# Solving Arithmetic Word Problems Automatically Using Transformer and Unambiguous Representations

**Kaden Griffith and Jugal Kalita**
University of Colorado Colorado Springs
kadengriffith@gmail.com and jkalita@uccs.edu

## Abstract

Constructing accurate and automatic solvers of math word problems has proven to be quite challenging. Attempts using machine learning have so far been trained on corpora specific to math word problems to produce arithmetic expressions in infix notation before answer computation. Neural networks have struggled to generalize, even when trained on large and diverse datasets. This paper outlines the use of Transformer networks trained to translate simple math word questions to equivalent mathematical expressions in infix, prefix, and postfix notations. We use a pretraining approach to translation, followed by training on corpora specific to word problems, and compare results produced by a large number of neural configurations. We find that the use of the prefix notation produces the best results, surpassing the state of the art.

## Introduction

Solving a math word problem (MWP) starts with one or more sentences describing a problem to be understood. These sentences are processed to produce an arithmetic expression, which is evaluated to provide an answer. Recent neural approaches to solving arithmetic word problems have used various flavors of recurrent neural networks (RNN) as well as reinforcement learning. Such methods have had difficulty achieving a high level of generalization. Often, they can extract the relevant numbers, but misplace them in the generated expressions. At other times, they have replaced the numbers in the problem statement with arbitrary other numbers when formulating corresponding math expressions. The infix notation used requires pairs of parenthesis to be balanced and also placed correctly, bracketing the right numbers. There have been problems with both of these requirements as well.

Figure 1: Misinterpretations of a MWP.

**Question:**
At the fair Adam bought 13 tickets. After riding the ferris wheel he had 4 tickets left. If each ticket cost 9 dollars, how much money did Adam spend riding the ferris wheel?
**Some possible expressions that can be produced:**
(13 - 4) * 9, 9 * 13 - 4, 5 * 13 - 4, 13 - 4 * 9,
13 - (4 * 9), (9 * 13 - 4)

Figure 1 gives examples of some infix representations that a machine learning word problem solver can produce from a simple word problem. Of the expressions shown, only the first one is correct. We start with a hypothesis that the sole use of the infix expression representation as a precursor to solving such problems attributes to some of these problems in automatic solvers. In modern application of math, we are used to using infix notation, but the formulation of expressions does not make this requirement.

We have also noticed that the actual numbers used in MWPs vary widely from problem to problem. Real numbers can take any conceivable value, making it almost impossible for a neural network to learn good representations for them. Thus, we hypothesize that replacing the numbers in the problem statement with generic tags like $\langle \mathrm{n1} \rangle$, $\langle \mathrm{n2} \rangle$, and $\langle \mathrm{n3} \rangle$ and saving their values as a pre-processing step, will not take away from the generality of the solution, but will suppress the problem of fertility in number generation, which leads to the introduction of numbers not present in the question sentences, in the relevant solution.

Another hypothesis that forms the basis of our approach is that a neural network which has been pre-trained on general language knowledge will be better able to “understand” the semantics of the problem to produce the correct arithmetic expressions with the tags we use.

We use the Transformer model [Vaswani et al., 2017] to solve arithmetic word problems as a particular case of machine translation from text to the language of mathematical expressions. Transformers in various configurations have become a staple of NLP in the past two years. Past neural approaches did not treat this problem as pure translation like we do but augmented the neural architectures with various external modules such as parse trees or used deep reinforcement learning. In this paper, we demonstrate that Transformers can be used to solve MWPs successfully. We show that our results outperform state-of-the-art results by [Wang et al., 2018, Hosseini et al., 2014, Kushman et al., 2014, Roy, Vieira, and Roth, 2015, Robaidek, Koncel-Kedziorski, and Hajishirzi, 2018], which we use for comparison.

We organized our paper as follows. The second section presents related work. Then we discuss our approach. We follow by an analysis of experimental results and how they compare to recent networks. We also discuss our successes and shortcomings in this section. Finally, we share our concluding thoughts and end with our direction to future work.

## Related Work

Many past strategies to solve math word problems have utilized rules and templates to match sentences to arithmetic expressions. Some such approaches seemed to solve problems impressively within a narrow domain, but performed poorly when out of domain, lacking generality that is necessary to solve common questions [Bobrow, 1964, Bakman, 2007, Liguda and Pfeiffer, 2012, Shi et al., 2015]. Kushman et al. (2014) used feature extraction and template-based categorization by representing equations as expression tree forests and finding a near match. Such methods required human intervention in the form of feature engineering and development of templates and rules, which is not desirable for expandability and adaptability. Hosseini et al. (2014) performed statistical similarity analysis to obtain acceptable results, but did not perform well with texts that were dissimilar to training examples.

Existing approaches have used various forms of auxiliary information. For example, Hosseini et al. (2014) used verb categorization to identify important mathematical cues and contexts. Mitra and Baral (2016) used predefined formulas to assist in matching. Koncel-Kedziorski et al. (2015) parsed the input sentences, enumerated all parses, and learned to match, requiring expensive computations. Roy and Roth (2017) performed searches for semantic trees over significant computational spaces.

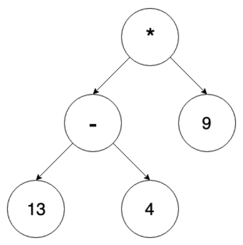
Some recent approaches have transitioned to using artificial neural networks. Semantic parsing of MWPs is a recently developed strategy which takes advantage of RNN architectures to parse math word problems directly into equations or expressions in a uniquely developed math-specific language [Shi et al., 2015, Sun et al., 2019]. Systems using RNNs have shown promising results, but they have had difficulties correctly learning balanced parenthesis, and also, sometimes incorrectly choose numbers when generating equations. Most recently, Sun et al. (2019) used a Bi-Directional LSTM architecture for math word problems. Huang et al. (2018) used a deep reinforcement learning model to achieve character placement in both seen and novel equation templates. Wang et al. (2018) also used deep reinforcement learning to solve MWPs more comprehensively than RNNs.

## Approach

We view MWP solving as a sequence-to-sequence translation problem. Systems using RNNs have excelled in sequence-to-sequence problems such as text translation and question answering. The recent introduction of attention mechanisms has improved the performance of RNN models. In particular, Vaswani et al. (2017) introduced the Transformer network, which uses stacks of attention layers instead of recurrence. The use of Transformers in various configurations has shown vastly improved results, achieving state-of-the-art performance in many natural language processing tasks.

Figure 2: Representations of a MWP.

**Question:**
At the fair Adam bought 13 tickets. After riding the ferris wheel he had 4 tickets left. If each ticket cost 9 dollars, how much money did Adam spend riding the ferris wheel?
**Expression Tree:**



**Output:**
Prefix: - 13 4 * 9
Postfix: 13 4 - 9 *
Infix: (13 - 4) * 9
**Computed Answer:**
81 dollars

We use several configurations of Transformer networks to learn the prefix, postfix, and infix notations of each MWP equation independently. Prefix and postfix representations of equations do not contain parentheses, which has been a source of unnecessary confusion in some approaches. Figure 2 shows the three notations for a problem. We expect that the use of suitable training data, in particular, if the learned target sequences are less likely to mislead or throw off a neural network, may help the learning of the model to be more robust.

We train on standard datasets, which are readily available and commonly used for our task. Our method considers the translation of English text to simple algebraic expressions, although the general approach can be used for other tasks as well. After performing experiments by training directly on math word problem corpora, we perform a different set of experiments by pretraining our network on general language corpora. The success of pre-trained models such as ELMo [Peters et al., 2018], GPT-2 [Budzianowski and Vulić, 2019], and BERT [Devlin et al., 2018] for all manners of natural language tasks, may encourage better learning of our system. However, in our case, the output is not natural language, but algebraic expressions, and that is why we were apprehensive at first. Our success in such heterogeneous translations opens up other possible usages of such pre-trained networks.

### Data

Four individual datasets are combined to create a super-collection of MWPs that we call MWP-Data. The datasets contain addition, subtraction, multiplication, and division word problems. Questions in MWP-Data pair with equivalent math equations.

1. **AI2** [Hosseini et al., 2014]. AI2 is a collection of 395 problems. Each problem contains multiple numeric values, where some may not be relevant for answering the question. These questions are addition and subtraction problems.
2. **CC** [Roy and Roth, 2016]. The Common Core dataset contains 2-step algebra word questions. The Cognitive Computation Group gathered these questions. We use a total of 600 unique MWPs from this set in training. This included all of the set's examples.
3. **IL** [Roy, Vieira, and Roth, 2015]. The Illinois dataset contains 1-step algebra word questions. The Cognitive Computation Group also compiled these questions. All of 562 MWPs were used in training from this set.
4. **MAWPS** [Koncel-Kedziorski et al., 2016]. MAWPS is a relatively large collection of MWPs which are primarily from other MWP datasets. We use 2,373 of 3,915 MWPs from this set. The problems not used from this set were more complex problems requiring systems of equations to derive a final solution.

We take a randomly sampled $95\%$ of MWP-Data for training. Networks test on the $5\%$ of withheld examples from each MWP dataset. Training and testing are repeated three times, and reported results are an average between the three outcomes.

## Representation Conversion

We take a simple approach when converting infix expressions found in MWP-Data to the other two representations. We strip out the equals sign and variable representing the solved value in each equation. Then two stacks are filled; one with operators found in the equation and the other with the operands. From these stacks, we form a binary tree structure. A given expression tree resembles the one depicted in Figure 2. Traversing an expression tree in pre-order results in a prefix conversion. Post-order traversal gives us a postfix expression. We pre-process the original infix expression as well, removing the variable, and equals sign. Three versions of our training and testing data are used to account for each conversion. By training on different representations, we expect our test results to change.

## Pretraining

We pre-train half of our networks to endow them with a good foundational knowledge of English. Pretraining models on significant-sized language corpora has been a common approach recently. We explore the pretraining approach using a general English corpus because our MWPs resemble the English language. For this task, we use the IMDb Movie Reviews dataset [Maas et al., 2011]. This set contains 314,041 unique sentences. Since movie reviewers wrote this data, it is a reference to natural language not necessarily related to arithmetic. Although we pre-train, we do so using a relatively small and specialized corpus that is readily available, to investigate the efficacy of pretraining for math word problems. Training on a much bigger and general corpus like Wikipedia may make the language model stronger, but we leave this for future work.

We compare pre-trained models to non-pre-trained models to observe performance differences. Our pre-trained models are trained in an unsupervised fashion to improve the encodings of our fine-tuned solvers.

## Method: Training and Testing

The input sequence is a natural language specification of an arithmetic word problem. The MWP questions and equations have been encoded using the subword text encoder provided by the TensorFlow library. The ideal output will be an expression in prefix, infix, or postfix notation, which then can be manipulated further and solved to obtain a final answer.

Many of the examples in MWP-Data contain unique numbers. Rare terms are adverse for generalization since many of these numbers occur only once in the dataset. The networks may have difficulty dealing with the presence of unique values. As a precaution to this issue, our networks do not consider any relevant numbers during training. Before the networks attempt any translation, we pre-process each question and expression in MWP-Data by a number mapping algorithm. This algorithm replaces each numeric value with a corresponding identifier (e.g., $\langle n1 \rangle$, $\langle n2 \rangle$, etc.), and remembers the necessary mapping. We find that this approach significantly improves how networks interpret each question. When translating, the numbers in the original question are tagged and cached. From the encoded English and tags, a predicted sequence resembling an expression presents itself as output. Since each network's learned output resembles something like $\langle n1 \rangle + \langle n2 \rangle * \langle n3 \rangle$, we use the cached tag mapping to replace the tags with the corresponding numbers and return a final mathematical expression.

Three representation models are trained and tested separately: Prefix-Transformer, Postfix-Transformer, and Infix-Transformer. For each experiment, we use representation specific Transformer architectures. Each model uses the Adam optimizer with $beta_1 = 0.95$ and $beta_2 = 0.99$ with a standard epsilon of $1e^{-9}$. The learning rate is reduced automatically in each training session as the loss decreases. Throughout the training, each model respects a 10% dropout rate. We employ a batch size of 128 for all training. The networks are trained on a machine using 1 Nvidia 1080 Ti graphics processing unit (GPU).

We compare medium-sized, small, and minimal networks to show if network size can be reduced to increase training and testing efficiency while retaining high accuracy. Networks over six layers have shown to be non-effective for this task. We later discuss the issues that arose when attempting such configurations. We tried many configurations of our network models, but report results with only three configurations of Transformers.

- **Transformer Type 1:** This network is a small to medium sized network consisting of 4 Transformer layers. Each layer utilizes 8 attention heads with a depth of 512 and a feed-forward depth of 1024.
- **Transformer Type 2:** The second model is small in size,

using 2 Transformer layers. The layers utilize 8 attention heads with a depth of 256 and a feed-forward depth of 1024.

- **Transformer Type 3:** The third type of model is minimal, using only 1 Transformer layer. This network utilizes 8 attention heads with a depth of 256 and a feed-forward depth of 512.

**Objective Function** We calculate the loss in training according to the mean categorical cross-entropy formula. Evaluation between the possible translation classes (all vocabulary subword tokens) and the produced class (predicted token) is the metric of performance here. During each evaluation, target terms are masked, predicted, and then compared to the masked (known) value. This process is applied to every determined subword in a translation and then the model loss is adjusted according to the mean of the translation accuracy.

$$loss = \sum_{i=1}^{I} \frac{1}{J} \sum_{j=1}^{J} \left( - \sum_{k=1}^{K} target_{j,k} * log\big(p(j \in k)\big) \right) \quad (1)$$

where $K = |Translation\ Classes|$, $J = |Translation|$, and $I$ is the number of examples.

**Experiment 1: Representation** Many of the problems encountered by prior approaches seem to be encouraged by the use of infix notation. In this experiment, we compare translation BLEU-2 scores. Traditionally, a BLEU score is a metric of translation quality. Our presented BLEU scores represent an average of scores a given model received over each of the target test sets. We use a standard bi-gram weight to show how accurate translations are within a window of two adjacent terms. After testing translations, we calculate an average BLEU-2 per test set, which is related to the success over that data. The scores for each dataset are then averaged for the presented value.

$$model_{avg} = \frac{1}{N} \sum_{n=1}^{N} BLEUavg_n \quad (2)$$

where $N$ is the number of test datasets, which is 4.

**Experiment 2: State-of-the-art** This experiment compares our networks to recent previous work. We count a given test score by a simple "correct versus incorrect" method. The answer to an expression directly ties to all of the translation terms being correct, which is why we do not consider partial precision. We compare average accuracies over 3 test trials on different randomly sampled test sets from each MWP dataset. This calculation more accurately depicts the generalization of our networks.

**Effect of Pretraining** We also explore the effect of language pretraining. Half of the models are pre-trained on unlabelled English data. This training occurs over 30 iterations to install an advanced level of language understanding before training on MWP-Data. The same Transformer architectures are also trained solely on MWP-Data. Each model is trained on MWP-Data for 300 iterations before testing.

$$model_{avg} = \frac{1}{R} \sum_{r=1}^{R} \left( \frac{1}{N} \sum_{n=1}^{N} \frac{C \in n}{P \in n} \right) \quad (3)$$

where $R$ is the number of test repetitions, which is 3; $N$ is the number of test datasets, which is 4; $P$ is the number of MWPs, and $C$ is the number of correct equation translations.

## Results

We now present the results of our various experiments. We compare the three representations of target equations and three architectures of the Transformer model in each test. Not all past work has used the data we use here, but we attempt to communicate our capabilities compared to past neural approaches to the best of our ability.

Table 1: BLEU-2 comparison for Experiment 1.

| **(Type) Model** | **Average** |
|---|---|
| *Pre-trained* | |
| (1) Prefix-Transformer | 94.03 |
| (1) Postfix-Transformer | 92.71 |
| (1) Infix-Transformer | 93.59 |
| (2) Prefix-Transformer | 93.51 |
| (2) Postfix-Transformer | 92.92 |
| (2) Infix-Transformer | 93.34 |
| (3) Prefix-Transformer | 93.39 |
| (3) Postfix-Transformer | 92.84 |
| (3) Infix-Transformer | 93.14 |
| *Non-pre-trained* | |
| (1) Prefix-Transformer | 94.95 |
| (1) Postfix-Transformer | 87.55 |
| (1) Infix-Transformer | 93.56 |
| (2) Prefix-Transformer | **95.57** |
| (2) Postfix-Transformer | 94.01 |
| (2) Infix-Transformer | 93.39 |
| (3) Prefix-Transformer | 95.13 |
| (3) Postfix-Transformer | 94.03 |
| (3) Infix-Transformer | 93.36 |

Results of Experiment 1 are given in Table 1. By using BLEU scores, we assess the translation capability of each network. This test displays how networks understand different math representations to a character summary level. We compare by average BLEU-2 accuracy among our tests in the *Average* column of Table 1 to communicate these translation differences.

From our results, prefix representation of our target language performs better than the generally used infix notation. The best performing network was a non-pre-trained type 2 prefix Transformer arrangement. Infix notation was better understood than postfix in the smaller models.

The reasoning behind why similar representations provide different results is somewhat unclear. One hypothesis to explain this is related to the way attention in the systems is accumulated. A network is more likely to produce a successful translation if the operators, determined by language, are close to one another. Numbers and operators appear partially grouped in both prefix and postfix representations, which could be the cause of observable enhanced learning.

Table 2 provides detailed results of Experiment 2. Results by [Wang et al., 2018, Hosseini et al., 2014, Roy, Vieira, and Roth, 2015, Robaidek, Koncel-Kedziorski, and Hajishirzi, 2018] are sparse but indicate the scale of success in past approaches. Prefix, postfix, and infix representations in Table 2 shows that network capabilities are changed by how teachable the target data is.

While our networks fell short of Wang et al., 2018 AI2 testing accuracy, we present state-of-the-art results for the remaining three datasets. The type 2 prefix Transformer received the highest testing average of 86.73% accurate. Comparatively, the Transformer network performs well and requires little computation time.

Our attempt of language pre-training fell short of expectations. It is odd to see that more examples of English does not improve the understanding of MWPs. Use of advanced embedding techniques tried by Robaidek, Koncel-Kedziorski, and Hajishirzi, 2018 also seem to limit MWP coherence. In future attempts, using a more general corpora of language could help grow semantic ability.

## Analysis

All of the network configurations used were very successful for our task. To display the capability of our most successful model (type 2 prefix Transformer), we present some outputs of the network in Figure 3.

Even when incorrect, the syntax of math expressions was strongly held. For the majority of questions, our translators were able to determine operators based solely on the context of language.

Larger networks in tandem with pre-training have been shown, in many cases, to relate semantics of natural language better. Our pre-training was unsuccessful in improving accuracy, even when applied to networks larger than those reported. We may need to use more general language, or pre-train on very math specific texts to be successful. Our results support our thesis of infix limitation.

**Error Analysis** Our system, while performing above standard, could still apply some improvements. One issue originates from the algorithmic pre-processing of our questions and expressions. In Figure 4 we show an example of one such issue. The excerpt comes from a type 3 non-pre-trained Transformer test. The example shows that identifier $\langle n1 \rangle$ was seemingly overlooked after translation. The issue is attributed to the identifier algorithm only considering numbers in the problem. Observe in the question that the word "eight" is the number we expect to relate to $\langle n2 \rangle$. Our identifying algorithm could be improved by considering such number words and performing conversion to a numerical value. If our algorithm performed as expected, the identifier $\langle n1 \rangle$ would be related with 4 (the first occurring number in the question) and $\langle n2 \rangle$ with 8 (the converted number word appearing second in the question).

Figure 3: Successful prefix translations.

**AI2**
A spaceship traveled 0.5 light-year from earth to planet x and 0.1 light-year from planet x to planet y. Then it traveled 0.1 light-year from planet y back to Earth. How many light-years did the spaceship travel in all?
*Translation Produced:*
+ + 0.5 0.1 0.1

**CC**
There were 16 friends playing a video game online when 7 players quit. If each player left had 8 lives, how many lives did they have total?
*Translation Produced:*
* 8 - 16 7

**IL**
Lisa flew 256 miles at 32 miles per hour. How long did Lisa fly?
*Translation Produced:*
/ 256 32

**MAWPS**
Debby's class is going on a field trip to the zoo. If each van can hold 4 people and there are 2 students and 6 adults going, how many vans will they need?
*Translation Produced:*
/ + 2 6 4

Figure 4: Number replacement errors.

**Question (MAWPS)**
Melanie is selling 4 gumballs for eight cents each. How much money can Melanie get from selling the gumballs?
**Correct Translation (Infix)**
4 * 8
**Hypothesized Translation**
4 + $< n1 >$

The overall translation was incorrect whether or not our algorithm was successful, but it is essential to analyze problems like these that result in future improvements. Had all questions been tagged correctly, our performance would have likely improved.

One approach not tried here is the use of alphabetical assignment for numerics. Instead of $\langle n1 \rangle$, $\langle n2 \rangle$, we could easily use $\langle a \rangle$, $\langle b \rangle$; which, reduces the chance of prediction error by 25%. In addition, better and more accurate replacement techniques exist which could further improve our work, but for our initiative, we wish to focus less on algorithm improvement and more on neural performance. In an ideal application of a MWP solver, no pre-processing would be necessary, but our approach significantly changed the focus of the system from rare occuring numbers to important language cues found in problem text.

Table 2: Test results for Experiment 2 (* denotes averages on present values only).

| (Type) Model | AI2 | CC | IL | MAWPS | Average |
|---|---|---|---|---|---|
| Hosseini et al., 2014 | 77.7 | – | – | – | *77.7 |
| Kushman et al., 2014 | 64.0 | 73.7 | 2.3 | – | *46.7 |
| Roy, Vieira, and Roth, 2015 | – | – | 52.7 | – | *52.7 |
| Robaidek et al. 2018 | – | – | – | 62.8 | *62.8 |
| Wang et al., 2018 (MathDQN) | **78.5** | 75.5 | 73.3 | – | *75.4 |
| *Pre-trained* | | | | | |
| (1) Prefix-Transformer | 70.2 | 91.1 | 95.2 | 82.4 | 84.7 |
| (1) Postfix-Transformer | 66.7 | 90.0 | 92.9 | 82.7 | 83.1 |
| (1) Infix-Transformer | 70.2 | 93.3 | **96.4** | 82.4 | 85.6 |
| (2) Prefix-Transformer | 66.7 | 91.1 | **96.4** | 82.1 | 84.1 |
| (2) Postfix-Transformer | 68.4 | 93.3 | 94.1 | 82.4 | 84.6 |
| (2) Infix-Transformer | 70.2 | 94.4 | 94.1 | 84.4 | 85.8 |
| (3) Prefix-Transformer | 68.4 | 91.1 | 95.2 | 82.4 | 84.3 |
| (3) Postfix-Transformer | 66.7 | 92.2 | 94.1 | 82.1 | 83.8 |
| (3) Infix-Transformer | 68.4 | 93.3 | 95.2 | 84.1 | 85.2 |
| *Non-pre-trained* | | | | | |
| (1) Prefix-Transformer | 71.9 | 94.4 | 95.2 | 83.4 | 86.3 |
| (1) Postfix-Transformer | 63.2 | 81.1 | 92.9 | 75.7 | 78.2 |
| (1) Infix-Transformer | 68.4 | **97.8** | 92.9 | 83.4 | 85.6 |
| (2) Prefix-Transformer | 73.7 | 94.4 | 94.1 | **84.7** | **86.7** |
| (2) Postfix-Transformer | 68.4 | 94.4 | 94.1 | 83.1 | 85.0 |
| (2) Infix-Transformer | 68.4 | 95.6 | 94.1 | 83.1 | 85.3 |
| (3) Prefix-Transformer | 73.7 | 93.3 | 95.2 | 84.1 | 86.6 |
| (3) Postfix-Transformer | 68.4 | 94.4 | 94.1 | 82.4 | 84.8 |
| (3) Infix-Transformer | 66.7 | 95.6 | 94.1 | 81.7 | 84.5 |

Figure 5: Failure of large Transformer.

**Question (IL)**
There are 4 cards. 3 cards more are added. How many are there total?
**Correct Translation (Infix)**
4 + 3
**Hypothesized Translation**
⟨⟨⟨⟨⟨ ... ⟨⟨⟨⟨⟨

Others have had great success with large Transformer architectures, especially when tasked with translation. We observe problems that result in an unusable arrangement when attempting networks of large sizes. Figure 5 demonstrates issues of attempted large networks. This occurance was common among pre-trained and non-pre-trained models of various dimensions above 6 Transformer layers.

## Conclusions and Future Work

We show that alternative math representations provide more stability in automatic solvers. Use of Transformer networks improves automatic math word problem solving. We make improvements through very accessible means with thoughtful data and training of general natural language. Automatically solving math word problems will undoubtedly be very useful for entities such as question answering services. Transformers with unambiguous intermediate representations produce state-of-the-art arithmetic word problem translations.

Datasets such as Dolphin18k Huang et al., 2016, consisting of web-answered questions from Yahoo! Answers, require more variety of questions to be understood by the system. We hope to expand to step-based calculations, more complex concepts such as probabilities or calculus, and push the limits of machine understanding of classical mathematics.

Extensive pre-training over a large corpora of language has extended the capabilities of many neural approaches. Networks like BERT Devlin et al., 2018, trained extensively on data from Wikipedia, perform relatively better in many contexts. Alternatively, creating intentional bias toward words which determine a specific operation (i.e., +, *, -, /) could increase the connection between natural language and mathematics.

With a hope to further advance this area of research and heighten interests, all of the code and data used is available on GitHub.

## Acknowledgement

The National Science Foundation supports the work reported in this paper under Grant No. 1659788. Any opinions, findings any conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## References

Bakman, Y. 2007. Robust understanding of word problems with extraneous information. *arXiv preprint math/0701393*.

Bobrow, D. G. 1964. *Natural language input for a computer problem solving system*. Ph.D. Dissertation, Massachusetts Institute Of Technology.

Budzianowski, P., and Vulić, I. 2019. Hello, it's gpt-2–how can i help you? towards the use of pretrained language models for task-oriented dialogue systems. *arXiv preprint arXiv:1907.05774*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Hosseini, M. J.; Hajishirzi, H.; Etzioni, O.; and Kushman, N. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 523–533.

Huang, D.; Shi, S.; Lin, C.-Y.; Yin, J.; and Ma, W.-Y. 2016. How well do computers solve math word problems? large-scale dataset construction and evaluation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 887–896.

Huang, D.; Liu, J.; Lin, C.-Y.; and Yin, J. 2018. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, 213–223. Santa Fe, New Mexico, USA: Association for Computational Linguistics.

Koncel-Kedziorski, R.; Hajishirzi, H.; Sabharwal, A.; Etzioni, O.; and Ang, S. D. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics* 3:585–597.

Koncel-Kedziorski, R.; Roy, S.; Amini, A.; Kushman, N.; and Hajishirzi, H. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1152–1157.

Kushman, N.; Artzi, Y.; Zettlemoyer, L.; and Barzilay, R. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 271–281.

Liguda, C., and Pfeiffer, T. 2012. Modeling math word problems with augmented semantic networks. In *International Conference on Application of Natural Language to Information Systems*, 247–252. Springer.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 142–150. Portland, Oregon, USA: Association for Computational Linguistics.

Mitra, A., and Baral, C. 2016. Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 2144–2153.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Robaidek, B.; Koncel-Kedziorski, R.; and Hajishirzi, H. 2018. Data-driven methods for solving algebra word problems. *arXiv preprint arXiv:1804.10718*.

Roy, S., and Roth, D. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Roy, S., and Roth, D. 2017. Unit dependency graph and its application to arithmetic word problem solving. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Roy, S.; Vieira, T.; and Roth, D. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics* 3:1–13.

Shi, S.; Wang, Y.; Lin, C.-Y.; Liu, X.; and Rui, Y. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1132–1142.

Sun, R.; Zhao, Y.; Zhang, Q.; Ding, K.; Wang, S.; and Wei, C. 2019. A neural semantic parser for math problems incorporating multi-sentence information. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)* 18(4):37.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Wang, L.; Zhang, D.; Gao, L.; Song, J.; Guo, L.; and Shen, H. T. 2018. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.