Tree-structured Decoding for Solving Math Word Problems

Qianying Liu[*1], **Wenyu Guan**[*23], **Sujian Li**[2] **and Daisuke Kawahara**[1]
[1] Graduate School of Informatics, Kyoto University
[2] Key Laboratory of Computational Linguistics, MOE, Peking University
[3] School of Software and Microelectronics, Peking University
ying@nlp.ist.i.kyoto-u.ac.jp; guanwy@pku.edu.cn;
lisujian@pku.edu.cn; dk@i.kyoto-u.ac.jp

Abstract

Automatically solving math word problems is an interesting research topic that needs to bridge natural language descriptions and formal math equations. Previous studies introduced end-to-end neural network methods, but these approaches did not efficiently consider an important characteristic of the equation, i.e., an abstract syntax tree. To address this problem, we propose a tree-structured decoding method that generates the abstract syntax tree of the equation in a top-down manner. In addition, our approach can automatically stop during decoding without a redundant stop token. The experimental results show that our method achieves single model state-of-the-art performance on Math23K, which is the largest dataset on this task.

1 Introduction

Math word problem (MWP) solving is the task of obtaining a solution from math problems that are described in natural language. One example of MWP is illustrated in Figure 1. Early approaches rely on either predefined rules or statistical machine learning-based methods to map the problems into several predefined templates in classification style or retrieval style. The major drawback of these approaches is that they are inflexible for new templates and require extra effort to design rules, features and templates.

Modeling the tree structure of math equations has been considered as an important factor when building models for MWP. As shown in Figure 1, each equation could be transformed into an abstract syntax tree (AST). Roy and Roth (2017) built an expression tree and combined two classifiers for quantity relevance prediction and operator classification. Koncel-Kedziorski et al. (2015) designed a template ranking function based on the AST of the equations. However, these approaches are based on traditional methods and require feature engineering.

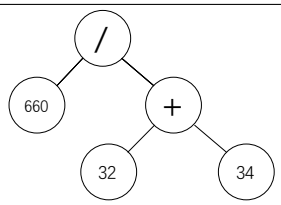
Problem	The distance between the two places A and B is 660 km, the car starting from A drives 32 km/h, and the car starting from B drives 34 km/h. The two cars are starting from the two places at the same time in inverse direction. How many hours later would the two cars meet?
Equation	x = 660 / (32 + 34)
Prefix	x = / 660 + 32 34
Answer	10
AST	

Figure 1: One example of MWP. Problem refers to the natural language descriptions. Equation refers to the formal math equation. Prefix refers to the prefix notation of the equation. Answer refers to the final quantity solution. AST refers to the AST of the equation.

Recently, the appearance of large-scale datasets and the development of neural generative models have opened a new research line for MWP. Wang et al. (2017) cast this task as a sequence generation problem and used a sequence-to-sequence (seq2seq) model to learn the mapping from natural language text to a math equation. Recent approaches use the Reverse Polish notation, also called suffix notation of equations in which operators follow their operands to implicitly model the tree structure (Wang et al., 2018a; Chiang and Chen, 2018). However, these studies lost sight of important information of the math equation ASTs

[*] This denotes equal contribution.

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 2370–2379, Hong Kong, China, November 3–7, 2019.

(e.g., parent and siblings of each node), despite of their promising results. Thus, their model has to use extra effort to memorize various pieces of auxiliary information such as the sibling node of the current step and learn the implicit tree structure in the sequence.

Meanwhile, in similar tasks such as semantic parsing and code generation, which also try to convert natural language into a well-formed executable tree structured output, they first decode the function or operator at the root node before decoding the variables. Such top-down decoding matches with the Polish notation, also called the prefix order of the AST of math equations, shown in Figure 1. Human also does reasoning and reference in a similar order, usually first determines the function before filling in the variables, but not the inverse. Thus we present a top-down hierarchical sequence-to-tree (seq2tree) model which explicitly models the tree structure. This decoder considers the prefix order of the equation and feeds the information of the parent and sibling nodes during decoding.

Another advantage of this system is that it can use a stack to monitor the decoding process and automatically end. By pushing in new generated nodes to the stack and popping out the completed subtrees, the decoding process can naturally end when the tree is completed and the stack is empty. There is no need for a special redundant end-of-sequence (EOS) token which is used in ordinary text sequence generation. Without the EOS token, the model has a higher possibility to generate valid equation answers.

In summary, the contributions of this work are as follows:

1. We design a hierarchical seq2tree model that can better capture information of the AST of the equation.

2. We are the first to use prefix order decoding for MWP.

3. We abandon the EOS token for end-to-end equation generation.

4. Our model outperforms previous systems for solving math word problems. On the large-scale dataset Math23K, we achieve state-of-the-art single model performance.

2 Related Work

Our work synthesizes two strands of research, which are math word problems and seq2tree encoder-decoder architectures.

2.1 Math Word Problems

Early approaches hand engineered rule-based systems to solve MWPs (Mukherjee and Garain, 2008). The rules could only cover a limited domain of problems, while math word problems are flexible in real-world settings.

There are currently three major research lines in solving MWPs. The first research line maps a problem text into logical forms, and then uses the logical forms to obtain the equation (Shi et al., 2015; Roy and Roth, 2015; Huang et al., 2017; Roy and Roth, 2018). Shi et al. (2015) defined a Dolphin language to connect math word problems and logical forms. The major drawback is that it requires extra human annotation for the logic forms.

Another research line uses either a retrieval or classification model to maintain a template, and then fills in the slots with quantities. Kushman et al. (2014) first introduced the idea of 'equation template'. For example, x = 6*7 and x = 10*5 belong to the same template x = n_1*n_2. They collected the first dataset of this task, ALG514, which contained 514 samples. They brought out a two-step pipeline model, which first used a classifier to select a template and then mapped the numbers into the slots. One major drawback is that it cannot solve problems beyond the templates in the training data. This two-step pipeline model was further extended with tree-based features, ranking style retrieval models and so on (Upadhyay and Chang, 2017; Roy and Roth, 2017). Huang et al. (2016) released the first large-scale dataset Dophin18K and trained a similar system on it.

The third research line directly generates the equation. Hosseini et al. (2014) cast the problem into a State Transition Diagram of verbs and trained a binary classifier that could solve problems with only add and minus operators. Wang et al. (2017) first used a seq2seq model to directly generate the equation template and released a Chinese high-quality large-scale dataset, Math23K. Reinforcement learning was used to further improve the seq2seq framework. Wang et al. (2018b) first extended the seq2seq model by decoding the suffix order sequence of the equations. Wang et al. (2018a) introduced equation normalization techniques that leverage the duplicated template problem. Chiang and Chen (2018) used the copy

mechanism to improve the semantic representation of quantities. However, in these work the model needs extra effort to learn the implicit tree structure and memorize tree information in the sequence. Wang et al. (2019) proposed a two-step pipeline method that first generates a template with unknown operators, and then uses a recursive neural network to combine tree structure information and predict the operators. However, the topology of the AST is determined in the first step without tree structure information. To our best knowledge, we are the first to explicitly give the model guidance of parent and sibling nodes.

2.2 Seq2Tree Architectures

Seq2Tree-style encoder-decoder is mainly used in two fields both of which try to bridge natural language and a tree structured output.

Semantic parsing is the task that translates natural language text to formal meaning logical forms or structured queries. Code generation maps a piece of program description to programming language source code. Dong and Lapata (2016) first used recurrent neural networks (RNNs) based seq2tree for semantic parsing and out performed the seq2seq model. One drawback is that their generation is at token level so it cannot guarantee the result is syntactically correct. Grammar rules were used to solve this problem. Another drawback is that they needed special tokens for predicting branches, which are not necessary for MWPs because all operators are binary operators. The similar framework is also used in code generation (Zhang et al., 2016; Yin and Neubig, 2017). Alvarez-Melis and Jaakkola (2017) presented doubly recurrent neural networks to predict tree topology explicitly. Rabinovich et al. (2017) presented a abstract syntax network that combines edge information for code generation. Convolution neural networks (CNNs) were used for code generation decoding because the output program is much longer than semantic parsing and MWPs, and RNNs suffer from the long dependency problem (Sun et al., 2018).

3 Model

Our model consists of two stages as shown in Figure 2: the encoder stage that encodes the input natural language into a sequence of representation vectors and the decoder stage that receives these vectors and decodes the AST of the equation with the help of an auxiliary stack.

Equation	x = 130 * (1 - 0.8)
Quantities	n_1: 130, n_2: 0.8
Template	n_1 * (1 - n_2)
Prefix template	* n_1 - 1 n_2

Table 1: One example of prefix template

3.1 Preprocessing

Significant Number Identification A Significant Number Identification (SNI) unit is used for reducing the noise in the input numbers. Significance refers to whether the number appears in the equation. In MWPs, it is very common that the input text contains irrelevant numbers, such as date or descriptive text such as 'third grade student'. We follow Wang et al. (2017) and simply use whether the numbers appear in the equation as gold labels and build an LSTM-based binary classification model to determine the significance of the input numbers. The accuracy of this unit is 99% and thus it can efficiently reduce the noise.

Prefix Order Equation Template For the output equations, we first turn them into prefix order equation templates before using them to train the model. In Table 1 we show one example of prefix templates. Given a problem-equation pair, we first build the mapping between numbers in the problem and equation, n_i denotes the ith number in the problem after SNI. We use this mapping to convert the equation into a template by replacing the numbers with n_i tokens, and at last convert the template into prefix order.

To be noticed, equations can be mapped to more than one AST. For example, $n1 + n2 + n3$ could be mapped to an AST with either the first $+$ or the second $+$ as the root node, and in that case the prefix order notation would also be different. We assume the first operator is the root node of the AST here. Further details are shown in the appendix.

Equation Normalization One math word problem can be solved by different but equivalent equations, which bring noise during training. For example, 10 - (8 + 5) and 10 - 8 - 5 are equivalent, but the templates $n_1 - (n_2 + n_3)$ and $n_1 - n_2 - n_3$ are different. This problem is called the equation duplication problem. We follow Wang et al. (2018a) and use several rules for equation normalization

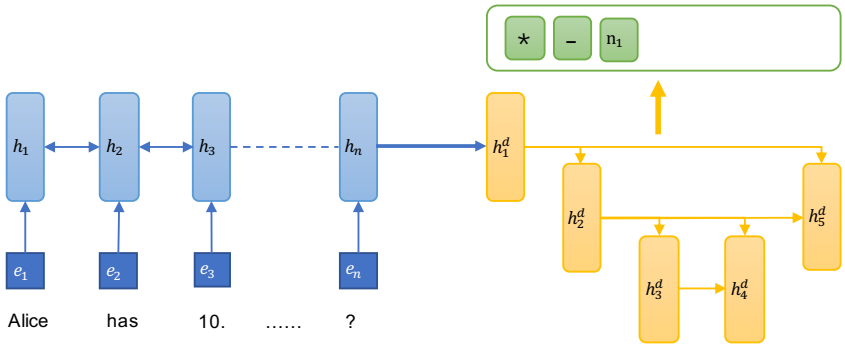


Figure 2: Framework of our seq2tree model. The blue blocks refer to the encoder. The yellow blocks refer to the decoder. The green blocks refer to the auxiliary stack.

to alleviate the equation duplication problem. The rules are listed below.

- If one long equation template could be converted into a shorter one, then it should be shortened. For example, $n_1+n_2+n_3+n_3-n_3$ and $n_1+n_2+n_3$ are equivalent. In this case the former one should be normalized as the latter one.
- The order of number tokens in the template should follow their occurrence order in the problem text as much as possible. For example, $n_1+n_3+n_2$ should be normalized as $n_1+n_2+n_3$.

3.2 Encoder

Bi-directional Long Short Term Memory Network (BiLSTM) is an efficient method to encode sequential information. Formally, given an input math word problem sentence $x = \{x_t\}_{t=1}^{n}$, we first embed each word into a vector e_t. Then these embeddings are fed into a BiLSTM layer to model the sequential information.

$$h_t = (LSTM(h_{t-1}^f, e_t); LSTM(h_{t-1}^b, e_t)), \quad (1)$$

where h_t is the concatenation of the hidden states h_t^f and h_t^b, which are from both forward and backward LSTMs. These representation vectors are then fed into the following decoder stage.

3.3 Tree-structured Decoder

For decoding, we follow Dong and Lapata (2016) and build a top-down hierarchical tree-structured decoder. However, their model is built for semantic parsing and some components are unnecessary and redundant for equation template decoding. Benefited by the fact that operator nodes must have two children and number nodes must be leaf nodes, we build a decoder which is specialized for math equation AST, as shown in Figure 3. It extends a vanilla sequence-based LSTM decoder by using tree-based information feeding as the input, and also an auxiliary stack to help the model know which is the next token to decode and automatically end the decoding process without a special token.

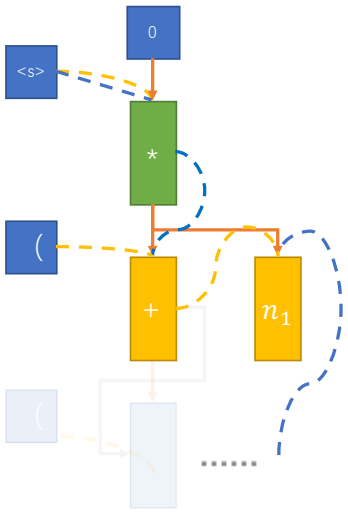


Figure 3: Framework of the tree structured decoder. The yellow dotted line refers to sibling feeding. The blue dotted line refers to previous token feeding. The orange solid line refers to parent feeding. The dotted line means it uses embedding information, while the solid line means it uses the hidden state information.

3.3.1 Tree-based Information Feeding

The input of each time step consists of three parts: parent feeding, sibling feeding and previous token feeding.

The parent feeding h^{parent} refers to using the LSTM hidden state of the parent node as the input when decoding children nodes, which is shown as

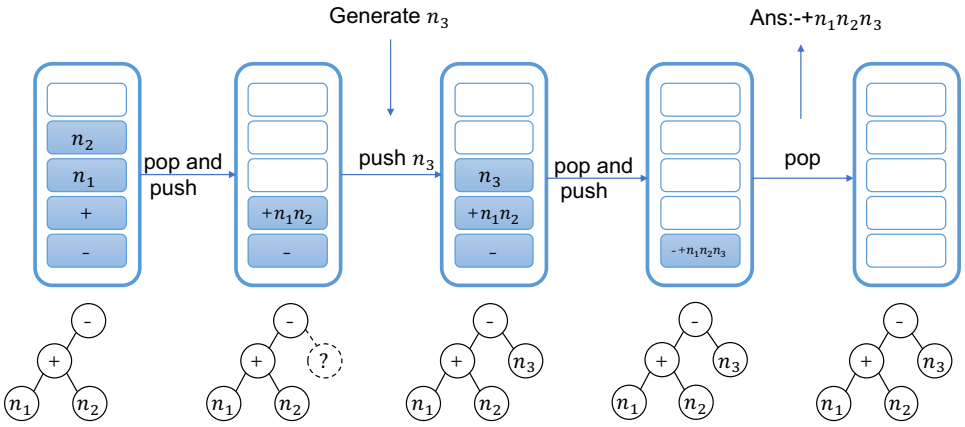


Figure 4: One example of tree-structured decoding with the auxiliary stack.

the orange solid line in Figure 3. This can let the model be informed of the parent node status. For the root node, this part of the input is padded as zeros.

The sibling feeding $e^{sibling}$ refers to using the embedding of the left sibling node as the input when decoding the right sibling node, which is shown as the yellow dotted line in Figure 3. This can let the model be informed whether we are decoding the left or right sibling. For the root node, we use a special token $\langle s \rangle$ for sibling feeding. For the left-most node, we use a special token (for sibling feeding.

The previous token feeding e^{prev} refers to using the previous token in prefix order as the input when decoding the next token, which is shown as the blue dot line in Figure 3. This can let the model be informed of what part is already decoded by the tree. For the root node, we also use the special token $\langle s \rangle$ for previous token feeding.

At time step t, the input e_t^d of the LSTM unit is the concatenation of these three components.

$$e_t^d = (h^{parent}; e^{sibling}; e^{prev}). \quad (2)$$

3.3.2 Tree-Structured LSTM

The tree-structured decoder uses LSTM to generate the equation template in a top-down manner, as the grey solid line in Figure 3, and use an auxiliary stack to guide the decoding process. Given the input e_t^d shown in the previous section, we generate the output token y_t with one LSTM layer and one Multi-layer Perceptron (MLP) layer.

$$h_t^d = LSTM(h_{t-1}^d, e_t^d) \quad (3)$$

$$y_t = argmax(MLP(h_t^d)) \quad (4)$$

Algorithm 1: The algorithm of using the auxiliary stack to guide tree decoding.

Input: Encoder Output h_n
Output: The Predicted Prefix Notation Equation $Equa^p = \{s_i^p\}_{i=1}^{n_2}$

initialize empty stack S and list $Equa^p$;
while *$S.size!=1$ or $S.top$ is not quantity* **do**
  generate token y_t;
  tmp = S.top;
  S.push(y_t);
  if *y_t is quantity* **then**
    while *tmp is a quantity* **do**
      subtree t = S.top[3];
      S.pop;S.pop;S.pop;
      tmp = S.top;S.push(t);
$Equa^p$ = S.pop;

As shown in Algorithm 1 and Figure 3, if the predicted token y_t is an operator, then we predict the left child node of the operator and push this token into the stack S. If the predicted token y_t is a quantity, we check the stack to determine which is the next token that needs to be decoded. If the top of the stack is an operator, then we push y_t into the stack and go on to decode the right sibling node of the current node. If the top of S is a quantity, we follow Algorithm 1 to find the next position to decode. We push y_t to the stack and pop out the top three tokens, which should be $\langle op \rangle \langle num \rangle \langle num \rangle$. These three tokens form a subtree t and we regard this subtree t as one quantity unit in the following process. Then we examine the status of the stack again, if the top of the stack is still a number, we push t back to the stack and continue until the top of the stack is not a quantity. When the loop stops,

if the top of the stack is an operator, we push back t and continue to decode the operator's right child node, and if the stack is empty, the decoding process ends because the AST is completed. Here we still push back the tree unit to the stack and treat the status that only one number is in the stack as the ending condition, which refers to line 2 in Algorithm 1. In this way the condition that the first generated token is the answer number can be unified. With the help of this stack, we can guide the decoding process, including which token to generate next and when to stop naturally without any special tokens.

We show one example of the decoding process with the an auxiliary stack in Figure 4. The upper half part shows the status of the stack, where the solid blocks stand for the inserted tokens. The lower half part shows the status of the AST during decoding, where the solid line stands for the generated nodes and the dotted line stands for the node that should be generated in the next step. The decoder first generates $-$, $+$, n_1 and n_2 and forms a complete subtree in the AST. This subtree $+$, n_1 and n_2 is then popped out of the stack and $+n_1n_2$ are pushed back as one unit. The model then continues to predict the sibling node of the subtree's root node, which is the dotted line circle in Figure 4. The top three tokens of the stack now form a complete subtree again and is popped out of the stack. The stack is now empty, we push it back and the stack only contains one number unit, then the decoding process ends and the equation template is popped out.

3.3.3 Attention Mechanism

An attention mechanism has shown its effectiveness in various natural language processing tasks. We extend the decoder with an attention mechanism by adjusting Equation 4. Instead of directly using the hidden state h_t^d to predict the output token y_t, we consider relevant information from the input vectors to better predict y_t. Formally, given the LSTM hidden state h_t^d and the encoder outputs $\{h_t\}_{t=1}^n$, we calculate the attention weights α_t^i and attention vector s_t as follows:

$$s_t = \sum_{i=1}^{n} \alpha_t^i \cdot h_i = \sum_{i=1}^{n} \frac{exp(h_i \cdot h_t^d)}{\sum_{j=1}^{n} exp(h_j \cdot h_t^d)} \cdot h_i \quad (5)$$

In lieu of Equation 4, we use the attention vector s_t, which considers relevance of the encoder information to predict the output token y_t.

4 Experiments

To demonstrate the effectiveness of our model, we conduct experiments on the Math23K dataset. Our method achieves the state-of-the-art (SOTA) single model performance and also exceeds the previous ensemble model SOTA.

4.1 Dataset

Math23K Math23K is one large-scale Chinese MWP dataset that contains 23,162 math problems and math equation solutions. The questions are elementary school level. Every question is linear and contains only one unknown variable.

Although there are other large-scale datasets such as Dolphin18K and AQuA, which are in English, they either contain many small typos (e.g., using x to represent $*$) or contain wrong answers and templates. Other datasets such as ALG514 and MAWPS are much more smaller. Therefore, we decide to conduct experiments on Math23K, which is the only large-scale, clean and high-quality dataset.

4.2 Implementation Details

The embedding vectors are pretrained on the training set with the word2vec algorithm. The dimension of the embedding is 128. We use a two-layer BiLSTM with the hidden size 512 for the encoder. The decoder is a two-layer LSTM with 1024 hidden size. We use a teacher forcing ratio of 0.83 during training. We use cross entropy as the loss function and Adam to optimize the parameters. We also use dropout to avoid over-fitting. The batch size is 128.

4.3 Results

Table 2 shows the results of our system and other novel systems of MWP on the Math23k test set. The retrieval-style models compare a question in the test set with the questions in the training set, choose the template that has the highest similarity, and then fill in the numbers into the template (Upadhyay and Chang, 2017; Robaidek et al., 2018). The classification-style models train a classifier to select an equation template, and then map the numbers into the template (Kushman et al., 2014). For retrieval and classification models, we use the results of Robaidek et al. (2018). The generation models use end-to-end style encoder-decoder systems to generate an equation template and then fill in the numbers.

Model		Acc
Retrieval	Cosine (Robaidek et al., 2018)	23.8%
	Jaccard (Robaidek et al., 2018)	47.2%
Classification	Self-Attention (Robaidek et al., 2018)	56.8%
	Bi-LSTM (Robaidek et al., 2018)	57.9%
Generation	DNS (Wang et al., 2017)	58.1%
	BiLSTM+Suffix+EN (Wang et al., 2018a)	66.7%
	Semantically-Aligned† (Chiang and Chen, 2018)	65.8%
	T-RNN (Wang et al., 2019)	66.9%
	Ours	**69.0%**
Ensemble	DNS+Retrieval (Wang et al., 2017)	64.7%
	DNS+suffix+EN Ensemble (Wang et al., 2018a)	68.4%
	T-RNN+Retrieval (Wang et al., 2019)	68.7%

Table 2: Math word problem solving accuracy on Math23K. † denotes that the result is 5-fold cross validation performance. All other models are tested on the test set.

Model	Acc
Prefix Baseline	66.8%
+ Sibling Feeding	67.8 %
+ Parent Feeding	68.1 %
Full Model	**69.5%**

Table 3: Ablation study on Math23K by removing modules.

Model	Invalid Templates
EOS as terminator	1.3%
Stack as terminator	**0.2%**

Table 4: Ablation study on Math23K by using different terminating methods.

Wang et al. (2017) proposed a DNS model, which used seq2seq with SNI to generate an equation template. Wang et al. (2018a) proposed a BiLSTM+Suffix+EN model, which extends the DNS model by decoding the suffix notation and applies equation normalization. Chiang and Chen (2018) introduced a copy mechanism to generate equation templates in a semantically-aligned manner. T-RNN (Wang et al., 2019) extends the BiLSTM+Suffix+EN model by first generating a template with unknown operators and then filling in the operators with a Recursive Neural Network. The ensemble models use bagging to combine the results of different models.

Our seq2tree model is also a generation-style model. As shown in Table 2, we achieve state-of-the-art single model performance on the test set, and even better results than all the previous ensemble models, which can demonstrate the effectiveness of our proposed method.

5 Analysis and Discussion

5.1 Ablation Study

To get better insight into our seq2tree system, we conduct ablation study on Math23K development set, which is shown in Table 3. The prefix baseline denotes the model that removes parent feeding and sibling feeding, but only uses previous token feeding for the input. Thus, this model loses parent and sibling information and falls into a linear seq2seq model based on the prefix notation. The prefix baseline performs competitive results compared to previous single model SOTA (66.9%), which proves the effectiveness of top-down decoding. Parent feeding and sibling feeding separately improve the baseline model by 1.1% and 1.0%, demonstrating the importance of informing the model of AST structure information.

In Table 4, we also report the percentage of invalid templates by using different terminating methods. We remove the auxiliary stack and use a special end-of-sentence (EOS) token, which was used in previous studies to terminate the decoding process (Wang et al., 2017, 2018a). We can see that using the stack as a terminator can let the model generate very low percentage of invalid templates and outperforms the EOS method.

Problem	A person is taking a trip from A to B. He took a train for n_1 of the trip the first day. He took a bus and travelled for n_2 km the second day. He still needs to travel for n_3 of the total distance. How far is it from A to B?
Gold	**suffix order**: x = n_1 1 n_2 - n_3 - / **prefix order**: x = / n_1 - - 1 n_2 n_3
Prediction	**BiLSTM+Suffix+EN**: n_1 n_2 - n_3 / (error) **Ours**: / n_1 - - 1 n_2 n_3 (correct)

Figure 5: One example of our system compared with BiLSTM+Suffix+EN (Wang et al., 2018a).

#Operators	Proportion(%)	Acc(%)
0	0.1	100.0
1	17.3	82.7
2	52.2	74.5
3	19.1	59.9
4	6.6	42.4
5	3.4	44.1
6	0.9	55.6
7	0.3	0
8	0	0
9	0.1	100.0

Table 5: Accuracy of different template lengths on Math23K.

Domain	Proportion(%)	Acc(%)
Distance & Speed	20.5	70.2
Tracing	27.0	74.1
Engineering	5.4	64.1
Interval	0.2	50.0
Circle Geometry	1.5	33.3
Plane Geometry	1.1	81.8
Profit	0.5	40.0
Solid Geometry	1.3	46.2
Interest Rate	0.5	80.0
Production	0.4	100.0

Table 6: Accuracy of different question domains on Math23K.

5.2 Case Study

Here we give an example that is improved by our tree-structured decoding system. As shown in Figure 5, in the gold equation of this example, there is a long distance between two pairs of parent-child nodes n_1 and $/$ and also 1 and $-$. The BiLSTM+Suffix+EN model failed to capture the relationship between these two pairs of parent-child nodes and caused an error. Our model has a better ability to capture the relation between pairs of parent-child nodes even if there is a long distance between them in the notation.

5.3 Error Analysis

In Table 5, we show the results of how the accuracy changes as the template becomes longer. We can see that our model's performance becomes very low when the equation becomes longer. This shows that our model has limitations at predicting long templates. This is because longer templates often match with more complex questions which are more difficult to solve. Thus, this model still has room for improvement on reasoning, inference and semantic understanding. Meanwhile, only a few examples in Math23K match with complex templates, so introducing data augmentation techniques or constructing a new dataset with more complex examples may further improve the model's performance.

In Table 6, we examine the performance of the model in different question domains. MWPs in the same domain usually share similar logic, while there is an obvious difference between questions across different domains. Accurately detecting the question domain is very laborious, so we do this experiment by simply detecting frequent keywords of each domain in the question. We show further details in the appendix. The results show that the performance of the model has obvious variance among different domains and limitations in some domains such as solid geometry. This is because these domains require complicated external knowledge for solving these questions, such as $S_{circle} = \pi r^2$. It is difficult for the model to automatically summarize these kinds of information with only supervision of the equation templates. Adding external knowledge for this task may further improve the model.

6 Conclusion

We proposed a sequence-to-tree generative model to improve template generation for solving math word problems. The hierarchical top-down tree-structure decoder can use the information of the abstract syntax tree of an equation during decoding. With the help of an auxiliary stack, this

decoding process can end without any redundant special tokens. Our model achieves state-of-the-art results on the large-scale dataset, Math23k, demonstrating the effectiveness of our approach.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number JP18H03286.

References

David Alvarez-Melis and Tommi S. Jaakkola. 2017. Tree-structured decoding with doubly-recurrent neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Ting-Rui Chiang and Yun-Nung Chen. 2018. Semantically-aligned equation generation for solving and reasoning math word problems. *arXiv preprint arXiv:1811.00720*.

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)*, volume 1, pages 33–43.

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, 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.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 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, pages 887–896.

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)*, volume 1, pages 271–281.

Anirban Mukherjee and Utpal Garain. 2008. A review of methods for automatic understanding of natural language mathematical problems. *Artificial Intelligence Review*, 29(2):93–122.

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.

Benjamin Robaidek, Rik Koncel-Kedziorski, and Hannaneh Hajishirzi. 2018. Data-driven methods for solving algebra word problems. *arXiv preprint arXiv:1804.10718*.

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

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

Subhro Roy and Dan Roth. 2018. Mapping to declarative knowledge for word problem solving. *Transactions of the Association of 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.

Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2018. A grammar-based structural cnn decoder for code generation. *arXiv preprint arXiv:1811.06837*.

Shyam Upadhyay and Ming-Wei Chang. 2017. Annotating derivations: A new evaluation strategy and dataset for algebra word problems. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 494–504.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to a expression tree. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1064–1069.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018b. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bingtian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks.

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.

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.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2016. Top-down tree long short-term memory networks. In *Proceedings of NAACL-HLT*, pages 310–320.

A Prefix Notation Algorithm

Algorithm 2: The algorithm of converting ordinary equations to its prefix notation.

Input: The Ordinary Equation String $Equa = \{s_i\}_{i=1}^{n_1}$

Output: The Prefix Notation Equation String $Equa^p = \{s_i^p\}_{i=1}^{n_2}$

initialize empty stack S and list $Equa_{inv}^p$;
for $i \leftarrow n_1$ ***to*** 1 **do**
  if s_i *is quantity* **then**
    $Equa_{inv}^p$.append(s_i);
  else if s_i *is*) **then**
    S.push(s_i);
  else if s_i *is operator* **then**
    while *True* **do**
      if s_i *is prior to* S.*top or* S.*top is*) *or* S *is empty* **then**
        S.push(s_i);
        break;
      else
        $Equa_{inv}^p$.append(S.top);
        S.pop();
  else if S.*top is (* **then**
    while S.*top is not)* **do**
      $Equa_{inv}^p$.append(S.top);
      S.pop();
    S.pop()
$Equa^p$ = $Equa_{inv}^p$.inverse()

Here we show the details of the converting algorithm. The input is the ordinary equation string and the output is the prefix notation of the equation.

B Domain Keywords

We show the table of the keywords for each domain. These keywords were manually collected by observation.

Domain	Keywords
Distance & Speed	速 度,千 米,路 程,相距,全程
Tracing	相 遇,相 对,相 反,相背,相向
Engineering	工程,零件,工程队,公路,修路
Interval	利润
Circle	半径,圆,直径,周长
Plane Geometry	三角形,正方形,长方形,边长
Profit	间隔,隔
Solid Geometry	体 积,侧 面 积,横 截面,表面积,圆柱,长方体
Interest Rate	利息,利率
Production	超产,减产

Table 7: The list of keywords in the domain specific studies.