Improving Large Language Model Arithmetic with Efficient Tokenization: Using Special Tokens in Neural Networks

Jesus A. Fregoso

February 25 2025

Abstract

In human brains myelination is one of the ways to increase performance and efficiency of certain actions. Using specialized tokenization and little augmentation of data, it is possible to see improvements in performance of basic arithmetical operations. The struggle of large-language models (LLMs) in basic arithmetic is usually due to the inability to adequately train in order to sufficiently generalize. The inherent limitations imposed by using stochastic models to solve deterministic problems still leave further room for efficiency and improvement, but, this idea greatly improves the performance of even the most basic LLMs, even solving mathematical problems using data not included in its training datasets. This work addresses the problem in which LLMs can struggle in generalization of certain topics, especially mathematics, this paper solely focuses on arithmetic operations due to the researchers hardware limitations, please read further research for more details.

1 Introduction

In current large-language models we see struggles especially in terms of arithmetic operations that the LLMs struggle to generalize algorithms. The current performance seems to come with memorization of certain results or generalization of certain number theory concepts that helps approximation in the couple first and last numbers. The problem with large language models arises with the inefficiency to generalize with data, where it works great with extreme amounts of data and extremely large models; this inefficiency with data and infrastructure for some results has vast environmental impacts which achieve sub-par results in areas where information cannot be memorized. With these problems, it is proposed to improve these results with a small tweak to the tokenizer to greatly improve results

2 Implementation Background

There are many ways to implement this but the approach implemented used the commonly used tokenizers but simple changes in the training and augmentation of the data, without losing or changing the underlying information, with temporary tags. These temporary tags are used in very special occasions and temporarily as they are used to help models more efficiently generalize algorithms for in this case arithmetic, this is done by implementing rare "special tokens" which are added during encoding of data. These rare "special tokens" which are seen in very specific circumstances have no actual direct functionality and mimic the myelination of neurons in the brain as they do not directly deal with the signals but are used to help speed and performance. This also relies on the tokenization methods of integers being changed to all being single digit, this is to force the model to generalize algorithms instead of relying on memorization. With this we can see a form of reinforcement learning able to be applied where the model is able to solve problems not originally contained in its training data. This is great news especially if it can be replicated for higher levels of mathematics and especially coding/logic. We see that many of the results of these fields in LLMs are solely from memorization of data scraped from the internet (including copyrighted book/textbook material) whereas with this model it is able to more efficiently perform without this data at a higher accuracy. This presents an exciting opportunity especially if it can be replicated and expanded further, using the amount of data that can be generalized with smaller amounts of data. While this is a rather informal and ad hoc attempt at improving LLMs, it seems that is an effective method. With these results it's hoped that we can further expand these results and apply them in related areas or even improved and optimized.

3 Approach

This approach uses a simple model based on a modified decoder transformer shown in attention is all you need (Vaswani et al. (2017)). The minimal size was necessary due to the infrastructure restraints as well the need to prevent over-fitting by making a model too large for the data. The transformer was trained on 350,000 randomly generated addition, subtraction, multiplication, and hybrid equations(division omitted from the experiment as to only deal with integers, although may appear in future research), the arrangement of multiplication and addition/subtraction problems also switches as to enforce proper order of operations is learned. Due to the infrastructure and simplicity of the data only 84 embedding dimensions were used. This small model was used to prevent memorization and over-fitting of the data. Initially the data was composed of only strings of the arithmetic equations, then before encoding and embedding of the data, we augmented it to include a special token before and after the operator as shown in Figure 1. The intuition in this approach is that including and pairing rare tokens together in certain combinations, will force the model

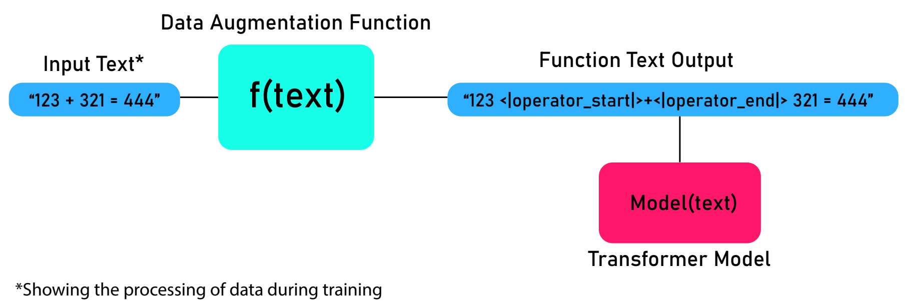
Data Augmentation Function
Input Text*
"123 + 321 = 444"
f(text)
Function Text Output
"123 <|operator_start|>+<|operator_end|> 321 = 444"
Model(text)
Transformer Model
*Showing the processing of data during training

Figure 1: Example of Data Augmentation Function in Training

to generalize better by giving a specific operation 3 tokens instead of just one.

The encoding and tokenization was forced to a minimum as these all needed to be single digit, so simple encoding and decoding methods worked. The testing of the functions utility was done by having the data trained and tested on two different models, one with and one without the augmentation function. Taking into the account the fact that predictions would be made easier as it was essentially guaranteed 2 free correct predictions, it was found that just adding these special tokens results in approximately 6 percent of superficial improvement of loss function and the results will include the removal of that 6 percent.

4 results

Due to the nature of this model and how it can be accuracy tested, we include the results for improvement of loss function and how the models compare in the accuracy of its arithmetic.These improvements were measured by calculating the percent difference between the loss of the control and the special tokenized model, taking into account the free 6 percent. Initially we saw improvements of the validation loss of 4 percent, quickly moving to 12 percent and finally stabilizing at 10 percent using the augmentations. We get similar results even after moving the model to larger embedding dimensions and halving the dropout rate. Now for testing the actual accuracy of the model we needed to design a certain cost function, which measures the accuracy while taking into account the size of the problem. We do this so, if a model incorrectly returns $1 + 1 = 52$, it will cost much more than $5 * 10 = 550$, due to the relative size of the actual result. The cost function was a simple mean squared error, but divided by the correct answer, capped at 50000 for wrong answers due to results becoming too skewed after only one, albeit extremely wrong, answer; Z is the correct answer and Y is the predicted answer.

$$min((Y - Z)^2/Z^2, 50000)$$

Using this, we tested the models outputs on 5,000 randomly generated test questions and ended with a 23 percent improvement in accuracy when using the augmentation model. Showing better thane expected results.

The code for the model, the augmentation function, dataset, and the function to generate a similar dataset all published in my GitHub repository . I'd like to note that much of the code for the actual model trained is based off of an excellent YouTube video by Andrej Karpathy linked here. Further tuning and messing around with hyper-parameters is encouraged to potentially find greater results. Applying the idea is also highly encouraged for fields where LLMs struggle. If any inspiration arose from this paper all that is asked for a citation in any papers published, which one can only hope will be many!

5 Conclusion

It seems like specialized tokenization methods are especially useful for logic and mathematical equations. Although it requires a little preprocessing of data this can be used to achieve similar or better results of extremely large models with huge amounts of data. This opens the possibilities and door for more to be explored, it can push the boundaries of current LLMs while making some smaller and more efficient, grappling with its environmental impacts and cost barriers. With some additional testing, preliminary results indicate favorable scaling conditions, with greater improvements available. Only the future will show the power of efficient tokenization, where it can have great results in hopefully more than math.

6 Important Note to Reader

This research was highly concentrated on one topic and trained on extremely small models. This was independent research with extremely limited access to hardware due to funding, all the work was done on older-mid tier GPUs severely limiting the potential scope of the project and hampering the ability to work on further research. Continuation of independent research (only 1 researcher/author) is suboptimal making any interested potential teams or partnerships extremely attractive. If you have access to more abundant computation or funding resources, please reach out to the author if interested in co-publishing to expand on this work or even other new topics in the field (check website for more on authors interests, but if it's an interesting concept it's something that would definitely be considered!), it would be extremely appreciated and fruitful.

7 Further Research

For further research, there are many ways to help expand or improve these results such as the following: Improving and optimizing the transformation function and encoding techniques. Seeing if better results are achievable with BPE tokenization of integers (without memorization). Trying to work with parentheses to augment a form of order of operations that would be more rigid. Expanding work to more operators such as exponentiation, division, functions, such as explicitly defined ones or more commonplace ones such as trigonometric functions. Crutching- Try this function as a crutch to help models improve quickly and remove the function, where it will only process the raw data. Expand these results to higher-level mathematics, such as algebra, geometry, calculus, and linear algebra. Finding a way to apply this automatically or find a model that can self detect and self apply this function or similar functions. Testing to see potential improvements in large language reasoning models.

Acknowledgments

Thank you to Andrej Karpathy for public lessons, providing the teachings to make the base for the transformer in this paper. Manning Publications and Sebastian Raschka for making great books that improved and started my LLM research. Grant Sanderson with 3blue1brown for helping me get to the point where I can understand the underlying mathematics in a deeper manner, as well as the neural networks series. Thanks to Anil Ananthaswamy for writing books that helped my conceptual very lower level understanding of Neural Networks.

References

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.