Sequence to General Tree: Knowledge-Guided Geometry Word Problem Solving

Shih-hung Tsai, Chao-Chun Liang, Hsin-Min Wang, Keh-Yih Su
Institute of Information Science, Academia Sinica, Taiwan
{doublebite,ccliang,whm,kysu}@iis.sinica.edu.tw

Abstract

With the recent advancements in deep learning, neural solvers have gained promising results in solving math word problems. However, these SOTA solvers only generate binary expression trees that contain basic arithmetic operators and do not explicitly use the math formulas. As a result, the expression trees they produce are lengthy and uninterpretable because they need to use multiple operators and constants to represent one single formula. In this paper, we propose sequence-to-general tree (S2G) that learns to generate interpretable and executable *operation trees* where the nodes can be formulas with an arbitrary number of arguments. With nodes now allowed to be formulas, S2G can learn to incorporate mathematical domain knowledge into problem-solving, making the results more interpretable. Experiments show that S2G can achieve a better performance against strong baselines on problems that require domain knowledge.[1]

1 Introduction

Math word problem (MWP) solving is a special subfield of question answering. It requires machine solvers to read the problem text, understand it, and then compose the numbers and operators into a meaningful equation (as shown in Table 1). This process, even for the simplest problem in elementary school, demands language understanding and numerical reasoning capabilities, making this task a long-standing challenge in AI (Bobrow, 1964; Zhang et al., 2019).

As with any QA task, solving an MWP requires the introduction of external knowledge or domain knowledge (Mishra et al., 2020). However, current state-of-the-art solvers (Xie and Sun, 2019; Zhang et al., 2020; Wu et al., 2020) do not address this issue explicitly. They learn to map the problem text into binary expression trees regardless of whether it requires any knowledge. This is counterintuitive for problems that need math concepts or formulas. As illustrated in Figure 1(a), without explicitly using the corresponding area formula, the expression tree for the problem is lengthy and uninterpretable.

To address this issue, we propose a sequence-to-general tree (S2G) architecture where the nodes

[1]Data and code are available at the GitHub repository: https://github.com/doublebite/Sequence-to-General-tree/

Problem: The outer radius and the inner radius of a circular annulus are 5m and 3m repsectively. Find the area of this circular annulus.
Equation: $x = 5 * 5 * 3.14 - 3 * 3 * 3.14$ **Answer:** 50.24
With formula: x = circle_area(5) - circle_area(3)

Table 1: Example problem that requires geometry knowledge.

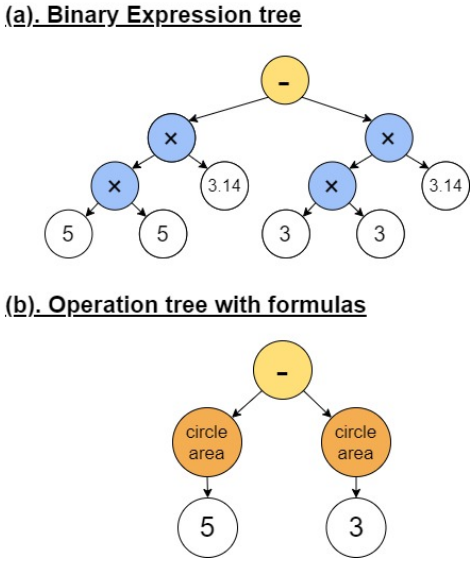


Figure 1: (a). binary expression tree and (b). operation tree along with formulas for the problem in Table 1.

can be arbitrary math concepts or formulas with arbitrary number of arguments. In this way, our S2G model can learn to map the problem text into executable operation trees that contain different formulas across different domains. For example, S2G can learn to generate tree nodes that contain the required geometry formula for circles, as shown in Figure 1(b), making the result more intuitive and explainable.

In addition, we propose a knowledge-guided mechanism to guide tree-decoding using a mathematical knowledge graph (KG). To evaluate our model, we also construct a middle-sized dataset consisting of 1,398 geometry word problems which require a diversified set of formulas. Experimental results show that our S2G model can provide better performance and more interpretable results against strong baselines on problems that require domain knowledge.

The main contributions of this paper are:

1. We propose a seq-to-general tree model that learns to map the problem text into operation trees where the nodes can be formulas with arbitrary number of arguments. This helps to incorporate domain knowledge into problem solving and produce interpretable results.

2. We design a knowledge-guided mechanism that guides tree decoding using mathematical knowledge graphs and GNNs.

3. We curate a middle-sized dataset that contains 1,398 geometry word problems. In addition, we annotate them with detailed formulas that can be readily converted into operation trees.

2 Seq2seq v.s. Seq2tree v.s. Seq2general

Our goal is to design a sequence-to-general tree model that learns to map the problem text into its corresponding operation tree. Before diving into the model, we first compare the decoding mechanisms between seq-to-seq, seq-to-tree and our seq-to-general tree solvers. Figure 2 illustrates the tree decoding process of these three types of model, respectively.

For seq2seq models, their decoder basically does two things: (1) predicting the current output and (2) generating the next state. These two steps can be conditioned on different information including the current state, the current input, or a context vector calculated using attention. The decoder would repeat these two steps until it outputs an end token.

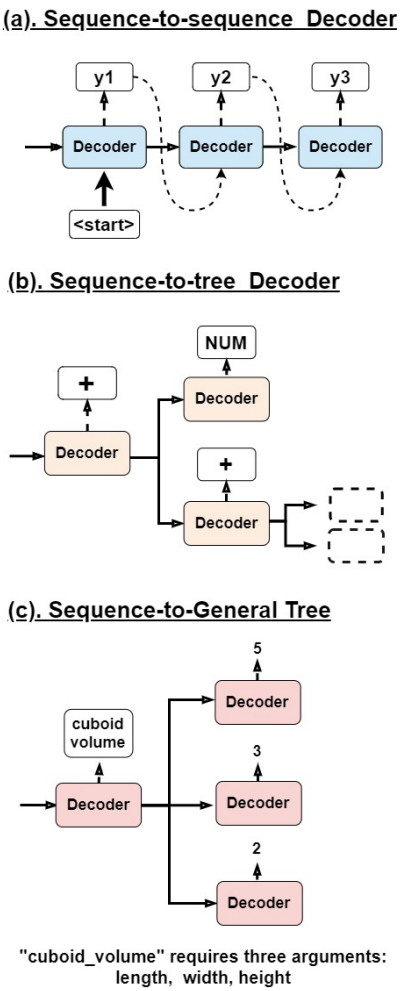


Figure 2: Comparison between three types of decoding: (a) seq2seq, (b) seq2tree, and (c) seq-to-general tree.

For seq2tree models, however, this process is slightly different. The decoder predicts the current output as in seq2seq, but it will decide whether to generate the next state based on the current output. If the current output is a arithmetic operator, the decoder knows it should produce two child states, and these states are used to expand into its left and right children. If the current output is a number, then the decoder would end the decoding process, so the current node becomes a leaf node. As a result, the whole decoding process resembles generating an expression tree in a top-down manner.

In our work, we generalize the decoding process by making the decoder produce a variable number of children based on the type of the current output. If the output is a number or operator, the decoder would produce zero or two child states as before. If the output is a formula, the decoder will generate the pre-specified number of child states for this formula.

3 Sequence-to-General Tree Model

In this section, we give a detailed description for each part of our S2G model.

3.1 Encoder

The main function of the encoder is to encode the problem text $P = (x_1, x_2, ..., x_n)$ into a sequence of hidden states $(h_1, h_2, ..., h_n)$ and their summary state $h_{encoder}$. The hidden states h_1 to h_n are expected to contain the information for each input token x_1 to x_n, while the summary state $h_{encoder}$ is expected to capture the overall information of the problem.

Specifically, we use bidirectional gated recurrent units (GRU) (Cho et al., 2014) as our encoder. Given the current input x_t, the previous state h_{t-1}, and the next state h_{t+1}, the current state $h_t \in (h_1, h_2, ..., h_n)$ can be calculated with:

$$\overrightarrow{h_t} = \mathrm{GRU}(x_t, \overrightarrow{h_{t-1}}), \quad (1)$$

$$\overleftarrow{h_t} = \mathrm{GRU}(x_t, \overleftarrow{h_{t+1}}), \quad (2)$$

where the arrows represent different directions in the bidirectional encoding. After calculating the hidden state for each input token, we combine the last state of the forward and backward directions to get the summary state for the encoder:

$$h_{encoder} = \overleftarrow{h_0} + \overrightarrow{h_n} \quad (3)$$

3.2 Geometry Knowledge Graph

To incorporate domain knowledge into problem solving, we propose to utilize the knowledge from mathematical knowledge graphs. The main idea is that given a formula predicted as the current node, we could use the physical meaning of its arguments to help us better predict its children. For example, if the current node is the formula for rectangle area, then we know its child nodes should be related to "length " and "width". We can thus use the node embeddings of "length" and "width" from a geometry KG to provide additional information for our solver.

We manually collect a geometry knowledge graph which contains the common geometry shapes (e.g., square, circle) and their geometry quantities (e.g., area, length), and we link these nodes to each other if they belong to the same shape. To embed this KG, we employ a graph convolutional network (GCN) (Kipf and Welling, 2017) that transforms the KG into some vector space and calculates the embedding of each node. Given the feature matrix X and the adjacency matrix A of the KG, we use a two-layer GCN to encode it as follows:

$$Z = \mathrm{GCN}(X, A), \quad (4)$$

where $Z = (z_1, ..., z_n)$ are the node embeddings for each node in the graph. Then, we can use the embedding to represent the physical meaning of a certain formula argument in the decoding process.

3.3 General Tree Decoder

In the decoding stage, the decoder learns to produce the target operation trees in a recursive manner. It first predicts the current output y_t in order to determine the number of children of the current node. Given the current decoder state s_t, the embedding of the last output $e_{(y_{t-1})}$, and the node embedding z_t which represents the physical meaning in the knowledge graph, the probability of the current output $P(y_t)$ is calculated using:

$$c_t = \mathrm{Attention}(\mathrm{e}_{(\mathrm{y_{t-1}})}, \mathrm{h}_1^{\mathrm{n}}) \quad (5)$$

$$z'_t = \mathrm{Attention}(\mathrm{z_t}, \mathrm{h}_1^{\mathrm{n}}) \quad (6)$$

$$P(y_t) = \mathrm{Softmax}(W_y[s_t; e_{(y_{t-1})}; c_t; z'_t]), \quad (7)$$

where h_1^n is the encoder states $(h_1, ..., h_n)$, c_t is the context vector of $e_{(y_{t-1})}$ with respect to h_1^n, and z'_t is another context vector calculated using the node embedding z_t and h_1^n. Specifically, we use additive attention (Bahdanau et al., 2015) to calculate these context vectors and use $h_{encoder}$ as the first decoder state s_0. Given the probability $P(y_t)$, we can then determine the output token $\hat{y}_t$:

$$\hat{y}_t = \mathrm{argmax} P(y_t). \quad (8)$$

Next, we predict the child states conditioned on the required number of children for $\hat{y}_t$. Unlike previous binary-tree decoders that use two distinct DNNs to predict the left and right children respectively (Xie and Sun, 2019; Zhang et al., 2020; Wu et al., 2020), we employ a GRU to predict a variable number of children. Given the current state s_t, its child states $s_{t_1}, ..., s_{t_n}$ are generated in a recurrent manner:

$$s_{t_i} = \mathrm{Decoder}(s_{t_{i-1}}; e_{(y_t)}; c_t), \quad (9)$$

where we generate the first child s_{t_1} using s_t, and the following child state s_{t_i} using its previous sibling $s_{t_{i-1}}$until we reach the required number of children. The decoder is basically a GRU followed

by a linear projection layer and an activation function:

$$s'_{t_i} = \text{GRU}([e_{(y_t)}; c_t], s_{t_{i-1}}), \quad (10)$$

$$s_{t_i} = \text{ReLU}(W_s s'_{t_i}), \quad (11)$$

where the input of GRU is the concatenation of $e_{(y_t)}$ and c_t, W_s is the linear projection layer, and ReLU is used as the activation function. After getting these child states, we push them into a stack and repeat the steps from Equation (5) to Equation (11) until all the states are realized into tokens.

3.4 Training Objective

For a problem and operation tree pair (*P, T*), we follow previous seq2tree work (Xie and Sun, 2019; Wu et al., 2020) and set our objective to minimize the negative log likelihood:

$$L(T, P) = \sum_{t=1}^{n} -logP(y_t|s_t, P, KG). \quad (12)$$

4 Dataset

To evaluate our S2G model on problems that require formulas, we curate a middle-sized dataset, *GeometryQA*, that contains 1,398 geometry word problems. These problems are collected from Math23K (Wang et al., 2017) using the keywords of common geometric objects (e.g., circle, square, etc.) and their shapes (e.g., rectangular, circular, etc.). Then, we re-annotate each problem with their associated formulas if the problem belongs to one of the six major shapes: *square*, *cubic*, *rectangle*, *cuboid*, *triangle* and *circle*. Table 2 shows the overall statistics of GeometryQA and Table 7 in Appendix B shows the 11 formulas we used to annotate these problems.

Note that not all problems in GeometryQA are annotated with formulas. About 16% of the problems belong to other shapes (e.g., *parallelogram*, *rhombus*, etc.) which currently are not covered in our formula set. About 40% are problems that contain geometric keywords but do not actually require any formulas. Table 3 shows such an example. We use these problems to test the robustness of our model. That is, S2G has to learn to apply the correct formulas or equations from misleading keywords (as shown in Table3) and has to learn to generate both binary expression trees and operation trees as a whole.

GeometryQA	
Number of problems	1,398
Number of sentences/words	5.4k / 41.1k
Vocabulary size	2,872
Annotated with formulas	604 (43.20%)
Problems of other shapes	225 (16.09%)
Formulas not required	569 (40.70%)

Table 2: Dataset statistics of GeometryQA

Problem: The perimeter of a rectangular swimming pool is 300 m. If you place a chair every 10 m all the way around its perimeter, how many chairs do you need?
Equation: $x = 300/10$ **Answer:** 30

Table 3: Example problem that contains misleading keywords (perimeter, rectangular) but do not require any geometry formulas.

5 Experiments

5.1 Implementation Details

We implement our S2G model and the GNN module using Pytorch[2] and Pytorch Geometric[3]. We set the dimension of word embedding to 128 and the dimension of the hidden state of GRU and GNN to 512. The dropout rate (Srivastava et al., 2014) is set to 0.5 and the batch size is 64. For optimization, we use ADAM (Kingma and Ba, 2015) with a learning rate of 10^{-3} and a weight decay of 10^{-5}. Besides, we use a learning rate scheduler to reduce the learning rate by half every 20 epochs. During evaluation, we use beam search (Wiseman and Rush, 2016) with a beam size of 5.

5.2 Experimental Results on GeometryQA

We evaluate our S2G model on GeometryQA to check whether it can learn to predict the corresponding operation tree for the geometry word problems. Table 4 shows the results of our S2G against other seq2tree SOTA models. S2G is trained using the re-annotated equations that contain formulas, while the baselines are trained using the original equations.

First, we find that S2G has about 3.8% perfor-

[2]https://pytorch.org/

[3]https://pytorch-geometric.readthedocs.io/en/latest/

mance gain over its baselines (with p-value < 0.01). We attribute this to the fact that operation trees are easier to learn and generate since they are less lengthy and complex than binary expression trees. Hence, there is a better chance for S2G to produce the correct trees and arrive at the correct answers.

Second, there is a small performance gain by adding Geometry KG. However, the improvement is not significant (with p-value$\approx$0.8). We guess that is because the dataset currently has only six geometric objects, which is not complex enough to show the effectiveness of adding knowledge graphs.

Model	Accuracy(%)
KA-S2T (Wu et al., 2020)	49.61%
GTS (Xie and Sun, 2019)	51.01%
S2G	**54.79%**
S2G + Geometry KG	**54.99%**

Table 4: Answer accuracy of S2G and other SOTA seq2tree models on GeometryQA (all evaluated with 5-fold cross validation).

6 Conclusion

In this work, we proposed a sequence-to-general tree model (S2G) that aims to generalize previous seq2tree architectures. Our S2G can learn to generate executable operation trees where the nodes can be formulas with arbitrary number of arguments. By explicitly generating formulas as nodes, we make the predicted results more interpretable. Besides, we also proposed a knowledge-guided mechanism to guide the tree decoding using KGs and constructed a dataset in which problems are annotated with associated formulas. Experimental results showed that our S2G model can achieve better performance against strong baselines.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*.

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

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

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

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533.

Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. 2018. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 213–223.

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to solve math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 805–814.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 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)*, pages 271–281.

Jierui Li, Lei Wang, Jipeng Zhang, Yan Wang, Bing Tian Dai, and Dongxiang Zhang. 2019. Modeling intra-relation in math word problems with different functional multi-head attentions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6162–6167.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. 2020. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2841–2852, Online. Association for Computational Linguistics.

Chao-Chun Liang, Shih-Hong Tsai, Ting-Yun Chang, Yi-Chung Lin, and Keh-Yih Su. 2016. A meaning-based English math word problem solver with understanding, reasoning and explanation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 151–155, Osaka, Japan. The COLING 2016 Organizing Committee.

Chao-Chun Liang, Yu-Shiang Wong, Yi-Chung Lin, and Keh-Yih Su. 2018. A meaning-based statistical English math word problem solver. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 652–662, New Orleans, Louisiana. Association for Computational Linguistics.

Qianying Liu, Wenyv Guan, Sujian Li, and Daisuke Kawahara. 2019. Tree-structured decoding for solving math word problems. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2370–2379.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, and Chitta Baral. 2020. Towards question format independent numerical reasoning: A set of prerequisite tasks. *arXiv preprint arXiv:2005.08516*.

Arindam Mitra and Chitta Baral. 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)*, pages 2144–2153.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada. Association for Computational Linguistics.

Subhro Roy and Dan Roth. 2018. Mapping to declarative knowledge for word problem solving. *Transactions of the Association for Computational Linguistics*, 6:159–172.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1132–1142.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA. MIT Press.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *International Conference on Learning Representations*.

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

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas. Association for Computational Linguistics.

Qinzhuo Wu, Qi Zhang, Jinlan Fu, and Xuan-Jing Huang. 2020. A knowledge-aware sequence-to-tree network for math word problem solving. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7137–7146.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *IJCAI*, pages 5299–5305.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen. 2019. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE transactions on pattern analysis and machine intelligence*, 42(9):2287–2305.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937, Online. Association for Computational Linguistics.

A Data Preprocessing

In this section, we describe the data preprocessing steps required for our S2G model.

A.1 Converting to prefix notation

To perform top-down tree decoding, we follow (Xie and Sun, 2019) to convert the equations into their *prefix notation*, where the operators are placed in front of their operands, rather than in between. In this way, the order of the equation tokens would match the order of decoding. In our case, we also need to consider the formulas used in the equation. For a formula in the form "$F(arg1, arg2)$", we turn it into "$[F, arg1, arg2]$" so that it can fit into the prefix notation. Table 5 shows an example of this infix-to-prefix conversion for an equation with formulas.

Problem: The outer radius and inner radius of a circular annulus are 5m and 3m respectively. Find the area of this circular annulus.
Equation: x = circle_area(5) - circle_area(3) **Prefix form:** [-, circle_area, 5, circle_area, 3]

Table 5: Infix-to-prefix conversion for an equation with formulas.

A.2 Vocabulary

We follow the canonical sequence-to-sequence architecture (Sutskever et al., 2014) to prepare for the source vocabulary. For the target vocabulary, however, we have to take into consideration the way that humans solve MWPs. To solve a math problem, we use the numbers from the problem text (a dynamic vocabulary) and the mathematical operators learned before (a static vocabulary) and try to compose them into an equation. Sometimes, we also need to use external constant numbers (a static vocabulary) that are not in the problem text but would appear in the equation (e.g., 1, 2, or 3.14). These three types of vocabulary make up the vocabulary for the equations in arithmetic problems (equation 13).

$$V_{arith} = V_{number} \cup V_{op} \cup V_{const} \quad (13)$$

We follow (Xie and Sun, 2019) to use a copy mechanism (Gu et al., 2016) to copy the numbers from the problem text. Hence, we can dynamically get the problem numbers during decoding. Besides, we extend the original vocabulary by adding domain-specific formulas into it so that the decoder can generate formulas during decoding (equation 14). Table 6 shows the overall vocabulary that we use for our decoder.

Vocab Type	Instances
Operator	+, -, *, /, ^
Number	$\langle N0 \rangle$, $\langle N1 \rangle$, $\langle N2 \rangle$, ...
Constant	1, 2, 3.14
*Formula	circle_area, square_area, rectangle_perimeter, and so on.

Table 6: Types of the vocabulary.

$$V_{target} = V_{number} \cup V_{op} \cup V_{const} \cup V_{formula} \quad (14)$$

B GeometryQA

Table 7 shows the 11 formulas used for annotation.

Name	Formula	# args
Square		
square_area	side * side	1
square_perimeter	4 * side	1
Cubic		
cubic_volume	side*side*side	1
Circle		
circle_area	π * radius^2	1
circumference_r	2 * π * radius	1
circumference_d	π * diameter	1
Triangle		
triangle_area	base*height / 2	2
Rectangle		
rectangle_area	length * width	2
rectangle_perimeter	2 * (l+w)	2
Cuboid		
cuboid_volume	l* w* height	3
cuboid_surface	2*(l*w+w*h+l*h)	3

Table 7: Eleven geometry formulas used in annotating GeometryQA.

C Related Work

In this section, we briefly introduce the progress of MWP solvers, and then we focus on topics that are

closer to our work, including seq2tree solvers and knowledge graphs for problem solving.

C.1 Math Word Problem Solving

Ever since 1960s, efforts have been made to build automatic math word problem solving systems (Feigenbaum et al., 1963; Bobrow, 1964). Statistical solvers learn to map problem features into corresponding equation templates or operations to solve the problem (Kushman et al., 2014; Hosseini et al., 2014; Mitra and Baral, 2016; Liang et al., 2016, 2018; Roy and Roth, 2018). For example, Kushman et al. (2014) propose to align MWPs to their templates, while Hosseini et al. (2014) propose to find the operations by verb categorization. Semantic parsing approaches, on the other hand, parse the problem into intermediate representations using semantic parsers (Shi et al., 2015; Koncel-Kedziorski et al., 2015; Huang et al., 2017).

Recently, neural architectures have emerged as a dominant paradigm in math word problem solving. Wang et al. (2017) first attempt to use a seq2seq solver that utilize encoder-decoder architectures to encode the problem text and then decode into equations in a way similar to machine translation. Copy mechanism (Huang et al., 2018) or attention mechanisms (Li et al., 2019) are introduced to improvement the performance of seq2seq models. These seq2seq models, however, suffer from producing invalid equations, like a binary operator with three operands, because there is no grammatical constraint on its sequential decoding. To solve this problem, seq2tree models are proposed to bring into the grammatical constraints (Xie and Sun, 2019; Liu et al., 2019). We will give a more detailed introduction to seq2tree models in Section C.2.

C.2 Sequence-to-Tree Models

To convert text into structured representations, several research strands have utilized sequence-to-tree models. Dong and Lapata (2016) first use seq2tree on semantic parsing to translate text into structured logical forms. Similar frameworks are also adopted for code generation (Yin and Neubig, 2017; Rabinovich et al., 2017) where they translate code snippets into executable representations or abstract syntax trees (ASTs).

Inspired by their ideas, MWP solving also adopts seq2tree to map the problem text into expression trees. This introduces a constraint that the non-leaf nodes of the tree should be operators and leaf nodes be numbers, and thus the resulted equations are always guaranteed to be valid. Most seq2tree solvers choose bidirectional LSTM or GRU as their text encoder and use two separate models to predict the left and right nodes during decoding respectively (Xie and Sun, 2019; Zhang et al., 2020; Wu et al., 2020; Li et al., 2020). Our model differs from the previous that we use a single RNN-based decoder to predict a variable number of children nodes during decoding. In addition, our model can predict formulas as nodes that increase the interpretability of the model outputs, while previous solvers can only predict basic arithmetic operators.

C.3 Knowledge Graph for Math Word Problem Solving

To incorporate external knowledge into problem solving, some solvers utilize graph convolutional networks (Kipf and Welling, 2017) or graph attention networks (Veličković et al., 2018) to encode knowledge graphs (KGs) as an additional input. Wu et al. (2020) proposed to incorporate commonsense knowledge from external knowledge bases. They constructed a dynamic KG for each problem to model the relationship between the entities in the problem. For example, ”daisy” and ”rose” would be linked to their category ”flower” so that the solver can use this hyperonymy information when counting the number of flowers. Zhang et al. (2020) proposed to build graphs that model the quantity-related information using dependency parsing and POS tagging tools (Manning et al., 2014). Their graphs provide better quantity representations to the solver. Our model differs from previous models that we aim to incorporate domain knowledge from mathematical KGs rather than from commonsense knowledge bases.