Outline of a Project Contrasting Computer Algebra Systems

and Deep Algorithms for Integration

It’s a real shame that I did not have enough time to write up a formal response to Dr Strogatz’s question, for two reasons. The first reason is selfish: I already read all of the background material, including a rather long book, and I have specific things I would like to say; I just didn’t have the time to condense everything into a very decent book. The second is that Lample and Charton’s “outperformacne” of Mathematica and Matlab has a field of asterisks, and I would have been greatly honored to help explain why exactly why they won’t replace rule-based systems anytime soon. However, since it really seems like I bit off too many A exam questions for me to chew, I would like to present at least this brief outline, summarizing my thesis statement and the sources I used to reach this conclusion, in the hopes that someone more committed to this specific project might do a better job writing such a paper, and might even build on the suggestions for further work that I will place in the conclusion. I will put all of my sources and my notes on a github page for easy access.

In short, my conclusion is that Lample and Charton’s method is very limited, with no rigorous method for reducing the arguments of their integral into more useful forms, and no defence against errors of any kind. They only beat Computer Algebra Systems (CASs) Mathematica and Matlab on a limited dataset they created themselves, and also because of a false equivalency between CAS timeouts and their system’s genuine mistakes.

It is possible that a method similar to theirs might make the “heuristic methods” portion of most automatic integration techniques faster, they are nowhere close to something as general and guranteed to produce results as the Risch algorithm which is the workhorse of CASs (see ch. 12 in [3]). (More likely, this “heuristic integration” will be done by a method that tries to capture some aspects of problem solving beyond transforming sequences, like the variable binding in the tensor product transformer from Microsoft Research [2], but this is unfortunately beyond this summary.)

*Classical Methods*

Besides the Lample and Charton paper ([1]), my second most important source would have been *Algorithms for Computer Algebra,* [3], written by cofounders of the MAPLE language. Essentially, the entire book builds up a collection of algorithms for polynomials in multiple variables, often over Galois Fields and other constructions of abstract algebra which I never imagined had a practical application, such as factorizations, GCDs, simplifications / standard forms, etc. Probably one of my personal favorites was Berlekamp’s algorithm, which factors a polynomial a(x) over a Galois Field GF(q) by computing the dimensionality of the vector space GF(q)[x] mod a(x), using standard technques out of linear algebra.

The entire book build up to chapters 11 and 12, which show how all of the methods for working with polynomials over multiple variables can instead be used to split up complex equations containing elementary functions (powers of x, logarithms, exponentials, trigonometric functions; see 12.2 in [3] for a full description) into rational functions with many different expressions, and systematically apply a technique known as the Risch algorithm to either compute the indefinite integral or prove that none exists that can be expressed using those functions. (There are also clear discussions about what makes a collection of elementary functions, and how to expand these techniques to larger collections.)

Besides a fascination with Applied Abstract Algebra, which I had always imagined to be a contradiction in terms, this book helps understand why CASs provide such good guarantees about accuracy, and why they are capable of handling so many very convolouted examples. The section on Most importantly, the algorithm either converges or proves the absence of a solution in simple terms, which is very far from the guarantees provided by [1].

*The Innovation*

In a brief summary, the authors of [1] standardize how they write mathematical expressions by first producing a tree representation of the expressions to be integrated, then write them in normal Polish notation. They then use an out-of-the-box deep learning sequence-to-sequence model (known as a transformer, see [5]) to produce the integrated “output” based only on the unintegrated “input.” They produce these pairs in 3 ways: using CASs to integrate randomly generated input, producing a FWD dataset; using CASs to differentiated randomly generated output, producing a BWD dataset; and using integration by parts to start from random u,v pairs and produce both the integral and the integrand, making an IBP dataset. They compared the performance of their system with CASs on the BWD dataset only.

In contrast to the guarantees in the Risch algorithm, the work in [1] achieves impressive results on functions with a similar length and structure to the elements of the training dataset, but has no guarantees about generalizing with functions that have any other structure or form. Like automatic translators (and unlike CASs with their rigorous guarantees and custom-made algortrihms), the neural networ cannot check if the transformations “make sense,” but only repeat the same sorts of transformations that its seen done before.

There are many more points I could make, but Ernest Davis did a very good job explaining the weaker points of this paper in [4] (). I’ll mention two:

1) Unlike CASs, which have a strong, formal understanding of normal forms and canonical forms (see ch. 3 in [3]), the Lample / Charton system will likely struggle with any simple function that is written in a convolouted way (Ernest Davis gives two great examples in section 3).

2) The test set has too many properties that are different from what is likely in the real world (eg, relative length of integral and integrand), and its unlikely for any function encountered by a researcher “in the wild” to share those properties. (CASs, of course, don’t need to worry about this since they don’t have a training distribution, only specific rules to follow).

Since I couldn’t produce a complete response, anyone who is really interested in properly contextualizing [1] should read [4]. The only element really missing to the discussion in [4] is a good description of the CASs which [1] tries to displace, which I hope I have helped provide here.

*Conclusion / Further Work*

Is there anything positive that can come out of [1]? Well, the Risch algorithm is computationally expensive and unwieldy. On pg 514 in [3], the authors mention that most Computer Algebra Systems try some transformations and a table lookup before invoking heavier artillery. It’s very likely that a deep learning technique can easily replace this initial procedure. Wrong results are not a major concern, since the output of the neural network can be differentiated automatically (usually a far simpler procedure) and compared to the input. Since the Risch algorithm breaks down the argument into simpler components multiple times, its possible that a deep network might have more than one opportunity to preserve computational resources.

Furthermore, any CAS with a large user base must already have a log of functions often thrown at it; it would be very easy to train it using this real distribution (possibly augmented with some examples calculated to showcase corner cases), so that the output of the network will not suffer from the biases Ernest Davis discussed in section 6 of [4]. I find it very likely that a deep-learning based heuristic guess will become one of the many subroutines available within advanced computer algebra systems.

However, sequence-to-sequence models, or any other simple system copied from deep learning, will unlikely replace automatic integration entirely in the coming years. We will need a far more sophisticated AI system, capable of understanding and obeying strict logical rules, before modern mathematical software might be displaced

*Sources*

1. Guillaume Lample, François Charton. “Deep Learning for Symbolic Mathematics.” ICLR 2020 submission. Dec. 2019. Available at <https://arxiv.org/abs/1912.01412>

2. Kezhen Chen, Qiuyuan Huang, Hamid Palangi, Paul Smolensky, Kenneth D. Forbus, Ji Gao. “Natural-to-Formal-Language Generation Using Tensor Product Representations.” ICLR 2020 submission. Oct. 2019. Available at <https://arxiv.org/abs/1910.02339>

3. Keith O. Geddes, Stephen R. Czapor, George Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

4. Ernest Davis. “The Use of Deep Learning for Symbolic Integration: A Review of (Lample and Charton, 2019)”. December 2019. Available at <https://arxiv.org/pdf/1912.05752.pdf>

5. Ashish Vaswani, Llion Jones, Noam Shazeer, Niki Parmar, Aidan N. Gomez, Jakob Uszkoreit, Łukasz Kaiser, Illia Polosukhin. “Attention is all you need.” NIPS 2017. Available at <https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf>