Project paper - Question Answering on Mathematics Dataset

Caterina Borzillo
1808187
borzillo.1808187@studenti.uniroma1.it

1 Introduction

The project I developed aims to solve the interesting task of question answering on a dataset containing mathematics questions and answers in a free-form textual input/output format at roughly school-level difficulty. The task is challenging because the mixed input format of the questions, for example "Solve 24 = 1601*c - 1605*c for c.", requires a deep knowledge and understanding of symbol rules, mathematics laws and axioms that are not easy to learn and to generalize without any basic knowledge. In this project paper I will explain what I implemented to solve this demanding task: one simple baseline method, an additional baseline method and the State-of-the-Art approach.

About the dataset representation Given that in the dataset the questions and the answers use a common alphabet of size 95 (upper and lower case characters, digits, and punctuation characters), I encoded each character of questions and answers following the ASCII standard character encoding. ASCII has just 128 code points, of which only 95 are printable characters (from 32 to 126) so I will use the 95 code points related to the 95 printable characters shifted by 32 positions to have the code points range going from 1 to 95 plus the zero (0) that is used for PAD; thus overall, my vocabulary size will be 95 + 1 (PAD=0). To see the ASCII table check on the Colab notebook of my project.

2 Baseline method: Simple LSTM

As first baseline method, I decided to implement a Simple LSTM. I take the general idea of the "Simple LSTM" method described by the original paper of the mathematics-dataset (https://arxiv.org/pdf/1904.01557v1.pdf).

The peculiarity of the Simple LSTM network is that, during training, the network receives in input not only the input encoded question but also the target answer, therefore it's like the network has a "teacher" that during training helps it to learn faster. During the validation and test steps, instead, the network makes use only of the input question in order to predict the answer.

About the Simple LSTM structure The network works in this way: it takes in input the mathematics question and the target answer (that are index sequences in which each index corresponds to a certain character) and transforms them in one-hot encoding inputs. After that, the input question is fed into the LSTM (with hidden_size = 2048) one character at a time and then, always one character at a time, the same LSTM net outputs the answer using as input - in the first time step - the last output of the initial LSTM and then - for the subsequent time steps - the target answers (if we are in the training loop) or the input question (if we are in the validation or test loop) until the predicted answer is generated.

About the training As loss function I used the binary cross entropy loss (BCEWithLogitsLoss), as optimizer the Adam optimizer with learning rate lr = 0.0001, beta1 = 0.9, beta2 = 0.995, epsilon = 1e-09. In this approach, as for all the others, the batch size is equal to 64, because after multiple attempts it seems to give the better results.

The model predicts the characters of the answer in a greedy-way, it means that the most probable character is chosen among all the other characters (using softmax function) for the generation of the output sequence.

To define and to train the model I used pytorch lightning in order to make the training, validation and test steps more easier to write and read.

The accuracy is computed by picking each predicted answer and comparing it with the target an-

swer with equality measure. The final accuracy is: num_correct_answers/num_tot_answers*100.

3 Additional baseline method: Transformer

As additional baseline I decided to implement a Transformer in order to be able to make a more comparable analysis with the SOTA approach that uses a TP-Transformer which is a variation of the standard Transformer. Transformers deal with sequential data and thus are very suitable for this task; moreover, the results that can be obtained by this type of network are outstanding compared with the other sequential nets like recurrent neural networks.

About the Transformer structure For the implementation of the Transformer I created multiple modules (to enhance the modularity of the code) and each of them has a specific function within the network. In general, the Transformer is made up by a stack of encoder and decoder layers (6 encoders + 6 decoders connected sequentially): the first encoder receives in input the mathematics question, previously passed into an embedding layer, and feeds it first into a Multi-Head Attention layer and then into a Feed-Forward layer; finally it sends the output to the next encoder; concerning the stack of decoders, the first decoder (like the first encoder) receives in input the target question (previously passed into an embedding layer) and feeds it into a first Multi-Head Attention Layer; then, a second Multi-Head Attention layer is used to combine both the result of the previous Multi-Head Attention layer as well as the output of the Encoder stack; in the end the decoder sends the overall output to the Feed-Forward layer.

The reason why the self-attention mechanism is fundamental and very important is that it is used to allow the model to focus on different parts of the input sequence when processing it, thus it allows the model to weigh the importance of different characters in the input sequence when making predictions. In this implementation the number of heads in the self-attention mechanism is equal to 8, it means that the model is able to attend to 8 parts of the input sequence simultaneously. In the Transformer I use 3 times the Multi-Head attention mechanism:

- In the encoders:

– the self-attention layer means that the input sequence pays attention to itself

- In the decoders:

– the first self-attention layer means that the target sequence pays attention to itself

– the second self-attention layer (called the encoder-decoder attention layer) means that the decoder, in order to generate the output answer, uses also the information achieved by the encoder from the input question.

About the loss function, the optimizer and all the hyperparameters they are the same used for the Simple LSTM (and will be the same also for the TP-Transformer).

About the embedding Concerning the embedding layer, the Transformer, as the TP-Transformer, uses two different embeddings: the input embedding (the characters embedding of the questions and answers) and the position encoding embedding. The second one is used in order to keep track of the position of each character in the sequence; this embedding is computed independently of the input sequence because the values depends only on the length of the sequence.

4 State-of-the-Art approach: TP-Transformer

As State-of-the-Art approach a TP-Transformer is implemented.

About the TP-Transformer structure In general, the architecture of the TP-Transformer (TP stands for "Tensor Product") is the same of the standard Transformer: it consists of a stack of Encoder layers and Decoder layers that adopt a self-attention process. The difference is that the TP-Transformer has a novel attention mechanism which is based on the idea of thinking the transformer as a graph neural network in which particular vectors, called relation vectors, are discovered and trained for searching the main relations between the characters of the question's sequence in order to generate the output answer. In other words, we can imagine to see the input question's characters as nodes of a graph that are initially separate and not connected by any edge; then, the task of the network and of the self attention mechanism is to learn to "build" those edges which represents a specific relation between 2 different nodes. This relation between one node and another is weighted depending on the strength of the attention (the value of the attention score) captured and learned by the network. An ex-

ample of a relation (example taken from https://arxiv.org/pdf/1910.06611.pdf) is the relation "second-argument-to" that the network discover due to the attention mechanism that "highlights" the denominators that appear in the input sequence.

5 Final considerations and comparisons

Here a detailed analysis on the results obtained by the 3 approaches: the 2 baselines (SimpleLSTM and Transformer) and the State-of-the-Art approach (TP-Transformer).

Concerning the first baseline method I decided to use a simple LSTM because I wanted to try to solve the problem in the most straightforward and direct way: an LSTM that, one character at a time, analyzes the input question and another LSTM that, always one character at a time, based on the results obtained by the previous LSTM and on the target answer, generates the output answer. To solve the task with this model first I tried to give in input to the model only one single type of sub-problem questions (algebra_linear_1d) and results were not as good as I expected: after 10 epochs of training (100 000 samples) the accuracy was about 5 %. It is like the LSTM was not able to capture the meaning of the characters in the input sequence and thus it is not able to generate the correct answer as well. I thing to say, however, is that, even if an answer has 2 out of 3 correct characters, that predicted answer is considered wrong, since there are no intermediary evaluations, it is either true or false and thus it is probable that there are some predicted answers that are mostly correct.

Then, in order to have a more various dataset with different types of sub-problems and to verify also the capacity of the model to generalize among different type of questions, I decided to use 3 sub-problem questions (algebra__linear_1d, arithmetic__add_or_sub, numbers__place_value) that, according to the test made by the original paper, are among the "easiest" problems to solve.

A note to mention is that in the mathematics-dataset paper, the publishers made their test experiments with 2 different types of test sets: the *interpolation test*, in which each question occurred in the training set is tested, and the *extrapolation test*, that measures generalization by testing question never seen during training. For this reason, I will report both the validation and test results on the double set of tests: for the interpolation test I used during validating and test loop the samples used during training; instead for the extrapolation test I used, as usual, samples that the network has never seen during training.

I take 10 000 samples for training, 6666 for validation and 3334 for testing for each sub-problem (the last 2 sets for extrapolation tests). As I expected the results are not good since the accuracy reached after 3 epochs is 3,7%; however some improvements in the results could be obtained by training the network for more and more epochs. On validation and test set (interpolation), the accuracies are 3,20% and 2,16% respectively; while on validation and test set (EXTRAPOLATION) the accuracies are lower, as expected (2,57% and 1,90%).

Concerning the second baseline method and the SOTA approach the results are completely different; it could be also due to the fact that the number of trainable parameters of the Simple LSTM is about an half with respect to the Transformer, thus Simple LSTM learns much less information. Or another thing that could make the difference in the results, in addition to the network structure and the self-attention mechanism, is that in the Simple LSTM the input and output sequences are encoded in one-hot vectors while in the other 2 approaches two types embeddings are used (as I specified earlier).

But most of all, it is like the self-attention mechanism plays a fundamental role in examining and analyzing this type of math-question sequences since the results cannot even be compared to those of the first baseline method. The table with the results of both Transformer and TP-Transformer is shown on the page below. As I write in the table description the accuracy results are related to the execution of the task not on 3 types of sub-problems (as the Simple LSTM) but on 5 types of sub-problems (trained for 10 epochs): 'algebra__linear_1d', 'arithmetic__add_or_sub', 'numbers__place_value', 'numbers__round_number', 'calculus__differentiate'. The reason behind this choice is because I wanted to increment the difficulty of the problem given that both the Transformer and the TP-Transformer achieve very good results.

As it's possible to notice, the SOTA approach gives accuracy results higher with respect to the Transformer approach. This means that the novel attention mechanism, which learns the relation

Accuracy results (%)					
Model	Train	Val (*inter*)	Val	Test (*inter*)	Test
Transformer (3 epochs)	22,00	20,36	16,36	13,74	12,13
Transformer (10 epochs)	39,50	37,65	34,87	32,49	32,72
TP-Transformer (10 epochs)	99,20	94,71	87,89	82,10	77,91

Table 1: Accuracy results obtained by the baseline model (Transformer) and by the SOTA approach (TP-Transformer) on 5 types of different sub-problems. Word "inter" stands for interpolation test, if it is nothing written in brackets means that they are the results on the extrapolation test. In the table are present also the results of the Transformer trained for 3 epochs in order to show that the more the network trains, the more the accuracy increases.

vectors between characters in the input and output sequence does its job well. It's like the TP-Transformer in some way learns the structure of the mathematical questions (for example it learns where the numerators and the denominators are in the input sequence) and is able to successfully generate the correct answer most of the time. The fact that the Transformer and the TP-Transformer networks use a different attention mechanism, in particular, the latter requires the computation of the additional vector called relation vector, implies that the total number of trainable parameters in the SOTA approach is higher with respect to the Transformer and most likely, this is what makes the difference in the results.

A final thing to say about the transformers is that, while during training the model decoder receives as input the target answer, during the validation and the test loop it receives not the target question but the input question, thus the "help" that the model receives during training is not received during the validation and test step; this in some way explain the reason why with the validation and test step less good results are obtained.