Transformer Model for Mathematical Reasoning - CS 230 Final Report

Will White
Department of Mathematics
Stanford University
will2@stanford.edu
Justin Dieter
Department of Mathematics
Stanford University
jdieter@stanford.edu

8 December 2019

1 Abstract

One chief goal of natural language processing is to encode the embedded logic of natural language. Hence, a potentially interesting research direction is to test the limits of the complexity of the logic that can be learned from symbolic statements of language. To this end, we were interested in deploying a modern sequence model, specifically a Transformer model, on a dataset of a wide range of math problems.

2 Introduction

In a world where highly dense and technical academic research abounds in text form in journals and databases, there is utility in automatically searching through articles to cull research that might together lead to further findings. For example, in an exciting 2019 study published in *Nature* [2], a team of researchers used Word2vec to successfully predict the existence of now known thermoelectric materials from earlier research in the field. Generalizing the search to the most technical of literature necessitates a flexible understanding of mathematical formulae in the context of natural language.

With this distant vision of automatic research in mind, the work begins with encoding the logic of basic math problems. The input to our model is a text string corresponding a problem statement, and we use a Transformer model to output a text string meant to be the answer to the mathematical task given.

3 Related Work

In their 2019 paper [3], Google DeepMind introduced the dataset of math problems used in this project, as well as good performance on most tasks with a Transformer. The main difference between the approach taken in this paper versus ours is tokenization. Deepmind takes a character-by-character approach for all string data, but our model tokenizes each English word while tokenizing each chunk of mathematical symbols character by character.

The current state of the art in this dataset is a Tensor Product Transformer [1]. This model adds explicit relational encoding to each multi-headed attention layer by computing an additional relational vector, which is adjoined to the multi-headed attention weighted vectors by the tensor product.

4 Dataset and Features

The dataset is organized into different classes of problems including algebra, arithmetic, comparison, probability, calculus, and many more. The dataset consists of pairs of text strings corresponding to problem statement and answer. The problem statements contain English words as well as mathematical symbols, and while most answer data takes the form of single numbers, answers can be mathematical expressions as well as words. The following examples from the dataset capture the range of input and output well:

(1) **Problem:** Solve -n = 11*z - 14*z + 17, 22 = 3*z - 2*n for z. **Answer:** 4

(2) **Problem:** Is 1128546091 a prime number? **Answer:** False

(3) **Problem:** Suppose 3*b - 51 = -9. Let t(n) be the first derivative of
5 - b*n**3 + 13*n + 29*n - 9*n - 1. Differentiate t(v) wrt v. **Answer:** -84*v

Due to memory constraints, we trained on 40 percent of the available 2 million training examples for a total of 800,000 training examples. As discussed before, the data has been tokenized as whole words if present in the English dictionary, and character-by-character for mathematical expressions. In the text data for each training example, problem and answer are separated by a period or question mark, so examples are split accordingly. The model was trained with a mini-batch size of 400. All training data is from the DeepMind GitHub for the dataset, available at https://github.com/deepmind/mathematics_dataset.

5 Methods

The model used is a Transformer as laid out by tutorials referenced here as [4], [5], [6]. The Transformer model makes use of attention vectors, which tell the algorithm which parts of the input sequence are more important when computing the next parts of the output sequence. Attentions are computed by taking the sum of hidden state vectors generated by the encoder layers, which are weighted by their softmax value. This makes the most likely hidden states factor more heavily into the next output generation step.

The hyperparameters of our implementation are as follows: embedding dimension of 200, 6 encoder layers, 6 decoder layers, 8 attention heads, feed-forward dimension of 2048, and dropout with probability 0.1. The loss function used is cross-entropy. The optimizer used is Adam with $\beta_1 = 0.9, \beta_2 = 0.995$ and $\epsilon = 1e - 9$, and a learning rate of $\alpha = 0.0001$.

6 Experiment, Results, Discussion

The performance of the model is evaluated on both interpolated and extrapolated test sets. That is, the interpolated test set comes from the same distribution as the training data, and the extrapolated test set is generated by creating new examples with, for example, larger numbers, more compositions, or larger samplers (for probability questions). Accuracy is assessed by assigning a 1 if the predicted output matches the answer label exactly and a 0 otherwise, and taking the proportion of correct predictions over the total number of test examples. After a few days of training on an NVIDIA Titan RTX, and over 250,000 training epochs, we can exhibit a steadily decreasing loss function and the accuracy plots on Page 4, indicating interpolation accuracy as high as approximately 0.41 and extrapolation accuracy as high as approximately 0.38.

7 Conclusion, Future Work

Ultimately, more compute is required to reach the accuracy levels of 0.76 interpolation and 0.50 extrapolation reported by DeepMind. Note that the accuracy for interpolation and extrapolation are not too far apart at approximately 0.03, so the degree of overfitting is acceptably low. This may be in part attributed to the use of dropout. Also, clearly we would've liked to do more experiments and hyperparameter tuning, but due to long training time and short time in the quarter, we were unfortunately left with this single experiment.

Ultimately, even the state of the art accuracy marks are significantly less than human level accuracy (for properly trained humans), so there is room for trying new algorithms on this dataset. DeepMind observes that their algorithm performs poorly on problems involving "several intermediate calculations", so future work might include memory augmented architectures such as a Neural Turing Machine in the hope that the information of these intermediate steps might be learned and stored.

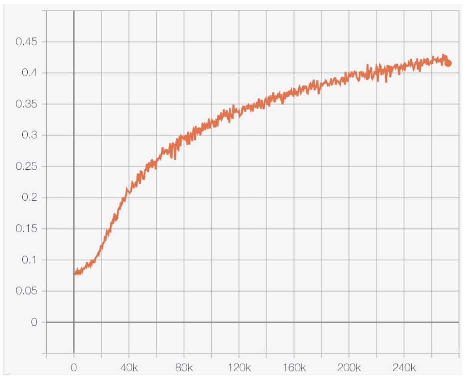

Figure 1: Interpolation accuracy - vertical axis is accuracy and horizontal axis is number of training epochs.

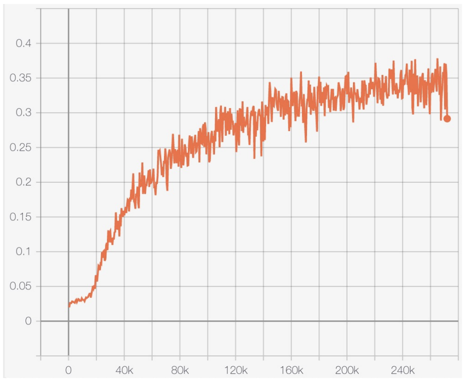

Figure 2: Extrapolation accuracy - vertical axis is accuracy and horizontal axis is number of training epochs.

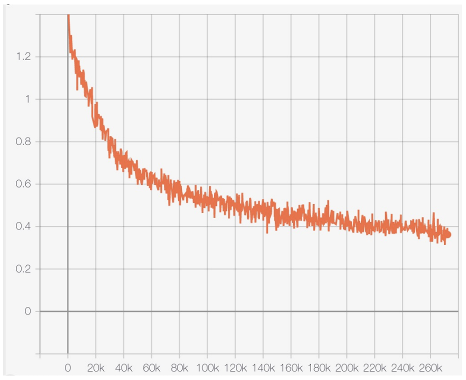

Figure 3: Loss function - vertical axis is the value of the loss function and horizontal axis is the number of training epochs.

8 Contributions

- Justin Dieter (who is enrolled in CS 330 and using this dataset further in a multitask setting) did the bulk of the coding and provided the plots necessary for evaluating the model.
- Will White helped with the tokenization step of the data pipeline and wrote up the final report and poster.

References

[1] Imanol Schlag, Paul Smolensky, Roland Fernandez, Nebojsa Jojic, Jurgen Schmidhuber, and Jianfeng Gao. Enhancing the transformer with explicit relational encoding for math problem solving. ArXiv, abs/1910.06611, 2019.

[2] Tshitoyan, Vahe Dagdelen, John Weston, Leigh Dunn, Alexander Rong, Ziqin Kononova, Olga Persson, Kristin Ceder, Gerbrand Jain, Anubhav. (2019). Unsupervised word embeddings capture latent knowledge from materials science literature. Nature. 571. 95-98. 10.1038/s41586-019-1335-8.

[3] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathe- matical reasoning abilities of neural models. ArXiv, abs/1904.01557, 2019.

[4] SEQUENCE-TO-SEQUENCE MODELING WITH NN.TRANSFORMER AND TORCHTEXT, https://pytorch.org/tutorials/beginner/transformer_tutorial.html

[5] How to use TorchText for neural machine translation, plus hack to make it 5x faster, https://towardsdatascience.com/how-to-use-torchtext-for-neural-machine-translation-plus-hack-to-make-it-5x-faster-77f3884d95

[6] The Annotated Transformer, https://nlp.seas.harvard.edu/2018/04/03/attention.html