Analyzing the Nuances of Transformers' Polynomial Simplification Abilities

Vishesh Agarwal[‡], Somak Aditya[†] and Navin Goyal[†]
[‡]Microsoft Vancouver (Work done at MSR India) [†] Microsoft Research, India
(Accepted in ICLR 2021 MATH-AI Workshop)

Abstract

We explore an approach to generate human friendly step-by-step solutions to math problems using transformers. We try out our approach on the toy problem of simplifying polynomials.

Motivation:

- Recent success of transformers on solving math tasks in end-to-end manner [1, 2]
- Recent success in step-wise deduction on logic tasks [3, 4, 5]
- Opaqueness of end-to-end models to probing how they work and where they might not. [6]

Main Results (on polynomial task):

- Stepwise solvers could usually outperform end-to-end model, while training on similar number of datapoints.
- It significantly helped to offload arithmetic to external calculators while using transformers for symbol manipulation and deducing solution steps.
- In many cases one can define a reasonable curriculum [7, 8] for learning math tasks. Using such a curriculum also greatly boosted performance.

Transformers on complex symbolic tasks

- Complex tasks often require multiple steps

> Problem: Along-with the final solution, can we also "show" intermediate steps to reach the solution?

- Complex mathematical tasks often have well-defined sub-tasks

> Problem: Which sub-tasks Transformers don't generalize on automatically? What are possible solutions?

Problem Setup

- Starting polynomial → Generated as sum of products

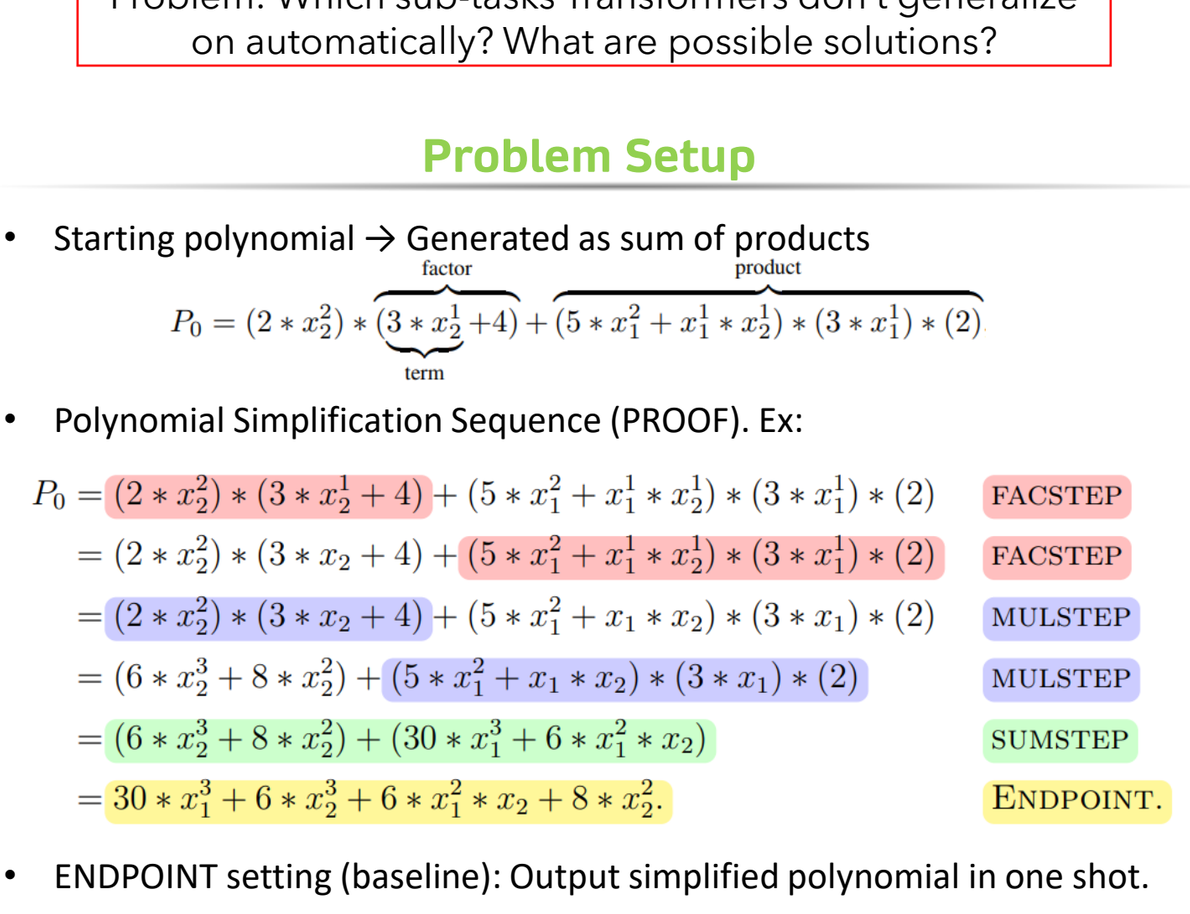
$$P_0 = (2 * x_2^2) * \overbrace{(\underbrace{3 * x_2^1}_{\text{term}} + 4)}^{\text{factor}} + \overbrace{(5 * x_1^2 + x_1^1 * x_2^1) * (3 * x_1^1) * (2)}^{\text{product}}$$

- Polynomial Simplification Sequence (PROOF). Ex:

$$\begin{aligned}
P_0 &= (2 * x_2^2) * (3 * x_2^1 + 4) + (5 * x_1^2 + x_1^1 * x_2^1) * (3 * x_1^1) * (2) && \text{FACSTEP} \\
&= (2 * x_2^2) * (3 * x_2 + 4) + (5 * x_1^2 + x_1^1 * x_2^1) * (3 * x_1^1) * (2) && \text{FACSTEP} \\
&= (2 * x_2^2) * (3 * x_2 + 4) + (5 * x_1^2 + x_1 * x_2) * (3 * x_1) * (2) && \text{MULSTEP} \\
&= (6 * x_2^3 + 8 * x_2^2) + (5 * x_1^2 + x_1 * x_2) * (3 * x_1) * (2) && \text{MULSTEP} \\
&= (6 * x_2^3 + 8 * x_2^2) + (30 * x_1^3 + 6 * x_1^2 * x_2) && \text{SUMSTEP} \\
&= 30 * x_1^3 + 6 * x_2^3 + 6 * x_1^2 * x_2 + 8 * x_2^2. && \text{ENDPOINT.}
\end{aligned}$$

- ENDPOINT setting (baseline): Output simplified polynomial in one shot.

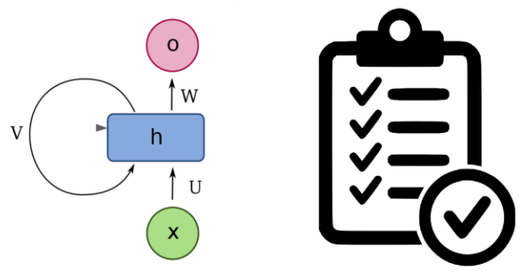
Informal Summary of Related Work



Experiments & Task Metrics

Main task metrics:

- Proof Accuracy: Compared against ENDPOINT baseline accuracy.



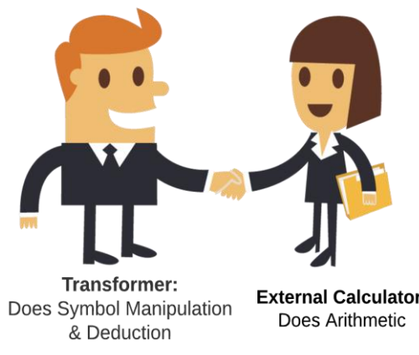
- Step-wise Error Rate:
  Errors classified on basis of simplification step type (FAC, MUL, SUM).

Analyzed performance across multiple dimensions of task complexity:

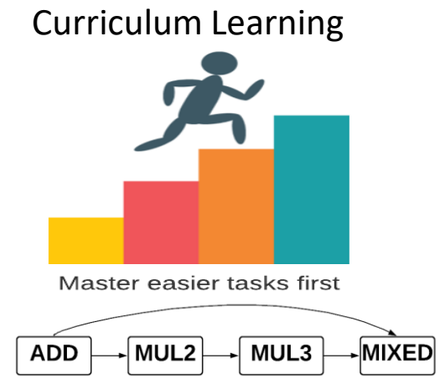
→ Increasing number of variables: 1VAR vs 2VAR
→ Increasing coefficient range: SMALL vs MEDIUM vs LARGE COEFF
→ Increasing degree of terms: MEDIUM COEFF vs MEDIUM DEGREE
→ Increasing number of terms: MEDIUM COEFF vs MEDIUM TERMS
→ Maxing out everything at once: MEDIUM COEFF vs NO BACKTRACK

Interesting Approaches

- Symbolic Calculator



- Mastering-Rate Based Curriculum Learning



Experimental Results

- Most errors occur in multiplication step. This motivated the symbolic calculator setting.
- Symbolic calculator setting beat ENDPOINT baseline proof accuracy by ~10% in LARGE COEFF and NO BACKTRACK config.
- Curriculum Learning provided gain of ~10% on LARGE COEFF and ~20% on NO BACKTRACK config over vanilla transformer implementation.

Additional Observations:

- As expected, longer proofs lead to poor proof accuracy.
- Greedy decoding performs better than beam search.

Future Work

- Generating proofs for more complex math tasks like inequalities [9] and differentiation.

References

[1]: Bartosz Piotrowski and Josef Urban and Chad E. Brown and Cezary Kaliszyk. Can neural networks learn symbolic rewriting? 2019
[2]: Guillaume Lample and Francois Charton. Deep learning for symbolic mathematics. *In International Conference on Learning Representations,* 2019.
[3]: Paliwal, Loos, Rabe, Bansal, Szegedy (2020). Graph Representations for Higher-Order Logic and Theorem Proving. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(03), 2967-2974.
[4]: Bansal, Loos, Rabe, Szegedy, Wilcox. Holist:An environment for machine learning of higher order logic theorem proving. *In Thirty-sixth International Conference on Machine Learning,* 2019
[5]: Kshitij Bansal, Christian Szegedy, Markus Norman Rabe, Sarah M. Loos, and Viktor Toman. Learning to reason in large theories without imitation, 2021
[6]: Ernest Davis. The use of deep learning for symbolic integration: A review of (Lample and Charton, 2019).
[7]: Graves, Bellemare, Menick, Munos, Kavukcuoglu. Automated curriculum learning for neural networks. *In International Conference on Machine Learning*, pp. 1311–1320, 2017.
[8]: Willems, Lahlou, Bengio. Mastering rate-based curriculum learning, 2020
[9]: Wu, Jiang, Ba, Grosse. INT: an inequality benchmark for evaluating generalization in theorem proving. *In International Conference on Learning Representations,* 2021.