Can Neural Networks Distinguish High-school Level Mathematical Concepts?

Sebastian Wankerl*†, Andrzej Dulny*, Gerhard Götz†, Andreas Hotho*
* CAIDAS, Julius-Maximilians University of Würzburg, Germany
{wankerl,dulny,hotho}@informatik.uni-wuerzburg.de
† Baden-Württemberg Cooperative State University, Mosbach, Germany
gerhard.goetz@mosbach.dhbw.de

***Abstract*—Processing symbolic mathematics algorithmically is an important field of research. It has applications in computer algebra systems and supports researchers as well as applied mathematicians in their daily work. Recently, exploring the ability of neural networks to grasp mathematical concepts has received special attention. One complex task for neural networks is to understand the relation of two mathematical expressions to each other. Despite the advances in learning mathematical relationships, previous studies are limited by small-scale datasets, relatively simple formula construction by few axiomatic rules and even artifacts in the data. With this work, we aim at overcoming these limitations and provide a deeper insight into the representation power of neural networks for classifying mathematical relations. We introduce a novel data generation algorithm to allow for more complex formula compositions and fully include mathematical fields up to high-school level. We research several tree-based and sequential neural architectures for classifying mathematical relations and conduct a systematic analysis of the models against rule-based as well as neural baselines with a focus on varying dataset complexity, generalization abilities, and understanding of syntactical patterns. Our findings show the potential of deep learning models to distinguish high-school level mathematical concepts.**

***Index Terms*—learning mathematical relations, mathematical syntax understanding, deep learning**

I. Introduction

Mathematics can be considered as a symbolic language for expressing universal concepts that can either originate from mathematics itself or express concepts in more applied disciplines like deep learning, chemistry, or physics [1]. Similar to every natural language, there is not just one mathematical formulation of a certain concept but it can be written down in endless, syntactically distinct, but mathematically equal ways. While in a natural language one can express the same statement by varying wording and sentence structure, in mathematics one can exchange operators, symbols, and constants. As a very elementary example, one can consider the functions $f(x) = x + x$ and $g(x) = 2 \cdot x$ which both describe the same operation, namely the doubling of their inputs.

Apart from equal formulations of the same concept, mathematics also allows to model more sophisticated relations. One of those is the derivative. Considering the function $f(x)$ defined above, its (most simplified) derivative is $f'(x) = 2$. It is, of course, possible to reformulate this derivative in infinitely many equal ways.

For human beings, the difficulty of recognizing the equivalence or other relationships between a given pair of mathematical expressions highly depends on their complexity. While one can argue that above examples are very trivial for someone who has at least a basic mathematical education, the relationship between more complex terms is not as easily recognizable. Hence, researchers developed computer algebra systems (CAS) which, among other tasks, calculate the equality of two expressions or their derivatives [2], [3].

However, these systems typically employ a large number of rule-based algorithms (e.g. [4]–[7]), each tailored to a very specific mathematical subproblem where the development of the algorithm itself is a complex and time-consuming task. Neural networks, on the other hand, have in recent years successfully been applied in domains like natural language processing [8], [9] or recommender systems [10], [11], where traditional machine learning or rule-based systems used to be dominant. Also in the field of mathematical problem solving, exploring the representative capabilities of neural networks to overcome the limitations of rule-based methods has become an active field of research as described in the following.

A. Related Work

We contribute to the field of mathematical relation classification which is subject to several existing works [12]–[14]. It requires the networks to explicitly recognize mathematical patterns instead of generating one single sample solution for a given input. For example, all pairings of the expressions x^2, $x \cdot x$, and $x \cdot (0 + x)$ should be recognized as equivalent whereas the pairs x^2 and $2 \cdot x$ as well as x^2 and $2 \cdot (x \cdot 1)$ should be identified as the derivative.

Arabshahi et al. [12] first evaluate neural networks on the binary classification task of mathematical equivalence using a dataset they previously introduced [15]. They apply neural networks working on textual representations of mathematical expressions as well as recursive neural networks like TreeSMU. Their evaluation shows that representing the mathematical expressions as trees is beneficial compared to text. Their work is further extended by Mali et al. [13] who introduced higher-order tree-recursive neural networks. However, the models proposed in both works do not scale up to larger datasets since they require recursive function calls which consequently impede fast data-parallel GPU computations.

Moreover, the quality of the dataset used in these works is criticized [16]: First, the set of axioms used as basis for generating more complex expressions is arbitrary and does not fully cover any mathematical field. For example, the distributive law is missing. Second, the generated negative examples tend to be longer and more nested compared to the positive examples and, most importantly, some mathematical functions can even only appear in the negative examples. These biases make the usefulness of this dataset and the obtained results unconvincing since the networks are not required to learn mathematical patterns but can exploit unrelated information.

A new dataset which averts above points of criticism is presented in our previous work [14]. This dataset is based on another set of axioms sufficient to generate arbitrary multivariate polynomials with up to three variables and constants from the set of $\{-4, \dots, 4\}$. Moreover, we included two additional mathematical relations namely derivative and variable substitution. Still, the sole focus on polynomials leaves many mathematical fields and more complex formulas unexplored. In addition, with approximately $32,000$ samples, this dataset is of rather limited size. It is referred to as *polynomial dataset* in the following.

B. Contributions

In this work, we address the limitations of previous studies. First, we explore additional axioms for exponential and logarithmic calculus as well as the field of trigonometry to cover all relevant mathematical concepts up to high-school level. We additionally augment the data in terms of set size and mathematical relations to get a better insight into the representation capabilities of neural networks on this task. Overall, we provide a large scale dataset to evaluate the task of mathematical relation classification and make the dataset as well as the data generator publicly available[1].

Second, we research the efficiency of several neural network architectures that seem promising but remained so far unacknowledged for the task of mathematical relation classification. Inspired by the success of other tree-based architectures on this task [14], we analyze both the TreeTransformer (tTf) [17] as well as the tCNN model [18], a convolutional neural architecture which is able to leverage tree-structured data and has shown promising performances in other research areas [18], [19]. We also incorporate a standard sequence based Transformer (Tf) model [20] and the BERT model [9], as both proved to be successful in other mathematical areas [21].

Third, we carry out an in depth analysis of the representation capabilities of neural networks for mathematical relation classification. Investigations on the impact of the training dataset size, the extrapolation capabilities to deeper formula trees and specific tests on formula syntax understanding show the potential of neural relation solvers.

II. Data

This section discusses our data and its generation.

[1]Find our code at https://github.com/LSX-UniWue/neural-highschool-math

TABLE I
EXAMPLES FOR OUR TASK: CLASSIFY THE RELATION BETWEEN TWO MATHEMATICAL EXPRESSIONS (e_1 AND e_2) AS ONE OF EQUIVALENT, DERIVATIVE, SUBSTITUTION, CONSTANT OFFSET, AND UNRELATED.

e_1	e_2	label
$2x + xy$	$x(2 + y)$	equivalent
$xy - 3x$	$y - 3$	derivative
$z(x + y)$	$(y + z)x$	substitution
$xy - 3$	$2 + yx$	constant offset
$1 + zx$	$z + 2x$	unrelated

A. Datasets

We introduce two datasets of different mathematical scope. Both datasets contain pairs of mathematical expressions and a label that defines the relation between them. Hence the data can be described as triples of the form (e_1, e_2, l) where e_1, e_2 denote mathematical expressions and l denotes the class label.

The first dataset (*trigonometry dataset*) contains mathematical expressions using the basic arithmetic operations, i.e. addition, subtraction, multiplication, division, and power, as well as the natural logarithm and the integers from -4 to 4 including zero. In addition, the trigonometric functions $\sin$, $\cos$, and $\tan$ as well as the constants π and e are used for its generation. It includes five different mathematical relations (see table I for examples):

1) equivalence: the two terms are mathematically equivalent for all positive values of the free variables, i.e. $\forall x, y, z > 0 : e_1 = e_2$
2) derivative: e_1 is the derivative of e_2 w.r.t. the first free variable x, i.e. $\frac{\partial}{\partial x} e_2 = e_1$
3) constant offset: the two terms e_1 and e_2 only differ in a constant c, that is $\forall x, y, z : e_1 - e_2 = c$.
4) substitution: there exists a permutation $\sigma \in S_3$ of variables x, y and z such that, after substituting the variables in e_1, it is equivalent to e_2, that is $\exists \sigma \in S_3 \forall x, y, z : \sigma(e_1) = e_2$
5) unrelated: none of the above relations apply

Naturally, the expressions in all classes are not simplified. For example, the simplest derivative of $x \cdot x$ would be $2x$, but also $3x - x$ is a valid derivative that can occur in the dataset.

The second dataset (*syntax dataset*) only contains two mathematical relations (equal and unrelated) and is constructed such that the models can not exploit the token distribution by basically reshuffeling the elements of a given tree (details see section II-C).

The datasets are created starting from a set of predefined axioms, as explicitly described in section II-B. For that, we use the axioms introduced in our previous work [14] and extend them with axioms of powers, logarithms, and trigonometry. All expressions can contain up to three free variables x, y, z.

Following previous research [12]–[14] we consider the depth of the parse tree of the mathematical expression as a measure of its complexity. In such a parse tree, the leaf nodes correspond to the symbols in an expression whereas the non-terminal nodes correspond to operators and functions. Hence,

the deeper the tree, the more operators and functions have been applied successively. See fig. 1 for examples of parse trees.

For our main experiments, we create samples of a depth up to 10 and discard those which are of higher depth. Since we hypothesize that the size of the data used in previous research is a limitation, we increase the number of samples in our new datasets. Precisely, the trigonometric dataset contains 8,208,926 training samples, 82,918 validation samples, and 83,756 test samples. The syntax dataset contains 2,956,662 training samples, 29,865 validation samples, and 30,166 test samples. The datasets were split randomly. For our ablation study, we also create an additional trigonometry test set consisting of 70,000 samples of depth 11-12.

B. Data Generation of Trigonometry Dataset

We developed a data generation algorithm that works in four major steps, where the first two steps of the algorithm are alike to the data generation algorithm described in [14]. To generate datasets of above size, we run it simultaneously on multiple machines. Subsequently, we merge the generated data and remove the duplicates.

1) Axiom Substitution: We start with a set of 57 predefined mathematical axioms (e.g. $x+0=x$). To generate increasingly more complex expressions based on these axioms, randomly selected free variables in an axiom are first substituted with another expression. The expressions constructed in this way are then added to the set of axioms.

2) Equal Transformation: As a next step, we sample one of the previously generated expressions and then find a matching axiom and apply it. For this step, we can substitute variables in the axioms with parts of the expression. We then search all possible matches between the expression and the axiom and apply it to one of the matches.

3) Creation of Equivalent Sets: As a next step, we partition all previously generated expressions $e_1, \dots e_n$ into sets $S_1, \dots, S_m$ of equal expressions, such that $\forall i, j, k : e_i \in S_k \wedge e_j \in S_k \Leftrightarrow e_i = e_j$. To determine the equivalence we sample ten random real numbers $a_1, \dots, a_{10}$, $b_1, \dots, b_{10}$, $c_1, \dots, c_{10}$ between 10^{-5} and 2 for every free variable x, y, z, respectively. If two expressions e_1, e_2 evaluate to the same result for all sampled values, i.e. $\forall i : e_1[x/a_i, y/b_i, z/c_i] = e_2[x/a_i, y/b_i, z/c_i]$ we consider these two expressions as equivalent. We evaluate the expressions to complex numbers since this allows us to use negative numbers as arguments to logarithms. Mathematically invalid samples are removed.

4) Sampling: In the final step, we create pairs of expressions for all of the relations we consider.

a) Equal Expressions: To create pairs of equal expressions we iterate over the equivalent sets and randomly sample pairs from each set.

b) Derivatives: To create a pair of derivatives we sample an expressions from one of the equivalent sets. Then, we calculate its derivative using Sympy [3]. If it is equal to one of the equivalent sets, we sample an expression from that set and include it as the derivative in the sample. Otherwise we transform the derivative multiple times (cf. section II-B2).

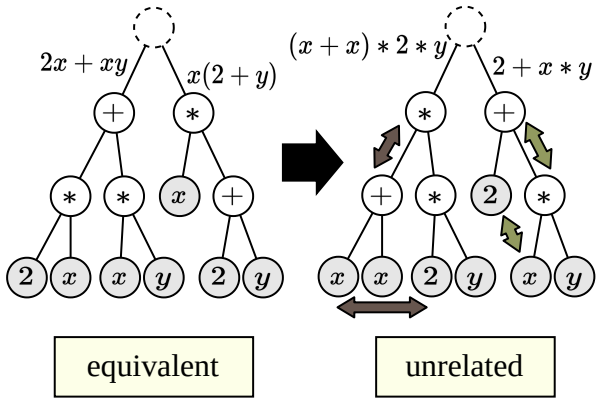

Fig. 1. Example samples from the syntax dataset. The expression pair to the right has been generated by permuting the leaf nodes and operators separately for each side (marked by the arrows $\leftrightarrow$ and $\leftrightarrow$).

c) Constant Offset: Here, we iterate over all pairings of the equivalent sets and, similar to section II-B3, check if the difference between the elements of the classes evaluates to the same constant. If this is the case, we sample one expression from each set and add it as a pair to the dataset. However, we exclude all samples where one class always evaluates to 0 as this would overlay with the class of derivatives.

d) Permutation: This is similar to sampling the equivalent expressions. Yet, instead of just adding the equal expressions, we randomly substitute the variables in one of them, for example by replacing all x with y, y with z and z with x.

e) Unrelated: To create this class we either sample pairs of expressions from two different equivalent sets or sample a pair of expressions from the same set and modify one of them such that they are not equivalent anymore. The latter could be achieved in two ways. We either replace the value of a node in the tree, like replacing a multiplication with an addition or one constant with another. Alternatively, we remove a branch from the tree. Since this procedure could by accident generate a pair that falls into one of the other classes, we ensure that none of these relations apply using above described tests.

C. Data Generation of Syntax Dataset

Finally, for measuring the ability of neural networks to classify mathematical relations without any indication of the token distribution, we generate a separate binary *syntax dataset*, where each equivalent formula corresponds to exactly one unrelated sample. Precisely, these two samples should neither be distinguishable by the frequency of their consisting tokens, nor by the structure of their corresponding trees.

We obtain this dataset by selecting equivalent samples $e_i = e_j$ constructed as above and shuffling the tokens of e_i and e_j to obtain a non-equivalent sample $\tilde{e}_i \neq \tilde{e}_j$ consisting of the same tokens and having the same tree-structure (cf. fig. 1). To make sure that the newly generated expressions are syntactically valid, we separately permute the leaf nodes of the expression tree (variables and constants), the binary operators and the unary functions. Since random shuffling

TABLE II
ACCURACY AND F_1 SCORES OF THE TREETRANSFORMER (TTF) COMPARED WITH WANKERL ET AL. [14]. IN EACH COLUMN WE BOLDED OUT THE BEST PERFORMING MODEL AND UNDERLINED THE SECOND-BEST.

Model	Acc	F_1			
		Eqiv	Deriv	Subst	Unrel
tRNN	0.754	0.669	0.820	0.817	0.689
tLSTM	0.832	0.857	0.830	0.940	0.726
tSMU	0.838	0.860	0.840	0.950	0.740
tCNN	0.657±0.007	0.600±0.004	0.770±0.008	0.660±0.011	0.613±0.012
Tf	0.676±0.005	0.622±0.003	0.833±0.003	0.707±0.005	0.476±0.003
tTf	0.832±0.004	0.773±0.003	0.886±0.005	**0.962±0.009**	0.742±0.004
Sympy	**0.914**	**0.888**	**0.946**	0.865	**0.999**
BoW	0.412	0.268	0.358	0.502	0.295

TABLE III
ACCURACY AND F_1 SCORES (MEAN ±STDEV) OF THE MODELS FOR THE TRIGONOMETRY DATASET. IN EACH COLUMN WE BOLDED OUT THE BEST PERFORMING MODEL AND UNDERLINED THE SECOND-BEST.

	Model	Acc	F_1				
			Unrel	Eqiv	Deriv	Subst	Offset
Shallow	BoW	0.634±0.005	0.498±0.002	0.682±0.001	0.532±0.001	0.757±0.001	0.660±0.002
	Sympy	0.672	0.610	0.804	0.730	0.810	0.512
	BERT	**0.970 ±0.001**	**0.948±0.002**	**0.947±0.001**	**0.980±0.000**	**0.991±0.000**	**0.976±0.001**
	Tf	0.793±0.012	0.779±0.087	0.732±0.025	0.814±0.016	0.902±0.008	0.779±0.011
	tCNN	0.947±0.022	0.895±0.039	0.943±0.015	0.970±0.022	0.977±0.008	0.954±0.024
	tTf	0.893±0.002	0.860±0.004	0.861±0.003	0.897±0.002	0.977±0.001	0.891±0.003
Deep	BoW	0.635±0.005	0.680±0.005	0.542±0.004	0.757±0.005	0.658±0.005	0.496±0.003
	Sympy	0.670	0.729	0.730	0.733	0.500	0.640
	BERT	**0.908 ±0.004**	**0.858±0.006**	**0.908±0.001**	**0.958±0.006**	**0.986±0.001**	**0.813±0.014**
	Tf	0.743±0.013	0.641±0.015	0.753±0.009	0.843±0.008	0.891±0.013	0.508±0.032
	tCNN	0.853±0.043	0.788±0.054	0.861±0.025	0.947±0.033	0.958±0.017	0.649±0.141
	tTf	0.847±0.012	0.798±0.015	0.857±0.002	0.884±0.010	0.980±0.002	0.677±0.073

could potentially generate another equivalent expression, we check if the permuted expression is indeed not equivalent to the original expression. If this criterion is met, we add (e_i, e_j) and ($\tilde{e_i}$, $\tilde{e_j}$) as a positive resp. negative example to the dataset.

III. EXPERIMENTS AND DISCUSSION

In this section we discuss our experiments and obtained results for all datasets. Moreover, we present the results of our ablation study, namely the extrapolation of the models to unknown depth.

A. Models

In our study, we explore several model architectures that were so far not researched for the task of mathematical relation classification. For all models, we optimize the hyperparameters using Optuna [22] and train them using the Adam optimizer with $lr = 1e-4$. We evaluate two models making use of tree-structured input: the tCNN [18] and the TreeTransformer [17]. For the tCNN, we optimized the number of layers l_c and filters in each layer, the size of the input embeddings E_i, the output size of the tree representation t, and the number of layers l_f and their size s of the fully-connected layer block subsequent to the convolutions. The best configuration uses ($l_c = 7, E_i = 32, t = 128, l_f = 3, s = 64$). For the TreeTransformer, we explored configurations varying the number of encoder and decoder layers l_e, l_d, the sizes of the embeddings E and the feed-forward layers l_f, the number of attention heads h, and the dropout values d. The best configuration uses ($l_e = 3$, $l_d = 6$, $E = 256$, $l_f = 512$, $h = 4$, $d = 0.025$).

Additionally, we finetune the BERT-base [9] encoder on our task to evaluate the application of general purpose language models for mathematical relation classification. Last, we include a standard transformer encoder-decoder model in our experiments to be able to quantify the benefit of the tree-structured design of the TreeTransformer model as opposed to the standard sequential approach. Hereby, we also optimize its hyperparameters, similar to the TreeTransformer. The best configuration uses ($l_e = 1$, $l_d = 1$, $E = 256$, $l_f = 1024$, $h = 16$, $d = 0.28$). We train the transformer model to output one single token corresponding to the class label of the input expression. The expressions are given in prefix, i.e. Polish, notation. This notation has proved to be beneficial on other symbolic mathematical tasks [21], [23] since it reflects a flattened representation of the parse tree and hence represents the precedence of each part of the input in a compact way.

So far, related work did not assess whether the proposed models are able to outperform existing rule-based systems or even simple neural network approaches for the task [12]–[14]. To this end, we include both the rule-based and freely available CAS Sympy [3], as well as a neural bag-of-words model which is able to capture simple statistical token distributions from the input. The baselines are incorporated in our experiments as described in the following.

1) Sympy: To apply Sympy, we parse the expressions with its `sympify` method, configured to already simplify the equation as much as possible. Moreover, while parsing we pass the assumption that all variables are positive, i.e. $x, y, z \in \mathbb{R}_{>0}$. This is necessary since some equalities only hold for positive assignments of the variables and thus this information helps Sympy to correctly simplify expressions.

Given two parsed expressions p_1 and p_2 we check their relations as follows. If the difference $p_1 - p_2$ equals 0 we assign them the label equivalent. Otherwise, if the difference does not have any free variables, we assign them to the constant offset class. Subsequently, we let Sympy calculate the derivative of p_2 and check if it is equivalent to p_1 like above. As a last step we calculate all variable permutations of p_2, and check if any of them equals p_1. If none of these conditions applies, we assign it to the class of unrelated examples.

2) Bag-of-words: Additionally, we use a multilayer perceptron (MLP) trained on a bag-of-words representation of the mathematical expressions as a baseline in our experiments. For each sample, consisting of two mathematical expressions, we count the number of occurrences of each symbol from the vocabulary in either of the formulas and normalize it with respect to the total number of symbols. This generates a bag-of-words vectors of the formulas consisting of relative frequencies of each vocabulary token. This representation does

not contain any syntactical information about the expressions.

In our experiments we train a MLP classifier using the described bag-of-words representation of the formulas. The MLP consists of five hidden layers, each with size 256 and rectified linear units as nonlinear activation functions after each layer. The final layer calculates the logits for each class. The network is trained using the cross-entropy loss and a learning rate of 0.001 with the Adam optimizer.

B. Experiment 1: Polynomial Dataset

In this first experiment, we compare the so far disregarded TreeTransformer and tCNN approaches against existing tree-structured models from literature using the polynomial dataset [14]. The results are presented in table II. The TreeTransformer hereby outperforms the standard sequential transformer approach and yields a comparable accuracy to the models used in our previous work [14]. Precisely, the accuracy of the TreeTransformer is 83.2% which is only 0.6 percentage points (pp.) behind the best performing model of [14] (TreeSMU). Moreover, we observed that the TreeTransformer outperformed the TreeSMU on three of the four examined classes.

However, the rule-base Sympy baseline outperforms all neural models in this setting, an aspect which has not been examined in previous work. It is most likely due to the fact that the axioms used are of simple structure in a mathematical sense and hence can be easily covered by rule-based systems motivating our following experiments on more complex datasets. Another aspect is the limited size of the dataset that strongly influences the performance of neural approaches [24].

C. Experiment 2: Trigonometry Dataset

The performance of the models and the baselines on our newly introduced dataset is shown in the upper half of table III. We report both the overall accuracy of the models as well as the F_1 scores of the respective classes. We state the mean as well as the standard deviation obtained after 5 runs.

It is clearly visible that the finetuned BERT model outperforms all others by a large extend and that the bag-of-words baseline performs poorly. This observation holds for both the overall accuracy as well as for the F_1 score of each class. In total, the BERT model predicts 97.0% of the samples correctly which is approximately 30 pp. over the accuracy of Sympy.

The weak performance of the bag-of-words model compared to the other neural architectures supports the hypothesis that neural networks can exploit the structural patterns underlying the mathematical relationships. Nevertheless, it also indicates that the token distribution can in fact be used for classifying the mathematical relations clearly above the level of chance. We hypothesize that it is due a shift in the token distribution between the mathematical relations, as we will further elaborate when we discuss the separate F1 scores.

Considering the tree-based neural models, both of them outperform the transformer. This aligns with previous findings which also indicate that tree-structured input is beneficial for this task [12], [13] when training the models from scratch: The transformer model only receives a sequential representation of the input and thus has to deduce the grammar of mathematics from it. In contrary, the tree-based models are given a much stronger knowledge since the trees directly encode the link between operators and operands as well as functions and their arguments. While the transformer model only achieves an accuracy of 79.3%, the tCNN already achieves an accuracy of 94.7%. The TreeTransformer is slightly weaker, achieving an accuracy of 89.3%. Finally, all neural models outperform the rule-based Sympy which only achieves an accuracy of 67.2%.

Considering the different mathematical relations, all models perform best on the class of *substitution* with an F_1 score between of 90.2% for the transformer and 99.1% for BERT. It is also the relation where the bag-of-words baseline performs best (75.7% F1 score). This can be explained by a characteristic change in the token distribution of the expressions. Since all variables from one side are exchanged by another variable from the other side, this leads to a rather balanced frequency distribution of the three possible variables x, y, z.

The second most easy to recognize relation is the one of *derivatives* with F1 scores between 81.4% (transformer) and 98% (BERT). By contrast, the bag-of-words baseline performs particularly poor on this class (53.2%). We hence hypothesize that instead of an easily perceivable change in the token distribution, the derivation results in very prominent structural patterns which can only be recognized by the models that can learn mathematical syntax. Examples for such patterns are the product rule, the chain rule, or the derivation of powers.

Finally, we want to point out that the unrelated class is the hardest to identify for most neural models, including the bag-of-words. Since this class consists of samples where no well-defined mathematical relation holds, we assume it to be difficult because expressions do not follow a certain pattern.

D. Experiment 3: Mining Patterns on the Syntax Dataset

In this experiment we evaluate the models' capabilities of learning mathematical syntax. As described in section II-C, the syntax dataset is constructed in a way that models can neither use the token distribution nor the overall structure of the expressions when making a decision. Hence, this dataset only has the two classes *equivalent* and *unrelated*.

The results of this experiment are summarized in table IV. As expected, the bag-of-words model cannot learn anything

TABLE IV
ACCURACY AND F_1 SCORES OF THE MODELS ON THE SYNTAX DATASET. IN EACH COLUMN WE BOLDED OUT THE BEST PERFORMING MODEL AND UNDERLINED THE SECOND-BEST.

Model	Accuracy	F_1	
		Equal	Unrelated
Bag-of-words	$0.503_{\pm 0.001}$	$0.668_{\pm 0.001}$	$0.00_{\pm 0.000}$
Sympy	0.830	0.824	0.835
BERT	$0.975_{\pm 0.004}$	$0.975_{\pm 0.004}$	$0.975_{\pm 0.004}$
Transformer (Tf)	$0.889_{\pm 0.001}$	$0.885_{\pm 0.001}$	$0.893_{\pm 0.001}$
tCNN	$\mathbf{0.992_{\pm 0.000}}$	$\mathbf{0.992_{\pm 0.000}}$	$\mathbf{0.991_{\pm 0.000}}$
TreeTransformer (tTf)	$\underline{0.981_{\pm 0.001}}$	$\underline{0.980_{\pm 0.001}}$	$\underline{0.981_{\pm 0.001}}$

on this dataset and hence reaches only an accuracy of 50.3%. In contrary, all models that can principally learn mathematical syntax (BERT, transformer, tCNN, TreeTransformer) are able to recognize the syntactical patterns and outperform the rule-based Sympy baseline by a large extent: while Sympy only predicts 83% of the data correctly, the neural models reach accuracies between 88.9% (transformer) and 99.2% (tCNN).

For this task, both the tCNN as well as the TreeTransformer outperform the BERT model. We conclude that explicitly passing the parse tree of the expressions to the models is especially helpful when it comes to learning the syntax.

E. Generalization Capabilities: Evaluation on Deeper Trees

As a last experiment we evaluate whether the models still recognize mathematical patterns if the data they receive as input is of higher complexity than the data they were trained on. As discussed in section II, we use the depth of the parse tree as the measure of complexity. While the networks were trained on trees of a depth up to ten, they are now evaluated on depths of 11 and 12. We use the models trained on the trigonometry dataset for this experiment. The results are presented in the bottom half of table III.

All tested networks still outperform the Sympy baseline, although we notice a large decline in their performance. It is highest for the tCNN which looses 9.4 pp of performance compared to the shallow trees. BERT looses 6.2 pp. The most stable are the sequence-to-sequence models, both loosing only around 5 pp (transformer) and 4.6 pp (TreeTransformer). Hence, we conclude that the TreeTransformer learns the most stable representations for the mathematical patterns and can hence most successfully mine them in trees of unseen depths. The models which explicitly receive the tree structure of the expressions (TreeTransformer and tCNN) are still outperforming the transformer on sequential input data, however, the highest absolute performance can still be obverved for BERT.

IV. Conclusion

In the study, we explored the capabilities of various neural network models for the task of classifying mathematical relationships. We show that on a small dataset from literature built from few axioms, a rule-based CAS still outperforms neural networks and that the type-token distribution can be used for identifying the mathematical relationship with high accuracy.

Hence, we argue that previous datasets lack of size and complexity and propose a more complex data generation approach which includes additional relations and mathematical axioms. We showed that as the complexity of the dataset increases, the performance of the rule-based CAS decreases while the neural approaches improve. In further experiments we examined the ability of the models to learn the task of mathematical equivalence solely from the syntax and studied the ability to generalize to more complex mathematical equations.

References

[1] J. O. Bullock, "Literacy in the language of mathematics," *The American Mathematical Monthly*, vol. 101, no. 8, pp. 735–743, 1994.

[2] J. Von Zur Gathen and J. Gerhard, *Modern computer algebra*. Cambridge university press, 2013.

[3] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, 2017.

[4] D. H. Bailey, J. M. Borwein, and A. D. Kaiser, "Automated simplification of large symbolic expressions," *Journal of Symbolic Computation*, vol. 60, pp. 120–136, 2014.

[5] B. Buchberger and R. Loos, *Algebraic simplification*. Springer, 1982.

[6] A. H. Gebremedhin and A. Walther, "An introduction to algorithmic differentiation," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 10, no. 1, p. e1334, 2020.

[7] M. Bronstein, *Symbolic integration I: transcendental functions*. Springer Science & Business Media, 2005, vol. 1.

[8] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, no. 10, pp. 1872–1897, 2020.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[10] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[11] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *2018 IEEE international conference on data mining (ICDM)*. IEEE, 2018.

[12] F. Arabshahi, Z. Lu, P. Mundra, S. Singh, and A. Anandkumar, "Compositional generalization with tree stack memory units," *arXiv preprint arXiv:1911.01545*, 2019.

[13] A. Mali, A. G. Ororbia, D. Kifer, and C. L. Giles, "Recognizing and verifying mathematical equations using multiplicative differential neural units," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 5006–5015.

[14] S. Wankerl, A. Dulny, G. Götz, and A. Hotho, "Learning mathematical relations using deep tree models," in *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2021.

[15] F. Arabshahi, S. Singh, and A. Anandkumar, "Combining symbolic expressions and black-box function evaluations for training neural programs," in *International Conference on Learning Representations*, 2018.

[16] E. Davis, "A flawed dataset for symbolic equation verification," *arXiv preprint arXiv:2105.11479*, 2021.

[17] X.-P. Nguyen, S. Joty, S. Hoi, and R. Socher, "Tree-structured attention with hierarchical accumulation," in *International Conference on Learning Representations*, 2020.

[18] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[19] M. B. Kusharki, S. Misra, B. Muhammad-Bello, I. A. Salihu, and B. Suri, "Automatic classification of equivalent mutants in mutation testing of android applications," *Symmetry*, vol. 14, no. 4, p. 820, 2022.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[21] G. Lample and F. Charton, "Deep learning for symbolic mathematics," in *International Conference on Learning Representations*, 2019.

[22] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[23] S. D'Ascoli, P.-A. Kamienny, G. Lample, and F. Charton, "Deep symbolic regression for recurrence prediction," in *Proceedings of the 39th International Conference on Machine Learning*, 2022.

[24] A. Alwosheel, S. van Cranenburgh, and C. G. Chorus, "Is your dataset big enough? sample size requirements when using artificial neural networks for discrete choice analysis," *Journal of Choice Modelling*, vol. 28, 2018.