Technical Report

Anna Bogdanova

May 2023

1 Introduction

This technical report outlines the development of a language model (less than 4B parameters in size) capable of performing addition operations on long numbers of varying sizes. The approach is based on the findings from the article [4], which demonstrated that language models can teach themselves to use external tools to solve complex tasks, including arithmetic operations. The model described in this report has been fine-tuned using an automatically generated dataset which includes various prompts written in natural language, describing addition operations. The report includes a detailed description of the model architecture, training process and evaluation results, as well as recommendations for potential improvements of the method.

2 Related Work

Large language models (LLMs) have proved their abilities in many NLP tasks. However, their performance in certain domains, such as mathematics, can be unpredictable. For example, recent work [2] reveals that ChatGPT's capacity to solve mathematical problems is inconsistent: in many cases, the model fails at basic arithmetic tasks and in many cases it easily performs complicated ones.

To improve performance on mathematical tasks, fine-tuning language models on specific data can be highly effective. For example, Google's Minerva, a large language model trained on both general natural language and technical datasets [1], has achieved impressive results on various quantitative reasoning benchmarks, including 80% accuracy on 10-digit addition and over 20% accuracy on 18-digit addition.

A novel approach to enhancing language models performance on mathematical tasks was recently introduced by a Microsoft Research team [3]. Their method, called MathPrompter, utilizes chain of-thought prompting techniques to increase model's confidence in predictions. The model is presented with multiple math prompts, and the solutions are evaluated with random values, and a statistical significance test is then applied to determine a final solution. A 540B model with MathPrompter demonstrates state-of-the-art results on MultiArith dataset.

Another method for improving LLM performance on mathematical tasks was presented in [4]. The Toolformer language model learns, in a self-supervised manner, how to use external tools such as calculators through simple API calls. This is achieved by fine-tuning the language model on a dataset that includes sampled API calls. These findings were used in this work to fine-tune a language model that is capable of performing addition operation.

3 Method

The method described in [4] implies that the model can utilize tools via API calls. API inputs and outputs are represented by text sequences and described by the (a_c, i_c), where a_c is the name of the API and i_c is the input. The result of calling the API is r. To decorate such calls, dictionary tokens " [", "]" and " ->" are used, so that the line with the call looks like " [$a_c(i_c)$]", and the line with the result looks like " [$a_c(i_c)$ -> r] ".

The article explains how to create a dataset that includes API calls. To generate a dataset that is relevant for the addition operation tool, we can use simple prompts in natural language and automatically augment them with the dictionary tokens mentioned earlier. The input numbers are randomly generated from a range of $1 * 10^9$ to $1 * 10^{19}$. The Appendix A provides a comprehensive list of prompts used to generate the dataset. An example entry in the dataset follows the format: "The sum of 35251682569828301 and 6553723329634353846 is [Sum(35251682569828301,6553723329634353846) -> 6588975012204182147]". The train dataset consisting of 5000 entries was collected in this way.

The next step involved fine-tuning a language model. We selected the smallest version of the GPT-2 model, which has 124M parameters[1], and fine-tuned it using the dataset we generated. During fine-tuning, we used a batch size of 20, a learning rate of $2 * 10^{-5}$ with linear warmup for the first 10% of training steps. We set the maximum sequence length to 63, which is the maximum length of tokenized entries in the dataset. The model was trained for 150 epochs using the free GPUs provided by Google Colab.

The inference step involves generating text using the fine-tuned model with regular greedy decoding until the token "[" indicating the beginning of the API call block is produced. Tokens produced after the token "[" and before the token "->" are passed to the external addition operator tool (a basic Python script returning the sum of two integers). The token produced after the token "->" is replaced with the result of the API call, and the decoding process continues.

4 Results

We evaluated the fine-tuned GPT-2 model using the same prompts that were used to generate the dataset. During the evaluation process, only the first part of the prompts (without API call blocks) was used. The prompts used in the

[1] https://huggingface.co/gpt2

Prompt	**Acc ***	**Acc ****
1. The sum of <NUM1> and <NUM2> is	0.84	0.92
2. Add <NUM1> to <NUM2>. Answer is	0.9	0.9
3. Sum <NUM1> and <NUM2>. Answer is	0.57	0.17
4. The output of addition of <NUM1> and <NUM2> is	0.91	0.93
5. How much is <NUM1> + <NUM2>? Answer is	0.89	0.96
6. The result if addition of <NUM1> and <NUM2> is	0.89	0.87
7. Increase <NUM1> by <NUM2>. Answer is	0.84	0.9
8. <NUM1> + <NUM2> is	0.03	0.04
9. <NUM1> + <NUM2> =	0.05	0.04
10. <NUM1> plus <NUM2> is	0.04	0.02

Table 1: Accuracy achieved by the fine-tuned GPT-2 model on different prompts during inference (* on random numbers from the range of $1 * 10^9$ to $1 * 10^{19}$, ** on random numbers from the range of 1 to $1 * 10^9$)

evaluation stage can be found in Appendix B. The placeholders <NUM1> and <NUM2> in the prompts were replaced with randomly generated numbers from the range of $1 * 10^9$ to $1 * 10^{19}$, and the resulting prompts were passed to the model to generate an answer. The first number in the generated answer was assumed to be the model's predicted sum and was compared to the correct sum. The overall accuracy score on 2000 pairs of numbers (200 pairs per prompt) was 0.52.

We noted that the accuracy score varies depending on the prompt used (see Table 1). The highest score of 0.91 was achieved with the longest prompt, "The output of addition of <NUM1> and <NUM2> is" while the lowest scores were achieved with the shortest prompts, "<NUM1> + <NUM2> is", "<NUM1> plus <NUM2> is" and "<NUM1> + <NUM2> =" – 0.03, 0.04, and 0.05, respectively. The significant variation in scores indicates that the model is highly sensitive to the input query. The highest-scoring prompt was selected as the default query for evaluating the model's performance on two randomly generated numbers.

Analysis of the mistakes made by the model shows that the most common errors are: repeating numbers multiple times, missing parts of the numbers, or replacing them with other numbers when generating them inside API call blocks. We believe that these errors can be corrected by better controlling the generation process, for example, by selecting the API call beginning token from the k tokens with the highest probability, as described in the article, or by using more advanced strategies such as top-p and top-k sampling.

We also evaluated the performance of the model on smaller numbers. Table 1 shows the accuracy scores for random numbers ranging from 1 to $1 * 10^9$. While some prompts showed improvement in metrics, others showed a decrease in accuracy from 0.57 to 0.17. The accuracy metrics for the shortest prompts remained the same.

5 Conclusion

The Toolformer approach appears to be a promising method for improving the performance of language models on arithmetical tasks. However, to thoroughly evaluate its effectiveness, it is important to consider baselines, analyze various decoding strategies, test the method with larger models, and examine common mistakes made by the models. These steps can help provide a more comprehensive understanding of the capabilities and limitations of the approach.

References

[1] Aitor Lewkowycz et al. *Solving Quantitative Reasoning Problems with Language Models.* 2022. arXiv: `2206.14858 [cs.CL]`.

[2] Simon Frieder et al. *Mathematical Capabilities of ChatGPT.* 2023. arXiv: `2301.13867 [cs.LG]`.

[3] Shima Imani, Liang Du, and Harsh Shrivastava. *MathPrompter: Mathematical Reasoning using Large Language Models.* 2023. arXiv: `2303.05398 [cs.CL]`.

[4] Timo Schick et al. *Toolformer: Language Models Can Teach Themselves to Use Tools.* 2023. arXiv: `2302.04761 [cs.CL]`.

Appendices

A Prompts used in generating dataset

1. The sum of <NUM1> and <NUM2> is [Sum(<NUM1>,<NUM2>) -> <SUM>]
2. Add <NUM1> to <NUM2>. Answer is [Sum(<NUM1>,<NUM2>) -> <SUM>]
3. Sum <NUM1> and <NUM2>. Answer is [Sum(<NUM1>,<NUM2>) -> <SUM>
4. The output of addition of <NUM1> and <NUM2> is [Sum(<NUM1>,<NUM2>) -> <SUM>]
5. How much is <NUM1> + <NUM2>? Answer is [Sum(<NUM1>,<NUM2>) -> <SUM>]
6. The result if addition of <NUM1> and <NUM2> is [Sum(<NUM1>,<NUM2>) -> <SUM>]
7. Increase <NUM1> by <NUM2>. Answer is [Sum(<NUM1>,<NUM2>) -> <SUM>]
8. <NUM1> + <NUM2> is [Sum(<NUM1>,<NUM2>) -> <SUM>]
9. <NUM1> + <NUM2> = [Sum(<NUM1>,<NUM2>) -> <SUM>]
10. <NUM1> plus <NUM2> is [Sum(<NUM1>,<NUM2>) -> <SUM>]

B Prompts used in model inference

1. The sum of <NUM1> and <NUM2> is
2. Add <NUM1> to <NUM2>. Answer is
3. Sum <NUM1> and <NUM2>. Answer is
4. The output of addition of <NUM1> and <NUM2> is
5. How much is <NUM1> + <NUM2>? Answer is
6. The result if addition of <NUM1> and <NUM2> is
7. Increase <NUM1> by <NUM2>. Answer is
8. <NUM1> + <NUM2> is
9. <NUM1> + <NUM2> =
10. <NUM1> plus <NUM2> is