MATHEMATICAL REASONING OF AN ATTENTIONAL ENCODER DECODER NETWORK

PROJECT DEEP LEARNING

Alexis Stockinger
salexis@student.ethz.ch
15-916-034

Bernhard Walser
walserb@student.ethz.ch
16-195-241

January 16, 2020

ABSTRACT

Mathematical reasoning is difficult even for humans. Mathematical concepts have to be remembered and chained together to solve different problems. In this paper we give insight into how an Attentional Encoder-Decoder Network makes their decisions in two specific mathematical reasoning problems, namely arithmetic addition/subtraction and taking derivatives. The Architecture and Dataset is based on the paper from Saxton et. al. [1], where they show different general purpose models and a new dataset of different mathematical problems to train them and measure their performance. We show that an attentional encoder decoder architecture based on GRUs is able to learn specific rules of differentiation and basic arithmetic. This way it is possible for us humans to understand and reason about the predictions that were made by the model.

1 Introduction

Mathematical reasoning abilities of neural models based on attentional recurrent neural networks and transformer architectures are studied in the paper from Saxton et. al. [1]. As opposed to previous work, they cover a broad area of mathematics instead of narrowing to one specific subject, testing the general reasoning skills of the network. They obtain moderate performance overall, in particular areas requiring several intermediate calculations remain largely unsolved.

In their work they provide a new dataset which contains mathematical problems based on a national school mathematics curriculum (up to age 16) which should be used as a new benchmark for mathematical reasoning problems. As we initially wanted to investigate the performance of a Neural Turing Machine [2] on the same problem set, we soon discovered that training such a model is computationally really expensive and we do not have the time and resources to approach this problem. We therefore wanted to investigate how such Attentional Encoder Decoder Architectures make their decisions on such mathematical reasoning problems based on the input questions.

To reduce the computational complexity of our models we investigated the reasoning on two specific subproblems (arithmetic addition or subtraction and taking derivatives) and on one specific architecture the Attentional GRU architecture which is based on the Attentional LSTM architecture introduced in [1], but with less hidden units and less parameters. This architecture is explained in Section 2.

Finally we show how the output sentence of the Model correlates to the input sentence given to it and give some insights what they were able to learn in Section 3.

2 Models and Methods

The chosen architecture is based on the popular encoder-decoder attentional architecture published by Bahdanau et al. in 2014 [3], which is widely used for neural machine translation. It makes use of LSTM building blocks which were introduced by Hochreiter & Schmidhuber in 1997 [4] and have since then achieved many state-of-the-art results in different domains. Saxton et. al. [1] then used this architecture in the domain of mathematical reasoning on their new

developed dataset and were able to achieve moderate performance averaged over all different subproblems. To get good insight into what the models learn, we chose two subproblems for which their model was able to get good performance and investigate it with our computationally less demanding GRU Attentional model. GRUs are similar to LSTM units and were first introduced by Cho et al. in 2014 [5]. They combine the forget and input gate into a single update gate and merge the cell state with the hidden state. This leads to generally less parameters than in LSTM units which makes them potentially less powerful than them but also computationally less demanding.

2.1 Dataset

The dataset contains mathematical problems based on a national school mathematics curriculum (up to age 16). The dataset was synthetically generated such that linguistic variation or complexity is not included. The problems are given as a Question and the corresponding Answer in string format. When predicting the Answer, it has to match the given Answer character-for-character to be regarded as correct.

The dataset is provided by DeepMind in a GitHub repository[1], either as pre-generated dataset or as generator to have the possibility to generate more data if needed. For the generator they also make sure that data which is generated has a low probability to be generated twice, which is essential for the usability of the test data later.

From this entire dataset we chose two different subproblems which performed reasonably well on their model. Arithmetic Add or Sub describes problems which include simple addition or subtraction and Calculus Differentiate asks the model to find derivatives with respect to certain variables (See Figure 1).

```
Put together -22456 and 5.
-22451
Work out 225007 - 2.
225005
Add together -0.08 and 4954.
4953.92
What is 7 take away -45.4?
52.4
What is 601 less than -92.9?
-693.9
```

(a) Arithmetic Add or Sub

```
Find the first derivative of -1373*u**3 + 81 wrt u.
-4119*u**2
What is the derivative of -2612*k - 37?
-2612
Find the third derivative of w**6 - 2*w**5 + 579*w**4 - 5032*w**2.
120*w**3 - 120*w**2 + 13896*w
What is the third derivative of -70*f**5 + 6*f**2?
-4200*f**2
Differentiate 745*b**4 - 287 with respect to b.
2980*b**3
```

(b) Calculus Differentiate

Figure 1: Dataset

2.2 Attentional GRU Architecture

The Attentional GRU Architecture is basically an encoder-decoder architecture, where the encoder encodes a variable-length input vector into a single fixed length vector or here in distinct annotation vectors $(h_1, \ldots, h_T)$ and the decoder then decodes this vector/s into a variable-length target sentence. As opposed to previous encoder-decoder architectures [6][7], the model of Bahdanau et al. [3] relieves the encoder from encoding all the information in a single output vector and gives the decoder the possibility to use attention to select parts of the source sentence which he wants to pay attention to.

Notice the following explanation is based on the original paper of Bahdanau et al. 2014 [3].

The model starts by feeding the input question into the encoder, which then produces distinct annotations $(h_1, \ldots, h_T)$, from which the decoder then is trained to predict the next character y_i, given the context vector c_i and all previously predicted characters $\{y_1, \ldots, y_{i-1}\}$. This can be regarded as a probability over the translation y by decomposing the joint probability into the ordered conditionals:

$$p(y) = \prod_{i=1}^{T} p(y_i|\{y_1, \ldots, y_{i-1}\}, x) = \prod_{i=1}^{T} g(y_{i-1}, s_i, c_i) = \prod_{i=1}^{T} g(y_{i-1}, f(s_{i-1}, y_{i-1}, c_i), c_i)$$

where g is a nonlinear function that outputs the probability of y_i and s_i is an RNN hidden state for time i.

The context vector depends on a sequence of the annotations the encoder generated previously. Different than in the original paper of Bahdanau et al. [3] where each annotation h_i includes information about the whole input sequence,

[1] https://github.com/deepmind/mathematics_dataset

our annotations only include information of preceding characters of the i-th input character. This is because we do not use a bidirectional model like in the original paper [3], as in the paper from Saxton et. al. [1] they did not notice much difference in performance between those model architectures. The context vector c_i is then computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})} \qquad e_{ij} = a(s_{i-1}, h_j) \qquad a(s_{i-1}, h_j) = v_a^T \tanh\left(W_a s_{i-1} + U_a h_j\right)$$

where α_{ij} is the weight of each annotation and e_{ij} is an alignment model which scores how well the inputs around position j and the output at position i match. The alignment model is based on the hidden state s_{i-1} and the j-th annotation h_j of the input sentence. This can be implemented using a feedforward neural network which is jointly trained with all other components of the models.

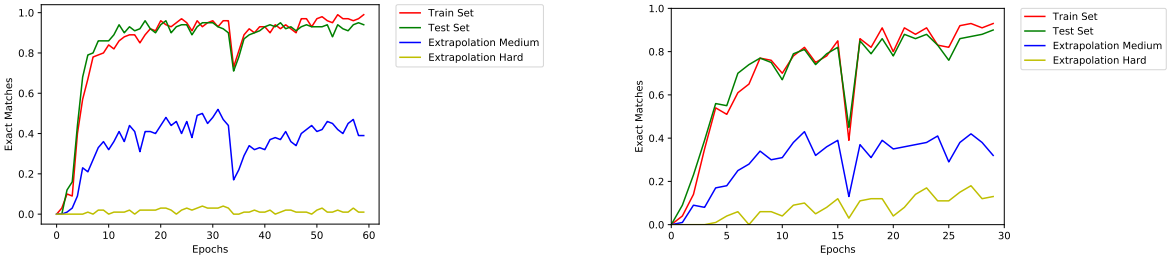
This context vector is then given together with the embedded input of the decoder into the GRU of the decoder, which then generates a prediction for our next character.

3 Results

3.1 Training and Comparison

We trained our models on a ASUS GeForce GTX 1080 Ti GPU for 4 hours (Arithmetic) and 2 hours (Calculus) with a batch size of 128 for 781 batches and there minimized the categorical crossentropy via the Adam optimizer. In Figure 2 we can see that the character by character exact matches between target label and prediction increased with further epochs, but stagnated at around epoch 20. For further comparisons and investigations we decided to use the model at epoch 30 for each problem.

We compare our Attentional GRU Model to the Attentional LSTM Model[2] in the paper of Saxton et. al. [1]. We can see that the interpolation values (data of same difficulty as in training) behave quite similar. For extrapolation (more difficult data than in training) our model can not keep up with the model from the paper.



(a) Arithmetic Add or Sub Training

(b) Calculus Differentiate Training

Figure 2: Training

	Task	Interpolation	Extrapolation
Attentional **GRU**	Arithmetic Add or Sub	0.94	$[0.01 - 0.39]$
Attentional **LSTM**	Arithmetic Add or Sub	$\approx$ **0.96**	$\approx$ **0.82**
Attentional **GRU**	Calculus Differentiate Training	0.9	$[0.13 - 0.32]$
Attentional **LSTM**	Calculus Differentiate Training	$\approx$ **0.97**	Not provided

Table 1: Baseline Comparison

3.2 Mathematical Reasoning of GRU Models

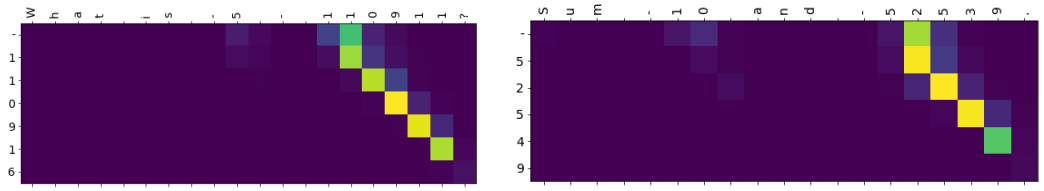
In the following we present the most interesting insights on how our models made predictions on certain questions. In each plot you can see the attention weights which the decoder used to make the prediction for each character. This

[2] Approximative Model performance taken from the Appendix in paper [1]

gives us some insight which input characters are most important for the prediction of each character and we can make suggestions on how the model uses them to take his decisions.

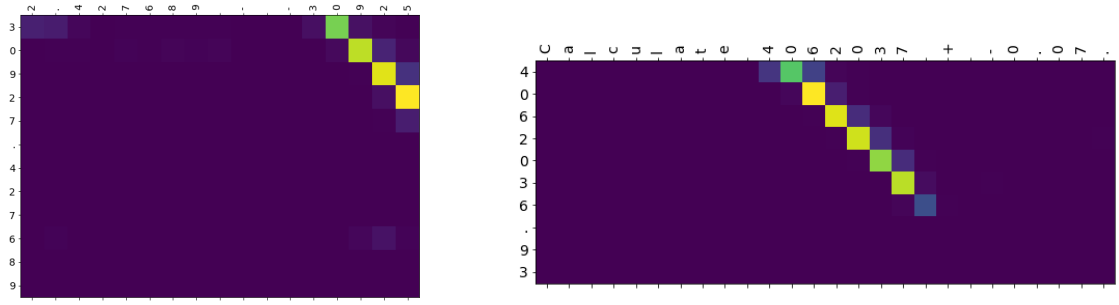
Arithmetic Add or Sub

Each annotation of a character includes information about preceding characters, therefore we see that it is possible for the model to copy characters by looking at the following character annotation and have a high attention weight for it. We assume that it learns to do calculations by copying those specific input sequences which do not change by the calculation and as soon as there is some calculation necessary for an output character, the model relies on its internal GRU state to compute it. This can be seen in Figure 3 and also in Figure 4, where the attention weights go to zero as soon as there is some computation necessary. Also interesting to see in Figure 4 is that it learns that two minus signs following each other are the same as an addition and also that a plus followed by a minus is canceled out and results in a subtraction.



(a) Basic Subtraction (b) Basic Addition

Figure 3: Interpolation Calculations



(a) Minus Minus (b) Plus Minus

Figure 4: Sign Comprehension

Calculus Differentiate

For the Calculus Differentiate task it is harder to reason why it takes certain decisions. Signs of the output are mostly explained by a corresponding sign in the input, which in a lot of cases is also a reasonable choice as the dataset does not include negative exponents, which could change the sign of the gradient. Also for constants in front of variables it gives attention to the exponent of the variable which influence the gradient in the output. One impressive example is given in Figure 5 where it computes the second derivative of $-3w^3 * z$ with respect to w as $-18 * w * z$. So it essentially learned to multiply the constant in front of the variable by the exponent of the variable and also decrease it such that the exponent gets 2 and multiply it again in the same step. As you look at the attention weights you can see that it has large attention in the exponent and simultaneously in the constant in front of the variable. It also learned to not remove the other variable z as it does not differentiate with respect to it.

Large Number Dilemma

Both models struggle to handle large numbers, especially larger number sizes which the models are not trained on. In the arithmetic add or sub problem this is mostly shown when the model is not able to do correct calculations with larger numbers, in Figure 6 (a) it gets stuck in a rounding of 79 to 80 which would not even be necessary. In the

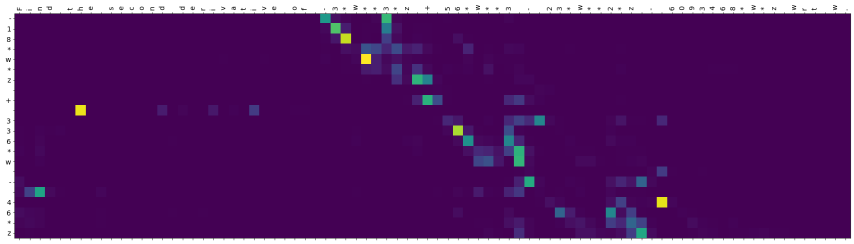
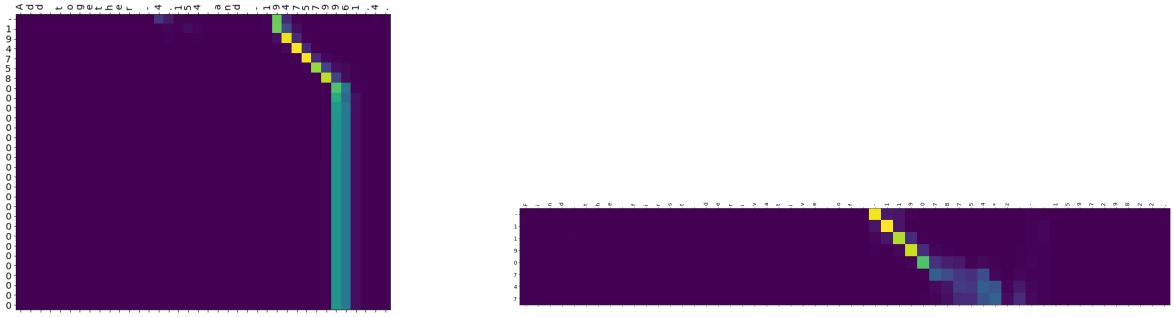


Figure 5: Impressive Gradient Calculation

calculating derivatives problem this is shown as the model is not even able to calculate first order derivatives of large numbers which only require copying the input sequence. In Figure 6 (b) it would require to calculate the derivative of $-119078754 * z - 159729822$ and it is only able to recover the first five characters correctly (Prediction: -1190747).



(a) Early Rounding

(b) First Derivative Larger Number Fail

Figure 6: Extrapolation Difficulties

4 Discussion

This paper has shown that even with this simpler model which uses GRU cells instead of LSTM cells, we were able to train mathematical reasoning problems with satisfactory performance. Unfortunately our simpler models lack even more in extrapolation performance than the ones with more parameters in the original paper. We think this difference mainly comes from three different sources, first our model uses GRU cells instead of LSTM cells which provide generally less parameters to fit and make the model less powerful, secondly our models had much shorter training time (about two hours per subproblem) with less computational power than in the original paper and finally the difference in training set size, where in the original paper they made use of 500M inputs as opposed in our paper we used 100K inputs per subproblem. And it is also worth mentioning that they trained a general purpose model, which should be suitable for every of their subproblems and therefore is a lot more difficult and computationally demanding to get right than our model which focuses just on specific subproblems.

5 Summary

This paper showed some insight in how attentional encoder decoder architectures make their decisions in mathematical reasoning problems. It is amazing what kind of rules the models are able to learn but sometimes they are not able to learn the most trivial things or fail to extrapolate to larger numbers when applying the previously learned rules. With the new dataset provided by DeepMind one can do similar reasoning approaches for different subproblems and we hope that mathematical reasoning of neural networks is largely understood in the near future.

References

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

[2] Ivo Danihelka Alex Graves, Greg Wayne. Neural turing machines. *arXiv:1410.5401*, 2014.

[3] Yoshua Bengio Dzmitry Bahdanau, KyungHyun Cho. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2016.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[5] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[6] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[7] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.