Temporal Downsampling for Byte level Transformers

Pawan Jayakumar
pjayakumar@ucsd.edu

1 Introduction

One of the classic shortcomings of language models even as sophisticated as GPT-4 is failing to answer questions like "How many r's are in strawberry?" Part of the reason this question is tricky is because language models train on a sub-word level tokens. This issue also arises when there are typos or noise in input prompts or training examples(Linting Xue, 2021). Despite this weakness, they have held an advantage over byte-level tokenization because of their inductive bias and ability to compress text sequences. Towards this end I investigated current methods to deal with this. Main contributions of this work:

- Repurpose a simple dataset for this task
- Train a model based on gpt2's tokenizer
- Train a model based on ByteT5 tokenizer
- Train models with improved performance over the previous two

2 Related work

(Jacob Devlin, 2019) I thought It was important to read up on the BERT paper because they do some analysis on how they trained their encoder which I thought could be helpful if I pre-train mine. In the interest of simplicity though I endedup just jointly training it with the text classifier. (Linting Xue, 2021) This paper took the popular T5 model and made a byte level version of it. This paper also introduced the value proposition for these kinds of byte level models in the first place, as well as some challenges encountered while training them. (Yi Tay, 2022) This paper inspired this project because it was the first one I read which directly gave me a method to "learn" a subword based tokenizer during training time. While it was good work, one key limitation is the rigidity of the solution. Since it uses fixed 'subwords' or block sizes, the learned representations aren't as good as some other approaches (Nathan Godey, 2022) This paper showed really good results and even provided a state of art language model comparable to ByteT5. Their key innovation was using soft block boundaries using probabalistic methods that weight each byte as likely to be part of a particular block in the downsampled sequence. My issue with their approach was that they kind of just attach another transformer block before their down sampling and I would have liked to seen ablations. Furthermore, their innovative technique was only applied to the encoder and not "upsampled" for the decoder. This means the decoder still struggles with the same issues as regular byte level models. (Lukas Edman, 2022) This paper explored how we can transfer some of the inductive biases that subword tokenizers bring into the byte level models. Unfortunately I don't think the paper is quite ready yet as much of the details of their method don't seem to be explained well and there is no link to the code. Still I think the idea of somehow boostrapping model representations with pre-identified rules and then using additional data to refine them seems like a cool idea.

3 Dataset

Since I was limited on compute and time, I chose to repurpose the speeches dataset we used for PA2. This has many advantages 1) its already cleaned. 2) it was small enough to easily modify on the fly (2000 sentences for the train split and 750 for the test split). I created a version of the train and test splits that have been corrupted with typos. I used a variable to change the percentage of corrupted characters from 0 (no typos) to 100 (essentially random text). The objective of the task is simply

to predict whether the speaker is Obama, George W Bush, or H W Bush. The task of learning on corrupted data is difficult because it may result in many unknown tokens showing up in the input to the model. For examples of the corrupted speech text, see the error analysis section.

3.1 Data preprocessing

This is not a traditional preprocessing procedure per-se, but I wanted to make this a realistic dataset so the way I created it is interesting. In order to simulate someone actually making typos of varying frequency, I go character by character and with probability α, replace it with a character that is *physically near the current character.* This was achieved by the use of openAI's o1 model where I told it to create a dictionary mapping keys to near by keys and the resulting table was pretty accurate. For example, the character 'r' could be replaced (with equal probability) with 'e', 't', '3', '4', '5','f', and 'd'.

4 Baselines

4.1 Model selection

I wanted to highlight the weakness of both subword tokenization as well as the weakness of pure byte-level tokenization and therefore used both as baselines. For subword tokenizaiton I used GPT2's tokenizer and for byte level (as well as all byte based methods) I used the ByteT5 (Linting Xue, 2021)tokenizer. Both operate on a string of text and return id's for each token. The difference is that the vocabulary size for gpt2 is 50000 instead of 256 for ByteT5. As for the model architecture it follows the original transformer paper except that I only have an encoder. The encoder final output is averaged over the sequence dimenion and then fed into a dense neural network for text classification. I trained all models with 1e-4 learning rate, AdamW optimizer, for 15 epochs. For subword based tokenization I used a batch size of 16 while for byte tokenization models I used a batch size of 8. This was because the training for byte level was more unstable. Another difference is that the byte models had a context length of 256 as opposed to a context length of 64. This was done because the average token length in the gpt2 tokenizer is 4, and it made sense to give models equal length input text. More details about specific model architectures and values can be seen in the code.

5 Methods

5.1 Concept

Having read the related work, I knew that the biggest challenge byte level tokenizers face is the loss of the inductive bias that subword tokenization provides. While its hard to figure out how to fix that, an easier problem to fix is that subword tokenization results in shorter sequences. This lets the model run faster (linear attention cost even with optimizations) and focus on less tokens. In some sense, a transformer is already compressing along the sequence dimension as you work through the encoder stack, but this is being made explicit in my approach because I have two main ways to do compression: convolutions and sequence models. The reasoning is simple. In order to emulate subword tokens but make them something that is learned, we must find a way to aggregate information between nearby bytes. Convolutions are great at this. I also tried RNN's because in theory they also propogate information through the hidden state. RNNs have been unfavored recently because of their inability to hold state for long term, but in this case, it is actually a good thing because subword blocks only should depend on nearby context. To downsample, we can then perform max pooling. However the issue with both methods is that there is some degree of inflexibility because all blocks are the same size. This was also an issue in charformer. The approach in (Nathan Godey, 2022) was smart because even though they ultimately truncate their finals sequence to 1/4th the original length, the different blocks could be different sizes and correspond to semantic segmentations. I thought the sliding window approach was basically just another transformer layer so I wanted to do the same thing but with less compute. Instead I tried using a linear RNN and then using a linear layer to calculate probabilities of the current token being the start of a new block. Then you can use gumbel softmax to "sample" from this (without breaking the autograd during back propogation. and create a binary sequence where the 1's represent block starts. Using cumulative summation, you can take sequence elements from the linear RNN and pool them together if they have the same corresponding number in the cumsum array.

5.2 Working implementation

I was able to implement the two simple methods described above. The complex approach with 'soft' block lengths was not working because I had too many complex operations and the gradients werent able to smoothly flow and update the layers. I tried many techniques to help this for example gradient clipping, different learning rate schedulers, normalization, but none of this ended up working. I feel like there may be something small Im missing that could help this, but having gone through this I understand why the Manta paper did the approach they did (even if it seemed quite involved).

5.3 Compute

- I ran all experiments on my laptop as this was not too computationally intensive
- I also ran some on collab when I was more ambitious about what model sizes i could use but I did not end up going that direction.
- In terms of run time each training run took about 2-3 minutes for 15 epochs

5.4 Results

Typo%:	**0.00%**	**8.00%**	**16.00%**	**32.00%**
ByteT5	44.8	49.87	49.93	42.13
gpt2	**75.2**	**54.4**	49.07	41.87
conv	74.4	54.27	**50.0**	**46.67**
rnn	67.6	55.07	48.67	46.13

Table 1: Performance of different tokenizers under varying levels of corruption.

- As seen in Table 1, on clean text with no typos, subword tokenization performs best
- For higher corruption rates, byte level models are on par or better than subword tokenization
- Modifications on top of byte level models yielded better results
- Convolutions work slightly better than RNNs

6 Error analysis

Generally as the corruption rate goes up, performance drops. Furthermore, the gap between the train and test performance widens. This is probably due to not having a very large dataset and over fitting. The models learn representations that are too specific to the train set and since they have no help from the tokenizer, they tend to struggle. To be honest, even for me it is very hard to classify these speeches or even see any significant differences in the ones it struggled at. For example these sentences have a corruption rate of 0.32 which is barely even legible.

- 'We came in abd gefe i4 emdrgency frrwtmen6: Go4 ghe tempe5atu44 f9sn gg lo2er9nt regulay9kh, goy tve bl8kd pressure zlw when'
- W3 gqh2 the moat pjw2rfuo milita5y om Earth, vu6 that's bot whad jajew ua syrjkg.'
- 'Our mh;ira4y f8rces jm Anbzs sfe kulling qnd csp5urimg am Qaeda ldazerwl and rhey are p5ptextikg tue logal pkpuladikkl'

I dont think you could ever guess this even with all the samples, but it got the first two wrong and the third right. Most likely though its probably the case that the more corruptions, the more likely it got it wrong. To test this theory you could train a classifier to do this to see the correlation but I think its pretty intuitive.

In hindsight I should have picked a dataset which is easy for humans but hard for AI, but these are usually reasoning related and there is no chance a model this small can do anything relevant if even models with billions of parameters struggle. This also gets into model interpretability because it would be nice to see how byte level models learn subwords as part of training.

7 Conclusion

Coming up with a new architecture is definitely not an easy task. Generally its quite hard to overcome inductive biases. The key in the past was to overcome these biases using better data, and more of it (as well as the associated compute). These elements were not present in my design. I had many complicated ideas, but none of them were able to work in practice because the training was too unstable and the gradients didn't provide stable updates towards convergence. If I could continue working on this, I would like to scale up the idea to training small language models on the order of byt5-small and do more in depth research

into training stability to see what methods I can find for both downsampling and upsampling. I was honestly surprised at how well gpt2 was able to adapt to not having good tokens for the task. This may be due to the simplicity of the task as it is able to learn quite well on the train set but then not generalize to the test set.

8 Acknowledgements

- Parts of the brainstorming, debugging, and small snippits were done using Claude 3.5
- Slight change

References

Jacob Devlin, Ming-Wei Chang, K. L. K. T. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding]. *ACL Anthology*.

Linting Xue, Aditya Barua, N. C. R. A.-R. S. N. M. K. A. R. C. R. (2021). Byt5: Towards a token-free future with pre-trained byte-to-byte models.

Lukas Edman, Antonio Toral, G. v. N. (2022). Subword-delimited downsampling for better character-level translation. *ArXiv*.

Nathan Godey, Roman Castagné, d. l. C.-B. S. (2022). Manta: Efficient gradient-based tokenization for end-to-end robust language modeling. *EMNLP*.

Yi Tay, Vinh Q. Tran, S. R. J. G. H. W. C. D. B. Z. Q. S. B. C. Y. D. M. (2022). Fast character transformers via gradient based subword tokenization. *ICLR*.