Effects of Tokenization on Low-Resource Transformer-Based NMT Systems

Derek Mease **Jie Wang** **Yan Zhan**

Abstract

Past research has shown that NMT models for low-resource languages benefit from using different or reduced parameters than those typically utilized for larger models trained on massive amounts of data. Similarly, the choice of tokenization strategy can have a great impact on translation quality. In this paper, we analyze the effect of several popular tokenizers on translation quality for transformer-based NMT models in a low resource setting. We experiment with word, subword, and character units across several language pairs, and employ automatic evaluation metrics to report which tokenizers may be better suited for low-resource NMT transformer training.

1 Introduction

Despite the impressive performance of Neural Machine Translation (NMT) (Sutskever et al., 2014; Bahdanau et al., 2016; Vaswani et al., 2017), past research has shown that the performance of NMT drops sharply in low-resource conditions (Koehn and Knowles, 2017; Gu et al., 2018). So far, there has been much research focused on exploiting supplemental data to deal with this problem (Sennrich et al., 2016; He et al., 2016; Fadaee et al., 2017). Some research addresses this problem by optimizing NMT models with hyperparameter tuning (Sennrich and Zhang, 2019; Araabi and Monz, 2020).

Tokenization involves decomposing input text into smaller segments, and is an essential preprocessing step when training NMT systems. The choice of tokenization method can also have a significant influence on translation accuracy. However, little has been explored about the effects of tokenization on low-resource transformar-based NMT.

In this paper, using four different types of tokenization methods, Byte Pair Encoding (BPE) (Sennrich et al., 2016), WordPiece (Schuster and Nakajima, 2012), SentencePiece (Kudo and Richardson, 2018), and Character, we reassess the performance of NMT under the low-resource condition and how different tokenization methods affect the output quality of a NMT task. We then evaluate the NMT system with BLEU and CHRF3 metrics.

2 Related Work

Several previous works have explored the effects of tokenization on translation quality using LSTM-based models. Wu et al. (2016) introduce the WordPiece tokenization method, and show that its use with Google's Neural Machine Translation System (GNMT) produces higher BLEU scores than word and character tokenization. Similarly, Kudo (2018) presents a tokenization method based on the unigram language model along with subword regularization, which outperforms other tokenization methods with the same GNMT model. Durrani et al. (2019) examine word, byte pair encoding (BPE), morphological unit, and character tokenization methods, and compare their effect on the quality of embeddings learned by NMT models. Through experiments with external tasks such as morphological, semantic and syntactic tagging, they find that morphological segments are better for modeling syntax and semantics while character-based segments are better for morphology. Domingo et al. (2019) examine SentencePiece along with different implementations of word-level and morphological tokenizers. The study found that tokenization can greatly impact translation quality, but found no single best tokenizer, with some tokenizers leading to better results for specific language pairs.

While little research is found on the effects of tokenization on NMT performance in a low-resource setting, several papers suggest that hyperparameter tuning is essential to performance. Sennrich and Zhang (2019) show that for an RNN architecture, tuning BPE vocabulary size and word dropout can greatly improve BLEU scores for low-resource languages. For transformer models, van Biljon et al. (2020) find that reducing the number of layers in the encoder and decoder results in better quality. Araabi and Monz (2020) expand on the previous work by further reducing the dimensions of the feed-forward network and the number of attention heads. The common theme amongst this research is that a reduction in the number of model parameters is often beneficial in a low-resource setting.

3 Tokenization Methods

The first tokenization method we use is basic *word* tokenization. Tokens are simply produced by splitting on whitespace and punctuation. This method often leads to large vocabulary sizes which are prohibitive to training. A maximum vocabulary size is often set, and out-of-vocabulary words are replaced with a special *OOV* token. Translation quality suffers as a result, since we are unable to accurately model rare or unseen words. We use the NLTK *word_tokenize* method as our implementation.

We employ several subword tokenizers: Byte Pair Encoding, WordPiece, and SentencePiece. These tokenizers split words into subword units, reducing vocabulary size and better handling rare and unseen words. A maximum vocabulary size is defined, then the tokenizer is trained on a data set to build the vocabulary. For our experiments, we use a maximum vocabulary size of 20k for all subword tokenizers. We utilize *huggingface*[1] implementations for all subword tokenizers.

Byte Pair Encoding (BPE) (Gage, 1994) is a simple data compression method which replaces the most frequent byte pairs with an unseen byte. This merge process is repeated until the specified vocabulary size is reached or until there are no more possible merge operations. Sennrich et al. (2016) extend the BPE algorithm for language modeling by merging characters instead of bytes.

WordPiece (Schuster and Nakajima, 2012; Wu et al., 2016) has gained popularity as the tokenization method used for BERT. It trains by building a language model from a training dataset. Like BPE, it repeatedly merges word units to form new words until the desired vocabulary size is reached. Unlike BPE, it chooses the merged units that increase the likelihood of the training data for the language model, as opposed to highest frequency.

[1]https://github.com/huggingface/tokenizers

SentencePiece (Kudo and Richardson, 2018) is a subword tokenizer that combines BPE and unigram language model. SentencePiece is capable of learning word boundaries without the need for pre-tokenization, which can be useful for unsegmented languages such as Chinese.

Lastly, we use *character* tokenization, which simply splits each sentence into tokens of individual characters. This results in a very small vocabulary made up mostly of letters, plus punctuation and any other special characters. The small vocabulary size results in larger input vectors, leading to higher GPU memory requirements and longer training times. However, some research has suggested that character-level models can outperform subword-level models for NMT tasks (Lee et al., 2017).

Examples of the tokens produced by each tokenization method on a sample sentence are shown in Table 1.

4 Experimental Setup

4.1 Corpora

We train our model on four language pairs: German-English (de-en), French-English (fr-en), Italian-English (it-en), and Danish-English (da-en), using data from the European Parliament Proceedings Parallel Corpus (Europarl) (Koehn, 2005). To simulate a low-resource training setting, we randomly select sentence pairs from the full corpora to generate training, validation, and test sets of size 100k, 25k, and 25k respectively for each language pair. The Europarl data contains some sentences that do not have corresponding translations, and we ignore these during data set creation. To keep memory usage and training times within reasonable limits, we only select sentences containing 30 words or less. For preprocessing, sentences are converted to ASCII and lower-cased.

4.2 Model Architecture

For the transformer model, we use our own PyTorch implementation[2] adapted from code by Agarwal (2020). The model is based closely on the architecture of Vaswani et al. (2017). We implement a greedy search for the decoder

We initially perform a manual hyperparameter search using the *en-de* data and determine hyperparameters that result in the lowest validation loss (Ta-

[2]Code available at https://github.com/mease/csci-ling-5832-project

Table 1: Tokenization examples.

Original	phrase-based statistical approaches use separately engineered subcomponents.
Word	[’phrase-based’, ’statistical’, ’approaches’, ’use’, ’separately’, ’engineered’, ’subcomponents’, ’.’]
BPE	[’phrase’, ’-’, ’based’, ’Gstatistical’, ’Gapproaches’, ’Guse’, ’Gseparately’, ’Gengine’, ’ered’, ’Gsub’, ’com’, ’pon’, ’ents’, ’.’]
Word-Piece	[’phrase’, ’-’, ’based’, ’statistical’, ’approaches’, ’use’, ’separately’, ’engine’, ’##ered’, ’sub’, ’##com’, ’##pone’, ’##nt’, ’##s’, ’.’]
Sentence-Piece	[’_phrase’, ’-based’, ’_statistical’, ’_approaches’, ’_use’, ’_separately’, ’_engine’, ’ered’, ’_sub’, ’com’, ’pon’, ’ents’, ’.’]
Char	[’p’, ’h’, ’r’, ’a’, ’s’, ’e’, ’-’, ’b’, ’a’, ...]

Table 2: Transformer hyperparameters used in experiments.

Hyperparameter	Value
embedding dimension	512
attention heads	8
encoder layers	6
decoder layers	6
feed-forward dimension	2048
Adam learning rate	.0001
dropout	0.1
batch size	64 (16 for char)

Table 3: Best validation loss attained by each model.

	da-en	de-en	fr-en	it-en
Word	32.15	35.62	29.12	31.55
BPE	29.04	36.94	34.31	40.35
WordPiece	30.46	35.64	33.27	35.92
SentencePiece	32.57	36.42	32.90	33.21
Character	37.25	73.47	34.40	71.43

ble 2). While the models could likely be optimized for each language/tokenizer combination, we hold these hyperparameters constant for all training experiments to better observe the effects of the tokenization.

We train a transformer model for each language-tokenizer combination, resulting in twenty trained models. For each model, the source and target languages use the same tokenization method, but maintain separate vocabularies. We find that models tend to start overfitting after 15 to 20 epochs, so we train for a maximum of 30 epochs, and save the model from the epoch with the lowest validation loss. The best model for each language-tokenizer combination is evaluated against the corresponding test set using the metrics described in the later section.

Models were trained on a combination of Google Colab, Google Cloud Platform, and local workstation GPUs. Typical training times ranged between 6 to 9 hours. However, models using character tokenization took over three days to complete, and loss seemed to be improving still after 30 epochs. The large input sequences for character-tokenized sentences, along with the need to reduce batch size to fit GPU memory constraints, causes training time to skyrocket. We find that we do not have the compute resources or time to properly train character-based models. Our character models show the the highest validation loss out of all the model types (Table 3), especially for German-English and Italian-English, which were especially slow to train. Nonetheless, we include our findings for character tokenization here for comparison, with the caveat that these results could be improved given more time to train.

4.3 Evaluation Metrics

We use automatic evaluation metrics that have been reported to have correlations with human translation rankings. Our primary metric, BLEU (Papineni et al., 2002), evaluates the quality of a candidate translation against one or more reference translations by comparing the number of matching n-grams. We use the default NLTK *sentence_bleu* method, which implements a cumulative 4-gram BLEU score (BLEU-4).

We also use CHRF3 (Popović, 2015) as a secondary metric, which evaluates the translation based on the F-score of character n-grams with maximum length of 6. CHRF3 assigns 3 times more importance to recall in the score calculation, and is the CHRF variant shown by Popović (2015) to have the highest correlation with human rankings.

Both metrics produce a value between 0 and 1, with higher values indicating better alignment between the candidate and reference translations. For each trained model, we translate each sentence in the test set and calculate the average scores across all sentences.

5 Results

We find that WordPiece consistently produces significantly higher BLEU scores than other tokenization methods across all language pairs (Table 4). This is especially true for German-English and French-English, where the use of WordPiece increases the score by 3 to 4 BLEU points. For Italian-English, we see that SentencePiece and WordPiece result in nearly the same score. It is interesting to note that SentencePiece performs very well for Italian-English, but quite poorly for Danish-English. As mentioned before, the character-based models did not train to completion during our experiments and produced the lowest scores, as expected.

The CHRF3 score results are presented in Table 5. For Danish-English and German-English, WordPiece once again produces higher scores, but the difference between scores achieved with other tokenizers is less significant than what was observed for BLEU scores. For French-English, simple word tokenization attained a higher score than the other more advanced methods. Consistent with the BLEU scores, SentencePiece produces the highest CHRF3 score for Italian-English and the poorest score for Danish-English.

It is observed that some of the models using character tokenization (*da-en* and *fr-en*) achieved surprisingly high CHRF3 scores considering they were only partially trained. It is likely that the models are generating enough matching *character* n-grams between candidate and reference translations to boost the CHRF3 score, but failing to generate enough matching *word* n-grams to produce a reasonable BLEU score.

Table 4: Average BLEU score for translations from the test set.

	da-en	de-en	fr-en	it-en
Word	30.42	28.14	30.12	31.13
BPE	30.81	28.60	29.79	30.58
WordPiece	**32.93**	**32.75**	**33.63**	32.04
SentencePiece	15.64	28.26	29.67	**32.08**
Character	18.02	0.59	15.84	1.04

Table 5: Average CHRF3 score for translations from the test set.

	da-en	de-en	fr-en	it-en
Word	54.95	52.67	**56.51**	54.60
BPE	56.80	53.47	55.82	53.85
WordPiece	**56.82**	**54.16**	56.11	54.24
SentencePiece	50.71	53.82	56.20	**54.92**
Character	52.71	22.49	52.17	25.53

6 Future Work

Potential future work involves researching the linguistic properties of the different languages to determine why some tokenizers produce better NMT results when paired with specific languages. For example, SentencePiece appears to work best for translating Italian to English. Additional model training to translate in the other direction from English to the target languages could provide more data and insight for better analysis. Utilizing different data sets for actual low-resource languages would provide a more realistic setting for experiments, as opposed to trimming down the larger data set to simulate a low resource setting.

In our implementation, we use a greedy method to decode the translations, but quality could be improved by implementing beam search. Higher scores might be attained with more tuning of model parameters and tokenizer vocabulary size. Additionally, character-based models should be explored further. By utilizing more compute resources and longer training times, character tokenization could produce results competitive with the other methods.

References

Rahul Agarwal. 2020. Transformers in NLP: Creating a Translator Model from Scratch.

Ali Araabi and Christof Monz. 2020. Optimizing Transformer for Low-Resource Neural Machine Translation. *arXiv:2011.02266 [cs]*. ArXiv: 2011.02266.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*.

Elan van Biljon, Arnu Pretorius, and Julia Kreutzer. 2020. On Optimal Transformer Depth for Low-Resource Language Translation. *arXiv:2004.04418 [cs]*. ArXiv: 2004.04418.

Miguel Domingo, Mercedes Garcıa-Martınez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. 2019. How Much Does Tokenization Affect Neural Machine Translation? *arXiv:1812.08621 [cs]*. ArXiv: 1812.08621.

Nadir Durrani, Fahim Dalvi, Hassan Sajjad, Yonatan Belinkov, and Preslav Nakov. 2019. One Size Does Not Fit All: Comparing NMT Representations of Different Granularities. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1504–1516, Minneapolis, Minnesota. Association for Computational Linguistics.

Marzieh Fadaee, Arianna Bisazza, and Christof Monz. 2017. Data Augmentation for Low-Resource Neural Machine Translation. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 567–573.

Philip Gage. 1994. A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

Jiatao Gu, Hany Hassan, Jacob Devlin, and Victor O. K. Li. 2018. Universal Neural Machine Translation for Extremely Low Resource Languages. *arXiv:1802.05368 [cs]*.

Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. 2016. Dual Learning for Machine Translation. page 9.

Philipp Koehn. 2005. Europarl: A Parallel Corpus for Statistical Machine Translation.

Philipp Koehn and Rebecca Knowles. 2017. Six Challenges for Neural Machine Translation. *arXiv:1706.03872 [cs]*.

Taku Kudo. 2018. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. *arXiv:1804.10959 [cs]*. ArXiv: 1804.10959.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. *arXiv:1808.06226 [cs]*. ArXiv: 1808.06226.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully Character-Level Neural Machine Translation without Explicit Segmentation. *arXiv:1610.03017 [cs]*. ArXiv: 1610.03017.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, USA. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and Korean Voice Search. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. *arXiv:1508.07909 [cs]*. ArXiv: 1508.07909.

Rico Sennrich and Biao Zhang. 2019. Revisiting Low-Resource Neural Machine Translation: A Case Study. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 211–221, Florence, Italy. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. page 9.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *arXiv:1706.03762 [cs]*. ArXiv: 1706.03762.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv:1609.08144 [cs]*. ArXiv: 1609.08144.