3 Target language variation

As discussed in chapter 1, detecting morphemes by comparing languages may produce different results depending on the languages being compared. In order to test this in the context of morphological segmentation, we created seven morphological segmentation

Figure 4.6: Precision, recall, and F-measure reported by BPR with varying target languages



(See table B.6 for complete data)

models for English trained using seven different translations[4] from the TED Talks 2013 corpus (Tiedemann, 2012), and evaluated each using BPR analysis.

Figure 4.6 shows the results of this test. The differences between the models trained on different target languages are minimal. The only slight outlier is the model trained with an English-Slovenian translation with an F-measure 5% lower than the average of the other models. This variability is probably not due to any difference in language however, and is more likely due to the fact that the English-Slovenian corpus was significantly shorter than any of the others[5].

The 'all' column in figure 4.6 shows the results from a model trained using all seven of the translated corpora. This model performs much better than any of the single language models by at least a 10% increase in the F-measure score. Note that the number of English words used to train a WuGS model would not be any higher than for any of the single language models, but the amount of semantic data would be much greater. In other words, the number of English words in the training data would not increase by adding more translations of the same text to the model's input. This allows us to rule

[4] Arabic, German, French, Russian, Slovenian, Turkish, Chinese (simplified).

[5] 14,822 sentences for Slovenian compared to all the other languages which contained between 132,640 and 159,241 sentences.

out a "reverse Slovenian effect" in which the performance increases as the raw number of words in the source language that the model is trained on increases. This indicates that increasing the amount of semantic data provided to WuGS does in fact improve its overall performance. It also shows that providing data from multiple languages also increases the amount of information available to the model instead of just repeating the same semantic information over and over again. For example, the model may learn that *music* is semantically related to *musical* from one source and that it is related to *musician* in another. This finding that combining data from a variety of languages has implications for the future improvement of the WuGS algorithm and will be discussed in more detail in chapter 5.

Chapter 5

Discussion

The goal of this paper was to describe a method for constructing a morphological segmentation model that improves on the current state of the art by reducing the tendency for over-segmentation by incorporating semantic data from parallel translated corpora into the model. This chapter discusses whether or not this goal was reached, other observations encountered in the pursuit of achieving this goal, and suggestions for future improvements.

5.1 Reducing over-segmentation with semantic data

When compared to the morfessor 2.0 algorithm (Virpioja et al., 2013), the models created using the method described in this paper do drastically reduce the amount of over-segmentation. Especially, when presented with a monomorphemic word, this model is likely to correctly parse it as containing a single morpheme. From this observation alone, we can conclude that the algorithm is successful at achieving its stated goals.

However, stopping the analysis here would be misleading. As the results of the experiments in chapter 4 show, the reduction of over-segmentation came at a price — namely an increase in under-segmentation. In other words, the advantage gained in precision was drastically offset by a decrease in recall when compared to the morfessor 2.0 baseline. However, this trade-off was not unique to WuGS — tests using the MorphoChain algorithm also showed the same pattern of high precision and low recall. This indicates that the problem may not be with any single algorithm but with an over-reliance on semantic data in general.

5.1.1 Hunting for more data

Both WuGS and the MorphoChain algorithm use semantic information (albeit from different sources) in order to decide when not to segment a given word. It seems that in many cases, the semantic information provides too strong a restriction and many segmentations that should be made by the model are prevented. In order to improve the recall of WuGS models, we should therefore limit the influence the semantic information has on making the decision to segment or not. However, blindly ignoring the semantic data in the pursuit of a better recall score would just bring us back around to an algorithm with an over-segmentation problem. The question is therefore, not how do we reduce the limiting effect of the semantic data on segmentation but how do we make better use of this data.

One possibility is to simply look for other sources of semantic data. Figure 4.6 in chapter 4 already shows that by providing additional semantic information, the recall score can be improved without negatively affecting the precision score. This is because a word is only segmented if it can be compared to another word that it is semantically related to. By increasing the amount of semantic data, we also increase the chance that the semantic relationships present in the model reflect the state of the language itself. We have already seen that more semantic data can be obtained by including multiple translations of the same text. However, this evidence is not always available, especially when we are building a model for a low-resource language where only a few parallel corpora may be available. Instead, it may be possible to incorporate semantic data extracted by other means. For example, we could use the same technique used by the MorphoChain algorithm and extract word embeddings and then use those embeddings to supplement our model's semantic network. It might even be possible to use other texts entirely. Suppose you have a single parallel corpus for a given language and several additional monolingual texts; it would be possible to use the translated corpus to initially build a semantic network which could then be supplemented with word embedding data from the monolingual texts. This raises the question of whether it is even worth extracting semantic data from parallel corpora in the first place. If semantic data can be extracted from a monolingual text using word embeddings, what is the point in restricting the type of possible input data to the model? Firstly, as previously discussed, translation data can be used to supplement monolingual data, which means you can get more information out of a single source. Secondly, the results from the experiments in chapter 4 seem to show that the quality of semantic data from translated corpora may be better than monolingual corpora for morphological segmentation tasks. This comes from the observation that WuGS seemed to outperform the MorphoChain model in most cases. This of course

could be down to the model design and not the quality of the input data, but it does seem to indicate that both sources of semantic data are valid and useful.

Finally, the model is not using all the semantic information it is given. The semantic network created by WuGS currently contains unweighted edges. However, it would be simple to add weights to the edges according to the translation probabilities used to build this network. At the moment, the translation probabilities are only considered to ensure that they exceed some minimum threshold before being added to the network. The information that weighted edges provide could theoretically be used to determine the most likely group a given word belongs to or even to help determine the most likely of several segmentation options. In practice however, any attempt to incorporate weighted edges into the model design drastically reduced the performance of the model and so any attempt to incorporate this information was abandoned. This drop in performance seemed to be due to the fact that the translation probabilities were simply messy. The word-to-word translation algorithm almost always found a large number of possible translations for each word, and the ones which looked to be the best subjectively were often not the ones with the highest translation probabilities. This meant that relying on the reported probabilities to determine degrees of semantic similarity was often misleading. No good method was discovered for reliably parsing this messy data so the weights were simply ignored. It may be possible to make use of a weighted network in future iterations of the WuGS algorithm if we can solve the puzzle of extracting useful information from this data.

5.1.2 Better data processing

Simply adding more data, whether semantic or otherwise, is not always an option. Ideally the WuGS algorithm should be more robust for all languages, even if the amount of available semantic data is limited. It is likely that the main problem lies with the fact that this model only predicts morphological segmentations for a given word if there is another word in the same group that it can be compared to; and only words that can be shown to be semantically related are ever in the same group. This means that if we have insufficient evidence of semantic relatedness, the model will not be very effective. It may be useful to look for other evidence that two words should be grouped together beyond semantic relatedness.

One possibility is to combine this approach with a more permissive segmentation algorithm like morfessor 2.0 that provides multiple possible segmentations per word. By combining the two approaches we could first collect the top n most likely segmentations

from the more permissive model and then use the morphemes predicted by these segmentations as possible group labels as a way of initializing this model (see chapter 3). In other words, we could use an existing, more permissive, morphological segmentation algorithm's predictions to provide the evidence for grouping words instead of or as well as using semantic data. It is unclear whether or not this hybrid approach would yield better results than the current models and the exploration of this hybrid model approach is left to future research.

As well as having to be semantically related, the WuGS algorithm also restricts the ability to group words together unless they share a common root morpheme. This shared root is automatically discovered using some string comparison algorithm such as the longest common substring (LCS) algorithm. The LCS approach is what is used to build all the models in chapter 4, since it seems to provide the best results after some initial tests and it was relatively easy to implement. However, this approach may also be too restrictive and it may prevent two words from joining a group if they share a common root in a way that cannot be detected by the LCS algorithm due to allomorphy. This is addressed in the step during training that combines likely groups together — but this combination step favours words that differ at morpheme boundaries as opposed to words that differ internally. For example, the algorithm may combine the groups *wak* and *wake* into a common group with the label *wak(e)*. However, the words in this group (*wake*, *awaken*, *waking*, etc.) are also semantically related to the past tense verb *woke*, which due to its irregular form is unlikely to ever be grouped with the others. This may be overcome by introducing a string comparison algorithm that takes phonological similarity into account. It may be possible that a string comparison algorithm that conflates phonological classes may be more successful. In the *woke* example, it should be able to recognize that the *a* and *o* in *wake* and *woke* are vowel sounds and so may be conflated for the purposes of grouping. The development of a phonologically motivated string comparison algorithm is of course the subject its own research project. However, if it were developed, it would be interesting to see if it could be used to improve the grouping accuracy of this model, especially for irregular words.

We have already seen that simply using a phonological representation of a given word does not always improve results overall. The initial assumption was that phonological representations of words would reduce the variability in the way morphemes are represented and would clarify the boundaries between morphemes. For example, the word *creation* can be represented phonologically using the international phonetic alphabet (IPA) as /kɹieʃn̩/ which reduces the suffix *(t)ion* down to two characters /ʃn̩/. However, this reduction in complexity in one case is offset by additional complexity elsewhere. For

example, the regular English plural suffix can be represented orthographically as *es* or *s* whereas it can be represented phonologically as any of /s/ /z/ /əz/ /ɪz/.

The use of phonological data in the experiment in chapter 4 may have been misguided but that does not mean we should rule out phonological representations entirely. As previously discussed, phonological data may still be used to reduce complexity in the representations of words by conflating phonemes that share some relevant set of features. It may be possible to use both the orthographic and phonological representations of words to determine some simplified representation which makes segmentation easier. Using a simplified phonological representation is not a new idea, early work in this area (Harris, 1955) argues for the use of a limited phonological representation over a more complex phonetic one[1]. However, this paper shows that a phonological representation may not be simple enough. The task of determining the best technique of phonological simplification is left up to future work.

5.2 Language independence

The WuGS algorithm was tested by generating morphological segmentation models for three languages; English, German and Dutch. The results from each of these languages showed that the precision scores were fairly consistent but that the recall scores for English were much higher. This raises the question of whether WuGS is truly language independent or if it has been inadvertently designed to favour some languages over others. I am a native English speaker and so there is a possibility that despite my best efforts, I have inadvertently designed an algorithm that favours English over German and Dutch. However, other than inherent bias, there may be other reasons why the results are clearly better for English.

Like the MorphoChain models, we have shown that WuGS models typically under-segment, which results in a low recall score during most tests. If a model under-segments then it will inevitably perform better when there are fewer segmentations to make (fewer morphemes per word). Consider that the English Celex2 corpus that was used for testing contains 0.25 fewer morphemes per word on average than either of the German or Dutch corpora (Baayen et al., 1995). This could explain the difference in the recall scores — an algorithm that is very conservative with regards to making segmentations is going to make the correct decision more often when there are fewer segmentations to make. This can be seen in the results from our simple baseline model that never segments anything

[1]For an explanation on the difference between a phonological and phonetic representation see Pierrehumbert (1990).

and still gets recall scores that are only 7–10% lower than the other models.

Another possible explanation is that the English data contains many fewer compound words than either the German or Dutch data[2]. Both WuGS and the MorphoChain algorithm work from an assumption that there is a single root morpheme onto which affixes are added. Compounds are only considered at segmentation time and are therefore handled as variations on single root words, not as a separate construction entirely. This assumption that compounds do not need to be treated differently from single root words may have resulted in a model that is less equipped to handle words with multiple roots.

While it is not necessarily the case that WuGS is biased specifically towards English, it is likely that it is biased towards languages that have lower segmentation rates and lower rates of compounding in general. Any future iterations of WuGS should seek to incorporate the existence of compounds at the training stage instead of just at the segmentation stage. Combining this with more flexible grouping algorithms (see section 5.1.2) will likely improve the recall scores across all languages, but specifically for those with higher segmentation rates, making WuGS more consistent for a wider variety of languages.

5.3 Additional future work

This chapter has already discussed several ways that WuGS can potentially be improved in terms of the way it processes both semantic and orthographic data. In addition to these proposed changes, we should also consider improvements that focus on the preprocessing and online training steps as well as the tests themselves.

Improving alignments

It has already been shown that the quality of the semantic data used as input to the WuGS algorithm has an effect on its accuracy. One way to improve the quality of this data is to improve the word-alignment algorithm that generates this semantic data in the first place. All the evaluations in chapter 4 use the same word alignment algorithm (Liang et al., 2006). However, this may not necessarily be the best algorithm for this purpose and alternatives should be explored. Some word alignment algorithms may even reveal semantic relationships that others do not which would mean that the best practice may be to use multiple algorithms that complement each other.

[2]In the Celex2 corpora, 20% of the English words are compounds, whereas 36% of the German words and 40% of the Dutch words are compounds. Compounds are defined as any word that contains at least two root morphemes.

Another possible way to improve semantic data with word alignment algorithms would be to explore the hypothesis that word alignments are more accurate if there are fewer word types to compare. For example, a word alignment model may say that the English word *eat*, *eats*, *eating* can be translated to French as any of *mangent*, *manger*, *mangeons*, etc. However, if the target corpus has reduced all these words to the root morpheme *mange*, this simplifies the word alignments. In turn this may make it more likely that the semantic network built from these simplified alignments will show that all of *eat*, *eats*, and *eating* are semantically related because there is a greater chance that they all can be translated as a single word.

On the basis of this hypothesis, we could try to iteratively update a model by using it to reduce the number of word types in the input data of the target language. The first step would involve creating two morphological segmentation models for both languages in the input data using the method described in chapter 3. The second step would involve using the two segmentation models to extract the root of each word for both languages in the original parallel translated corpora. Now we would have two corpora for each language, one with the the original text and another where all the words are reduced to their roots. We could then rerun the word-to-word alignment for each language by comparing the original corpus for language A with the root corpus for language B (and the original for B with the roots for A). It is likely that the models created after the second iteration of model building would be more accurate than the models from the first iteration. This method to improve the model would be time consuming but is a method that does not require any additional input data and so might be a useful way to improve a model for low-resource languages.

User-guided training

The existence of potential vertices and edges in the graph means that a model may be able to guide its own learning by forming hypotheses and then seeking out data to confirm or invalidate them. As a very basic example, it is possible to interactively train a model by iterating over all potential vertices and edges (those edges and vertices with a 'potential' label) and asking a speaker of the language either: "Is X a word?", where X is a potential vertex, or "Is the word X related to the word Y", where there is a potential edge between vertices X and Y. This interactive training would add a new way to update the semantic network that would require minimal input from a user and crucially does not require an expert with specific linguistic knowledge to make decisions.

Updating the model in this way may only have minor effects on its performance since it can only improve the model by confirming or denying the potential vertices and edges

that the model already knows about. Nevertheless, this sort of guided self-improvement is something that is unique to the models produced by WuGS.

Incorporating homographs into the model

Currently, most morphological segmentation algorithms (including this one) do not attempt to differentiate between homographs (words with a different meaning and the same spelling). In some cases these can be differentiated by incorporating phonological information into the model but this only helps in the case where a homograph is also a heteronym (homograph that has a different pronunciation). A more reliable way to differentiate may be to use the semantic data instead. Homographs may be detected by looking for words that are highly associated to two or more distinct clusters of vertices in the graph. A cluster is a group of vertices that have a lot of connecting edges and distinct clusters are those that do not have many edges connecting the vertices in the clusters. In other words, homographs can be considered 'hubs' which act as the sole connection between two clusters.

Although the structure of this model and the fact that semantic information already exists in the model makes it easy to detect homographs, discovering how to treat these words for segmentation purposes is another problem entirely. Some homograph pairs should be segmented in the same way even though the meaning may be different. For example, both *boring* (tedious) and *boring* (making a hole) can be segmented as *bor(e)-ing*. On the other hand, *evening* (end of the day) should not be segmented whereas *evening* (making even) should be segmented as *even-ing*. Differentiating between these two types of homomorphs is not a trivial task but the key may be in correctly defining the clusters that a given word belongs to. In the *evening* case we may see that one cluster contains words like *night*, *supper*, etc. and another containing *smooth*, *iron*, but also *even*, *evens*, etc. The existence of words with the same root as *evening* in one cluster but not the other may help the model detect that the word should be treated differently in these two contexts. It may be necessary to put some limits on this procedure, since a word may appear to be a homograph simply because there is not enough information available during training to concatenate two clusters that should be one. The implementation of an algorithm that is aware of the existence of homographs may require significant additional work but WuGS may be a good starting point given that it already defines a network that can be used to differentiate words with multiple meanings.

5.4 Conclusion

The WuGS algorithm presented in this paper does achieve the goal of reducing over-segmentation and it does so by incorporating semantic data in a novel way. However, there are still many ways that WuGS can be improved, especially with regards to improving its recall capabilities. The main weakness of WuGS is that it requires too much information to decide to make a segmentation. This overly cautious approach results in under-segmentation in most cases. However, by comparing WuGS to another which also relies on semantic information, we show that this weakness is actually a feature of algorithms that rely on semantic information in general as opposed to being unique to this one specifically. Additionally, by comparing the performance of multiple language-models we know that WuGS under-performs when there are multiple root morphemes present in a word. Any future work should be better at modeling the relationship between multiple roots in a given word.

One major advantage of WuGS is that it automatically discovers allomorphs as a side-effect of the training process. Currently allomorphs that differ only at the right or left edge of a given morpheme can be discovered. Incorporating detection of allomorphs with internal variation may require the automatic detection of grapheme or phoneme classes, which would be an interesting direction to take future research.

WuGS is far from perfect, but experiments show that it can be competitive with other algorithms currently in use. Also, it shows that parallel translated corpora are a useful source of semantic input data for use in morphological segmentation algorithms in general and can be used as an alternative or a supplement to word embeddings. Furthermore, the network structure allow us to explore novel ways to train models and potentially allows us to make use of clusters to tackle previously ignored problems such as homographs.

Bibliography

H. R. Baayen, R. Piepenbrock, and L. Gulikers. The CELEX lexical database, 1995.

E. M. Bender. Linguistically naïve != language independent: Why NLP needs linguistic typology. In *Proceedings of the EACL 2009 Workshop on the Interaction Between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, ILCL '09, pages 26–32, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1642038.1642044`.

R. Cotterell, A. Kumar, and H. Schütze. Morphological segmentation inside-out. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2325–2330, 2016a.

R. Cotterell, T. Vieira, and H. Schütze. A joint model of orthography and morphological segmentation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 664–669, 2016b.

M. Creutz and K. Lagus. Inducing the morphological lexicon of a natural language from unannotated text. 2005.

M. Creutz and K. Lagus. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)*, 4 (1):3, 2007.

M. S. Dryer and M. Haspelmath, editors. *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig, 2013. URL `https://wals.info/`.

R. Eskander, O. Rambow, and T. Yang. Extending the use of adaptor grammars for unsupervised morphological segmentation of unseen languages. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 900–910, 2016.

J. A. Goldsmith. *Segmentation and Morphology*, chapter 14, pages 364–393. Wiley-Blackwell, 2010. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781444324044.ch14.

Z. S. Harris. From phoneme to morpheme. *Language*, 31(2):190–222, 1955. URL http://www.jstor.org/stable/411036.

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

M. Haspelmath. *Understanding morphology*. 2002. ISBN 9780340760260.

R. Jorma. *Stochastic complexity in statistical inquiry*, volume 15. World scientific, 1998.

O. Kohonen, S. Virpioja, and K. Lagus. Semi-supervised learning of concatenative morphology. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 78–86. Association for Computational Linguistics, 2010.

M. G. Kossmann. *Grammaire du parler berbre de Figuig (Maroc oriental)*. Peeters, Paris, 1997.

M. Kurimo, S. Virpioja, V. T. Turunen, G. W. Blackwood, and W. Byrne. Overview and results of morpho challenge 2009. In *Workshop of the Cross-Language Evaluation Forum for European Languages*, pages 578–597. 2009.

P. Liang, B. Taskar, and D. Klein. Alignment by agreement. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 104–111. Association for Computational Linguistics, 2006.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013. URL http://arxiv.org/abs/1301.3781.

D. R. Mortensen, S. Dalmia, and P. Littell. Epitran: Precision G2P for many languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Paris, France, May 2018. European Language Resources Association (ELRA).

K. Narasimhan, R. Barzilay, and T. Jaakkola. An unsupervised method for uncovering morphological chains. *Transactions of the Association for Computational Linguistics*, 3:157–167, 2015. URL http://aclweb.org/anthology/Q15-1012.

J. Pierrehumbert. Phonological and phonetic representation. *Journal of phonetics*, 18 (3):375–394, 1990.

C. Quiles, K. Kūriákī, and F. López-Menchero. *A Grammar of Modern Indo-European*. Indo-European Language Association, 2012. ISBN 9781461022138.

T. Ruokolainen, O. Kohonen, S. Virpioja, et al. Painless semi-supervised morphological segmentation using conditional random fields. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 84–89, 2014.

T. Ruokolainen, O. Kohonen, K. Sirts, S.-A. Grönroos, M. Kurimo, and S. Virpioja. A comparative study of minimally supervised morphological segmentation. *Computational Linguistics*, 42(1):91–120, 2016.

P. Schone and D. Jurafsky. Knowledge-free induction of morphology using latent semantic analysis. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning*, volume 7, pages 67–72. Association for Computational Linguistics, 2000.

E. O. Selkirk. *The syntax of words*. Cambridge, Mass MIT Press, 1982.

K. Sirts and S. Goldwater. Minimally-supervised morphological segmentation using adaptor grammars. *Transactions of the Association of Computational Linguistics*, 1:255–266, 2013.

S. Spiegler and C. Monson. Emma: a novel evaluation metric for morphological analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1029–1037. Association for Computational Linguistics, 2010.

J. Tiedemann. Parallel data, tools and interfaces in opus. In *Lrec*, volume 2012, pages 2214–2218, 2012.

S. Virpioja, P. Smit, S.-A. Grnroos, and M. Kurimo. Morfessor 2.0: Python implementation and extensions for morfessor baseline. Technical report, 2013. URL `http://urn.fi/URN:ISBN:978-952-60-5501-5`.

Appendices

Appendix A

Additional Algorithms

Algorithm 7 edge to hash

```
# this function takes an array of edges represented by a
# triple (wordA, wordB, translation_probability) and a
# minimum probability and returns a hash table where keys
# are all wordA values and values are an array of all
# translations (wordB) of the key that have a translation
# probability above or at the min_prob threshold
FUNCTION edge_to_hash(Edges, min_prob)
    hash = Hash() # a new hash table
    FOR edge IN Edges
        word, translation, prob = edge
        IF prob >= min_prob
            IF hash.has_key(word)
                hash[word] = hash[word] + [translation]
            ELSE
                hash[word] = [translation]
    return hash
```

Algorithm 8 get affixes

```
# this function takes a list of words with a common root and splits
# each word using the root as a separator. The remaining parts of
# each word are considered affixes and are returned in a list
FUNCTION get_affixes(words, root)
    all_affixes = []
    FOR word IN words
        affixes = words.split(root) # split the word using the root as a separator
        FOR affix IN affixes
            all_affixes.append(affix)
    return all_affixes
```

Algorithm 9 get groups

```
# this function is a reverse look up function that
# returns all keys of the hash groups whose value
# contains the string word
FUNCTION get_groups(groups, word)
    all_groups = []
    FOR group IN groups
        IF word IN groups[group]
            all_groups.append(group)
    return all_groups
```

Algorithm 10 From word translation to semantic network using symmetric relations

```
FUNCTION create_semantic_network2(edges, min_prob)
    hash = edges_to_hash(edges, min_prob) # see helper functions
    network = Hash() # a new hash table
    FOR word_a IN hash
        translations_a = hash[word_a]
        FOR word_b IN hash
            IF word_a != word_b
                translations_b = hash[word_b]
                IF translations_a INTERSECTS translations_b
                    IF NOT network.has_key(word_a)
                        network[word_a] = []
                    network[word_a] = network[word_a] + [word_b]
    return network

min_prob = 0.2 # for example
SemNetA = create_semantic_network2(EdgesA, min_prob)
SemNetB = create_semantic_network2(EdgesB, min_prob)
```

Appendix B

Data Tables

Table B.1: Precision, recall, and F-measure reported by EMMA2

Model	Language	F-measure	Precision	Recall
morfessor	de	0.685	0.601	0.796
	en	0.706	0.589	0.882
	nl	0.715	0.616	0.853
MorphoChain	de	0.515	0.851	0.369
	en	0.673	0.955	0.519
	nl	0.471	0.892	0.320
WuGS	de	0.499	0.945	0.339
	en	0.729	0.924	0.602
	nl	0.477	0.958	0.318

(See figure 4.1 for a visualization of this data)

Table B.2: Precision, recall, and F-measure reported by BPR

Model	Language	F-measure	Precision	Recall
morfessor	en	0.615	0.485	0.839
	de	0.695	0.605	0.816
	nl	0.746	0.626	0.921
MorphoChain	en	0.612	0.945	0.452
	de	0.535	0.838	0.393
	nl	0.522	0.881	0.371
WuGS	en	0.688	0.935	0.545
	de	0.512	0.956	0.349
	nl	0.531	0.958	0.367
No Segs	en	-	-	0.447
	de	-	-	0.278
	nl	-	-	0.305

(See figure 4.2 for a visualization of this data)

Table B.3: Precision, recall, and F-measure reported by BPR for previously seen and novel words

Model	In training	F-measure	Precision	Recall
morfessor	not in training	0.640	0.513	0.863
	in training	0.750	0.683	0.843
MorphoChain	not in training	0.418	1.000	0.267
	in training	0.606	0.578	0.762
WuGS	not in training	0.503	0.960	0.345
	in training	0.721	0.928	0.593

(See figure 4.3 for a visualization of this data)

Table B.4: Precision, recall, and F-measure reported by BPR by morphs per word

Model	n Morphs	F-measure	Precision	Recall
morfessor	1	0.425	0.272	1.000
	2	0.740	0.680	0.812
	3	0.752	0.813	0.701
	>3	0.721	0.881	0.611
MorphoChain	1	0.868	0.771	1.000
	2	0.169	0.937	0.098
	3	0.104	0.958	0.057
	>3	0.087	0.971	0.047
WuGS	1	0.949	0.903	1.000
	2	0.233	0.968	0.134
	3	0.174	0.985	0.098
	>3	0.149	0.992	0.082

(See figure 4.4 for a visualization of this data)

Table B.5: Average morpheme count delta from gold-standard for orthographic and phonological training data

n Morphs	Orthographic	Phonological
1	0.111	0.097
2	−0.602	−0.669
3	−1.379	−1.531
>3	−2.402	−2.561

(See figure 4.5 for a visualization of this data)

Table B.6: Precision, recall, and F-measure reported by BPR with varying target languages

Target Language	F-measure	Precision	Recall
Arabic (ar)	0.707	0.914	0.577
German (de)	0.700	0.925	0.563
French (fr)	0.701	0.911	0.570
Russian (ru)	0.694	0.928	0.555
Slovenian (sl)	0.644	0.977	0.480
Turkish (tr)	0.704	0.933	0.565
Chinese (zh)	0.688	0.935	0.545
All	0.776	0.969	0.648

(See figure 4.6 for a visualization of this data)

Table B.7: Precision, recall, and F-measure reported by BPR for previously seen and novel words by language

Model	In training	Language	F-measure	Precision	Recall
morfessor	in training	en	0.688	0.579	0.849
		de	0.768	0.772	0.764
		nl	0.793	0.699	0.916
	not in training	en	0.536	0.396	0.830
		de	0.651	0.533	0.839
		nl	0.734	0.609	0.922
MorphoChain	in training	en	0.689	0.889	0.562
		de	0.591	0.464	0.815
		nl	0.537	0.381	0.908
	not in training	en	0.516	0.999	0.348
		de	0.348	1.000	0.211
		nl	0.390	1.000	0.242
WuGS	in training	en	0.757	0.914	0.646
		de	0.659	0.944	0.506
		nl	0.747	0.927	0.625
	not in training	en	0.610	0.955	0.448
		de	0.435	0.961	0.281
		nl	0.465	0.965	0.306

Table B.8: Precision, recall, and F-measure reported by BPR by morphs per word and language

Model	n Morphs	Language	F-measure	Precision	Recall
morfessor	en	1	0.444	0.286	1.000
		2	0.657	0.611	0.710
		3	0.755	0.808	0.709
		>3	0.749	0.894	0.644
	de	1	0.494	0.328	1.000
		2	0.718	0.657	0.791
		3	0.707	0.788	0.641
		>3	0.660	0.836	0.546
	nl	1	0.337	0.203	1.000
		2	0.845	0.771	0.936
		3	0.795	0.842	0.752
		>3	0.754	0.913	0.643
MorphoChain	en	1	0.939	0.885	1.000
		2	0.018	0.993	0.009
		3	0.008	0.998	0.004
		>3	0.002	1.000	0.001
	de	1	0.813	0.684	1.000
		2	0.296	0.886	0.177
		3	0.202	0.916	0.114
		>3	0.173	0.941	0.095
	nl	1	0.852	0.743	1.000
		2	0.195	0.931	0.109
		3	0.100	0.961	0.053
		>3	0.085	0.971	0.045
WuGS	en	1	0.948	0.902	1.000
		2	0.297	0.958	0.176
		3	0.296	0.980	0.174
		>3	0.248	0.995	0.142
	de	1	0.952	0.909	1.000
		2	0.218	0.965	0.123
		3	0.099	0.990	0.052
		>3	0.091	0.992	0.048
	nl	1	0.947	0.899	1.000
		2	0.185	0.981	0.102
		3	0.128	0.984	0.068
		>3	0.109	0.989	0.058