Federal State Autonomous Educational Institution for Higher Education
National Research University Higher School of Economics

Faculty of Computer Science
Applied Mathematics and Information Science

BACHELOR'S THESIS

Research project

"Empirical Study of Transformers for Symbolic Mathematics"

Prepared by the student of group 171, 4th year of study,
Gelvan Kirill Pavlovich

Supervisor:
Senior Research Fellow, Chirkova Nadezhda Alexandrovna

Moscow 2021

Contents

Abstract

Developed for natural language processing (NLP), the Transformer architecture is now widely used with all kinds of sequences, including symbolic mathematical expressions. Given an example as a sequence of tokens, the model is able to solve complex math tasks like differentiation or integration. In this work, we investigate whether feeding data structure to the Transformer improves its performance on integration and solving ordinary differential equations (ODEs). We study recently developed tree-based model modifications and compare them. In our experience, the use of these alterations provides no benefit over the base approach. We assume this is due to an uncommonly large amount of data.

Аннотация

Архитектура трансформер, разработанная для обработки естественного языка, широко используется со всеми видами последовательностей, включая математические выражения. Получив последовательность токенов, модель способна решать сложные математические задачи, такие как нахождение производной или интеграла. В данной работе мы исследуем, улучшает ли качество трансформера добавление в него структуры данных на задачах интегрирования и решения обыкновенных дифференциальных уравнений. Мы изучаем недавно разработанные модификации модели основанные на деревьях и сравниваем их. Мы обнаружили, что использование этих изменений не дает преимуществ по сравнению с базовой моделью. Мы предполагаем, что это связано со слишком большим объемом данных.

Keywords

NLP, Transformer, Symbolic Mathematics, Integration, Tree-based Methods

1 Introduction

Neural networks already outperform humans in many different tasks: image classification, object and voice recognition, art style imitation, and others. These days success comes to even more fields: sequence-to-sequence neural network-based models are becoming better and better in natural language processing. Transformer architecture suggested in Vaswani et al. 2017 is one of the most recognizable and outstanding architectures for working with sequences. It is not only best-performing in processing plain texts for machine translation, summarization, and question answering Radford et al. 2018, but also shows promising results in other sequential data domains. In this work, we focus on applying Transformers to math problems (e.g., calculus, differentiation, integration, or solving differential equations). This is a promising research direction since some of them can be solved neither by any computer algebra system nor by any human. The limitation of the existing works in this direction is that the natural structure of math expressions is not utilized.

In this work, we empirically answer whether feeding data structure to the Transformer improves its performance on integration and solving first-order differential equations (ODEs).

Lample and Charton 2019 make a colossal step towards solving the symbolic mathematics task using deep learning. The paper introduces new datasets with both integration and ODEs, the solutions, and shows that Transformers achieve impressive results in addressing the task. However, the presented architecture does not use the expressions' structure.

To solve the discussed problem, we take inspiration in the data domain closely related to symbolic mathematics, namely source code processing. A line of works utilize code structure in a various number of problems Shiv and Quirk 2019; Shaw, Uszkoreit, and Vaswani 2018; Kim et al. 2020. Chirkova and Troshin 2020 compare these approaches and give recommendations on the practical use. We are aiming to make a similar contribution to symbolic mathematics. Particularly,

we re-implement and compare 4 different ways of using a tree data structure in the Transformer architecture on 2 different symbolic math tasks.

The paper is structured as follows. At first, we discuss previous attempts of solving symbolic math problems and analyze Lample and Charton 2019 in detail. Then we talk about different approaches to helping the Transformer to utilize the data structure. Finally, we describe conducted experiments and the results.

2 Related work

Initially, math problems were addressed in deep learning using recurrent neural networks (RNNs), similarly to most other sequential data. Arabshahi, Singh, and Anandkumar 2018 investigate how passing the structure of data to the RNNs can improve the performance. The authors compare listed models on a mathematical question answering task. More precisely, the task consists of equation verification and completion. Even though there are complicated math problems in the dataset, the job is not as sophisticated as solving the expressions. Different versions of long short-term memory Hochreiter and Schmidhuber 1997 are examined, one of which is Tree-LSTM introduced in Tai, Socher, and Manning 2015. It utilizes the math problem's structure and leads to a higher final score in accordance with the study.

However, eventually, the Transformer architecture has shown its dominance over RNNs Saxton et al. 2019. The authors present a synthetically generated dataset for various math problems. The core idea of the paper is to investigate the capability of modern sequence-to-sequence architectures to generalize. The dataset consists of paired sequences of free-form questions and answers. It includes basic math textbook problems (e.g., calculus, number theory, or differentiation). To achieve the goals, the authors deliberately limit both the question and the answer structure, forcing models to learn it independently. One of the samples from the data: "Question: Calculate -841880142.544 + 411127. Answer: -841469015.544". The research shows that the Transformer with internal sequential memory outperforms LSTM across all tasks. Even though the analysis does

not cover harder math problems, the outcome brings us closer to understanding the supremacy of the Transformer.

Integration and solving ODEs are a lot more sophisticated than any problem discussed above. Adding to the fact that there is no complete set of rules for solving given tasks, there is no labeled data available. Generating expressions and gathering answers only with the computer algebra systems does not help solve problems not following the predefined rules. Lample and Charton 2019 make the learning of integration possible: the authors propose a tree-structured representation of math problems and a way to generate labeled data for both the integration and the first and second-order differential equations. Davis 2019 heavily criticize Lample and Charton 2019: the author points out the limitations and artificiality of the presented datasets (e.g., having only one variable - "x"). Thus, one should investigate the datasets with caution. Moreover, the work blames Lample and Charton 2019 for biased comparison with state-of-the-art computer algebra systems. A more detailed review of the paper is presented in section 3.1.

3 Approaches to compare

3.1 Base approach

We have chosen to base our work on Lample and Charton 2019. Even though the paper has been questioned, it provides unique datasets and lacks comparison with tree-based architectures.

3.1.1 Data processing

With the polish notation's help, Lample and Charton 2019 create a bijection from the trees to the math problems, and it can be treated as a sequence of tokens. One generated sample and its integral are presented in fig. 3.1. Even though it is not illustrated in fig. 3.1, we should mention that longer numbers are presented as a sequence of distinguished nodes of digits. Thus, number *149* is represented

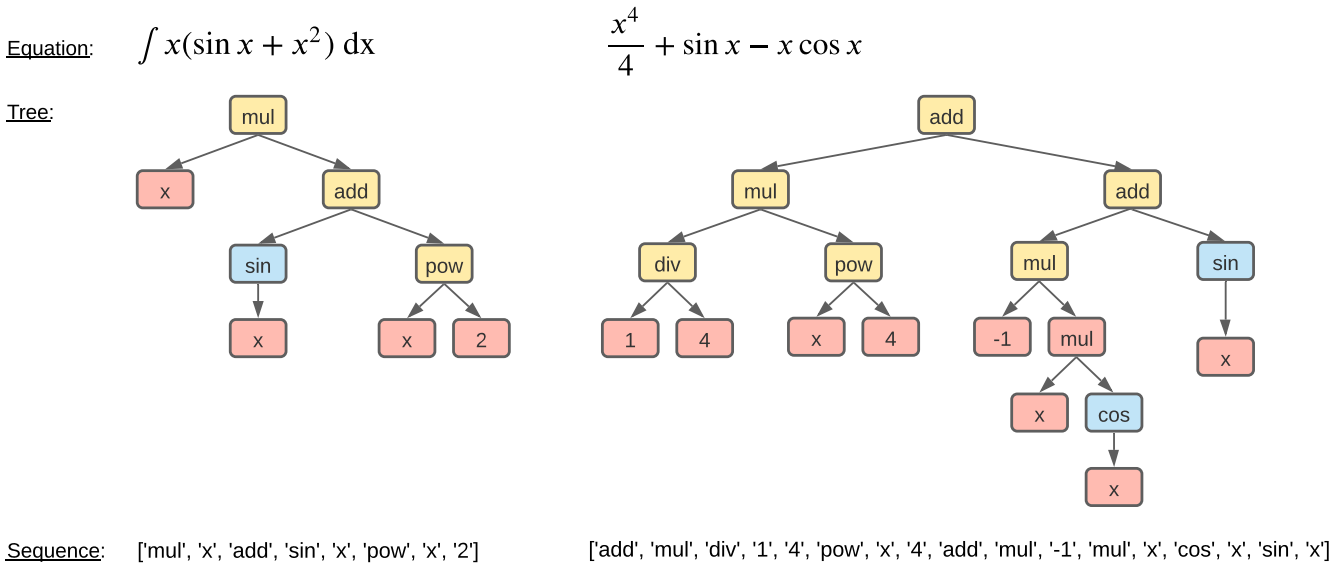


Figure 3.1: Example of an equation (left) and its integral (right)

as three consecutive nodes of *1*, *4* and *9* respectively.

Three ways of producing integration samples are proposed: FWD, BWD, and IBP. The first one consists of generating a random valid expression and finding its integral with simple hardcoded rules using SymPy Meurer et al. 2017. The second one stands for generating a function expression randomly, differentiating it, and using the reversed pair. The last abbreviation stands for Integration By Parts, one of the most well-known rules for finding integrals. This method generates two random functions and constructs an example using the rule if one of them has an integral in the training data. There are also two datasets with ordinary differential equations, the process of generating which is described in detail in Lample and Charton 2019.

The work also criticizes the bilingual evaluation understudy (BLEU, Papineni et al. 2002) metric, showing that it is incompatible with the task. There are countless ways to rewrite identical math expressions even after simplification. Fortunately, it is, in fact, computationally inexpensive to check math expressions equality using computer algebra systems. Thus, the authors use beam search and accuracy metric for evaluation. Specifically, all the hypotheses are checked for correctness, and if any hypothesis equals the answer, the equation is considered

adequately solved.

3.1.2 The Transformer architecture

Lample and Charton 2019 use a modification of the well-known Transfomer architecture for the conducted experiments. The model was introduced in 2017 and has become very famous during the next couple of years. It was designed to capture longer dependencies than RNNs for sequence-to-sequence problems like machine translation, so it appears suitable for the task. The Transformer consists of the encoder and the decoder. While the decoder acts as an autoregressive model, the encoder processes a whole sequence of tokens simultaneously. Both of them have stacked linear and multi-head attention layers (with *h heads* of attention).

Self-attention. Self-attention is a core concept of the Transformer architecture. Every head receives a sequence of input vectors $x = (x_1, \ldots, x_n)$ and outputs sequence of the same length $z = (z_1, \ldots, z_n)$. z_i is a vector of size d_k and is calculated as follows:

$$a_{ij} = \frac{x_i W^Q \left(x_j W^K\right)^T}{\sqrt{d_k}} \tag{3.1}$$

$$z_i = \sum_{j=1}^{n} \text{softmax}\left(a_{ij}\right)\left(x_j W^V\right) \tag{3.2}$$

Matrices W^Q, W^K and W^V are learnable parameters for queries, keys, and values, respectively. These are unique for every layer and head.

Positional encodings and embeddings. By the design the Transformer utilizes information about position of token in a sequence by using positional encodings or positional embeddings. Specifically, the input embedding $x_i \in R^{d_k}$ is summed with the positional representation $p_i \in R^{d_k}$: $\hat{x}_i = x_i + p_i$. Positional encodings are called p_i and are computed based on the sine and the cosine functions. Contrariwise, the positional embedding is learned for each position

$i \in 1 \dots n$: $p_i = e_i,\ e_i \in R^{d_k}$.

The architecture described in Lample and Charton 2019 uses *positional embeddings* by default. We aim at equipping the described approach with mechanisms of processing tree structure.

3.2 Different approaches of using structure in Transformer

3.2.1 Tree positional encodings

There are ways to utilize specifically tree structure in positional encodings. Shiv and Quirk 2019 propose positional encodings for tree-structured source code data. A token position in a tree is found by the path from the root to the associated node. Each child in the path is encoded with a one-hot vector. Thus, the encoding of a node is a concatenation of the one-hot vectors from the node to the root. Thus, backward or a "stack-like" encoding is proven by the authors to make it possible for any relationship between two nodes to be modeled by an affine transform of their positional encodings. An example can be found in fig. 3.2 on the right.

We implement the extension of this method described in Chirkova and Troshin 2020 for non-binary trees. Firstly, two hyperparameters are chosen: the maximum width and depth. Secondly, the nodes with larger depth values are clipped: the root is clipped first. Thirdly, exceeding width numbers are replaced with the maximum width.

Shiv and Quirk 2019 show that this method improves the performance of the Transformer with positional encodings and the Tree-LSTM architectures in source code-related tasks.

3.2.2 Relative position representations

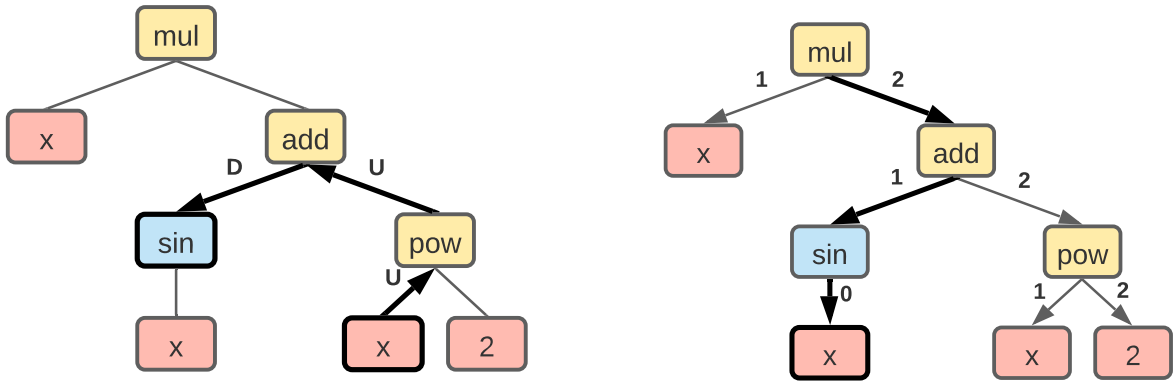
Shaw, Uszkoreit, and Vaswani 2018 learn embeddings for relations of tokens and plug them into the attention mechanism:

$$a_{ij} = \frac{x_i W^Q \left(x_j W^K + e_{ij}^K\right)^T}{\sqrt{d_k}} \tag{3.3}$$

$$z_i = \sum_{j=1}^{n} \text{softmax}(a_{ij}) \left(x_j W^V + e_{ij}^V\right) \tag{3.4}$$

where $e_{ij}^V, e_{ij}^K \in \mathbb{R}^{d_k}$ are learned embeddings for each relative position $i - j$. This method also occurs in literature under the name of sequential relative attention.

While Chirkova and Troshin 2020 incorporate relative position representations only to self-attention blocks, we hypothesize that modifying encoder-decoder attention may be profitable for the symbolic mathematics data and study it in section 4.2.



(a) Tree relative attention.
The relation between "x" and "sin" is encoded as "UUD".

(b) Tree positional encodings.
The node "x" is encoded as "012" (stack-like).

Figure 3.2: Illustration of tree based methods

3.2.3 Tree relative attention

Kim et al. 2020 propose an extension to relative attention for tree-structured data while solving code prediction tasks. Instead of defining distance by the number of tokens between two positions, we can encode it with the shortest path

in a tree. It is found via depth-first search (DFS) and can be represented by the number of up and down steps in the path. An example of the path is presented in fig. 3.2 on the left. Now we can learn embeddings and use them in the self-attention mechanism. Unfortunately, for that method, we can only learn scalar embedding $r_{ij} \in \mathbb{R}$ due to the limits of computational resources as mentioned in Kim et al. 2020:

$$\tilde{a}_{ij} = \frac{\exp(a_{ij} \cdot r_{ij})}{\sum_j \exp(a_{ij} \cdot r_{ij})} \tag{3.5}$$

Then $\tilde{a}_{ij}$ is used instead of a_{ij} for computing z_i in eq. 3.2.

Even though Chirkova and Troshin 2020 show that using the multiplication instead of the summation ($\alpha_{ij} + r_{ij}$) gets better results, we explore both strategies for symbolic mathematics data.

4 Experiments

4.1 Experimental setup

The main goal of all the experiments is to check whether the approaches found for code-related tasks can improve the performance of the Transformer on the symbolic mathematics tasks. The Transformer with 4 heads, 4 encoder and decoder layers is used with a hidden size $d_k = 256$. For optimization, we use Adam optimizer with a learning rate 0.001. All runs were held using an Nvidia Tesla V100 GPU.

Datasets. We have chosen to use the dataset described in Lample and Charton 2019. We conduct all the experiments on two different sets: integration (FWD) and first-order differential equations (ODE). The choice has been made considering both the diversity of datasets and the lack of computational resources. The main differences are the solving task and the ratio of mean sizes of the source to target sequences. FWD tends to produce approximately 2.7 times longer outputs than inputs because of the generating algorithm. In contrast, the ODE

dataset has much shorter solutions than their equations (the ratio is close to 0.4). It may affect the quality of implemented methods because several of them change only the encoder in the Transformer model (i.e., affect only the input sequence). The FWD dataset consists of 45 million samples, while the other set is 65M. Since the sizes are extremely large, we train models for 2-3 epochs (less than 120M training steps).

Another essential part of our methodology is deduplication. We have found a lot of similar equations in the train and test sets and removed them. More detailed research of the effect is presented in section 4.2.

Quality measurement. Usually, in machine translation, using beam search boosts performance, and taking into consideration the specificness of the task and testing method, we observe significant improvements with wider beams. We follow the strategy described in Lample and Charton 2019: during the training, *token-wise* accuracy is measured, but we use beam search and hypothesis checking during the test phase. It saves time and leads to correct predictions simultaneously.

One can notice that checking the solutions in the examined math problems is an easy task (e.g., by simplification in computer algebra systems). Nevertheless, it is too time-consuming to do during the training. We measure token-wise accuracy in the same way as it is commonly done in sequence-to-sequence tasks. However, during an evaluation, we test the hypothesis to be mathematically equal to the answer after simplification using SymPy. Let us define the checking algorithm in detail:

- Both the target and the hypothesis are simplified
- The mathematical difference between simplified expressions is calculated
- Any hypothesis returning zero on the previous step leads to "correct" state

We call the rate of correctly solved (as described above) equations to the size of the test set *equation-wise* accuracy and use it as the primary metric to compare

models following Lample and Charton 2019.

Relative position representations. The Transformer model has two types of attention blocks. The first one is self-attention (one in every encoder and every decoder), and the second one is encoder-decoder attention (one connecting the last encoder to every decoder block). Shaw, Uszkoreit, and Vaswani 2018 describe relative position representations as an improvement to both attention types. Nevertheless, we conduct experiments with two options:

- Incorporating relative position representations to self-attention blocks
- Incorporating relative position representations to both self-attention and encoder-decoder attentions

We have analyzed the data and decided that encoder-decoder attention might be an essential addition.

Tree positional encodings. All the datasets presented in Lample and Charton 2019 are generated from unary-binary trees. This, in turn, means there are only 3 node types: nodes with two children, nodes with one child, and operands. We handle listed cases separately (fig. 3.2):

- Encoding the path from the node to the only child as '0'
- Encoding left and right children differently as '1' and '2', respectively

4.2 Preliminary experiments

Deduplication effect. We have discovered that the chosen dataset has many intersections between train and test sets. It has already been found in the literature Piotrowski et al. 2020. It turns out that replacing all the numbers with a single constant reveals a leak in the data. Since most of the constant terms in solving mathematical expressions are treated in the same way, samples from the test set that differ only by a constant with samples from the training set are significantly easier for the model to solve correctly. We have found all the repeating samples and removed them from the test dataset.

After performing the described deduplication process, it turns out there are only 35% of unique samples in the test data (FWD) while the other 65% are already presented in the train. For the ODE dataset: 86.5% of samples are unique after removing constants. Our experiments confirm the leak percentages found in Piotrowski et al. 2020. Thus, we have decided to test our models only on completely unseen data to avoid biases in comparison.

Table 4.1: Equaition-wise accuracy for the baseline model evaluated with beam search (w=10).

	Initial test set	Deduplicated test set	Difference
FWD, 120M steps	*87.70*	*80.17*	**-7.53**
ODE, 60M steps	*90.59*	*89.18*	**-1.41**

To test how the leak affects accuracy, we have evaluated the baseline model on both the initial and the deduplicated datasets. After removing the constants, the accuracy drops by an average of 4.5% (table 4.1). We plan to generate entirely original data for testing with the algorithm described in Lample and Charton 2019 in the future. As a result, we measure quality on the reduced test dataset consisting of only unique samples.

Table 4.2: Equaition-wise accuracy for relative position representations with different hyperparameters. Dataset: FWD. Evaluation after 120M training steps with beam search (w=10).

Maximum relative positions	16	32	250
Relative Position Representations	*79.57*	*79.60*	***79.74***

Relative position representations hyperparameter search. Relative position representations modification has two hyperparameters: the maximum number of relative positions and using a negative distance flag. Since previous works on the same topic show that using negative distance improves the performance Chirkova and Troshin 2020, we focus on varying the maximum number of the positions. While with greater values, we provide more information to the model, with lower values, it can hypothetically save only the meaningful knowledge. We find that the largest number of distinguishable relative positions outperforms the others (table 4.2).

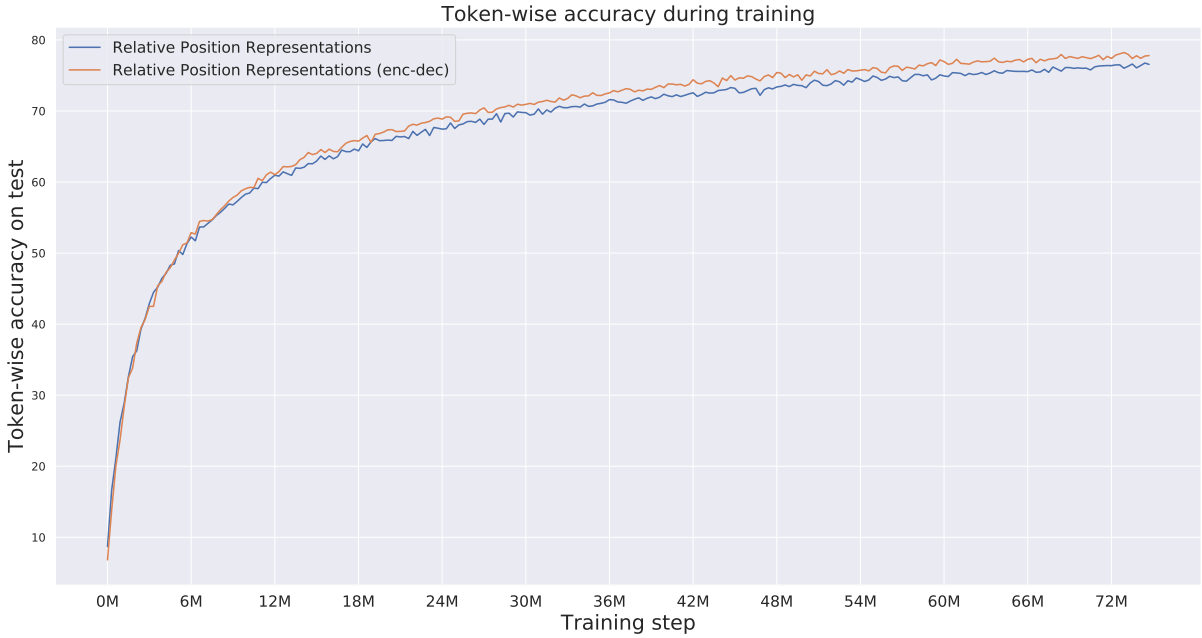


Figure 4.1: Token-wise accuracy with tuned hyperparameters on different modifications of relative position representations during training. Dataset: FWD.

Moreover, as described previously, we hypothesize that incorporating structural knowledge to the encoder-decoder attention might play a significant role in improving the quality. The data specificness gives us a reason to believe that it is crucial for the model to "see" the parallel links with the starting expression. We compared the models with and without incorporating relative positions into encoder-decoder attention and observed that this modification improves performance: 76.05 vs. 77.79 in terms of token-wise accuracy (fig. 4.1). For tree relative attention (as a similar approach), the same modification has been considered, but due to the limited resources is not implemented.

Tree positional encodings hyperparameter search. We have run a hyperparameter search for tree positional encoding in the encoder. The maximum possible width and length of the path have been searched using a logscale. All the relations longer than the max value have been truncated. We have also tried encoding only two types of relations (left relation and the only one are both encoded as '1') for changing maximum width (table 4.3). We use the maximum width of 4 and the maximum depth of 16 for all the consequent experiments.

Table 4.3: Equaition-wise accuracy for tree positional encoding with different hyperparameters. Dataset: FWD. Evaluation after 120M training steps evaluated with beam search (w=10)

max depth	16	32	64
width=2	*78.55*	*79.47*	*79.74*
width=4	***80.55***	*79.32*	—

Another addition we have tried was implementing tree positional encodings both in the encoder and the decoder networks. It requires generating trees on the fly for the evaluation process. It has not led to any benefits but requires more time and resources, so we abandoned the extension.

Positional embeddings and tree relative attention. For positional embeddings, there are no hyperparameters, and we use the modification in both the encoder and the decoder.

As for tree relative attention, there are two main options: to add or to multiply the embeddings as described in section 3.2.3. The multiplication quickly shows its dominance, and we plug the modification only in the encoder.

Choosing the number of training steps. Training models to achieve the quality presented in Lample and Charton 2019 requires many resources. Since the paper provides no information about the training time, we find an optimal number of training steps according to both our limited computational resources and preservation of representative evaluation. To compare existing methods on the FWD dataset, we have trained models for 3 epochs ($\sim$ 120M iterations). We stand by the point that even after 60M steps, the ordering of methods can be found and mostly does not change (fig. 4.2).

4.3 Comparing different approaches

The Transformer with positional embeddings is used as a baseline method. All the methods described in section 3 are implemented and tested. We perform a hyperparameter search described in preliminary experiments (due to the limited resources, we use the same hyperparameters for both ODE and FWD datasets).

Figure 4.2: Token-wise accuracy with tuned hyperparameters on different methods during training. Dataset: FWD.

The results of all trained models are presented in table 4.4. Equation-wise accuracy (section 4.1) is found with beam search after training for 120M iterations for the FWD dataset and 60M for the ODE, respectively. Based on the study of the FWD dataset and the findings described in section 4.2, we train the models only for 60M steps. Since the source data can consist of up to 500 tokens and methods like tree relative attention require enormous amounts of space and time, we limit the training time by 5 days.

Table 4.4: Equation-wise accuracy for both datasets and 4 models with tuned hyperparameters and beam search (width=10). Mean and standard deviation by 3 evaluations. *Runs were clipped by 5 days of training time.

	Integration	First order diff. eq.
Positional Embeddings	80.17 ± 0.40	89.18 ± 0.35
Tree Positional Encodings	79.88 ± 0.45	77.79 ± 0.40*
Relative Position Representations	80.10 ± 0.15	89.53 ± 0.27
Tree Relative Position Representations	79.76 ± 0.12	82.75 ± 0.25*

We examine that there is no statistically significant difference across all methods. We hypothesize that the reason for the obtained results is that the dataset is abnormally large. Thus, the model learns all the dependencies from the data

and needs neither prior knowledge of the tree structure of the data nor additional guidance in its utilization. We believe it happens because we have taken into account the specifics of the problem (section 4.2) and plugged the extension in all of the attention blocks.

	Simple example	*One sample finding*	*Different number of hypotheses required*	*Most cases with long numbers differ significantly*
Equation:	$\int x(\sin x + x^2)dx$	$\int -48x^5 + \frac{x\cos x}{20} + \frac{\sin x}{20} + 1\ dx$	$\int -\sin^2 x + \cos^2 x\ dx$	$\int 24690x + \frac{6789}{x} dx$
1. Positional embedding:	$\frac{x^4}{4} - x\cos x + \sin x$	$-8x^6 + \frac{x\sin(x)}{20} + x - \frac{\cos(x)}{20}$	$\sin x \cos x$ (hyp=86)	$12435x^2 + 6789\log x$
2. Tree positional encoding:	$\frac{x^4}{4} - x\cos x + \sin x$	$x^3(x+1)^{\frac{2}{3}} + \frac{x^3}{3}$	$\sin x \cos x$ (hyp=89)	$\frac{243x^2}{2} + 12\log x$
3. Relative position representations:	$\frac{x^4}{4} - x\cos x + \sin x$	$-8x^6 + \frac{x\sin(x)}{20} + x$	$\sin x \cos x$ **(hyp=1)**	$12345x^2 + 6789\log x$
4. Tree relative attention:	$\frac{x^4}{4} - x\cos x + \sin x$	$-8x^6 + \frac{x\sin(x)}{20} + x - \frac{\cos(x)}{20}$	$\sin x \cos x$ (hyp=9)	$12345x^2 + 6789\log x$
	beam width 1	*beam width 1*	*beam width 100*	*beam width 10-100*
Gold:	$\frac{x^4}{4} - x\cos x + \sin x$	$-8x^6 + \frac{x\sin(x)}{20} + x$	$\sin x \cos x$	$12345x^2 + 6789\log x$

Figure 4.3: Prediction analysis

We present a qualitative analysis of the implemented methods in fig. 4.3. The models are trained on the FWD dataset for 120M steps. There are four different equations displayed. In the leftmost column, one can find a simple example. All the methods solve it correctly with a greedy approach (i.e., beam search with width 1). It meets our expectations and proves that all the presented models generalize to the data outside of the training set. The second example from the left is more complex. When one integrates the second term by the chain rule, the expected result is $\frac{x\sin x}{20} + \frac{\cos x}{20}$. On the other hand, the outcome of the model lacks $+\frac{\cos x}{20}$. Due to that $-\frac{\cos x}{20}$ which is the integral of the third term is not canceled out. While the tree positional encoding method is completely lost, it is clear that the first and fourth methods "forget" to cancel out. We note that this is not systematic, rather an uncommon example. The next integral is solved perfectly with beam width 100. However, the number of hypotheses required to make a correct prediction varies significantly from method to method. We have found no

consistent patterns for the second and third samples, but we address it to future research.

The rightmost sample is the most demonstrative one (fig. 4.3). The baseline model tends to fail in the five-digit number by mixing the order of tokens even with beam width of 100. More than 90% of the hypotheses are unable to predict the number in the first term. Thus, our empirical finding is as follows: relative methods significantly improve the ability of the model to work correctly with longer numbers. To start with, one should remember that all the multi-digit numbers in the dataset are presented as a sequence of tokens. Thus, the longer the number is, the harder it is for the model to work with. While the baseline model fails, the methods relying on adding structural information to the attention mechanism usually get the correct answer within the first 10 hypotheses. Even though there are not many examples with long numbers in the test set, we check this finding on several handcrafted examples and claim the difference between methods consistent.

4.4 Additional experiments

While trying to understand how the methods differ, we form two additional hypotheses.

The first hypothesis is about the lengths of the equations. Additional information about the equations' structure may lead to preferred results on longer sequences. To find proof, we look at the samples of the FWD test set where one method works better than the others. We have calculated the mean lengths of both correctly and falsely predicted questions for each model. As a result, mean sizes are almost identical across all models, precisely lower for accurately solved equations and greater for the incorrect ones (27-28 tokens for question and 88-95 tokens for answers). We conclude that there is no connection.

Another checked hypothesis is the dependency on the class of the functions in the integrand. Again, we go through samples of the test set where one method

works the best. We have examined 4 function classes separately: trigonometric, inverse trigonometric, hyperbolic, and inverse hyperbolic. The idea behind this is that one of the models can learn how to treat specific class functions better or worse than others. Overall, we have not found any objective patterns.

5 Future plans

The core objective for the future is to find out why the research outcomes are indistinguishable. We believe that it is due to the abnormal amount of the training data and hope that better managing dataset and model sizes will change the situation. Moreover, we have discovered promising findings described in section 4.3 that require future investigation and experiments. Speaking about improving the quality of the Transformer for symbolic mathematics in general, we suggest applying large pretrained models to the task (e.g., BERT Devlin et al. 2018) because it has recently shown promising results in a lot of sequence-processing tasks.

6 Conclusion

In this work, we address the question of whether the Transformer architecture can benefit from using the structure of the sequences in solving symbolic expressions.

We have shown that there are ways to utilize the structure of the data and these methods to aid. Thus, we:

- experiment with positional embeddings
- apply relative position representations to all the attention types
- modify self-attention with tree relative attention
- replace vanilla positional encodings with tree positional encodings

We have conducted extensive experiments on two separate symbolic mathematics tasks, specifically integration and solving ordinary differential equations. Moreover, we have investigated the deduplication effect and performed a qualitative analysis of the trained models.

We have empirically observed the absence of a statistically significant difference between the base Transformer architecture and expanded versions. One possible explanation is that the training data size is too large; thus, the model does not need prior information about the tree structure. Analysis of how the data and model sizes affect the performance of various methods is an interesting direction for future work.

We have conducted qualitative analysis and additional experiments to find the strengths and weaknesses of every model. Thus, we have found out that tree-structure-based approaches work better with long numbers. At the same tine, we analyzed the specifics of the task and underlined the importance of incorporating encoder-decoder attention with the knowledge of the structure of the expressions. We have released the source code for our work for the convenience of further collective work.

Bibliography

1. Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. "Combining symbolic expressions and black-box function evaluations in neural programs". In: *arXiv preprint arXiv:1801.04342* (2018).
2. Nadezhda Chirkova and Sergey Troshin. "Empirical study of transformers for source code". In: *arXiv preprint arXiv:2010.07987* (2020).
3. Ernest Davis. "The use of deep learning for symbolic integration: A review of (Lample and Charton, 2019)". In: *arXiv preprint arXiv:1912.05752* (2019).
4. Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).
5. Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.
6. Seohyun Kim et al. "Code prediction by feeding trees to transformers". In: *arXiv preprint arXiv:2003.13848* (2020).
7. Guillaume Lample and François Charton. "Deep learning for symbolic mathematics". In: *arXiv preprint arXiv:1912.01412* (2019).
8. Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.
9. Kishore Papineni et al. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.
10. Bartosz Piotrowski et al. *Can Neural Networks Learn Symbolic Rewriting?* 2020. arXiv: 1911.04873 [cs.AI].
11. Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

12. David Saxton et al. “Analysing mathematical reasoning abilities of neural models”. In: *arXiv preprint arXiv:1904.01557* (2019).

13. Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. “Self-attention with relative position representations”. In: *arXiv preprint arXiv:1803.02155* (2018).

14. Vighnesh Leonardo Shiv and Chris Quirk. “Novel positional encodings to enable tree-based transformers.” In: *NeurIPS*. 2019, pp. 12058–12068.

15. Kai Sheng Tai, Richard Socher, and Christopher D Manning. “Improved semantic representations from tree-structured long short-term memory networks”. In: *arXiv preprint arXiv:1503.00075* (2015).

16. Ashish Vaswani et al. “Attention is all you need”. In: *arXiv preprint arXiv:1706.03762* (2017).