Help Transformer Improve Performance in Automatic Mathematics Word Problem-Solving

DONG LIU1, GUANFANG WANG2, AND JIALIANG YANG2

1Department of Information Technology, Luoyang Normal University, Luoyang, Henan 471022, China
2Geneis Beijing Company Ltd., Beijing 100000, China

Corresponding authors: Dong Liu (liudong@lynu.edu.cn) and Jialiang Yang (yangjl@geneis.cn)

ABSTRACT Solving Mathematics Word Problem (MWP) is a basic ability of humanity, which can be mastered by most students at a young age. The existing artificial intelligence system is not good enough in numerical questions, like MWP. The hard part of this problem is translating natural language sentences in MWP into mathematical expressions or equations. In recent researches, the Transformer network, which proved a great success in machine translation, is applied to automatic mathematic word problem-solving. While previous works have only shown the ability of Transformer model in MWP, how multiple factors such as encoding, decoding, and pre-training affect the performance of Transformer model has not received enough attention. The study is the first to examine the role of these factors experimentally. This paper proposes several methods to improve Transformer network performance in MWP under the basis of previous studies, achieves higher accuracy compared to the previous state of the art. Pre-training on target tasks dataset improves the translation quality of the Transformer model greatly. Different token encoding and search algorithms also benefit prediction accuracy at the expense of more training and testing time.

INDEX TERMS Machine translation, mathematics word problem, Seq2Seq model, transformer.

I. INTRODUCTION

In recent years, lots of machine reading comprehension (MRC) models [1], [2], [3] have been proposed, and achieved remarkable results in various public benchmarks such as SQuAD [4] and RACE [5]. However, numerical reasoning remains a challenge for most of existing systems. Especially, their exact match accuracy drops dramatically when it comes to numerical questions containing addition (+), subtraction (−), multiplication (∗), and division (∗) over numbers [6]. An artificial intelligent agent or system should be able to master such problems i.e. mathematics word problem (MWP) that most students can solve at a young age [42].

Mathematics word problems commonly describe a real-world state and pose questions about it. Table 1 shows one such problem. People have been asking to answer how many groups they could make according to the text. An MWP auto-solver needs to complete three steps: 1, Identifying operands (26, 46 and 9); 2, Identifying operators (+, /), and arranging these elements’ order correctly to give the math expression; 3, Evaluating the expression for a result as an answer. The 2nd step is the most challenging thing for an artificial intelligence system. This paper focuses on the 2nd step.

Automatically solving MWP has attracted a lot of research attention and has been considering as a way of evaluating machines’ ability. Previous work on automatic mathematics word problem solving can be roughly dividing into three categories: symbolic approaches, statistical learning approaches, and sequence to sequence(seq2seq) modeling approaches.
Early proposals can be dated back to the 1960s. Bobrow [7] describes a computer program called STUDENT to handle English algebraic problems. Charniak [8] proposes a similar program based on STUDENT. Both of them use pattern matching to give mathematical expressions based on a set of transformation patterns and rules. These are early typical representatives of symbolic approaches. Another symbolic approach, called semantic parsing techniques, which is at the heart of natural language understanding, has been used for mathematical word problems. Liguda and Pfeiffer [9] propose modeling MWP with augmented semantic networks. Shi et al. [10] implement a semantic parser based on lots of grammar rules to solve MWP, which is very effective for the specific type of math problems. Early application of semantic parsing techniques has mainly been in compositional question answering [11]. These models have achieved high performance nearing human performance [12], [13], yet failed to symbolic discrete reasoning, including arithmetic, counting, and sorting. Some more challenging datasets and models have been proposed [6], [14], and it is quite natural for researchers to apply semantic parsing techniques to these numerical reasoning problems over numbers [15], [16], [17].

Statistical machine learning methods have been used to solve MWP since 2014. Hosseini et al. [18] solve addition and subtraction problems by learning verb categories from the training data. Kushman et al. [19] extract templates of math expressions from the training data, train models to select templates, and then map quantities in the problem to the slots in the template. Similarly, Zhou et al. [20] generalized equations attached to problems with variable slots and number slots, and learning a probabilistic model for finding the best solution equation. Roy and Roth [21] solve arithmetic problems with multiple steps and operations by mapping quantities and words to candidate equation trees and selecting the highest probability tree. By using Tree-RNN (tree-based Recursive Neural Networks), Zaporojets et al. [41] score a list of potential candidate equations which are generated by Integer Linear Programming optimization algorithm. The equation with the highest score is chosen as the solution. These methods can achieve high accuracy in limited math problem categories and requirements to design manually different features, templates, or rules.

Seq2seq modeling approaches transform natural language sentences in MWP to mathematical expressions or equations. Wang et al. use an RNN to encode the word problem to a context vector and another RNN to decode the context vector to an equation template [22]. Huang et al. [23] use a deep reinforcement learning model to achieve character placement in both seen and novel equation templates. Xie and Sun [24] design to generate expression trees by explicitly modeling the tree-structured relationship between quantities, which mimic the goal-driven mechanism in human problem-solving. Griffith et al. [25] recognize MWP as a machine translation task and use Transformer networks to translate mathematics word problems to equivalent arithmetic expressions in three notations (i.e. prefix, postfix, and infix). Liu et al. [39] use the double-checking mechanism and combine reverse operation based data augmentation with seq2seq models to learn the reasoning logic. Liang and Zhang [40] design a teacher module to make MWP encoding vector more closely match the correct solution. Their method can separate those representations of MWP with different solutions but similar expression. An advantage of these end-to-end approaches exists in that they do not rely on hand-crafted features.

This paper belongs to the end-to-end category. Our approach is strongly getting tied with the work of Griffith et al. [25]. We reproduce their work based on the code that can be found at https://github.com/kadengriffith/MWP-Automatic-Solver, and improve it with our ideas. Our contributions are summarized here:

1) A novel pre-training step is proposed for solving MWP. The experiment result shows that it can affect remarkably translation quality.
2) Different encoder is verified and compared. The sub-word encoding commonly used in natural language processing (NLP), is discovered to be less effective than token encoding.
3) Our model use top-k beam search strategy other than greedy search in prediction decoding stage and result are reported.
4) We ran the whole computer program and achieved similar accuracy on the same datasets. However, by going carefully through the error cases in the test result, we found some defects in the code. The model performance is increased with those corrected and significantly outperforms state-of-the-art models on several open datasets for MWP.

The remaining content of this paper is presented as follows. In Section II, we describe the methods of our proposed MWP automatic solver and the datasets used as a benchmark test. The experiment setting and ablation results are reported in Section III. Section IV offers a conclusion about our works. We discuss future work in Section V.

II. DATASETS AND METHODS

We view the transition from a mathematics word problem to arithmetic expression as a sequence-to-sequence translation problem. The Transformer-based machine translation model is still the most popular nowadays. Transformer network was introduced by Vaswani et al. [26], which composes of encoder and decoder modules. Models based on Transformer have achieved state-of-the-art performance in many NLP tasks. We employ the same Transformer architecture for language processing to handle MWP. Our works focus on pre-training and testing stages to make the Transformer network perform relatively better. The next subsections describe the datasets and methods used in this paper.
TABLE 2. MWP datasets and the number of steps with relevant operations.

| Datasets | Problems | Relevant operations | Steps |
|----------|----------|---------------------|-------|
| AI2      | 391      | +, −                | 1 or 2|
| CC       | 600      | +, −, ×, ÷         | 2     |
| IL       | 562      | +, −, ×, ÷         | 1     |
| MAWPS    | 2373     | +, −, ×, ÷         | 2     |
| Dophin18K| 3567     | +, −, ×, ÷         | NaN   |

A. DATASETS

We train and evaluate our method on four individual datasets. All of them contain word problems that involve +, −, ×, ÷ four arithmetical operations. A summary of the datasets is presented in Table 2.

AI2 dataset was produced by Allen Institute for AI and Hannaneh Hajishirzi [27]. The focus is on high school level questions. CC (Common Core) dataset was gathered by the Cognitive Computation Group at the University of Pennsylvania [21]. IL dataset contain elementary math word problems with 1-step calculation [28]. Koncel-Kedziorski et al. [29] amassed 3320 problems, built an online repository, called MAWPS, which allows for the automatic construction of datasets with particular characteristics. We select a subset of the data excluding some more complex problems that generate system equations. Huang et al. [23] introduced a relatively large collection, Dophin18K, which contains over 18,000 annotated math word problems, and is constructed by semi-automatically obtained from community question-answering web pages. We use Dophin18K as a pre-training dataset, instead of a test set, that the intent here is to learn more knowledge about the target task, so we do not count of the steps for it, marked as NaN in Table 2. Thanks to Griffith et al. [25], all of these datasets can be centrally download from Github (https://github.com/kadengriffith/MWP-Automatic-Solver).

B. METHODS

A complete process is shown in Fig. 1. Several of these modules are described in the following subsections in detail.

Complete pre-processing, training and prediction process

The Transformer model has been shown to work well for a myriad of applications, including reading comprehension, machine translation, text classification, etc. It is designed based on the attention mechanism, especially its unique multi-head self-attention mechanism. As shown in the Fig. 1, Transformer is a seq2seq model including encoder and decoder. The input of its encoder part is a sequence of word embedding vectors containing both symbolic and positional embedding. The encoder part consists of several modules, each of which contains a multi-head self-attention layer and a feed-forward neural network, and makes it possible to build deeper neural networks by adding residual couplings. The output of the encoder will be saved as the initial input of the decoder. The output of the decoder will be passed through linear and softmax layers to obtain a probability distribution to predict the output symbols. The decoder part also consists of several decoder modules. The structure of Transformer eliminates the time dependence in processing sequences which makes it possible to build large-scale parallel and deeper neural networks.

C. REPRESENTATION CONVERSION

All examples in the datasets are extracted out into two parts: questions and equations, omitting everything not pertinent to the translation. Following Griffith et al. [25], we process the examples in the next two steps:

1) Converting all equations to prefix expressions. An issue is that the Seq2Seq-based models may generate invalid expressions for a reason not to be able to match up left and right parentheses, such as (16 + 39) × 5. This issue can be solved by using the post-order traversal of expression tree as target sequence ([30]), did pre-order traversal as well. Instead of postfix expressions, we choose prefix expressions because this representation performs best according to the report of Griffith et al. [25].

2) Replacing each numeric value in questions with a corresponding tag (e.g., x, y, z, etc.). Numbers in the datasets are unique and rare and affect the network generalization. A mapping relation is built and remembered between the numbers and tags. Transformer network learned an arithmetic expression like × x + y z. The corresponding numbers are switched back before comparing with the correct answer.

D. SUBWORDENCODER VS. TOKENTEXTENCODER

We investigate two example encoders, SubwordTextEncoder and TokenTextEncoder in the TensorFlow Datasets package. SubwordTextEncoder has an advantage over TokenTextEncoder in addressing the following problems: 1, OOV (out of vocabulary) problem. Tokens are simply replaced by a special <unk> token in TokenTextEncoder. 2, TokenTextEncoder
lost information regarding word structure, such as old, older, and oldest, but SubWordTextEncoder is able to produce an appropriate granularity between character and word, and is an effective way to alleviate the open vocabulary problems in a variety of NLP tasks, such as neural machine translation. However, according to our studies, we found that the two questions mentioned above do not exist in MWPs. Firstly, the vocabulary is small, especially for mathematic expression language. OOV problem has little influence on the results. Secondly, words structure information does not have a significant effect on model performance. Our experiment results also proved that the model with TextTokenEncoder performs better than with SubWordTextEncoder, however, has slower convergence. Sparse categorical cross-entropy is used as an objective function. The model’s loss is adjusted according to the mean of the translation accuracy after predicting every determined subword or token in a translation. The loss function is given by

$$\sum_{i=1}^{N} \frac{1}{J} \sum_{j=1}^{J} \left( - \sum_{t=1}^{T} \text{target}_{j,t} \times \log P(j \in t) \right)$$

(1)

where \(N\) is the number of examples, \(J\) is the length of produced express, and \(T\) is the number of all vocabulary tokens.

E. PRE-TRAINING: GENERAL-DOMAIN CORPUS VS. TARGET TASK DATA

Transfer learning has had a large impact on computer vision (CV). A general inductive transfer learning setting for NLP [31] is: Given a source task \(T_s\) and any target \(T_t\) with \(T_s \neq T_t\), we would like to improve performance on \(T_t\). Generally, language modeling (LM) can be seen as the ideal source task. A pre-trained LM can be easily adapted to the idiosyncrasies of a target task. Not so in the work of Griffith et al. [25], however, they pre-trained the same Transformer models on IMDB Movie Reviews dataset [32]. The average BLEU-2 scores of pre-trained models is 91.06, compared to 91.96 of non-pre-trained models.

Different from a general-domain corpus, we pre-train our model on target task data. Fine-tuning language model on the data of the target task converges faster as it only needs to adapt to the idiosyncrasies of the target data, and allows us to train a robust LM even for small datasets according to ULMFiT [34]. We select Dophin18K dataset because it requires a wider variety of arithmetic operators, consisting of web-answered questions from Yahoo! Answers. To make this a fair comparison, we evaluate the average BLEU-2 score in web-answered questions from Yahoo! Answers. To make this requires a wider variety of arithmetic operators, consisting of

$$\text{score}(y_1, \ldots, y_t) = \log P(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P(y_i | y_1, \ldots, y_{i-1}, x)$$

(3)

where scores are all negative and higher score is better.

F. DECODING: GREEDY VS. TOP-K BEAM SEARCH

In the prediction stage, the Transformer model uses greedy or top-k beam search. Many sophisticated algorithms can be used to generate (or “decode”) the target sentence. For example, top-k random sampling scheme [35] is found more effective than top-k beam search, because sentences produced by beam search tend to be short and generic. However, for the particularity of the MWPs, i.e. arithmetic expression usually is short, we speculate that top-k beam search might be more suitable for the MWPs. We compare greedy search and top-k beam search in our experiments. Formally, we describe top-k beam search below (where \(k\) is the beam size, and 2 in our experiments):

1) On each step of the decoder, keep track of the \(k\) most probable translations (called hypotheses).

2) A hypothesis \(y_1, \ldots, y_t\) has a score which is its log probability

$$\text{score}(y_1, \ldots, y_t) = \log P(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P(y_i | y_1, \ldots, y_{i-1}, x)$$

(3)

where scores are all negative and higher score is better.

3) We search for high-scoring hypotheses, tracking top \(k\) on each step until reach the end token. The top-2 beam search of model prediction is illustrated as Fig. 2. At each step, only \(k = 2\) nodes are retained to continue the search. Finally, the path corresponding to “9 4 - 11 7” has the highest score (as shown in blue in the figure), and backtrack to obtain the expression with the highest probability.
TABLE 3. Exact match results claimed on four datasets.

| Model                  | A12 | CC  | IL  | MAWPS |
|------------------------|-----|-----|-----|--------|
| Hosseini, et al.[18]   | 77.7| NaN | NaN | NaN    |
| Kushman, et al.[19]    | 64.0| 73.7| 2.3 | NaN    |
| Roy, et al. [28]       | NaN | NaN | 52.7| NaN    |
| Robai, et al.[36]      | NaN | NaN | NaN | 62.8   |
| Wang, et al.[37]       | 78.5| 75.5| 73.3| NaN    |
| Griffith, et al.[25]   | 77.2| 94.4| 96.4| 84.7   |
| Our model              | 80.65| 97.92| 86.36| 83.65  |
| - pre-training         | 78.82| 96.13| 85.24| 82.4   |
| - pre-training, top-k beam search | 78.3 | 95.95 | 85.1 | 82.8 |

TABLE 4. Average accuracy with different encoding methods.

| Encoding Method       | A12  | CC  | IL  | MAWPS  |
|-----------------------|------|-----|-----|--------|
| SubwordTextEncoder    | 83.23%| 97.92%| 87.88%| 91.03% |
| TokenTextEncoder      | 84.52%| 97.92%| 90.53%| 91.16% |

III. RESULTS

We evaluate our methods on four open datasets mentioned previously. Two metrics including Exact Match(EM) and BLEU-2 are adopted to evaluate our model following Griffith et al. [25]. EM means that the generated equation is consistent with the standard answer in terms of both characters and order. BLEU-2 a method of automatic machine translation evaluation for 2-gram, and according to Papineni, K. et al. [33], is computed as:

\[
\sum_{C \in \text{answers}} \sum_{2-gram \in C} \frac{\text{Count}(2-gram)}{\text{Count}(2-gram')}
\]

where \(\text{answers}\) is the produced express, and \(\text{Count}_{\text{clip}}(2-gram) = \min(\text{count}, \max\text{AnswerCounts})\) which clips the total count of each produced 2-gram by its maximum true answer count.

A. BASELINES AND ABLATION STUDY

For comparison, we select several public models that claimed their results on the aforementioned datasets. Table 3 provides detailed results where the number are absolute accuracies, i.e., the arithmetic express generated is exactly match the correct numeric answer. We keep only the best result regardless of what configuration is taken. Our best result is better than state-of-the-art on two datasets, is close to or worse than the best score on MAWPS, and IL, respectively.

We perform an ablation study and report the results in Table 3. “- pre-training” means removing the processing of pre-training on target task data. “- pre-training, top-k beam search” means removing both pre-training and top-k beam search. Both pre-training and top-k beam search contribute to the final performance, and relatively, pre-training on target task data is more effective.

B. EXPERIMENTAL SETTING

We use Transformer architecture to produce character sequences that are arithmetic expressions. The most common Transformer network is used, instead of the latest variant. Our model uses 2 Transformer layers. The layers utilize 8 attention heads with a depth of 256 and a feed-forward depth of 1024. We use the Adam optimizer [38] with \(\beta_1 = 0.95, \beta_2 = 0.99, \epsilon = 1 \times e^{-9}\). The model respects a 10% dropout rate throughout the training. A schedule learning rate is used with warmup steps=4000. We employ a batch size of 128 for all training. The model is trained on the datasets for 300 epochs before testing. Questions are trimmed to 60 tokens during training and prediction. Equations are turned into pre-fix arithmetic expressions.

C. EFFECT OF TOKENTEXTENCODER

In this part, we investigate the effect of different text encoding methods on the MWPs test set. The results are shown in Table 4. Networks, with the same architecture and same hyperparameters, are trained and tested three times on every test set. No pre-training on any dataset is used. Formally, the result is computed as:

\[
\text{accuracy} = \frac{1}{M} \sum_{n=1}^{M} \frac{C}{P}
\]

where \(M\) is the number of test repetitions, which is 3; \(P\) is the number of MWPs in the dataset, and \(C\) is the number of correct expression translations.

As shown in Table 4, the model with TokenTextEncoder significantly improves the average accuracies on all datasets except one having the same accuracy.

However, we also found that the subword encoding method results in more quickly loss falling. The network converges to a stable level faster. Fig. 3 shows the loss comparison of two encoding methods on training data.

D. EFFECT OF PRE-TRAINING ON TARGET TASK DATA

Inspired by ULMFiT [34] and Griffith, et.al. [25], We examined the effect of pre-training on target task data on the improvement of translation quality. We re-implement the network of Griffith, et.al. [25], and fixes some BUGs in code. Our network performed better on three out of four
test datasets, as shown in Table 5. After pre-trained on the Dolphin18K dataset, the scores are increased further on three datasets except on MAWPS being close to the non-pre-trained network. To make this a fair comparison, we used the same SubwordTextEncoder method as Griffith, et.al. [25] in Table 5.

We used TokenTextEncoder to compare the translation effect again, and the results are shown in TABLE 6 below. On the CC dataset, the pre-trained model performs as well as the non-pre-trained model. However, the pre-training effect is seen in the three others.

### E. EFFECT OF TOP-K BEAM SEARCH

Generally, the top-k beam search algorithm is considered to perform well than a greedy search. To test this idea further, the performance of greedy search and top-k beam search are compared on the strongest performing model with TokenTextEncoder. From TABLE 7, we could observe that top-k search algorithm indeed improves the average accuracy a bit in the prediction stage, but, the effect seems to be limited, and at the expense of more testing time.

### F. CASE STUDY

In this part, we investigate further some cases which are marked as errors by the model. The question texts are normalized, and the arithmetic expressions are converted to prefix representation. These cases can be divided into four classes as follows:

1) Mis-prediction cases. This class accounted for 56% of the total error cases. The model predicted the wrong operator or produced numbers in the wrong order (or both). Table 8 shows 3 typical examples. We think that these errors could be fixed through adequate training on more data.

2) Insufficient capacity of the model. Table 9 shows two such examples. There is only one number 45 in the second question text explicitly. The model needs to infer another operand (number 3) through a count nose. We think that the errors occur because the model did not understand the real meaning in the question text. It is also the hardest issue of all to grapple with and may be helpful to make a semantic analysis.

3) Text manipulation mistakes. The first example in Table 10 is considered as an error because an additional
V. FUTURE WORK

Although subword encoding has become a mainstream method in the NLP research, we demonstrate that TokenTextEncoder offers equal and often better performance for MWPs. This finding is nontrivial because many languages in the world are logographic writing systems, instead of alphabetic writing systems like English. For example, Chinese characters contain rich syntactic and semantic information. TokenTextEncoder may better suit for Chinese text tokenization. We would like to try to solve Chinese MWPs automatically in our next work.

MWPs can be simple (the single 2 or 3 step expression), or more complex (requiring to solve 2 or more equations, like Dolphin18K and Math23K. As one of the future works, we plan to expand our methods to more complex MWPs.

Furthermore, MWPs is merely a special case of symbolic reasoning. How to solve more sophisticated symbolic reasoning automatically is also a valuable future direction.

REFERENCES

[1] D. Chen, J. Bolton, and C. D. Manning, “A thorough examination of the CNN/daily mail reading comprehension task,” Aug. 2016, arXiv:1606.02858. Accessed: Mar. 8, 2020.

[2] Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu, “Attention-over-attention neural networks for reading comprehension,” in Proc. 55th Ann. Meeting Assoc. Comput. Linguistics, vol. 1, Vancouver, BC, Canada, 2017, pp. 593–602. doi: 10.18653/v1/P17-1055.

[3] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, “Bidirectional attention flow for machine comprehension,” Jun. 2016, arXiv:1611.01603. Accessed: Mar. 8, 2020.

[4] E. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “SQuAD: 100,000+ questions for machine comprehension of text,” in Proc. Conf. Empirical Methods Natural Lang. Process., Austin, TX, USA, 2016, pp. 2383–2392. doi: 10.18653/v1/D16-1264.

[5] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, “RACE: Large-scale ReAding comprehension dataset from examinations,” in Proc. Conf. Empirical Methods Natural Lang. Process., Copenhagen, Denmark, 2017, pp. 785–794. doi: 10.18653/v1/D17-1082.

[6] D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner, “DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs,” Apr. 2019, arXiv:1903.00161. Accessed: Dec. 29, 2019.

[7] D. G. Bobrow, “Natural language input for a computer problem solving system,” MIT, Cambridge, MA, USA, 1964, vol. 36.

[8] E. Charniak, “Computer solution of calculus word problems,” in Proc. IJCAI, 1969, pp. 303–316.

[9] C. Liguda and T. Pfeiffer, “Mathing math word problems with augmented semantic networks,” in Proc. Int. Conf. Appl. Natural Lang. Inf. Syst., vol. 7337, 2012, pp. 247–252. doi: 10.1007/978-3-642-31178-9_29.

[10] S. Shi, Y. Wang, C.-Y. Lin, X. Liu, and Y. Rui, “Automatically solving number word problems by semantic parsing and reasoning,” in Proc. Conf. Empirical Methods Natural Lang. Process., Lisbon, Portugal, 2015, pp. 1132–1142. doi: 10.18653/v1/D15-1135.

[11] P. Liang, M. I. Jordan, and D. Klein, “Learning dependency-based compositional semantics,” Comput. Linguistics, vol. 39, no. 2, pp. 389–446, Jun. 2013.

[12] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, “QNNet: Combining local convolution with global self-attention for reading comprehension,” 2018, arXiv:1804.05641.

[13] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, “Bidirectional attention flow for machine comprehension,” 2016, arXiv:1611.01603.

[14] J. Li, L. Wang, J. Zhang, Y. Wang, B. T. Dai, and D. Zhang, “Modeling intra-relation in math word problems with different functional multi-head attentions,” in Proc. 57th Ann. Meeting Assoc. Comput. Linguistics, Florence, Italy, 2019, pp. 6162–6167. doi: 10.18653/v1/P19-1619.

[15] J. Andreass, M. Rohrbach, T. Darrell, and D. Klein, “Neural module networks,” Jul. 2017, arXiv:1711.02799. Accessed: Dec. 29, 2019.
A. Fan, K. Lin, D. Roth, S. Singh, and M. Gardner, “Neural module networks for reasoning over text,” Dec. 2019, arXiv:1912.04971. Accessed: Dec. 29, 2019.

W. Ling, D. Yogatama, C. Dyer, and P. Blunsom, “Program induction by rationalize generation: Learning to solve and explain algebra word problems,” Oct. 2017, arXiv:1705.04146. Accessed: Dec. 29, 2019.

M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. Kushman, “Learning to solve arithmetic word problems with verb categorization,” in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), 2014, pp. 523–533, doi: 10.3115/v1/D14-1058.

N. Kushman, Y. Artzi, L. Zettlemoyer, and R. Barzilay, “Learning to automatically solve algebra word problems,” in Proc. 52nd Annu. Meeting Assoc. Comput. Linguistics, vol. 1, Baltimore, MD, USA, 2014, pp. 271–281, doi: 10.3115/v1/P14-1026.

L. Zhou, S. Dai, and L. Chen, “Learn to solve algebra word problems using quadratic programming,” in Proc. Conf. Empirical Methods Natural Lang. Process., Lisbon, Sep. 2015, pp. 817–822, doi: 10.18653/v1/D15-1096.

S. Roy and D. Roth, “Solving general arithmetic word problems,” in Proc. Conf. Empirical Methods Natural Lang. Process., Lisbon, Portugal, Sep. 2015, pp. 1743–1752, doi: 10.18653/v1/D15-1202.

Y. Wang, X. Liu, and S. Shi, “Deep neural solver for math word problems,” in Proc. Conf. Empirical Methods Natural Lang. Process., Copenhagen, Denmark, 2017, pp. 845–854, doi: 10.18653/v1/D17-1088.

D. Huang, S. Shi, C.-Y. Lin, J. Yin, and W.-Y. Ma, “How well do computers solve math word problems? Large-scale dataset construction and evaluation,” in Proc. 54th Annu. Meeting Assoc. Comput. Linguistics, vol. 1, Berlin, Germany, 2016, pp. 887–896, doi: 10.18653/v1/P16-1084.

Z. Xie and S. Sun, “A goal-driven tree-structured neural model for math word problems,” in Proc. 28th Int. Joint Conf. Artif. Intell., Macao, China, Aug. 2019, pp. 5299–5305, doi: 10.24963/ijcai.2019/356.

K. Griffith and J. Kalita, “Solving arithmetic word problems automatically using transformer and unambiguous representations,” Dec. 2019, arXiv:1912.00871. Accessed: Jan. 26, 2020.

A. Vaswani et al., “Attention is all you need,” in Advances in Neural Inform. Processing Systems, vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

R. Koncel-Kedziorski, H. Hajishirzi, A. Sabharwal, O. Etzioni, and S. D. Ang, “Parsing algebraic word problems into equations,” Trans. Assoc. Comput. Linguistics, vol. 3, pp. 585–597, Dec. 2015.

S. Roy, T. Vieira, and D. Roth, “Reasoning about quantities in natural language,” Trans. Assoc. Comput. Linguistics, vol. 3, pp. 1–13, Dec. 2015, doi: 10.1162/tacl_a_00118.

K. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi, “MAWPS: A math word problem repository,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., San Diego, CA, USA, 2016, pp. 1152–1157, doi: 10.18653/v1/N16-1136.

L. Wang, Y. Wang, D. Cai, D. Zhang, and X. Liu, “Translating a math word problem to a expression tree,” in Proc. Conf. Empirical Methods Natural Lang. Process., Brussels, Belgium, Oct. 2018, pp. 1064–1069, doi: 10.18653/v1/D18-1132.

S. Pan and Q. Yang, “A survey on transfer learning,” IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, Oct. 2010, doi: 10.1109/TKDE.2009.191.

A. Maas, R. Daly, P. Pham, D. Huang, A. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in Proc. 49th Annu. Meeting Assoc. Comput. Linguistics, Hum. Lang. Technol., 2011, pp. 142–150.

K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, “BLEU: A method for automatic evaluation of machine translation,” in Proc. 40th Annu. Meeting Assoc. Comput. Linguistics, 2002, pp. 311–318, doi: 10.3115/1073083.1073135.

J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” in Proc. 56th Annu. Meeting Assoc. Comput. Linguistics, 2018, pp. 328–339, doi: 10.18653/v1/P18-103.

A. Fan, M. Lewis, and Y. Dauphin, “Hierarchical neural story generation,” May 2018, arXiv:1805.04835. Accessed: Mar. 22, 2020.

B. Rolda, R. Koncel-Kedziorski, and H. Hajishirzi, “Data-driven methods for solving algebra word problems,” Apr. 2018, arXiv:1804.10718. Accessed: Mar. 25, 2020.

L. Wang, D. Zhang, L. Gao, J. Song, L. Guo, and H. T. Shen, “MathDQN: Solving arithmetic word problems via deep reinforcement learning,” in Proc. 32nd AAAI Conf. Artif. Intell., 2018, p. 8.

D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. Int. Conf. Learn. Represent., 2014, pp. 1–15.

Q. Liu, W. Guan, S. Li, F. Cheng, D. Kawahara, and S. Kurohashi, “RODA: Reverse operation based data augmentation for solving math word problems,” IEEE/ACM Trans. Audio, Speech, Language Process., vol. 30, pp. 1–11, 2022, doi: 10.1109/TASLP.2021.3126932.

Z. Liang and X. Zhang, “Solving math word problems with teacher supervision,” in Proc. 13th Int. Joint Conf. Artif. Intell., Aug., 2021, pp. 3522–3528, doi: 10.24963/ijcai.2021/485.

K. Zaporozjets, G. Bekoulis, J. Deleu, T. Demester, and C. Develder, “Solving arithmetic word problems by scoring equations with recursive neural networks,” Expert Syst. Appl., vol. 174, Jul. 2021, Art. no. 114704, doi: 10.1016/j.eswa.2021.114704.

K. Aggarwal, M. M. Mijwil, A. H. Al-Mistarehi, S. Alomari, M. Gök, A. M. Z. Alaabdin, and S. H. Abdulrhman, “Has the future started? The current growth of artificial intelligence, machine learning, and deep learning,” Iraqi J. Comput. Sci. Math., vol. 3, no. 1, pp. 115–123, Jan. 2022.