MathBot – A Deep Learning based Elementary School Math Word Problem Solver

Anish Kumar Nayak * **Rajeev Patwari** * **Viswanathan Subramanian** *

{anishkn, rpatwari, vsub} @stanford.edu

Abstract

We built a Deep Neural Network architecture based framework to make the *MathBot* learn to convert English language based math word problems into equations involving few unknowns and arithmetic quantities, and solve the equations thus generated. There have been many semantic parser and rule based math word problem solvers, but application of any learning algorithm to reduce natural language based math problems into equations is a topic of recent research. In this work, We show that the use of deep learning based natural language processing techniques, such as, Recurrent Neural Networks and Transformers, can help build such a learning system. Our work primarily focused on the use of transformers to predict the equation. We also added an equation solver to get the final result from the equation. In addition to traditional BLEU score we used an ingenious solution accuracy metric to evaluate our models. To improve solution accuracy, we introduced number mapping for word embedding as a novel technique to achieve high accuracy. With our transformer architecture and solver, we achieved a BLEU score of 33.4% and solution accuracy of 62.07% on our biggest dataset.

1. Introduction

Similar to the way we teach our elementary school kids to learn to solve natural language based math word problems, we should be able to train a machine learning system to solve same problems. These elementary school word problems involve one or more algebraic equations comprising of any combination of four arithmetic operations, namely addition, subtraction, multiplication and division. The goal of the machine learning system would then be to understand the arithmetic operation from the language context, identify the number of unknowns, extract the arithmetic quantities from the problem statement and construct the equation. Our *MathBot* first predicts the equation from the word problem using deep learning based natural language processing techniques, such as Recurrent Neural Networks (RNN) and Transformers (Vaswani et al., 2017), and then solves the predicted equation to get the final answer.

Table 1. Math Word Problem Example

MATH WORD PROBLEM
Benny found 696 seashells and 109 starfish on the beach. He gave 248 of the seashells to Sally. How many seashells does Benny now have ?
OUTPUT EQUATION: $x = 696 + 109 - 248$ SOLVER OUTPUT: 557

Our *MathBot* accepts a text-based math word problem and outputs the solution for the word problem. An example of math word problem that our *MathBot* solves is illustrated in Table 1. The output equations are intermediate representations that are predicted by the deep learning system. Our equation solver then computes the final solution.

We began by doing an extensive survey of the existing literature and related work done in (Zhang et al., 2019), which gave an extensive overview of several work which has been done in the field of math word problem solvers.

As a next step, our work involved procuring math word problem datasets and cleaning them up, eventually carrying out necessary pre-processing on cleaned up datasets. Next we worked on establishing baseline neural models by using previous work in this area (Sizhu Cheng, 2019) and making it work on datasets of our interest. We were able to reproduce results published in the prior work, and present it here as our baseline results.

Earlier works used BLEU score (Papineni et al., 2002) for evaluation models. For math word problems, ordering of predicted words is not so important. For example, $x = 4{+}5$ and $5 + 4 = x$ are equivalent mathematically, but BLEU score for this prediction would be very low. Similarly, if the predicted equation is $x = 4{-}5$ instead of $x = 4{+}5$, BLEU score will be high, but mathematically it would yield a very wrong answer. So, to evaluate our models, we implemented an equation solver to compute the final result and compared

*Equal contribution

it with the expected result.

We explored several transformer architectures to improve our model accuracy. For transformers, we varied the number of layers, embedding size, hidden size and attention heads. We ran hyperparameter tuning experimenets to find the right size transformer for our use case, and also tuned learning rate, dropout and batch size hyperparameters.

We got very good results with our elementary dataset. However, when we ran our best models on bigger datasets the solution accuracy was low. When we did error analysis, we found out for larger datasets the model had difficulty understanding new numbers it hadn't seen before. In order to overcome this limitation, we came up with number mapping for word embedding technique. This novel technique allowed us to achieve higher solution accuracy. Figure 1 shows the modified math word problem. We first extract the numbers and replace them with n# numbers. We provide the number mapping to our solver to solve the equation.

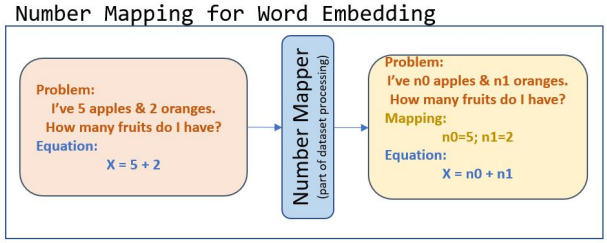


Figure 1. Math Word Problem Example with Number Mapping for Word Embedding

2. Related Work

The inspiration for creating *MathBot* came from observing parents of elementary school kids spending significant time every day to check their kids' homework and tests. We thought that *MathBot* will be of immense help to the parents, if it can scan the kids' word problem and output an equation and final solution.

Further, as mentioned by (Zhang et al., 2019), designing an automatic solver for mathematical word problems has a long history dating back to the 1960s (Bobrow, 1964), (Feigenbaum et al., 1963), (Charniak, 1969). The problem is particularly challenging because there remains a wide semantic gap to parse the human-readable words into machine-understandable logic so as to facilitate quantitative reasoning. Hence, math word problem solvers are broadly considered as good test beds to evaluate the intelligence level of agents in terms of natural language understanding (Clark, 2015), (Clark & Etzioni, 2016) and the successful solving of math word problem solvers would constitute a milestone towards general artificial intelligence.

The work done by (Wang et al., 2017), and recently by (Sizhu Cheng, 2019) is of particular interest to our approach to using deep neural networks in solving math word problems. We delved deeper on work done by (Sizhu Cheng, 2019), using it as guidance for baseline for our project.

3. Dataset

There are a few datasets available for the algebraic problem solving that have been used for both, rule-based solvers as well as Machine Learning based solvers. that were used in the published literature. They are listed in the table (**?**).

Dolphin18k and MAWPS datasets comprise of both elementary and complex algebraic problems while Alg514, Draw-1k is an elementary problem dataset in its entirety.

3.1. Pre-processing

This Dolphin18k dataset was built using the scripts and tools provided by the authors. The scripts collect the data from Yahoo Answers. Data clean up was performed as per the authors's instructions. The train and dev test splits are also provided by the scripts. The cleaned up dataset is available in a json format. Alg514, Draw-1k and MaWPS are readily available as json files and no webscraping is necessary to build them.

Our models, both Bi-LSTIM-Attn and Transformer both require input source file and target file. So, for each example entry, "text" section from json files is written to .txt and "equation" section is written to .txt. Further more, each entry in source and target is converted to lower case to avoid emphasis on uppercase letters in the dataset.

A subset of examples were extracted from all the datasets, that contain a single unknown. This was achieved by writing a script that parses the equations entry of the json file, search for number of unknowns and equations, extract only the examples that contain a single unknown.

3.2. Train-Dev Splits

The final goal of our Deep Learning model is to understand and solve elementary math word problems. In order to achieve this, the train/dev set distributions have to be clearly distinguished. The train set can comprise of elementary as well as complex problems but the dev set can only contain elementary problems. The final combinations of datasets that we used for our Deep Learning based elementary math word problem solver described in the table below.

Table 2. Dataset Sources

DATASET	COUNT	DESCRIPTION	REFERENCE
DOLPHIN-18K	18,460	Dolphin18K is a dataset created for math word problem solving, containing 18,460 problems posted by users on the community question answering site, Yahoo! Answers.	(Huang et al., 2016)
ALG514	514	514 elementary algebraic problems used in referred literature	(Alg)
MAWPS	3,065	MAWPS dataset, also used in (Sizhu Cheng, 2019)	(Kushman et al., 2014)
DRAW-1K	1,000	Diverse Algebra Word (DRAW-1K), consists of 1000 word problems crawled from algebra.com	(Kushman et al., 2014)

Table 3. Math Word Problem Example with Number Mapping for Word Embedding

MODIFIED MATH WORD PROBLEM
Benny found 696 seashells and 109 starfish on the beach. He gave 248 of the seashells to Sally. How many seashells does Benny now have ?
NUMBER MAPPING: $n0 = 696; n1 = 109; n2 = 248$
OUTPUT EQUATION: $x = n0 + n1 - n2$

MathBot Datasets	Train	Dev
MaWPS-Full	2965	100
MaWPS-Elem	1811	100
Ext-Elem	2107	250
Ext-Elem-Mapped	2107	250
Combined	9568	1000
Combined-Mapped	9568	1000

4. Number Mapping for Word Embedding

After exploring performance in both Bi-LSTM-Attn 5.1.1 and Transformer 5.2 models and performing error analysis, we discovered that the model would emphasize on the constant values in the problem statement. For example, consider the following scenario: "Benny found 696 seashells and 109 starfish on the beach. He gave 248 of the seashells to Sally. How many seashells does Benny now have ?" In this example, the numbers 696, 109 and and 248 become a part of vocabulary and the model cannot generalize well enough. After some initial thoughts we came up with an idea to map the constants to variables in order to help the model generalize the problem better, without emphasizing on any particular constants. We then did a literature survey and came across (Wang et al., 2018), which reinforced our idea.

To achieve number mapping, we wrote a parser in Python that would parse equation of every example, convert constants into variables, generate a translation file and replace the constants in input text to mapped variable. The script replaces first constant with $n0$, second with $n1$ and so on. Table 3 shows the intermediate mapping results. The mapping information is provided as input to the equation solver.

With these changes, the model learns to generalize the target better as numbers now maps to a smaller set of quantities, i.e. instead of number representations, to mapped $n\#$ representations (# = 0,1,2..).

5. Methods

5.1. Baseline Learning Models

For establishing our baseline model, we reproduced the results presented in (Sizhu Cheng, 2019) for **Neural Machine Translation with RNNs**

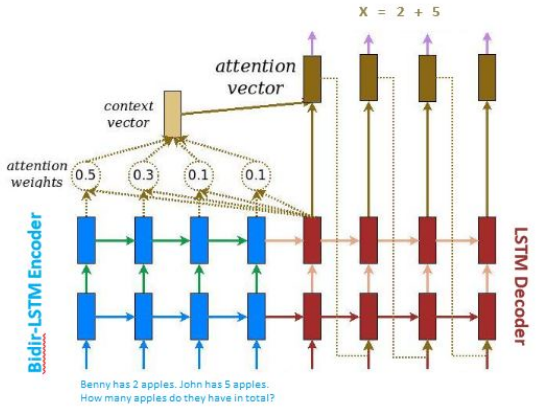


Figure 2. Baseline architecture

5.1.1. NEURAL MACHINE TRANSLATION WITH RNN

Figure 2 shows the baseline RNN architecture. Baseline RNN model model uses a Bidirectional LSTM Encoder and a Unidirectional LSTM Decoder. This Seq2Seq Model has multiplicative attention, as shown on the third step of the decoder. The baseline RNN produces a probability distribution P_t over target words at the t^{th} timestep. Here, V_t is the size of the target vocabulary. Loss function is shown below as well. Here, θ represents all the parameters of the model and $J_t(\theta)$ is the loss on step t of the decoder. g_t is the 1-hot vector of the target word at timestep t.

$$P_t = Softmax(W_{vocab} o_t), \text{ where } P_t \in^{V_t \times 1}, W_{vocab} \in^{V_t \times h}$$

$$J(\theta) = CE(P_t, g_t)$$

5.1.2. BASELINE SETUP

To get the baseline RNN model to run, we setup a python conda environment to run neural machine translation code that came with (cs2). We modified the code base to suit our needs. In our case, the math question dataset was the source language and the equations were the target language. We had to change how the equations were preprocessed, as the

Table 4. Baseline BLEU scores (* *Embed Size, Hidden Size, Dropout*)

HYPERPARAMETERS*	MAWPS-FULL	MAWPS-ELEM
32, 128, 0.5	34.25	27.38
32, 512, 0.5	86.86	89.81
32, 512, 0.3	88.38	88.92
32, 1024, 0.3	88.89	88.88
256, 256, 0.3	91.93	44.23

equations had various symbols for the arithmetic operators and parentheses. We established the baseline with the MAWPS dataset. Table 5.1.2 shows the baselines scores for the RNN bi-LSTM, LSTM Attention Model architecture.

5.2. Transformer

Transformer architecture was presented in (Sizhu Cheng, 2019) as well. However, we weren't able to reproduce the transformer results as we didn't have access to the dataset that was used to report results. Instead, we created our baseline Transformer by downloading the Transformer model for language understanding notebook from (tft), which implements the transformer model presented in (Vaswani et al., 2017). We list below the important functions from the (Vaswani et al., 2017).

Attention Function

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where, head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Figure 3 shows the Transformer Framework. It uses the transformer model to predict equations, and includes the solver to solve equations. In the next section, we discuss the architecture exploration and hyperparameter tuning results.

5.2.1. BASELINE SETUP

The transformer notebook only has the model implementation. In order to evaluate various transformer architectures, we added tensorboard logging. Figure 4 shows the loss and accuracy for the baseline transformer configuration. The baseline transformer we chose had number of layers as 4, dimensionality of input and output (Embed Size) as 128; inner layer dimensionality (Hidden Size) as 512, and number of heads as 8.

6. Experiments

6.1. Evaluation Metrics

For evaluating the various transformer architectures, we used BLEU score and solution accuracy. We looked at n-gram, where n=1..4, BLEU scores, as well as the cumulative score of 1..4 n-grams. We also considered solution accuracy as an evaluation metric. In the case of solution accuracy, the score for an example is either 1 or 0.

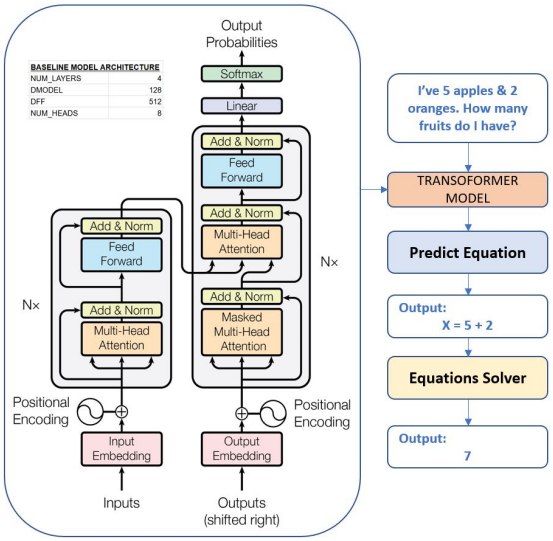
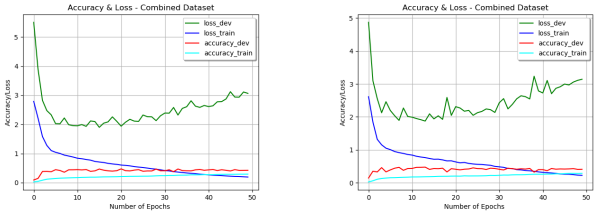


Figure 3. Transformer Framework. Figure reproduces the transformer model from (Vaswani et al., 2017).



Combined(4/128/512/8) Combined(6/256/1024/8)

Figure 4. Loss and Accuracy obtained using Tensorboard for the Combined dataset with two different transformer configurations

The loss function we used is SparseCategoricalCrossentropy from Tensorflow Keras implementation. We also used SparseCategoricalAccuracy from Tensorflow Keras for analysis the accuracy every steip of the model. We also implemented our Equation Solver to compute solution accuracy.

6.2. Hyperparameter Tuning

Table 5 shows the range of values for the hyperparameters we explored. For the transformer architecture, we changed the number of layers, embed size, hidden size and the number of attention heads. For improving model accuracy, we explored various settings for learning rate, dropout and batch size. Figure 6 shows a sample of how the hyperpa-

ARCHITECTURE & HYPERPARAMETER TUNING

Hyperparameter	Range of Variations (guided by error analysis at each step)
Number of Layers	4, 6, 8
Embed Size	64, 128, 256 512
Hidden Size	128, 256, 512, 1024, 2048
Number of Attention Heads	8, 16
Dropout	0.05, 0.1, 0.15, 0.2, 0.3, 0.5
Batch Size	32, 64, 128
Learning Rate	0.0001 through 0.1, and CustomSchedule $lrate = d_{model}^{-0.5} * min(step\_num^{-0.5}, step\_num * warmup\_steps^{-1.5})$

Figure 5. Hyper Parameter

Figure 6. Hyperparameter Tuning Results

rameter tuning performed on our Ext-Elem dataset. We ran similar tuning for all our datasets, and different datasets performed well with different settings for transformer architecture, learning rate, dropout and batch size. We found the best setting for each of the dataset. We found that smaller values for learning rate, batch size and dropout did well for the smaller datasets. For bigger datasets and bigger transformers, higher dropouts and batch sizes worked better. We faced capacity issues on GPUs when doing large training runs.

6.3. Error Analysis

We analyzed the results from Ext-Elem and Combined datasets. One of the things we noticed is that the model was not generalizing to new numbers found in the word problems. We implemented the number mapping for word embedding to overcome this limitation. We also improved on our data pre-processing, which corrected and/or removed malformed questions and equations.

6.4. Discussion and Results

Figure 7 shows the results for our four datasets. The data we present are from the fine tuned models we obtained for each of the datasets. We trained our model on AWS EC2 instance, NVIDIA V100, P100 and GTX 1070 GPUs that were available to us. The bigger datasets and transformer architectures required bigger GPU machines and longer hours to train. Our fine tuning runs took several days on multiple machines. The biggest models we training had approximately 13 million parameters.

Our Ext-Elem dataset achieves a very high accuracy of 84% after we did the number mapping. The combined dataset achieved an accuracy of 62%. Our number mapping for word embedding helped both datasets, but it did exceptionally well on the Combined dataset. One of the reasons for could be that the combined dataset had examples form Dolphin-18K, and many of the questions in that dataset weren't very well written.

Dataset	Model Architecture	BLEU-4	Solution Accuracy
Ext-Elem-Mapped	0.1 - 6 - 256 - 1024 - 8	43.50	84.40
	0.1 - 4 - 256 - 1024 - 8	40.08	83.60
Ext-Elem	0.1 - 4 - 256 - 1024 - 8	27.66	63.20
	0.1 - 6 - 256 - 1024 - 8	25.94	58.00
Combined-Mapped	0.1 - 4 - 128 - 512 - 8	33.40	62.07
	0.1 - 4 - 128 - 256 - 8	31.60	61.45
Combined	0.3 - 6 - 256 - 1024 - 16	7.12	10.70
	0.1 - 6 - 256 - 1024 - 8	6.93	9.90

Figure 7. Results for different datasets

7. Conclusion and Future Work

We reproduced Bi-LSTM, LSTM Attn Model based work from (Sizhu Cheng, 2019) for our initial setup. We analyzed BLEU scores and predicted equations to conclude that BLEU score alone is not sufficient evaluation metrics. We developed our equation solver, and used it to compute solution accuracy scores. Further error analysis on baseline transformer results helped us to develop number mapping technique. Using number mapping for word embedding, we obtained much improved results. Dataset with elementary problems in general gave better results since they were cleaner. Combined dataset didn't perform as well, since several examples had inconsistencies in problems and equations especially in Dolphin18k dataset. We tuned transformer with number mapping for word embedding, and equation solver accuracy metrics resulted in improved prediction.

As part of future work, we plan to implement beam search for transformer model to improve prediction quality. We also would like to try larger AQUA-RAT (aqu) dataset to extract equations, and obtain results. We plan to use transformer-XL (Dai et al., 2019) and BERT (Devlin et al., 2018), (Song et al., 2019) like pre-training to get better models. We would like to do further research on how to generalize to entirely new problem sets. Our eventual goal is to build an end-to-end application, that can scan a textbook problem (printed or hand-written), convert the scanned image into text, understand the text as a math word problem and output its solution.

Acknowledgements

We'd like to show our gratitude towards CS230 course teaching staff in general for helping us with queries related to Deep Learning that helped us develop strong understanding of the subject which we could apply in this project. Further, we're immensely grateful to teaching assisstant **Advay Pal** who guided us at each step of the project work. Without his constant support and advice, this project would not have been possible.

Contributions

All the team members have contributed equally to this project through very strong collaboration. We had all discussions and brainstorming sessions together. We divided the work amongst ourselves for the research and development to go in parallel as much as possible. All of us worked in some capacity on all the pieces.

References

Diverse algebra word problem dataset with derivation annotations. URL https://www.microsoft.com/en-us/download/details.aspx?id=52628.

Aqua-rat (algebra question answering with rationales). URL https://deepmind.com/research/open-source/aqua-rat-algebra-question-answering-with-rationales.

Cs224n, assigment #4. URL http://web.stanford.edu/class/cs224n/assignments/a4.pdf.

Transformer model for language understanding. URL https://www.tensorflow.org/tutorials/text/transformer.

Bobrow, Daniel G. Natural language input for a computer problem solving system. 1964.

Charniak, Eugene. Computer solution of calculus word problems. In *Proceedings of the 1st international joint conference on Artificial intelligence*, pp. 303–316. Morgan Kaufmann Publishers Inc., 1969.

Clark, Peter. Elementary school science and math tests as a driver for ai: take the aristo challenge! In *Twenty-Seventh IAAI Conference*, 2015.

Clark, Peter and Etzioni, Oren. My computer is an honor student—but how intelligent is it? standardized tests as a measure of ai. *AI Magazine*, 37(1):5–12, 2016.

Dai, Zihang, Yang, Zhilin, Yang, Yiming, Carbonell, Jaime G., Le, Quoc V., and Salakhutdinov, Ruslan. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019. URL http://arxiv.org/abs/1901.02860.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. BERT: pre-training of deep bidirectional transformers for language understanding. 2018. URL http://arxiv.org/abs/1810.04805.

Feigenbaum, Edward A, Feldman, Julian, et al. *Computers and thought*. New York McGraw-Hill, 1963.

Huang, Danqing, Shi, Shuming, Lin, Chin-Yew, Yin, Jian, and Ma, Wei-Ying. How well do computers solve math word problems? large-scale dataset construction and evaluation. In *Proceedings of the 54nd Annual Meeting of the Association for Computational Linguistics*, pp. 887–896, 2016.

Kushman, Nate, Artzi, Yoav, Zettlemoyer, Luke, and Barzilay, Regina. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 271–281, 2014.

Papineni, Kishore, Roukos, Salim, Ward, Todd, and Zhu, Wei-Jing. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pp. 311–318, 2002.

Peters, Matthew E, Neumann, Mark, Iyyer, Mohit, Gardner, Matt, Clark, Christopher, Lee, Kenton, and Zettlemoyer, Luke. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Shi, Shuming, Wang, Yuehui, Lin, Chin-Yew, Liu, Xiaojiang, and Rui, Yong. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1132–1142, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1135. URL https://www.aclweb.org/anthology/D15-1135.

Sizhu Cheng, Nicolas Chung. Simple mathematical word problems solving with deep learning. 2019.

Song, Kaitao, Tan, Xu, Qin, Tao, Lu, Jianfeng, and Liu, Tie-Yan. MASS: masked sequence to sequence pre-training for language generation. *CoRR*, abs/1905.02450, 2019. URL http://arxiv.org/abs/1905.02450.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Ł ukasz, and Polosukhin, Illia. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pp. 5998–6008, 2017.

Wang, Lei, Wang, Yan, Cai, Deng, Zhang, Dongxiang, and Liu, Xiaojiang. Translating a math word problem to an expression tree. 2018. URL http://arxiv.org/abs/1811.05632.

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

Zhang, Dongxiang, Wang, Lei, Zhang, Luming, Dai, Bing Tian, and Shen, Heng Tao. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE transactions on pattern analysis and machine intelligence*, 2019.