Transformers to Predict the Applicability of Symbolic Integration Routines

Rashid Barket[1], Uzma Shafiq[1], Matthew England[1], and Jürgen Gerhard[2]

[1] Coventry University, Coventry, United Kingdom
[2] Maplesoft, Waterloo, Canada

1 Introduction

Symbolic integration is well-studied, proven to be undecidable. Recently there have been attempts to use Machine Learning (ML) to perform symbolic integration [1,3–6]. We are interested in improving the efficiency of indefinite integration within a Computer Algebra System using ML, while still ensuring correctness is guaranteed, through ML-based optimisation. We focus on the popular commercial CAS, Maple. Maple's user-level integration call is essentially a meta-algorithm: it can employ several different **methods** for symbolic integration. They are currently tried in a deterministic order until one succeeds, at which point the answer is returned without trying the other methods. A key part of this implementation is what we call the **guards**: code that is run prior to calling one of the methods to decide whether or not attempting to use the method is worthwhile. The reasoning behind having a guard is that some methods are computationally expensive: it is a waste of time to go through a complex algorithm just to find out that it fails. We create small ML models to predict success for methods and investigate whether these can provide guards for methods that lack them, or replace the computationally expensive guards.

2 Results

Table 1 shows the results for methods that do *not* have a guard. In this case, the transformers are compared to a hypothetical guard that simply predicts a positive label every time, since the current workflow would always attempt such methods. These transformers range in accuracy from 93% to 98%. Hence, there is much scope for using ML to prevent wasteful computation. Table 2 shows the results for the methods that do already have a guard. In each of these cases, we see that a transformer can beat the guards, and some by a margin of over 30%. We note that the guards for `Trager` and `PseudoElliptic` are particularly weak compared to the transformers, with much lower accuracy and precision. The transformers can avoid the false positives that the guards cannot.

3 Layer Integrated Gradients to Interpret the Classifiers

We have seen that ML can optimise the work of CASs here, but we are also interested in *how* the ML models make their decisions: both to give confidence in

Table 1: Comparing each transformer to predicting positive every time on the test dataset. These methods have no guard; they always run when Maple's algorithm reaches that specific method.

Method	Accuracy (%)	
	Transformer	Guard
`Default`	**94.86**	82.15
`DDivides`	**94.13**	28.18
`Parts`	**93.10**	37.05
`Risch`	**94.53**	89.35
`Norman`	**95.74**	71.67
`Orering`	**97.21**	37.88
`ParallelRisch`	**97.82**	82.73

Table 2: Comparing each transformer to the Maple guard on the full test dataset.

Method	Accuracy (%)		Precision (%)	
	Transformer	Guard	Transformer	Guard
`Trager`	**98.21**	67.55	**92.21**	15.88
`MeijerG`	**96.78**	88.72	**89.58**	57.78
`PseudoElliptic`	**99.28**	62.61	**62.86**	2.03
`Gosper`	**94.04**	92.51	**92.08**	80.21

their results and perhaps to inform better hard-coded guards. We further experiment with interpreting the transformers using Layered Integrated Gradients [7]; an extension of Integrated Gradients that gives insight into different layers of a neural network by calculating attribution scores of tokens. One interesting observation concerns the `Risch` method and the attribution scores found for the token of the absolute value function (`abs`). These always contribute negatively, usually highly so, in the observed samples which indicates that the presence of this token will adversely impact the prediction for use of this method. A visualisation for a particular example is shown in Figure 1. After discussing with domain experts, we find this a satisfying interpretation: the Risch algorithm is suited for elementary functions and the absolute value function does not fall in this category [2].

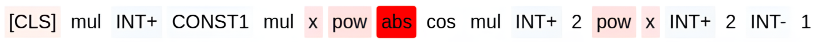
[CLS] mul INT+ CONST1 mul x pow abs cos mul INT+ 2 pow x INT+ 2 INT- 1

Fig. 1: Example sequence to depict the attribution scores corresponding to different tokens where blue is positive and red is negative. Note the strongly negative score for the `abs` token.

References

1. Barket, R., England, M., Gerhard, J.: Symbolic integration algorithm selection with machine learning: LSTMs vs tree LSTMs. In: Mathematical Software – ICMS 2024. pp. 167–175. Springer Nature Switzerland (2024), https://doi.org/10.1007/978-3-031-64529-7_18
2. Gerhard, J., Von zur Gathen, J.: Modern computer algebra. Cambridge University Press (2013)
3. Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., Steinhardt, J.: Measuring mathematical problem solving with the MATH dataset. In: Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2) (2021), https://openreview.net/forum?id=7Bywt2mQsCe
4. Lample, G., Charton, F.: Deep learning for symbolic mathematics. In: Proc. Intl. Conf. on Learning Representations (ICLR) (2020). https://doi.org/10.48550/arxiv.1912.01412
5. Noorbakhsh, K., Sulaiman, M., Sharifi, M., Roy, K., Jamshidi, P.: Pretrained language models are symbolic mathematics solvers too! ArXiv **abs/2110.03501** (2021), https://api.semanticscholar.org/CorpusID:238419670
6. Sharma, V., abhinav nagpal, Balin, M.F.: SIRD: Symbolic integration rules dataset. In: The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23 (2023), https://openreview.net/forum?id=WWDsbsgyhS
7. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 3319–3328. PMLR (06–11 Aug 2017), https://proceedings.mlr.press/v70/sundararajan17a.html