ProblemSolver at SemEval-2019 Task 10: Sequence-to-Sequence Learning and Expression Trees

Xuefeng Luo, Alina Baranova, Jonas Biegert
Linguistics Department, University of Tuebingen, Germany
firstname.lastname@student.uni-tuebingen.de

Abstract

This paper describes our participation in SemEval-2019 shared task "Math Question Answering", where the aim is to create a program that could solve the Math SAT questions automatically as accurately as possible. We went with a dual-pronged approach, building a Sequence-to-Sequence Neural Network pre-trained with augmented data that could answer all categories of questions and a Tree system, which can only answer a certain type of questions. The systems did not perform well on the entire test data given in the task, but did decently on the questions they were actually capable of answering. The Sequence-to-Sequence Neural Network model managed to get slightly better than our baseline of guessing "A" for every question, while the Tree system additionally improved the results.

1 Introduction

The data set for the task (Hopkins et al., 2019) includes questions used in the Math SAT. There are three broad categories of questions: closed-vocabulary and open-vocabulary algebra questions, and geometry questions. All types of questions consist in large part of natural language. Closed-vocabulary algebra questions typically contain math equations and many math-specific words, while open-vocabulary algebra questions include more everyday vocabulary and quantities which are expressed in letters, numbers or a combination of both. Geometry questions are usually provided with diagrams the analysis of which is necessary for solving the problems. Most questions of all categories are multiple-choice questions with five possible options, but some questions have a numeric answer.

We present two systems to tackle these math problems. One of them, a sequence-to-sequence LSTM model pre-trained with augmented data, is applied to all three types of questions, while the other, a system based on expression trees produces answers exclusively for open-vocabulary algebra questions.

2 Related Work

In their work, Roy and Roth (2015) introduced binary *expression trees* that represent and solve math word problems. To choose the best expression tree out of all possible trees, the authors employed two classifiers: a relevance classifier, which determined if the quantity should be included into the expression tree, and a Lowest Common Ancestor (LCA) classifier, which output the most probable mathematical operation for a pair of quantities. Both classifiers were trained on gold annotations.

Subsequently, two other systems were developed based on Roy and Roth (2015). One of the systems belongs to the same authors and uses the concept of Unit Dependency Graphs (UDGs) to capture the information between units of the quantities (Roy and Roth, 2017). UDGs are then united with expression trees, allowing the information about dependencies between units improve the math problem solver.

Another system (Wang et al., 2018) suggests a method to improve Roy and Roth's approach. By applying deep reinforcement learning, which has proved to be suitable for problems with big search space, the authors achieve better accuracy and efficiency.

An earlier system introduced by Wang et al. (2017) used gated recurrent units (GRU, Chung et al., 2014) and long short-memory (LSTM, Hochreiter and Schmidhuber, 1997) to automatically solve simple math word problems by converting words into math equations.

Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval-2019), pages 1292–1296
Minneapolis, Minnesota, USA, June 6–7, 2019.

3 Model Description

3.1 Sequence-to-Sequence Neural Network

Our model is based on a sample implementation provided by the Keras team (Chollet et al., 2015). This model was able to calculate addition, such as from "535+61" to "596" with a Sequence-to-Sequence model using LSTM (Hochreiter and Schmidhuber, 1997). Similar to this model, our model also had 128 hidden units and started with an embedding layer with an alphabet of 96 characters. The longest question was 650 characters. Then, we used a LSTM as encoder. For all dights in the answers, we have seperate vectors representing them. Thus, we have repeated vectors of outputs 5 time, in order to represent 5 dights. Our decoder was another LSTM layer with returing sequences, followed by a time distributed layer of dense layers where activation function was softmax. In addition to this, we added a 0.2-rate Dropout layer (Srivastava et al., 2014) after embeding layer, encoder LSTM and decoder LSTM, to prevent over-fitting. On top of that, we found that reversing and doubling the inputs can greatly improve training performance, according to Zaremba and Sutskever (2015). The seq2seq model is shown by Figure 1. We did not encoded answers along with questions. We only compared the answers strings to the questions' choices and made our decisions.

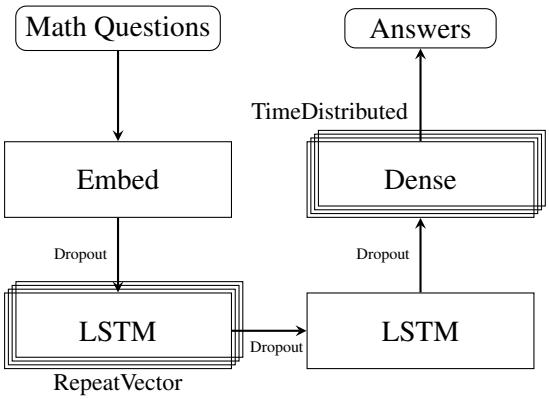

Figure 1: Seq2seq Model

We padded all answers into same length with extra space characters, but our model still was not able to produce exact answers with sequence-to-sequence. However, the sequences the model produced were good enough to predict the correct answer for multiple choice questions. For instance, for question "If $x+345 = 111$, what is the value of x?", the output of the system would be "-234444", which is very close to the correct answer "-234". Thus, we wrote a program which was able to compare the initial characters (including "-") regardless the extra characters at the end with answer options and predict the correct answer.

3.2 Tree System

The system of Roy and Roth (2015) has a lot of advantages: for instance, it can solve math problems that require multiple steps and different operations, and it can handle problems even if it did not see similar problems in the training set. That is why we chose to implement this approach for solving open-vocabulary algebra questions.

The expression trees the authors used in their system have a special structure that allows to calculate them in a simple and unambiguous way. In such a tree, the leaves are quantities extracted from the problem text, and the internal nodes are mathematical operations between the quantities. By calculating values of all internal nodes, one can obtain the value of the tree route, which corresponds to the answer of the problem.

Similarly to Roy and Roth (2015), we used the relevance and the LCA classifiers to evaluate all possible expression trees and choose the one that answers the problem correctly. However, instead of using gold annotations, we decided to train the classifiers on all the trees that result in right answers, partly because annotations were not available, and partly because we were curious how well the system could perform with no manual effort invested in annotating training data.

Tree evaluation was done by two simple multilayer perceptrons. As described earlier, the first one returns the probability of a given quantity to be relevant, as in a tree that answers the question correctly contains that quantity, the second one returns the probabilities for each of the possible operations to be the lowest common ancestor of a pair of given quantities in a tree that answers the question correctly.

For every possible tree per question, the product of the probabilities of each quantity to be relevant was added to the product of the probabilities of the lowest common ancestor of each quantity pair being correct. These scores, as well as the result of the tree were put in a list and ordered by score. The results of the trees were then matched against the answer options of each question and the answer

option that was first matched in the list was given as the answer to the question. If the question had no answer options, the result of the highest rated tree was given as the answer to the question.

4 Experimental Settings

4.1 Sequence-to-Sequence Neural Network

4.1.1 Data Augmentation

Initially, we tried to trained our model directly on the questions, but it turned out that model could not learn at all. In total, we had slightly more than 1000 SAT questions, which was insufficient for an RNN model. Not to mention that the small training set contained questions with a variety of types – open- and closed-vocabulary algebra questions as well as geometry questions, leaving an even smaller training set for each subtype. Thus, data augmentation was a necessary step. In order to strengthen the connection of numbers, we did not provide original SAT data with numbers modified, but more than 600,000 simple closed-vocabulary algebra questions.

Among them, there were two types of questions augmented for our model. These included two types of questions within 3 digits like "If $x + 345 = 111$, what is the value of x?" and "If $x - 345 = 111$, what is the value of x?". Not only numbers and variable names were randomized but the positions of variables were switched. In toal, there were 614,236 questions, where "plus" had 330,620 and "minus" had 283,616 questions. Even though augmented data had large differences with SAT questions, results showed they still prodigiously contributed to our training.

4.1.2 Training

Rather than training our model together with original SAT data and augmented data, we chose to trained with augmented data first, and then continued to train with original data. There were 40 iterations of 614,236 questions dealing with addition and subtraction. Fractions were also present in the training set. After training with the augmented questions set, our model was trained with actual questions from the Math SAT. In total, there were 200 iterations of 805 Math SAT Questions. Nevertheless, since the training data was so small, it is highly possible that our model was prone to over-fitting to the training data.

Example 1
On a certain map, 100 miles is represented by 1 inch. What is the number of miles represented by 2.4 inches on this map?

4.2 Tree System

4.2.1 Quantities

Quantities were extracted from questions and answers using a rule-based approach. Before the extraction, all mentions of quantities were normalized to digits (e.g. *one* to *1*). Then, numbers, number-word combinations (e.g. *13-inch*), small letters denoting quantities (all letters except *a*) and LaTeX expressions were retrieved. LaTeX expressions that contained only numbers were transformed into numbers (e.g. *\frac{1}{10}* into *0.1*). In general, all questions that contained quantities other than numbers or the answer of which had several quantities were filtered out, leaving us with 75% of open-vocabulary questions from the training set. In the next stage, while constructing trees for the training data, we heuristically set the maximum number of quantities in a question to 7, which led to using 59% of the training data.

4.2.2 Operations and Tree Enumeration

Once quantities from the question were extracted, all their possible combinations were obtained, with size from two to the total number of quantities. The order of quantities in these combinations, however, stayed the same. Consider Example 1. For this word problem, the combination [100 2.4] would be possible, but the combination [2.4 100] would not.

For every combination obtained in the previous step, all possible expression trees with quantities as leaves and empty inner nodes were generated. These inner nodes were filled with all possible combinations of operation signs. As in earlier studies (Roy and Roth, 2017; Wang et al., 2018), we used six operations: apart from the standard $+$, $-$, $\times$ and $\div$, we included reverse operators $-_{\text{rev}}$ and $\div_{\text{rev}}$ to account for the fact that the order of quantities stays the same in their combinations.

Like Roy and Roth (2015), we implemented constraints that define *monotonic* trees. These constraints are concerned with the order of multiplication operator in regard to division operator, and the order of addition operator in relation to subtraction operator. However, unlike the authors, we used these constraints to decrease the number

System	Accuracy
Baseline (always "A")	14.3%
Baseline + seq2seq	15.0%
Baseline + trees	15.9%
Baseline + seq2seq + trees	16.7%

Table 1: Results

of trees resulting in right answers, not to guarantee that any monotonic tree for the solution expression has the same LCA operation for any pair of quantities in it, as in Roy and Roth (2015).

4.2.3 Features

We used UDPipe (Straka and Straková, 2017) to parse questions' text and extract features for the classifiers. The features are identical to the ones that Roy and Roth (2015) describe.

5 Results

The results that our systems achieved are shown in Table 1. Our official submission consists only of the neural network, which achieved 15%, with the accuracy on closed-vocabulary algebra questions being 16% and the accuracy on the other two categories being 15% each. This result, however, was achieved by guessing "A" whenever the question could not be answer by the model. When guessing is removed, the overall accuracy drops to 2%. However, on the 109 questions the model could actually answer, it achieved 21% accuracy.

In post-evaluation, after combining the results of the neural network and the tree system, we were able to achieve 17% accuracy overall by increasing the accuracy on open-vocabulary algebra questions by 20%. If we remove the guessing, the tree system achieves 3% accuracy overall, which stems from its 13% accuracy on open-vocabulary algebra questions. If we only count the questions that could actually be answered by the system, its accuracy would be equal to 26%. Without guessing, the combination of both systems produces 4% accuracy overall, with the distribution being 2% on closed-vocabulary algebra questions, 13% on open-vocabulary algebra questions and 0.4% on geometry questions. On the 205 questions answered by the combination of both systems, the accuracy was 23%.

6 Discussion/Conclusion

The results of our systems on the full data set are, frankly put, rather poor. Nevertheless, the tree system shows promising results in solving open-vocabulary questions, if it is refined and improved, while the neural network seems not to perform well on any specific type of questions, although its overall performance is similar to that of the tree system.

Concerning the neural network, it might be beneficial to focuse on specific types of questions, instead of trying to train a model that deals with mixed types of questions. RNN best learnt on closed- and open-vocabulary algebra questions, therefore training separate models for these types could be one way to improve the system. In addition to that, a much larger dataset is critical in enhancing the model, thus promoting the accuracy of its predictions. Lastly, data augmentation would further improve the model. If we were to train a versatile model for mixed types of math questions, we could perform data augmentation on each type.

The current problem of the tree system lies to a large extent within the quality of the tree evaluation. It heavily relies on answer options to be available, as the average index of the first tree that produces an answer option in the score list is 47 (for the test data). Therefore, the answer of the highest-rated tree would most likely be wrong. Other aspects that could be improved include choosing other features for the classifiers, decreasing the scores of trees with low amounts of quantities (those trees are currently overrated) or using a different machine learning algorithm altogether, such as deep reinforcement learning (e.g Wang et al., 2018).

Apart from that, using no additional quantities in constructing trees, and including every quantity once made it difficult to obtain trees that not only gave the right result for the questions from the training set, but also answered them in a right way. Moreover, expanding expression trees to problems that involve letters to denote quantities would definitely contribute to improving the performance of the tree system.

7 Acknowledgements

Part of the experiments reported on this paper was run on a Titan Xp donated by the NVIDIA Corporation.

References

François Chollet et al. 2015. Keras. https://keras.io.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

Sepp Hochreiter and Jrgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Mark Hopkins, Ronan Le Bras, Cristian Petrescu-Prahova, Gabriel Stanovsky, Hannaneh Hajishirzi, and Rik Koncel-Kedziorski. 2019. Semeval-2019 task 10: Math question answering. In *Proceedings of International Workshop on Semantic Evaluation (SemEval-2019)*, Minneapolis, USA.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752. Association for Computational Linguistics.

Subhro Roy and Dan Roth. 2017. Unit dependency graph and its application to arithmetic word problem solving. In *AAAI*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Milan Straka and Jana Straková. 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *AAAI*.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854.

Wojciech Zaremba and Ilya Sutskever. 2015. Learning to execute. *Computing Research Repository*, arXiv:1410.4615. Version 3.