Achieving Machine Reasoning for Math Problems Through Step-By-Step Supervision Signal

Brian Do
Stanford University
brianhdo@stanford.edu

Phillip Hoovestol
Stanford University
philhoov@stanford.edu

Max Sobol Mark
Stanford University
joelmax@stanford.edu

Abstract

Multi-step reasoning problems pose major challenges to traditional supervised learning approaches. Solving math questions in a variety of different topics falls into this category. While approaches exist that solve math expressions, they rely on encoded explicit prior knowledge. In this work, we aim to improve algebraic generalization by including intermediate steps to shape the loss landscape to train deep learning models without any explicit domain-specific prior knowledge. We create code to generate intermediate steps for a variety of math problems and test on linear regression, LSTM, and the Transformer.

1 Introduction and Motivation

Deep learning approaches have seen major successes with pattern recognition in a variety of contexts, from computer vision to speech recognition. However, there are still problem domains for which current approaches lack the necessary robustness and flexibility, such as solving multi-step reasoning problems. Math problems fall within this domain.

Mathematics can be viewed not only from the perspective of following a set of axioms but as recognizing and exploiting the patterns encoded in mathematical rules. However, mathematical reasoning presents unique challenges. Many problems, such as simplifying arithmetic expressions involving parentheses, necessitate the creation of intermediate results in order to produce correct final outputs, resulting in non-smooth loss functions. This presents significant difficulty to traditional supervised learning approaches. In Saxton et al., many of the problems for which the baseline models work well have relatively smooth loss functions with respect with the parameters [1]. For example, for the problem of rounding numbers, the model can learn to imitate the input characters and simply modify the last few digits.

There are no such shortcuts for evaluating multi-step problems [1]. Consider the question "Evaluate (3*5) + (60/3)". The answer is 35. Replacing the second ‘3’ with a ‘5’ in the expression changes the answer to ‘27’, meaning that both digits from the previous answer are incorrect. Thus, small changes to the input questions can drastically effect the correct output answer.

Interestingly, in humans, explicit prior knowledge of mathematical rules may be neither sufficient nor necessary for correct mathematical performance [2, 3]. Education studies have shown that problem solving in math is an acquired skill which emerges by having students solve a number of example problems [4, 5, 6, 7]. Thus, the process of solving problems step-by-step and then comparing to correct solutions is an important part of the learning process.

Building on this, we propose including intermediate steps as additional labels for multi-step problems in a manner which imitates how humans learn math. Specifically, we aim to improve algebraic generalization and accuracy in deep learning models, particularly in the Transformer model, by shaping the reward function to include intermediate steps to solve the problem. We propose that doing so could alleviate the issues with the otherwise non-smooth loss function. This could hopefully provide insight to generate new models that can solve more interesting, challenging mathematical and reasoning problems.

2 Related Work

Although the use of intermediate steps while solving math problems has not previously been investigated, solving mathematical problems has been an area of interest within the NLP community for over five decades. More recently, various machine learning techniques, particularly deep learning, have been applied.

For example, Kushman et al. demonstrate end-to-end solving of word problems via a deep learning model [8]. Roy et al. demonstrates mapping between linguistic and mathematical expressions [9]. These works and [10, 11] focus more on the linguistic understanding of word problems rather than mathematical reasoning. Meanwhile, Ling et al. train a model to generate "answer rationales" - natural language descriptions that walk through the problem solving method - for word problems and demonstrate that this allows for better accuracy than other neural models [12].

There is some prior work that focuses on algebraic reasoning. Kaiser and Sutskever demonstrate that a highly parallel, shallow neural network model can learn long binary multiplication and addition from examples [13].

In particular, Saxton et al. investigate the performance of the Transformer model and compare its performance with a simple long short-term memory (LSTM) and an attentional LSTM [1]. Their results suggest that none of these architectures are doing much "algorithmic reasoning" and found that the Transformer performed best among the tested models due to several architecture advantages. The performance for baseline models indicates that the models can solve problems where relatively linear/shallow/parallel reasoning is required, but the performance drops dramatically where intermediate values need to be calculated.

Also notice that for the particular dataset we are using, almost every problem type would be solvable by computational engines like WolframAlpha. The difference between using one of these services and our machine learning models is that we don't embed prior knowledge in our systems; math problems are simply an instantiation of a higher level class of problems that require multiple steps of reasoning to be solved. WolframAlpha on the other hand would only be able to solve tasks for which it has been explicitly programmed.

3 Dataset and Features

Our models are trained on the dataset of mathematical problems in the form of character by character question answering which DeepMind released in 2019 [1]. A sample question-answer pair from this dataset is:

Q: Find the second derivative of $q^5 - 391 * q^4 + 1600 * q^2$
A: $20 * q^3 - 4692 * q^2 + 3200$

Our project focused on six problem types which produced relatively low test accuracy with the deep learning models tested in [1]: evaluating and simplifying polynomials, evaluating arithmetic expressions using order of operations, finding polynomial roots, and finding remainders. The accuracy for these ranged from 17% to 61%.

We modified this dataset in two keys ways: 1) by augmenting existing problem types with intermediate steps and 2) by modifying dataset questions to make them amenable for application of baselines.

We wrote a program to produce intermediate steps for each of the question-answer pairs and append these steps to the existing answers. For simple problems, such as finding the remainder, intermediate steps were calculated directly. For more complex problems, such as simplifying polynomials, we interfaced with a JavaScript library called 'mathsteps' produced by socratic.org to generate intermediate steps. An example modified question is:

Q: Calculate (-168)/(-2) + (28 - 74)
A: 168 / 2 + (28 + -74)||84 + (28 + -74)||84 + -46||38||

Instead of just having the target '38', now there are intermediate steps, where the end of a step is denoted by '||'

Separately, the dataset was also modified to allow for application of a linear regression baseline. Firstly, the scope of questions supplied to the linear regression was limited to arithmetic problems. This is because for more complex questions that require semantic understanding, such as "Differentiate -186089*h**2 - 122007 twice with respect to h", there is no straightforward way to encode the linguistic information into a format that linear regression can use. Secondly, words such as "Evaluate", "Solve", etc. were removed due to the difficulty in encoding this information into a format amenable to linear regression. Thirdly, problems were adapted from strings to vectors. To do this, mathematical operands like + or $\times$ were encoded into the input vector with a 1 if they were present and 0 if they were not. So, for example, a problem like $3 + 5 \times 7$ would be encoded as '3 1 0 5 0 1 7', where every 2nd index is a flag denoting addition and every 3rd index is a flag denoting multiplication. All problems were also fixed in length. These were only done for the linear regression experiments.

4 Methods

We tested four different machine learning models: 1) kernelized ridge regression, 2) a long short-term memory recurrent neural network (LSTM RNN) without attention, 3) an LSTM RNN with attention, and 4) the Transformer model. To run the experiments for all four, we created a dataset of math problems based on the released DeepMind dataset, as detailed in the "Dataset and Features" section.

4.1 Linear Regression

As a simple baseline measure, we used a kernelized linear regression. To make the model more robust against overfitting, L2 regularization was used. The loss function for this kernel ridge regression is shown in Equation 1.

$$J(\theta) = \sum_{i=0}^{n}(y^{(i)} - \theta^T \phi(x^{(i)}))^2 + \lambda||\theta||_2^2 \tag{1}$$

In an attempt to capture some of the inherent non-linearity of solving math problems while following order of operations, a kernel was used using a custom feature-map $\phi(x)$ based on the interaction terms of polynomial features. The linear regression was implemented using the scikit-learn library.

4.2 Deep Learning Models

4.2.1 Long Short-Term Memory (LSTM)

As a more complex baseline, we used an LSTM. LSTM is a type of recurrent neural network (RNN) which can handle sequences of data thanks to its feedback connections. One problem with standard RNNs is the issue of vanishing gradients during backpropagation of long sequences of data. To address this, the LSTM has a cell which acts as a "memory" for the network and input, output, and forget gates which mediate the transfer of information back into the network. Because the dataset is text data with important relational information, the LSTM is a suitable choice. For our experiments, we used both a sequence-to-sequence LSTM with attention and without attention implemented using the Tensor2Tensor library.

4.2.2 Transformer

In part based on its its success in the DeepMind experiments [1], we chose the Transformer model for our experiments. Unlike traditional neural models, the Transformer does not involve recurrent connections. Instead, it applies a self-attention mechanism which directly models relationships between inputs, allowing it to better retain relationships between distant characters in text inputs. While typically used for translation, the Transformer can be applied generally to NLP problems as well as any sequence-to-sequence problem, such as mathematics. The strengths of the Transformer thus make it an attractive choice for the generation of answers in multi-step mathematics questions. For our experiments, we used the Transformer model from the Tensor2Tensor library.

	Addition only	Multiplication only	Mixed arithmetic
Without intermediate steps	100%	100%	0.8%
With intermediate steps	100%	100%	0.8%

Figure 1: Test accuracies for linear regression baseline with and without intermediate steps

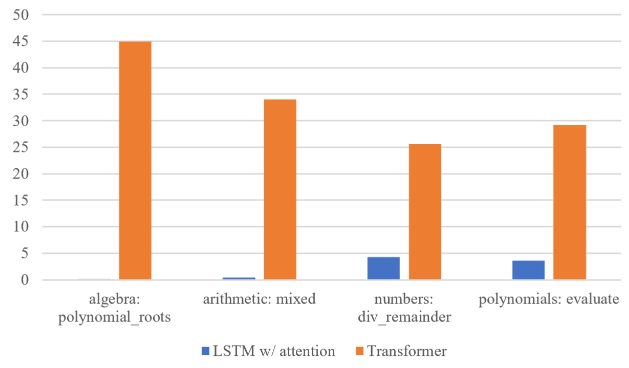

Figure 2: Test accuracies for LSTM with attention and Transformer model on the unmodified DeepMind dataset

4.3 Accuracy Metric

For all models, the accuracy of their predictions was assigned by calculating the percentage of predictions which matched the actual answer exactly character by character. However, we only apply this accuracy metric to calculate the validation accuracy and not during training.

The rationale behind using this ‘exact character match’ metric rather than a more traditional regression accuracy metric such as mean square error or mean absolute error is that it is not useful in the real world to obtain an answer to a math problem that is only approximately correct; thus, such answers are penalized accordingly by not counting at all towards accuracy. Additionally, for models trained on intermediate steps, we only considered the final answer when calculating accuracy.

5 Experiments and Results

5.1 Linear Regression Baseline

For the linear regression baseline, experiments with and without intermediate steps were performed for three different problem types: 1) addition only, 2) multiplication only, and 3) mixed arithmetic. 5.1 shows the accuracies for each experiment.

Given the polynomial-based kernel, linear regression is able to achieve 100% accuracy for addition and multiplication only tasks since these operations are linear given the kernel. We can also see there is a dramatic drop-off in accuracy for the mixed arithmetic task; this is because there is no easy linearization for mixed arithmetic, which is highly nonlinear due to order of operations. The inclusion of intermediate steps also does not improve performance. Linear regression treats these intermediate steps as more values to regress on, but given the simplistic nature of the model, it is unable to utilize the additional information to produce a better model. More complex problem types also had close to 0% test accuracy, and this highlights the need to use more complex models for these solving multi-step reasoning problems.

5.2 Results with Unmodified Dataset

For our deep learning experiments, we set up the LSTMs to run with actual GPUs and the Transformer model with an Cloud TPU v2 by using a combination of Google Colaboratory and Google Cloud Machine Learning Engine resources. Our experiments involved training with the full unmodified dataset released by Google DeepMind.

Figure 2 shows the results from the sequence-to-sequence LSTM with attention and the Transformer model.

We can see from figure 2 that the accuracy of the LSTM with attention model is significantly worse than the Transformer model. One possible explanation for this observation might be that we didn't do extensive hyperparameter sweeping, so it might be the case that the Transformer architecture is simply more robust to untuned hyperparameters.

Additionally, when comparing both families of models, the transformer has a significant advantage when processing text like the math questions we are working with. Notice that for most tasks, the type of question is mentioned at the beginning of the question (E.g. "Find the derivative of $q^5 - 301 * q^4 + 1009 * q^2 + 3$"). It is easier for the Transformer architecture to learn to pay attention to the characters that are further back in the question when compared to the simple LSTM and the attentional LSTM, since the LSTMs have to learn a relationship between just observed information and information that was observed several recurrent steps before. On the other hand, the Transformer has direct access to both the current characters and the relevant previous characters.

5.3 Results with Dataset with Intermediate Steps

To train with intermediate steps, we replaced files from the original DeepMind dataset with the corresponding files that we generated containing question-answer pairs with intermediate steps.

Although we trained the LSTM without attention, the LSTM with attention, and the Transformer on the newly modified dataset, we were unable to properly decode the resulting files; thus, unfortunately, we are ultimately unable to calculate the accuracies for the models and make any conclusions about the effectiveness of including intermediate steps as a way to improve algebraic generalization. We have included the raw outputs of all the models trained on the intermediate dataset in this Google Drive folder: https://drive.google.com/open?id=1VPTjASh8JxjOTb8yPnHW7fHDc1O7Hipy. Obviously, for future work, we would work on decoding these files, from which we would be able to rather quickly calculate the accuracy and then investigate the results.

6 Conclusions

We have built on the dataset produced by DeepMind and created a framework to extend it to produce intermediate steps. In general, it was seen that including intermediate steps had no effect for simple models such as linear regression and we were ultimately unable to decode the output data for the trained models on the intermediate steps, so no firm conclusions can be made.

Furthermore, we want to continue to better understand how the structure of models can impact their multi-step reasoning ability, both in the context of algebraic generalization and beyond. Thus, future work includes performing a grid search for hyperparameters using additional computational resources for LSTM and Transformer models to better understand what types of architectures can best benefit from intermediate steps.

Our code can be found at at https://github.com/MaxSobolMark/cs229project

7 Contributions

- **Brian Do:** Created handcrafted features for the linear regression model, wrote code to create modified dataset for linear regression, and implemented the linear regression baseline. Created the poster. Main contributor to the report.
- **Phillip Hoovestol:** Implemented the Transformer architecture and LSTM architectures using Tensor2Tensor and ran experiments on the unmodified and modified Google DeepMind dataset.
- **Max Sobol Mark:** Implemented the Transformer architecture with the original dataset with the original and modified dataset using Tensor2Tensor, and implemented the code that creates the modified dataset with intermediate steps.

References

[1] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

[2] Philip J Kellman and Christine M Massey. Perceptual learning, cognition, and expertise. In *Psychology of learning and motivation*, volume 58, pages 117–165. Elsevier, 2013.

[3] Robert Siegler and Roberto Araya. A computational model of conscious and unconscious strategy discovery. In *Advances in child development and behavior*, volume 33, pages 1–42. Elsevier, 2005.

[4] Alan Schoenfeld. Learning to think mathematically: Problem solving, metacognition, and sense-making in mathematics. *Coleccion Digital Eudoxus*, (7), 2009.

[5] Jill L Quilici and Richard E Mayer. Role of examples in how students learn to categorize statistics word problems. *Journal of Educational Psychology*, 88(1):144, 1996.

[6] Alexander Renkl. Learning mathematics from worked-out examples: Analyzing and fostering self-explanations. *European Journal of Psychology of Education*, 14(4):477–488, 1999.

[7] Stephen I Brown and Marion I Walter. *The art of problem posing*. Psychology Press, 2005.

[8] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 271–281, 2014.

[9] Subhro Roy, Shyam Upadhyay, and Dan Roth. Equation parsing: Mapping sentences to grounded equations. *arXiv preprint arXiv:1609.08824*, 2016.

[10] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, 2014.

[11] Subhro Roy and Dan Roth. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*, 2016.

[12] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

[13] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.