Predicting the Sum of Long Numbers with Language Models

Abstract

In this report I will describe the method I used to adapt a language model that can solve addition problems for long numbers, and evaluate its performance on a randomly generated dataset with numbers of varying lengths. The deliverables include the code, a technical report describing the methodology and data used, a quality assessment, and a literature review on the topic.

1. Introduction

The field of natural language processing (NLP) has experienced rapid progress in recent years, thanks to advances in deep learning. Language models have been shown to be effective in a wide range of NLP tasks, such as machine translation or question answering. Nevertheless, solving even simple arithmetic problems like addition or substraction has been a long-standing challenge for neural networks due to the lack of underlying logic for arithmetic computations.

In recent years, pre-trained transformer models such as T5 have emerged as a powerful tool for natural language processing tasks. These models can be fine-tuned on specific tasks, allowing them to adapt to new domains and achieve impressive performance with limited additional training. So, to solve the addition problem for long numbers I will take advantage of the effectiveness of fine-tuning a pre-trained T5 model. To evaluate the performance of our fine-tuned T5 model, I measure its accuracy on a randomly generated set of numbers of varying lengths.

The report is structured as follows: In Section 2, I provide a brief review of the literature on the use of methods for solving mathematical problems. In Section 3, I describe the methodology used for fine-tuning the T5 model for the addition task. In Section 4, I present the results of our experiments and conclude the report.

2. Literature review

One popular approach for solving addition tasks with neural networks is the use of sequence-to-sequence learning (Sutskever et al., 2014). These models consist of an encoder and a decoder, which are typically implemented as recurrent neural networks (RNNs), particularly LSTM models. The encoder takes in the input expression and converts it into a vector, which is then passed to the decoder to generate the output expression. Several studies have shown the effectiveness of sequence-to-sequence models in solving mathematical problems, including addition (Lample & Charton, 2019).

Zaremba & Sutskever, 2014 show that it is possible to train LSTM in the sequence-to-sequence regime using curriculum learning and teaching forcing to add two 9-digit numbers. Kalchbrenner et al., 2015 propose Grid Long Short-Term Memory, a network of LSTM cells arranged in a multidimensional grid, which is able to add 15-digit integer numbers. Similarly, Kaiser & Sutskever, 2016 present Neural GPU, a neural network architecture based on a type of convolutional gated recurrent units, which is able to solve algorithmic tasks such as long binary summations and multiplications. Chen et al., 2018 present a solver for arithmetic expression calculation based on multi-level hierarchical reinforcement learning method.

Trask et al., 2018 proposed the Neural Arithmetic Logic Unit (NALU), which is a neural architecture that can represent the mathematical relationships by the units of the network to learn operations such as summation, subtraction or multiplication. Despite the enhancements made to the NALU (iNALU by Schlor et al., 2020), training instability still persists, necessitating random reinitialization.

More recently, pre-trained transformer models, such as BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020), has achieved state-of-the-art performance on a wide range of NLP benchmarks. Also GPT-3 is able to carry out basic arithmetic tasks such as two-digit addition without any additional fine tuning (Brown et al., 2020). Nevertheless, these models can be fine-tuned on specific tasks to achieve better results. Several studies have demonstrated the effectiveness of fine-tuning pre-trained transformer models for mathematical problem-solving tasks, including addition.

In this task, I will rely on the solution presented by Nogueira et al., 2021. In the paper they introduced position tokens to make pretrained T5 learns to accurately add and subtract numbers up to 60 digits.

3. Methodology

Our task is the addition of two long numbers. We define it as sequence-to-sequence tasks in which both inputs to the models and target outputs are treated as sequences of tokens. An example input is “23, 2”, and the output is ”25”. Also we will generate data and train the model as Nogueira et al., 2021 did in their paper. The code was mainly taken from paper’s repository.

3.1 Metric

We use accuracy to evaluate model. The answer is correct if it matches the target output exactly.

3.2 Model

We will use a pretrained sequence-to-sequence model T5-base (T5 with 220M parameters). We feed the input as sequence of tokens to the model and train it to generate the answer, token by token.

3.3 Data generation

To generate data, we set the maximum number of digits D, choose d from $[2; D]$ and create two numbers from $[10^{d-1}, 10^d - 1]$, and compute the answer as sum of this numbers. With this method we have equal proportion of d-digit numbers.

3.4 Tokenization

We would use "10e-based" representation as it achieved the best accuracy in the experiments conducted in the paper in comparison with decimal (example: 107), character (1 0 7), fixed-character (0 1 0 7 in case all numbers have maximum 4 digits), underscore (1_0_7), words (one hundred seven) and 10-based (1 100 0 10 7 1, when digits are separated by powers of 10) representations. In "10e-based" representation digits are separated by powers of 10 using scientific notation. For example, 107 is presented like 1 10e2 0 10e1 7 10e0. It is more compact than using "10-based" representation and allows model to determine digit significance and ensures consistent tokenization.

3.5 Training and evaluating

We train the models using the AdamW optimizer (Loshchilov & Hutter, 2018), batches of 128 examples, and a learning rate of 0.0003. Training set consists of 20.000 examples. The model is trained on up to 50-digits numbers and is tested on dataset of 1000 examples with up to 60-digits numbers. To avoid overfitting, we check validation accuracy after every epoch.

We set 10 epochs and save checkpoint after every epoch. The best checkpoint is chosen using a validation set of 10,000 examples. The model is evaluated on test set of 10,000 examples.

3.6 How to use

All code is available here. To train the model on the task of two 50-digits numbers addition, you need to install requirements by

```
pip install -r requirements.txt
```

and run the command

```
python train.py \
--output_dir=. \
--model_name_or_path=t5-base \
--train_size=20000 \
--val_size=1000 \
--min_digits_train=2 \
--max_digits_train=50 \
--seed=1 \
--train_batch_size=4 \
--accumulate_grad_batches=32 \
```

```
--val_batch_size=32 \
--max_seq_length=512 \
--num_workers=4 \
--gpus=1 \
--optimizer=AdamW \
--lr=3e-4 \
--weight_decay=5e-5 \
--scheduler=StepLR \
--t_0=2 \
--t_mult=2 \
--gamma=1.0 \
--step_size=1000 \
--max_epochs=10 \
--step_size=1000 \
--max_epochs=10 \
--gradient_clip_val=1.0
```

The best accuracy on validation set was observed after the fourth epoch and was 0.996. The correspoing checkpoint is available here. After fourth epoch the accuracy on validation set started to decrease, and we stopped training.

The model was tested on datasets of 1000 examples of diffent length (up to 50, 60, 70, 80, 90 digits in number) to check its extrapolation quality. To test the model with best checkpoint, you need to run this command:

```
python test.py \
--output_dir=. \
--checkpoint_name='epoch=4-val_exact_match=0.9960.ckpt' \
--seed=1 \
--test_size=1000 \
--num_workers=4 \
--min_digits_test=2 \
--max_digits_test=50 \
--gpus=1
```

To predict the sum of tho numbers, you can use this command:

```
python predict.py \
--output_dir=. \
--first_number=4 \
--second_number=5 \
--checkpoint_name='epoch=4-val_exact_match=0.9960.ckpt' \
--gpus=1
```

The answer will be printed and saved to the file "prediction.txt".

4. Results

We tested model on 1000-examples dataset with up to 50-, 60-, 70-, 80-, 90-digits numbers. A graph below illustrates dependence between numbers' length and accuracy.

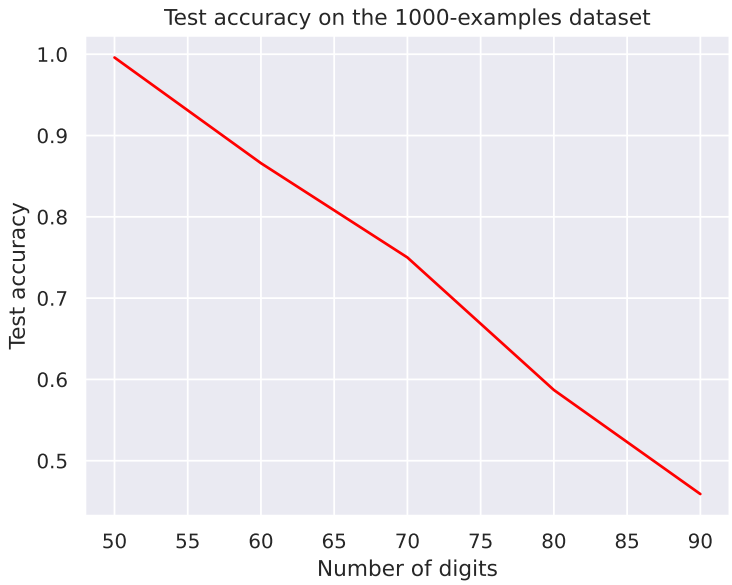

Figure 1: Test accuracy depending on the maximum length of numbers in dataset on the addition task

When we try to predict sum of two 50-digits numbers, the accuracy is approximately 100%. But as the length of the numbers increases, the accuracy decreases. The model does not demonstrate good extrapolation abilities, but it can handle the addition of numbers containing up to 65 digits with an accuracy exceeding 80%. To improve the model, we can train it using longer numbers, but it takes more time and resources.

References

[1] Rodrigo Nogueira, Zhiying Jiang and Jimmy Lin. Investigating the Limitations of Transformers with Simple Arithmetic Tasks. In 1st Mathematical Reasoning in General Artificial Intelligence Workshop, ICLR 2021

[2] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Networks, 2018.

[3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In International Conference on Learning Representations, 2018

[4] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learn-

ing with a unified text-to-text transformer. Journal of Machine Learning Research, 21(140):1–67, 2020.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186, 2019.

[6] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. arXiv preprint arXiv:1511.08228, 2015

[7] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In International Conference on Learning Representations, 2019

[8] Wojciech Zaremba, Ilya Sutskever. Learning to execute. In ICLR 2015

[9] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. arXiv preprint arXiv:1507.01526, 2015

[10] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. arXiv preprint arXiv:1511.08228, 2015.

[11] Andrew Trask, Felix Hill, Scott E. Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In Advances in Neural Information Processing Systems, pp. 8035–8044, 2018

[12] D. Schlor, M. Ring, and A. Hotho. inalu: Improved neural arithmetic logic unit. Frontiers in Artificial Intelligence, 3, 2020.