Template-Based Math Word Problem Solvers with Recursive Neural Networks

Lei Wang,[1] Dongxiang Zhang,[1,2*] Jipeng Zhang,[1] Xing Xu,[1,2] Lianli Gao,[1] Bing Tian Dai,[3] Heng Tao Shen[1]
[1]Center for Future Media and School of Computer Science & Engineering, UESTC
[2]Afanti Research
[3]School of Information Systems, Singapore Management University
{demolwang,zhangjipeng20}@std.uestc.edu.cn, {zhangdo,xing.xu,lianli.gao}@uestc.edu.cn
btdai@smu.edu.sg, shenhengtao@hotmail.com

Abstract

The design of automatic solvers to arithmetic math word problems has attracted considerable attention in recent years and a large number of datasets and methods have been published. Among them, Math23K is the largest data corpus that is very helpful to evaluate the generality and robustness of a proposed solution. The best performer in Math23K is a seq2seq model based on LSTM to generate the math expression. However, the model suffers from performance degradation in large space of target expressions. In this paper, we propose a template-based solution based on recursive neural network for math expression construction. More specifically, we first apply a seq2seq model to predict a tree-structure template, with inferred numbers as leaf nodes and unknown operators as inner nodes. Then, we design a recursive neural network to encode the quantity with Bi-LSTM and self attention, and infer the unknown operator nodes in a bottom-up manner. The experimental results clearly establish the superiority of our new framework as we improve the accuracy by a wide margin in two of the largest datasets, i.e., from 58.1% to 66.9% in Math23K and from 62.8% to 66.8% in MAWPS.

Introduction

Developing computer systems to automatically solve math word problems (MWPs) has been studied by NLP researchers since the 1960s (Feigenbaum and Feldman 1963; Bobrow 1964). It requires mapping the human-readable words into machine-understandable logic forms, followed by an inference procedure to derive the numeric answer. An example of math word problem is illustrated in Figure 1. The unknown variable x refers to the weight of an apple and the math expression involves basic arithmetic operators such as $-$ and $/$. To solve the problem, we need to identify the relevant quantities from the text and determine the correct operators and computation order among these numbers.

Previous research efforts have been mainly based on statistical machine learning (Kushman et al. 2014; Amnueypornsakul and Bhat 2014; Zhou, Dai, and Chen 2015; Mitra and Baral 2016; Roy and Roth 2018) and semantic parsing (Shi et al. 2015; Koncel-Kedziorski et al. 2015; Roy and Roth 2015; Huang et al. 2017). These methods can achieve satisfactory results on small-scale datasets, yet they require considerable manual efforts for feature extraction and template annotation. Furthermore, as evaluated by (Huang et al. 2016), these methods exhibit low generality and robustness. They suffer from sharp performance degradation when handling large and diversified datasets.

Problem : A basket of apples has a mass of 19kg. After picking 6 apples, the same basket with remaining apples weighs 10kg. What is the weight of an apple?
Equation : (19 - 10) / 6 = x **Ans :** 1.5
Position Symbol : {n_1: 19, n_2: 6, n_3: 10}
Mapped Expression : (n_1 - n_3) / n_2
Mapped Post Expression : $n_1\ n_3 - n_2/$
Mapped Expression w/o operator : (n_1 <op> n_3) <op> n_2
Post Expression w/o operator : $n_1\ n_3$ <op> n_2 <op>

Figure 1: An example of math word problem.

To reduce human intervention and enable the automatic extraction of discriminative features, applying deep learning (DL) models in MWPs has become a promising research direction. (Wang, Liu, and Shi 2017) proposed an end-to-end framework that converts the input of question text into the output of math expression. It is then a natural idea to apply a seq2seq model to encode the text input and decode the hidden features into a math expression. The drawback is that the seq2seq model is a black-box that lacks interpretability and it cannot guarantee the output is in valid math format and normally requires a post-processing step. (Wang et al. 2018b) modeled the math expression as a tree structure and made the first attempt to apply deep reinforcement learning for iterative tree construction. However, it still requires manual feature extraction to design the state representation.

In this paper, we propose a new math problem solver that combines the merits of (Wang, Liu, and Shi 2017) and (Wang et al. 2018b), i.e., we apply deep neural networks for discriminative feature extraction and leverage the concept of expression tree for answer generation. In this way, we can avoid human intervention and leverage expression

*Corresponding Author: Dongxiang Zhang

tree for better interpretability as the tree construction process corresponds to the step-by-step solution procedure. In our implementation, we first apply a seq2seq model to predict a tree-structure template, with quantities as leaf nodes and unknown operators as inner nodes. As shown in Figure 1, $\underline{(n_1\ \langle op\rangle\ n_3)\ \langle op\rangle\ n_2}$ is a template in which the quantities n_1, n_2 and n_3 can be directly determined from the question text. The operators are encapsulated as $\langle op\rangle$ to reduce template space. Each equation template can be represented by an expression tree that can be further serialized in the form of suffix expression. We also propose equation normalization to further reduce the number of possible templates. Consequently, our task of template prediction can be considered to be less challenging than the direct generation of math expression as in (Wang, Liu, and Shi 2017). With the derived template, the remaining job is to fill the unknown operators. We design an effective quantity embedding network with Bi-LSTM and self attention to vectorize the tree leaf nodes. Then, we propose a Recursive Neural Network (abbreviated as Recursive NN) to infer the inner nodes in a recursive manner. When all the operator nodes are predicted, we obtain a complete expression to generate the answer.

We conduct experiments on two of the largest datasets for arithmetic word problems, in which **Math23K** contains $23,164$ math problems and **MAWPS** contains $2,373$ problems. The experimental results clearly establish the superiority of our new framework when compared to other seq2seq models. The accuracy is boosted from 58.1% to 66.9% in Math23K and from 62.8% to 66.8% in MAWPS.

To sum up, we have made the following contributions:

1. We use structural template, which can be serialized into suffix expression, to annotate the math problems.
2. We propose equation normalization and operator encapsulation to significantly reduce the template space.
3. We made the first attempt to apply Recursive NN in answer module for MWP solving.
4. Experimental results show that our proposed framework is remarkably better than the state-of-the-art models.
5. We release the source code of our model in Github[1].

Related Work

In this section, we review literature upon MWP solver and present the applications of Recursive NN in natural language processing.

Algebra Word Problem Solver

ARIS (Hosseini et al. 2014) and Formula (Mitra and Baral 2016) are two systems that only support the operators of addition and subtraction. The former used verb categorization and the latter utilized formulas defined in advance. To improve the generality, tag-based approach (Liang et al. 2016) was proposed with map rules to convert identified variables and values into logic forms, which were further transformed into logic statements for inference.

[1]https://github.com/uestc-db/T-RNN

Another major category of solutions is tree-based. (Roy and Roth 2015) built an expression tree and two classifiers were trained for quantity relevance prediction and operator classification, respectively. A scoring function was proposed to rank the candidate trees and champion the one with the highest score as the answer. (Koncel-Kedziorski et al. 2015) enumerated all the possible trees with integer linear programming to find the best solution. UnitDep (Roy and Roth 2017) further took into account the consistence of rate unit associated with the quantities and treated it as a scoring factor. (Wang et al. 2018b) modeled the math expression as a tree structure and made the first attempt to apply deep reinforcement learning for iterative tree construction. (Wang, Liu, and Shi 2017) was the first to propose a seq2seq model to convert the input of question text into the output of math expression. (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018) applied Bidirectional LSTM w or w/o self attention to encode the problem text and used a softmax function as the classifier for equation template. The model can be superior to seq2seq models in small datasets with fewer number of templates.

The equation set problems are more challenging as they involve multiple unknown variables to resolve. In (Kushman et al. 2014), a template-based solution was proposed. Given a corpus of predefined equation sets with unknown slots for variables and numbers, it finds a matching template and infers the unknown slots from text information. In (Zhou, Dai, and Chen 2015), an improved method was proposed to reduce the hypothesis space by only enumerating the permutation of number slots. Without any inference capability, these template-based methods are rather rigid and not extensible for complex scenarios. Upadhyay et al. proposed a structured-output learning framework to learn both explicit and implicit signals jointly (Upadhyay et al. 2016). Overall, these methods are tuned for small datasets to achieve promising results. According to an experimental study in (Huang et al. 2016), their accuracies degrade sharply in a larger and more diverse dataset. The findings imply that this line of research still has great room for improvement and calls for more general and robust solutions. For more comprehensive review on MWP solvers, readers can refer to a recent survey paper (Zhang et al. 2018).

NLP Applications for Recursive Neural Networks

Recursive NNs (Goller and Kuchler 1996) have been widely applied in syntactic parsing which is based on tree structure. They can naturally process the tree nodes in a recursive order. (Socher, Manning, and Ng 2010) introduced a context-aware Recursive NN to learn vector space representations for variable-sized inputs and predict the phrase structure. (Socher et al. 2012) introduced a Recursive NN model to learn compositional vector representations for phrases and sentences. They assigned a vector and a matrix to every node in a parse tree with the purpose of capturing the semantic compositionality. (Socher et al. 2013) proposed a recursive neural tensor network for understanding compositionality sentiment detection.

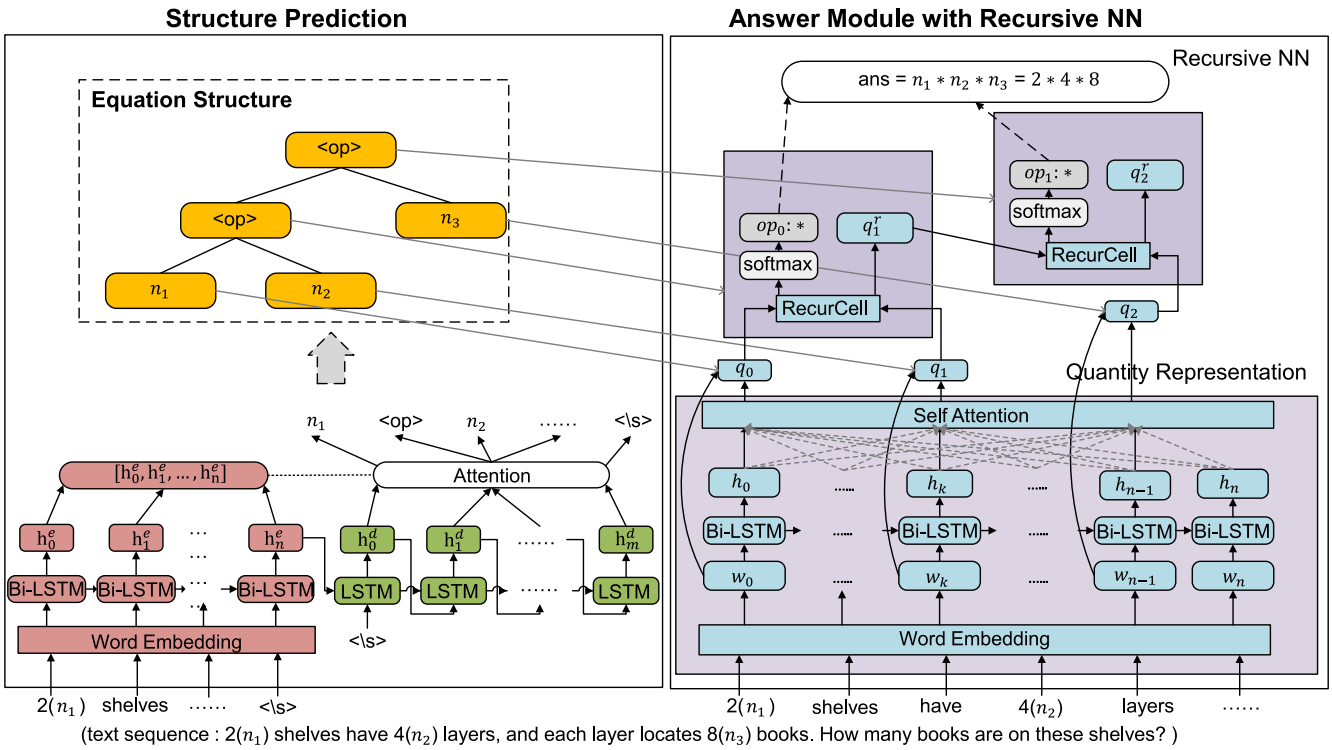

Figure 2: Framework of MWP solver with Recursive NN.

MWP Solver with Recursive Neural Network

In this section, we present the new MWP solver based on Recursive NN. The whole framework, as illustrated in Figure 2, consists of two primary components. The first component, namely *structure prediction*, converts the input text into a tree-structural template that is serialized by suffix expression. The leaf nodes are quantities whose values can be directly inferred from the problem text. The inner nodes contain unknown operators that are to be predicted by the answer generation module illustrated in the right side of Figure 2. It contains a quantity embedding network to vectorize the quantities more effectively. The network is constructed by Bidirectional LSTM followed by a self attention layer. After that, we design a recursive neural network to predict the operators by taking advantage of the derived quantity features and structural template. When all the operators have been successfully determined, we obtain a complete math expression which is able to return the final answer. In the following, we introduce the structural template representation and explicitly explain each function module in Figure 2.

Equation Template

An arithmetic word problem is often solved by an equation with one unknown variable, a number of quantities and basic operators $\{+, -, \times, \div\}$. Brackets can also be used to change the operation priority. In the template-based solutions, the problems are annotated with equation templates and models are designed to learn the mapping. In (Kushman et al. 2014) and (Zhou, Dai, and Chen 2015), templates with multiple unknown slots, such as

$$\begin{aligned} u_1 + u_2 + n_1 &= 0 \\ n_2 \times u_1 + n_3 \times u_2 - n_4 &= 0 \end{aligned}$$

are pre-defined to solve equation set word problems. In their solutions, they first identify a candidate template and then fill the number slots and unknown slots with the information extracted from the text.

Our structural template has three main differences with their definition. First, since we are handling arithmetic word problems which involve only one unknown variable, there is no need to preserve the unknown slot in the template. Given an equation template $x = n_1 - n_2 - n_3$, we simplify it as $n_1 - n_2 - n_3$. As to constants in the equation templates (e.g., 1 and π), we use additional vectors to represent them, with the same dimension as other quantity vectors n_i. These vectors will be trained by Recursive NN to derive the final representation. Second, our n_i strictly refers to the i-th detected quantity from the question text. This is because our template is directly generated by the seq2seq model and we need to be able to automatically ignore the irrelevant numbers in the text. Given an equation template $n_1 - n_3$, we know that the second quantity n_2 in the text is irrelevant and the solution is the difference between the first and third quantities. Hence, there is a straightforward quantity alignment between the template and question text. Our strategy runs much faster than (Kushman et al. 2014; Zhou, Dai, and Chen 2015) as they need to examine all the

possibilities of quantity filling in a given template. Third, we encapsulate the detailed operators in the template with an abstract representation of $\langle op \rangle$. For example, $n_1 + n_2$, $n_1 - n_2$, $n_1 \times n_2$, and $n_1 \div n_2$ are mapped to the same template $n_1 \langle op \rangle n_2$. The operator encapsulation can further reduce the number of equation templates to facilitate template prediction. It motivates us to break the math problem solving into two stages with better interpretability, i.e., we can first predict an equation template and then fill the unknown operators.

Equation Normalization

It is possible that a math problem can be solved by multiple choices of equations. For example, the equations $x = 10 - 5 - 2$ and $x = 10 - (5 + 2)$ are essentially identical and lead to the same answer, but they correspond to different equation templates. To reduce the variety of equation templates, we propose a normalization approach. A side product is that the template prediction model can benefit from the reduction of equation space.

There are two types of common equation duplication that we attempt to resolve in this paper: 1) *order duplication* such as $n_1 + n_3 + n_2$ and $n_1 + n_2 + n_3$, and 2) *bracket duplication* such as $n_1 + n_3 - n_2$ and $n1 + (n_3 - n_2)$. To normalize the *order-duplicated* templates, we require that the number tokens n_i in the template should follow their occurrence order in problem text as much as possible. For example, the equation templates $\{n_1 + n_3 + n_2, n_2 + n_1 + n_3, n_2 + n_3 + n_1, n_3 + n_1 + n_2, n_3 + n_2 + n_1\}$ should be normalized to $n_1 + n_2 + n_3$. To solve the *bracket duplication* problem, we simply convert it into an expression tree. The inner nodes are operators and the leaf nodes are quantities. The tree structure determines the calculation priority and there are no brackets involved in the tree nodes. Up to here, we can use expression tree to uniformly represent equation templates w or w/o brackets. Finally, each tree-based template can be serialized to its suffix expression which is stored as a string sequence. It is worth noting that the proposed equation normalization cannot guarantee the uniqueness of the equation template. For example, $n_1 - n_2 - n_3$ and $n_1 - (n_2 + n_3)$ are still represented by different expression trees.

Figure 3 depicts an example of equation normalization. Given an equation $x = n_3 - (n_2 + n_1)$, it is simplified as $n_3 - (n_2 + n_1)$ and then re-ordered as $n_3 - (n_1 + n_2)$ to eliminate order duplication. Then, an expression tree is built to remove the brackets. The final template is represented by the suffix expression "$n_3 n_1 n_2 + -$", which can be naturally processed with stack operation. The first three quantities are first pushed into the stack. When the operator "+" is met, the top two quantities are popped and the value of $n_1 + n_2$ is pushed back to the stack. Finally, when "$-$" arrives, the top two quantities are popped and $n_3 - (n_1 + n_2)$ is calculated.

Template Prediction

We propose a seq2seq model based on popular recurrent neural networks (abbreviated as Recurrent NN, with instances like LSTM, GRU and Bi-LSTM) to transform the problem text into a suffix expression which corresponds to a tree structure. The goal of the seq2seq model

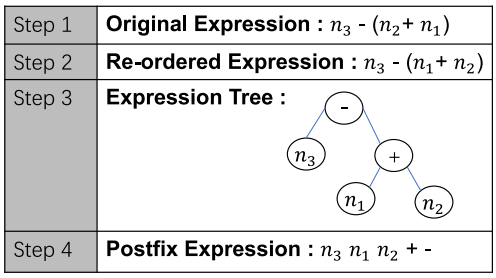

Figure 3: An example of equation normalization.

is to estimate the conditional probability $P(Y|X)$, where $X = \{x_1, x_2, \cdots, x_n\}$ is an input sequence and $Y = \{y_1, y_2, \cdots, y_m\}$ is its corresponding output sequence.

To yield this conditional probability, we first obtain the fixed-dimensional representation vector of the input problem description $\{x_1, x_2, \cdots, x_n\}$ given by the last hidden state vector h^e of RNN, and then compute the probability of $\{y_1, y_2, \cdots, y_m\}$ as below :

$$p(y_1, y_2, \cdots, y_m | x_1, x_2, \cdots, x_n) = \prod_{t=1}^{m} p(y_t | h^e, y_1, \cdots, y_{t-1})$$

where $p(y_t | h^e, y_1, \cdots, y_{t-1})$ distribution is represented by a softmax function over all the vocabulary candidates.

In practice, there is no clear preference for the selection of Recurrent NNs. We have witnessed the wide application of LSTM, GRU and Bi-LSTM in various seq2seq models. For example, (Wang, Liu, and Shi 2017) used LSTM as the encoder and GRU as the decoder for math problem solving. Alternatively, (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018) evaluated the adoption of LSTM as the encoder and decoder at the same time. A proper selection may often require significant engineering efforts and hyper-parameter tuning. In our implementation, we use Bi-LSTM as the encoder and LSTM as the decoder because Bi-LSTM can capture the term dependency in both directions.

The output of the seq2seq model is passed to a sequence-to-sequence attention layer to better capture the relationship between the question words and the equation template.

$$\begin{aligned} \beta_{ij} &= h^d_{i-1} \odot h^e_j \\ \alpha_{ij} &= \frac{exp(\beta_{ij})}{\sum_{k=1}^{n} exp(\beta_{ik})} \\ c_i &= \sum_{j=1}^{n} \alpha_{ij} h^e_j \\ h^d_i &= f(h^d_{i-1}, c_i, y_{i-1}) \end{aligned}$$

where β_{ij} is the attention weight of every encoder vector. We compute a new context vector at each decoding step. First, with a dot product operation $h^d_{i-1} \odot h^e_j$, we compute a score for each hidden state vector h^e_j of the encoder. Then, we normalize the sequence of α_i using a softmax and compute c_i as the weighted average of the h^e_j. Subsequently, we combine

c_i with the previous hidden state vector h^d_{i-1} and output y_{i-1} by a non-linear function to obtain the hidden vector h^d_i.

It is noticeable that similar to the issue raised in (Wang, Liu, and Shi 2017; Wang et al. 2018a), the output of our seq2seq model may not be a valid equation template, i.e., it cannot be converted into a tree in which the inner nodes are operators and the leaf nodes are quantities. If the template prediction fails and an invalid template is generated, we terminate the math problem solver and consider that the problem cannot be solved by our model. From the experimental results, we will see that our seq2seq model can generate valid templates most of the time. The failure rate in Math23K is only 0.7%.

Answer Generation with Recursive NN

The answer generation module consists of two function components. First, we use Bi-LSTM with self attention mechanism to derive effective quantity embedding. Second, given the new quantity features, a recursive neural network is proposed to infer the unknown operators in the template. In the following, we explain how each component works.

The input to the quantity embedding layer is the raw question words. We have a vocabulary V. The given input sequence $X = \{x_1, x_2, \cdots, x_n\}$ is transformed into a sequence of vectors $\{w_1, w_2, \cdots, w_n\}$ through a word embedding matrix E for linear projection, i.e., $w_t = Ex_t$. The word vectors w_t act as the input of Bi-LSTM to generate hidden features $H = \{h_1, h_2, \cdots, h_n\}$. Note that the word embedding and Bi-LSTM encoder in the answer generation module are similar to those in the template prediction module. The difference lies in how to further process H. In the template prediction, H is decoded by LSTM and an attention layer to generate the suffix expression. Here, the objective is to generate effective embeddings for the quantities. Given the output H from Bi-LSTM, each h_i actually corresponds to the i-th term w_i in the question text. From the equation template, we are aware of the positions where the quantities are located. In other words, we can extract $H^q = \{h^q_1, h^q_2, \cdots, h^q_l\}$ from H based on the position information of n_i in the equation template. With H and H^q, we conduct self attention (Lin et al. 2017) that is able to capture the the long distance dependencies.

$$\begin{aligned} A &= \text{softmax}(H^q H^T) \\ C &= AH \\ Q^{'} &= \tanh([C, H^q]W_1 + b) \end{aligned}$$

where H^T is the transpose of H. Matrix A is the attention weight matrix computed by the multiplication of H^q and H^T. Then, the context vector C is the weighted sum of H. These two vectors C and H^q are concatenated to generate the temporary quantity representation $Q^{'}$ through non-linear projection. Finally, the new quantity representation is computed by

$$Q = W_q \oplus Q^{'}$$

where W_q are the quantity embeddings extracted from text embeddings W.

With the new quantity vectors $Q = \{q_1, q_2, \cdots, q_l\}$ with l quantities and the suffix expression generated from template prediction, we are ready to introduce how recursive NN is applied to infer the unknown operators. Note that the suffix expression in fact determines the access order of leaf nodes which is in a bottom-up manner. Given the expression $\underline{(n_1\ \langle op\rangle\ n_3)\ \langle op\rangle\ n_2}$, n_1 and n_3 will be first accessed by the recursive NN to determine the operator and generate a new quantity n^1_c as their parent node. Then, n^1_c and n_2 are accessed to determine their operator. When the two unknown operators are determined, we obtain a complete math expression that allows us to calculate the answer.

In our recursive NN, the representation of parent node is calculated by

$$q_c = \tanh(W_2([q_l, q_r] + b)$$

where q_l and q_r are the quantity representation for the child nodes. With q_c, we can estimate the probability of an operator for the parent node via a softmax function:

$$P(o_c|q_l, q_r) = \text{softmax}(W_3 q_c)$$

This process is performed recursively until all the operators of inner nodes have been predicted.

In the training of the answer generation module, the parameters in the answer generation module, including Bi-LSTM, self attention and recursive NN, are trained jointly by minimizing the loss function

$$J(\theta) = -\frac{1}{k}\sum_{i=1}^{k} \log P(o_c(i)|q_l(i), q_r(i))$$

where k denotes the number of inner nodes in the tree.

Experimental Study

In this section, we conduct experiments on two of the largest datasets for arithmetic word problems. All the experiments were conducted on the same server, with 4 CPU cores (Intel Xeon CPU E5-2650 with 2.30GHz) and 32GB memory.

Datasets

1. **MAWPS** (Koncel-Kedziorski et al. 2016) is another testbed for arithmetic word problems with one unknown variable in the question. Its objective is to compile a dataset of varying complexity from different websites. Operationally, it combines the published word problem datasets used in (Hosseini et al. 2014; Kushman et al. 2014; Koncel-Kedziorski et al. 2015; Roy and Roth 2015). There are $2,373$ questions in the harvested dataset.
2. **Math23K** (Wang, Liu, and Shi 2017). The dataset contains Chinese math word problems for elementary school students and is crawled from multiple online education websites. Initially, $60,000$ problems with only one unknown variable are collected. The equation templates are extracted in a rule-based manner. To ensure high precision, a large number of problems that do not fit the rules are discarded. Finally, $23,162$ math problems are remained.

The statistics of the two datasets are shown in Table 1. We report the number of templates w or w/o equation normalization (abbreviated as EN). We can see that when EN is applied, the number of templates in Math23K is reduced from $3,527$ to $2,302$, by around 35%.

Table 1: Statistics of datasets.

	MAWPS	**Math23K**
# questions	2,373	23,162
# templates w/o EN	344	3,527
# templates w/ EN	311	2,302
# sentences	6.3K	70.1K

Parameter Setting

In the template prediction module, we use a pre-trained word embedding with 128 units, a two-layer Bi-LSTM with 256 hidden units as encoder, a two-layer LSTM with 512 hidden units as decoder. As to the optimizer, we use Adam with learning rate set to $1e^{-3}$, $\beta_1 = 0.9$ and $\beta_2 = 0.99$. In the answer generation module, we use a embedding layer with 100 units, a two-layer Bi-LSTM with 160 hidden units. SGD with learning rate 0.01 and momentum factor 0.9 is used to optimize this module. In both components, the number of epochs, mini-batch size and dropout rate are set 100, 32 and 0.5 respectively. Since Math23K has split the problems into training and test datasets when it was published, we simply follow its original setup. For MAWPS, we use 5-fold cross validation.

Accuracy of MWP Solving

We compare our structural template based Recursive NN with recent deep learning models proposed for the large-scale datasets. These methods can be classified into two categories based on either generation model or classification model. The former relies on a seq2seq model to directly generate the math expression. The latter first classifies the problem text into one of the pre-defined templates with explicit operators and then fills the unknown numeric variables. DNS (Wang, Liu, and Shi 2017) was the first seq2seq generation model proposed for arithmetic word problem solving in Math23K. In (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018), both generation and classification models are evaluated in MAWPS and Math23K. The authors implemented two types of their own generation models as the baselines, using LSTM and CNN, respectively. Their classification models are implemented using Bi-LSTM w or w/o self attention. The experimental results are reported in Table 2. As pointed out in (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018), not all the test equation templates appear in the training dataset and the model accuracy is upper bounded by an oracle accuracy.

Among the seq2seq generation models, DNS performs much better than simple LSTM or CNN. One of the reasons is that LSTM and CNN are used as baseline models in (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018) and may not be well optimized. In our offline experiment, we implemented our own version of LSTM-based seq2seq model and can achieve an accuracy of 50.1% in MAWPS without much tuning effort. MathDQN (Wang et al. 2018b) achieves slightly higher accuracy than seq2seq models in MAWPS. Similar to T-RNN, it also relies on the concept of expression tree for answer generation, which has been shown to be robust and widely adopted by previous solvers. The difference is that MathDQN applies DQN to guide tree construction. Its performance in the Chinese dataset Math23K is not reported because it uses hand-crafted feature extraction strategy as proposed by (Roy and Roth 2015), which works only for English dataset.

Table 2: Math problem solving accuracy.

		MAWPS	**Math23K**
Oracle		84.8	87.0
Seq2seq	LSTM	25.6	51.9
	CNN	44.0	42.3
	DNS	59.9	58.1
Deep RL	MathDQN	60.25	-
Classification	Bi-LSTM	62.8	57.9
	Self-Att	60.4	56.8
Our Approach	T-RNN	**66.8**	**66.9**
	- EN	63.9	61.1
	- Bi-LSTM	31.1	34.1
	- Self-Att	66.3	65.1
Retrieval	Jaccard	45.6	47.2
	Cosine	38.8	23.8
Hybrid	DNS+Retrieval	59.9	64.7
	T-RNN+Retrieval	**67.0**	**68.7**

The classification models can achieve higher accuracy in the small dataset, i.e., in MAWPS. However, when the dataset size increases with more templates as in Math23K, they become inferior to the generation model DNS. It implies the classification models are more sensitive to the increase of template space.

We denote our approach by T-RNN and we can see from Table 2 that it performs significantly better than the existing generation and classification models, boosting the accuracy from 62.8 to 66.8 in MAWPS and from 58.1 to 66.9 in Math23K. This is owning to the effectiveness of template prediction, quantity embedding and recursive NN for operator inference. We also examine the effect of equation normalization, Bi-LSTM and self attention in the answer generation module. We can see that EN plays a more significant role in the dataset with more templates. When it is applied, the accuracy in Math23K can improve from 61.1 to 66.9. The Bi-LSTM network is also crucial to obtain effective quantity embedding. If we directly apply the self attention layer on top of the question word embeddings, the accuracy drops sharply. The self attention itself also has minor effect on accuracy enhancement.

The similarity-based retrieval approach was proposed in (Huang et al. 2016). Given a query math problem, its idea is to find the most similar question in the training dataset and use its annotated equation template to solve the problem. We can see that the simple retrieval model works poorly in both datasets. However, it can be combined with generation or classification models to improve the accuracy. In the hybrid model, a similarity threshold is pre-defined. In (Wang, Liu,

and Shi 2017), the retrieval model is used if the similarity between a candidate and the query problem is larger than the threshold; otherwise, DNS is used. We also conduct an experiment to evaluate the performance of our model when combined with retrieval approach. Results show that there is noticeable improvement in the Math23K dataset.

Break-down Analysis

We report the accuracy of template prediction by the seq2seq model in Table 3. If the predicted template exactly matches the annotated template, we consider it a positive hit. It is interesting to observe that the accuracy of template prediction is even lower than the final accuracy of problem solving in MAWPS, i.e., $65.8 < 66.8$. The reason is that a problem can be correctly solved by multiple different templates, but only one of them is annotated. For example, $10 - 5 - 2$ is equal to $10 - (5 + 2)$ but they correspond to different suffix expressions. The finding implies that the seq2seq model is flexible in learning the tree structures even when not all the correct annotations are provided. We also report the percentage of illegal templates in the two datasets. We can see that Math23K has many more training samples and very low percentage of illegal templates are generated in this dataset.

Table 3: Accuracy of template prediction module.

	MAWPS	**Math23K**
Accuracy w/o EN	62.2	59.6
Accuracy w EN	65.8	69.1
Percentage of illegal templates	9.0	0.7

Given the set of problems P_t whose templates are correctly predicted, we derive a subset of P_t, denoted by P_s which contains the problems correctly solved. We use $\frac{|P_t|}{|P_s|}$ to represent the accuracy of the answer generation module. As shown in Table 4, the accuracy in Math23K reaches as high as 94.4. This verifies the effectiveness of Bi-LSTM, self-attention and recursive NN in the answer generation module.

Table 4: Accuracy of the answer generation module.

	MAWPS	**Math23K**
Accuracy of Recursive NN	84.9	94.4

In Table 5, we examine the performance of our MWP solver with the increasing length of templates, which is defined as the number of operators in a template. We investigate the cases where the length increases from 1 to 10 and report the percentage of templates with the particular length in the whole test dataset. We can see that around 90 percent of the problems in Math23K are associated with a template with no greater than 3 operators. Around half the problems require two operators to generate the answer. There is an obvious accuracy descending pattern, from 80.9 downward to 37.9, when the length of templates increases from 1 to 4. When the length further increases from 4 and 6, the accuracy is located in the range $[33, 40]$. When the problems become more complex, with length $\geq$ 7, they cannot be correctly solved by our approach.

Table 5: Accuracy for increasing length of templates.

Math23K		
# Operators	Proportion (%)	Acc (%)
1	19.9	80.9
2	49.6	73.6
3	19.9	53.3
4	5.8	37.9
5	3.3	39.4
6	0.9	33.3
7	0.2	0
8	0	0
9	0.1	0
10	0.2	0

Similar to the work in (Huang et al. 2017), we show the top-10 most frequent templates in Table 6 and report the accuracy of our method when solving each type of template without operator encapsulation. We observe that the template $n_1 1\, n_2 + /$ can be perfectly solved by our model. The "1" in the template means it does not explicitly appear in the problem text. An example question matching the template is like "112 *is greater than a number by* 12%. *What's the number?*". These questions basically follow the similar text pattern and thus are relatively easy to solve. However, our model does not perform well for the template $n_1 n_2 * n_3 *$, with accuracy 58.8. The reason is that the problems related to this template are rather diversified in terms of problem description. Without sufficient amount of training data, the seq2seq model is not able to well capture the relationship between the quantities.

Table 6: Accuracy of per template.

Math23K		
Template	Proportion (%)	Acc (%)
$n_1 n_2 *$	2.4	91.2
$n_1 n_2 /$	2.02	85.4
$n_2 n_1 /$	1.31	74.2
$n_1 n_2 * n_3 *$	0.72	58.8
$n_1 n_2 * n_c /$	0.72	88.2
$n_1 1\, n_2 + /$	0.67	100
$n_1 n_2 + n_3 *$	0.67	81.3
$n_1 1\, n_2 - *$	0.63	80
$n_2 1\, n_1 - *$	0.59	85.7
$n_1 1\, n_2 - /$	0.55	92.3

Error Analysis

Finnaly, we conduct error analysis and explain the bad cases that cannot be well solved by our model. First, the seq2seq model is not good at predicting long templates , mainly due to the lack of training data. As shown in Table 5, there are only a small portion of problems with complex templates in the dataset. This type of error occupies the major proportion ($142/331$ in Math23K, $184/788$ in MAWPS). Second, the problems that require external knowledge have been very

challenging for existing MWP solvers and our model is not exceptional. Third, the semantic understanding of the problem text is not perfect and still has much room for improvement. Hence, the relation between the quantities may not be well captured, leading to wrong template prediction and operator inference.

Conclusion

In this paper, we proposed to use tree structure template represented by suffix expression for math word problem annotation. To reduce the number of templates and improve the accuracy of template prediction, we proposed equation normalization and operator encapsulation. The unknown operators in the template are inferred by a recursive neural network. We conducted extensive experiments on two of the largest datasets and the results showed that our proposed model outperformed the existing deep learning approaches by a wide margin.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China under grants No. 61602087, 61632007, the Fundamental Research Funds for the Central Universities under grants No. ZYGX2016J080, and the 111 Project No. B17008.

References

Amnueypornsakul, B., and Bhat, S. 2014. Machine-guided solution to mathematical word problems. In *PACLIC*, 111–119.

Bobrow, D. 1964. Natural language input for a computer problem solving system. 146–226.

Feigenbaum, E. A., and Feldman, J. 1963. *Computers and Thought*. New York, NY, USA: McGraw-Hill, Inc.

Goller, C., and Kuchler, A. 1996. Learning task-dependent distributed representations by backpropagation through structure. *Neural Networks* 1:347–352.

Hosseini, M. J.; Hajishirzi, H.; Etzioni, O.; and Kushman, N. 2014. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, 523–533.

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

Huang, D.; Shi, S.; Lin, C.; and Yin, J. 2017. Learning fine-grained expressions to solve math word problems. In *EMNLP*, 805–814.

Koncel-Kedziorski, R.; Hajishirzi, H.; Sabharwal, A.; Etzioni, O.; and Ang, S. D. 2015. Parsing algebraic word problems into equations. *TACL* 3:585–597.

Koncel-Kedziorski, R.; Roy, S.; Amini, A.; Kushman, N.; and Hajishirzi, H. 2016. MAWPS: A math word problem repository. In *NAACL*, 1152–1157.

Kushman, N.; Zettlemoyer, L.; Barzilay, R.; and Artzi, Y. 2014. Learning to automatically solve algebra word problems. In *ACL*, 271–281.

Liang, C.; Hsu, K.; Huang, C.; Li, C.; Miao, S.; and Su, K. 2016. A tag-based statistical english math word problem solver with understanding, reasoning and explanation. In *IJCAI*, 4254–4255.

Lin, Z.; Feng, M.; dos Santos, C. N.; Yu, M.; Xiang, B.; Zhou, B.; and Bengio, Y. 2017. A structured self-attentive sentence embedding. *CoRR* abs/1703.03130.

Mitra, A., and Baral, C. 2016. Learning to use formulas to solve simple arithmetic problems. In *ACL*.

Robaidek, B.; Koncel-Kedziorski, R.; and Hajishirzi, H. 2018. Data-driven methods for solving algebra word problems. *CoRR* abs/1804.10718.

Roy, S., and Roth, D. 2015. Solving general arithmetic word problems. In *EMNLP*, 1743–1752.

Roy, S., and Roth, D. 2017. Unit dependency graph and its application to arithmetic word problem solving. In *AAAI*, 3082–3088.

Roy, S., and Roth, D. 2018. Mapping to declarative knowledge for word problem solving. *TACL* 6:159–172.

Shi, S.; Wang, Y.; Lin, C.; Liu, X.; and Rui, Y. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *EMNLP*, 1132–1142.

Socher, R.; Huval, B.; Manning, C. D.; and Ng, A. Y. 2012. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP-CoNLL*, 1201–1211.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 1631–1642.

Socher, R.; Manning, C. D.; and Ng, A. Y. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *NIPS*.

Upadhyay, S.; Chang, M.; Chang, K.; and Yih, W. 2016. Learning from explicit and implicit supervision jointly for algebra word problems. In *EMNLP*, 297–306.

Wang, L.; Wang, Y.; Cai, D.; Zhang, D.; and Liu, X. 2018a. Translating math word problem to expression tree. In *EMNLP*, 1064–1069.

Wang, L.; Zhang, D.; Gao, L.; Song, J.; Guo, L.; and Shen, H. T. 2018b. MathDQN: Solving arithmetic word problems via deep reinforcement learning. In *AAAI*.

Wang, Y.; Liu, X.; and Shi, S. 2017. Deep neural solver for math word problems. In *EMNLP*, 845–854.

Zhang, D.; Wang, L.; Xu, N.; Dai, B. T.; and Shen, H. T. 2018. The gap of semantic parsing: A survey on automatic math word problem solvers. *CoRR* abs/1808.07290.

Zhou, L.; Dai, S.; and Chen, L. 2015. Learn to solve algebra word problems using quadratic programming. In *EMNLP*, 817–822.