A Bottom-Up DAG Structure Extraction Model for Math Word Problems

Yixuan Cao[1,2], **Feng Hong**[1,2], **Hongwei Li**[1,2], **Ping Luo**[1,2,3]
[1]Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS),
Institute of Computing Technology, CAS, Beijing 100190, China.
[2]University of Chinese Academy of Sciences, Beijing 100049, China.
[3]Peng Cheng Laboratory, Shenzhen, China.
{caoyixuan, hongfeng18g, lihongwei, luop}@ict.ac.cn

Abstract

Research on automatically solving mathematical word problems (MWP) has a long history. Most recent works adopt the Seq2Seq approach to predict the result equations as a sequence of quantities and operators. Although result equations can be written as a sequence, it is essentially a structure. More precisely, it is a Direct Acyclic Graph (DAG) whose leaf nodes are the quantities, and internal and root nodes are arithmetic or comparison operators. In this paper, we propose a novel Seq2DAG approach to extract the equation set directly as a DAG structure. It extracts the structure in a bottom-up fashion by aggregating quantities and sub-expressions layer by layer iteratively. The advantages of our approach are three-fold: it is intrinsically suitable to solve multivariate problems, it always outputs valid structure, and its computation satisfies commutative law for +, × and =. Experimental results on DRAW1K and Math23K datasets demonstrate that our model outperforms state-of-the-art deep learning methods. We also conduct detailed analysis on the results to show the strengths and limitations of our approach.

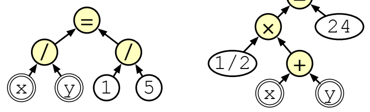
Introduction

Automatically solving Math Word Problem (MWP) is a classical problem in artificial intelligence. Research on automatically solving MWP has a long history back to the 1960s (Bobrow 1964). The task of MWP is to solve a math problem described in natural language. We show an example in Table 1. The input is a verbal description of a problem which involves some quantities and variables. The output is the answer which specifies the quantities of the variables, as shown in the "Answer" row. To get the answer, one needs to translate the natural language into a mathematical equation system, and then solve the equation system. If we set the age of Tom as x and the age of his father as y, the equation system of the problem is shown in the "Equations" row in Table 1. There are many mature tools like Octave to solve a given equation system, but how to translate the problem into an equation system is a challenging task till now.

Recently, research on this task is flourishing with the presence of deep learning models in the field of natural language processing. Inspired by machine translation research, many recent work formulate the task as a sequence to sequence

Input
Problem: The ages of Tom and his father are in the ratio of 1: 5, 1/2 of their sum is 24. Find their ages.

Output (In different forms)
Answer: 8, 40
Equations: $x/y = 1/5, \quad \frac{1}{2} \times (x+y) = 24$
Equation Trees:



Aligned DAG:

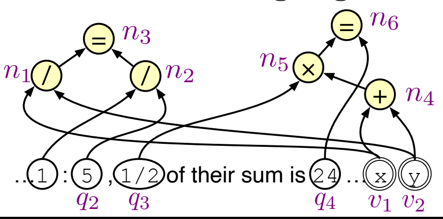


Table 1: Word problem example.

translation problem from natural language to mathematical language, and adopt Seq2Seq approach (Wang, Liu, and Shi 2017; Huang et al. 2018; Wang et al. 2019) on this problem. The mathematical language is a sequence composed of quantities, variables and operators. Although humans write the equation from left to right as a sequence, the equation system is actually a structure. Such Seq2Seq methods overlook the structure information within equations, and might result in invalid results.

As shown in the "Equation Trees" row in Table 1, each equation in the equation system is a tree whose leaf nodes are quantities and variables and other nodes are operators. Xie and Sun (2019) proposed a Seq2Tree model that decodes the equation as a tree from root to leaf. The last hidden vector from the problem encoder is fed into the decoder. The root node is first decoded. Then, the tree is decoded in preorder, from top to bottom, from left to right. They obtained a significant improvement compared with Seq2Seq models. This indicates the importance of the structure information.

But Seq2Tree methods focus on arithmetic problems which only have one variable. This is because the result of arithmetic problem are simpler that have only one equation or expression, like $2 * x = 1 + 2$, or $(1 + 2)/2$. They cannot solve equation set problems that have multiple equations and variables such as the example in Table 1. Extending them to decode multiple trees for equation set problems would require substantial revisions. For example, as they have to decode multiple equations one by one, the order of equations have to be determined. But equations in an equation set are unordered intrinsically, so determining their order puts unnecessary burden on models. This issue applies to Seq2Seq models too.

So, we want a model that leverages the structure information while is capable of solving equation set problems. To this end, we propose to extract the equation as a Direct Acyclic Graph (DAG) structure upon problem description. In the bottom row of Table 1, we show the composition structure of the equation set upon problem description. It is a DAG structure. The leaf nodes are quantities and variables linked to text tokens (we append special tokens x, y, z to text as variables), the internal nodes are operations like $+, -, \times, /$, and root nodes represent equations. A leaf node may have more than one parent representing multiple occurrences in equations. The structure may have multiple roots representing multiple equations, and the number of roots depends on the problem. Thus, formulating the MWP as a DAG structure extraction problem is more general than tree to solve equation set problems, and is easier to exploit structure information than Seq2Seq approaches.

To extract the DAG, we adopt a bottom-up strategy. The bottom-up strategy is different from left-to-right or top-down strategy in Seq2Seq and Seq2Tree models. It is motivated by the abstracting process of humans when solving complex problems. Humans compose sub-expressions when reading and understanding the problem. Such sub-expressions have meanings with them, and by aggregating quantities and variables into larger meaningful terms, the problem is abstracted and simplified. As for the example in Table 1, when we see the term "in the ratio of", reflexively, we build a sub-expression x/y. Similarly, we get $1/5$ for "1:5", and $x+y$ for "their sum". Here, these sub-expressions have their meanings: x/y is the ratio of the age of Tom and his father, and $x+y$ is their sum. Then, the problem becomes "(x/y) is $(1/5)$, 1/2 of $(x + y)$ is 24", which is abstracted and becomes easier than the original one.

Based on these ideas, we propose a bottom-up DAG extraction model that extracts nodes in DAG layer by layer. Before extraction, we assume all the quantities in the problem are recognized in advance (by regular expression), and we append constants like $1, 3.14$, and variables x, y, z to the problem. These quantities, constants and variables may become the leaf nodes. Then, the layer by layer extraction procedure starts. In the first layer, we enumerate all the possible ordered triples (o, i, j), where i, j are two quantities, constants, or variables, and o is operator like $+$. These pairs are candidates to be classified. Positive candidate becomes an internal node in DAG representing a sub-expression like $i + j$. Then, in the second layer, we enumerate more new candidate pairs from quantities, constants, variables and sub-expressions from the first layer, and classify them. This process repeats until we cannot extract new expressions (detailed in Section). To classify candidates, we propose a neural network based on DAG-LSTM to represent each candidate.

Moreover, mathematical properties of math equations have been long ignored in MWP studies. For example, the commutative law for addition, and other laws like associative law. In this paper, we carefully design the model so that it behaves like the math expression that satisfies the commutative law between two nodes for operations like $+, \times$. That is to say, it will output the same hidden representation for $a + b$ and $b + a$ nodes. Such a property is not observed or discussed from existing deep learning models so far.

Experimental results demonstrate that our model outperforms existing models on two datasets `Math23K` and `DRAW1K`. Detailed analysis on model design and results are conducted to discuss the strengths and limitations of our model.

Our contributions are summarized as follows:

1. We propose a DAG extraction model for MWP that leverages the structure information of equations while is capable of solving equation set problems.
2. We propose a new bottom-up decoding strategy imitating the abstracting process of humans on solving complex problems.
3. Moreover, we are the first study to consider the math properties (the commutative law) in deep learning model design, and such design is tested to be effective.

Related Work

Research on automatically solving MWP has a long history back to the 1960s (Charniak 1969). The technique behind the solver evolves from rule-based (Bakman 2007), statistical learning (Koncel-Kedziorski et al. 2015) to deep learning.

We first introduce works based on statistical machine learning. Koncel-Kedziorski et al. (2015) generated several complete expression trees by integer programming and chose the correct one by SVM. Roy et al. (2016) predicted the expression tree by Cocke-Younger-Kasami algorithm (Cocke 1969) and SVM. They have strong assumptions like read-once and projective property.

The majority of the current deep learning based methods adopt the Seq2Seq approach. Wang et al. (2017) proposed the first neural network model for MWP. It is an encoder-decoder model. Quantities in the problem are replaced by special tokens $n_1, n_2, ...$. And the model outputs the equation like $x = n_1 + n_3 - n_4$, where n_i is copied from problem sequence. It predicts whether a quantity is used in the equation (significant number identification), and proposes a hybrid model that combines the Seq2Seq model and a retrieval model. Wang et al. (2019) extended this approach to two steps. First, the model predicts a template like n_1`<op>`n_3`<op>`n_4 using Seq2Seq model, where the operators are placeholders. Second, it predicts the operators by a recursive neural network which has the same tree structure as the template. This work indicates the importance of modeling the structure information. But Seq2Seq model is too flexible and may output spurious numbers and output at wrong positions. Thus, Huang et al. (2018) incorporated copy and alignment mechanisms to alleviate this problem. Some Seq2Seq models are able to solve multivariate problems, but models like Wang et al. (2019) cannot because they have other additional modules.

The first deep learning model that predicts the tree structure is from Xie and Sun (2019). It predicts the tree structure from root to leaf in a pre-order traverse. Then Zhang et al. (2020) extended this model from Seq2Tree to Graph2Tree by adding a graph encoder. These models outperform Seq2Seq models by a large margin. The limitation of these models is that they only work for problems with exactly one variable. Please refer to Mukherjee and Garain (2008) and Zhang et al. (2019) for a more comprehensive review.

There are some works utilizing structure information like dependency tree for classification and information extraction (Tai, Socher, and Manning 2015; Miwa and Bansal 2016). They use Tree or DAG-structured LSTM to represent the given structure. Cao et al. (2019) adopt the DAG-LSTM to represent and extract formula expressed in finance reports, where we focus on solving word problems.

Problem Statement

There are two categories of word problems (Roy and Roth 2015).

- *Arithmetic word problems* ask for the answer of one variable. The result is an expression. According to the number of operations in the expression, there are one-step problems whose expressions are like $1 - 0.5$ and multi-step problems whose expressions are like $(12 + 8)/5$.
- *Algebra word problems* require identifying variables, and form equations with these variables to solve the problem. According to the number of variables in the problem, there are univariate problems whose results should be one equation like $5 \times x = 12 + 8$, and multivariate problems or equation set problems whose results should be an equation set like $x/y = 1/5, \frac{1}{2} \times (x + y) = 24$.

Arithmetic word problems are univariate algebra word problems if we add "$x =$" on the left side of the expression. Such categorization exists because some studies focus on only one category. In this paper, we aim to solve both arithmetic and algebra MWP with one or multiple variables. Hereafter, we use "equation set" as an umbrella term for expression, equation, and equation set.

A word problem sample (P, E, A) consists of a problem description P, an equation set E and an alignment A. P is a sequence of tokens $(w_1, ..., w_m)$. We define Q_P as the quantities in P, like $5, 1/2$. Since some quantities might be referred implicitly in the problem, like $1, 2$ and π, we define C as the set of constants. Moreover, $V = \{x, y, z\}$ is the set of unknown variables. We append elements in C and V to P. For example, the problem in Table 1 becomes:

> The ages of Tom and his father are in the ratio of 1:5, 1/2 of their sum is 24. Find their ages. 1, 2, π, x, y, z.

Algorithm 1 The Solving Process

```
 1: Operators $O = \{+, -, \times, \div, =\}$
 2: Given a problem $P$, quantities $Q_P$, constants $C$, and
    variables $V$
 3: Terms set $T = Q_P \cup C \cup V$
 4: Existing candidates $D = \emptyset$
 5: repeat
 6:     $S = \{(o, i, j) \mid i, j \in T, o \in O\} - D$
 7:     $S^p = \{(o, i, j) \mid (o, i, j) \in S, \text{classify}(o, i, j) = 1\}$
 8:     $T = T \cup S^p$
 9:     $D = D \cup S$
10: until $S^p = \emptyset$
11: return $T$
```

The equation set E is a set of equations $\{E_1, E_2, \dots\}$. An example of equation set is shown in the Equations row in Table 1. We assume that each quantity and variable in E is aligned to a quantity in P after appending C and V, and that information is recorded in A. Thus, E can always be transformed to a DAG structure G as shown in Table 1. The leaf nodes of G are quantities and variables aligned in P. Each internal or root node represents a sub-expression or equation. For an internal node n, it is a triple:

$$n : (o, i, j)$$

where $o \in \{+, -, \times, /, =\}$ is an operation or comparison, i and j are operands of n which could be quantities or sub-expressions. For the example in Table 1, $n_5 : (\times, q_3, n_4)$ represents the expression $\frac{1}{2} \times (x + y)$, and $n_6 : (=, n_5, q_4)$ represents the second equation.

The task of automatic solving MWP is as follows. Given a problem P and recognized quantities in P, predict the equation set E. Our structure extraction model solves this by predicting G and then transforming it to E.

The Bottom-Up DAG Extraction Model

We now introduce our model that solves a word problem P by directly predicting the DAG structure G. Our model consists of a sequential encoder and a DAG-structured decoder. We first introduce the overall solving process on one problem, then introduce the detail of the encoder and decoder.

Solving Process

Our model predicts the structure from bottom (the quantities and variables) to top (the equations) layer by layer iteratively. This is different from Seq2Seq models which predict the equation set in a left-to-right fashion.

We maintain a *term set* T which contains all the expression terms to compose the equation set (each term corresponding to a node in G). At the beginning, $T = Q_P \cup C \cup V$, which contains all the quantities, constants and variables.

At the first layer, we enumerate all the triplets (o, i, j) to form a candidate set, where $i, j \in T$ and $o \in \{+, -, \times, /, =\}$. For each triplet, we classify it by our model. The positive triplets represent sub-expressions of one operation between two quantities or variables. Then, we update T with these positive triplets. Now, T contains quantities, variables,

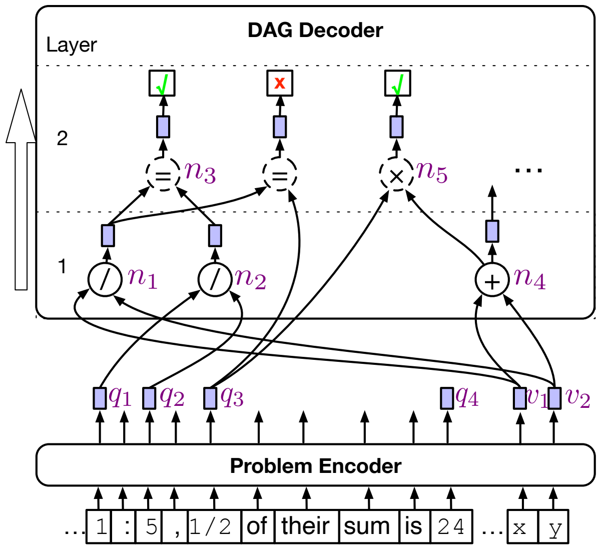

DAG Decoder
Layer
2
1
n_3
n_5
n_1
n_2
n_4
q_1 q_2 q_3 q_4 v_1 v_2
Problem Encoder
… 1 : 5 , 1/2 of their sum is 24 … x y

Figure 1: The bottom-up DAG extraction model. Each circle is a DAG-LSTM cell.

and sub-expressions with one operation. At the second layer, we enumerate triplets (o, i, j) where i, j come from the updated T. Then we classify them and update T. This process repeats until there are no positive triplets in the candidate set. The detailed process is shown in Algorithm 1. In line 6, we remove existing candidates D to avoid generating repeat candidates.

The `classify` function in line 7 Algorithm 1 is a deep neural network consisting of a problem encoder and a DAG decoder, as shown in Figure 1. In the figure, the decoder has similar structure as the equation set in Table 1. Each circle is a DAG-LSTM cell (introduced later) that corresponds to a node in the Aligned DAG row in Table 1. The cell takes as input hidden vectors from two operands and output one hidden vector as the representation of this node. Hidden vectors are shown as shaded rectangles. More details are introduced as follows.

Problem Encoder

For a problem $P = (w_1, ..., w_m)$, the encoder encodes each token as a distributed representation (a high-dimension, dense vector) containing its context information. First, each token is transformed into an embedding vector by looking up an embedding matrix or pre-trained models, so that the token sequence becomes a vector sequence $(e_1, ..., e_m)$. Then, these embeddings are fed into Gated Recurrent Unit network (GRU) (Cho et al. 2014). GRU returns a hidden vector sequence $(h_1, ..., h_m)$, where h_i as the representation of the i-th token. Each vector contains both local and contextual information of the corresponding token. The quantities in the problem are represented by the hidden vectors of the corresponding tokens. In Figure 1, the vectors of quantities and variables are shown by shaded rectangles above the problem encoder.

DAG Extraction Decoder

After obtaining the vector representation for each quantity and variable from the problem encoder, the decoder extracts the DAG structure upon these quantities and variables. The decoder model consists of DAG-LSTM cells that compose the same structure as the extracted DAG. A DAG-LSTM cell takes as input hidden vectors of two operands, and outputs a new vector. Details about DAG-LSTM are introduced in Section .

Figure 1 demonstrates the structure of the decoder when classifying the 2nd layer candidates. The candidates are shown in dashed circles in the 2nd layer. Only three of them are drawn due to the limit of space. Existing terms include quantities $q_1 \sim q_4$, variables v_1, v_2, sub-expressions n_1, n_2 and n_4. Their hidden vectors are obtained from encoder and decoder in layer 1. Suppose we are classifying n_3 which represents $n_1 = n_2$ or $x/y = 1/5$. We need a hidden representation of n_3 for classification. This is achieved by a DAG-LSTM cell with type "=", which takes as input hidden vectors of n_1 and n_2, and outputs a new vector h that represents n_3. Moreover, we use an attention module that let the candidate attends on the problem words to re-read the problem again before classification. We denote the hidden matrix of tokens from encoder as $H = (h_1, ..., h_m)$. The attention module follows the attention model described in BERT (Devlin et al. 2018; Vaswani et al. 2017):

$$c = \text{softmax}(\frac{qK^T}{\sqrt{d_h}})V \tag{1}$$

where $q = W^q h$ is the query vector, $K = W^k H$ is the key matrix, $V = W^v H$ is the value matrix, and d_h is the dimension of the hidden vectors.

We use the hidden vector h from DAG-LSTM and context vector c from the attention module for classification. Concretely, for a candidate (o, i, j), the probability of being positive is computed as follows:

$$\begin{aligned} h &= \text{DAG-LSTM}_o(h_i, h_j) \\ c &= \text{Attention}(h, H) \\ p &= \text{Softmax}(\text{FeedForward}([h; c])) \end{aligned} \tag{2}$$

where h_i and h_j are hidden vectors of nodes i and j, DAG-LSTM$_o$ is the DAG-LSTM cell with type o.

Training and Inference

During training, given the dataset $D = \{(P^{(i)}, G^{(i)}) \mid 1 \leq i \leq N\}$, the loss function is the sum of log probabilities:

$$L = \sum_{s \in D} \sum_{c \in C(s)} y_c \log p(c) + (1 - y_c)(1 - \log p(c)) \tag{3}$$

where s=(P, G) is a sample consists of problem description and equation set graph, $C(s)$ is all the candidates of s, c=(o, i, j) is a candidate, y_c is the ground truth label of c, and $p(c)$ is the probability of being positive.

Notice that, for one word problem, we store H and hidden vectors of positive candidates and predict layer by layer. So the whole structure is extracted in one pass, instead of recompute H and child node vectors for each candidate.

During inference, our model may output multiple equations as we do not constrain the number of roots in DAG. We denote the number of equations predicted as n_p, the number of variables involved as n_v. If $n_p = n_v$, we solve the equation set comprised of predicted equations. If $n_p < n_v$, we cannot solve the problem. If $n_p > n_v$, we need to select some equations to comprise the equation set. For the example shown in Table 1, the model might predict three equations: a) $x/y = 1/5$, b) $\frac{1}{2} \times (x + y) = 24$, and c) $y/x = 1/5$. Combining equations a) and b) gives a correct equation system, where x, y are the ages of Tom and his dad. Combining equations c) and b) also gives a correct equation system where y, x are the ages of Tom and his dad. We solve this conflict by choosing the solvable equation set with the highest cumulative probability. The number of equations selected is determined by how many variables are involved. We denote the positive probability that the model assigned on node n as $p(n)$, and its cumulative probability $p_c(n)$ is defined as follows:

$$p_c(n) = p(n) \times p_c(l) \times p_c(r)$$

where l, r are its left and right children. We define p_c of leaf nodes as 1.

DAG-LSTMs

In this section, we introduce the details of DAG-LSTM. Like ordinary LSTM, the basic unit of DAG-LSTM is cell. Unlike ordinary LSTM that takes as input from one previous cell, a DAG-LSTM cell accepts inputs from two children and outputs a vector for this node. This is in line with an operator in an equation set that connects its two operands to form a larger sub-expression. Many DAG-LSTM cells are connected to form the decoding structure.

Nodes having the same operator share the same DAG-LSTM cell parameters. For example, all DAG-LSTM cells that represent $+$ operator share the same parameters. Noted that, different operators in math have different properties. According to whether satisfying commutative law or not, operators are grouped into two types. Correspondingly, there are two types of DAG-LSTM cells: one type for operators like $+, *, =$ that satisfy commutative law, and the other type for operators like $-, /$ that do not satisfy commutative law. We use different types of cells to construct the model so that it behaves as similar as possible with the math equation set (here we focus on the commutative law).

For Seq2Seq models, as the model will result in different result for $a+b$ and $b+a$, some methods apply "equation normalization" which explicitly adjusts the order of operands in the equation (Wang et al. 2018) in the training data to get a consistent order. For example they rearrange the terms like $b+a$ to $a+b$ if quantities a occurs in front of b in the problem description. However, such technology is heuristic and can only change the order of quantities. In our model design, we are able to directly output the same representation of $a + b$, even a and b are sub-expressions. Next, we introduce these two types of cells.

Cells Not Satisfying the Commutative Law. For operators that do not satisfy the commutative law, different orders of left and right children should result in different results. For example, $a - b$ has different result with $b - a$. Correspondingly, we want a type of DAG-LSTM cell that outputs different hidden vectors if we change the order of its operands. We design the cell based on the N-ary Tree-LSTM cell introduced in Tai, Socher and Manning (2015) as follows. For node $n : (op, l, r)$, its operator op does not satisfy the commutative law, and its left and right operands are l, r. We denote x as the embedding of op, and set $\tilde{h}$ as the concatenation of h_l and h_r. Then the hidden vector of n is computed as:

$$\begin{aligned}
i &= \sigma\left(W^i \tilde{h} + U^i x + b^i\right) \\
f_k &= \sigma\left(W_k^f \tilde{h} + U^f x + b^f\right), \text{for } k \in (l, r) \\
o &= \sigma\left(W^o \tilde{h} + U^o x + b^o\right) \\
\hat{c} &= \tanh\left(W^c \tilde{h} + U^c x + b^c\right) \\
c &= f_l \odot c_l + f_r \odot c_r + i \odot \hat{c} \\
h &= \tanh(c) \odot o
\end{aligned} \tag{4}$$

where σ denotes the logistic function, $\odot$ denotes element-wise multiplication, W, U and b are weight matrices and bias vectors, i, f, and o are input, forget, and output gates, and h, c are hidden and cell vectors. Here, the parameters of the forget gate are different for left and right operands. Thus changing the order of operands will result in different h.

Cells Satisfying the Commutative Law. For operators that satisfy the commutation law, the DAG-LSTM cell should output the same h for both (op, l, r) and (op, r, l). We use the Child-Sum structure introduced in Tai, Socher and Manning (2015) as follows. We set $\tilde{h} = h_l + h_r$, and the hidden vector of n is computed as follows:

$$\begin{aligned}
i &= \sigma\left(W^i \tilde{h} + U^i x + b^i\right) \\
f_k &= \sigma\left(W^f h_k + U^f x + b^f\right), \text{for } k \in (l, r) \\
o &= \sigma\left(W^o \tilde{h} + U^o x + b^o\right) \\
\hat{c} &= \tanh\left(W^c \tilde{h} + U^c x + b^c\right) \\
c &= f_l \odot c_l + f_r \odot c_r + i \odot \hat{c} \\
h &= \tanh(c) \odot o
\end{aligned} \tag{5}$$

Here, the parameters of forget gate are the same for h_l and h_r, thus will output the same h even we change the order of operands.

Experiments

Datasets and Settings

We conduct experiments on two datasets, DRAW1K and Math23K:

- DRAW1K (Upadhyay and Chang 2017) contains 1,000 algebra word problems, the questions are like the example shown in Table 1. We use this dataset to demonstrate the model's ability on equation set problems. It has more than 200 equation set templates.
- Math23K (Wang, Liu, and Shi 2017) is the largest published MWP dataset with 23,161 problems. All of the

	Model	Acc.(%)
Similarity	SIM (Huang et al. 2016)	25.5*
Template	KAZB (Kushman et al. 2014)	43.2*
	MixedSP (Upadhyay et al. 2016)	59.5
Deep Learning	DNS (Wang, Liu, and Shi 2017)	31.0*
	Seq2DAG	44.4
	w/o BERT	37.5

Table 2: Results on DRAW1K.

problems are arithmetic problems that have one variable. Thus, the result of a problem is an expression like 0.8×(5+8) instead of an equation set. We use this dataset to demonstrate the model's ability on large scale arithmetic problems. It has over 2,000 expression templates.

We briefly introduce other related datasets that are not included. For algebra problems, Alg514 (Kushman et al. 2014) is a small dataset with 514 problems, similar but smaller than DRAW1K. Dolphin18K (Huang et al. 2016) is a large dataset with 18,460 problems, but it does not have quantity-equation alignment information, and it is hard to align them as the dataset is noisier comparing with DRAW1K and Math23K. For arithmetic problems, other datasets such as AllArith (Roy and Roth 2017) and MAWPS (Koncel-Kedziorski et al. 2016) are smaller than Math23K.

We use the pre-trained BERT model (Devlin et al. 2018) as the embedding layer. The dimensions of other components are set to 512. We update the last 2 layers of BERT during training, using the Adam optimizer suggested in Devlin et al. (2018) with learning rate 5e-5. The batch size is set to 64. We set the dropout rate to 0.5 to prevent overfitting. The result is reported using 5-fold cross-validation. We split the data following Xie and Sun (2019) for Math23K, and Upadhyay and Chang (2017) for DRAW1K. Our model is implemented based on Pytorch (Paszke et al. 2019). All the experiments are conducted on a NIVIDIA 1080Ti GPU.

The predicted equation set is fed into an equation set solving tool to get the answer, i.e. result of variables. We regard the answer of multiple variables as a set. If the answer sets of annotation and prediction are the same, the prediction is correct. Otherwise the prediction is wrong. We do not ask to predict the same equation set structure as there might be more than one correct equation set.

Results on DRAW1K

The results on DRAW1K are reported in Table 2. Numbers with "*" are not reported by the original paper but by the survey (Zhang et al. 2019). MixedSP utilizes external weak supervision to get a 59.5% accuracy, so we should not compare with this result. Then, there is only one deep learning(DL)-based model on this dataset except ours. This is because most DL models focus on arithmetic problems, and have difficulty on algebra problems that output equation set. Our model is a novel DL-based model capable of algebra problems. It achieves 13.4% absolute improvement compared with DNS, it also outperforms similarity and template-based models. If we replace BERT with ordinary embedding (an embedding matrix E where E_i is the embedding of token i), the result is still better than DNS by 6.5%.

Our model on DRAW1K is flexible and is able to extract arbitrary numbers of equations on one word problem. If the model predicts more equations than needed, we use the method introduced in Section to select proper equations.

Model	Acc.(%)
DNS (Wang, Liu, and Shi 2017)	64.7
T-RNN (Wang et al. 2019)	68.7
Ensemble (Wang et al. 2018)	68.4
Seq2Tree (Xie and Sun 2019)	74.3
Graph2Tree (Zhang et al. 2020)	75.5
Seq2DAG	**77.1**
−Attention	76.1
−BERT	72.5

Table 3: Results on Math23K

Results on Math23K

The results on Math23k are reported in Table 3. All the results we can find are deep learning-based. The first three models are Seq2Seq models, whose results are less than 70%. Seq2Tree (Xie and Sun 2019) is the first deep learning model that does not follow the Seq2Seq approach. It achieves a high accuracy which demonstrates the power of structured output approach. Graph2Tree model further improve the result to 75.5%. Our model achieves a better performance with 1.6% absolute improvement over Graph2Tree model. Following Zhang et al. (2020), We also conduct a one-sample t-test to compare the results of our full model on 5-fold cross-validation with the average score of Graph2Tree, the p-value is 0.013, which shows that our improvement is statistically significant. Note that while Seq/Graph2Tree adopting a top-down decoding strategy, our model adopts a bottom-up strategy. In the ablation study, removing attention module during classification decreases the performance by 1.0%, removing BERT decreases the performance by 4.6%.

Although Math23K is an arithmetic problem dataset where each problem has only one variable and one expression, the model might output multiple expressions for one problem. We choose the expression with the highest cumulative probability as the result. In our prediction, there are 4.1% of problems where the model predicts multiple expressions and the expression with the highest probability is correct. There are 2.1% of problems that the model predicts multiple expressions including the correct one but not with the highest probability. Moreover, there are 1.2% problems that the model predicts multiple correct expressions. This shows that the model will predict any expressions that are correct, instead of memorizing the similar problems by rote. This is the most significant difference between DL-based and template-based models. But as far as we know, such results have never been reported before, since Seq2Seq and Seq2Tree models only output one result.

Per problem ↓	DRAW1K	Math23K
# of internal nodes	10.7	5.2
# of candidates	395.7	93.1
inference time w/ BERT (ms)	53.1	40.0
inference time w/o BERT (ms)	36.2	16.4

Table 4: Average inference time per word problem.

	Acc. (%)
Consider CL	44.4
Not consider CL	41.9
Not consider CL & order disruption	41.0

Table 5: Results on whether considering the commutative law (CL).

Efficiency

To show the efficiency of our model, we report the statistics about the running time of our model in Table 4. Although there are hundreds of candidates per problem, candidates in the same layer are computed in parallel. The longest average inference time is 53.1ms, indicating that our model is able to solve MWP problems in real time. Moreover, we can see that problems in DRAW1K have more internal nodes and candidates than Math23K, which indicates that algebra word problems are harder than arithmetic problems on the aspect of the output structure and the decoding space. Thus, more effort is needed on algebra problems.

Model Design on the Commutative Law

In previous studies, such as Seq2Seq and Seq2Tree models, they do not take mathematical properties into consideration during generating the equation set. To the best of our knowledge, we are the first work that concern about this. Specifically, in Section , we introduce two types of DAG-LSTM cells: cells that satisfy the commutative law and cells that do not, corresponding to two types of operators $\{+, \times, =\}$ and $\{-, /\}$. We examine the effectiveness of our model design by allowing all cells to not satisfy the commutative law, using Equation 4. Thus, the representation of $a + b$ is different from $b + a$. The experiments are carried out on DRAW1K dataset, and the result is shown in the "Not consider CL" row in Table 5. The performance drops by 2.5% compared with "Consider CL" row that consider the commutative law. Moreover, we deliberately modified the training data by randomly change the order of the operands if the operator is "+", "×" or "=" and the operands are all quantities. This is to simulate the situation that during annotation, the order of operands of "+", "×" and "=" are answered randomly. The result is shown in the "not consider CL & order disruption" row in Table 5. The performance drops further by 0.9%. These experiments show that our model design that considers the commutative law is reasonable and effective.

Case Study

We use three cases in Table 6 to demonstrate our model's strengths and limitations. In the "prediction" rows, parentheses are used to enclose the terms in each node.

Case 1. A park has 26 boats, and the rent revenue is $910 per day. If we add 6 more boats, how much rent the park will get each day? Ground truth: $910/26 \times (26 + 6)$ Prediction : $(910/26) \times (26+6); 910+((910/26) \times 6)$
Case 2. Ben is 3 years younger than Dan. The sum of their ages is 53. How old is each? Ground truth: $y = x + 3, \quad x + y = 53$ Prediction : $y = (x - 3), (x + y) = 53$
Case 3. In Longgang orchard, the number of apple trees is (2/5) of pear trees, and (3/4) of peach trees. There are 480 pear trees in total. How many peach trees are there? Ground truth: $480 \times \frac{2}{5} / \frac{3}{4}$ Prediction : $(480 \; / \; \frac{2}{5})/\frac{3}{4}$ (wrong)

Table 6: Cases study.

In Case 1, the model predicts two different expressions that are both correct. The first one multiplies the rent of one boat to the total number of boats which is the same as the ground truth. The second one sums the rent of 26 boats and 6 new boats. We search through the Math23K dataset, and find two similar problems with questions "how much **more** rent the park will get each day". Equations of such questions will only compute the rent of new boats, which is later part of the second expression in predictions. The model differentiates these problems and learns the correct structures.

In Case 2, the model sets x as the age of Dan while x is the age of Ben in the ground truth. Although the model predicts a different equation set with ground truth, they yield the same answer.

In Case 3, the model predicts the multiplication between 480 and 2/5 as division. We think this is because the model is not able to correctly distinguish among three entities: apple, pear and peach trees. It wrongly recognizes the order of apple and pear trees in text and predicts the result of "the number of pear trees is (2/5) of apple trees". This indicates that the ability of coreference resolution is important for math word problems. Without such ability, the model cannot achieve a comparable performance like a human, especially on adversarial problems.

Conclusion and Discussion

We propose a novel Seq2DAG approach to solve math word problems. The model extracts the DAG structure of the equation set in a bottom-up fashion. This bottom-up strategy is suitable to solve multivariate algebra problems and its structure satisfies the commutative law for nodes with operations like +. The model outperforms previous deep learning-based models on two datasets. There are several future works to further improve this model. For example, extend the model to satisfy more mathematical laws such as association law; design better decoder cells based on transformer; combine with better encoders such as the graph encoder from Graph2Tree.

Acknowledgments

The research work supported by the National Key Research and Development Program of China under Grant No. 2017YFB1002104, the National Natural Science Foundation of China under Grant No. 62076231, U1811461.

References

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

Bobrow, D. G. 1964. Natural Language Input for a Computer Problem Solving System. *AI Technical Reports* .

Cao, Y.; Chen, D.; Li, H.; and Luo, P. 2019. Nested Relation Extraction with Iterative Neural Network. In *CIKM*.

Charniak, E. 1969. Computer Solution of Calculus Word Problems. In *IJCAI*.

Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *EMNLP*.

Cocke, J. 1969. *Programming languages and their compilers: Preliminary notes*. New York University.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Huang, D.; Liu, J.; Lin, C.-Y.; and Yin, J. 2018. Neural Math Word Problem Solver with Reinforcement Learning. In *COLING*.

Huang, D.; Shi, S.; Lin, C. Y.; Yin, J.; and Ma, W. Y. 2016. How well do computers solve math word problems? Large-scale dataset construction and evaluation. In *ACL*.

Koncel-Kedziorski, R.; Hajishirzi, H.; Sabharwal, A.; Etzioni, O.; and Ang, S. D. 2015. Parsing Algebraic Word Problems into Equations. In *TACL*.

Koncel-Kedziorski, R.; Roy, S.; Amini, A.; Kushman, N.; and Hajishirzi, H. 2016. MAWPS: A Math Word Problem Repository. In *NAACL*.

Kushman, N.; Artzi, Y.; Zettlemoyer, L.; and Barzilay, R. 2014. Learning to automatically solve algebra word problems. In *ACL*.

Miwa, M.; and Bansal, M. 2016. End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. In *ACL*.

Mukherjee, A.; and Garain, U. 2008. A Review of Methods for Automatic Understanding of Natural Language Mathematical Problems. *Artif. Intell. Rev.* .

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeuralPS*.

Roy, S.; and Roth, D. 2015. Solving general arithmetic word problems. In *EMNLP*.

Roy, S.; and Roth, D. 2017. Unit Dependency Graph and its Application to Arithmetic Word Problem Solving. In *AAAI*.

Roy, S.; Upadhyay, S.; and Roth, D. 2016. EQUATION PARSING : Mapping Sentences to Grounded Equations. In *EMNLP*.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *ACL*.

Upadhyay, S.; and Chang, M. W. 2017. Annotating derivations: A new evaluation strategy and dataset for algebraword problems. In *EACL*.

Upadhyay, S.; Chang, M. W.; Chang, K. W.; and Yih, W. T. 2016. Learning from explicit and implicit supervision jointly for algebra word problems. In *EMNLP*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS*.

Wang, L.; Wang, Y.; Cai, D.; Zhang, D.; and Liu, X. 2018. Translating a Math Word Problem to an Expression Tree. In *EMNLP*.

Wang, L.; Zhang, D.; Zhang, J.; Xu, X.; Gao, L.; Dai, B. T.; and Shen, H. T. 2019. Template-Based Math Word Problem Solvers with Recursive Neural Networks. In *AAAI*.

Wang, Y.; Liu, X.; and Shi, S. 2017. Deep neural solver for math word problems. In *EMNLP*.

Xie, Z.; and Sun, S. 2019. A Goal-Driven Tree-Structured Neural Model for Math Word Problems. In *IJCAI*.

Zhang, D.; Wang, L.; Zhang, L.; Dai, B. T.; and Shen, H. T. 2019. The Gap of Semantic Parsing: A Survey on Automatic Math Word Problem Solvers. *TPAMI* .

Zhang, J.; Wang, L.; Lee, R. K.-W.; Bin, Y.; Wang, Y.; Shao, J.; and Lim, E.-P. 2020. Graph-to-Tree Learning for Solving Math Word Problems. In *ACL*.