Structural information in mathematical formulas for exercise difficulty prediction: a comparison of NLP representations

Ekaterina Loginova
Ghent University
ekaterina.loginova@ugent.be

Dries F. Benoit
Ghent University
dries.benoit@ugent.be

Abstract

To tailor a learning system to the student's level and needs, we must consider the characteristics of the learning content, such as its difficulty. While natural language processing allows us to represent text efficiently, the meaningful representation of mathematical formulas in an educational context is still understudied. This paper adopts structural embeddings as a possible way to bridge this gap. Our experiments validate the approach using publicly available datasets to show that incorporating syntactic information can improve performance in predicting the exercise difficulty.

1 Introduction

Online learning platforms aim to provide personalised tutoring at scale using data-driven personalisation (Romero and Ventura, 2010). A key component of a personalised system is a recommendation algorithm that suggests the next learning activity. To ensure that the recommendation is tailored to the student's level and learning needs, not only should the student's ability level model be considered, but also the learning content characteristics, such as its difficulty. Learning content can contain multiple media types (images, text, formulas), each of which must be converted to a numeric format compatible with machine learning models. While natural language processing (NLP) and computer vision allow us to efficiently represent texts and images, the meaningful representation of formulas in an educational context is still understudied. This paper proposes a method for representing mathematical expressions (considered as a form of text) based on an structural embeddings and investigates its effectiveness in predicting exercise difficulty.

2 Related work

Research directions in mathematics can be broadly categorised into three branches: generation, assessment and solving. In each task, we need to represent a mathematical exercise that may include a text description, a formula, and a picture. The majority of works in this area focuses on word problems that can be represented as bag-of-words (John et al., 2015) (optionally with binary indicators of whether the word is a mathematical term). Such a representation allows the use of rich semantic taggers that provide additional information about lexical units, such as their degree of concreteness or associated emotions (Slater et al., 2016). However, semantic taggers are usually developed for general language use rather than for a specialised domain such as mathematics with its large variety of special characters. Previous work accommodated this by manually introducing additional mathematical symbols to be parsed (Slater et al., 2016) or by considering entire mathematical expressions as tokens, an approach called bag-of-expressions (Lan et al., 2015). However, such an approach ignores the order of mathematical symbols. A possible extension is to use n-grams to represent chunks of symbols, thus preserving partial information about their order (Jurafsky and Martin, 2009). Its downside is that it is limited by the chosen length of the n-gram and thus cannot fully account for deeply nested expressions. In short, these approaches to representing content tend to be simplistic and do not allow syntactic or semantic information to be fully exploited. Therefore, previous research suggests that hierarchical representations should be used to capture deep features and generate higher quality content features (Li et al., 2013), an observation that motivates our study.

In natural language processing, the next level of representation after n-grams is a parse tree of a sentence. It captures syntactic information by representing words as nodes connected by syntactic dependencies: for example, an adjective used as a modifier of a noun. Similar to a natural lan-

Proceedings of the 17th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2022), pages 101 - 106

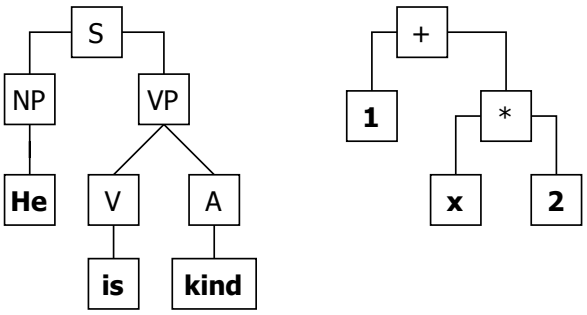

Figure 1: Left: a parse (constituency) tree for the sentence "He is kind" (simplified). Right: a parse tree for the mathematical expression `1 + 2*x`. Leaf nodes are in bold.

guage utterance, a mathematical formula can be represented by such a parse tree (Li et al., 2012). Algebraic trees have been successfully used to automatically solve algebra problems with template approaches (Roy and Roth, 2017; Huang et al., 2017). More recent approaches aim at generalisable solutions, for example, by using knowledge graphs (Zhao et al., 2019). Hierarchic representations have rarely been used for the task of predicting exercise difficulty. The approach closest to ours is using AST parse trees to analyse Python programs (Paaßen et al., 2021). However, mathematical formulas present their own unique challenges: for example, different formats for writing formulas, which can vary across datasets, even when the same typesetting system (e.g., LaTeX) is used. In addition, formulas use domain-specific alphabet and are much shorter than typical coding exercises.

This paper compares possible ways to encode syntactic information in mathematical exercises and adopts the structural embeddings approach to represent mathematical formulas (Liu et al., 2017). Each sentence is represented as a constituency parse tree. In such a tree for a natural language utterance, the non-terminal nodes correspond to grammatical relations (for example, `NP` stands for "noun phrase"), while the leaf nodes contain words. Each word is then represented as a sequence of nodes in a parse tree from its leaf to the root of the sentence. For example, if we want to represent the word "kind" in the sentence "He is kind", we construct the parse tree (Figure 1) and obtain the corresponding syntactic sequence `[S, VP, A]`. A representation obtained this way captures hierarchical information while facilitating the use of standard neural network models, e.g., an LSTM (Long-Short Term Memory) (Hochreiter and Schmidhuber, 1997). We refer the reader to the original paper for architectural details and intuition behind them. Generally speaking, it encodes a variable-length syntactic sequence into a fixed-length vector representation — the syntactic-semantic embeddings — and the final hidden state serves as input to a decision-making model.

3 Data

The experiments are conducted on two datasets: a recently released MATH (Hendrycks et al., 2021)[1] and a synthethic DeepMind Mathematics(Saxton et al., 2019)[2], one of the largest publicly available datasets for mathematics in educational data mining.

The DeepMind dataset consists of question and answer pairs, and each pair has a label corresponding to the difficult level, easy or hard. An advantage of this dataset is its clear and unified formatting: exercise often have a consistent phrasing that mostly differs on a formula step, which allows us to have a clearer comparison of how effective different formula representations are. For our experiments, we used a subset of 37 244 problems, covering a broad range of topics: algebra, arithmetic, calculus, comparison, numbers, polynomials. 18 608 problems are of high level and 18 636 are of low. Word descriptions are in English and formulas are not specifically separated but can be extracted using regex-based approach due to limited variability of the rest of the sequence.

The MATH dataset contains 12 498 problems from mathematics competitions in the US. Problems are labelled with five levels of difficulty (1001, 2242, 2723, 2904 and 3628 problems, respectively) and cover the following topics: Algebra, Counting & Probability, Geometry, Intermediate Algebra, Number Theory, Prealgebra, Precalculus. Word descriptions are in English and formulas are written in LaTeX and defined by $ operators.

4 Methodology

4.1 Data representation

As mentioned above, each exercise contains a textual description and a formula. For example, it can be the following task: `Calculate sqrt(121) - sqrt(36).` In our case, a parse tree can be extracted with open-source libraries, such as AST

[1] https://github.com/hendrycks/math
[2] https://github.com/deepmind/mathematics_dataset

and SymPy[3]. A notable challenge at this step is the wide variety of notation conventions that renders converting a formula without errors a non-trivial problem. For example, differentiation can be written using `f\^{\prime}(x)` or `$f'(x)$`. Quite often, multiplication symbols are omitted or individual symbols are encoded; there might be several pieces of formula expressions per exercise. As a result, running a popular converter latex2sympy[4] on the MATH dataset results in only 1673 correctly parsed formulas out of 12500 (13% success rate). While natural language processing tasks such as tokenisation are well-explored and a plethora of high-quality public solutions exist, there appears to be no robust package. Thus we have developed processing scripts for mathematics.

A formula is pre-processed so that all numbers are replaced with a special `NUM` token (alternative per digit replacement did not seem to alter the results). It is important to consider differences in input types, as it prompts adjustments to the tokenisation procedure: for example, for AST parses and formulas, we need to consider a broader range of special symbols as separators (e.g., `(=)*/+-.\'^{}`) to avoid contaminating the vocabulary with too complex tokens that are actually sub-pieces of large expressions. `log` and `power` are transformed using regular expressions to act as functions accepting multiple arguments: `(a-1)\^\{3\}` becomes `power(a-1, 3)`. Decorative commands like `mathbb` are removed. Operators are also converted into their programming language equivalent (e.g., `\\neq` is replaced with `!=`) and a rule-based processing script unifies the notation by for example transforming different fraction encodings such as LaTeX's `\\frac{}{}` into `()/()`. Some tasks also include systems of formulas — while it is possible to try and represent them with special joining operators, in this study, we opted to use the longest correctly parsed formula. As a result, we obtain a more programmatic representation of a formula that drastically improves parsing correctness (7298 correct out of 12 498, 58%). We then construct a parse tree of mathematical expression and represent leaf nodes with their syntactic sequences (paths to the root). Parsing is done by either 1) using AST parser and NodeVisitor; or 2) using topological sorting on networkx[5] graph — and subsequently finding the shortest path with built-in library functions. As an example output, `x` from an algebraic expression `1 + 2*x` would be represented as `[*, +]`.

In resulting nested sequences, each formula term is represented as a syntactic sequence of nodes to the root of the syntax parse tree, and an entire formula then comprises a sequence of terms: `[[Add, Integer], [Add, Integer]]`. It is possible to simplify this representation by flattening sequences and concatenating them into a single string with an arbitrary separator as follows: `Add Integer . Add Integer`. A flattened sequence is a simplified representation of syntax information for which we can use more traditional methods, such as bag-of-words or vanilla LSTM.

In the end, we work with four types of exercise content: textual description (`Calculate`), raw formula text (`sqrt(121) - sqrt(36)`), and formula syntactic sequences (nested or flattened). They can be used independently as the only input to the classification model or combined. More details are provided in the following subsection.

4.2 Prediction models

We investigate the effectiveness of the proposed content representation by using them to estimate the difficulty of exercises. For the DeepMind dataset, it is a binary classification problem since the model must predict whether the exercise comes from an easy or hard level. For MATH, it is a multiclass classification problem with five classes (a range of levels from 1 to 5).

Our first model type is a vanilla LSTM that uses only one input source at a time, e.g., only the textual description. If we want to add syntax information to this model, it must be a string and can then be concatenated with the rest directly with an arbitrary separator (a space in our case). The second type is a multi-input modification that processes two different input types in a more nuanced manner similar to an idea of Siamese architectures in automated question answering domain: it passes them to individual submodels, and concatenates the output representations to feed into a feed-forward layer with softmax output for the final classification decision. This is motivated by different alphabet and structure of the sources: we hypothesise that it might be easier for the network

[3] https://www.sympy.org/en/

[4] https://github.com/augustt198/latex2sympy

[5] https://networkx.org/

Model 1: single input	DeepMind	MATH	Model 2: multiple input	DeepMind	MATH
Description	0.66	0.71	Formula & Description	0.66	0.72
Description + Formula	0.69	0.73	AST parse & Description	0.66	0.72
Formula	0.58	0.68	AST rootpaths (flat) & Description	0.66	0.72
AST parse	0.56	0.67	Sympy rootpaths (flat) & Description	0.66	0.72
AST rootpaths (flat)	0.58	0.66	Model 3: single input with structural embeddings	DeepMind	MATH
Sympy rootpaths (flat)	0.64	0.66	AST rootpaths	0.58	0.65
Description + AST rootpaths (flat)	0.72	0.72	Sympy rootpaths	0.6	0.65
Description + Sympy rootpaths (flat)	0.71	0.72	Model 4: multiple input with structural embeddings	DeepMind	MATH
Description + AST pars	0.7	0.72	AST rootpaths & Description	0.72	0.73
			Sympy rootpaths & Descriptions	0.73	0.73

Table 1: 10-fold cross-validated ROC AUC. + corresponds to concatenating the input strings, & to adding a separate input layer to the network. Best results are highlighted in bold. We can see that adding syntax sequences improves the performance on DeepMind dataset.

to learn if mathematical expressions and the natural language representation are disentangled. While the described models operate on 2-dimensional data, the third type of model works with nested root path sequences as described above to obtain syntactic formula embeddings and therefore uses 3-dimensional input. It includes time-distributed wrappers to apply identical embedding and feature engineering layers to each term. Again, we can add another input that can work with conventional flat sequences and concatenate the resulting embeddings to make a classification decision, leading to the fourth and final model type.

5 Experiments

Neural networks were implemented in Tensorflow with Keras (Chollet et al., 2015) and trained on Google Colab Pro GPUs. We used early stopping, monitoring validation loss with the patience of 3 epochs.

5.1 Results

We compare data representations to investigate whether adding syntactic sequences improves classification performance. Performance was evaluated using 10-fold stratified cross-validation ROC AUC and is shown in Table 1. Regarding the baselines, majority and random baselines produce ROC AUC of 0.5 on a single run, and the best results of logistic regression models trained on the length of input sequences are 0.57 for MATH (on descriptions) and 0.66 for DeepMind (on formula), respectively.

Regarding other possible neural approaches to feature engineering, using word2vec algorithm (Mikolov et al., 2013) to produce pretrained embeddings, contrary to our expectations, did not improve our results. We have also experimented with the graph embedding method node2vec (Grover and Leskovec, 2016), but the individual formulas prove to be too shallow for the approach to produce a meaningful representation. A promising direction is to use graph neural networks. Preliminary experiments with Graph Convolutional Networks (Kipf and Welling, 2017) using Spektral[6] on DeepMind dataset led to an improvement from 0.79 to 0.81 of a single-run accuracy score, but in this study for the rest of this section we continue to focus on structural embeddings extracted with LSTMs. Considering individual inputs, the parse tree representation alone, whether flat or nested, could not outperform the other models because the word description dominates it. Interestingly, the AST root paths are on par with the raw formula, and the SymPy root paths outperform it on the DeepMind dataset. Using nested syntactic sequences

[6]https://graphneural.network/

Exercise topic	D	F	SRP-F	D + F	D + SRP-F
numbers__is_factor_composed	0.64	0.53	*0.54*	0.63	*0.68*
algebra__linear_1d_composed	0.77	0.50	*0.52*	0.82	0.81
numbers__is_prime_composed	0.59	0.54	*0.56*	0.65	0.63
numbers__list_prime_factors_composed	0.68	0.53	*0.57*	0.66	*0.73*
arithmetic__add_sub_multiple	0.46	0.77	0.69	0.77	0.75
polynomials__simplify_power	0.51	0.86	0.78	0.86	0.84
polynomials__collect	0.50	0.56	*0.64*	0.59	*0.69*
numbers__round_number_composed	0.53	0.53	*0.59*	0.52	*0.53*
numbers__place_value_composed	0.80	0.56	*0.58*	0.76	*0.81*
calculus__differentiate	0.83	0.54	0.52	0.86	0.86
comparison__pair_composed	0.77	0.58	*0.61*	0.73	*0.78*
polynomials__coefficient_named	0.48	0.56	*0.57*	0.53	*0.57*
algebra__linear_2d	0.49	0.59	0.56	0.62	*0.63*
comparison__sort_composed	0.75	0.54	*0.55*	0.71	0.70
polynomials__expand	0.50	0.50	*0.53*	0.49	*0.52*
comparison__closest_composed	0.74	0.46	*0.60*	0.63	*0.74*
arithmetic__simplify_surd	0.47	0.94	0.84	0.94	0.87
algebra__linear_2d_composed	0.83	0.53	*0.55*	0.79	*0.81*
algebra__linear_1d	0.52	0.71	0.71	0.73	*0.78*
arithmetic__mixed	0.49	0.75	0.59	0.75	0.63
comparison__kth_biggest_composed	0.73	0.54	*0.56*	0.66	*0.71*
Average	0.62	0.60	0.60	0.70	0.72

Table 2: Per-topic single-run accuracy results on DeepMind dataset (test subset). D = description, F = formula, SRP-F = Sympy root paths, flat (were chosen for this comparison because of better individual results). Cases when using only root paths outperforms using only formula are highlighted in italic; similarly when description is added. We can see that the largest improvement is on `numbers__list_prime_factors_composed`, `polynomials__collect` and `comparison__closest_composed` exercise topics.

instead of flat sequences leads to comparable or slightly worse results. Nevertheless, adding syntactic sequences to descriptions noticeably increases performance on the DeepMind dataset, from 0.69 to 0.73 ROC AUC. Per topic accuracy scores for a single run are given in Table 2. Thus, we argue that structural embeddings have the potential to inform predictive models, especially when formula is an essential differentiating part of a task.

6 Conclusion & Future work

We proposed an adaptation of an NLP technique (Liu et al., 2017) from the field of machine comprehension to the area of mathematical educational data mining. We enrich the content representation by parsing mathematical formulas into syntax trees and embedding them with neural networks. Our experiments validate the approach using publicly available datasets and show that incorporating syntactic information can improve performance in predicting the difficulty of an exercise. These results suggest that the method may be of interest for personalised learning solutions. We hypothesise that the advantage of structural embeddings will be more evident for more advanced tasks. Therefore, as a next step, we plan to apply our approach to more complex state exams. Data have been collected and OCR-processed for initial experiments, and we intend to make the dataset publicly available. Our future research will also focus on predicting the similarity of mathematical formulas by comparing their syntax trees.

References

François Chollet et al. 2015. Keras. https://keras.io.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,*

San Francisco, CA, USA, August 13-17, 2016, pages 855–864. ACM.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. *CoRR*, abs/2103.03874.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to solve math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 805–814. Association for Computational Linguistics.

Rogers Jeffrey Leo John, Thomas S. McTavish, and Rebecca J. Passonneau. 2015. Semantic graphs for mathematics word problems based on mathematics terminology. In *Workshops Proceedings of EDM 2015 8th International Conference on Educational Data Mining, EDM 2015, Madrid, Spain, June 26-29, 2015*, volume 1446 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Dan Jurafsky and James H. Martin. 2009. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Andrew S. Lan, Divyanshu Vats, Andrew E. Waters, and Richard G. Baraniuk. 2015. Mathematical language processing: Automatic grading and feedback for open response mathematical questions. In *Proceedings of the Second ACM Conference on Learning @ Scale, L@S 2015, Vancouver, BC, Canada, March 14 - 18, 2015*, pages 167–176. ACM.

Nan Li, William W. Cohen, and Kenneth R. Koedinger. 2012. Efficient cross-domain learning of complex skills. In *Intelligent Tutoring Systems - 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14-18, 2012. Proceedings*, volume 7315 of *Lecture Notes in Computer Science*, pages 493–498. Springer.

Nan Li, William W. Cohen, and Kenneth R. Koedinger. 2013. Discovering student models with a clustering algorithm using problem content. In *Proceedings of the 6th International Conference on Educational Data Mining, Memphis, Tennessee, USA, July 6-9, 2013*, pages 98–105. International Educational Data Mining Society.

Rui Liu, Junjie Hu, Wei Wei, Zi Yang, and Eric Nyberg. 2017. Structural embedding of syntactic trees for machine comprehension. pages 815–824.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119.

Benjamin Paaßen, Jessica McBroom, Bryn Jeffries, Irena Koprinska, and Kalina Yacef. 2021. ast2vec: Utilizing recursive neural encodings of python programs. In *Proceedings of the 14th International Conference on Educational Data Mining, EDM 2021, virtual, June 29 - July 2, 2021*. International Educational Data Mining Society.

Cristóbal Romero and Sebastián Ventura. 2010. Educational data mining: A review of the state of the art. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 40(6):601–618.

Subhro Roy and Dan Roth. 2017. Unit dependency graph and its application to arithmetic word problem solving. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 3082–3088. AAAI Press.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Stefan Slater, Jaclyn Ocumpaugh, Ryan S. Baker, Peter Scupelli, Paul Salvador Inventado, and Neil T. Heffernan. 2016. Semantic features of math problems: Relationships to student learning and engagement. pages 223–230.

Tianyu Zhao, Chengliang Chai, Yuyu Luo, Jianhua Feng, Yan Huang, Songfan Yang, Haitao Yuan, Haoda Li, Kaiyu Li, Fu Zhu, and Kang Pan. 2019. Towards automatic mathematical exercise solving. *Data Science and Engineering*, 4(3):179–192.