A Goal-Driven Tree-Structured Neural Model for Math Word Problems

Zhipeng Xie[*†] and **Shichao Sun**[†]

Shanghai Key Laboratory of Data Science, Fudan University
School of Computer Science, Fudan University

{xiezp, scsun17}@fudan.edu.cn

Abstract

Most existing neural models for math word problems exploit Seq2Seq model to generate solution expressions sequentially from left to right, whose results are far from satisfactory due to the lack of goal-driven mechanism commonly seen in human problem solving. This paper proposes a tree-structured neural model to generate expression tree in a goal-driven manner. Given a math word problem, the model first identifies and encodes its goal to achieve, and then the goal gets decomposed into sub-goals combined by an operator in a top-down recursive way. The whole process is repeated until the goal is simple enough to be realized by a known quantity as leaf node. During the process, two-layer gated-feedforward networks are designed to implement each step of goal decomposition, and a recursive neural network is used to encode fulfilled subtrees into subtree embeddings, which provides a better representation of subtrees than the simple goals of subtrees. Experimental results on the dataset `Math23K` have shown that our tree-structured model outperforms significantly several state-of-the-art models.

1 Introduction

The task of Math word problems (MWPs) aims to automatically answer a mathematical query according to the text description. A typical MWP is a short narrative that describes a partial state of the world and poses a question about an unknown quantity. An example is shown in Table 1. Students are asked to answer how many baggies Robin could make according to problem text. Recently, the task of MWPs has attracted a lot of research attention. Researchers have proposed several approaches [Ling *et al.*, 2017; Wang *et al.*, 2017; Huang *et al.*, 2018; Wang *et al.*, 2018a] to solving MWPs based on the Seq2Seq model [Sutskever *et al.*, 2014]. These Seq2Seq-based solvers have been proved to have the power of generating new expressions that do not exist in the training dataset [Wang *et al.*, 2017]. Besides, another advantage of the Seq2Seq-based models exists in that they do not rely on hand-crafted features.

Problem: Robin was making baggies of cookies with 6 cookies in each bag. If she had 23 chocolate cookies and 25 oatmeal cookies, how many baggies could she make?	
Solution Expression: $(23 + 25) \div 6$	**Solution:** 8

Table 1: A typical math word problem

However, the Seq2Seq-based models do not match the goal-driven mechanism in human problem solving. When human reads the problem text of a MWP, he firstly figures out which target quantity is to be derived as the goal, and then pays attention to the relevant information of the problem which can help to realize the goal. If the goal can be realized directly by the relevant information, the problem solving has been completed; otherwise, human needs to decompose the goal into two sub-goals combined by an operator based on the relevant information. Next, the same process is repeated for each sub-goal until all goals get realized. As for the example shown in Table 1, the goal is to calculate how many baggies Robin could make. With this goal, human may pay attention to the relevant information "6 cookies in each bag; She had 23 chocolate ... oatmeal cookies." in the problem. Then, human infers that the goal can be realized as two sub-goals connected by the operator "÷": the first sub-goal is about "how many cookies she had in total", the second is "how many cookies in each bag". Similarly, the first sub-goal is realized by two sub-goals("how many chocolate cookies" and "how many oatmeal cookies") combined by an operator "+". This moment, all sub-goals can be easily realized by the existing numeric values in the problem, and the expression tree is generated as illustrated in Figure 1. As a summary, human decomposes the goal recursively for solving a math word problem and finally generates an expression tree.

Another issue is that the Seq2Seq-based models may generate invalid expressions which can not be calculated. For example, it is difficult to know exactly how many consecutive left parentheses would be needed before the inside operations are handled, which may sometimes lead to the wrong expression such as "$((23 + 25) \div 6$" for the example in Table 1. Although this issue could be solved by using the postorder traversal of expression tree as target sequence in the Seq2Seq model [Wang *et al.*, 2018a], it is challenging for Seq2Seq

[*]Contact Author.
[†]Equal Contribution.

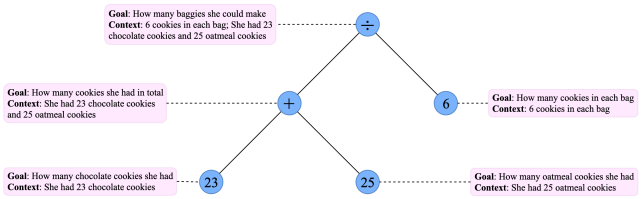

Figure 1: The expression tree of the problem in Table 1

to model the tree-structured relationship of expression tree through its postorder traversal sequence during decoding.

Motivated by the goal-driven mechanism in human problem solving, we design a novel model to generate expression tree. Following a top-down goal decomposition process, our model firstly initializes the root goal vector which represents the final goal of the problem, and then summarizes relevant information of the problem into the context vector. Next, a token is predicted using the goal vector and its context vector, which implicitly decides whether the goal should be decomposed further. If the predicted token is a numeric value or constant quantity, the goal is realized directly; otherwise (i.e., the predicted token is an operator), two new sub-goal vectors (one for left sub-goal and the other for the right) will be generated. Lastly, prediction and the goal decomposition process are repeated for them. However, for a commutative operator such as "$+$" or "$\times$", its right sub-goal may be the same as the left one, due to its commutative property. To address this issue, our model completes the construction of the left subtree before generating the right sub-goal. The generation of right sub-goal takes the information of its left sibling subtree into consideration, which is encoded as a subtree embedding by a recursive neural network.

Our contributions are summarized here:

1. We propose a novel neural model to generate an expression tree in a human-like goal-driven way for solving math word problems. To the best of our knowledge, this is the first tree-structured neural model for MWPs.
2. Our model is tree-structured and the information explicitly flows through the expression tree in both top-down (goal decomposition) and bottom-up (subtree embedding) manners.
3. The experimental results show our model significantly outperforms several state-of-the-art systems on the dataset `Math23K`.

2 Related Work

Research in automatically solving MWPs has a very long history, ranging from rule-based methods [Fletcher, 1985; Bakman, 2007; Yuhui *et al.*, 2010], statistical machine learning methods [Kushman *et al.*, 2014; Zhou *et al.*, 2015; Mitra and Baral, 2016; Roy and Roth, 2018] and semantic parsing methods [Shi *et al.*, 2015; Koncel-Kedziorski *et al.*, 2015; Huang *et al.*, 2017] to deep learning methods [Ling *et al.*, 2017; Wang *et al.*, 2017; Wang *et al.*, 2018b; Huang *et al.*, 2018]. Here we will review some recent works based on Seq2Seq models. For a more thorough review of automatic MWPs solver, please refer to a recent survey paper [Zhang *et al.*, 2018].

Wang *et al.* [2017] made the first attempt to generate expression using Seq2Seq model with Recurrent Neural Network (RNN) in its encoder and decoder. Robaidek *et al.* [2018] have tried to use Convolutional Neural Network (CNN) instead of RNN [Wang *et al.*, 2017]. These models have obtained promising results. However, the use of Seq2Seq models has resulted in some shortcomings. Huang *et al.* [2018] pointed out that the Seq2Seq model may generate spurious numbers or predict numbers at wrong positions. They added copy-and-alignment mechanism to the standard Seq2Seq model to solve these issues. Wang *et al.* [2018a] observed that the Seq2Seq model always suffers from an equation duplication problem: a MWP can be solved by multiple expressions. They proposed an equation normalization method to solve this problem.

Different from previous Seq2Seq-based works, we design a novel tree-structured model to directly generate the expression tree in human-like goal-driven way.

3 Problem Statement

A math word problem (P, T) consists of a problem text P and a solution expression tree T, where:

- The problem text P is a sequence of word tokens and numeric values, which usually begins by describing a partial quantitative state of a world, followed by simple updates, and ends with a query about an unknown quantity. Let n_P denote the ordered list of numeric values in P according to their order in the problem text. At a preprocessing step, all the number tokens are treated as a special word token `NUM`, because we usually do not care about their exact values in solving a math word problem.
- The solution expression tree T is a mathematical expression tree, which can be easily transformed from the solution expression. What's more, the tree T captures the relations among these numeric values which are described or implied literally by the problem text P. Generally, T may contain constant quantities, mathematical operators, and numeric values in n_P from problem text P. The set of mathematical operators (denoted as V_{op}) contains commonly-used operators such as "$+$" and "$-$". The set of constant quantities (denoted as V_{con}) contains some special numeric values that may occur in the solution but not in the problem text, such as π and 2. Such as the problem "The radius of a circle is 3.5, what is the circumference?", the solution is "$2 \times \pi \times 3.5$" where the constant quantities 2 and π cannot be found in the text. Therefore, the target vocabulary of P is denoted as $V^{dec} = V_{op} \cup V_{con} \cup n_P$. It should be noticed that the target vocabulary V^{dec} varies in different problem P as a result of varied n_P. Besides, when the problem text P contains two identical numbers since the number occurs in two different positions of P, we treat the two numbers as different target tokens in n_P, and during the training process, choose the occurrence of higher probability (Equation (8)) as the target.

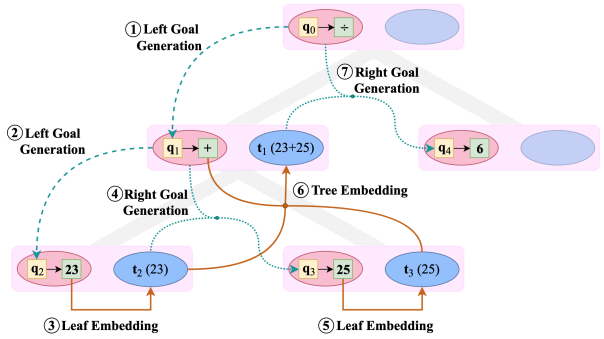

Figure 2: Goal-driven Tree-structured Model

4 Model Description

Motivated by the commonly-used goal-driven mechanism in human problem solving, our method is designed to generate expression trees by explicitly modeling tree-structured relationship between quantities. Each node $\mathbf{n}$ in the expression tree T consists of three main components: a goal vector $\mathbf{q}$, a token $\hat{y}$, and a subtree embedding $\mathbf{t}$ of $\mathbf{n}$. The goal vector $\mathbf{q}$ is used to instruct how the subtree root from node $\mathbf{n}$ should be constructed, and the subtree is generated to realize the goal. To do so, our method first predicts the token $\hat{y}$ according to the goal vector $\mathbf{q}$. In turn, the predicted token naturally decides whether the goal should be decomposed further. That is, if the predicted token is a mathematical operator, the goal will be decomposed into two sub-goals (a left sub-goal $\mathbf{q}_l$ and a right one $\mathbf{q}_r$) combined by the operator, where the left (right) sub-goal in turn serves to drive the construction of the left (right) subtree of $\mathbf{n}$; otherwise, the goal will be simply realized by the predicted numeric value or constant quantity. Such a goal decomposition process is conducted recursively just like a depth-first traversal. An example of partly constructed expression tree is illustrated in Figure 2.

More specifically, the goal decomposition process is implemented as the pre-order operation during the traversal. As such, the left sub-goal is generated according to the goal vector and the predicted token of its parent node (Equation (10)), as illustrated by the blue dashed lines of steps ①② in Figure 2. When it comes to the right sub-goal, the construction of its left sibling subtree has been completed. To make full of available information, the generation of right sub-goal should take the information of its left sibling subtree into consideration, in addition to the parent goal and the left sub-goal (Equation (11)), as blue dotted lines of steps ④⑦ in Figure 2. To encode the subtree information of a non-leaf node, bottom-up recursive neural network is defined which fuses the token embedding of its mathematical token, and the embeddings of its left and right subtrees (Equation (12) and Equation (13)), as red curved lines of step ⑥ in Figure 2. For a leaf node, its subtree embedding is simply represented as the embedding of its token (Equation (12) and Equation (5)), as red straight lines of steps ③⑤ in Figure 2).

4.1 Encoder

In encoder, every word of the problem text P is encoded as context representation. As for an input problem text $P = x_1 x_2 \dots x_n$, each word token x_i is firstly transformed into the corresponding word embedding $\mathbf{x}_i$ by looking up an encoder-embedding matrix $\mathbf{M}_{sen}$. Then, the sequence of embeddings is inputted to the Gated Recurrent Unit (GRU) [Cho *et al.*, 2014] from left to right and from right to left. Formally, the encoder takes the words x_1, x_2, …, x_n of problem text P one-by-one as the input, and produces a sequences of encoder hidden states $\overrightarrow{\mathbf{h}_1^p}$, $\overrightarrow{\mathbf{h}_2^p}$, …, $\overrightarrow{\mathbf{h}_n^p}$ as the output, where the hidden state $\overrightarrow{\mathbf{h}_s^p}$ at step s is calculated according to the previous hidden state $\overrightarrow{\mathbf{h}_{s-1}^p}$ and the current input $\mathbf{x}_s$:

$$\overrightarrow{\mathbf{h}_s^p} = \text{GRU}(\overrightarrow{\mathbf{h}_{s-1}^p}, \mathbf{x}_s) \tag{1}$$

where $\text{GRU}(\cdot,\cdot)$ denotes the function of a two-layer GRU. On the other direction, the hidden state $\overleftarrow{\mathbf{h}_s^e}$ at step s is calculated according to the following hidden state $\overleftarrow{\mathbf{h}_{s+1}^e}$ and the current input $\mathbf{x}_s$:

$$\overleftarrow{\mathbf{h}_s^p} = \text{GRU}(\overleftarrow{\mathbf{h}_{s+1}^p}, \mathbf{x}_s) \tag{2}$$

The final hidden state $\mathbf{h}_s^p$ has incorporated contextual information of the source token x_s, which is calculated as follows:

$$\mathbf{h}_s^p = \overrightarrow{\mathbf{h}_s^p} + \overleftarrow{\mathbf{h}_s^p} \tag{3}$$

4.2 Root Goal Initialization and Token Embedding

To start the top-down goal decomposition process, our method initializes the goal vector $\mathbf{q}_0$ of the root node $\mathbf{n}_0$ (or equivalently, the whole expression tree T) according to the hidden states of the encoder of P:

$$\mathbf{q}_0 = \overrightarrow{\mathbf{h}_n^p} + \overleftarrow{\mathbf{h}_0^p} \tag{4}$$

where $\overrightarrow{\mathbf{h}_n^p}$ and $\overleftarrow{\mathbf{h}_0^p}$ are the final hidden states of forward sequence and backward sequence respectively.

For each token y in the target vocabulary V^{dec} of P, its token embedding $\mathbf{e}(y|P)$ is defined as:

$$\mathbf{e}(y|P) = \begin{cases} \mathbf{M}_{op}(y) & \text{if } y \in V_{op} \\ \mathbf{M}_{con}(y) & \text{if } y \in V_{con} \\ \mathbf{h}^p_{loc(y,P)} & \text{if } y \in n_P \end{cases} \tag{5}$$

The token embeddings of mathematical operators and constant quantities are obtained by looking up two trainable embedding matrices $\mathbf{M}_{op}$ and $\mathbf{M}_{con}$ respectively, which are independent of the specific problems. However, for a numeric value in n_P, its token embedding takes the corresponding hidden state $\mathbf{h}^p_{loc(y,P)}$ from the encoder, where $loc(y, P)$ is the index position of y in P. Clearly, the token embeddings of numeric values are dependent on the specific problems in which they occur.

4.3 Top-down Goal Decomposition

Given a goal vector $\mathbf{q}$, we first derive a context vector $\mathbf{c}$ that summarizes relevant information of the problem at hand,

which is expected to help predict the token and make the following decisions. The context vector $\mathbf{c}$ is calculated as a weighted representation of the source tokens by a vector $\mathbf{a}$ of attention weights a_s:

$$\mathbf{c} = \sum_s a_s \mathbf{h}_s^p \tag{6}$$

where

$$a_s = \frac{\exp(\text{score}(\mathbf{q}, \mathbf{h}_s^p))}{\sum_i \exp(\text{score}(\mathbf{q}, \mathbf{h}_i^p))}$$

and

$$\text{score}(\mathbf{q}, \mathbf{h}_s^p) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\mathbf{q}, \mathbf{h}_s^p]).$$

where $\mathbf{v}_a$ and $\mathbf{W}_a$ are trainable parameters.

Next, the unnormalized log probability $\text{s}(y|\mathbf{q}, \mathbf{c}, P)$ of generating a token y from the target vocabulary V^{dec} is formulated as:

$$\text{s}(y|\mathbf{q}, \mathbf{c}, P) = \mathbf{w}_n^\top \tanh\left(\mathbf{W}_s\left[\mathbf{q}, \mathbf{c}, \mathbf{e}(y|P)\right]\right) \tag{7}$$

where $\mathbf{e}(y|P)$ is the token embedding of y in Equation (5), $\mathbf{w}_n$ is a trainable vector, and $\mathbf{W}_s$ is a trainable matrix.

The probability $\text{prob}(y|\mathbf{q}, \mathbf{c}, P)$ over target vocabulary is the normalization of $\text{s}(y|\mathbf{q}, \mathbf{c}, P)$ through softmax:

$$\text{prob}(y|\mathbf{q}, \mathbf{c}, P) = \frac{\exp\left(\text{s}\left(y|\mathbf{q}, \mathbf{c}, P\right)\right)}{\sum_i \exp\left(\text{s}\left(y_i|\mathbf{q}, \mathbf{c}, P\right)\right)} \tag{8}$$

The token $\hat{y}$ with the highest probability is generated as the prediction:

$$\hat{y} = \underset{y \in V^{dec}}{\arg\max}\, \text{prob}(y|\mathbf{q}, \mathbf{c}, P) \tag{9}$$

The predicted token $\hat{y}$ implies a decision about how to realize the goal $\mathbf{q}$: if $\hat{y}$ is a numeric value or a constant quantity, the goal is realized directly by $\hat{y}$; otherwise (i.e., $\hat{y}$ is an operator), the goal will be decomposed into two sub-goals.

Left Sub-Goal Generation

Let the predicted token $\hat{y}$ is an operator, the current goal q will be realized by a left sub-goal $\mathbf{q}_l$ and a right sub-goal $\mathbf{q}_r$. The left sub-goal vector $\mathbf{q}_l$ is calculated by a two-layer feedforward neural network with gating mechanism:

$$\begin{aligned} o_l &= \sigma\left(\mathbf{W}_{ol}\left[\mathbf{q}, \mathbf{c}, \mathbf{e}(\hat{y}|P)\right]\right) \\ C_l &= \tanh\left(\mathbf{W}_{cl}\left[\mathbf{q}, \mathbf{c}, \mathbf{e}(\hat{y}|P)\right]\right) \\ \mathbf{h}_l &= o_l \odot C_l \\ g_l &= \sigma\left(\mathbf{W}_{gl}\mathbf{h}_l\right) \\ Q_{le} &= \tanh\left(\mathbf{W}_{le}\mathbf{h}_l\right) \\ \mathbf{q}_l &= g_l \odot Q_{le} \end{aligned} \tag{10}$$

where $\mathbf{W}_{ol}$, $\mathbf{W}_{cl}$, $\mathbf{W}_{gl}$, $\mathbf{W}_{le}$ are trainable matrices. $\mathbf{h}_l$ is the hidden state, which parent node delivers to its left child.

Right Sub-Goal Generation

The right sub-goal quantity of right child takes into account the left child subtree, which has been generated prior to the right child owing to the essence of pre-order traversal. The left subtree is encoded bottom up as $\mathbf{t}_l$ according to the recursive neural network described by Equation (12) in Section 4.4. Then the goal vector $\mathbf{q}_r$ of the right child is calculated as follows:

$$\begin{aligned} o_r &= \sigma\left(\mathbf{W}_{or}\left[\mathbf{q}, \mathbf{c}, \mathbf{e}(\hat{y}|P)\right]\right) \\ C_r &= \tanh\left(\mathbf{W}_{cr}\left[\mathbf{q}, \mathbf{c}, \mathbf{e}(\hat{y}|P)\right]\right) \\ \mathbf{h}_r &= o_r \odot C_r \\ g_r &= \sigma\left(\mathbf{W}_{gr}\left[\mathbf{h}_r, \mathbf{t}_l\right]\right) \\ Q_{re} &= \tanh\left(\mathbf{W}_{re}\left[\mathbf{h}_r, \mathbf{t}_l\right]\right) \\ \mathbf{q}_r &= g_r \odot Q_{re} \end{aligned} \tag{11}$$

where $\mathbf{t}_l$ is the tree embedding of its sibling node and $\mathbf{W}_{or}$, $\mathbf{W}_{cr}$, $\mathbf{W}_{gr}$, $\mathbf{W}_{re}$ are trainable matrices. $\mathbf{h}_r$ is the hidden state, which parent node top-down delivers to its right child.

4.4 Subtree Emebedding via Recursive Neural Network

In this subsection, we design a recursive neural network to encode a subtree in a bottom-up manner, which will play a role in the right sub-goal generation (Section 4.3).

Let t be the subtree at hand, and $\hat{y}$ denote its predicted token. The embedding $\mathbf{t}$ of t is defined recursively as:

$$\mathbf{t} = \begin{cases} comb(\mathbf{t}_l, \mathbf{t}_r, \hat{y}) & \text{if } \hat{y} \in V_{op} \\ \mathbf{e}(\hat{y}|P) & \text{if } \hat{y} \in n_P \cup V_{con} \end{cases} \tag{12}$$

If the predicted token is an operator ($\hat{y} \in V_{op}$), which means that the subtree t must have two child subtrees t_l and t_r, the embedding of t needs to fuse the information from the operator $\hat{y}$, the left child t_l and the right child t_r, as done by the function $comb(\mathbf{t}_l, \mathbf{t}_r, \hat{y})$ with gating mechanism:

$$\begin{aligned} comb(\mathbf{t}_l, \mathbf{t}_r, \hat{y}) &= g_t \odot C_t \\ g_t &= \sigma\left(\mathbf{W}_{gt}\left[\mathbf{t}_l, \mathbf{t}_r, \mathbf{e}(\hat{y}|P)\right]\right) \\ C_t &= \tanh\left(\mathbf{W}_{ct}\left[\mathbf{t}_l, \mathbf{t}_r, \mathbf{e}(\hat{y}|P)\right]\right) \end{aligned} \tag{13}$$

where $\mathbf{W}_{ct}$ and $\mathbf{W}_{gt}$ are trainable parameter matrices, $\mathbf{e}(\hat{y}|P)$ is the embedding of the operator $\hat{y}$ (Equation (5)).

On the other hand, if $\hat{y}$ is a numeric value or a constant quantity, the recursion stops and t becomes a leaf node. The embedding of subtree t is simply set as the corresponding token embedding $\mathbf{e}(y|P)$ (Equation (5)).

4.5 Training Objective

Given the training dataset $\mathbb{D} = \{(P^i, T^i) : 1 \le i \le N\}$, where T^i is the solution expression tree of problem P^i, our objective to minimize is the negative log-likelihood of $\mathbb{D}$:

$$J = \sum_{(P,T)\in\mathbb{D}} -\log p(T|P) \tag{14}$$

In the training stage, for each sample $(P, T) \in \mathbb{D}$, the decoder generates one target token at each step, in the pre-order traversal of T. It ensures that the ground truth is used as the tree structure during training. Therefore, the conditional probability $p(T|P)$ can be decomposed as:

$$p(T|P) = \prod_{t=1}^{m} \text{prob}(y_t|\mathbf{q}_t, \mathbf{c}_t, P) \tag{15}$$

where m denotes the size of T, $\mathbf{q}_t$ and $\mathbf{c}_t$ are the goal vector and its context vector at the t-th node, the probabilities $\text{prob}(\cdot|\cdot)$ are calculated by Equation (8).

Model	Accuracy(%)
Hybrid model w/ SNI [Wang *et al.*, 2017]	64.7
Ensemble model w/ EN [Wang *et al.*, 2018a]	68.4
GTS model w/o Subtree Embedding	70.0
GTS model	**74.3**

Table 2: Model comparison on answer accuracy

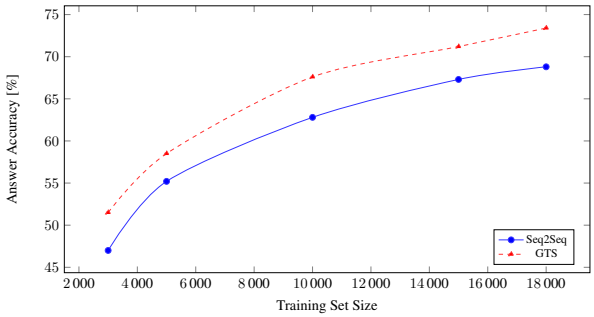

Figure 3: Answer accuracy versus training set size

5 Experimental Evaluation

In this section, we conduct experiments to evaluate our model with several state-of-the-arts on a large dataset `Math23K`[1].

Dataset. The dataset `Math23K` contains 23,161 math word problems annotated with solution expressions and answers. To the best of our knowledge, `Math23K` is the largest among the datasets of math word problems, such as `Alg514` [Kushman *et al.*, 2014] with 514 problems and `AllArith` [Roy and Roth, 2017] with 831 problems. All problems in this dataset can be solved by one linear algebra expression, and the solution expression can be easily transformed into the corresponding expression tree.

Models for Comparison. The methods to be compared are listed as below:

- Hybrid model w/ SNI [Wang *et al.*, 2017]: Hybrid model combines the retrieval model and the seq2seq model with *significant number identification*(SNI).
- Ensemble model w/ EN [Wang *et al.*, 2018a]: Ensemble model selects the result according to models' generation probability among **BiLSTM**, **ConvS2S** and **Transformer** with *equation normalization*(EN).
- Goal-driven tree-structured MWP solver (GTS in short): the method proposed in this paper.
- GTS model w/o Subtree Embedding: To make clear the effect of subtree embedding component on the performance of GTS, we implement right sub-goal generation in the same way as the left sub-goal (that is, the subtree-embedding component gets removed).

5.1 Implementation Details

Our model is implemented using PyTorch[2] on a Ubuntu system with GTX1080Ti. All the words with less than 5 occurrences are converted to a universal token "`<unk>`". The dimensionality of word embedding layer is set to 128, and the dimensionalities of all hidden states for the other layers are set to 512.

Our model is trained for 80 epochs by Adam optimization algorithm [Kingma and Ba, 2014] where the mini-batch size is set to 64. In each epoch, all the training data get shuffled randomly, and then cut into mini-batches. The initial value of learning rate is set to 0.001, and the learning rate will be halved every 20 epochs. In addition, we set the dropout probability [Hinton *et al.*, 2012] as 0.5 and weight decay as 1e-5 to prevent overfitting. Last but not least, we set the beam size to 5 in beam search to generate expression trees.

[1]Available from http://ai.tencent.com/ailab/Deep_Neural_Solver_for_Math_Word_Problems.html

[2]http://pytorch.org

5.2 Results and Analyses

Answer Accuracy. We use *answer accuracy* as the evaluation metric: the predicted expression tree is thought of correct, if its calculated value equals to the answer. The *answer accuracy* metric is reasonable as a result of expression equivalence. For example, the predicted expression tree of "$n_0 + n_1$" is different from the target expression tree of "$n_1 + n_0$", but they are equivalent and their calculated values are equal. The results are summarized in Table 2, where the *answer accuracies* are evaluated on the `Math23K` dataset via 5-fold cross-validation. Several observations can be made from Table 2: First, the GTS model without subtree embedding module has achieved slightly better performance than the two state-of-the-art models. It proves that the goal-driven mechanism is feasible for solving the math word problems. Second, with the subtree embedding module, our GTS model has gained additional absolute 4.3% on accuracy, which means that the subtree embedding module is helpful and complementary to top-down goal decomposition process, especially for the generation of right sub-goal. Last, the GTS model with subtree embedding module outperforms the compared state-of-the-art systems by a large margin of nearly 6% on absolute accuracy.

Answer Accuracy vs. Training Set Size Training deep models typically needs large training data, so we conduct experiment to check how the performance of our model varies with respect to different numbers of training instances. We implement a Seq2Seq model with attention mechanism as the baseline model, which contains the same encoder as GTS model but takes GRU as decoder. The test set contains 4,632 randomly sampled instances (20% of the whole dataset). The GTS model, together with the Seq2Seq baseline, is trained on different numbers of the remaining training instances ranging from 3,000 to 18,000. From the results as illustrated in Figure 3, it can be observed that the accuracy increases with the number of training instances. The increase in accuracy is faster for smaller training set (as seen from 3,000 to 10,000), and this trend of increase is slowing with the growth of training set (beyond 10,000 instances). In addition, the GTS outperforms the Seq2Seq for all cases, and the gap between GTS with Seq2Seq takes on a weak increasing trend.

Case 1: The store shipped in a batch of leather shoes. NUM(n_0 [$\frac{1}{3}$]) of the total was sold on the first day, and NUM(n_1 [$\frac{3}{5}$]) of the first day's sale was sold on the second day. There were NUM(n_2 [280]) pairs left. How many pairs of leather shoes did the store bring in? **Seq2Seq:** $\div n_2 - 1 n_0 * n_0 n_1$;**(error)** **GTS:** $\div n_2 - -1 n_0 * n_0 n_1$;**(correct)**
Case 2: Of the NUM(n_0 [697]) combined shipment equipments of Shenzhou NUM(n_1 [7]) spacecraft, NUM(n_2 [346]) are followed, NUM(n_3 [237]) are updated, and the rest are newly developed. How many new equipments are there? **Seq2Seq:** $- - n_0 n_3 n_4$;**(error)** **GTS:** $- - n_0 n_2 n_3$;**(correct)**
Case 3: Guangming Primary School spent NUM(n_0 [288]) yuan on NUM(n_1 [12]) chairs. And then NUM(n_2 [36]) chairs of the same kind were bought. How much did the school spend on chairs? **GTS w/o Subtree Embedding**: $\times \div n_0 n_1 \div n_0 n_1$;**(error)** **GTS:** $\times \div n_0 n_1 + n_1 n_2$;**(correct)**

Table 3: Typical cases. Note that the results are represented as pre-order traversal of expression trees.

Expr. Tree Size	3-	5	7	9+
# Test Instances	907	2303	921	501

Table 4: Numbers of test instances over expression tree sizes

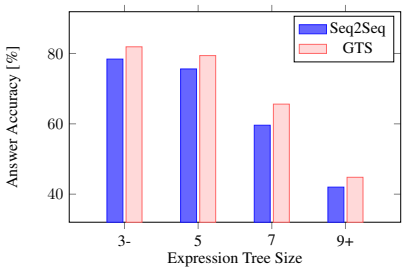

Figure 4: Answer accuracies for different sizes of expression trees

Performance on Expression Length Intuitively, the larger the size of expression tree is, the more complex the mathematical relationship of the problem is, and the more difficult it is to solve the problem. Here, we also use the Seq2Seq model as the baseline and randomly select 20% instances from Math23K as test dataset and all the rest instances as training dataset. The test dataset is partitioned into 4 subsets according to the sizes of expression trees, as shown in Table 4. The *answer accuracies* of all the 4 subsets are listed in Figure 4. We can see that there is a clear tendency for *answer accuracy* to degrade with the growth of the problem complexity measured as the size of expression tree, and our GTS model outperforms the baseline Seq2Seq model in all situations of different expression tree sizes, which indicates that our GTS model can better model the mathematical relationships of the problem in an explicit tree structure. It can also be noticed that the promotion of our GTS model over the Seq2Seq model is increasing when the problem complexity becomes more complex. When the expression tree size is no less than 9, the tree will contain at least 4 mathematical operators, the corresponding problem may be hard to solve in some degree, and it is observed that there is a clear drop on accuracy for both GTS and Seq2Seq models.

Results on a Small Dataset. Apart from the large Math23K dataset, we also test the performance of the GTS model on another small dataset called AllArith[3], which consist of 831 instances. We use the same 5-fold cross-validation as in [Roy and Roth, 2017]. To improve the reproducibility, we repeat the experiment 20 times, and the average accuracy of GTS is 69.5%, compared with 66.5% of Seq2Seq model. Due to the dataset is small, we also perform **McNemar's test** and get p-value 0.001, which shows that it rejects the null hypothesis and this increase is statistically significant.

[3]Available from https://github.com/CogComp/arithmetic

5.3 Case Study

Further, we conduct a case analysis and provide three cases in Table 3. Our analyses are summarized as follows:

- From **Case 1**, we can see that the GTS model can avoid generating mathematically invalid expressions. This is because our model generates the expression tree directly, and its sequence of pre-order traversal can be guaranteed to be computable;
- From **Case 2**, it is obvious that the GTS model can avoid predicting spurious numbers, such as "n_4", which can not be converted back to a number. This is because the effective size of target vocabulary is set dynamically according to the specific problem;
- From **Case 3**, we find that the subtree embedding component can prevent generating the same subtree as its left sibling when the parent node is "$+$" or "$\times$".

6 Conclusion

Motivated by the goal-driven mechanism in human problem solving, we propose a novel neural model (called GTS) for math word problems by directly predicting an expression tree. In our model, the information is able to flow explicitly through the expression tree by top-down goal decomposition and bottom-up subtree embedding. Experimental results have demonstrated that the proposed GTS model can significantly outperform previous state-of-the-art systems. Case study has shown that the GTS model can avoid generating mathematically invalid expressions and spurious numbers.

Acknowledgements

This work is supported by National Key Research and Development Program of China (No.2018YFB1005100). We are grateful to the anonymous reviewers for their valuable comments.

References

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

[Cho *et al.*, 2014] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. pages 1724–1734, October 2014.

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

[Hinton *et al.*, 2012] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[Huang *et al.*, 2017] Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 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, 2017.

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

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[Kushman *et al.*, 2014] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 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, 2014.

[Ling *et al.*, 2017] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

[Mitra and Baral, 2016] Arindam Mitra and Chitta Baral. 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, 2016.

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

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

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

[Shi *et al.*, 2015] Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 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, 2015.

[Sutskever *et al.*, 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

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

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

[Wang *et al.*, 2018b] Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. 2018.

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

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

[Zhou *et al.*, 2015] Lipu Zhou, Shuaixiang Dai, and Liwei Chen. Learn to solve algebra word problems using quadratic programming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 817–822, 2015.