Patching Leaks in the Charformer for Generative Tasks

Anonymous ACL submission

Abstract

Character-based representations have important advantages over subword-based ones, including increased robustness to noisy input and removing the need of tokenization preprocessing. However, they also have a crucial disadvantage: they notably increase the length of text sequences. The GBST method from Charformer groups (aka downsamples) characters to solve this, but allows information to leak when applied to a Transformer decoder. We introduce novel methodology to solve this information leak issue, which opens up the possibility of using character grouping in the decoder. We show that Charformer downsampling has no apparent benefits in NMT over previous downsampling methods.

1 Introduction

Most state-of-the-art neural machine translation (NMT) systems operate on the subword level, typically using a preprocessing technique like Byte-Pair Encoding (BPE) (Sennrich et al., 2016) for combining characters into subwords. However, using a subword representation can often mask important information given by characters, from the syntactic relatedness of words (e.g., "bake" and "bakes" may be assigned each their own unique token) to literary devices such as rhyme and alliteration. Although a byte-level pretrained language model, ByT5 (Xue et al., 2021), demonstrated that it was more robust to misspellings than its subword counterpart, T5 (Raffel et al., 2019), results on common benchmarks such as GLUE did not reflect this inherent advantage. Similarly, in NMT, we have not seen any advantage from character-level models reflected by their BLEU, chrF, or COMET scores to date (Libovický et al., 2021).

The major inhibitor for character-level models is inefficiency, which is due to the increased length of the input and output sequences. For example, the average number of characters per subword for English is around 4 (Xue et al., 2021), so the input sequence to a character-level model is effectively 4 times longer. With a Transformer model, the problem is compounded by the complexity of self-attention. To address this, models such as the Charformer (Tay et al., 2021) introduce a downsampling method prior to the Transformer, which combines characters into pseudo-words, reducing the length of the sequence. The downsampling method used in the Charformer, GBST, was originally intended only for use in the encoder, and recent works attempting to apply the GBST layer to the decoder have failed (Libovický et al., 2021).[1] Our contributions include:

1. We show that there is an information leak in the GBST layer which breaks the typical NMT training scheme of a Transformer model.
2. We resolve the information leak issue, allowing it to be used in a Transformer decoder.
3. We provide a simple test to check for information leaks in any task which exhibits causality.
4. We show that despite Charformer's current popularity, the GBST layer does not perform as well as earlier methods such as Lee et al. (2017) for NMT.

We first discuss the information leak issue, including our test to confirm the problem and our potential solutions in Section 2. Details of our experiments are in Section 3, results and discussion thereof in Section 4, and concluding remarks in Section 5.

2 Patching Information Leaks

The Charformer modifies ByT5 with the addition of the gradient-based subword tokenization (GBST)

[1] In consultation with the authors, we determined that their reported results using GBST in the decoder were in fact using a different downsampling method.

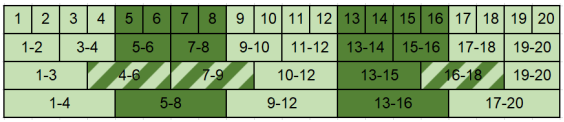

Figure 1: Example of the mean mixing in the GBST layer, with alternating block colors assigned according to what information is accessible via mean mixing. Numbers indicate the range of positions accessible to each n-gram. The striped n-grams indicate a source of information leak.

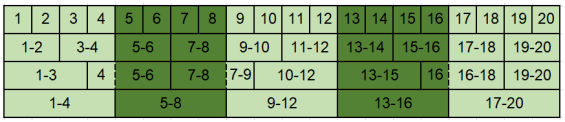

Figure 2: Our third proposed solution to preventing information leak.

layer. This layer mixes character representations (which are also informed of their relative positions via a convolution) using a simple mean computed over character n-grams up to length 4.[2] The model then selects a weighted average of these representations before a downsampling via a block-wise mean pooling. The result is a sequence of pseudo-word-level embeddings that is fed in directly to the Transformer (in contrast with the standard sequence of subword embeddings).

The GBST layer can be applied directly to a Transformer encoder without issue, but it cannot be applied to a Transformer decoder for generative tasks. This is due to an information leak, where information about characters in future blocks can end up in prior blocks. This occurs for 2 reasons: the convolution used to inform position, and the character n-gram means, both of which can overlap with the block-wise separations.

Concerning the convolution, each character is informed by its neighboring characters, which is necessary for the convolution to serve as a positional embedding. However, this creates the issue that characters on the right side of a block will be informed by characters to the left in a future block.

The character n-gram averaging similarly can obtain information from future blocks. As seen in Figure 1, with a downsampling factor of 4, the issue occurs with trigrams, where the 4th position is averaged with the 5th and 6th, despite being in separate blocks. Therefore, when the Transformer must predict the block containing the 5th and 6th characters, it has already received information about them, and thus can learn to simply copy the characters.

2.1 A Simple Test for Information Leaks

To confirm that there is indeed a leak in the GBST layer, we set up a simple model consisting of a GBST layer, followed by an upsampling layer, which is simply a linear layer that takes the downsampled block representation and upsamples it back to characters. We then train the model to predict a sequence of random characters, conditioned on the prior characters in the sequence, using a left-padding of BOS tokens equal to the downsampling factor. (e.g., '[BOS] [BOS] [BOS] a b c' is used to predict 'a b c d e f'). We expect that, if there is no information leak, the model will have a near-random accuracy, and conversely if the accuracy is significantly higher than random, there is an information leak.

In Table 1, we show the specific positions in a sequence affected by the leaks. Details of these experiments can be found in Appendix A. We use a vocabulary of size 100, so the accuracy of a perfectly random classifier should be around 1%, however we see in several cases that the models achieve significantly better. In the worst case, with a downsampling factor of 4 and using the convolutional positional embeddings, 75% of the tokens are leaked.

To resolve this issue, we considered three potential solutions: adding more padding, removing any n-grams causing a leak, or applying a mask to these n-grams.

Approach 1: Additional Padding

The simplest potential solution is to pad the sequences enough so that there is no possibility of leaking. Preliminary experiments indicated that padding with 2 times the downsampling factor prevented any leak. This approach however introduces the issue that the predictions are no longer conditioned on the block 1 step back, but rather the block 2 steps back.

Approach 2: Remove Overlapping N-Grams

Our second approach instead addresses the sources of information leak. First, the convolution used as a positional embedding is replaced with a static sinusoidal positional embedding, as used in Vaswani et al. (2017). Second, the means for n-grams where there is overlap are removed. For example, for a

[2] For decoding, we include n-grams up to the length of the downsampling factor.

δ	Conv?	1	2	3	4	5	6	7	8	9	10	11	12
2	No	0.0103	0.0109	0.0103	0.0122	0.0106	0.0113	0.0141	0.0122	0.0147	0.0116	0.0113	0.0075
	Yes	1.0000	0.0116	0.9941	0.0097	0.9947	0.0100	0.9959	0.0113	0.9966	0.0097	0.9962	0.0113
3	No	0.1356	0.0078	0.0109	0.0100	0.0078	0.0103	0.0241	0.0106	0.0088	0.0091	0.0103	0.0081
	Yes	1.0000	1.0000	0.0072	0.9987	0.9981	0.0106	0.9772	0.9619	0.0131	0.9966	0.9981	0.0081
4	No	0.1347	0.0338	0.0137	0.0113	0.0262	0.0106	0.0122	0.0100	0.0113	0.0097	0.0063	0.0072
	Yes	1.0000	0.9994	1.0000	0.0119	0.9775	0.9744	0.9781	0.0137	0.9966	0.9900	0.9950	0.0122

Table 1: Accuracies obtained by the simple model for each position in a target sequence of 12 random tokens, with varying downsampling factor (δ) and either convolutional or sinusoidal positional embeddings. Blue denotes accuracies within random chance ($p > 1e-3$), while red denotes accuracies that are significantly higher than random chance ($p < 1e-10$), showing an information leak.

downsampling factor of 4, the means for all trigrams are removed. These 2 alterations remove the possibility for information leak, however it is possible that the removal of trigrams or other n-grams will substantially worsen the model's ability to learn contextual character embeddings before downsampling into blocks.

Approach 3: Apply Causal Mask

Our third approach builds on our second approach by keeping the sinusoidal positional embeddings and, rather than removing problematic n-grams, applies a causal mask when computing the mean. This can be seen in Figure 2, where overlapping n-grams are split by block, with the right side being informed by the left, but the left side being uninformed by the right. This approach allows n-gram information such as trigram information to remain when it is not overlapping, however the information present in the blocks is not always consistent. This inconsistency might be difficult for the model to discern.

3 Experimental Setup

Our experiments use the IWSLT2017 data[3] for English–German and English–Arabic, using the test sets from 2010 and 2015 for validation and testing, respectively. We remove sentence pairs where the English sentence is longer than 256 characters from the training data. To keep our vocabulary size consistent across all languages, we tokenize according to UTF-8 bytes rather than using a character vocabulary.[4] We also include subword-level models, with vocabularies generated with SentencePiece (Kudo and Richardson, 2018). We chose a vocabulary size of 16 thousand, as that roughly corresponds to a downsampling factor of 4.

We use the Transformer model with the same parameters as Transformer Base (Vaswani et al., 2017). Our choice of hyperparameters can be found in Appendix B. All of our models are trained on a single Nvidia V100 (32GB) GPU.

For decoding with Lee et al.'s and the Charformer downsampling methods, we apply the two-step decoder of Libovický et al. (2021), which adds an LSTM layer to the head of the Transformer, receiving its hidden states in addition to character embeddings of the so-far generated output sequence, and outputting the next characters. The parameters used for Lee et al.'s method also follow Libovický et al. (2021).

For evaluation we use BLEU (Papineni et al., 2002) and COMET (Rei et al., 2020). For COMET we use the reference-based `wmt20-comet-da` model. Our code is made freely available.[5]

4 Results

We compare the performance of our modified GBST layers in Table 2. First, we observe the GBST with additional padding, like the non-causal GBST, fails to learn any form of translation. This indicates that predicting two blocks into the future is too difficult a task for the model to make any meaningful progress in learning to translate.

δ	Non-Causal	Padding	Removal	Masking
2	0.00	0.04	25.49	24.59
3	0.00	1.35	24.90	22.16
4	0.00	0.62	22.07	17.47

Table 2: BLEU scores of our 3 approaches on German→English.

We can also see that the simpler approach of dropping overlapping n-grams works better than applying a causal mask. We suspect this is due to a lack of consistency within the masked n-gram

[3] https://sites.google.com/site/iwsltevaluation2017/TED-tasks

[4] Operating on the byte-level also follows Tay et al. (2021), despite the name "Charformer" perhaps suggesting otherwise.

[5] Link to code will be added here.

representations, as some are simply duplicates of the lower order n-gram representations.

The downsampling factor also plays no role in the difference between the methods. Although the average length of a subword in English and German is close to 4 characters, making a downsampling factor of 4 an intuitive choice, a higher downsampling factor leads to worse performance.

Method	δ	de-en	en-de	ar-en	en-ar	avg
Subword	1	27.23	24.08	25.59	**11.22**	**22.03**
Char	1	**27.37**	**24.32**	**26.34**	8.73	21.69
r-GBST	2	25.49	22.17	24.21	7.81	19.75
	4	22.07	20.25	22.24	8.32	18.22
Lee	2	27.10	23.27	25.05	9.94	21.34
	4	24.32	21.06	22.63	9.24	19.31

Method	δ	de-en	en-de	ar-en	en-ar	avg
Subword	1	0.2034	**0.0730**	0.0628	**0.1203**	**0.1149**
Char	1	**0.2215**	0.0396	**0.1417**	-0.0079	0.0987
r-GBST	2	0.1097	-0.1010	0.0349	-0.1149	-0.0178
	4	-0.1230	-0.3519	-0.1354	-0.1380	-0.1871
Lee	2	0.2153	-0.0187	0.0737	0.0439	0.0786
	4	0.0191	-0.2374	-0.0726	-0.0440	-0.0837

Table 3: BLEU (top) and COMET (bottom) scores.

Despite the modified GBST with n-grams removed (henceforth r-GBST) being the best performing modification, Table 3 shows us that it is still outperformed by the downsampling method from Lee et al. (2017). The mode of operation between Lee et al.'s method and the GBST method is similar: both achieve a mixture of characters, focusing only on neighboring characters to reduce computational complexity. While GBST achieves this with averaging across unigrams to 4-grams, Lee et al. uses convolutions with differing kernel sizes. The mixing via convolution is more uniform in nature; for example, the convolution with kernel size 3 is analogous to the tri-gram mixing, however the convolution does not have a hard boundary after every 3rd character. This may be the reason for Lee et al.'s method's superior performance.

Regardless to the reason for its superior performance in NMT, these results raise the question of the significance of the performance of the Charformer for any NLP task. If Lee et al.'s method was used in the same pretraining setup used in Tay et al. (2021), would we perhaps see superior performance?

Both downsampling methods also perform worse than using no downsampling method, however there is a trade-off in training time. In Table 4, we show the time it takes to train and evaluate on the test set post-training. We include character and subword-level models where we use the two-step decoding method, in order to separate the effect of downsampling from the decoding method.[6] We can see that our downsampling methods are faster for both training and generation than the character model. However the subword model is still the fastest, all while achieving the best performance.

Method	Decoding	δ	Epoch	Completion	Eval
Subword	Normal	1	5:34	3:52:31	1:33
	2-Step	1	5:47	5:15:57	1:40
Char	Normal	1	14:37	14:52:21	5:15
	2-Step	1	16:45	17:02:35	5:19
r-GBST	2-Step	2	10:38	10:15:02	3:19
		4	8:08	8:16:32	2:03
Lee	2-Step	2	13:09	13:22:48	5:45
		4	10:07	11:55:09	2:43

Table 4: Average time to train for 1 epoch and to completion, and to evaluate on the German→English dataset (hours + minutes + seconds). Times for other language pairs are proportionally similar.

5 Conclusion

Character-level or byte-level models are intuitive for a variety of reasons but currently suffer from a lack of efficiency due to the longer sequences. Downsampling methods such as the GBST layer in the Charformer looked promising, however it is not usable for generative tasks without modification due to a leak of information flowing from future blocks. With modification, the GBST layer does not perform as well as older methods such as that of Lee et al. (2017).

Despite the intuitiveness of downsampling from characters to pseudo-words, we see a clear trade-off of performance versus time, with performance decreasing by several BLEU points, but the training and generation time being reduced to up to 50% of the non-downsampled model.

Although both downsampling methods tested do not reach the performance of the standard character-level model, subword-level models show that shortening the sequence length can lead to appreciably faster models without any sacrifice in performance. Thus some form of downsampling is likely beneficial, and our method for checking for information leaks can serve as a useful debugging tool for future research.

[6]The performance of the 2-step character and subword models are similar to their normal counterparts.

References

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378.

Jindřich Libovickỳ, Helmut Schmid, and Alexander Fraser. 2021. Why don't people use character-level machine translation? *arXiv preprint arXiv:2110.08191*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

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 of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Yi Tay, Vinh Q Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2021. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2021. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *arXiv preprint arXiv:2105.13626*.

A Test for Information Leaks

Our approach for testing for information leaks in a downsampling method trains for 5000 iterations in batches of 32. We use the Adam optimizer (Kingma and Ba, 2014), with a learning rate of 1e-4. These numbers were determined empirically, based on the degree of separation seen from the accuracies of the leaking versus non-leaking tokens. The accuracies in Table 1 are obtained over 100 batches, or 3200 samples.

B Main Experiments Hyperparameters

We train our models using AdamW (Loshchilov and Hutter, 2017) with a learning rate of 2e-4, a linear warmup of 4000 steps, a batch size of 128, and label smoothing factor of 0.1. The learning rate was chosen from a grid search, the batch size chosen empirically, and the warmup steps and label smoothing factor were based on Libovickỳ et al. (2021). We use an early stopping criterion of no improvement on the validation set with a patience of 10.