Enhancing the Transformer with Explicit Relational Encoding for Math Problem Solving

Imanol Schlag*
The Swiss AI Lab IDSIA / USI / SUPSI
imanol@idsia.ch

Paul Smolensky
Microsoft Research, Redmond
Johns Hopkins University
psmo@microsoft.com

Roland Fernandez
Microsoft Research, Redmond
rfernand@microsoft.com

Nebojsa Jojic
Microsoft Research, Redmond
jojic@microsoft.com

Jürgen Schmidhuber
The Swiss AI Lab IDSIA / USI / SUPSI
juergen@idsia.ch

Jianfeng Gao
Microsoft Research, Redmond
jfgao@microsoft.com

Abstract

We incorporate Tensor-Product Representations within the Transformer in order to better support the explicit representation of relation structure. Our Tensor-Product Transformer (TP-Transformer) sets a new state of the art on the recently-introduced Mathematics Dataset containing 56 categories of free-form math word-problems. The essential component of the model is a novel attention mechanism, called TP-Attention, which explicitly encodes the relations between each Transformer cell and the other cells from which values have been retrieved by attention. TP-Attention goes beyond linear combination of retrieved values, strengthening representation-building and resolving ambiguities introduced by multiple layers of standard attention. An initial analysis indicates that the role representations learn task-relevant relational structure.

1 Introduction

In this paper we propose a variation of the Transformer that is designed to allow it to better incorporate structure into its representations. We test the proposal on a task where structured representations are expected to be particularly helpful: math word-problem solving. For this task, the model takes a free-form math question like *Let r(g) be the second derivative of 2*g**3/3 – 21*g**2/2 + 10*g. Let z be r(7). Factor –z*s + 6 – 9*s**2 + 0*s + 6*s**2.* and must produce the answer *–(s + 3)*(3*s – 2).* Our proposed model successfully predicts such answers.

We begin by viewing the Transformer (Vaswani et al., 2017) as a kind of Graph Neural Network. For concreteness, consider the encoder component of a Transformer with H heads. When the h^{th} head of a cell t of layer l issues a query and as a result concentrates its self-attention distribution on another cell t' in layer l, we can view these two cells as joined by an edge in an information-flow graph: the value vector at t' in effect passes via this edge to affect the activation state of t. The strength of this attention can be viewed as a weight on this edge, and the index h of the head can be viewed as a label on this edge. Thus, each layer of the Transformer can be viewed as a complete, directed, weighted,

*Work done while at Microsoft Research.

Workshop on Context and Compositionality in Biological and Artificial Neural Systems (NeurIPS 2019)

labeled graph. Prior NLP work has interpreted certain edges of these graphs in terms of linguistic relations (Sec. 5), and we wish to enrich the relation structure of these graphs to better support the explicit representation of relations within the Transformer.

Here we propose to replace each of the discrete edge labels $1, \ldots, H$, with a **relation vector**: we create a bona fide representational space for the relations being learned by the Transformer. This makes it possible for the hidden representation at each cell to approximate the vector embedding of a symbolic structure built from the relations generated by that cell. This embedding is a **Tensor-Product Representation (TPR)**. TPRs provide a general method for embedding symbol structures in vector spaces. TPRs support compositional processing by directly encoding constituent structure: the representation of a structure is the sum of the representation of its constituents. The representation of each constituent is built from two vectors: one vector that embeds the content of the constituent, the **'filler'** — here, the vector resulting from attention — and a second vector that embeds the structural **role** it fills — here, a relation conceptually labeling an edge of the attention graph. The vector that embeds a filler and the vector that embeds the role it fills are **bound** together by the tensor product to form the tensor that embeds the constituent that they define.[2] The relations here, and the structures they define, are learned unsupervised by the Transformer in service of a task: post-hoc analysis is then required to interpret those roles.

In the new model, the **TP-Transformer**, each head of each cell generates a key-, value- and query-vector, as in the Transformer, but additionally generates a **relation-vector**. The query is interpreted as seeking the appropriate filler for that role. Each head binds that filler to its role via the tensor product (or some contraction of it), and these filler/role bindings are summed to form the TPR of a structure with H constituents (details in Sec. 2.).

On the Mathematics Dataset (Sec. A), the new model sets a new state of the art for the overall accuracy, and for all the individual-problem-type module accuracies (Sec. 3). Initial results of interpreting the learned roles for the arithmetic-problem module show that they include a good approximation to the second-argument role of the division operator and that they distinguish between numbers in the numerator and denominator roles (Sec. 4).

2 The TP-Transformer

The TP-Transformer's encoder network, like the Transformer's encoder (Vaswani et al., 2017), can be described as a 2-dimensional lattice of cells (t, l) where $t = 1, ..., T$ are the sequence elements of the input and $l = 1, ..., L$ are the layer indices with $l = 0$ as the embedding layer. All cells share the same topology and the cells of the same layer share the same weights. More specifically, each cell consists of an initial layer normalization (LN) followed by a **TP-Multi-Head Attention (TPMHA)** sub-layer followed by a fully-connected feed-forward (FF) sub-layer. Each sub-layer is followed by layer normalization (LN) and by a residual connection (as in the original Transformer; Eq. 1). Our cell structure follows directly from the official TensorFlow source code by Vaswani et al. (2017) but with regular Multi-Head Attention replaced by our TPMHA layer.

The input into $\text{cell}_{t,l}$ is the output of $\text{cell}_{t,l-1}$ and doesn't depend on the state of any other cells of the same layer, which allows a layer's outputs to be computed in parallel.

$$\begin{aligned} \boldsymbol{h}_{t,l} &= \boldsymbol{z}_{t,l} + \text{TPMHA}(\text{LN}(\boldsymbol{z}_{t,l}), \text{LN}(\boldsymbol{z}_{1:T,l})) \\ \boldsymbol{z}_{t,l+1} &= \text{LN}(\boldsymbol{h}_{t,l} + \text{FF}(\text{LN}(\boldsymbol{h}_{t,l}))) \end{aligned} \tag{1}$$

We represent the symbols of the input string as one-hot vectors $\boldsymbol{x}_1, ..., \boldsymbol{x}_T \in \mathbb{R}^{d_v}$ where d_v is the size of the vocabulary and the respective columns of the matrix $\boldsymbol{E} \in \mathbb{R}^{d_z \times d_v}$ are the embedding vectors of those symbols. We also include a positional representation $\boldsymbol{p}_t$ using the same sinusoidal encoding schema introduced by Vaswani et al. (2017). The input of the first-layer $\text{cell}_{t,1}$ is $\boldsymbol{z}_{t,0}$:

$$\begin{aligned} \boldsymbol{e}_t &= \boldsymbol{E}\boldsymbol{x}_t\sqrt{d_z} + \boldsymbol{p}_t \\ \boldsymbol{r}_t &= \boldsymbol{W}^{(p)}\boldsymbol{e}_t + \boldsymbol{b}^{(p)} \\ \boldsymbol{z}_{t,0} &= \boldsymbol{e}_t \odot \boldsymbol{r}_t \end{aligned} \tag{2}$$

[2] The tensor product operation is what enables the sum of constituents representing the structure as a whole to be uniquely decomposable back into individual pairs of roles and their fillers, if necessary.

where $\boldsymbol{W}^{(p)} \in \mathbb{R}^{d_z \times d_z}$, $\boldsymbol{b}^{(p)} \in \mathbb{R}^{d_z}$, $\boldsymbol{r}_t$ is a position- and symbol-dependent role representation, and $\odot$ is elementwise multiplication (a contraction of the tensor product: see Sec. **??**).

2.1 TP-Multi-Head Attention

The TPMHA layer of the encoder consists of H heads that can be applied in parallel. Every head $h, 1 \leq h \leq H$ applies separate affine transformations $\boldsymbol{W}_l^{h,(k)}, \boldsymbol{W}_l^{h,(v)}, \boldsymbol{W}_l^{h,(q)}, \boldsymbol{W}_l^{h,(r)} \in \mathbb{R}^{d_k \times d_z}$, $\boldsymbol{b}_l^{h,(k)}, \boldsymbol{b}_l^{h,(v)}, \boldsymbol{b}_l^{h,(q)}, \boldsymbol{b}_l^{h,(r)} \in \mathbb{R}^{d_k}$ to produce key, value, query, and relation vectors from the hidden state $\boldsymbol{z}_{t,l}$, where $d_k = d_z/H$:

$$\begin{aligned} \boldsymbol{k}_{t,l}^h &= \boldsymbol{W}_l^{h,(k)} \boldsymbol{z}_{t,l} + \boldsymbol{b}_l^{h,(k)} & \boldsymbol{q}_{t,l}^h &= \boldsymbol{W}_l^{h,(q)} \boldsymbol{z}_{t,l} + \boldsymbol{b}_l^{h,(q)} \\ \boldsymbol{v}_{t,l}^h &= \boldsymbol{W}_l^{h,(v)} \boldsymbol{z}_{t,l} + \boldsymbol{b}_l^{h,(v)} & \boldsymbol{r}_{t,l}^h &= \boldsymbol{W}_l^{h,(r)} \boldsymbol{z}_{t,l} + \boldsymbol{b}_l^{h,(r)} \end{aligned} \tag{3}$$

The filler of the attention head t, l, h is

$$\bar{\boldsymbol{v}}_{t,l}^h = \sum_{i=1}^{T} \boldsymbol{v}_{i,l}^h \alpha_{t,l}^{h,i}, \tag{4}$$

i.e., a weighted sum of all T values of the same layer and attention head (see Fig. 1). Here $\alpha_{t,l}^{h,i} \in (0, 1)$ is a continuous *degree of match* given by the softmax of the dot product between the query vector at position t and the key vector at position i:

$$\alpha_{t,l}^{h,i} = \frac{\exp(\boldsymbol{q}_{t,l}^h \cdot \boldsymbol{k}_{i,l}^h \frac{1}{\sqrt{d_k}})}{\sum_{i'=1}^{T} \exp(\boldsymbol{q}_{t,l}^h \cdot \boldsymbol{k}_{i',l}^h \frac{1}{\sqrt{d_k}})} \tag{5}$$

The scale factor $\frac{1}{\sqrt{d_k}}$ can be motivated as a variance-reducing factor under the assumption that the elements of $\boldsymbol{q}_{t,l}^h$ and $\boldsymbol{k}_{t,l}^h$ are uncorrelated variables with mean 0 and variance 1, in order to initially keep the values of the softmax in a region with better gradients.

Finally, we bind the filler $\bar{\boldsymbol{v}}_{t,l}^h$ with our relation vector $\boldsymbol{r}_{t,l}^h$, followed by an affine transformation $\boldsymbol{W}_{h,l}^{(o)} \in \mathbb{R}^{d_z \times d_k}, \boldsymbol{b}_{h,l}^{(o)} \in \mathbb{R}^{d_z}$ before it is summed up with the other heads' bindings to form the TPR of a structure with H constituents: this is the output of the TPMHA layer.

$$\mathrm{TPMHA}(\boldsymbol{z}_{t,l}, \boldsymbol{z}_{1:T,l}) = \sum_h \left[\boldsymbol{W}_{h,l}^{(o)} (\bar{\boldsymbol{v}}_{t,l}^h \odot \boldsymbol{r}_{t,l}^h) + \boldsymbol{b}_{h,l}^{(o)} \right] \tag{6}$$

Note that, in this binding, to control dimensionality, we use a contraction of the tensor product, pointwise multiplication $\odot$: this is the diagonal of the tensor product. For discussion, see the Appendix.

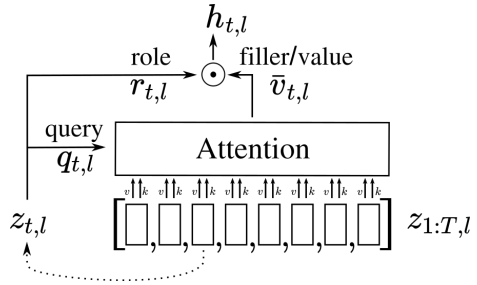

Figure 1: A simplified illustration of our TP-Attention mechanism for one head at position t in layer l. The main difference from standard Attention is the additional role representation that is element-wise multiplied with the filler/value representation.

3 Experimental results

We evaluate our trained model on the concatenated interpolation and extrapolation datasets of the pre-generated files, achieving a new state of the art: see Table 1. For a more detailed comparison, we include in the appendix the interpolation and extrapolation performance of every module separately. Our model never quite converged, and was stopped prematurely after 1.7 million steps. We trained our model on one server with 4 V100 Nvidia GPUs for 25 days. Additional experimental details can be found in the Appendix.

Table 1: Model accuracy averaged over all modules. A sample is correct if all elements of the target sequence have been predicted correctly. >95% counts how many of the modules are over 95% accuracy.

	weights	steps	train	interpolation		extrapolation	
				acc	>95%	acc	>95%
Simple LSTM	18M	500k	-	57.00%	6	41.00%	1
Transformer (Saxton et al.)	30M	500k	-	76.00%	13	50.00%	1
Transformer (ours)	44.2M	500k	83.06%	75.33%	12	52.42%	1
		700k	85.01%	77.42%	14	52.00%	2
TP-Transformer (ours)	49.2M	500k	85.41%	78.30%	16	52.22%	2
		700k	**87.25%**	**80.67%**	**18**	**52.48%**	**3**
		1.7M	91.04%	84.24%	25	55.40%	3

4 Interpreting the learned roles

We report initial results of analyzing the learned structure of the encoder network's last layer from our 700k-step TP-Transformer. To this end, we sample 128 problems from the interpolation dataset of the *arithmetic__mixed* module and collect the role vectors from a randomly chosen head. We use k-means with $k = 20$ to cluster the role vectors from different samples and different time steps of the final layer of the encoder. Interestingly, we find separate clusters for digits in the numerator and denominator of fractions. When there is a fraction of fractions we can observe that these assignments are placed such that the second fraction reverses, arguably simplifying the division of fractions into a multiplication of fractions (see Fig. 2).

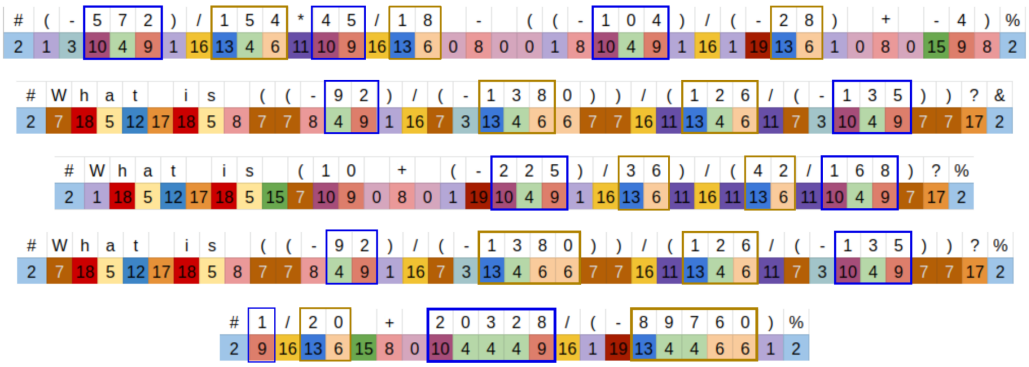

Figure 2: Samples of correctly processed problems from the *arithmetic__mixed* module. '#' and '%' are the start- and end-of-sentence symbols. The colored squares indicate the k-means cluster of the role-vector assigned by one head in the final layer in that position. Blue and gold rectangles respectively highlight numerator and denominator roles. They were discovered manually. Note how their placement is correctly swapped in rows 2, 3, and 4, where a number in the denominator of a denominator is treated as if in a numerator. Role-cluster 9 corresponds to the role *ones-digit-of-a-numerator-factor*, and 6 to *ones-digit-of-a-denominator-factor*; other such roles are also evident.

5 Related work

Several recent studies have shown that the Transformer-based model BERT (Devlin et al., 2018) captures linguistic relations such as those expressed in dependency-parse trees. This was shown for BERT's hidden activation states in (Hewitt & Manning, 2019; Tenney et al., 2019) and, most directly related to the present work, for the graph implicit in BERT's attention weights (Coenen et al., 2019; Lin et al., 2019). Future work applying the TP-Transformer to language tasks (like those on which BERT is trained) will enable us to study the connection between the *explicit* relations $\{\boldsymbol{r}^h_{t,l}\}$ the TP-Transformer learns and the *implicit* relations that have been extracted from BERT.

6 Conclusion

We have introduced the TP-Transformer, which enables the powerful Transformer architecture to learn to explicitly encode structural relations using Tensor-Product Representations. On the novel and challenging Mathematics Dataset, TP-Transformer beats the previously published state of the art by 8.24%. Our initial analysis of this model's final layer suggests that the TP-Transformer naturally learns to cluster symbol representations based on their structural position and relation to other symbols.

References

Andy Coenen, Emily Reif, Ann Yuan, Been Kim, Adam Pearce, Fernanda Viégas, and Martin Wattenberg. Visualizing and measuring the geometry of BERT. *arXiv preprint arXiv:1906.02715*, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. 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. 4129–4138, 2019.

Yongjie Lin, Yi Chern Tan, and Robert Frank. Open sesame: Getting inside BERT's linguistic knowledge. *arXiv preprint arXiv:1906.01698*, 2019.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1gR5iR5FX`.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

A The Mathematics Dataset

The Mathematics Dataset (Saxton et al., 2019) is a large collection of math problems of various types, including algebra, arithmetic, calculus, numerical comparison, measurement, numerical factorization, and probability. Its main goal is to investigate the capability of neural networks to reason formally. Each problem is structured as a character-level sequence-to-sequence problem. The input sequence is a free-form math question or command like `What is the first derivative of 13*a**2 - 627434*a + 11914106?` from which our model correctly predicts the target sequence `26*a - 627434`. Another example from a different module is `Calculate 66.6*12.14.` which has `808.524` as its target sequence.

The dataset is structured into 56 modules which cover a broad spectrum of mathematics up to university level. It is procedurally generated and comes with 2 million pre-generated training samples per module. The authors provide an interpolation dataset for every module, as well as a few extrapolation datasets as an additional measure of algebraic generalization.

We merge the different training splits *train-easy*, *train-medium*, and *train-hard* from all modules into one big training dataset of 120 million unique samples. From this dataset we extract a character-level vocabulary of 72 symbols, including *start-of-sentence*, *end-of-sentence*, and *padding* symbols[3].

B Implementation Details

We initialize the symbol embedding matrix $\boldsymbol{E}$ from $\mathcal{N}(0, 1)$, $\boldsymbol{W}^{(p)}$ from $\mathcal{N}(1, 1)$, and all other matrices $\boldsymbol{W}^{(\cdot)}$ using the Xavier uniform initialization as introduced by Glorot & Bengio (2010). The model parameters are set to $d_z = 512, d_f = 2048, d_v = 72, H = 8, L = 6$. We were not able to train the TP-Transformer, nor the regular Transformer, using the learning rate and gradient clipping scheme described by Saxton et al. (2019). Instead we proceed as follows: The gradients are computed using PyTorch's Autograd engine and their gradient norm is clipped at 0.1. The optimizer we use is also Adam, but with a smaller learning_rate $= 1 \times 10^{-4}$, beta1 $= 0.9$, beta2 $= 0.995$. We train with a batch size of 1024 up to 1.7 million steps.

C Feed-forward Layer

The feed-forward layer of a cell consists of an affine transformation followed by a ReLU activation and a second affine transformation:

$$\mathrm{FF}(\boldsymbol{x}) = \boldsymbol{W}_l^{(g)} \operatorname{ReLU}(\boldsymbol{W}_l^{(f)} \boldsymbol{x} + \boldsymbol{b}_l^{(f)}) + \boldsymbol{b}_l^{(g)} \tag{7}$$

Here, $\boldsymbol{W}_l^{(f)} \in \mathbb{R}^{d_f \times d_z}$, $\boldsymbol{b}_l^{(f)} \in \mathbb{R}^{d_f}$, $\boldsymbol{W}_l^{(g)} \in \mathbb{R}^{d_z \times d_f}$, $\boldsymbol{b}_l^{(g)} \in \mathbb{R}^{d_z}$ and $\boldsymbol{x}$ is the function's argument. As in previous work, we set $d_f = 4d_z$.

D The Decoder Network

The decoder network is a separate network with a similar structure to the encoder that takes the hidden states of the encoder and auto-regressively generates the output sequence. In contrast to the encoder network, the cells of the decoder contain two TPMHA layers and one feed-forward layer. We designed our decoder network analogously to Vaswani et al. (2017) where the first attention layer attends over the masked decoder states while the second attention layer attends over the final encoder states. During training, the decoder network receives the shifted targets (teacher-forcing) while during inference we use the previous symbol with highest probability (greedy-decoding). The final symbol probability distribution is given by

$$\hat{\boldsymbol{y}}_{\hat{t}} = \operatorname{softmax}(\boldsymbol{E}^T \hat{\boldsymbol{z}}_{\hat{t},L}) \tag{8}$$

where $\hat{\boldsymbol{z}}_{\hat{t},L}$ is the hidden state of the last layer of the decoder at decoding step $\hat{t}$ of the output sequence and $\boldsymbol{E}$ is the shared symbol embedding of the encoder and decoder.

[3]Note that Saxton et al. (2019) report a vocabulary size of 95, but this figure encompasses characters that never appear in the pre-generated training and test data.

E Accuracy per-module

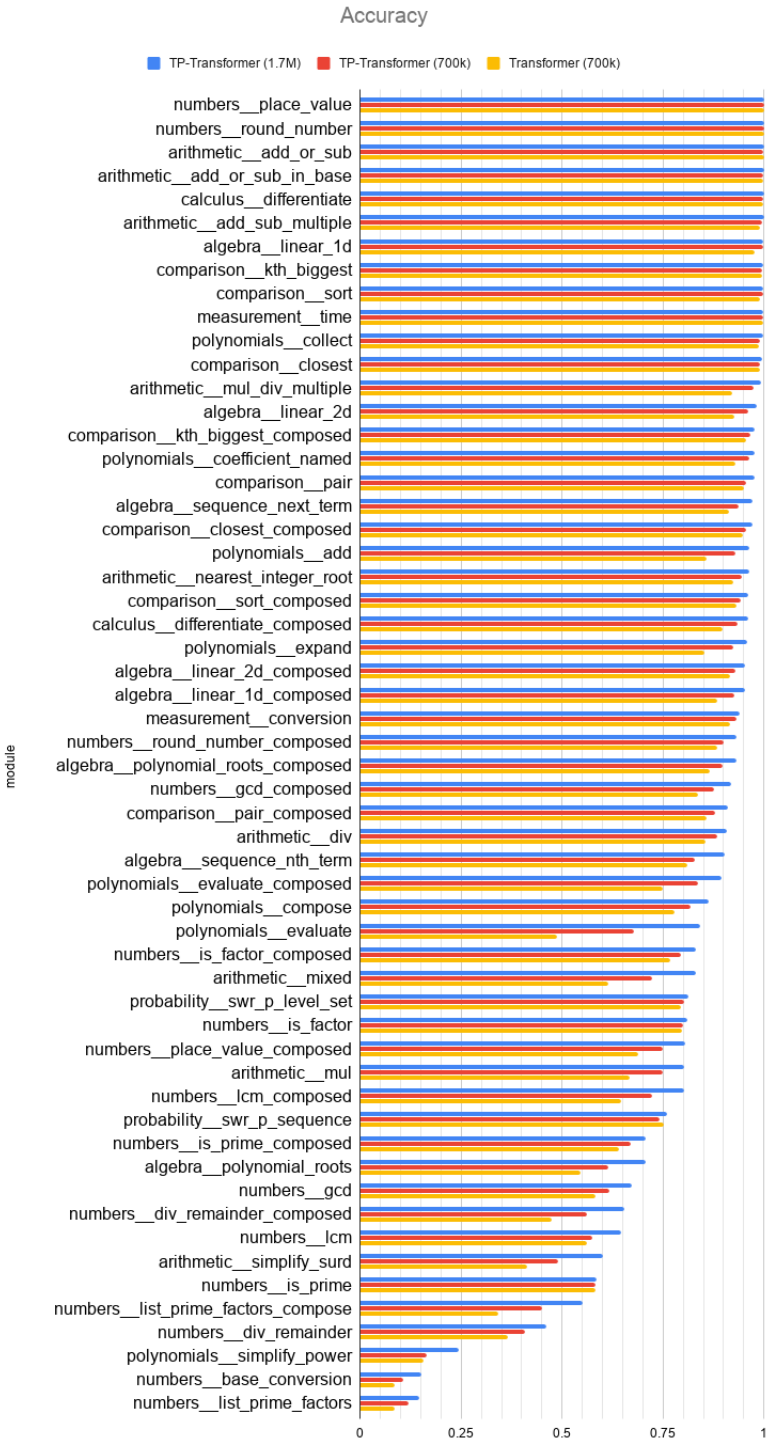

Figure 3: The accuracies of our implementation of the Transformer (700k steps) and the TP-Transformer (700k and 1.7M steps) for every module of the Mathematics Dataset.