Transformer Model for Mathematical Reasoning

Justin Dieter jdieter@stanford.edu, Will White will2@stanford.edu

Stanford University

Introduction

- Solving math problems is an excellent benchmark of a machine learning model's ability to implement logical reasoning. In 2019, DeepMind introduced a dataset of problem-solution pairs in a wide range of mathematical tasks, including arithmetic, comparison, algebra, calculus, polynomials, probability and many more.
- 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 some math problems, which contain more complicated embedded logic.
- Prior work has shown modern machine learning approaches to be accurate for some tasks in the dataset while performing poorly on tasks involving "several intermediate calculations", that still might be considered easy for (applicably trained) humans.

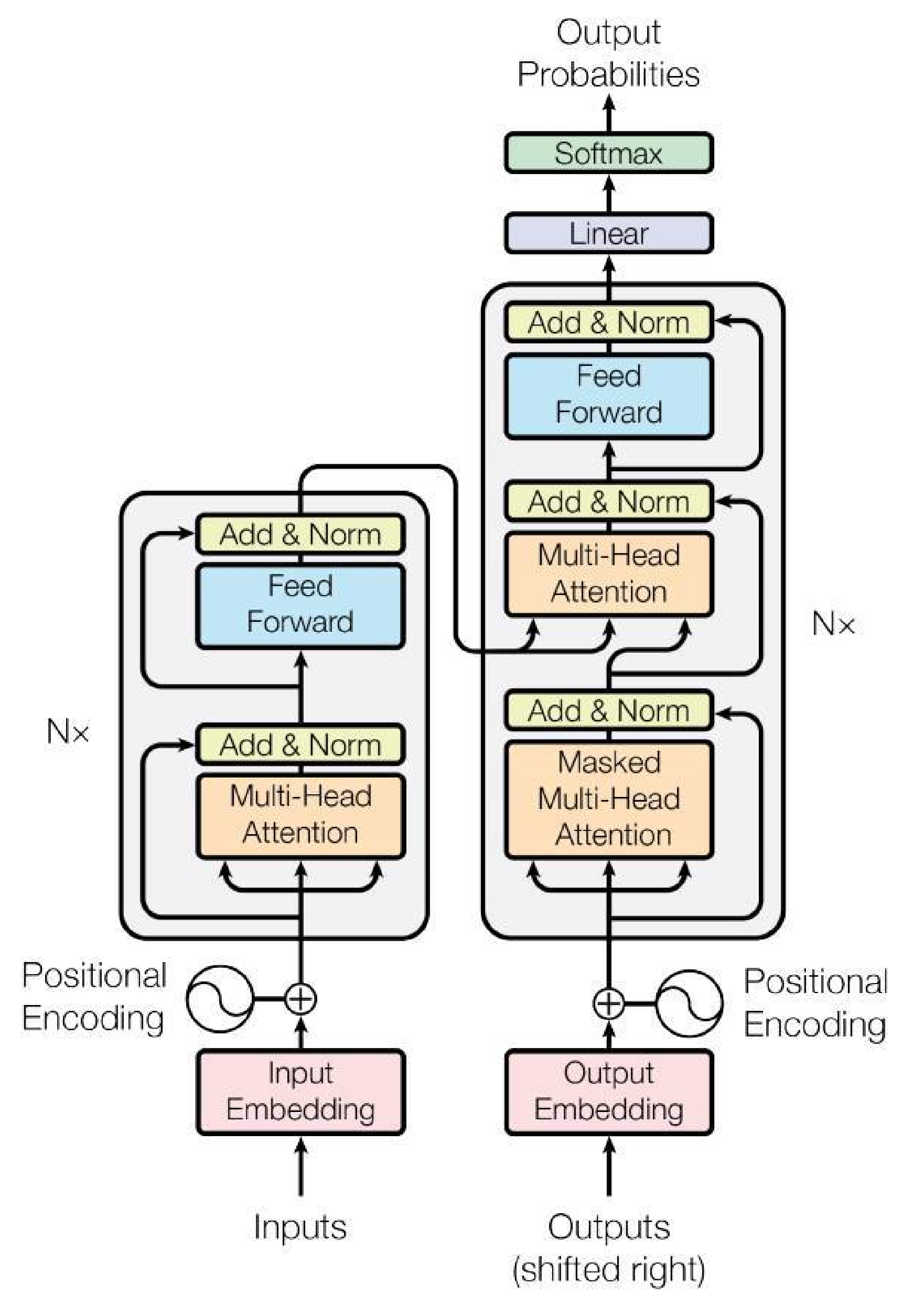
Example Problem	Desired Output
Solve -n = 11*z - 14*z + 17, 22 = 3*z - 2*n for z.	4
Suppose 3*b - 51 = -9. Let t(n) be the first derivative of 5 - b*n**3 + 13*n + 29*n - 9*n - 1. Differentiate t(v) wrt v.	-84*v
Is 1128546091 a prime number?	False

Note that the problem statements contain English words as well as mathematical symbols, and while most problems are formatted to have numerical solutions, solution data may be mathematical expressions or words (e.g. True, False).

Data Pipeline Considerations

- **Tokenizing Words and Mathematical Expressions**
  - As seen in the examples provided above, the problem statements contain both words and mathematical expressions. DeepMind elects to tokenize the dataset character-by-character in order to encode arbitrary mathematical expressions. However, this means that each word in the dataset is split into each letter, taking up more space in its one-hot encoding. Our solution is to check if space-split chunks of the text data are in the Python English dictionary and tokenize as a whole word if so, and tokenize character-by-character if not. This saves memory in generating the one-hot encoding for the training examples.

Model



A diagram of the Transformer Model courtesy of pytorch.org

- 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 $\beta_1 = \mathbf{0.9}, \beta_2 = \mathbf{0.995}$ and $\epsilon = \mathbf{1e-9}$, and a learning rate of $\alpha = \mathbf{0.0001}$.

Conclusion

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.

Evaluation

- Due to memory constraints, we trained on 40 percent of the available 2 million training examples for a total of 800,000 training examples. Training was carried out in mini-batches of 400.
- Our test accuracy was assessed by assigning a 1 if the predicted output matched the answer exactly and assigning a 0 otherwise, then taking the proportion of correctly answered problems over total number of training examples.
- Our evaluation was carried out on both interpolated and extrapolated test sets. The interpolated test set comes from the same distribution as the training set, while the extrapolated test set is altered to include, for example, larger numbers or more function compositions. The idea is that if the algorithm performs well on the extrapolated test set, it is generalizing well, and so extrapolation performance can be used as an assessment of overfitting.
- Training was carried out on an NVIDIA Titan RTX.

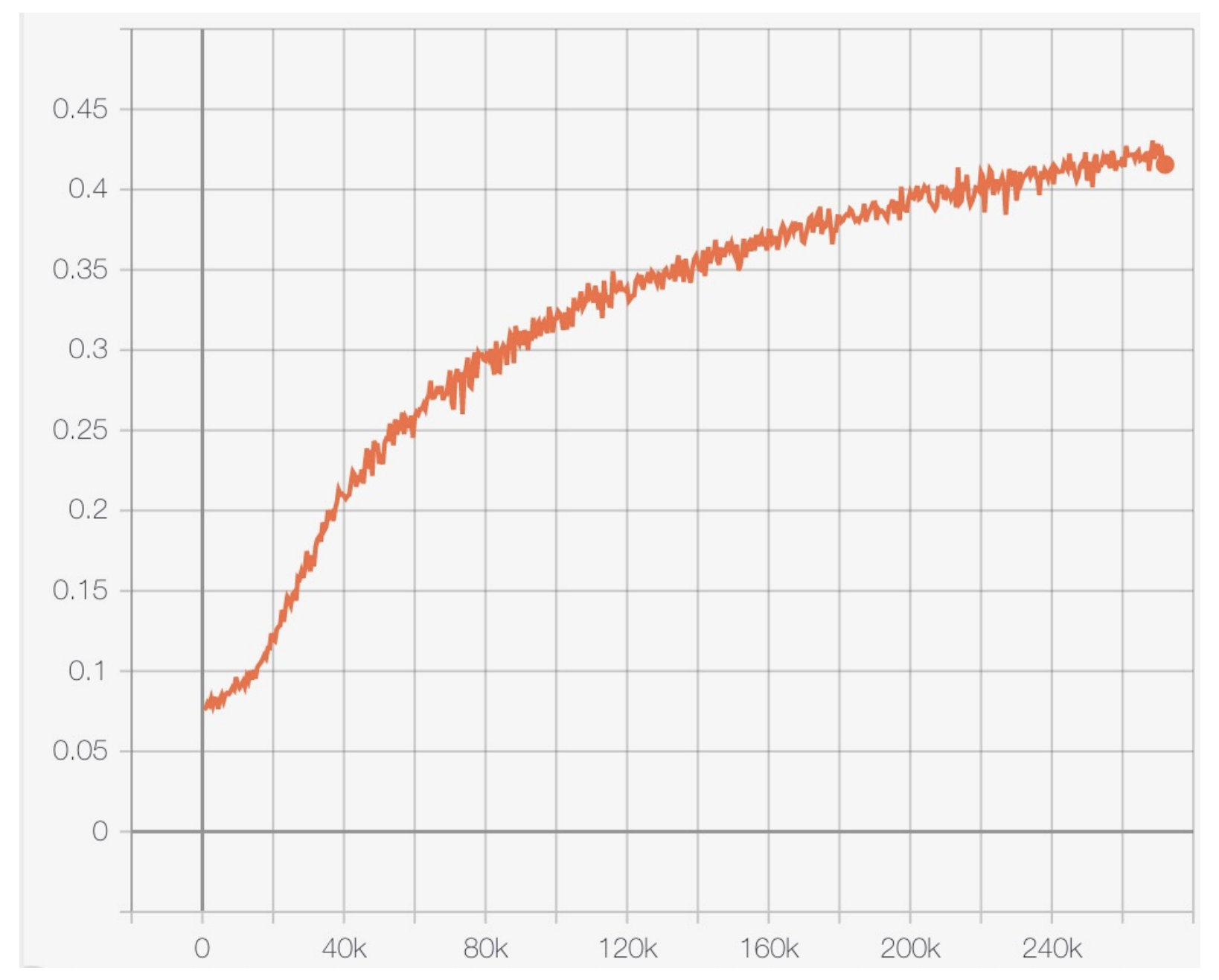


Figure: Interpolation accuracy - vertical axis is accuracy and horizontal axis is number of training epochs.

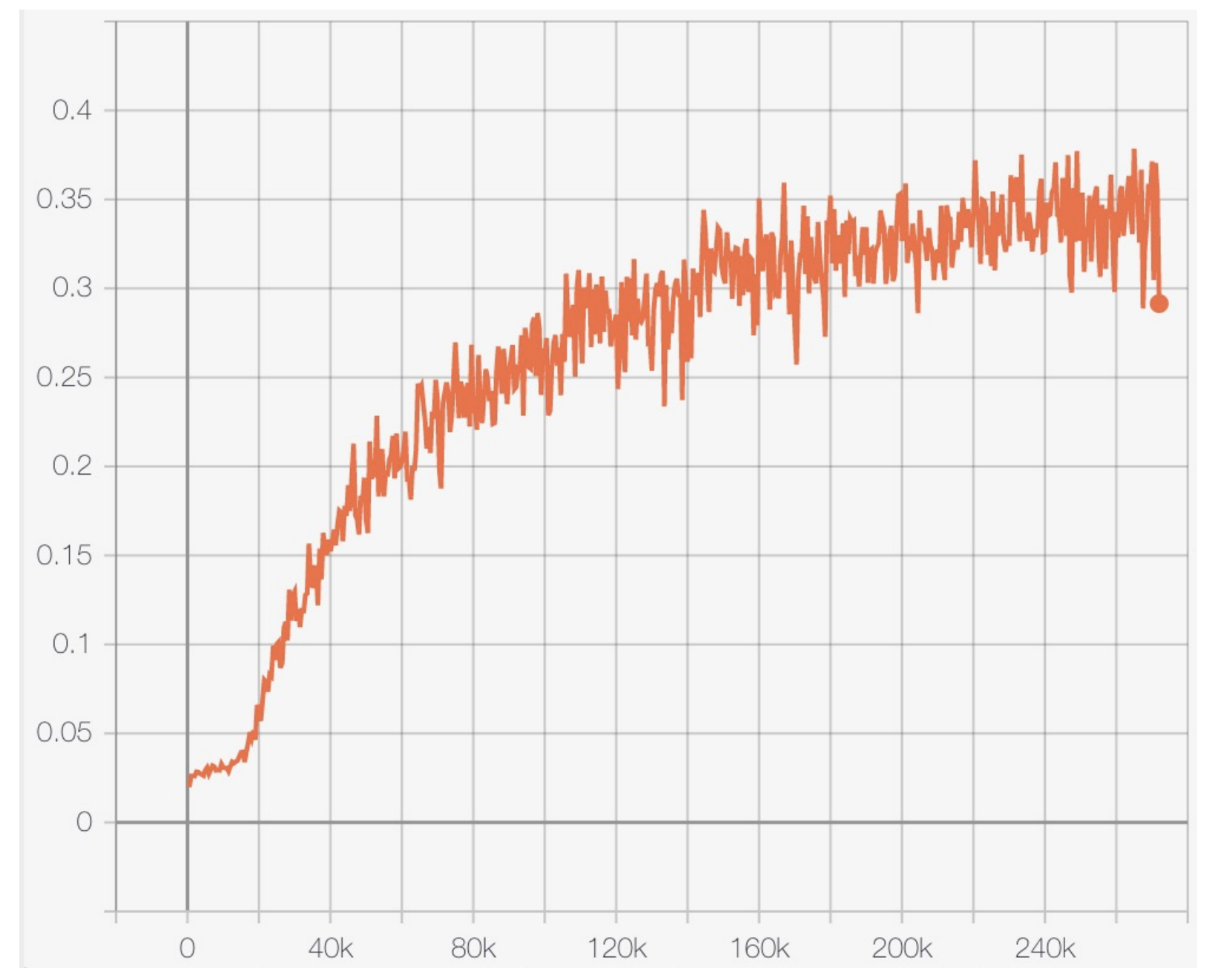


Figure: Extrapolation accuracy - vertical axis is accuracy and horizontal axis is number of training epochs.

- The plots show interpolation accuracy as high as **0.41** and extrapolation accuracy as high as **0.38**.