Solving Math Word Problems with Multi-Encoders and Multi-Decoders

Yibin Shen
East China Normal University
ybshen@stu.ecnu.edu.cn

Cheqing Jin✉
East China Normal University
cqjin@dase.ecnu.edu.cn

Abstract

Math word problems solving remains a challenging task where potential semantic and mathematical logic need to be mined from natural language. Although previous researches employ the Seq2Seq technique to transform text descriptions into equation expressions, most of them achieve inferior performance due to insufficient consideration in the design of encoder and decoder. Specifically, these models only consider input/output objects as sequences, ignoring the important structural information contained in text descriptions and equation expressions. To overcome those defects, a model with multi-encoders and multi-decoders is proposed in this paper, which combines sequence-based encoder and graph-based encoder to enhance the representation of text descriptions, and generates different equation expressions via sequence-based decoder and tree-based decoder. Experimental results on the dataset Math23K show that our model outperforms existing state-of-the-art methods.

1 Introduction

Math word problems (MWPs) solving, a task that transforms text descriptions into solvable equation expressions, is considered a crucial step towards general AI (Wang et al., 2018b). Since semantic understanding and mathematical logic reasoning both contribute to correct answers, MWPs solving remains a challenging topic in NLP. Table 1 shows a typical example of MWPs.

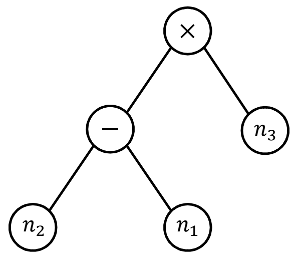
Problem:	A slow car drives $58(n_1)$ km/h, and a fast car drives $85(n_2)$ km/h. The two cars drive at the same time in inverse direction, and they meet after $5(n_3)$ hours. How many kilometers does the fast car drive more than the slow car when they meet?	**AST**: 
Equation:	$(n_2 - n_1) \times n_3$	
Prefix:	$\times - n_2 n_1 n_3$	
Suffix:	$n_2 n_1 - n_3 \times$	
Answer:	135	

Table 1: A typical example of MWPs.

Researches on MWPs solving has a long history. Early researches focused on rule-based methods (Fletcher, 1985; Bakman, 2007; Yuhui et al., 2010) and statistical machine learning methods (Kushman et al., 2014; Hosseini et al., 2014; Mitra and Baral, 2016) that map problems into predefined templates. The main drawbacks of these methods lie in their heavy dependency on manual features and incapacity to generate new templates for new problems. Consequently, they can only achieve satisfactory results on small-scale datasets (Zhang et al., 2018).

Recently, more researchers have been introducing Seq2Seq models, which are capable of generating new equation expressions that do not exist in the training set (Wang et al., 2017; Wang et al., 2018a; Wang et al., 2019; Li et al., 2019). However, these models may generate invalid expressions since the sequence-based decoder cannot control the generation process. Based on the fact that each equation expression could be transformed into an abstract syntax tree (AST), some studies (Liu et al., 2019; Xie and Sun, 2019) changed the pattern of sequence generation from left to right and followed the top-down decoding process. Such tree-based decoders match the prefix order of AST. Although these models considered the structural information of equation expressions, they ignored that text descriptions also contain rich structural information, such as dependency parse tree and numerical comparison information.

The dependency parse tree represents various grammatical relationships between pairs of text words, for example, nouns are usually matched with verbs, and numerals are usually matched with quantifiers. In Table 1, n_1 can be subtracted from n_2 because n_1 and n_2 have the same quantifiers. Therefore, considering the dependency parse tree can reduce the situation of unreasonable operators between number pairs. In addition, most of MWPs solving replace numbers with special tokens (i.e. n_1, n_2), which loses important numerical comparison information contained in text descriptions. For example, in Table 1, the underlined words 'slow car' and 'fast car' imply the fact that '$n_1 < n_2$'. Similarly, we incline to ask 'How many kilometers does the fast car drive more than the slow car?' rather than 'How many kilometers does the slow car drive more than the fast car?'. In other words, text descriptions match the numerical comparison information. Provided that a model knows numerical comparison information in advance, the model can better understand potential semantic without wasting a lot of time in mining these established facts from a large number of corpus.

Now back to the design of decoder, existing methods only adopt one decoder, which limits the generation ability of the model. (Wang et al., 2018a) provided an ensemble model that selects the result according to various models' generation probability. However, there is still one decoder for a single model. (Meng and Rumshisky, 2019) integrated two decoders in one model, but both are sequence-based decoders. The same type of decoder cannot significantly improve generalization performance.

With the aim of solving aforementioned challenges, we propose a novel model with multi-encoders and multi-decoders, which combines sequence-based encoder and graph-based encoder to enhance the representation of text descriptions, and obtains different equation expressions via sequence-based decoder and tree-based decoder. Specifically, we leverage a sequence-based encoder to get the context representation of text descriptions, and integrate the dependency parse tree and numerical comparison information via a graph-based encoder. In the decoding stage, a sequence-based decoder is used to generate the suffix order of AST, and a tree-based decoder is used to generate the prefix order. The final result is selected according to the generation probability of different decoders. The main contributions of this paper are summarized as follows:

- We integrate the dependency parse tree and numerical comparison information in the model, which enhances the representation of text descriptions.
- We use two types of decoders to generate different equation expressions, which strengthens the generation ability of the model.
- We evaluate our model on a large-scale dataset Math23K. The experimental results show that our model outperforms all existing state-of-the-art methods.

2 Related Work

MWPs solving may date back to the 1960s and continues attracting current NLP researchers. Here we will introduce recent studies based on the Seq2Seq framework. The work presented in (Zhang et al., 2018) reviews more early approaches.

(Wang et al., 2017) made the first attempt to directly generate equation expressions by using the Seq2Seq model and published a high-quality Chinese dataset Math23K. (Wang et al., 2018a) found that using the suffix order of AST can eliminate brackets in the original expressions, and proposed an equation normalization method to reduce the number of duplicated equations. (Wang et al., 2019) proposed a two-stage model that first used a Seq2Seq model to generate expressions without operators, and then used a recursive neural network to predict the operator between numbers. (Chiang and Chen, 2019) adopted a stack to track the semantic meanings of numbers. (Li et al., 2019) added different functional multi-head attentions to the Seq2Seq framework. (Meng and Rumshisky, 2019) applied double sequence-based decoders in one model. However, these Seq2Seq models only consider input/output objects as sequences, ignoring the important structural information of equation expressions. Consequently, they cannot guarantee the generation of valid equation expressions.

The idea of the tree-based decoder was proposed in (Liu et al., 2019; Xie and Sun, 2019). They changed the pattern of sequence generation from left to right and followed the top-down decoding process. However, these methods ignored rich structural information contained in text descriptions.

(Li et al., 2020; Zhang et al., 2020) proposed the graph-based encoder. (Li et al., 2020) integrated the dependency parse tree and constituency tree of text descriptions. (Zhang et al., 2020) constructed the quantity cell graph and quantity comparison graph. Since these methods considered the structure information of text descriptions, they have been current state-of-the-art models.

The encoders and decoders designed by these Seq2Seq models are summarized in Table 2. As we can see, our model is the first model to adopt multi-encoders and multi-decoders.

Model	**Seq-Encoder**	**Graph-Encoder**	**Seq-Decoder**	**Tree-Decoder**
DNS (Wang et al., 2017)	✓		✓	
Math-EN (Wang et al., 2018a)	✓		✓	
T-RNN(Wang et al., 2019)	✓		✓	
S-Aligned (Chiang and Chen, 2019)	✓		✓	
Group-ATT (Li et al., 2019)	✓		✓	
D-Decoder (Meng and Rumshisky, 2019)	✓		✓	
AST-Dec (Liu et al., 2019)	✓			✓
GTS (Xie and Sun, 2019)	✓			✓
Graph2Tree (Li et al., 2020)	✓	✓		✓
Graph2Tree (Zhang et al., 2020)	✓	✓		✓
Ours	✓	✓	✓	✓

Table 2: The encoders and decoders designed by various Seq2Seq models.

3 Methodology

The framework of our model is shown in Figure 1, which consists of four components: the sequence-based encoder obtains the context representation of text descriptions; the graph-based encoder integrates the dependency parse tree and numerical comparison information; the sequence-based decoder generates the suffix order of AST, and the tree-based decoder generates the prefix order. The final generation result is selected according to the generation probability of different decoders.

3.1 Sequence-Based Encoder

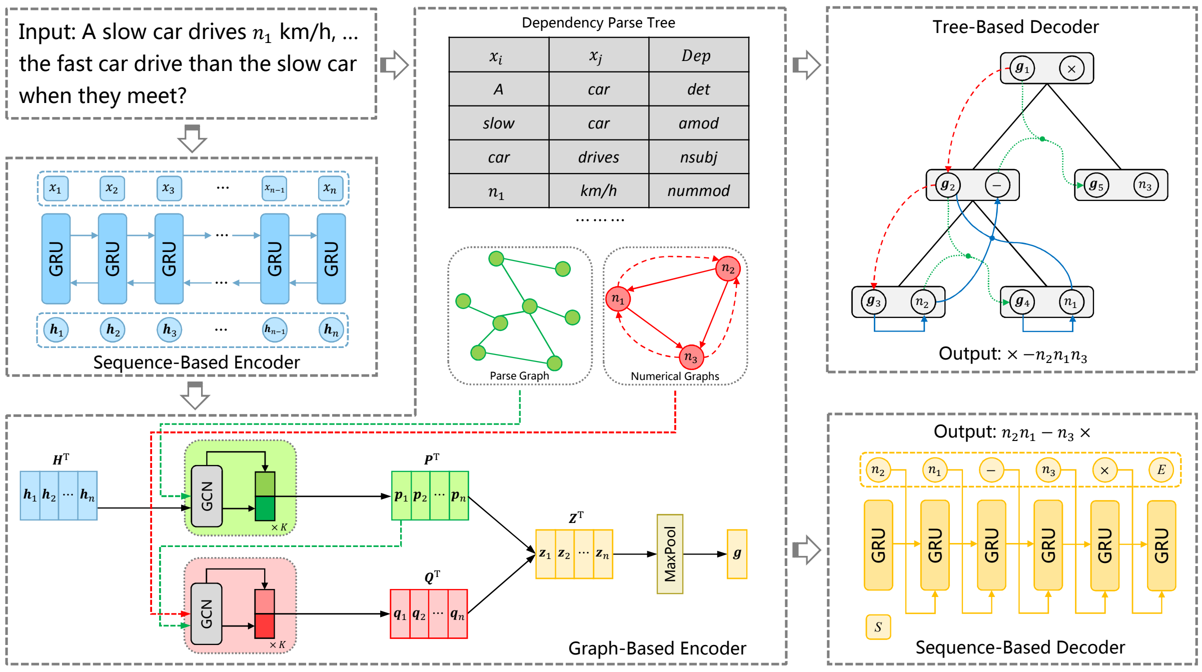
The goal of sequence-based encoder is to get the context representation of text descriptions. Without loss of generality, we use a BiGRU to encode text words. Formally, given the text words $P = \{x_1, \cdots, x_n\}$, we first embed each word token x_i to a word embedding vector $\boldsymbol{e}_i$[1], and then feed these embedding vectors into a BiGRU to produce hidden state sequences $\boldsymbol{H} = \{\boldsymbol{h}_1, \cdots, \boldsymbol{h}_n\}$.

3.2 Graph-Based Encoder

3.2.1 Dependency Parse Tree

As is discussed in Section 1, the dependency parse tree represents various grammatical relationships between pairs of text words, which is helpful to find reasonable operators between number pairs. We can

[1]For each word token, we also embed its POS tagging.



x_i	x_j	Dep
A	car	det
slow	car	amod
car	drives	nsubj
n_1	km/h	nummod

……

Figure 1: The framework of our model. We first exploit a sequence-based encoder to obtain the context representation of text descriptions. Later, a graph-based encoder is used to integrate the dependency parse tree and numerical comparison information. In the decoding process, the sequence-based decoder and tree-based decoder generate different equation expressions.

easily obtain the graph-based structure of the dependency parse tree by using dependency relationships in the parse tree. Hence, we consider the following parse graph.

- **Parse Graph** ($\mathcal{G}$): For two words $x_i, x_j \in P$, there is an edge $e_{ij} = (x_i, y_j) \in \mathcal{G}$ if the pair has dependency relationship in the dependency parse tree, referring to the table in Figure 1.

Note that the parse graph is an undirected graph. After building the graph-based structure of the dependency parse tree, we need to find an effective way to learn the graph representation. Here we introduce GraphSAGE (Hamilton et al., 2017), which is a flexible graph neural network. Specifically, we first use the sequence $\boldsymbol{H} = \{\boldsymbol{h_1}, \cdots, \boldsymbol{h_n}\}$ obtained by the sequence-based encoder as the initial embedding of each node. Then each node updates its embedding vector from neighborhood nodes, which can be expressed as

$$\boldsymbol{P}_N^k = \text{GCN}(\boldsymbol{P}^{k-1}, \mathcal{G}) = \text{ReLU}(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \boldsymbol{P}^{k-1} \boldsymbol{W}) \tag{1}$$

$$\boldsymbol{P}^k = \text{ReLU}([\boldsymbol{P}^{k-1}; \boldsymbol{P}_N^k] \cdot \boldsymbol{W}_P) \tag{2}$$

where $\boldsymbol{P}_N^k$ denotes the aggregating information from neighborhood nodes, $\boldsymbol{P}^k$ denotes the updated embedding of each node and $\boldsymbol{P}^0 = \boldsymbol{H}$. $\widetilde{\boldsymbol{D}} = \boldsymbol{D} + \boldsymbol{L}$, $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{L}$, $\boldsymbol{D}$ represents the degree matrix and $\boldsymbol{A}$ represents the adjacency matrix of the parse graph. $k \in \{1, \cdots, K\}$ is the iteration index and $\{\boldsymbol{W}, \boldsymbol{W}_P\}$ are parameter matrices.

3.2.2 Numerical Comparison Information

Numerical comparison information also plays an important role in enhancing text descriptions. We also use a graph-based structure to represent the numerical comparison information. We denote the numbers in the text words as $V_n = \{n_1, \cdots, n_l\}$ and consider the following two types of numerical graphs.

- **Greater Graph** ($\mathcal{G}_g$): For two numbers $n_i, n_j \in V_n$, there is an edge $e_{ij} = (n_i, n_j) \in \mathcal{G}_g$ if $n_i > n_j$, referring to the red solid lines in Figure 1.

- **Lower Graph** ($\mathcal{G}_l$): For two numbers $n_i, n_j \in V_n$, there is an edge $e_{ij} = (n_i, n_j) \in \mathcal{G}_l$ if $n_i \leq n_j$, referring to the red dashed lines in Figure 1.

Unlike the parse graph, there are two types of numerical graphs and they are directed graphs. Hence we extend GraphSAGE to fit the integration of numerical comparison information. The updating rule of each number can be expressed as

$$\boldsymbol{Q}^k_{N_g} = \text{GCN}(\boldsymbol{Q}^{k-1}, \mathcal{G}_g) = \text{ReLU}(\widetilde{\boldsymbol{D}}_g^{-1}\widetilde{\boldsymbol{A}}_g\boldsymbol{Q}^{k-1}\boldsymbol{W}_g) \tag{3}$$

$$\boldsymbol{Q}^k_{N_l} = \text{GCN}(\boldsymbol{Q}^{k-1}, \mathcal{G}_l) = \text{ReLU}(\widetilde{\boldsymbol{D}}_l^{-1}\widetilde{\boldsymbol{A}}_l\boldsymbol{Q}^{k-1}\boldsymbol{W}_l) \tag{4}$$

$$\boldsymbol{Q}^k_N = \boldsymbol{M}_a * \boldsymbol{Q}^k_{N_g} + (\boldsymbol{1} - \boldsymbol{M}_a) * \boldsymbol{Q}^k_{N_l} \tag{5}$$

$$\boldsymbol{M}_a = \sigma([\boldsymbol{Q}^k_{N_g}; \boldsymbol{Q}^k_{N_l}; \boldsymbol{Q}^k_{N_g} + \boldsymbol{Q}^k_{N_l}; \boldsymbol{Q}^k_{N_g} - \boldsymbol{Q}^k_{N_l}] \cdot \boldsymbol{W}_a) \tag{6}$$

$$\boldsymbol{Q}^k = \text{ReLU}([\boldsymbol{Q}^{k-1}; \boldsymbol{Q}^k_N] \cdot \boldsymbol{W}_Q) \tag{7}$$

where $\{\boldsymbol{Q}^k_{N_g}, \boldsymbol{Q}^k_{N_l}\}$ represent the aggregating information from neighborhood nodes in two graphs, $\boldsymbol{Q}^k$ represents the updated embedding of each node and $\boldsymbol{Q}^0 = \boldsymbol{P}^K$. $\boldsymbol{M}_a$ controls the weight of two graphs, '$*$' denotes element-wise multiplication and 'σ' denotes the 'Sigmoid' function. $k \in \{1, \cdots, K\}$ is the iteration index and $\{\boldsymbol{W}_g, \boldsymbol{W}_l, \boldsymbol{W}_a, \boldsymbol{W}_Q\}$ are parameter matrices.

The final encoder vectors of text descriptions incorporate the node embedding vectors in the parse graph and numerical graphs, which can be calculated as

$$\boldsymbol{Z} = \boldsymbol{P}^K + \boldsymbol{Q}^K \tag{8}$$

$$\boldsymbol{g} = \text{MaxPool}(\boldsymbol{Z}) \tag{9}$$

where $\boldsymbol{Z} = \{\boldsymbol{z}_1, \cdots, \boldsymbol{z}_n\}$ denotes the final encoder vectors of each word, and $\boldsymbol{g}$ represents the global vector of text descriptions for further decoding.

3.3 Sequence-Based Decoder

The sequence-based decoder is used to generate the suffix order of AST. We use a GRU with attention layer to generate the sequence, which can be expressed as

$$\boldsymbol{s}_i = \text{GRU}(\hat{y}_{i-1}, \boldsymbol{s}_{i-1}, \boldsymbol{c}_i) \tag{10}$$

$$\boldsymbol{c}_i = \sum_{j=1}^{n} \alpha_{ij}\boldsymbol{z}_j \tag{11}$$

$$\alpha_{ij} = \frac{\exp(\text{score}(\boldsymbol{s}_{i-1}, \boldsymbol{z}_j))}{\sum_{j=1}^{n}\exp(\text{score}(\boldsymbol{s}_{i-1}, \boldsymbol{z}_j))} \tag{12}$$

$$\text{score}(\boldsymbol{s}_{i-1}, \boldsymbol{z}_j) = \boldsymbol{v}_s^{\text{T}} \cdot \tanh(\boldsymbol{W}_s \cdot [\boldsymbol{s}_{i-1}; \boldsymbol{z}_j]) \tag{13}$$

where $\boldsymbol{s}_i$ denotes the hidden state vector of the decoder, $\boldsymbol{c}_i$ denotes the context vector. α_{ij} controls the attention weight of every encoder vector. $\hat{y}_i$ is the output and $\{\boldsymbol{v}_s, \boldsymbol{W}_s\}$ are parameter matrices.

3.4 Tree-Based Decoder

The tree-based decoder is used to generate the prefix order of AST. We follow the Goal-driven Tree Structure (GTS) proposed in (Xie and Sun, 2019), which not only realized the top-down decoding process but also used the bottom-up subtree embedding manners. Here we simply introduce the decoding process, which can be expressed as

- **Step 1 (Root Goal Generation)**: GTS followed the pre-order traversal manner, so the primary goal is to generate the root node. We use $\boldsymbol{g}$ as the initial goal vector of the root node, and apply the same attention mechanism in the sequence-based decoder to get the context vector $\widetilde{\boldsymbol{c}}_1$.

$$\widetilde{\boldsymbol{c}}_1 = \text{Attention}(\boldsymbol{g}, \boldsymbol{Z}) \tag{14}$$

$$\hat{y}_1 = \text{Predict}(\boldsymbol{g}, \widetilde{\boldsymbol{c}}_1) \tag{15}$$

  Note that the algorithm terminates directly if $\hat{y}_1$ is a number; otherwise, we will go to step 2.

- **Step 2 (Left Goal Generation)**: The left goal $\boldsymbol{g}_l$ is generated according to the goal vector and the predicted token of its parent node, which can be expressed as

$$\boldsymbol{g}_l = \text{Left}(\hat{y}_p, \boldsymbol{g}_p, \widetilde{\boldsymbol{c}}_p) \tag{16}$$

$$\hat{y}_l = \text{Predict}(\boldsymbol{g}_l, \widetilde{\boldsymbol{c}}_l) \tag{17}$$

  where $\hat{y}_p, \boldsymbol{g}_p$ and $\widetilde{\boldsymbol{c}}_p$ stand for the predicted token, goal vector and content vector of the parent node respectively. The process of generating the left goal continues until $\hat{y}_l$ is a number, referring to the red dashed lines in Figure 1. Later, we will go to step 3.

- **Step 3 (Right Goal Generation)**: When the right goal node is being generated, its left sibling node has been completed. Therefore, GTS considered the subtree embedding of its sibling node to generate the right goal $\boldsymbol{g}_r$, which can be expressed as

$$\boldsymbol{g}_r = \text{Right}(\hat{y}_p, \boldsymbol{g}_p, \widetilde{\boldsymbol{c}}_p, \boldsymbol{t}_l) \tag{18}$$

$$\boldsymbol{t}_l = \text{SubTree}(\hat{y}_l, \boldsymbol{g}_l) \tag{19}$$

$$\hat{y}_r = \text{Predict}(\boldsymbol{g}_r, \widetilde{\boldsymbol{c}}_r) \tag{20}$$

  Here, $\boldsymbol{t}_l$ is the tree embedding of the left goal, as illustrated by the blue solid lines in Figure 1. Similarly, we will go back to step 2 if $\hat{y}_r$ is an operator. The algorithm backtracks to check whether there are right goals in the tree that need to be generated if $\hat{y}_r$ is a number. When the model cannot find any generation goal, the algorithm terminates; otherwise, we will continue step 3.

3.5 Model Training

Since our model integrates two types of decoders, we combine the loss functions of the sequence-based decoder and tree-based decoder. For each sample of problem-expression (P, T), the optimization objective of our model is defined as

$$L = -\frac{1}{m}\sum_{i=1}^{m}(\log p(y_i|\boldsymbol{s}_i, \boldsymbol{c}_i, T_s) + \log p(y_i|\boldsymbol{g}_i, \widetilde{\boldsymbol{c}}_i, T_t)) \tag{21}$$

where

$$p(y_i|\boldsymbol{s}_i, \boldsymbol{c}_i, T_s) = \text{softmax}(\boldsymbol{W}_1 \cdot \tanh(\boldsymbol{W}_2 \cdot [\boldsymbol{s}_i; \boldsymbol{c}_i])) \tag{22}$$

$$p(y_i|\boldsymbol{g}_i, \widetilde{\boldsymbol{c}}_i, T_t) = \text{softmax}(\boldsymbol{W}_3 \cdot \tanh(\boldsymbol{W}_4 \cdot [\boldsymbol{g}_i; \widetilde{\boldsymbol{c}}_i])) \tag{23}$$

where m denotes the number of tokens in the equation expression. T_s represents the suffix order and T_t represents the prefix order. $\{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3, \boldsymbol{W}_4\}$ are parameter matrices.

Finally, we use the log probability scores to perform a beam search. After obtaining the top equation expression from double decoders respectively, we select the one with a higher score as the final result.

4 Experiments

In this section, we evaluate our model on a large-scale dataset Math23K. We compare our model with several state-of-the-art methods and demonstrate the effectiveness of our model via a series of controlled experiments. Our code can be downloaded at `https://github.com/YibinShen/MultiMath`.

4.1 Experimental Setup

4.1.1 Dataset

Math23K (Wang et al., 2017): Math23K is a large-scale Chinese dataset that contains 23,162 elementary school level MWPs and corresponding equation expressions and answers. Although there are other large-scale datasets, such as Dolphin18K (Huang et al., 2016) (with 18,460 MWPs) and AQuA (Ling et al., 2017) (with 100,000 MWPs), they contain either some unlabeled problems or informal equation expressions (mixed with texts). Therefore, Math23K is still the most ideal large-scale and high-quality publish dataset.

4.1.2 Hyperparameters

In the sequence-based encoder, we use a two-layer BiGRU with 512 hidden units as the encoder, and the dimension of word embedding is set as 128. In the graph-based encoder, we set the number of iteration steps as $K = 2$. We also use a two-layer GRU with 512 hidden units as the decoder in the sequence-based decoder. The hyper-parameters of the tree-based decoder are consistent with GTS. As to the optimizer, we use Adam with an initial learning rate at 0.001, and the learning rate will be halved every 20 epochs. The number of epochs, batch size and dropout rate are set 80, 64 and 0.5 respectively. At last, we use a beam search with beam size 5 in the sequence-based decoder and tree-based decoder. Our model is implemented in PyTorch 1.4.0 and runs on a server with one NVIDIA Tesla V100. We use pyltp 0.2.1 to preform dependency parsing and POS tagging.

4.1.3 Metric

Since a math word problem can be solved by multiple equation expressions, we use the answer accuracy as the evaluation metric. For Math23K, some of the previous studies were evaluated on the publish test set, while others used the 5-fold cross-validation. We evaluate our model on the two situations.

4.1.4 Baselines

We compare our model with some state-of-the-art methods, including: **DNS** (Wang et al., 2017) made the first attempt to solve MWPs by using a Seq2Seq model. **Math-EN** (Wang et al., 2018a) proposed an equation normalization method to reduce the number of duplicated equations. **T-RNN** (Wang et al., 2019) used a two-stage model to generate expressions. **S-Aligned** (Chiang and Chen, 2019) adopted a stack to track the semantic meanings of numbers. **Group-ATT** (Li et al., 2019) added different functional multi-head attentions to the Seq2Seq framework. **AST-Dec** (Liu et al., 2019) used TreeLSTM to realize top-down decoding process. **GTS** (Xie and Sun, 2019) followed goal-driven tree structure. **Graph2Tree** (Zhang et al., 2020) integrated the quantity cell graph and quantity comparison graph.

4.2 Experimental Results

Type	**Model**	**Math23K(%)**	**Math23K*(%)**
Seq2Seq	DNS	-	58.1
	Math-EN	66.7	-
	T-RNN	66.9	-
	S-Aligned	-	65.8
	Group-ATT	69.5	66.9
Seq2Tree	AST-Dec	69.0	-
	GTS	75.6	74.3
Graph2Tree	Graph2Tree	77.4	75.5
Multi-E/D	Ours	**78.4**	**76.9**

Table 3: Performance comparison on Math23K. Note that Math23K denotes results on public test set and Math23K* denotes the 5-fold cross-validation.

Table 3 depicts the performance comparison of different models on Math23K. As we can see, Seq2Seq models cannot exceed 70% accuracy because they ignored the structural information of text descriptions and equation expressions. Seq2Tree models made full use of tree-based structure expressions and followed the top-down decoding process, which outperforms most of Seq2Seq models. In particular, GTS also realized bottom-up subtree embedding manners and have a good performance on Math23K. Graph2Tree considered the structure information of text descriptions, integrating the quantity cell graph and quantity comparison graph, so it achieves sub-optimal performance in all models. As to our model, our model not only uses multi-encoders to integrate the structural information of the dependency parse tree and numerical comparison graphs, but also enhances the generation ability of the model via multi-decoders, which outperforms aforementioned models.

4.3 Experimental Analysis

In Table 4, we show the accuracy of the top-5 most frequent expressions on Math23K*. Intuitively, our model achieves more than 90% accuracy in all situations and outperforms the other two models in most cases. Note that our model has a significant improvement over GTS under expressions with '$\div$' or '$-$'. This is because the division and subtraction operators do not meet the commutative law, which requires the model to learn the correct arithmetic order. Since GTS doesn't integrate the numerical comparison information in the model, it cannot deal with these expressions well.

Expression (prefix)	**Pro(%)**	**GTS(%)**	**Graph2Tree(%)**	**Ours(%)**
$\times n_1 n_2$	4.77	89.05	89.05	**90.23**
$\div n_1 n_2$	4.40	88.61	90.37	**91.85**
$\div n_2 n_1$	3.43	86.40	88.16	**90.81**
$\times n_1 - 1 n_2$	2.31	89.55	90.67	**91.23**
$\div \times n_1 n_2 n_3$	2.27	90.49	**92.40**	92.21

Table 4: Accuracy of the top-5 most frequent expressions on Math23K*.

Figure 2 depicts the accuracy of different expression lengths. The gray line represents the proportion of different expression lengths. Results show that our model outperforms GTS and Graph2Tree in all situations. However, the performance of our model has a rapid drop when the expression becomes longer. There are two reasons for this phenomenon: (1) longer expressions contain more operators, and the neural network cannot save the results of intermediate variables well; (2) longer expressions only account for a small part of the dataset (e.g. each expression longer than 9 can be matched to 1.67 problems on average), and the model lacks samples for training. In future work, we will consider question generation technology to generate more MWPs, which may solve this problem.

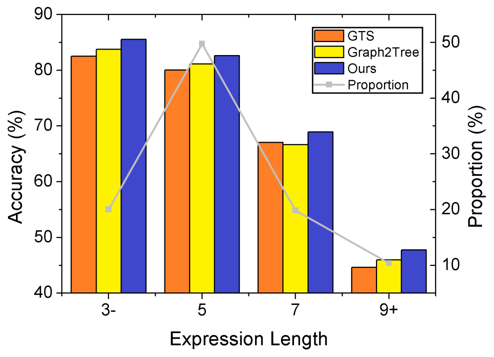


Figure 2: Accuracy of different expression lengths on Math23K*.

4.4 Case Study

To demonstrate the effectiveness of our model, we conduct a case study in Table 5. **Test 1** exchanges the order of text descriptions and **Test 2** changes the form of question description. These two simple tests are used to investigate whether the model can mine the correct mathematical logic from natural language.

In the original problem, GTS obtains a negative answer, which conflicts with the problem. It is funny that GTS obtains the correct answer when we change the order of text description. Note that GTS generates the same expression in **Test 1**, which implies that GTS only remembers the order of the numbers instead of the real mathematical logic within the problem.

Problem:	A slow car drives $58(n_1)$ km/h, and a fast car drives $85(n_2)$ km/h. The two cars drive at the same time in inverse direction, and they meet after $5(n_3)$ hours. How many kilometers does the fast car drive more than the slow car when they meet?
Result:	**GTS**: $\times - n_1 n_2 n_3 = -135$ (**error**) **Ours**: $\times - n_2 n_1 n_3 = 135$ (**correct**)
Test 1:	**A fast car drives $85(n_1)$ km/h, and a slow car drives $58(n_2)$ km/h**. The two cars drive at the same time in inverse direction, and they meet after $5(n_3)$ hours. How many kilometers does the fast car drive more than the slow car when they meet?
Result:	**GTS**: $\times - n_1 n_2 n_3 = 135$ (**correct**) **Ours**: $\times - n_1 n_2 n_3 = 135$ (**correct**)
Test 2:	A slow car drives $58(n_1)$ km/h, and a fast car drives $85(n_2)$ km/h. The two cars drive at the same time in inverse direction, and they meet after $5(n_3)$ hours. **How many kilometers does the slow car drive less than the fast car when they meet?**
Result:	**GTS**: $\times - n_1 n_2 n_3 = -135$ (**error**) **Ours**: $\times - n_2 n_1 n_3 = 135$ (**correct**)

Table 5: Case study of MWPs solving, where **Test 1** and **Test 2** are generative cases.

In **Test 2**, we change the form of question description, GTS and our model obtain the same expressions that generated in the original problem. This is because we use the attention mechanism in the model, changing the form of question description has no impact on generating correct expressions.

Since Graph2Tree also considered the quantity comparison graph in the model, the same results are obtained in this case as our model.

4.5 Ablation Study

Last but not least, we conduct an ablation study to better understand the effect of encoders and decoders in the model, as is shown in Table 6. When we use a fully connected layer to replace the sequence-based encoder, the performance of our model observably drops. This is because other encoders and decoders depend on the context representation obtained by the sequence-based encoder. We find that the performance has a drop if we discard any type of graph-based structure, which proves the importance of considering the structure information in text descriptions. When the model has only one decoder, the generation ability is limited, which indicates the necessity of designing multi-decoders.

Model	**Math23K (%)**
Full Model	78.4
- Sequence-Based Encoder	69.7
- Graph-Based Encoder (Parse Graph)	76.4
- Graph-Based Encoder (Numerical Graphs)	76.1
- Sequence-Based Decoder	76.6
- Tree-Based Decoder	71.3

Table 6: Effect of encoders and decoders in the model.

5 Conclusion and Future Work

Inspired by the fact that text descriptions and equation expressions have structural information, a model with multi-encoders and multi-decoders is proposed in this paper. To be specific, we use the sequence-based encoder to obtain the context representation, and the graph-based encoder is used to integrate the structure information of text descriptions. Two types of decoders generate different expressions, which strengthens the generation ability of the model. Experimental results on Math23K proves the advantages of our model over existing state-of-the-art methods. The Experiment analysis shows that the effectiveness of mathematical logic mining from the problem. In future work, we will explore the question generation technique to increase samples of the dataset and solve the problems with complex expressions.

Acknowledgements

Thanks to the anonymous reviewers for their helpful comments and suggestions. This work is partially supported by National Science Foundation of China (U1811264, U1911203 and 61877018) and ECNU Academic Innovation Promotion Program for Excellent Doctoral Students (YBNLTS2019-022). Cheqing Jin is the corresponding author.

References

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

Ting-Rui Chiang and Yun-Nung Chen. 2019. Semantically-aligned equation generation for solving and reasoning math word problems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, volume 1, pages 2656–2668.

Charles R. Fletcher. 1985. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods Instruments & Computers*, 17(5):565–571.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 1024–1034.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 523–533.

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.

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.

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 Conference of the Association for Computational Linguistics (Volume 1: Long Papers)*, 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. *arXiv preprint arXiv:2004.13781*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 158–167.

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.

Yuanliang Meng and Anna Rumshisky. 2019. Solving math word problems with double-decoder transformer. *arXiv preprint arXiv:1908.10924*.

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)*, volume 1, pages 2144–2153.

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.

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*, pages 5545–5552.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Thirty-Third AAAI Conference on Artificial Intelligence*, pages 7144–7151.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 5299–5305.

Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang Ronghuai. 2010. Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems. In *2010 Second International Workshop on Education Technology and Computer Science*, volume 2, pages 476–479.

Dongxiang Zhang, Lei Wang, Nuo Xu, Bing Tian Dai, and Heng Tao Shen. 2018. The gap of semantic parsing: A survey on automatic math word problem solvers. *arXiv preprint arXiv:1808.07290*.

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 Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937.