Training A Language Model (GPT) From Scratch And Harnessing It

Proposal

Manuel Sandoval Flores and Jeremy Herbst

October 1, 2024

Abstract

In this paper, we investigate the limitations of standard transformers, particularly GPTs, in addressing compositional tasks, with a focus on multi-digit multiplication. Our exploration centers around the training of *nanoGPT* [4], a scaled-down version of the GPT transformer designed for efficient training from scratch. We delve into the intricacies of transformer learning, emphasizing the challenges in handling carry-over and complex relationships between digits. Our digit-wise analysis reveals proficient predictions for the first and last digits, but difficulties in generalizing to longer sequences. Despite achieving strong overall accuracy, our models exhibit limitations in comprehensive multi-hop reasoning. The study highlights the need for further research on alternative model architectures, data augmentation techniques, attention mechanism analysis, and hyperparameter tuning to enhance the capabilities of transformers in compositional tasks.

1 Introduction

Transformer models have revolutionized sequence-to-sequence tasks, offering unparalleled capabilities in capturing long-range dependencies and providing a holistic view of input sequences through self-attention mechanisms. However, their effectiveness in addressing compositional tasks, particularly multi-digit multiplication, has raised significant challenges and limitations [2]. We delve into the intricacies of transformer learning, focusing on the development and training of *nanoGPT* [4], a scaled-down version of the GPT transformer designed for efficient training from scratch. The primary objective is to critically analyze the strengths and weaknesses of transformer models in tackling compositional multiplication tasks.

2 Background and Model Description

Transformer models have taken over in almost all areas. This is due to their unrivaled capabilities using the attention mechanism. These models, exemplified by *nanoGPT*[4], have shown adaptability in various applications, including natural language processing and image generation [8].

For compositional tasks such as multi-digit multiplication, transformer architectures, particularly the decoder part, play a crucial role in processing input sequences and generating output sequences through attention mechanisms and feed-forward neural networks. The tokenization process involves breaking input sequences into tokens for embedding and processing through the model. Notably, transformers exhibit remarkable reasoning abilities, leveraging self-attention to capture contextual information across the entire sequence, allowing for effective context-aware feature extraction.

Our models are based on *nanoGPT* [4] and only differ in model parameters such as *amount of layers*, *amount of heads*.

2.1 Positional encoding

Positional encodings are used to deal with the order of sequences. Because the transformer model doesn't have a built-in way to get information about the position of any given token, an encoding of position is essential.

Absolute positional encoding

This encoding is used in the initial transformer paper [8]. It keeps information about the absolute position of a given token t. If t appears in position 5 of a sequence, the transformer will know this token is in position 5. The encoding uses a sine function for its encoding schema.

Relative positional encoding

We focus on the encoding first presented by [6]. Using this encoding enables the transformer model to also learn information about the relative position of a given token. This means the model now knows that a token t in position 5 appears 2 positions before a token t_1 in position 3. Each token has two distinct edges for a connection to a different token. Therefore, there are $2k + 1$ unique edges, where k is the maximum distance that is chosen as a hyperparameter.

3 Related Work

Prior research has explored the challenges of generalization and grokking in transformers, emphasizing the limitations of standard transformers in addressing compositional tasks, particularly multi-digit multiplication [2]. These studies critically evaluate the successes and failures of transformers in compositional reasoning tasks, highlighting the difficulties encountered in effectively combining multiple steps for complex problems. It additionally delves into the implications of transformer learning in the context of compositional tasks, shedding light on the nuances of their application in educational settings.

Regarding multi-digit multiplications, some research has been done to show that transformers struggle to generalize and learn the multiplication, especially when sequences get longer. GPT-4 struggles when the length of multiplication numbers exceed 4. 4 by 4 multiplication only reached an accuracy of 23% [2]. A standard transformer model has been trained to solve multiple different arithmetic operations like multiplication and addition [9]. They reached an accuracy of 73% for a transformer trained on all operations. They also analyzed the accuracy for the different types of operations. Here they found that asymmetric multiplications only got an accuracy of 9%. However, they used a full transformer model, including the encoder part.

To improve generalization in transformers, it has been shown that relative position encodings (e.g.[6]) or universal transformers [1] improve the performance in the encoder part of the transformer for unseen sequence lengths [3]. However, this was only shown for decoders and while fixing the second number of the multiplications to 3 digits. They also showed a technique called *"priming"* that improved accuracy for longer sequences. This technique works similar to fine-tuning a model, however the extra samples are included while training the model and can be reduced in size by quite a lot in comparison to fine-tuning.

A graph based algorithm has been developed that when learned by a transformer improves accuracy a ton when compared to the state-of-the-art transformers (e.g. GPT, Bart etc.) [7].

4 Implementation

We trained *nanoGPT* with question answer pairs that have the following schema:

> What is 312 times 120?
>
> 37440

We tested our results with various model sizes, for the remains of this paper only the results of the biggest model for each positional encoding strategy are shown.

4.1 Training *nanoGPT* from Scratch

The approach we have chosen in our attempt to replicate the presented scenario is training *nanoGPT* [4]. Instead of using a pre-trained model, we train from scratch to understand the implications and stages of the model learning phases.

We used two different models with the following parameters:

Batch size	Block size	Number of layers	Number of attention heads	Initial learning rate
64	256	6	6	1e-3

Table 1: Parameters used for model training

Our two models only differ in their positional encodings. One model used *absolute positional encoding* [8], while the other used *relative positional encoding* [6]. For both, the learning rate is decayed by a cosine decay function.

The hyperparameters shown in table 1 are recommended by Andrej Karpathy when handling model training from scratch and training on a single GPU for a moderate amount of time. Based on Karpathys recommendations, we have also trained the model up to 100,000 iterations, evaluating every 250 iterations.

The environment we used to train our model is in *Google Colab*, with the following GPU specifications: Tesla T4 card containing 40 SMs with a 6MB L2 cache shared by all SMs. More details of this GPU can be found here [5].

4.2 Data

We have generated a list of numbers that include all possible values of 3 times 3 down to 3 times 1 multiplication.

The list has then been shuffled, and 90% of the generated question-answer pairs are written to the training and evaluation files, while the remaining 10% are written to a test file which is used later to evaluate the sampling on unseen data with the same length.

We also generated 10,000 random numbers that are of size 3 times 4 to check if our model can generalize to longer sequence lengths.

5 Results And Discussion

5.1 Training Results, Loss and Accuracy

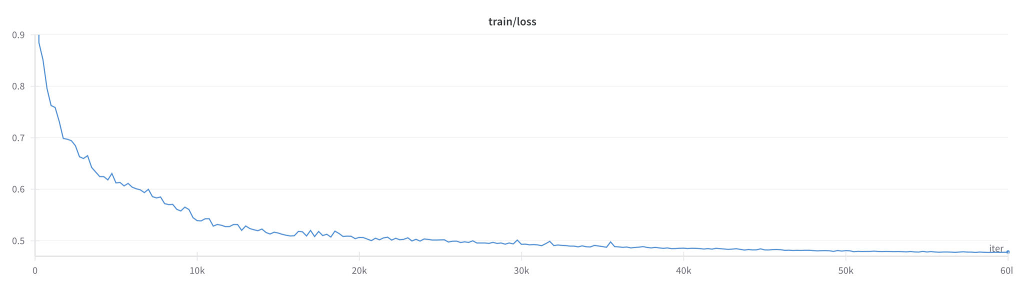

Figure 1: *nanoGPT* Training Loss per Iteration for the model with absolute positional encoding until 80,000 iterations.

In the case of the training of the model with absolute positional encoding, approximately after the iteration training 38, 000 the training loss went below 0.49 whereas the model with relative positional encoding achieved this loss value by approximately the iteration 29,000. In both cases, it stayed between 0.49 and 0.45 until the end of the training. This showcases how the relative positional encoding contributed to a faster capability of learning from the data.

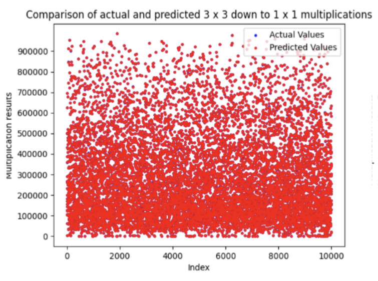

Figure 2

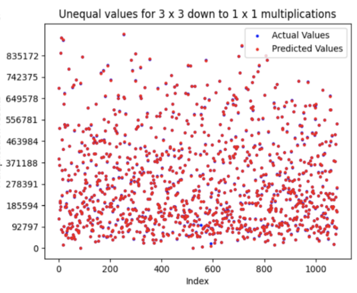

Figure 3

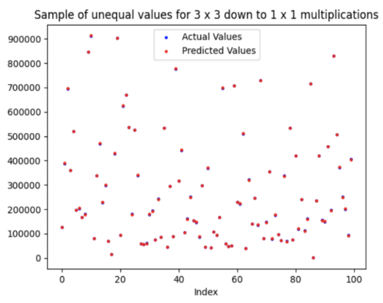

Figure 4

Figure 2: Multiplications from train data scatter-plotted, with the sampled value for all multiplication combinations (from 3 by 3 digits down to 1 by 1 digit). Figure 3: Multiplication for which the prediction was unsuccessful, compared to the expected values for all multiplication combinations (from 3 by 3 digits down to 1 by 1 digit). Figure 4: A random 100-item subset of Figure 3 for a less noisy visualization.

What Figure 3 and Figure 4 show is that even for wrong predictions, the data points are most of the time very close together. That means the transformer only struggles to predict the last digits for a given multiplication. This can be seen as an indication that the length of multiplication numbers will have an impact on the performance of the transformer. One can derive from that, that the model is not learning the multiplications rather it memorizes the numbers themselves

We present two accuracy metrics designed to analyze the performance of *nanoGPT* on four times three-digit multiplications. These metrics aim to provide a more granular understanding of the model's predictions and identify areas of improvement.

Digit Position Accuracy

This metric measures the accuracy of each digit's position in the predicted multiplication. High accuracy for lower digits indicates success in capturing basic arithmetic, while lower accuracy for higher digits may suggest challenges in handling carry-over or more complex relationships.

Accuracy Distance

Accuracy distance analyzes the difference between predicted and actual values. This metric provides insights into the model's precision and potential biases.

Performance Results

For the training data, an accuracy of 97.87% was reached while during the test on never-seen data, it reached 94.09%. In the tables below, we break down the accuracy by digits:

	Training Data		Evaluation Data	
Encodings	Absolute	Relative	Absolute	Relative
Accuracy	93.10%	97.87%	90.34%	94.09%

Table 2: Overall Accuracy

Digit Position	Training Data		Evaluation Data	
	Absolute	Relative	Absolute	Relative
1	99.87%	99.78%	99.74%	99.45%
2	98.27%	98.59%	97.50%	97.00%
3	93.87%	98.55%	91.70%	95.63%
4	99.79%	99.10%	99.73%	97.00%
5	99.97%	99.97%	99.98%	99.79%
6	99.26%	99.25%	98.66%	98.12%

Table 3: Per digit accuracy

Both of our models were trained with 100k iterations.

By narrowing down the issue to multiplications up to 3 digits by 3 digits, we have been able to replicate a fair representation of what the issue originally stated. Our accuracy results during training (93-97%) are even higher than the accuracy achieved in the original paper (Figure 3 in [2]) for the same type of multiplications, but rather similar to the test evaluation on never-seen data by comparing our result of 90-94% with results of.

When trying to generalize to longer sequences like 4 x 3 digits, multiplications that have never been seen by the model, the overall accuracy drops to 0.0% broken down by digit as follows:

Table 4: Evaluation Data for Each Digit Position

Digit Position	Absolute	Relative
1	99.60%	56.67%
2	95.33%	15.83%
3	66.81%	09.60%
4	16.66%	10.82%
5	18.63%	13.09%
6	16.76%	12.53%
7	00.00%	00.00%

Regarding the accuracy distance in the sampled results, when training the model with absolute positional encoding the most common distance found was 1000 and only for the results of 5 and 6 digits length. The model has mostly difficulties in predicting the third digit for 6-digit results and the second digit for the 5-digit results. This means the model did not have trouble memorizing the first and the last three digits, but the ones in between these two were the main challenge for it. Interestingly, this pattern was not found in the model trained with relative positional encoding, which suggests that this modification helped the model in its memorization.

In the attempt to generalize to 4 x 3 digits multiplication, no patterns in accuracy distances were found, suggesting that generalization failures were mostly random. However, the accuracy shows that the model with absolute positional encoding learns the first couple of digits of the multiplication and fails to start from the fourth digit. On the other hand, the relative positional encoding fails almost for all digit positions, this can be explained by the model trying to learn all digit positions on their own as shown in [3].

6 Conclusion

In this paper, we have undertaken the task of training a language model, specifically *nanoGPT* [4] from scratch to explore its capabilities in addressing compositional tasks, with a focus on multi-digit multiplication. Our implementation involved two models, one using absolute positional encoding and the other using relative positional encoding. The training process included question-answer pairs for multiplication problems of varying lengths, up to 3 digits by 3 digits.

Based on our performance results, we saw how the models demonstrated proficiency in predicting the first and last digits of the multiplication, and in the case of the relative positional encoding, it achieved the maximum loss faster than the absolute positional encoding, but challenges arose in accurately predicting the digits in between. Despite achieving strong overall accuracy, the models struggled to generalize to longer sequences, such as 4 x 3 digit multiplications.

Through digit-wise analysis, we identified areas of improvement, particularly in handling carry-over and more complex relationships between digits. The accuracy distance metrics revealed challenges in predicting certain digits, highlighting the model's limitations in comprehensive multi-hop reasoning.

Our attempt to replicate the scenario presented in a previous paper succeeded in showcasing the challenges of generalization in compositional tasks.

In conclusion, while our trained models demonstrated competency in certain aspects of multi-digit multiplication, the inherent limitations of transformer models in handling compositional tasks, especially in generalizing to unseen data, remain evident.

Future Work

Among our recommendations for future work are the following:

Model Architectures: Investigate alternative model architectures or modifications that may improve the handling of compositional tasks. This could involve experimenting with novel attention mechanisms or layer configurations. An option would be to test out the same experiments with Universal transformers [1].

Data Augmentation: Explore techniques for data augmentation to diversify the training dataset. This could involve generating additional examples or introducing variations in the existing dataset to enhance the model's ability to generalize. Specifically, *"priming"* [3] can be tested out.

Attention Mechanism Analysis: Conduct a detailed analysis of the attention mechanisms within the model during multi-digit multiplication tasks. Understanding how attention is distributed across input sequences may provide insights into areas that require improvement.

References

[1] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[2] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality, 2023.

[3] Samy Jelassi, Stéphane d'Ascoli, Carles Domingo-Enrich, Yuhuai Wu, Yuanzhi Li, and François Charton. Length generalization in arithmetic transformers, 2023.

[4] Andrej Karpathy. Nanogpt - the simplest, fastest repository for training/finetuning medium-sized gpts., 2022.

[5] Google Research. Colab gpu, Unkown.

[6] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[7] Turker Tuncer, Sengul Dogan, Mehmet Baygin, Prabal Datta Barua, Abdul Hafeez-Baig, Ru-San Tan, Subrata Chakraborty, and U Rajendra Acharya. Solving the multiplication problem of a large language model system using a graph-based method. *arXiv preprint arXiv:2310.13016*, 2023.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[9] Artit Wangperawong. Attending to mathematical language with transformers. *arXiv preprint arXiv:1812.02825*, 2018.