Ricardo Lukas Jung
6227492
Empirische Sprachwissenschaft (B.A.)
Phonetik & Digital Humanities
15th Semester
s2458588@stud.uni-frankfurt.de

Bachelor Thesis

Lexicalizing a BERT Tokenizer

Building Open-End MLM for Morpho-Syntactically Similar Languages

Ricardo Lukas Jung

Date of Submission:
February 8, 2023

Text Technology Lab
Prof. Dr. Alexander Mehler
Dr. Zakharia Pourtskhvanidze

Erklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen Quellen oder Hilfsmittel als die in dieser Arbeit angegebenen verwendet habe.

_____

Ort, Datum

_____

Unterschrift

I thank my supervisors Prof. Dr. Alexander Mehler and Dr. Zakharia Pourtskhvanidze for their competence and professionalism, without whom this thesis would not have happened. My explicit thanks go to Manuel Stoeckel and Guiseppe Abrami for their valuable patience, advice and encouragement. Thanks to my close ones for your trust and support.

Abstract

This is the abstract: what is this about? what was done? what where the results?

Contents

List of Figures

List of Tables

List of Acronyms

BERT Bidirectional Encoders from Transformers

CL Computational Linguistics

GerParCor German Parliamentary Corpus

LM Language Model

LSTM Long Short-Term Memory

ML Machine Learning

MLM Masked Language Model

NLP Natural Language Processing

POS Part of Speech

HanTa Hanover Tagger

1 Introduction

2 Overview

3 Methodology

in this section the whole methodoloy is covered. what do i use in this thesis, why do i use it and lastly, how? make sure the why covers methodological implications. (vergiss nicht alle pakete als quelle im Anhang)

1 Requirements

A series of tools will help to achieve lexicalized tokenization. They will be explained in this chapter along with their methodological edge.

1.1 Pipeline Structure & Transformers Library

Bidirectional Encoders from Transformers (BERT) is a language learning transformer model designed for Natural Language Processing (NLP) tasks (Vaswani et al. 2017). Upon release it achieved higher performance scores compared to previously used Long Short-Term Memory (LSTM) models (Devlin et al. 2018). Two main model characteristics can be observed for BERT. Firstly, it is the first Language Model (LM) to implement simultaneous attention heads, allowing for bidirectional reading. The methodological implication of reading to the left and right of a token is to include more information about the language in single embeddings. Secondly, BERT introduced the (at the time novel) Masked Language Model (MLM) method for training. The method involves masking a specified amount (default 15%) of random tokens in the input sequence. Masked tokens are guessed by the model which can then update its weights according to success or failure.

The NLP community has since developed BERT and adapted it to the needs of contemporary NLP problems (roberta, germanbert, mbert CITATION). Its wide support, comparability and versatility make BERT the model of choice for this thesis. Another notable feature in BERT is the implementation of the WordPiece tokenizer module (QUELLE?). Default BERT WordPiece tokenization is predominantly heuristic by combining strings based on a precalculated score. A variety of pre-trained tokenizers are available, although they come with a caveat. Once a tokenizer is trained on a dataset it is specific to that dataset. This means the application of a tokenizer on another dataset may result in out-of-vocabulary issues and different token/subtoken distributions.

Particularly relevant to this thesis is the option to train an own tokenizer from the base module. Usually, WordPiece generates its own set of subtokens called *vocabulary*, which is used by the standard tokenizer to generate unique input numerical IDs. These identifiers correspond to tokens or parts of tokens found in the original dataset that the model was trained on. Wordmap partly takes over the tokenization depending on the the type of input it receives, the exact process is given in subsection 2.1. Once a string is tokenized it gets passed

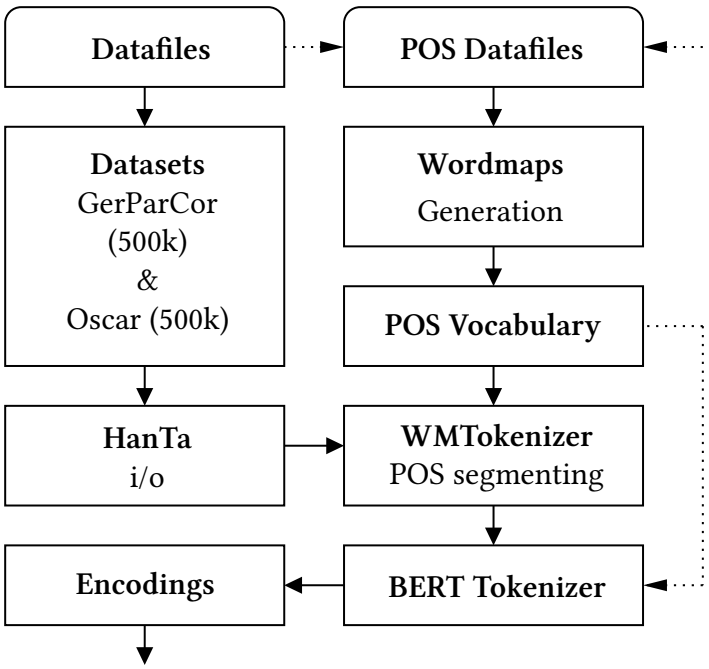

Figure 1: Schematic integration of Wordmap segmentation into the BERT architecture. Continuous lines show non-selective channels, while dotted lines only pass data selectively

on to the transformer model for contextualized embedding. Schematic Figure 1 shows the processing of data up until token encoding.

In Figure 1, Datafiles contains the raw unprocessed data coming from corpora. The input is provided in txt-files holding sentences line by line. Two datasets are generated from this input, each amounting to 500000 sentences per corpus for fine-tuning. POS Datafiles ideally contains only words that are identified as the same Part of Speech (POS) category. To increase the resulting POS vocabulary, POS Datafiles can be augmented by two channels: input can come from the corpus itself, but would need to be tagged first. Otherwise a predefined list of words can be used as an external and generic source for Wordmap generation. In POS Vocabulary, naive adjustments like removing outliers and interferences are performed on the set of subwords yielded by Wordmap generation. It is important to note that the POS vocabulary is separate from the the tokenizer vocabulary. This serves the specific purpose of keeping the vocabulary on which the model is trained clean from unwanted or unused subwords. Machine learning practice generally points to the trade-off between vocabulary size and model performance, hence the addition of POS Vocabulary to the BERT Tokenizer vocabulary is used only when necessary. Datasets are then passed through **hanta!** (**hanta!**) to provide WMTokenizer with flagged data to tokenize only selected tokens. Lastly BERT Tokenizer returns the canonical encodings known from language modeling.

1.2 Data

explain the data that is used wiktionary verbs hanta-crawled verbs of gerparcor oscar de gerparcor bundestag subset

1.3 Benchmark

The benchmark used on the model is oLMpics (Talmor et al. 2020)

2 Implementation

Tatsächliche Anwendung der Methoden auf die Daten

2.1 Tokenizer

ESSENTIALLY DERIVING SENSIBLE SUBTOKENS TO REPRESENT LEXEMES The methods of tokenization is explained in the following. This is the main part of the thesis containing two algorithms for vocabulary generation and token segmentation. Once the two algorithms have been explained, the last section of this chapter features the

Languages, as consistent form of communication, always Throughout this section, the words $target$ and $token$ are used to describe a word that is analyzed. One denotes the argument of the wordmapping and segmenter function ($target$), the other describes a word occurring in the corpus ($token$).

Generating a custom pre-training vocabulary

Embedding those subwords which take part in inflectional processes essentially means deriving sensible subwords to represent actual morphemes.

The vocabulary generated for this chapter is a list of inflected and non-inflected verbs (as to state the example) provided by two sources: (1) the german verb wiktionary **(**wiktionary) and (2) a crawl of the GerParCor corpus with the HanTa tagger **(**hanta). Initial experiments where done only with the wiktionary verb list since it provided sanitized input for the algorithm in development. Later, when the verbs where extracted German Parliamentary Corpus (GerParCor) from gerparcor

The Wordmap algorithm as shown in Algorithm 1 is the first step to extracting morphemes from a token. Its purpose is to compare two strings and store their intersections in a map of boolean values.

Wordmap requires two **inputs** $verbs$ and $target$. The resulting wordmap will be generated for $target$, while $verbs$ serves as comparison. Any map generated from this also has the same length as $target$. $verbs$ is a set of tokens pertaining to the same POS category. Note that $verbs$ should only contain those POS-tagged tokens that are expected to carry lexical information (e.g. verbs, adjectives, etc.). The set is previously extracted from the corpus by POS-tagging. Optionally, the set can be augmented by manually adding POS matching tokens from external sources. The 2-tuple $pair$ are the strings to be compared. It is passed on to (1) SHORTER, a function returning the shorter of both strings (2) LONGER, expectedly

Algorithm 1 Wordmap generation

Input: $verbs = \{v : POS\}, target$ ▷ *verbs*: set of single-POS lexemic tokens
Output: $maps = (map_1, \ldots, map_{|verbs|})$

function WORDMAP(w1, w2, d=0)
    $f_1 : c1, c2 \mapsto c1 == c2$
    $f_2 : \left(\sum_{i=0+d}^{|w1|} w1[i], w2[i] \mapsto f_1(w1[i], w2[i])\right)$
    return f_2
end function

for $v \in verbs$ **do**
    $pair = (target, v)$
    $s = \text{SHORTER}(pair)$
    $l = \text{LONGER}(pair)$
    $case = \text{MATCH\_ENDS}(pair)$ ▷ Returns if strings match in the last or first position
    $len = \text{LEN}(l)$
    $\delta = \Delta(len, \text{LEN}(s))$

    if *case: any match* **then**
        if δ **then**
            if *case: left match* **then**
                $\text{WORDMAP}(l, s)$
                Pad map from right side with 0s to match len
            end if
            if *case: right match* **then**
                $\text{WORDMAP}(l, s, \delta)$
                Pad map from left side with 0s to match len
            end if
        else
            $\text{WORDMAP}(l, s)$
        end if
    end if
end for

Table 1: Example wordmaps for $target =$ `verstehen`

String	Wordmap	Case	Padding
`verarbeiten`	111000000	left match	Yes
	001000011	right match	Yes
`variiert`	101001000	left match	Yes
`vormachen`	101000111	left match	No
	101000111	right match	No
`anstrebtest`		no match	
(...)			

returning the longer of both strings (3) MATCH_CASE, a function to determine the behavior of the algorithm later on. As two strings are compared MATCH_CASE captures three cases: $pair$ matches in the first, last or both positions. Finally len denotes the length of the longest string and δ difference in length between s and l. δ functions as an offset for index-based comparisons WORDMAP.

WORDMAP is a naive mapping function generating the wordmaps.

Once every v has been compared to $target$, $maps$ stores boolean counts of characters occurring in their respective positions in $target$. Every map is cleaned with a regular expression to reduce noise caused by natural character occurrence (some characters like <n>, <s> will be more frequent than others). Continuous concatenations of leading or trailing matches stay, while every match enclosed by 0 will be replaced the 0. As an example, $wordmap = 11101100101$ contains three matches on the inside which will result in 11100000001 as final output. In the penultimate stage of mapping $target$, all maps are summed up to receive the number of absolute positional occurrences of every character in $target$. This positional mapping allows for detecting relevant segments in a token based on a threshold. Characters in range of the predefined threshold are selected for the mapping of a target token. Functional morphemes (morphemes that are carriers of grammatical features) are typically much more frequent than their lexical counterparts. Consequently, lexical morphemes in the family of inflectional languages are - by definition - modified by functional morphemes, they occur much less frequently. In this case, the activation function for a concatenation of same boolean values to be selected as segment is the normalizing z-score function defined as: $z_i = \frac{x_i - \overline{x}}{S}$, where z is the z-score, $\overline{x}$ and S are the sample mean and the sample standard deviation.

BERT Tokenizer Modification

Algorithm 2 recursively computes every possible segmentation for a string $target$ from a given vocabulary pos_vocab from left to right. The vocabulary contains all the segments that were identified in the previous step by Algorithm 1. Every segmentation has to be complete so that its segments corresponds to non overlapping substrings of $target$. For every $target$ a subvocabulary $morpheme$ is defined, containing all strings that are in the vocabulary of **POS!** (**POS!**) members. This task is a weighted coverage problem in the NP-hard domain.

Algorithm 2 Target Segmentation

Input: $target, pos\_vocab$ ▷ short comment

Output: $\{(tuples\ of\ subwords)\} \approx \{t \in \mathcal{P}(s \in pos\_vocab : s \in target) : t \equiv target\}$

```
1:
2: segmentations = ()
3:
4: function SEGMENTER(token, stop, start=0, segments)
5:     if start = stop then
6:         Add segments to segmentations
7:     else
8:         morphemes = (m ∈ pos_vocab : target.STARTSWITH(m) ∧ |m| > 1)
9:         for m ∈ morphemes do
10:            start += LEN(m)
11:            Add m to segments
12:            rest = target[LEN(m) :]
13:            if LEN(rest) == 1 then
14:                start = stop
15:                Add rest to segments
16:                Add segments to segmentations
17:                Decrement start, crop segments
18:            else
19:                SEGMENTER(target = rest, stop, start = start, segments)
20:                Decrement start, crop segments
21:            end if
22:        end for
23:    end if
24: end function
25: MAXIMIZE_SEGMENTS(segmentations)
```

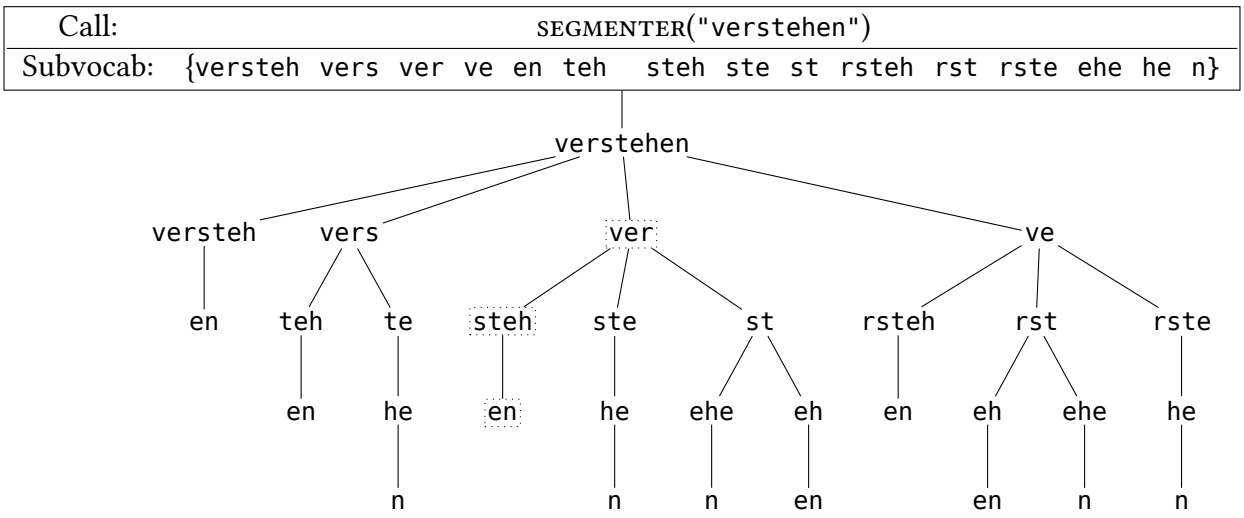

Figure 2: Permutations SEGMENTER generates from target i.e. `verstehen`. Dotted segments mark the segmentation chosen later by the maximization function during tokenization.

Unary morphemes to the left are excluded from the pre-selected subvocabulary to drastically reduce the number of possible permutations , as they can be embedded in n-ary tokens as well. The recursion can be seen as n-ary a trees containing every permutation of the set $morphemes$ where the sum of branches all satisfy $target$.

As shown in Figure 2, all morphemes that $target$ starts with are stored to form the first nodes of the permutation tree. Each time a morpheme is selected the index $start$ is incremented by the length of the morpheme to indicate when the string has been completely segmented. The new recursion is called with the updated index $start$ and $target$ sliced by the length of the morpheme contained in the parent node. Incomplete segmentations that miss exactly one character to the right are accepted with the added missing string. If the vocabulary cannot satisfy a segmentation because it is missing the necessary strings, segmentation is omitted and the original input token is returned as such.

Then, of all $segmentations$ a single segmentation is selected by a maximization function MAXIMIZE_SEGMENTS calculating weights for every segment. The maximization function for one segment is the defined as:

$$\arg\max_{s \in S} f(s) \coloneqq \left\{ s \in S : \sum_{i=1}^{|s|} \sqrt[s_i]{\frac{s_i}{t} \div |s|} \right\} \tag{3.1}$$

Where S is a set of segmentation tuples s, $|s|$ is the length of the tuple (read: number of segments) and s_i indicates the length of the segment at position i in tuple s. For every segment (vertex) in a segmentation tuple (branch) the segment's length is divided by the token length t to get the coverage the morpheme provides towards the target string. The number of segments $|s|$ is a divisor meant to cap the number of segments in a segmentation. It pre-

vents choosing a segmentation overflowing with too many short segments. Short segments are convenient for completing a segmentation, but will increase the chance of slicing a token where it linguistically lacks sense to do so. Lastly the $s_i{}^{\text{th}}$ root implements a bias against segments that are too long. While the accuracy decreases for longer words, this method of maximization performs reasonably well in the range of 1–4 syllable tokens, which make up around 87% of the verb vocabulary.

tweak des tokenizers: segmentation ist eine frage der interpretation.

Ultimately, segmentation is a matter of interpretation. If there are several possible interpretations by wich to segment a word this tokenization method relies on the assumption that the comparison with POS- members displays the probable shape of segmentation for a token. As mentioned in 1.1, the default WordPiece Tokenizer lacks A linguistically informed

The field of NLP (Glück and Rödel 2016) has been expanded ever since the emergence of the language models. Natural language processing is understood as the

2.2 Masked Language Model

Model implementation and parameters, runtimes?

2.3 oLMpics Benchmark

4 Results

5 Discussion

6 Conclusion

7 Testchapter

1 Citing

Abrami et al. 2022

2 Quoting

> “This is a quote by textquote” (DeepL 2021) “This is a quote by enquote”

3 Referencing

Short reference 1.1
Long reference subsection 1.1
monofont for code or string `monofont`

Bibliography

Abrami, Giuseppe, Mevlüt Bagci, Leon Hammerla, and Alexander Mehler (June 2022). "German Parliamentary Corpus (GerParCor)". In: *Proceedings of the Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, pp. 1900–1906. URL: https://aclanthology.org/2022.lrec-1.202.

Aikhenvald, Alexandra and R Dixon (2017). *The Cambridge Handbook of Linguistic Typology*. Cambridge Handbooks in Language and Linguistics. Cambridge: Cambridge University Press. DOI: 10.1017/9781316135716.

Baerman, Matthew, Dunstan Brown, and Greville G. Corbett, eds. (2017). *Morphological Complexity*. Cambridge studies in Linguistics 153. Cambridge University Press. ISBN: 1107120640. DOI: 10.1017/9781316343074. URL: www.cambridge.org/9781107120648.

Colman, Andrew M. (2009). *morpheme*. DOI: 10.1093/acref/9780199534067.013.5219. URL: https://www.oxfordreference.com/view/10.1093/acref/9780199534067.001.0001/acref-9780199534067-e-5219.

DeepL (Nov. 2021). *How does deepl work?* URL: https://www.deepl.com/en/blog/how-does-deepl-work. Last accessed: 28.12.2022.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

Glück, Helmut and Michael Rödel, eds. (2016). *Metzler Lexikon Sprache*. ger. 5th ed. Springer eBook Collection. Stuttgart: J.B. Metzler, pp. 141–142. ISBN: 978-3-476-05486-9. DOI: 10.1007/978-3-476-05486-9. URL: http://dx.doi.org/10.1007/978-3-476-05486-9.

Hofmann, Valentin, Hinrich Schuetze, and Janet Pierrehumbert (May 2022). "An Embarrassingly Simple Method to Mitigate Undesirable Properties of Pretrained Language Model Tokenizers". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Dublin, Ireland: Association for Computational Linguistics, pp. 385–393. DOI: 10.18653/v1/2022.acl-short.43. URL: https://aclanthology.org/2022.acl-short.43.

Peters, Ben and Andre F. T. Martins (July 2022). "Beyond Characters: Subword-level Morpheme Segmentation". In: *Proceedings of the 19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Seattle, Washington: Association for Computational Linguistics, pp. 131–138. DOI: 10.18653/v1/2022.sigmorphon-1.14. URL: https://aclanthology.org/2022.sigmorphon-1.14.

Schuster, Mike and Kaisuke Nakajima (2012). "Japanese and Korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

Talmor, Alon, Yanai Elazar, Yoav Goldberg, and Jonathan Berant (Dec. 2020). "oLMpics-On What Language Model Pre-training Captures". In: *Transactions of the Association for Computational Linguistics* 8, pp. 743–758. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00342. eprint:

https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00342/1923716/tacl_a_00342.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00342.

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

Wu, Yonghui et al. (2016). *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. DOI: 10.48550/ARXIV.1609.08144. URL: https://arxiv.org/abs/1609.08144.