Question Answering on Mathematics Dataset

Authors

Alessio Saladino, 2000727
saladino.2000727@studenti.uniroma1.it
Paolo Ferretti, 2039579
ferretti.2039579@studenti.uniroma1.it

Abstract

Transformer models represent a powerful evolution in the field of natural language processing. They are able to solve different types of tasks related to natural language: neural-machine translation, text-to-speech transformation, speech recognition and so on.

In this work, we implement the Tensor-Product Transformer (TP-Transformer), an architecture designed to address the challenge of solving mathematical problems expressed in natural language. By introducing the role vector in the multi-head attention mechanism, the Tensor-Product Transformer constructs an explicit representation of relations, allowing a more complex understanding of mathematical questions.

1 Introduction

In this work, we address the task of solving mathematical problems expressed in natural language. This task falls into the category of question answering, in which an answer is generated based on a question related to a corpus [8]. To address this task, we try to reproduce the work shown in [3], comparing the proposed implementation of the TP-Transformer with that of a classic transformer [7]. We first show a preview of the dataset in use, then we offer an overview of the structure of a classic transformer model, to then focus on the differences with respect to the TP-Transformer.
Finally, we report the results obtained by comparing the two different approaches.

2 Dataset

The dataset used in our work is the *Mathematics Dataset* [6]. It is organized into 56 modules, each of which represents a different mathematical area. For example, the ”Algebra” module contains problems related to linear algebra, equations, and functions. The ”Arithmetic” module contains problems related to addition, subtraction, multiplication, and division. And so on.
Each module comes in three different difficulty levels: easy, medium and hard, for a total of 2.01 million samples per module. Of these samples, 2 million are used for training and 10k samples are used for testing.

Question: `Solve -42*r + 27*c = -1167 and 130*r + 4*c = 372 for r.`
Answer: `4`
Question: `Calculate -841880142.544 + 411127.`
Answer: `-841469015.544`
Question: `Let x(g) = 9*g + 1. Let q(c) = 2*c + 1. Let f(i) = 3*i - 39. Let w(j) = q(x(j)). Calculate f(w(a)).`
Answer: `54*a - 30`
Question: `Let e(l) = l - 6. Is 2 a factor of both e(9) and 2?`
Answer: `False`
Question: `Let u(n) = -n**3 - n**2. Let e(c) = -2*c**3 + c. Let l(j) = -118*e(j) + 54*u(j). What is the derivative of l(a)?`
Answer: `546*a**2 - 108*a - 118`
Question: `Three letters picked without replacement from qqqkkklkqkkk. Give prob of sequence qql.`
Answer: `1/110`

Figure 1: An example of the dataset content

3 Transformer Architecture

As a first baseline for our work, we use the classic transformer architecture presented in [7]. In this section, we provide a recap of the various components of a transformer.
It is composed by two main modules: the encoder and the decoder. The encoder is designed to process an input sequence (a question) and to compress this information into a ”memory” tensor that the decoder can use. It is composed by multiple layers, each containing the following components:

- Self-Attention Layer
- Feed-Forward Neural Network
- Normalization
- Residual Connection + Layer Normalization

The decoder also has multiple layers and is responsible for generating the output sequence. It has more layers with respect to the encoder to include the memory information. Between each main block of the transformer, a residual connection and a layer normalization is applied [5] [4].

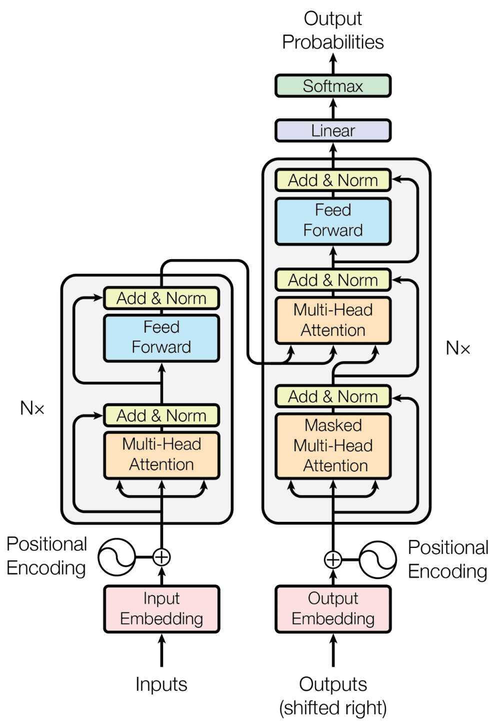

Figure 2: The Transformer Architecture

3.1 Tokenization and Embedding Layer

As first preliminary step, it is necessary to convert the inputs in a format that is understandable by the computer.[9]
To do so, we've first automatically built a vocabulary, by extracting the different characters contained in the dataset. Then, we've defined special characters *bos* and *eos* to respectively represent the beginning and the end of a sentence. The special character *pad* was finally used as a filler to allow all sentences in a batch to get the same length. The vocabulary contains 73 different tokens.
After having built the vocabulary, we've applied character-level tokenization, replacing each character in the sequence with their corresponding numerical index. The obtained tensor is fed to an embedding layer which produces an embedded representation of it and it is passed to the transformer.

3.2 Positional Encoding

Since we're dealing with a NLP task, the order of the characters in a sequence is important, as it is important to maintain the capability of generalizing the context independently of the length of the text [2]. The aim of positional encoding is to provide information about the absolute and relative position of the tokens. The positional encodings are added to the input embeddings at the bottom of the encoder-decoder stack.
The sine and cosine functions of different frequencies are used. The positional encoding for the transformer is computed as follows:
For position p and dimension i:

$$PE_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d}}\right) \tag{1}$$

$$PE_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d}}\right) \tag{2}$$

Where:

- $PE_{(p,2i)}$ is the $2i$-th dimension of the positional encoding for position p.
- $PE_{(p,2i+1)}$ is the $(2i+1)$-th dimension of the positional encoding for position p.
- d is the total dimensions of the model (the embedding size).

3.3 Multi-Head Attention

The attention mechanism is a fundamental building block of the transformer architecture and it is present in both the encoder an the decoder.
A self-attention layer computes attention scores for all word pairs in the input sequence and produces a new representations, capturing the relevant information. The scaled dot product attention function can be described as:

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{3}$$

Where:

- Q: Query matrix
- K: Key matrix
- V: Value matrix
- d_k: The dimensionality of the queries and keys, which is typically the embedding dimension divided by the number of attention heads.

In contrast, the cross-attention is used in the decoder and takes into account both the target sequence and the memory tensor from the encoder. Here, the memory tensor acts as key and value vectors.
To attend to different parts of the sequence, the attention mechanism runs in parallel multiple times and the results are concatenated. This process is known as Multi-Head Attention, and its mathematical formulation is given by:

$$\text{MH}(Q, K, V) = \text{Concat}(\text{H}_1, \text{H}_2, \ldots, \text{H}_n)W^O \tag{4}$$

Where:

$$\text{H}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{5}$$

3.4 Masking

The transformer architecture uses two types of masks: a mask for padding the sequence and a look ahead mask. The former is used to mask the position of the *pad* token in both the source and the target, since it must be ignored. The latter, is used only for the target, and is represented by a lower triangular matrix. Such mask is necessary to prevent the model to attend to future positions, since the target sentence is given as input to the decoder together with the source.

4 TP-Transformer

The goal of our work was to validate the model proposed in [3]. They propose an alternative attention mechanism based on a Tensor-Product Representation, to better support the explicit representation of relation structure.
In this variation of the classic Multi-Head Attention, they introduce the role vector *r* and the operation **values = values * r**, in which the attention values are multiplied element-wise with the role vector. Every head produces the query, key, value and role vectors. The filler in each attention head is determined by taking the weighted sum of all corresponding values within the same layer and head. This filler is then combined with the role vector. Subsequently, an affine transformation is performed, and the transformed bindings from all heads are summed together.

5 Training

Since we had limited memory and time resources, to train the model we reduced the size of the transformer components in order to adapt it to our hardware availability. We use a batch size of 256, an embedding and hidden size of 512, 4 layers and 8 heads. The maximum sequence length is set to 170 characters. The obtained model has approximately 20 millions of trainable parameters.

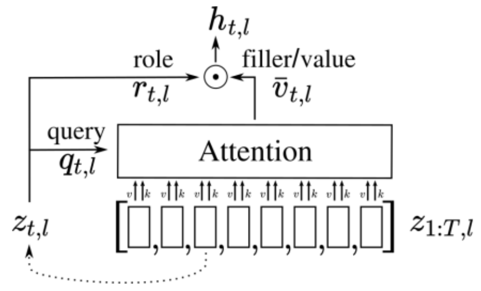

Figure 3: Tensor Product Attention

Moreover, due to constraints on computational resources and time, we were unable to replicate the general model training conducted in the original paper, which utilized a dataset composed with all the modules.
Therefore, we opted for two simpler approaches: training distinct models on individual modules and training a single model on a subset of modules. In order to do this, we selected 10 modules from the original dataset. These were chosen according to the performances reported in the original [3] paper. In particular, we tried to include the ones where the original TP-Transformer performs well, those where it achieved average performance, and those where the performance were worse.
For each module, we used 500,000 samples from the easy, medium, and hard levels. This gave us a total of 1.5 million training samples for each module.
We employed Adam as optimizer with learning rate set to 1e-4. The rate of convergence and the overall training time varied depending on the complexity of the module. Some modules reached convergence easily, while others required more time or failed to converge.

6 Results

We trained and tested the TP-Transformer and the classic Transformer architecture on single modules. Subsequently, we trained the TP-Transformer on a mixed dataset made up of the 10 selected modules, and then went on to test the performance on each individual module.
During the test phase, the answer to a question

is generated using a greedy generation method. Given the current sequence of characters (target), the best successor is selected according to the probabilities obtained by applying the SoftMax [1] function over the logits.
To evaluate model performance we used accuracy as metric: given the original answer and the generated one, the latter is considered correct if and only if it exactly matches the former. The following table shows the accuracy values achieved by the single-module transformers and by the mixed-module transformer.

	TP-Transformer Accuracy	Transformer Accuracy	Mixed TP-Accuracy
numbers–place-value	1	1	1
numbers–round-number	0.99	0.99	0.96
comparison–sort	0.97	0.97	0.92
arithmetic–add-or-sub	0.97	0.92	0.84
calculus-differentiate	0.93	0.85	0.83
comparison–pair	0.87	0.79	0.81
algebra–linear-1d	0.64	0.52	0.04
comparison–pair-composed	0.62	0.61	0.62
algebra–polynomial-roots	0.41	0.31	0.13
probability–swr-p-level-set	0.06	0.06	0.06

Table 1: Accuracy values obtained on the test set.

7 Conclusions

In this work, we focused on answering mathematical questions using natural language processing, trying to match the performance of the Tensor-Product Transformer on the Mathematics Dataset. Our tests confirmed that by introducing a role vector in the Multi-Head Attention mechanism of the original transformer architecture we were able to get slightly better results. When we trained one big model using multiple types of math problems, it did a good job but not as good as when we trained separate models for each type of problem. Still, the benefit of the big, multi-task model is that it can handle many different kinds of problems, making it more flexible.

Even though we only partially matched the results of the original paper that used 56 different modules, our work looks promising. In the future, this method could be useful for solving similar problems.

References

[1] John Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. 4

[2] Amirhossein Kazemnejad et Al. The impact of positional encoding on length generalization in transformers. 2023. 2

[3] Imanol Schlag et Al. Enhancing the transformer with explicit relational encoding for math problem solving. *ArXiv*, 2019. 1, 3

[4] Kaiming He et Al. Deep residual learning for image recognition. 2015. 2

[5] Geoffrey E. Hinton Jimmy Lei Ba, Jamie Ryan Kiros. Layer normalization. 2016. 2

[6] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *ArXiv*, abs/1904.01557, 2019. 1

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 1

[8] Zhen Wang. Modern question answering datasets and benchmarks: A survey. 2022. 1

[9] Jonathan J. Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 4*, COLING '92, page 1106–1110, USA, 1992. Association for Computational Linguistics. 2