Does the Choice of a Segmentation Algorithm Affect the Performance of Text Classifiers?

Rastislav Hronsky[1] and Emmanuel Keuleers[2]

[1] Jheronimus Academy of Data Science
Sint Janssingel 92, 5211 DA 's-Hertogenbosch
r.hronsky@tilburguniversity.edu
[2] Department of Cognitive Science and Artificial Intelligence, Tilburg University
Warandelaan 2, 5037 AB Tilburg
e.a.keuleers@tilburguniversity.edu

Abstract. Text segmentation, a pre-processing step present in most NLP pipelines, is a rich topic that tends to get little attention in most NLP studies. We provide a brief empirical account on whether the choice of a particular segmentation algorithm (BPE, SentencePiece, WordPiece, Unigram, Morfessor) has an effect on the performance on downstream tasks, without transfer learning: natural language inference (SNLI), and sentiment classification (SST2). In contrast to previous studies which have compared BPE to Unigram segmentations on 4 downstream tasks, or language models built with BPE and morphologically aligned segmentations, this paper provides a comparison between 5 common segmentation algorithms and runs experiments with the algorithms fitted on 20 random subsets of the training corpus. In all experiments, we use a simple LSTM-based artificial neural network (in a Siamese setup for the SNLI) in either binary, or three-way classification set-up. The resulting accuracy data across the segmentation algorithms do not conclusively favor any particular one of them, in contrast to what the related literature suggests.

Keywords: NLP · Segmentation · Text classification

1 Introduction

A variety of text segmentation algorithms (often referred to as tokenization) has emerged, more or less, as a by-product of the large interest in solving increasingly difficult NLP problems. Natural language text segmentation can be approached in widely varying degrees of sophistication, ranging from segmentation by whitespace, through algorithms based on byte-pair encoding [7] (BPE), to language modeling [11]. However, the extent to which the choice of a particular segmentation algorithm affects the system's performance on downstream tasks is rarely examined [1].

Arguably the most significant advances in NLP have been due to improvements in the architecture of neural networks (e.g., the LSTM architecture [8],

layer stacking and bi-directional networks [15], the attention mechanism [26], transformer [24], etc.) and increase in dataset sizes coupled with better optimization procedures and hardware, rendering the possible effects of segmentation marginal at most. A further reason for the reluctance to inspect this effect might be the high prevalence of transfer learning [15] (using pre-trained models on other tasks and datasets prior to experimentation) in NLP experiments, where the dimension of the inputs, and thus the tokenizer, is the same for all further fine-tuning tasks.

Perhaps more motivation to study segmentation algorithms can come from a researcher viewing the problem in a more fundamental way: not merely an annoyance in the pre-processing pipeline, but as an investigation of the principles by which meaningful, self-contained chunks of text or other types of sequential data, can be recognised. Most segmentation algorithms work in unsupervised fashion, which, apart from suggesting that they do exploit some low-level principles, means that they are difficult to evaluate, since defining a gold standard would require to actually know these principles. A potential difference in performance in a simple downstream task, across different segmentation algorithms, would provide an extrinsic, indirect method to evaluate and rank segmentations and algorithms. In order to explore the possibility of cheaply evaluating segmentation algorithms in this way, we employ an experimental set-up that uses a small-scale simple LSTM without any pre-training or transfer learning

The focus of the present work is not achieving performance comparable to the SOTA. Instead, we provide an empirical analysis oriented towards answering the following research question:

- Is there a marked difference in the performance of a small-scale LSTM-based classifier on text classification as a result of using different segmentation algorithms?

We ran multiple experiments across five popular segmentation algorithms trained on random subsets of the training corpora: the Stanford Natural Language Inference [2] (SNLI), and the Stanford Sentiment Treebank [22] (SST2) corpus [3]. The experiments were ran with an LSTM-based predictor fed with inputs resulting from segmentation algorithms (with a fixed target vocabulary size of 10000, a hyper-parameter that all of the segmentation algorithms share), all of which were variably fitted 20 times on a random subset of the task-corresponding corpus. The experiments were evaluated by comparing the best-achieved prediction accuracy on the validation split (provided by the original dataset) across the random subsets and segmentation algorithms via a one-way ANOVA.

While models trained with segmentations of all the aforementioned algorithms predicted the targets with above baseline accuracy, there was no a marked difference between the segmentation algorithms themselves. However, there was strong variance in performance across the randomized runs.

[3] The current SOTA accuracy at these tasks is 92.1% [16] and 97.5% [10], respectively.

2 Related Work

Prior to the vast interest of NLP researchers, text segmentation has been mainly researched from a linguistic perspective, e.g., analyzing distributional cues that language learners might use to segment speech [19], or designing algorithms to find morphological segmentation from raw text data [4, 5, 25].

Strong demand for precise machine translation in recent years has lead to the adoption of BPE [21] as a robust solution to text segmentation, which alleviates the problem of out-of-vocabulary words in an open domain setting. It also improves the translation of rare words which can often be broken down into more frequent sub-words, thereby greatly reducing the required vocabulary size of models and leading to more efficient training. Originally being developed as a greedy compression algorithm [7], it has the benefits of fast runtime and being unsupervised. Nowadays, BPE is used in most NLP systems: either in its original form in the GPT-models [17], or some of its derivatives, such as WordPiece [20] in BERT [6] or SentencePiece [12] in ALBERT [13], XLNet [28].

While there is no shortage of segmentation algorithms for NLP, it is rare to see them in direct comparison. The Unigram algorithm was developed as a method capable of providing alternative segmentations based on probability, which can be usefully employed as a regularization technique for, e.g., NMT. A study [11] has compared BLUE scores achieved by training an NMT system with segmentations as a result of the aforementioned subword regularization, the Unigram segmentation without subword regularization, BPE, and segmentation into words and characters. The study reported an approximately $\frac{1}{2}$ point increase in BLUE score by using the subword regularization technique over only using BPE, while BPE achieved a comparable score to the Unigram segmentation without regularization. The word and character models performed the worst.

Recently it has beed argued [1] that BPE is suboptimal for language model pre-training based on experiments on three English language tasks (SQuAD 1.1 [18], MNLI [27], and CoNLL NER [23]) and 1 Japanese language task (TyDi QA [3]). The authors pre-trained and fine-tuned two different BERT language models for each of the languages with segmentations from either BPE or the Unigram model and found subtle improvements in range of up to 1% in classification accuracy or F1-score in favor of the models pre-trained with Unigram segmentations on the English tasks. The TyDi QA task with Japanese language reportedly achieved an increase from 42.1 to 54.4 in F1-score by using BERT with Unigram segmented pre-training; the authors mainly explain the large gap by the small amount of data available in the Japanese portion of TyDi compared to SQuAD (5k vs. 88k examples). The scale of such an experimental set-up is orders of magnitude larger than the current work because (1) it employs the full process of language model pre-training and fine-tuning, and (2) it uses a full-sized transformer model (BERT) requiring a lot of computational resources for training.

In another study, the authors examined to what extent can difference in difficulty in language modeling be explained by morphological complexity of a language by comparing the perplexity of language models trained with BPE seg-

mentations and segmentations more closely aligned with language morphology produced by Morfessor [14]. They found that the latter class of segmentations lead to less negative impact on language model perplexity in languages with features indicating higher morphological complexity. A further study puts the WordPiece algorithm to a test against morphologically informed segmentations by devising a semantic probing task, where the authors compare the accuracy in predicting one of two possible semantic classes using representations of constructions resulting from either of the segmentation algorithms and processed by the BERT language model [9]. They constructed a dataset by identifying derivationally complex words which are dominantly prevalent in, for example, scientific articles about physics rather than computer science, or vice versa, and found that the representations produced by morphologically aligned segmentations predicted the classes $2-4\%$ more accurately across tasks.

3 Segmentation Algorithms

In this section we briefly introduce the segmentation algorithms in our experiments. All of these algorithms are unsupervised.

Morfessor The Morfessor is a family of unsupervised algorithms designed to produce segmentations that correlate with morpheme boundaries. It is implemented in a general way, such that it can be used for any language, or even other type of sequential data. The smallest units in Morfessor are *atoms* which occur in sequences that are called *compounds*. The output of the algorithm are *constructions*, which are sub-sequences of *compounds*. While various extensions to Morfessor have been proposed, the *baseline* version is based on a maximum a posteriori estimation of parameters θ (consisting of a lexicon and a grammar) given a training dataset D_W. The first term of the resulting loss function (1), the negative logarithm of the probability of the model parameters, is implemented such that simpler models (i.e., smaller vocabulary) are preferred over more complex models. The second term evaluates the likelihood of the data as a product of the probabilities of the individual segments, since they are assumed to occur independently. In practice, the algorithm iterates over *compounds* in the training data and calculates the decrease in cost for every possible split of a particular compound, it chooses the cheapest option and continues recursively.

$$L(\theta, D_W) = -log(p(\theta)) - log(p(D_W|\theta)) \tag{1}$$

Unigram Language Model The general idea of the Unigram language model algorithm is very similar to the Morfessor, but it is implemented differently. In the first step, the algorithm heuristically initializes a large vocabulary, typically by including all individual characters and sets of most frequent n-grams. Then, for each word in the vocabulary, it estimates how the likelihood of the data and thus the loss function in Formula 2 would change under the new parameters (i.e.

the vocabulary without this word.) Finally, the words are sorted according to their estimated differential loss, and the top portion of the list, given a desired vocabulary size, is retained as final vocabulary while the rest is discarded. As a rule, individual characters are never discarded from the vocabulary to avoid out-of-vocabulary situations. While the Morfessor includes an extra term in the loss function, penalizing for model complexity, one could argue that, if implemented as penalty for large vocabulary size, such a preference is implicitly included in Formula 2: a smaller vocabulary means a smaller event space, leading to larger probabilities estimated for individual events and overall higher likelihood.

$$L(\theta, D_W) = -log(p(D_W|\theta)) \tag{2}$$

Byte-Pair Encoding BPE is a greedy compression algorithm that works from the bottom up by first initializing the vocabulary with all of the individual characters in the input. It then proceeds in iterations, constructing all possible bigrams (pairs) in the input sequence, calculating their frequencies, and finally replacing all the instances of the most frequent pair with a newly introduced symbol, which is added to the vocabulary. The algorithm proceeds until a desired vocabulary size is achieved, or until there is no bigram occurring more than once.

WordPiece Similarly to BPE, WordPiece constructs a vocabulary from the bottom up by merging pairs of bytes or characters. As a criterion for choosing which pair to substitute, it looks for the pair with the highest ratio between the joint probability of the two elements and the product of their individual probabilities. This is a quantity proportional to the exponential of the pointwise mutual information between the two symbols (for comparison, in BPE, the bigram count represents the estimate of the joint probability of the pair's occurrence).

SentencePiece SentencePiece was designed to be able to easily train language models for languages that do not use whitespace for word separation. SentencePiece has been designed by taking into account various considerations that are mainly relevant in production, large-scale scenarios (see the original paper for more details [12]).

4 Experiments

4.1 Downstream Tasks

We conducted experiments on two English language tasks: natural language inference and sentiment classification. Both of these tasks are classification tasks (binary and ternary, respectively) and were evaluated by the best achieved accuracy on a development set during training, over several runs of the segmenter-training (see Section 4.2). All of the experiments used a neural network consisting of an embedding layer and a single LSTM [8] layer, in a one-directional setup

with no dropout. The outputs of the network at each time-step were processed via max-pooling, and finally fed in to a single-layer feed-forward classifier. See Table 1 for an overview of the hyper-parameters. The number of training epochs was set in such manner that the performance on test-set started to decrease or converged. We used PyTorch [4] for implementation.

The Stanford Natural Language Inference (SNLI) corpus consists of 570k training, 10k validation, and 10k test sentence pairs labeled as *entailment*, *contradiction*, or *neutral*. The dataset is balanced, thereby being classified accurately about 33% of the time by random guessing. Both of the sentences in a pair are encoded into a vector representation by a shared encoder module, and the two resulting vectors are then concatenated and fed through a classifier to produce a prediction. In this Siamese-style model, the sequence encoder is implemented via an LSTM, and the classification is made via a fully-connected layer with three softmax output-nodes.

The Stanford Sentiment Treebank (SST2) dataset consists of 67k training, and 872 validation examples labeled in a binary way either having a *positive* or *negative* sentiment. The dataset is balanced, implying an expected random guessing accuracy of 50%. The predictor for this task has a simple design: the sentence is fed in to an LSTM-based encoder, on top of which there is fully-connected layer with one output node with a sigmoid activation, producing a binary prediction.

Table 1. Hyperparameters used for training. The model for SST2 was smaller because of the smaller corpus.

Task	**Embedding Size**	**Hidden Size**	**Batch Size**	**Learning Rate**	**Epochs**
SNLI	128	64	1024	0.003	10
SST2	32	16	128	0.001	15

4.2 Segmentation

In order to robustly test for the differences in the segmentation algorithms, we trained each of them 20 times on different portions of the training corpus. There portions were later separately used to train the predictors. In case of the (larger) SNLI corpus, we worked with subsets sized 20% of the original corpus. In case of the SST2, the subsets had a size of 75% of the original corpus. The vocabulary size parameter, where available, was set to 10000. We also examined two additional segmentations: *segmentation by words*, and a primitive segmentation creating a segment boundary for every three characters of the full sentence. The

[4] https://pytorch.org/

latter algorithm, which we will refer to as *blabla* segmentation, provides a sanity check. The algorithms under examination were trained by fitting the sentences already split in to words. All of the text was pre-processed such that there were only lower-case letters, and all non-alphabetical characters, including digits and punctuation, were set apart from their surroundings by a whitespace, such that splitting the resulting string by whitespace results in segmentation by words. (See Table 2 for example segmentations.) In case of Morfessor, we used the Python interface to the Morfessor 2.0 baseline model [25]. For the remainder of the algorithms, we used the Huggingface implementations [5].

Table 2. Example segmentations of one sentence from the SST2.

BPE	just-as-the-lousy-tarantino-im-it-ations-have-sub-sided
SentencePiece	just-as-the-lousy-tarantino-im-itations-have-sub-s-ided
WordPiece	just-as-the-lousy-tarantino-imitation-s-have-sub-sided
Unigram	just-a-s-the-lous-y-tarant-in-o-imitation-s-have-sub-sided
Morfessor	just-as-the-lousy-tarantino-imitation-s-have-subsided

4.3 Results

A one-way ANOVA showed that there was no difference in classification accuracy between the segmentation algorithms for the SST2, $F(4, 15) = 0.432, P = 0.785$, or the SNLI, $F(4, 15) = 2.564, P = 0.043$. An overview of the results of the two experiments is contained in Table 3.

SST2 The highest accuracy was achieved by a model trained on the vanilla BPE segmentation (81.60%). The lowest achieved accuracy was with Morfessor, (80.47%). The baseline segmentation by words performed about $3 - 4\%$ worse, and the *blabla* segmentation performed more than 10% worse than all of the other segmentation algorithms. Measured over the random subsets for segmentation training, the standard deviation of the model accuracy varied in the range of $1.44 - 2.35\%$..

SNLI The highest accuracy was achieved by a model trained on a SentencePiece segmentation (72.11%). The lowest accuracy was with the Unigram segmentation model, (71.09%). The baseline segmentation by words performed on par with the segmentation algorithms at 71.36% accuracy, and the *blabla* segmentation performed about $9 - 10\%$ worse than the other segmentation algorithms. Over the random subsets, the standard deviation of the model accuracy varied in the range of $0.83 - 1.29\%$.

[5] https://huggingface.co/docs/tokenizers/api/trainers

Table 3. Overview of the results. Each result corresponds to aggregated results over 20 runs per segmentation algorithm and task.

	SST2 (dev.)			SNLI(dev.)		
	[len]	% acc. ($\mu \pm \sigma$)	% acc. (max)	[len]	% acc. ($\mu \pm \sigma$)	% acc. (max)
BPE	10.88	77.97±1.98	**81.60**	11.43	70.06±1.29	72.00
Unigram	14.13	78.03±1.45	81.58	13.66	70.04±0.83	71.09
SentencePiece	11.08	78.16±2.02	81.34	11.48	69.59±1.28	**72.11**
WordPiece	10.89	77.44±2.35	80.67	11.43	69.65±1.16	71.42
Morfessor	10.29	78.00±1.44	80.47	11.40	70.60±1.04	71.93
Words	10.16		77.25	11.23		71.36
BlaBla	15.03		68.82	14.58		62.55

5 Discussion

In this study, none of the tested segmentation algorithms produce superior segmentations.

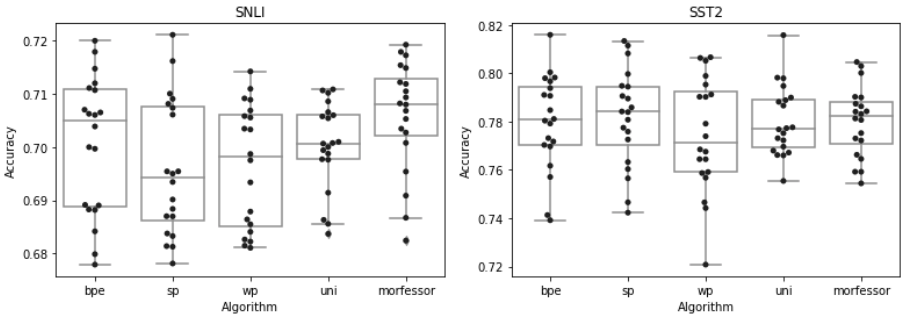

Fig. 1. Boxplots depicting the distribution of the best achieved accuracy across the two tasks and segmentation algorithms.

Interestingly, however, the results show that segmentation, as such, does matter. Figure 5, containing a boxplot for each of the tasks and distributions of the accuracy data over the randomized runs, depicts how the performance varies within groups: in ranges up to 4%, and 8% in case of SNLI, and SST2, respectively, which is a vastly larger range than that of the best achieved accuracy. In comparison to the figures for standard deviation per configuration in the results of [1] (about 0.2-0.3%), the numbers are significantly higher: $\sim 2\%, \sim 1\%$ for SST2 and SNLI, respectively. However, it is important to stress that the variation in [1] is a result of 5 runs with different random seeds for the initialization of the model parameters, while the source of variation in our work is the repeated creation of random subsets of the corpus for the training of the segmentation al-

gorithms, resulting in variability in model inputs. In order to pinpoint the cause of this high variation, a further investigation of the segmentations is required.

In contrast to [1, 14], our results do not suggest the superiority of either the Unigram model or the Morfessor. A first reason for this is that the benefits in performance gained by using morphologically aligned segmentations get more pronounced when training the segmentation on a large corpus, like in [1], such that the compound and rare words are better learned and represented. A second possible reason for the difference not being manifested via training a small-scaled classifier can be simply due to lack of representative capacity of the model, which a larger, multi-layer LSTM or a BERT model of [14, 1], respectively, does provide. This would be especially important for discovering subtle improvements provided by a particular segmentation of higher quality. Both of these factors limit us to conclude that there is never a difference in segmentation quality between the algorithms investigated. The results of the present study speak to the mere difficulty of cheaply evaluating segmentation algorithms. In light of the related work, one can conclude that, the shortest path to witnessing a difference in segmentation quality is most likely measuring the perplexity of a language model of reasonable expressive power, as demonstrated in [14].

6 Conclusion

Based on the presented evidence, we cannot conclude that there is any particular ranking of the quality of the segmentations produced by the tested algorithms when probed via a simple text classifier. Still, segmentation itself may cause significant difference in a downstream task, the reasons of which have yet to be determined.

References

1. Bostrom, K., Durrett, G.: Byte pair encoding is suboptimal for language model pretraining. arXiv preprint arXiv:2004.03720 (2020)
2. Bowman, S.R., Angeli, G., Potts, C., Manning, C.D.: A large annotated corpus for learning natural language inference. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics (2015)
3. Clark, J.H., Choi, E., Collins, M., Garrette, D., Kwiatkowski, T., Nikolaev, V., Palomaki, J.: TyDi QA: A benchmark for information-seeking question answering in typologically diverse languages. Transactions of the Association for Computational Linguistics **8**, 454–470 (2020). https://doi.org/10.1162/tacl$_{a0}$0317, $https://aclanthology.org/2020.tacl-1.30$
4. Creutz, M., Lagus, K.: Unsupervised discovery of morphemes. In: Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning - Volume 6. p. 21–30. MPL '02, Association for Computational Linguistics, USA (2002). https://doi.org/10.3115/1118647.1118650, https://doi.org/10.3115/1118647.1118650
5. Creutz, M., Lagus, K.: Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor 1.0. Publications in computer and information science. Report A, Helsinki University of Technology, Finland (2005)
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. ArXiv **abs/1810.04805** (2019)
7. Gage, P.: A new algorithm for data compression. C Users Journal **12**(2), 23–38 (1994)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (nov 1997). https://doi.org/10.1162/neco.1997.9.8.1735, https://doi.org/10.1162/neco.1997.9.8.1735
9. Hofmann, V., Pierrehumbert, J.B., Schütze, H.: Superbizarre is not superb: Derivational morphology improves bert's interpretation of complex words. arXiv preprint arXiv:2101.00403 (2021)
10. Jiang, H., He, P., Chen, W., Liu, X., Gao, J., Zhao, T.: SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 2177–2190. Association for Computational Linguistics, Online (Jul 2020). https://doi.org/10.18653/v1/2020.acl-main.197, https://aclanthology.org/2020.acl-main.197
11. Kudo, T.: Subword regularization: Improving neural network translation models with multiple subword candidates. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 66–75. Association for Computational Linguistics, Melbourne, Australia (Jul 2018). https://doi.org/10.18653/v1/P18-1007, https://aclanthology.org/P18-1007
12. Kudo, T., Richardson, J.: SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 66–71. Association for Computational Linguistics, Brussels, Belgium (Nov 2018). https://doi.org/10.18653/v1/D18-2012, https://aclanthology.org/D18-2012
13. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R.: Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942 (2019)
14. Park, H.H., Zhang, K.J., Haley, C., Steimel, K., Liu, H., Schwartz, L.: Morphology matters: A multilingual language modeling analysis. Transactions of the Association for Computational Linguistics **9**, 261–276 (2021)

15. Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). pp. 2227–2237. Association for Computational Linguistics, New Orleans, Louisiana (Jun 2018). https://doi.org/10.18653/v1/N18-1202, https://aclanthology.org/N18-1202
16. Pilault, J., Elhattami, A., Pal, C.: Conditionally adaptive multi-task learning: Improving transfer learning in nlp using fewer parameters & less data. arXiv preprint arXiv:2009.09139 (2020)
17. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)
18. Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.: Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250 (2016)
19. Saffran, J.R., Newport, E.L., Aslin, R.N.: Word segmentation: The role of distributional cues. Journal of memory and language **35**(4), 606–621 (1996)
20. Schuster, M., Nakajima, K.: Japanese and korean voice search. In: 2012 IEEE international conference on acoustics, speech and signal processing (ICASSP). pp. 5149–5152. IEEE (2012)
21. Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909 (2015)
22. Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 conference on empirical methods in natural language processing. pp. 1631–1642 (2013)
23. Tjong Kim Sang, E.F., De Meulder, F.: Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003. pp. 142–147 (2003), https://aclanthology.org/W03-0419
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
25. Virpioja, S., Smit, P., Grönroos, S.A., Kurimo, M.: Morfessor 2.0: Python implementation and extensions for morfessor baseline (2013)
26. Wang, Y., Huang, M., Zhu, X., Zhao, L.: Attention-based lstm for aspect-level sentiment classification. In: Proceedings of the 2016 conference on empirical methods in natural language processing. pp. 606–615 (2016)
27. Williams, A., Nangia, N., Bowman, S.: A broad-coverage challenge corpus for sentence understanding through inference. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). pp. 1112–1122. Association for Computational Linguistics (2018), http://aclweb.org/anthology/N18-1101
28. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R.R., Le, Q.V.: Xlnet: Generalized autoregressive pretraining for language understanding. Advances in neural information processing systems **32** (2019)