Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 3842–3850
Marseille, 11–16 May 2020

An unsupervised method for weighting finite-state morphological analyzers

Amr Keleg, Francis M. Tyers, Nicholas Howell, Tommi A. Pirinen
Faculty of Engineering, Department of Linguistics, School of Linguistics, Hamburger Zentrum für Sprachkorpora
Ain Shams University, Indiana University, Higher School of Economics, Hamburg University
Cairo, Bloomington, Moscow, Hamburg
amr.keleg@eng.asu.edu.eg, ftyers@iu.edu, nlhowell@gmail.com, tommi.antero.pirinen@uni-hamburg.de

Abstract

Morphological analysis is one of the tasks that have been studied for years. Different techniques have been used to develop models for performing morphological analysis. Models based on finite state transducers have proved to be more suitable for languages with low available resources. In this paper, we have developed a method for weighting a morphological analyzer built using finite state transducers in order to disambiguate its results. The method is based on a word2vec model that is trained in a completely unsupervised way using raw untagged corpora and is able to capture the semantic meaning of the words. Most of the methods used for disambiguating the results of a morphological analyzer relied on having tagged corpora that need to manually built. Additionally, the method developed uses information about the token irrespective of its context unlike most of the other techniques that heavily rely on the word's context to disambiguate its set of candidate analyses.

Keywords: FSTs, FST weighting, constraint grammar, word2vec

1. Introduction

Morphological Analysis is the task of mapping an input token to its morphemes. The morphological analysis of a token can change depending on its context (e.g.: The word wound is a verb <v> in the sentence "the device is wound with copper wire" and a singular noun <n><sg> in "a knife wound"). Researchers have been working on building models for languages using different techniques. For low resourced language, it's not an easy task to find / build a sufficiently large tagged corpus that can be used to train supervised machine learning models.

Finite state transducers (FSTs) map a set of input sequences to a set of output sequences. A transducer is defined by an input alphabet A, an output alphabet B , a set of states S, a set of initial states I, a set of final states F and a set of transitions between different states of the transducer T where $I \in S, F \in S, T \in S \times (A \cup \{\epsilon\}) \times (B \cup \{\epsilon\}) \times S$. Each transition has an input token, output token, source state and destination state. Given an input sequence, the output sequences correspond to the set of valid paths between the initial states and final states passing through a set of intermediate states. Finite state transducers have been used in morphologically analyzing text to convert tokens from the surface form to the lexical one (Beesley, 2003). These transducers are non-determinstic if multiple output sequences can be generated for the same input sequence/pattern. A weighthed finite state transducer associates a weight to each transition. The total weight of a path is corresponding to the multiplication of the path's transitions weights. For a tropical semiring, the multiplication operator corresponds to the algebric summation such that the total weight of a path is computed as the summation of the weights (Mohri, 2004).

In this paper, we are investigating the usage of word2vec as a word embedding model to disambiguate the analyses generated by a finite state transducer. A weighted morphological analyzer has a method of ordering for the ambiguous set of analyses for an input token. Our model is completely unsupervised and uses information about the word only irrespective of its context. Adding weights to a morphological analyzer will help in finding the most common morphological analysis for each token and also help in finding the most common tag for a morphologically unambiguous token (e.g.: The word fine can have two valid analyses, it can be considered to be an adjective <adj>as in "This is a fine house", and can be considered to be a singular noun <n><sg>as in "I got a fine".). This paper is organized as follows: Section 2 gives a brief overview of the research done in disambiguating morphological analyzers, Section 3 explains the background of building weighted morphological analyzers using finite state transducers, Section 4 describes the reference methods and the word2vec based method that are used for weighting transducers, Section 5 shows the experimental setup for the word2vec method, Section 6 reports the results for the weighting methods and Section 7 provides our conclusion of the experiments.

2. Related Work

The task of disambiguating the morphological analyses for a certain input token has been discussed for years. Yuret et al. (2006) used a supervised approach to learn a set of rules that can be used for disambiguating the analyses. They used an algorithm called GPA (Greedy Prepend Algorithm) which is an update to the Prepend Algorithm (Webb, 1993). Initially, there is a single rule that matches all the input tokens and assigns them to the most common class/tag in the training set. Then, New candidate rules are generated by prepending an attribute to all the rules that currently exist in the list. The gain of each rule is defined as the number of instances that were wrongly classified and will get classified correctly if the rule is prepended to the set of rules. Finally, the rule with maximum gain will be prepended to the set of rules, which means it will have higher priority than the previous ones. This seems logical since new rules act as special cases to the old ones.

Another way to build a rule-based morphological analyzer is to depend on a constraint grammar to disambiguate the results of the tagger (Uí Dhonnchadha and Van Genabith, 2006).

Schiller (2006) demonstrated a way to weight a finite state transducer for segmenting tokens into their constituent compounds. Although Segmentation differs from Morphological Analysis, The research demonstrated how using unsupervised rules can lead to better models. The author's intuition is to favor compounds with less number of segments. Then, using a training corpus, the uni-gram counts of token-compound pairs are used to disambiguate between compounds having the same number of segments.
Sánchez-Martínez et al. (2008) developed a way to build a part of speech tagger / disambiguator by training a Hidden Markov Model that is used in a pipeline of a machine translator. The authors proved how making use of information from the target language yields better disambiguators.
Shen et al. (2016) explored the usage of deep learning techniques for the morphological disambiguation task through Long short term-memory(LSTM)-based neural architectures depending on features extracted from the word and its context. They used two different Embedding models, One for the list of candidate analyses of the token and the other for the context's words and their respective analyses. These deep learning models depend on having large tagged data-sets that can be used to optimize the cost function.

3. Background

Finite state transducers can be used to transform words from surface form to lexical form. A FST is a directed graph having a set of initial and final states. Each edge in a weighted FST is characterized by three values: an input token, an output token and a weight taking the form `Input token:Output token/Weight`. Moreover, the final states may have weight values associated to them. For an input surface word, the corresponding lexical forms (analyses) can be generated by finding the set of paths that start from one of the initial states, end at one of the final states and pass through a set of edges such that the concatenation of the edges' input tokens matches the input surface word. A FST may have epsilon transitions (in the form ε`:Output token/Weight`) which are edges that generate an output token without consuming an input token. For a tropical semi-ring implementation of FSTs, the weight of a path is the summation of the weights of the path's edges and the weight of the final state.
Figure 1 shows how a simple FST can be used to transform the word `euro` to its corresponding lexical forms `euro<n><sg>` and `euro<n><pl>`. Initially, the weights of the edges and the final states are equal to zero. To disambiguate the results of the analyzer, different paths should have different weights such that the most probable path has the least weight.
One way to weight a FST is achieved by an operation called Composition. Composition is an operation to join two FSTs together. If the first FST is used to map a set of input tokens X to a set of output tokens Y and the second FST is used to map a set of input tokens Y to a set of output tokens Z, then composing both FSTs together will result in a FST that maps the set of input tokens X directly to the set of output tokens Z.
Figure 2 shows an example for a FST that can be used to weight the morphological analysis FST. The second

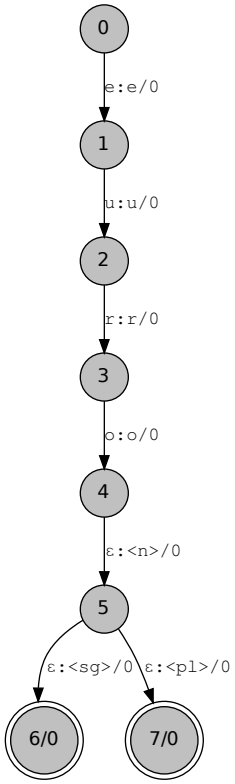

Figure 1: An unweighted FST

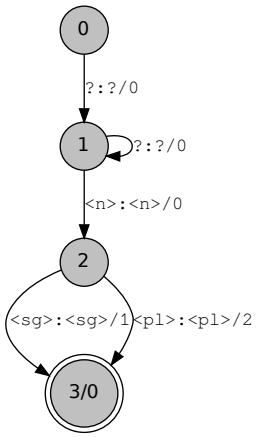

Figure 2: A weighted FST

FST weights any input with a suffix `<n><sg>` to 1 and weights any input with a suffix `<n><pl>` to 2. This FST is generated from two regular expressions in the form `?+<n><sg>::1` and `?+<n><pl>::2` where `?` means any token from the FST's alphabet and `?+` means that the input word must have a prefix of at least one token.

Using composition, a weighted morphological analyzer is formed as shown in Figure 3. This example shows that in order to weight a morphological analyzer, a set of weighted

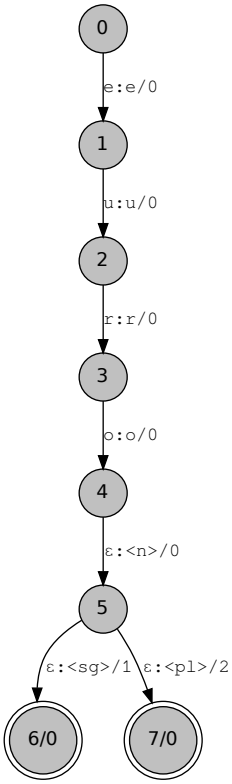

Figure 3: The composed FST

regular expressions (weightlist) needs to be formed so that each path in the analyzer is associated with a certain weight.

Another simple FST operation is the difference operation that resembles the difference operator between two sets. It removes all the paths that are found in the second FST from the first one. This operation is used in the case of having multiple successive weightlists.

4. Development

All the methods except the analysis length method generate a regexp weightlist in the form of mapping an unweighted lexical form to a weighted lexical form. This regexp weightlist is compiled and composed with the original unweighted FST (mapping surface form to lexical form) to generate a weighted FST. In the following section, we will describe how weightlists are generated in the reference methods and the word2vec method. In most of the models, we need to estimate the weights of unseen tokens (lexical forms). Two different methods were used for estimating the counts of these tokens. The first one is Laplace smoothing with simply adds a value of 1 to the counts of all the seen tokens and assumes that any unseen token has a count of 1. The second method is based on Good-Turing smoothing. We have used the simple good-turing implementation (Gale and Sampson, 1995) which mainly aims at adjusting the counts of seen tokens and estimating the counts of all the unseen tokens.

4.1. Reference models

We have evaluated the usage of different reference methods to compare them with the word2vec based method. These reference models are based on statistical theories or based on having a heuristic for weighting analyses.

4.1.1. Unigram-counts based weightlist

The unigram-counts based method is a supervised technique. It was proved that the unigram method can successfully disambiguate morphological analyses for Finnish (Lindn et al., 2009). Given a tagged corpus in the form (surface form, analysis), The method estimates the probability of a certain analysis, a given a surface form, s as $P(a|s)$. Using Bayes' theorem, The probability of an analysis given a surface form can be computed as in (1).

$$P(a|s) = \frac{P(s|a)P(a)}{P(s)}. \quad (1)$$

The method needs to estimate the values for the three terms $P(s|a)$, $P(a)$ and $P(s)$. The term $P(s|a)$ is the probability of a surface form s given that it has a lexical form a. This term will always be equal to 1. e.g: For the analysis `cat<n><pl>`, the only valid surface form is `cats`. Thus, $P(s|a) = 1$ if analysis is one of the possible analyses for the surface form ($P(\text{cats}|\text{cat} < \text{n} >< \text{pl} >) = 1$). Since the denominator term $P(s)$ is common for all the valid analyses of the input surface form, then this term can be omitted without affecting the order of the conditional probabilities. Therefore, $P(a|s)$ can be substituted by $P(a)$. Doing so will not make it possible to interpret the weights in a probabilistic way but it will not affect the order of these probabilities. The value $P(a)$ can be estimated as number of occurrences of analysis divided by the size of tagged corpus.

Instead of calculating small floating values for the probability, Tropical semirings are used such that the weight is equivalent to -log(probability). Thus the weight will be $-\log(P(a))$ which equals $-\log(N_a) + \log(|C|)$, where N_a is the number of occurrences of analysis a and $|C|$ is the size of the corpus.

Finally, a smoothing method is used to estimate the count of unseen tokens. We have tested both Laplace smoothing and Simple Good-Turing methods to generate a secondary weightlist for the tokens that weren't part of the training corpus. So, this unigram-counts based model will end-up generating two different weightlists:

- A weightlist based on the unigram counts of each analysis in the corpus.
- A default weightlist based on Laplace smoothing or simple Good-Turing smoothing.

4.1.2. Constraint Grammar-based weightlist

A constraint grammar is a set of rules for selecting or removing certain analyses given the lexical forms of the previous/next tokens (Karlsson, 1990). For English, a constraint grammar might have a rule that selects the infinitive analysis of a verb if it's preceded by the word `to`. The rule written in CG-3 format will be `"SELECT Inf IF (0C V) (-1C To) ;"` which can interpreted as select

the infinitive analysis if the current token is a verb and the previous token is to.
Given a large unannotated corpus, First the list of candidate analyses for the whole corpus is generated. Then, the Constraint Grammar is used to filter the lists of analyses. The remaining candidate analyses are considered to be a tagged corpus that is used to estimate the weightlist in the same fashion as the unigram based method.
Since this method makes use of a pre-built constraint grammar then it can be considered as a semi-unsupervised model.

4.1.3. Random weightlist

The random method represents the baseline for all the other models. Given a a raw unlabeled corpus and unweighted FST, Weights are generated as follows:

- Each token is analyzed using the unweighted FST generating multiple ambiguous analyses/lexical forms.
- A random analysis is chosen and added to a weighting corpus.
- The randomly selected analyses for the tokens are treated as a tagged corpus that is used to estimate the weights of the analyses.

The method breaks down any regularity/heuristic that could have improved the weighting process. This method's evaluation metrics act as a lower bound for the metrics of all the other methods.

4.1.4. Equal weightlist

The equally-probable method assigns a weight of one to all the analyses. After weighting the FST, the first candidate out of the list of valid lexical analyses is selected as the correct analysis. This method evaluates the default order of the analyses reached by the analyzer.

4.1.5. Analysis Length weightlist

The analysis length method depends on a heuristic that shorter analyses are more probable than long ones. Therefore, Complex analyses having many tags will have larger weight (less probability) than simple/short ones. One way to achieve such weighting is to assign a weight of 1 to all the edges of a FST.

4.2. The word2vec-based weightlist

Word embedding models have proved to be successful in capturing the semantic similarities between words even if they are syntactically different. Word2vec is one of the most popular models used for training word embedding models (Mikolov et al., 2013). After training a word2vec model using large raw corpus (such as: wikipedia dumps), a smaller untagged corpus is used. For each token in the corpus, a list of the top 10 similar words to the token is generated by finding the words whose vectors have the highest cosine similarity with the input token's vector. Then, the analyses of all the similar words is found using the unweighted morphological analyzer and all the ambiguous similar words (the words having more than one analysis) are discarded. Finally, the analyses of unambiguous similar words are used to disambiguate the analysis of the current token.

Table 1: Most common analyses of the unambiguous similar words of the word `wound`

Tag	Count
<n><sg>	7
<n><pl>	1

Example: For the token "wound", the possible analyses are:

wound	wind<vblex><pp>/ wind<vblex><past>/ wound<n><sg>/ wound<vblex><pres>/ wound<vblex><inf>/ wound<vblex><imp>

The words similar to the token "wound" using just the cosine similarity between the token and the vectors of the other tokens in the vocabulary without making use of the token's context are:

wounds	wound<n><pl>/ wound<vblex><pres><p3><sg>
injuries	injury<n><pl>
injury	injury<n><sg>
neck	neck<n><sg>
chest	chest<n><sg>
blow	blow<n><sg>/ blow<vblex><inf>/ blow<vblex><pres>/ blow<vblex><imp>
cord	cord<n><sg>
ulcer	ulcer<n><sg>
tendon	tendon<n><sg>
surgery	surgery<n><sg>

The ambiguous similar words (`wounds` and `blow`) will be discarded then a tag count for the tags of the remaining similar words' analyses is formed. (The most common tags among the analyses of the similar words are found in Table 1). Finally, the most common tags are compared to the possible analyses of the current token and the matching tags are weighted using the unigram tag counts. (i.e.: For the token "wound" the tag <n><sg> is the most probable analysis and is a possible analysis for the input token. On the other hand, the tag <n><pl> isn't a possible analysis for the input token thus it's ignored).

4.3. How are weightlists used

The generated weightlists using different methods are in the form of weighted analyses. These weightlists can be compiled into FSTs that transform an unweighted analysis into a weighted one. FST composition is then used to generate a FST that transforms an input surface form into a weighted analysis. The first FST is the unweighted morphological analyzer that maps surface forms to lexical forms. And the second FST is one that maps unweighted lexical forms to weighted ones. Each weighted analysis in the weightlist is converted a weighted FST using `hfst-regexp2fst`. These FSTs are disjuncted together to generate a single FST for mapping lexical forms to weighted lexical forms. On

composing both FSTs, a weighted FST mapping surface form to weighted lexical forms will be generated.
However, if the second FST doesn't have a path for a certain analysis then the surface-form:analysis pair will be dropped. To avoid this we use FST difference to find all the paths that were part of the unweighted FST and were dropped in the final composed FST. Then, we use another fallback weightlist for these remaining analyses.
Our implementation allows the usage of a set of weightlists such that the second weightlist acts as a fallback for the first weightlist and the third weightlist acts as a fallback for the first two and so on. We also need to make sure that the weights in the second weightlist are larger than those of the first one such that those paths are less favorable. Additionally, The final and default weightlist makes use of Laplace Smoothing such that every path that was part of the unweighted FST is added to the final weighted FST.

5. Experiments

Different weighting methods are implemented as a set of shell scripts and Python 3 scripts [1]. Apertium's lttoolbox (lexical toolbox) and hfst are used to apply different FST operations and transformations. Linguistic resources such as: unweighted morphological analyzers, tagged corpora and constraint grammars for English, Kazakh (Washington et al., 2014) and Serbo-Croatian[2] that were used throughout the experiments are actually available as part of Apertium's resources.

5.1. Word2vec models

Word2vec model has two common architectures: Continous Bag of Words (CBoW) and Skip-gram (SG). According to the results reached by Mikolov et al. (2013) on the Semantic-Syntactic Word Relationship test set, Models based on the skip-gram architecture perform better on semantic similarity tasks while the continuous bag of words perform better on syntactic similarity tasks. In our model, we are more interested in using word2vec models that can find the words semantically similar to a given word so we used the Skip-gram version of the word2vec models (e.g: for the word `wound`, a semantically similar word would be `injury` which can be used to disambiguate the morphological analysis of `wound`. On the other hand, a syntactically similar word would be `wounding` which isn't useful in disambiguating the input word as they don't share similar morphological analyses).
Fares et al. (2017) showed that researchers don't pay great attention to optimizing the training process of word2vec models. The trained models aren't reproducible due to the randomized nature of the architecture. Additionally, hyperparameter optimizations are required to build models that can perform well on the given task. They have also publicly shared a set of pre-trained models as a solution to the reproducibility issues. Pre-trained models for English and Kazakh only are avialable and we have used these models to evaluate our weighting method.

Table 2: Hyperparameters settings of the word2vec model

Hyperparameter	Value
model architecture	Skip-gram
minimum count	5
window size	2, 5, 10
vector size	100, 200, 300
downsampling threshold	10^{-5}
initial learning rate	0.025
minimum learning rate	0.0001
negative sampling	20
epochs	5
random seed	42

The three pretrained models that were used are: A Word2Vec Continuous Skipgram (ID:18) trained using gensim with vector size of 300 and window size of 5 trained using the English Wikipedia Dump of February 2017 with vocabulary size 291186, a Word2Vec Continuous Skipgram (ID:40) trained using gensim with vector size of 100 and window size of 10 trained using the English CoNLL17 corpus with vocabulary size 4027169 and a Word2Vec Continuous Skipgram (ID:54) with vector size of 100 and window size of 10 trained using the Kazakh CoNLL17 corpus of vocabulary size 176643.
Moreover, we have trained multiple word2vec models from scratch to investigate the effect of different parameters on the overall performance of the system. Wikipedia dumps for English, Kazakh and Serbo-Croatian are used to train models. We used a Python package called gensim to train the word2vec models. After thoroughly checking the gensim Python package, we found that using a single thread and setting the random seed are required to ensure the reproducibility of the results.
Some of the hyperparameters are already optimized by Mikolov et al. (2013) where they have recommended a set of values for the parameters.
We have trained the models with the settings shown in Table 2. The values for minimum count (the minimum count of word in corpus to be considered as a unique word), downsampling threshold (for downsampling high frequency words), initial learning rate and minimum learning rate are set to the values that were recommended by Mikolov et al. (2013). Regarding the number of epochs, it is set to only 5 as we found that increasing it results in worse models. This occurs due to the fact that word2vec models are based on neural networks with huge number of parameters and training them for a long time using the same data will cause over-fitting. Finally, we have investigated the effect of the window size and the vector size on the system's performance.
All the models are trained using wikipedia dumps (dumped

[1] `https://github.com/apertium/apertium-weighting-tools`

[2] We use the language name Serbo-Croatian to refer to the varieties spoken in Bosnia, Croatia, Montenegro and Serbia which may appear under a number of names. The languages are morphologically very similar and the morphological analyzer treats them as a single language system. Regional differences in phonology (e.g. the *yat* reflex as {*-e-*, *-je-*, *-ije-*}, morphology/orthography (spelling of the future tense) and lexicon are dealt with in the analyzer.

Table 3: Effect of vector size on skip-gram models (window size is 2)

	English			Serbo-Croatian			Kazakh		
Vector size	P	R	F	P	R	F	P	R	F
100 (la)	**0.710**	**0.713**	**0.712**	0.468	0.470	0.469	0.646	0.636	0.641
200 (la)	0.701	0.705	0.703	0.469	0.471	0.470	0.640	0.630	0.635
300 (la)	0.705	0.709	0.707	0.473	0.476	0.474	0.649	0.639	0.644
100 (sgt)	0.701	0.704	0.702	0.460	0.462	0.461	0.647	0.637	0.642
200 (sgt)	0.698	0.702	0.700	0.463	0.466	0.464	0.646	0.636	0.641
300 (sgt)	0.696	0.700	0.698	**0.476**	**0.480**	**0.478**	**0.652**	**0.642**	**0.647**

Table 4: Effect of window size on skip-gram models (vector size is 100 for English and 300 for Kazakh and Serbo-Croatian)

	English			Serbo-Croatian			Kazakh		
Window size	P	R	F	P	R	F	P	R	F
2 (la)	0.710	0.713	0.712	0.473	0.476	0.474	0.649	0.639	0.644
5 (la)	0.737	0.736	0.736	0.450	0.454	0.452	0.644	0.634	0.639
10 (la)	**0.740**	**0.739**	**0.739**	0.437	0.440	0.438	0.652	0.642	0.647
2 (sgt)	0.701	0.704	0.702	**0.476**	**0.480**	**0.478**	0.652	0.642	0.647
5 (sgt)	0.730	0.729	0.730	0.431	0.434	0.432	0.642	0.632	0.637
10 (sgt)	0.734	0.732	0.733	0.434	0.437	0.436	**0.654**	**0.645**	**0.650**

on the 22nd of February 2020). Since the size of the English dump is much larger than the size of the Kazakh and Serbo-Croatian dumps, a portion of the English dump was used to make the sizes of the corpora comparable.

6. Evaluation and Results

To evaluate the performance of a generated weightlist, first a weighted FST is generated. For each labeled token in the tagged corpus, the list of candidate morphological analyses are generated using the weighted FST. Finally, the most probable analysis (the one having the least weight) is considered as the model's prediction.

6.1. Evaluation corpora

Apertium[3] has tagged corpora for the supported languages. The evaluation process relied on Apertium's tagged corpora for English, Kazakh and Serbo-Croatian. These corpora are distributed under the GNU General Public License.

For the English corpus, the number of token/analysis pairs in corpus is 29,650. The number of unique tag combinations in the corpus is 198. The distribution for the number of candidate analyses for each token before disambiguation is shown in Figure 4.

Clearly, 65.8% of the corpus is actually not ambiguous. This is a result of the nature of the English Morphology.

On the other hand, Kazakh is a morphologically complex language. For the Kazakh corpus, the number of token/analysis pairs in corpus is 9762. The number of unique tag combinations in the corpus is 655. And the distribution for the number of candidate analyses for each token before disambiguation is shown in Figure 5. A single token in Kazakh might have more than 20 candidate analysis. More than 0.5% of the corpus has 16 or more analyses which makes the disambiguation process much harder. Additionally, the Kazakh corpus has 655 unique tag combinations compared to the 198 tags in the English corpus despite the fact that the Kazakh corpus size is about one third that of the English corpus.

Similarly, Serbo-Croatian has 20127 token/analysis pairs with 598 unique tag combinations. Figure 6 shows that more than 5% of the tokens in the tagged corpus had 16 or more candidate morphological analyses.

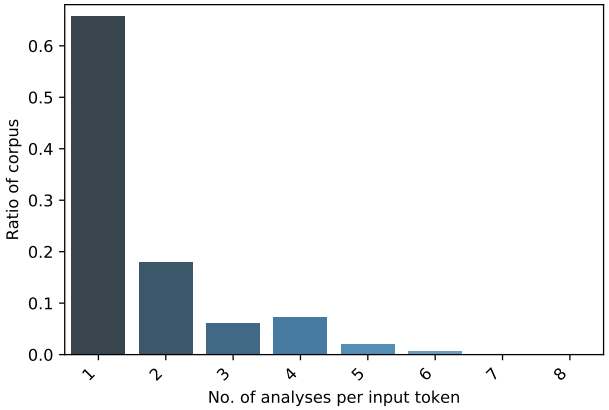

Figure 4: Ratio of tokens having 1, 2, .. number of candidate analyses in the English corpus

6.2. Results

To evaluate the weighted FSTs, precision, recall and F1 score are used. For each tag in the tagged corpus, precision is calculated as $P = \frac{TP}{TP+FP}$, recall is calculated as $R = \frac{TP}{TP+FN}$ and F1 score is calculated the harmonic mean of the precision and recall $F1 = \frac{2*(P*R)}{P+R}$ where TP

[3]Apertium is a machine translation program targeting lower resourced languages.

Table 5: Results of the reference and the word2vec methods. The unigram-count and constraint-grammar methods both rely on either annotated data or rule-based disambiguation.

	English			Serbo-Croatian			Kazakh		
Model name	P	R	F	P	R	F	P	R	F
unigram-counts (la smoothing)	0.848	0.844	0.846	0.830	0.841	0.835	0.849	0.841	0.845
unigram-counts (simple good-turing smoothing)	0.856	0.853	0.855	0.830	0.840	0.835	0.849	0.841	0.845
constraint grammar (la smoothing)	0.776	0.774	0.775	0.607	0.622	0.615	0.702	0.697	0.699
constraint grammar (simple good-turing smoothing)	0.771	0.768	0.769	0.557	0.572	0.564	0.641	0.637	0.639
random (la smoothing)	0.705	0.703	0.704	0.434	0.448	0.441	0.577	0.573	0.575
random (simple good-turing smoothing)	0.705	0.703	0.704	0.450	0.459	0.455	0.606	0.600	0.603
equal	0.687	0.687	0.687	0.554	0.558	0.556	0.647	0.633	0.640
analysis length	0.685	0.685	0.685	**0.542**	**0.549**	**0.545**	**0.670**	**0.662**	**0.666**
word2vec (la smoothing)	**0.740**	**0.739**	**0.739**	0.473	0.476	0.474	0.652	0.642	0.647
word2vec (sgt smoothing)	0.734	0.732	0.733	0.476	0.480	0.478	0.654	0.645	0.650
word2vec (la smoothing) (pretrain - WP dump 17)	0.708	0.707	0.708	-	-	-	-	-	-
word2vec (sgt smoothing) (pretrain - WP dump 17)	0.702	0.702	0.702	-	-	-	-	-	-
word2vec (la smoothing) (pretrain - CoNLL17)	0.720	0.720	0.720	-	-	-	-	-	-
word2vec (sgt smoothing) (pretrain - CoNLL17)	0.713	0.712	0.713	-	-	-	-	-	-
word2vec (la smoothing) (pretrain - CoNLL17)	-	-	-	-	-	-	0.642	0.632	0.637
word2vec (sgt smoothing) (pretrain - CoNLL17)	-	-	-	-	-	-	0.653	0.643	0.648

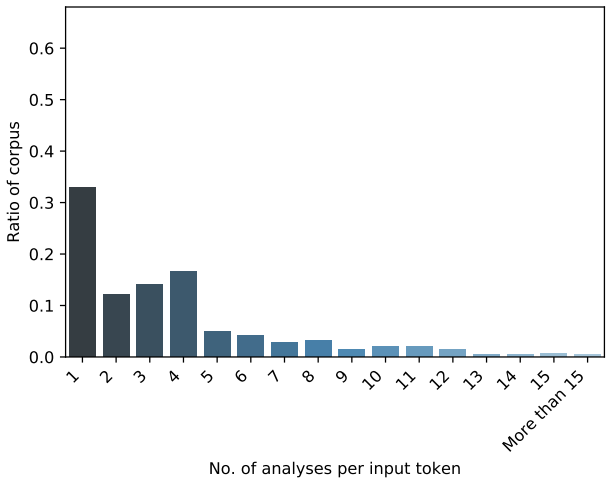

Figure 5: Ratio of tokens having 1, 2, .. number of candidate analyses in the Kazakh corpus

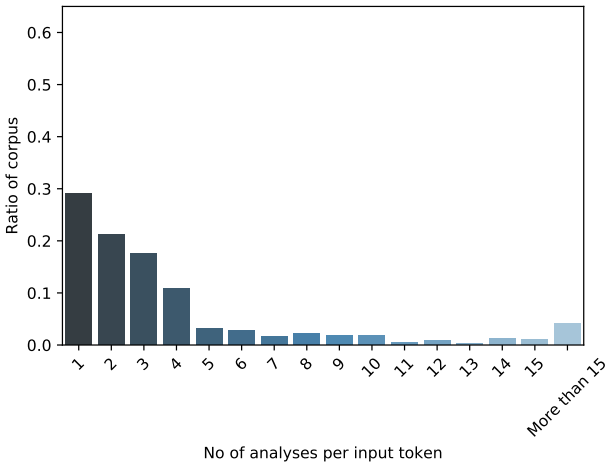

Figure 6: Ratio of tokens having 1, 2, .. number of candidate analyses in the Serbo-Croatian corpus

is the number of True positives, FP is the number of False positives and FN is the number of False negatives. For example, To calculate the precision, recall and F1 score for the analysis (classification class) `euro<n><sg>`, the correct labels and the predictions of the disambiguation model are divided into two bins: positive (the model's prediction is equal to the class in question `euro<n><sg>`) and negative (the model's prediction isn't equal to the class in question `euro<n><sg>`). The four combinations of the label and prediction are:

- True Positive (TP): The correct label is `euro<n><sg>` and the model's prediction is `euro<n><sg>`.
- False Positive (FP): The correct label isn't `euro<n><sg>` and the model's prediction is `euro<n><sg>`.
- False Negative (FN): The correct label is `euro<n><sg>` but the model prediction isn't `euro<n><sg>`.
- True Negative (TN): The correct label isn't `euro<n><sg>` and the model's prediction isn't `euro<n><sg>`.

After counting the values of TP / FP / FN for the current analysis `euro<n><sg>`, the precision, recall and F1 scores are computed as shown above. This operation is repeated for all the analyses (classification classes) in the evaluation data-set. The precision, recall and F1 metrics are finally merged using a weighted macro-average among all these tags.

Table 5 reports the evaluation metrics for the reference

methods in addition to our word2vec based method for the English, Kazakh and Serbo-Croatian languages.

6.2.1. Hyperparameter optimization

Results of the hyperparameter optimizations for the vector and window sizes are shown in Tables 3, 4. On optimizing the hyperparameters, both laplace smoothing (la) and simple good-turing (sgt) were used. First, three models are trained for each language with all the other parameters set as the values indicated in 2 and vector sizes are 100, 200, 300. It was found that the vector size of 100 suited English better while vector size of 300 was better for both Kazakh and Serbo-Croatian. For the window size, it was found that larger window sizes generated more precise models for English and Kazakh but weren't useful for Serbo-Croatian. For the counts smoothing method, Laplace smoothing worked better with the word2vec models for English while simple Good-Turing outperformed Laplace smoothing for Kazakh and Serbo-Croatian.

6.3. Discussion

We have noticed that the results of Skip-gram word2vec models is sensitive to values like the random seed used during the training of the model. We found that (Hellrich and Hahn, 2017) have made deeper investigation into the effect of the randomness of the word2vec models on the reliability of these models. They showed that the most similar words for an input word are highly affected by changing the random seed concluding that the results of these models aren't reliable enough and are sensitive to the seed.

Additionally, training word2vec models from scratch and tuning them seemed better than using the pretrained models even if these pretrained models used much larger corpora.

Word2vec models require the usage of relatively large untagged corpora. Fares et al. (2017) managed to train a word2vec model for languages with limited resources such as: Norwegian, Uyghur, Latvian and Irish. Despite the fact that these models might not be able to fully capture the semantic meanings of languages, they prove that training word embedding models for low-resourced languages is doable given that a moderately large untagged corpus is available which seems to be possible even for languages like Irish which has about 1,171,000 users according to Ethnologue.

Finally, it was found that all the reference methods performed better than the random method. This indicates that even having a simple heuristic for sorting the analyses is an improvement to the unweighted morphological analyzer.

7. Conclusion

The paper explores exploiting the word2vec model's (as an embedding model) ability to capture the semantic similarities between words for disambiguating the results of a morphological analyzer. This method proved to be useful for both morphological analysis and tagging tasks. Our novel word2vec-based method performed better than the basic unsupervised reference models. And despite the fact that it didn't improve over the supervised unigram method, it proved to be able to improve the analyses order and consequently the FST's accuracy without relying on manually tagged corpora. This method was very successful in disambiguating English analyses but its performance dropped for morphologically complex languages like Kazakh and Serbo-Croatian. We have also shown that directly weighting the analyses of words instead of relying on the context to disambiguate the results is a successful technique to deploy.

8. Bibliographical References

Beesley, K. R. (2003). Computational Morphology and Finite-state Methods. *NATO SCIENCE SERIES SUB SERIES III COMPUTER AND SYSTEMS SCIENCES*, 188:61–100.

Fares, M., Kutuzov, A., Oepen, S., and Velldal, E. (2017). Word Vectors, Reuse, and Replicability: Towards a Community Repository of Large-text Resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017, Gothenburg, Sweden*, number 131, pages 271–276. Linkping University Electronic Press, Linkpings universitet.

Gale, W. A. and Sampson, G. (1995). Good-Turing Frequency Estimation without Tears. *Journal of quantitative linguistics*, 2(3):217–237.

Hellrich, J. and Hahn, U. (2017). Don't Get Fooled by Word Embeddings - Better Watch their Neighborhood. In *DH*.

Karlsson, F. (1990). Constraint Grammar as a Framework for Parsing Running Text. In *COLNG 1990 Volume 3: Papers presented to the 13th International Conference on Computational Linguistics*.

Lindn, K., Pirinen, T., et al. (2009). Weighting Finite-state Morphological Analyzers using HFST Tools. In *Finite-State Methods and Natural Language Processing-FSMNLP 2009 Eighth International Workshop*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*.

Mohri, M. (2004). Weighted Finite-state Transducer Algorithms. An Overview. In *Formal Languages and Applications*, pages 551–563. Springer.

Sánchez-Martínez, F., Pérez-Ortiz, J. A., and Forcada, M. L. (2008). Using Target-language Information to Train Part-of-speech Taggers for Machine Translation. *Machine Translation*, 22(1-2):29–66, March.

Schiller, A. (2006). German Compound Analysis with wfsc. volume 4002, pages 239–246, 12.

Shen, Q., Clothiaux, D., Tagtow, E., Littell, P., and Dyer, C. (2016). The Role of Context in Neural Morphological Disambiguation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 181–191, Osaka, Japan, December. The COLING 2016 Organizing Committee.

Uí Dhonnchadha, E. and Van Genabith, J. (2006). A Part-of-speech Tagger for Irish Using Finite-state Morphology and Constraint Grammar Disambiguation. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May. European Language Resources Association (ELRA).

Washington, J., Salimzyanov, I., and Tyers, F. M. (2014). Finite-state Morphological Transducers for Three Kypchak Languages. In *LREC*.

Webb, G. (1993). Learning Decision Lists by Prepending Inferred Rules. In *In Proceedings of the Australian Workshop on Machine Learning and Hybrid Systems*, pages 6–10.

Yuret, D. and Türe, F. (2006). Learning Morphological Disambiguation Rules for Turkish. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 328–334, Stroudsburg, PA, USA. Association for Computational Linguistics.