ENHANCING THE TRANSFORMER WITH EXPLICIT RELATIONAL ENCODING FOR MATH PROBLEM SOLVING

Anonymous authors
Paper under double-blind review

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 regular attention. The TP-Transformer's attention maps give better insights into how it is capable of solving the Mathematics Dataset's challenging problems. Pretrained models and code will be made available after publication.

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, where, among other things, correctly parsing expressions and compositionally evaluating them is crucial. Given as input a free-form math question in the form of a character sequence 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.*, the model must produce an answer exactly matching the specified target character-sequence *–(s + 3)*(3*s – 2)*. Our proposed model is trained end-to-end and infers the correct answer for novel examples without any problem-specific structural biases.

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 information content at t' in effect passes via this edge to affect the 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. Thus, each layer of the Transformer can be viewed as a complete, directed, weighted, labeled graph. Prior NLP work has interpreted certain edges of these graphs in terms of linguistic relations (Sec. 8), 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**, Smolensky (1990)). 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.[1] 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 **role-vector** (which we refer to in some contexts as a 'relation vector'). The query is interpreted as seeking the appropriate filler for that role (or equivalently, the appropriate string-location for fulfilling that relation). 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.).

An interpretation of an actual learned relation illustrates this (see Fig. 3 in Sec. 5.1). One head of our trained model can be interpreted as partially encoding the relation *second-argument-of*. The top-layer cell dominating an input digit seeks the operator of which the digit is in the second-argument role. That cell generates a vector $\boldsymbol{r}_t$ signifying this relation, and retrieves a value vector $\boldsymbol{v}_{t'}$ describing the operator from position t' that stands in this relation. The result of this head's attention is then the binding of filler $\boldsymbol{v}_{t'}$ to role $\boldsymbol{r}_t$; this binding is added to the bindings resulting from the cells other attention heads.

On the Mathematics Dataset (Sec. 3), the new model sets a new state of the art for the overall accuracy, and for all the individual-problem-type module accuracies (Sec. 4). 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. 5).

2 THE TP-TRANSFORMER

The encoder network 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 a TP-Multi-Head Attention (TPMHA) layer followed by a fully-connected feed-forward (FF) layer, each preceded by layer normalization (LN) and followed by a residual connection. The input into $\text{cell}_{t,l}$ is the output of the $\text{cell}_{t,l-1}$ and doesn't depend on the state of any other cells of the same layer, which allows their outputs to be computed in parallel.

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

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 vector representation of those symbols. We also include a positional representation $\boldsymbol{p}_t$ using the same sinusoidal encoding schema as introduced by (Vaswani et al., 2017). The input of the first $\text{cell}_{t,1}$ is $\boldsymbol{z}_{t,0}$:

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

where $\boldsymbol{W}^{(p)} \in \mathbb{R}^{d_z \times d_z}$, $\boldsymbol{b}^{(p)} \in \mathbb{R}^{d_z}$, $\odot$ is elementwise multiplication (see Sec. 2.1), and $\boldsymbol{r}_t$ is a position- and symbol-dependent role representation.

2.1 TP-MULTI-HEAD ATTENTION

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

[1] The tensor product operation (when the role-embedding vectors are linearly independent) 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.

state $z_{t,l}$, where $d_k = d_z/H$:

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

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

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

i.e., a weighted sum of all T values of the same layer and attention head (see Fig. 1). Here $\alpha^{h,i}_{t,l} \in (0,1)$ 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:

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

The scale factor $\frac{1}{\sqrt{d_k}}$ can be motivated as variance reducing factor under the assumption that the elements of $\boldsymbol{q}^h_{t,l}$ and $\boldsymbol{k}^h_{t,l}$ 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 $\bar{\boldsymbol{v}}^h_{t,l}$ with our relation vector $\boldsymbol{r}^h_{t,l}$, followed by an affine transformation $\boldsymbol{W}^{(o)}_{h,l} \in \mathbb{R}^{d_k \times d_z}, \boldsymbol{b}^{(o)}_{h,l} \in \mathbb{R}^{d_z}$ before it is summed up to form the TPR of a structure with H constituents: this is the output of the TPMHA layer.

$$\mathrm{TPMHA}(\boldsymbol{z}_{t,l}) = \sum_h \boldsymbol{W}^{(o)}_{h,l}(\bar{\boldsymbol{v}}^h_{t,l} \odot \boldsymbol{r}^h_{t,l}) + \boldsymbol{b}^{(o)}_{h,l} \tag{6}$$

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.

It is worth noting that the l^{th} TPMHA layer returns a vector that is quadratic in the inputs $\boldsymbol{z}_{t,l}$ to the layer: the vectors $\boldsymbol{v}^h_{i,l}$ that are linearly combined to form $\bar{\boldsymbol{v}}^h_{t,l}$ (4), and $\boldsymbol{r}^h_{t,l}$, are both linear in the $\boldsymbol{z}_{i,l}$ (3), and they are multiplied together to form the output of TPMHA (6). This means that, unlike regular attention, TPMHA can increase, over successive layers, the polynomial degree of its representations as a function of the original input to the Transformer. Although it is true that the feed-forward layer following attention (Sec. 2.2) introduces its own nonlinearity even in the regular Transformer, in the TP-Transformer the attention mechanism itself goes beyond mere linear re-combination of vectors from the previous layer. This provides further potential for the construction of increasingly abstract representations in higher layers.

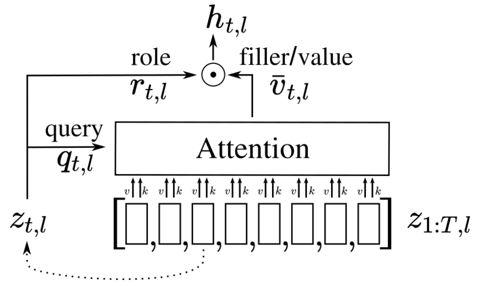

Figure 1: A simplified illustration of our TP-Attention mechanism for one head. The main difference with the regular Attention is the additional role representation that is elementwise multiplied with the filler/value representation.

2.2 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:

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

Here, $\boldsymbol{W}^{(f)}_l \in \mathbb{R}^{d_z \times d_f}, \boldsymbol{b}^{(f)}_l \in \mathbb{R}^{d_f}, \boldsymbol{W}^{(g)}_l \in \mathbb{R}^{d_f \times d_z}, \boldsymbol{b}^{(g)}_l \in \mathbb{R}^{d_z}$ and $\boldsymbol{x}$ is the function argument. As in previous work, d_f is chosen to be 4 times larger than d_z.

2.3 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

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

where $\hat{\boldsymbol{z}}_{\hat{t},L}$ 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 The Mathematics Dataset

The Mathematics Dataset (Saxton et al. (2019)) is a large collection of math problems of various types. Its main goal is to investigate the capability of neural networks to reason algebraically. Each problem is structured as a general sequence-to-sequence problem. The input sequence is a free-form math question or command like e.g. *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 train 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[2].

4 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. As Fig. 5 shows, in every module the TP-Transformer matches or out-performs the Transformer. 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.

4.1 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 $\mathrm{learning\_rate} = 1 \times 10^{-4}, \mathrm{beta1} = 0.9, \mathrm{beta2} = 0.995$. We train with a batch-size of 1024 up to 1.7 million steps.

[2]Note that Saxton et al. (2019) report a vocabulary size of 95 which is misleading in that this figure encompasses characters that never appear in the pre-generated train and test data.

Table 1: Model accuracy averaged over all modules. A sample is correct if all elements of the target sequence have been predicted correctly. The column ">95%" counts how many of the modules achieve over 95% accuracy. Boldface marks the best-performing model up to 700k steps.

	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%	-	-	-
		700k	**87.25%**	**80.67%**	**18**	**52.48%**	**3**
		1.7M	91.04%	84.24%	25	55.40%	3

5 INTERPRETING THE LEARNED ROLES

We analyse the learned role representations of the last layer of the encoder network. 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 fraction of fractions into a multiplication of fractions (see Fig. 2).

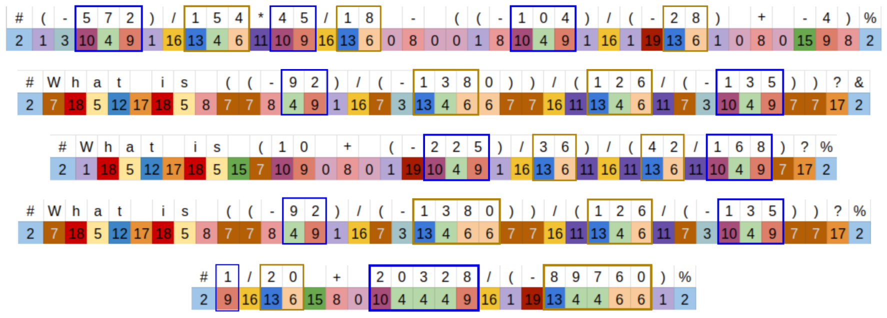

Figure 2: Samples from the *arithmetic__mixed* module. "#" denotes the start-of-sentence symbol and "%" the end-of-sentence symbol. The colored squares indicate the k-means cluster of the role-vector assigned by one head in the final layer in that position. Blue rectangles show numerator roles and golden rectangles denominator roles. They were discovered manually. Note how their placement is swapped in rows 2, 3, and 4. Role-cluster 9 corresponds to the role *ones-digit-of-a-numerator-factor*, and 6 to the role *ones-digit-of-a-denominator-factor*; other such roles are also evident.

5.1 INTERPRETING THE ATTENTION MAPS

In Fig. 3 we display three separate attention weight vectors of one head of the last TP-Transformer layer of the encoder. Gold boxes are overlaid to highlight most-relevant portions. The row above the attention mask indicates the symbols that give information to the symbol in the bottom row. In each case, they give to '/'. Seen most simply in the first example, this attention can be interpreted as encoding a relation *second-argument-of* holding between the attended digits and the '/' operator. The second and third examples show that several numerals in the denominator can participate in this relation. The third display shows how a numerator-numeral (-297) intervening between two denominator-numerals is skipped for this relation.

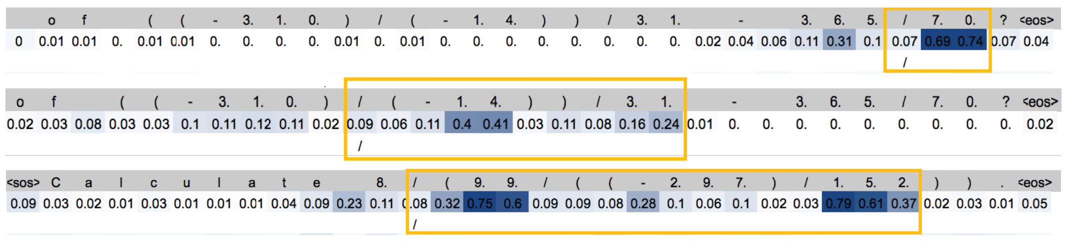

Figure 3: TP-Transformer attention maps for three examples as described in section 5.1.

6 Interpretation of a TP-Transformer column

We analyse the attention maps for one difficult sample to gain insight into how the TP-Transformer might be solving it. We will focus on the encoder part of the input sequence: *Solve –3225 = –17*p – 3480 for p.* To this end, we visualize in Fig. 4 the attention maps from the perspective of position $t = 10$ which is the column annotated by the second "2".

.5The heatmaps are the TP-Attention of 8 different heads (rows) over the different positions it can attend to (columns). The figure displays 6 heatmaps for the 6 layers from 1 to 6 ordered from top to bottom. The attention maps of the TP-Transformer are surprisingly instructive while the Transformer equivalent remains difficult to interpret (see Fig. 6).

In the first layer we can see one attention head that very strongly attends to the "=" position. While not shown here, it turns out that the activation of that head is shared among all other positions. We argue that it might demarcate the problem type and set the stage for the following layers because the "=" representation has been incorporated into every cell state.

The second layer is also striking because it again shares a common activation pattern among all cells. In this case, three attention heads capture the region surrounding cell t for every t. The final result is three parallel and slightly offset convolutions over the states of the previous layer. This allows every cell to collect representations from neighbouring cells.

The third layer follows a similar pattern: every cell_t attends to cell_{t-1}, though some additional spurious attentions exist.

The fourth layer is harder to interpret. However, we want to point out that most of the cells seem to attend to the same number. At other positions the conclusion is similar: the focus is on local pattern, roughly clustered around the different numbers.

The fifth layer is where we observe for the first time a strong attention across the three main numbers of the problem. We believe this is where the network divides in all cells the current number representations by "-17".

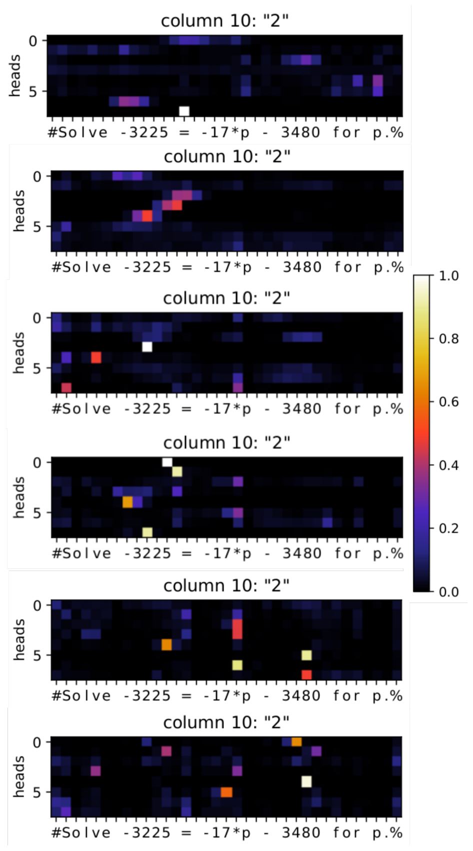

Figure 4: The attention maps of all layers for $t = 10$ from the 700k-steps TP-Transformer. Every row is the softmax attention of an attention head over the T positions. The input symbol of every position is given by the x-axis. The # represents the start-of-sentence symbol and the % represents the end-of-sentence symbol. The layers are ordered from the first layer at the top to the last layer at the bottom.

The last layer is where we believe the TP-Transformer performs the last step of the computation. It is here where it subtracts the third number from the first. If you observe closely you will notice that column 10 is the tens digit of the first number which only attends to the tens digit of the third number. It turns out that the cell at position 25 with input "8" symmetrically attends to column 10, i.e., the tens digit cell of the first number attends to the tens digit cell of the third number. This symmetry holds for all 4 digit positions and might be a direct consequence of Eq. 13.

7 Insights and deficits of multiple multi-head attention layers

7.1 Multi-Head Attention subspaces capture virtually all information

In this section we make an argument against the common misconception that Multi-Head Attention projections access only a small part of the full information content.

Let us consider a toy example where the attention layer of a $\text{cell}_{t,l}$ only attends to the $\text{cell}_{t',l}$. In this setting, the post-attention representation becomes

$$\begin{aligned}\text{AttentionHead}(z_{t,l}) &= z_{t,l} + \boldsymbol{W}_l^{(o)}(\boldsymbol{W}_l^{(v)} z_{t',l} + \boldsymbol{b}_l^{(v)}) + \boldsymbol{b}_l^{(o)} \\ &= z_{t,l} + o(v(z_{t',l})) \\ &\approx z_{t,l} + f(z_{t',l})\end{aligned} \tag{9}$$

where o, v, f are the respective affine maps. Note that even though $\boldsymbol{W}_l^{(o)}$ and $\boldsymbol{W}_l^{(v)}$ induce a projection into an 8 times smaller vector space, it remains in our setting highly unlikely that h is surjective and loses information about $\boldsymbol{z}_{t',l}$. This is due to the fact that our input $\boldsymbol{z}_{t',l}$ is a discrete and finite set of vectors. Consider the embedding representation $\boldsymbol{z}_{t',0}$ which can only represent 73 unique vectors for the 73 unique symbols in our vocabulary. For this setting it is trivial to find a map that is bijective even for just a single dimension. With more layers or a bigger vocabulary, the number of unique representations will grow but remains discrete and finite and can, at least theoretically, always be mapped into an unlimited real number representation.

We empirically test to what extent the trained Transformer and TP-Transformer lose information. To this end, we randomly select $n = 100$ samples and extract the hidden state of the last and most "populated" layer of the encoder $\boldsymbol{z}_{t,6}$, as well as the value representation $v(\boldsymbol{z}_{t,6})$ for every head. We then train an affine model to reconstruct $\boldsymbol{z}_{t,6}$ from $v(\boldsymbol{z}_{t,6})$:

$$\begin{aligned}\hat{\boldsymbol{z}}_{t,6} &= \boldsymbol{W}_h v_h(\boldsymbol{z}_{t,6}) + \boldsymbol{b}_h \\ e &= \frac{1}{n}(\hat{\boldsymbol{z}}_{t,6} - \boldsymbol{z}_{t,6})^2\end{aligned} \tag{10}$$

For both trained models, the TP-Transformer and the regular Transformer, the mean squared error e averaged across all heads is only 0.017 and 0.009 respectively. This indicates that the attention mechanism does not only incorporate a subspace of the vectors it attends to but affine transformations that preserve almost the full information content of the cell that had high attention.

7.2 The Binding Problem of stacked Attention layers

The *binding problem* refers to the problem of binding features together into objects while keeping them separated from other objects. It has been studied in the context of theoretical neuroscience (v.d. Malsburg (1981); Von Der Malsburg (1994)) but also with regards to connectionist machine learning models (Hinton et al. (1984)). The purpose of a binding mechanism is to allow the fully distributed representation of symbolic structure (like a hierarchy of features) which has recently resurfaced as an important research direction for recent neural network research Tang et al. (2018); Palangi et al. (2017); Schlag & Schmidhuber (2018); Schmidhuber (1993); van Steenkiste et al. (2019).

In this section, we describe how the regular Attention mechanism is ill suited to capture complex nested representations and provide an intuitive understanding of the benefit of our TP-Attention. We understand the attention layer of a cell as the means by which the *subject* (the cell state) queries all other cells for an *object*. We then show how a hierarchical representation of multiple queries becomes ambiguous in multiple layers of regular attention layer.

Consider the string "(a/b)/(c/d)". A good neural representation captures the hierarchical structure of the string such that it not be confused with the similar-looking but structurally different string "(a/d)/(c/b)". Our TP-Attention makes use of a binding mechanism in order to explicitly support complex structural relations by binding together the object representations with high attention with a subject-specific role representation.

Let us continue with a more technical example: consider a Transformer network where every cell only consists of a Multi-Head Attention layer with a residual connection. In this setting, assume that $\text{cell}_{a,l}$ only attends to $\text{cell}_{b,l}$, and $\text{cell}_{c,l}$ only attends to $\text{cell}_{d,l}$ where a, b, c, d are distinct positions of the input sequence. For simplicity, we will only consider one head of the Multi-Head Attention layer. In this case

$$\begin{aligned} \boldsymbol{z}_{a,l+1} &= \boldsymbol{z}_{a,l} + o_l(v_l(\boldsymbol{z}_{b,l})) \\ \boldsymbol{z}_{c,l+1} &= \boldsymbol{z}_{c,l} + o_l(v_l(\boldsymbol{z}_{d,l})) \end{aligned} \tag{11}$$

where o and v are affine maps introduced in the previous section. Consider now the next layer $\text{cell}_{e,l+1}$ that attends over $\text{cell}_{a,l+1}$ and $\text{cell}_{c,l+1}$ in a hierarchical manner. This results in the representation

$$\begin{aligned} \boldsymbol{z}_{e,l+2} &= \boldsymbol{z}_{e,l+1} + o_{l+1}(v_{l+1}(\boldsymbol{z}_{a,l+1} + \boldsymbol{z}_{c,l+1})) \\ &= \boldsymbol{z}_{e,l+1} + o_{l+1}(v_{l+1}(\boldsymbol{z}_{a,l} + \boldsymbol{z}_{c,l} + o_l(v_l(\boldsymbol{z}_{b,l})) + o_l(v_l(\boldsymbol{z}_{d,l})))) \end{aligned} \tag{12}$$

Note that the final representation is ambiguous in the sense that it is unclear by looking only at Eq. 12 if $\text{cell}_{a,l}$ has picked $\text{cell}_{b,l}$ or $\text{cell}_{d,l}$. Either scenario would have lead to the same outcome which means that the network would not be able to distinguish between these two different structures. In order to resolve this ambiguity the regular Transformer must recruit other attention heads or find suitable non-linear maps in-between attention layers but it remains uncertain how the network might achieve a clean separation.

Our TP-Attention mechanism, on the other hand, specifically removes this ambiguity. Just like in the previous paragraph, we can simplify the representation of our TP-Attention using the syntax in Eq. 9. In this case, the respective representations are

$$\begin{aligned} &\boldsymbol{z}_{a,l+1} = \boldsymbol{z}_{a,l} + o_l(\boldsymbol{r}_{a,l} * v_l(\boldsymbol{z}_{b,l})), \\ &\boldsymbol{z}_{c,l+1} = \boldsymbol{z}_{c,l} + o_l(\boldsymbol{r}_{c,l} * v_l(\boldsymbol{z}_{d,l})), \\ &\boldsymbol{z}_{e,l+2} = \boldsymbol{z}_{e,l+1} + o_{l+1}(v_{l+1}(\boldsymbol{z}_{a,l} + \boldsymbol{z}_{c,l} + o_l(\boldsymbol{r}_{a,l} * v_l(\boldsymbol{z}_{b,l})) + o_l(\boldsymbol{r}_{c,l} * v_l(\boldsymbol{z}_{d,l})))). \end{aligned} \tag{13}$$

Note that the final representation is not ambiguous anymore. Binding the filler symbols $v_l(\boldsymbol{z}_{b,l})$ (our objects) with a subject-specific role representation $\boldsymbol{r}$ as described in Eq. 6 breaks the structural symmetry we had with regular attention. It is now simple for the network to specifically distinguish the two different structures.

8 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.

9 Conclusion

We have introduced the TP-Transformer which combines the Transformer architecture with 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 the final TPMHA layer indicates 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.

Geoffrey E Hinton, James L McClelland, David E Rumelhart, et al. *Distributed representations*. Carnegie-Mellon University Pittsburgh, PA, 1984.

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

Hamid Palangi, Paul Smolensky, Xiaodong He, and Li Deng. Deep learning of grammatically-interpretable representations through question-answering. *arXiv preprint arXiv:1705.08432*, 2017.

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`.

Imanol Schlag and Jürgen Schmidhuber. Learning to reason with third order tensor products. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9981–9993, 2018.

J. Schmidhuber. On decreasing the ratio between learning complexity and number of time-varying variables in fully recurrent nets. In *Proceedings of the International Conference on Artificial Neural Networks, Amsterdam*, pp. 460–463. Springer, 1993.

P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intell.*, 46(1-2):159–216, November 1990. ISSN 0004-3702. doi: 10.1016/0004-3702(90)90007-M. URL `http://dx.doi.org/10.1016/0004-3702(90)90007-M`.

Shuai Tang, Paul Smolensky, and Virginia R de Sa. Learning distributed representations of symbolic structure using binding and unbinding operations. *arXiv preprint arXiv:1810.12456*, 2018.

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

Sjoerd van Steenkiste, Klaus Greff, and Jürgen Schmidhuber. A perspective on objects and systematic generalization in model-based rl. *arXiv preprint arXiv:1906.01035*, 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.

C. v.d. Malsburg. Technical Report 81-2, Abteilung für Neurobiologie, Max-Planck Institut für Biophysik und Chemie, Göttingen, 1981.

Christoph Von Der Malsburg. The correlation theory of brain function. In *Models of neural networks*, pp. 95–119. Springer, 1994.

A Appendix

A.1 Accuracy per-module

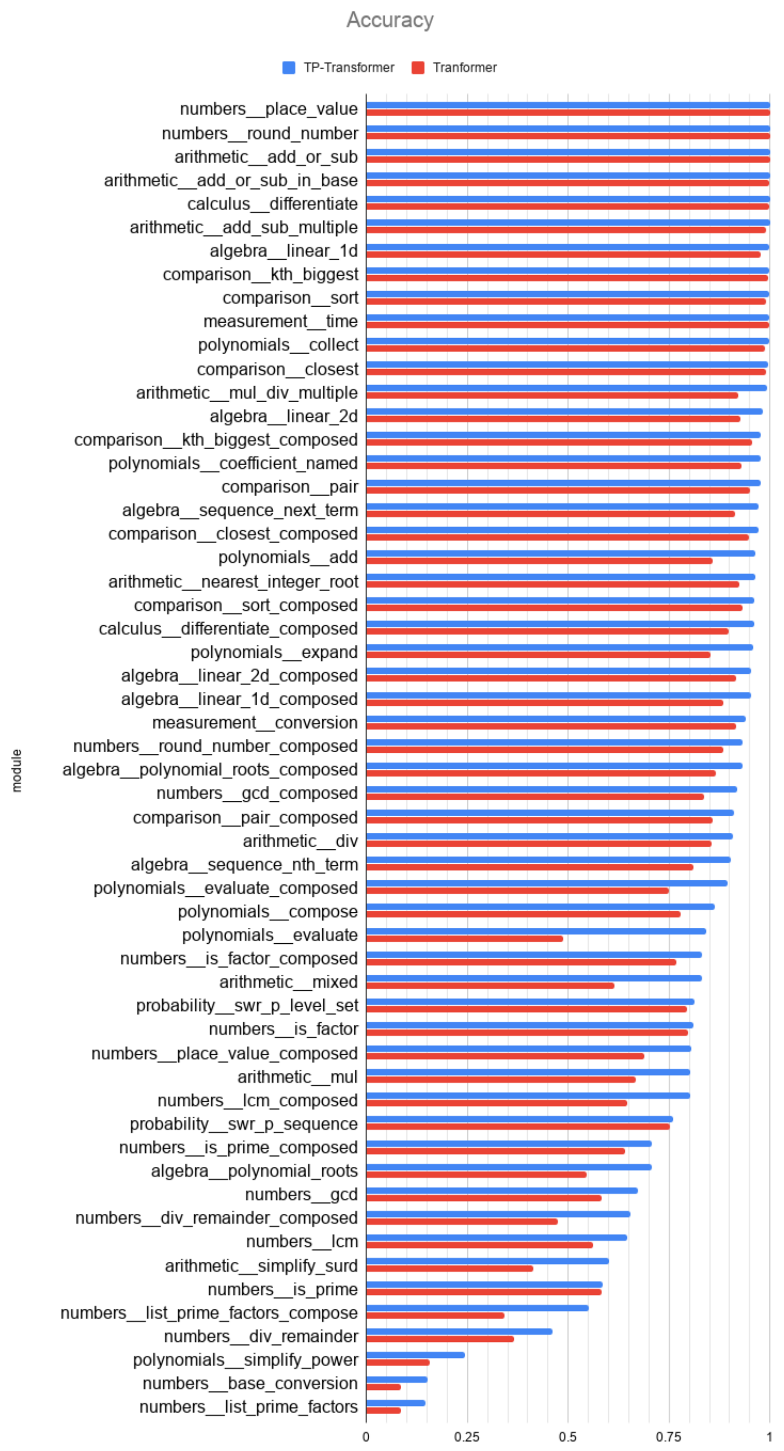

Figure 5: 1.7M-steps TP-Transformer (ours) and 700k-steps Transformer (ours) accuracies for every module of the Mathematics Dataset.

A.2 TP-Transformer vs Transformer Attention Maps

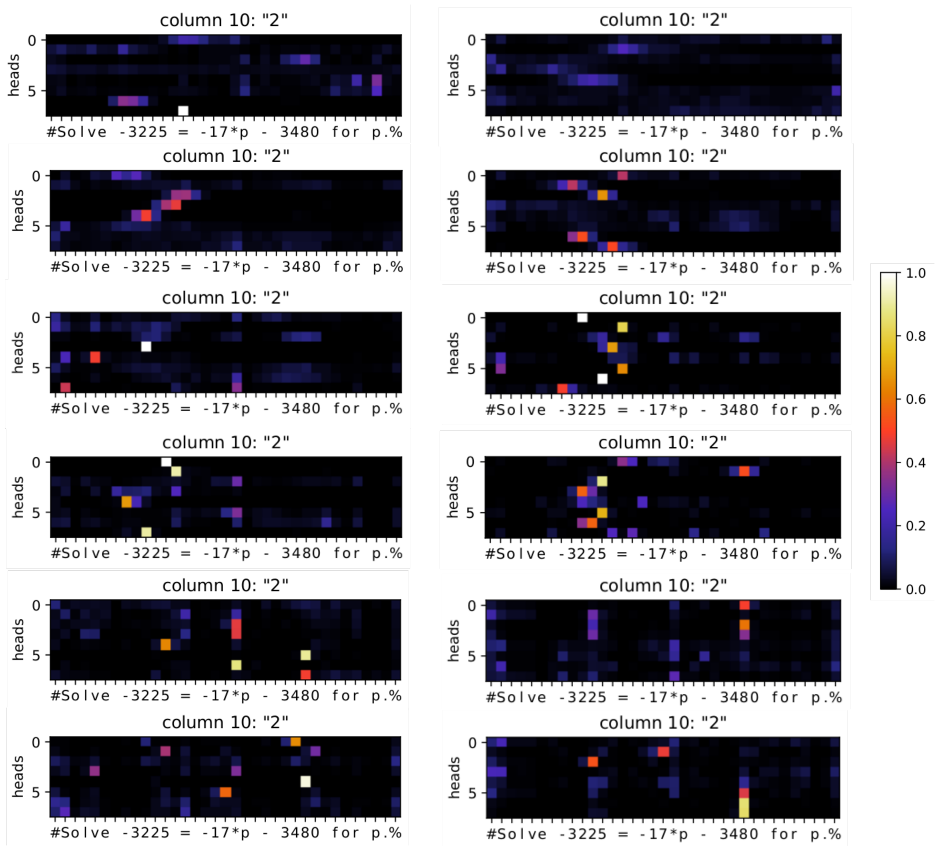

Figure 6: Attention map for all layers comparing the 700k-steps TP-Transformer (left) against the 700k-steps regular Transformer (right). Layers are ordered 1 to 6 from top to bottom. The input is ”Solve -3225 = -17*p - 3480 for p.” and both models correctly predicts ”-15”.

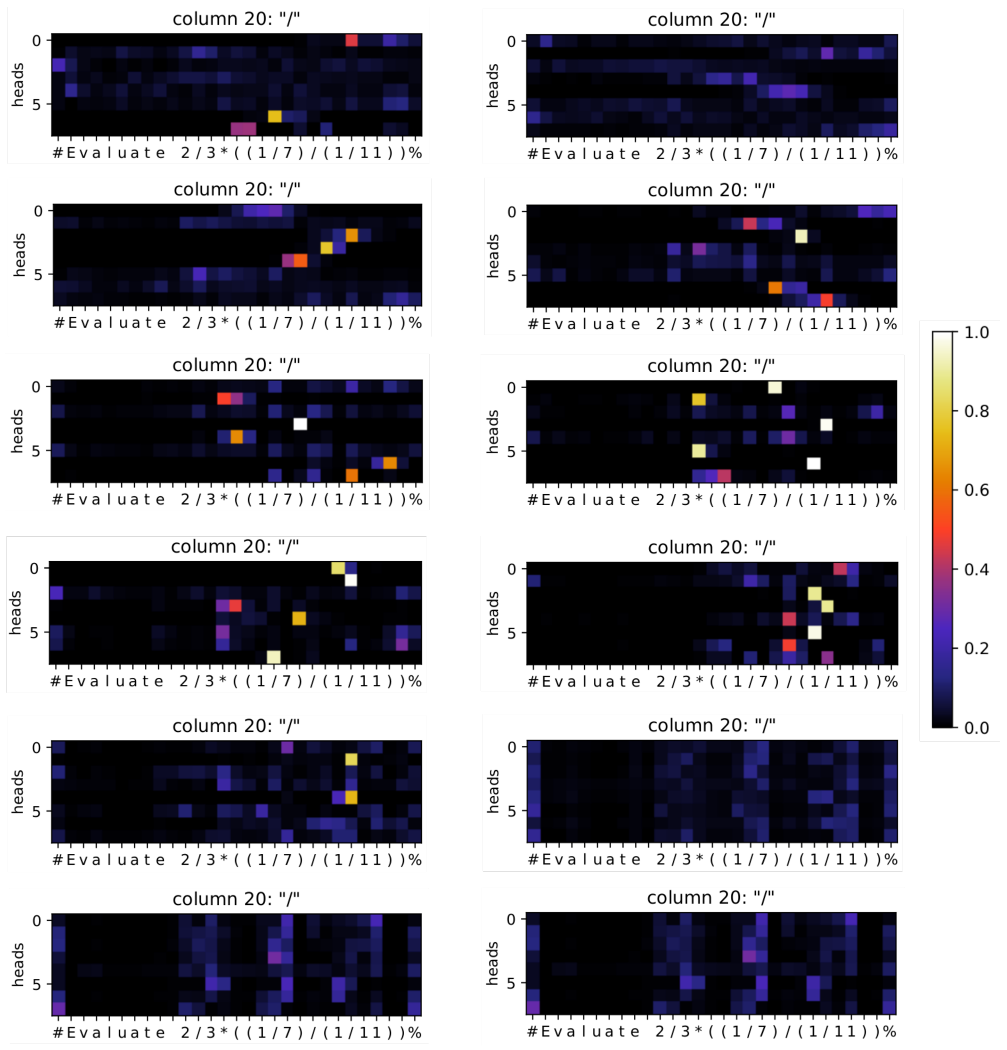

Figure 7: Attention map for all layers comparing the 700k-steps TP-Transformer (left) against the 700k-steps regular Transformer (right). Layers are ordered 1 to 6 from top to bottom. The input is ”Evaluate 2/3*((1/7)/(1/11))” and the TP-Transformer correctly predicts ”22/21” whereas the regular Transformer is wrong and predicts ”2/3”.

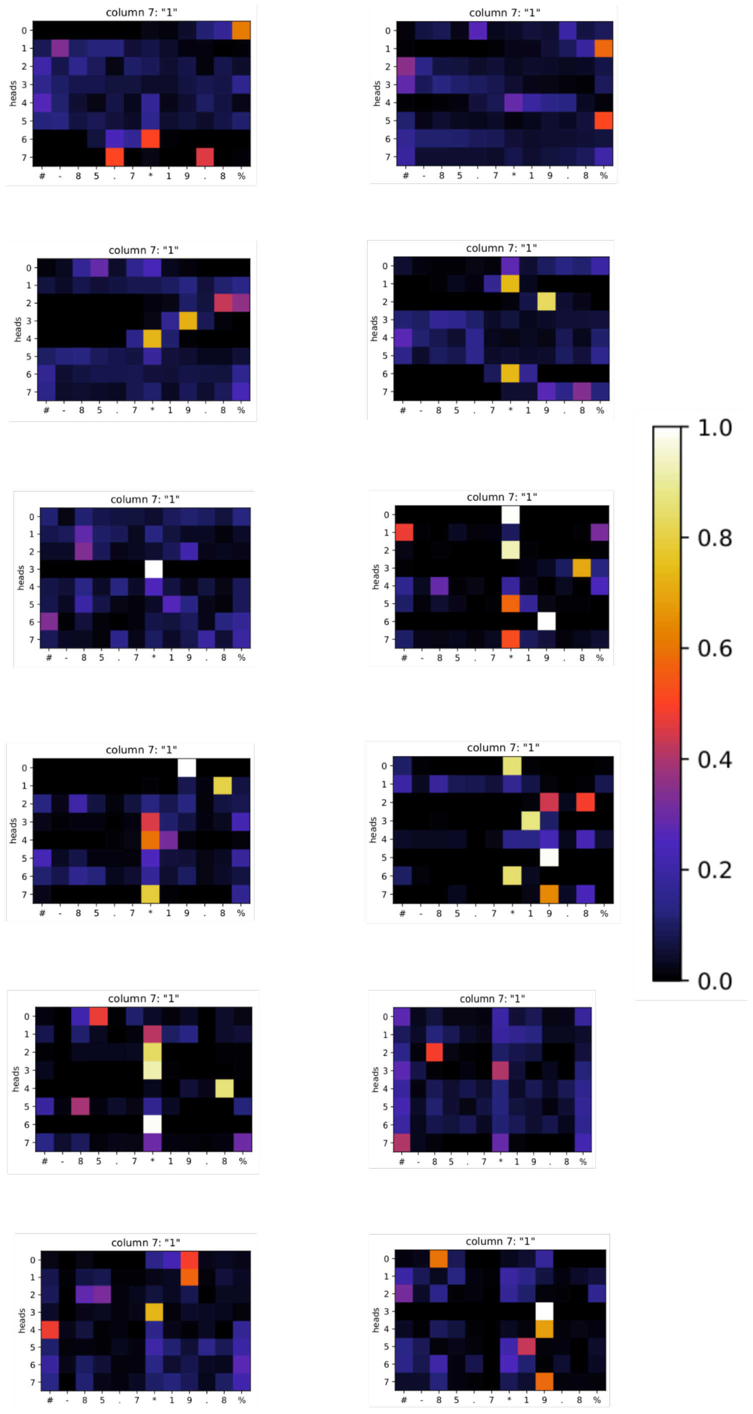

Figure 8: Attention map for all layers comparing the 700k-steps TP-Transformer (left) against the 700k-steps regular Transformer (right). Layers are ordered 1 to 6 from top to bottom. The input is ”-85.7*19.8” and the TP-Transformer correctly predicts ”-1696.86” whereas the regular Transformer is wrong and predicts ”-1708.86”.