HMS: A Hierarchical Solver with Dependency-Enhanced Understanding for Math Word Problem

Xin Lin, 1 Zhenya Huang, 1* Hongke Zhao, 2 Enhong Chen, 1 Qi Liu, 1 Hao Wang, 1 Shijin Wang 3
1 Anhui Province Key Lab. of Big Data Analysis and Application, School of Computer Science and Technology, University of Science and Technology of China
2 College of Management and Economics, Tianjin University
3 iFLYTEK Research & State Key Laboratory of Cognitive Intelligence, iFLYTEK Co., Ltd.
{linx, wanghao3}@mail.ustc.edu.cn, {huangzhy, cheneh, qiliuql}@ustc.edu.cn, hongke@tju.edu.cn, sjwang3@iflytek.com

Abstract

Automatically solving math word problems is a crucial task for exploring the intelligence levels of machines in the general AI domain. It is highly challenging since it requires not only natural language understanding but also mathematical expression inference. Existing solutions usually explore sequence-to-sequence models to generate expressions, where the problems are simply encoded sequentially. However, such models are generally far from enough for understanding problems as similar to humans and lead to incorrect answers. To this end, in this paper, we propose a novel Hierarchical Math Solver (HMS) to make deep understanding and exploitation of problems. In problem understanding, imitating human reading habits, we propose a hierarchical word-clause-problem encoder. Specifically, we first split each problem into several clauses and learn problem semantics from the local clause level to the global problem level. Then, in clause understanding, we propose a dependency-based module to enhance clause semantics with the dependency structure of the problem. Next, in expression inference, we propose a novel tree-based decoder to generate the mathematical expression for the answer. In the decoder, we apply a hierarchical attention mechanism to enhance the problem semantics with context from different levels, and a pointer-generator network to guide the model to copy existing information and infer extra knowledge. Extensive experimental results on two widely used datasets demonstrate that HMS achieves not only better answers but also more reasonable inference.

Introduction

Building machines to automatically solve math word problems (MWP) is a crucial issue in artificial intelligence (AI) and natural language processing (NLP), which has attracted much attention for a long history. It is of great significance to serve as a good testbed to evaluate the intellectual abilities of machines, such as language understanding and logical reasoning. Therefore, successfully solving MWP can be considered as a milestone towards general AI in the broad domains (Zhang et al. 2019). Besides, it can also be used to evaluate problem representations in online education systems, where great progress has been achieved (Huang et al. 2017b; Yin et al. 2018, 2019).

A typical example of math word problems is illustrated in Figure 1. Generally, the problem is described in natural language narratives (A rectangle is ...) and then asks a corresponding question (What is the ...?) for the answer (i.e., 22).

To solve this problem, we should first read and translate it into a mathematical expression composed of numbers (e.g., 4, 3) and operators (e.g., “+” and “×”). During this process, the machine mainly requires two key abilities: (1) language comprehension to understand the meaning of problems describing quantity relations (e.g., length, width and perimeter); (2) mathematical inference to generate the right expression in a fine-grained logic.

There is much effort in solving math word problems from early rule-based approaches (Fletcher 1985), statistical machine learning (Hosseini et al. 2014), semantic parsing (Koncel-Kedziorski et al. 2015) to recent sequence-to-sequence (seq2seq) methods (Wang et al. 2019; Xie and Sun 2019). In the literature, seq2seq methods have shown strong competitiveness in that they do not need any human intervention. Specifically, existing seq2seq methods mainly focus on the inference ability to generate expressions, where the problem is often simply encoded as a sequence of words by one single vector. In such a way, models are far from enough to understand problems as similar to humans because they only notice the straightforward effect of the word order (e.g., “longer” being after “cm” and before “than”), while ignoring the complicated structure among words in the

<table>
<thead>
<tr>
<th>Problem</th>
<th>A rectangle is 4 cm wide, and its length is 3 cm longer than its width. What is the perimeter of the rectangle?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>22</td>
</tr>
<tr>
<td>Expression</td>
<td>2× (4+3+4)</td>
</tr>
</tbody>
</table>

Figure 1: An example of MWP.
problem (e.g., relations among “3”, “longer” and “width”). Therefore, they generally lack the ability to understand problems, which may confuse important quantity relations, and result in incorrect answers. To this end, we hope to improve the MWP solver by imitating human reading habits.

However, it is non-trivial along this line due to the following challenges. First, instead of directly putting all words together, humans usually read problems part by part (e.g., sentence) to understand the semantic meanings. In this way, they can pay much attention to local details (e.g., numbers in Figure 1) and easily combine them together for the global goal (i.e., “What is ...?”). How to mimic such a procedure in MWP is challenging. Second, for understanding the local meaning of each part (sentence), words are often dependent on other headwords and provide supplementary details. As shown in Figure 1, number “3” describes the numeric difference (i.e., “longer”) between the “length” and “width”, which is highly necessary for solving the problem. Although humans can capture such semantic dependency without much effort, machines may struggle with that. Third, in the inference, the ability to translate logic in local semantics into an expression segment does matter, e.g., “3 cm longer” should be projected to “+3” in expression generation. Last, the solver often requires human knowledge in the math domain, which is not given in the problem. In Figure 1, the inference requires rectangle perimeter formula “2 × (NUM + NUM)” in addition to learning “length” and “width” from the problem, where the number “2” is inferred from human knowledge. How to exploit such knowledge remains much open.

To address the challenges above, in this paper, we propose a novel Hierarchical Math Solver (HMS) to make deep understanding and exploitation for solving MWP. In problem understanding, motivated by human reading habits, we propose a novel hierarchical word-clause-problem encoder. Specifically, we split the problem into several clauses so that our encoder can learn the whole problem semantics from local clauses to global problem considering inter-clause relations (e.g., descriptions of the same entity) on the basis of word semantics intuitively. Then in local clause understanding, we develop a dependency-based module to enhance the semantics with intra-clause relations. Here, we construct a dependency tree for each clause to preserve semantic structures (e.g., dependency) in the problem. Next, in expression inference, we propose a novel tree-based decoder to generate the mathematical expression for the answer in a recursive way. The decoder first extracts related context information and then predicts each symbol to generate the expression. In context extraction, We apply a hierarchical attention mechanism to explicitly enhance the problem semantics with context from different levels. Then in symbol prediction, a pointer-generator network is incorporated to guide the model to copy existing information and infer extra knowledge for the expression. Finally, we conduct extensive experiments on the two largest datasets in the domain. The experimental results demonstrate that HMS can achieve better solutions to answers as well as more reasonable inference.

**Related Work**

In this section, we review studies on automatic MWP solving and make a brief introduction to dependency and pointer-generator network in NLP tasks.

**Math Word Problem.** Studies on MWP solver have a long history dating back to the 1960s (Feigenbaum, Feldman et al. 1963; Bobrow 1964). In the literature, MWP solvers can be roughly divided into four categories: rule-based methods, statistical machine learning methods, semantic parsing methods and deep learning methods. Specifically, rule-based methods relied on manually crafted rules to match predefined problem templates, such as (Huang 1985) and (Yuhui et al. 2010). Machine learning methods achieved great success in many fields (Liu et al. 2011; Zhao et al. 2017) and were also widely used in MWP solvers. Statistical machine learning methods selected and completed predefined expression templates with traditional machine learning methods (e.g., SVM), such as (Kushman et al. 2014) and (Roy and Roth 2017). Semantic parsing methods mapped problem text to structured logic forms and inferred the answer with logic rules, such as (Shi et al. 2015) and (Huang et al. 2017a). These methods required tremendous human effort on feature engineering and annotation and lacked generality, which led to a restricted application. With the advantages of automatic feature extraction and strong generality, deep learning methods especially seq2seq models have been widely used in recent studies (Wang, Liu, and Shi 2017; Huang et al. 2018; Wang et al. 2019; Xie and Sun 2019; Wang et al. 2018c; Zhang et al. 2020). For example, Wang et al. (2019) applied a recursive neural network to predict missing operators on templates predicted by seq2seq model, and Xie and Sun (2019) proposed a goal-driven tree-structured decoder to exploit the tree structure of mathematical expressions. To model quantity relations in problems, Zhang et al. (2020) constructed graphs for quantity-related features to enhance problem understanding. In recent years, more difficult mathematical problems were also tackled (Huang et al. 2020) in addition to traditional MWP.

Although great success has been achieved, complex language structures of the problem were not considered in most of these seq2seq models, where the problem was simply encoded as a sequence of words by a RNN model such LSTM and GRU, which was far from enough for accurate and efficient problem modeling. Better results can be achieved with better modeling and representations of complex problem structures (Wang et al. 2018a).

**Dependency Parsing in NLP.** Dependency parsing, proposed in (Tesnière 1959), is a basic task in NLP, which aims to extract and explain relations between units in a sentence such as words and phrases. The result of dependency parsing can be demonstrated with a dependency tree, where each node represents a word in the sentence. In the dependency tree, the child is dependent on the parent and can be viewed as details of the parent, such as the modifier, qualifier of a noun and the subject, object of a verb. The dependency tree is highly corresponding to the semantics of the sentence, thus dependency parsing is helpful in sentence understanding and is widely used in NLP tasks such as machine translation (Bastings et al. 2017; Wu et al. 2018; Yang et al. 2020).
**Model Overview**

Motivated by human MWP solving, we propose a novel Hierarchical Math Solver (HMS) composed of a hierarchical encoder and a tree-based decoder following seq2seq architecture, which is shown in Figure 2. Specifically, given a MWP $P$, in problem understanding, the hierarchical word-clause-problem encoder (three levels in the encoder) splits the problem into several clauses (red boxes) and generate problem representations from the local clause level (green boxes) to the global problem level (orange boxes) on the basis of word-level semantics (blue boxes). In expression inference, the tree-based decoder generates the expression $E_P$ according to problem representation with a hierarchical attention mechanism (red arrows in decode module) and a pointer-generator network (blue arrows).

**Word-Clause-Problem Hierarchical Encoder**

In problem understanding, imitating human problem reading from local to global, we split the problem into clauses, and learn problem semantics from clauses to the whole problem.

Specifically, given a problem $P$, we split the problem into $m$ clauses $C_P = \{C_1, C_2, \ldots, C_m\}$ (red boxes in encoder in Figure 2) as units for local semantics, where $C_i =$ \{p_{i1}, p_{i2}, $\ldots$, $p_{il}$\} is a subsequence of words from $P$ (e.g., “A rectangle ... wide”, “and its ... width”). The reason is that semantics in a clause is relatively complete (i.e., contains almost all necessary semantic elements) and simple (i.e., describes only one relation). Many complex methods can be applied for an accurate split of clauses, but here we split by commas and periods for simplification.

Given the clause partition of the problem, we propose a novel word-clause-problem hierarchical encoder to generate semantic representations for words, clauses and the whole problem respectively from the local to the global in a bottom-up manner.

**Context-Enriched Word Embedding.** Given a problem $P = \{p_1, p_2, \ldots, p_n\}$, in word level the encoder aims to learn meaning representation $h_w$ for each word $p_i$ enriched with contextual information of sequential relations.

Specifically, the decoder first maps each word token $p_i$ to an embedding vector $x_i$, pre-trained with word2vec (Mikolov et al. 2013). Then the decoder learns the contextual representation $h_w$ for word $p_i$ (blue boxes) with a bidirectional GRU (blue arrows in encoder). We adopt bidirectional GRU for its ability to exploit contextual information from both directions. The generation of $h_w$ can be formulated as follows:

$$h_w^f = \text{GRU}_f(h_{w-1}^f, x_s), \quad h_w^b = \text{GRU}_b(h_{w+1}^b, x_s). \quad (1)$$

$$h_w = h_w^f + h_w^b. \quad (2)$$

The representation contains semantics of the word itself and information of the context, which serves as the basis of problem understanding in clause and problem levels. More advanced techniques can be applied for better word-level representations, and we adopt the simple one for simplification.

**Dependency-Enhanced Clause Module.** After obtaining the representation $h_w$ for each word $p_i$ in word level and corresponding partition $C_k$ for each clause, in clause level the encoder aims to model local clause semantics $c_k^e$ enhanced with intra-clause semantic relations, which are often represented as semantic dependency.

To capture the dependency in the clause, we construct a dependency tree $T_k = \{t_{k1}, t_{k2}, \ldots, t_{ki}\}$ for the clause $C_k$ with stanford corenlp toolkit (Manning et al. 2014), which is shown in Figure 3. In dependency tree $T_k$, each node $t_{ki}$ represents a word from $C_k$, and each word except the root is dependent on its parent and provides a supplementary detail to the parent. For example, “rectangle” is dependent on its parent “perimeter” and describes the subject of “perimeter”. Thus, to get the dependency-enhanced clause semantics, we can exploit the structure of the dependency tree.

To this end, we propose a dependency-based module to learn enhanced clause semantics by enriching words with local details from the children following the dependency tree.

**Hierarchical Math Solver**

In this section, we first formally introduce the MWP task and then describe the details of our proposed HMS framework.

**Problem Definition**

Generally, a math word problem $P$ is defined as a sequence of $n$ word tokens and numeric values: $P = \{p_1, p_2, \ldots, p_n\}$, where $p_i$ is either a word token (e.g., “length” and “width” in Figure 1) or a numeric value (e.g., “3” and “4”). We denote the numeric values set for problem $P$ as $N_P$ (e.g., \{3, 4\}). These numeric tokens are mapped to a special token “NUM” in problem solving for we do not care the value of these tokens.

The answer $s_P$ to problem $P$ is the numeric value (e.g., “22”) of the required unknown variable derived from $P$ by a mathematical expression $E_P$. Expression $E_P$ is defined as a sequence of $m$ operators, numeric constants and numeric variables from problem $P$: $E_P = \{y_1, y_2, \ldots, y_m\}$, where $y_i$ comes from a decoding target vocabulary $V_P$ for problem $P$. The decoding target vocabulary is composed of the operators set $V_O$ (e.g., \{+, -, \times, \div\}), numeric constants set $V_C$ (e.g., \{1, 2, \pi\}) and $N_P$, which is denoted as $V_P = V_O \cup V_C \cup N_P$ (e.g., \{+, -, \times, \div, 1, 2, \pi, 3, 4\}). Note that $V_P$ differs in different $P$ due to varied $N_P$.

Given the input sequence of problem $P$, the goal of solving math word problem is to learn a model that reads tokens from $P$ and generates the output sequence of the expression $E_P = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m\}$ towards the answer $s_P$ for $P$.

**Pointer-Generator Network**

Pointer network was proposed to address problems where the number of target classes of output depends on the length of the input (Vinyals, Fortunato, and Jaitly 2015). It has the ability to copy words directly from the input as output. Pointer-generator network combines the pointer network and traditional seq2seq model, thus is able to copy words from source text as well as generate new words from the vocabulary. It is widely used in NLP tasks such as text summarization (Gu et al. 2016; See, Liu, and Manning 2017; Sun et al. 2018). In MWP solvers, the pointer-generator network has been used to copy numbers from problems to avoid generating invalid numbers from other problems (Huang et al. 2018).
Figure 2: Overview of proposed HMS framework. Decoding process goes from $y_0$ to $y_6$ in a recursive manner.

Figure 3: Dependency-based module for clause level. Encoding process goes on dependency tree in a bottom-up manner from leaves to root.

which is shown in Figure 3. Specifically, given dependency tree $T_k$ (on the left) for the $k$th clause $C_k$ (at the bottom), for leaf nodes (e.g., “rectangle”) with no child to provide further details, corresponding word-level representations are enough to model the semantics. Formally, enriched representation (green boxes) $h^d_l = h_l$ for leaf node $t_l$ (dark cyan arrows), where $h_l$ denotes word-level representation for $t_l$ (blue boxes). Then for inner nodes (e.g., “perimeter”), the encoder applies an attention mechanism to select related and meaningful detailed semantics from children to merge into the word representations. Formally, for each child node $t_c$ of inner node $t_p$, the semantic representation $h_p$ for $t_p$ is initialized with its word-level representation (gray arrows) and enriched representation $h^d_c$ for $t_c$ (green boxes for the children) has merged semantics from its children if any. Then the enriched representation $h^d_p$ for $t_p$ is generated by attention mechanism (orange arrows) as follows:

$$S_{ac}(h_p, h^d_c) = w^T_{cs} \text{ReLU}(W_{ca}[h_p, h^d_c] + b_{ca}),$$

$$w_c = \frac{\exp(S_{ac}(h_p, h^d_c))}{\sum_i \exp(S_{ac}(h_p, h^d_i))},$$

$$h^d_p = \sum_c w_c h^d_c,$$

The enriched representation of each node contains the semantics of local details and the structure of the dependency tree, which provides a more accurate understanding of the corresponding word.

The generation of enriched semantic representations is performed following the dependency tree in a bottom-up manner from leaves to the root. Thus, the enriched representation of root node contains local semantics for all words in the clause enhanced with information of the structure of the whole dependency tree, so we take the representation of the root as the local semantic representation for the clause $c^d_k = h^d_{root}$.

Inter-Clause Relation-Enhanced Problem Module. After modeling the local semantics for clause $c^d_k$, in problem level the encoder aims to generate the global semantic representation for the whole problem from the local clause semantic representations. Note that the clause-level representations rely absolutely on intra-clause information without consideration of relations between clauses, which contain cross-clause quantity relations for problem understanding such as descriptions of the same entity in different views (e.g., the value of the width and the relation between the length and the width in Figure 1). Thus, in problem level the encoder is required to enhance clause representations with inter-clause relations and generate the representation of the global semantics for the whole problem.

To achieve the goal, given clause-level representation $c^d_k$ (green boxes in Figure 2) for the $k$th clause $C_k$, the encoder
first applies positional encoding with a learnable embedding PE to get position-enriched representation \( e_k^p \) with sequential relations between clauses as follows:

\[
e_k^p = c_k^d + PE(k).
\] (7)

Then the encoder applies a self-attention mechanism to model relevance between clauses and enhance clause representations with semantic information from related clauses. For clause \( C_i \) with representation \( e_i^p \) and each clause \( e_k^p \), the enhanced representation \( e_k^e \) (orange boxes) is generated as follows:

\[
S_{as}(e_k^p, e_i^p) = W_{sas}^T \text{ReLU}(W_{sas}[e_k^p, e_i^p] + b_{sa}),
\] (8)

\[
w_s = \frac{\exp(S_{as}(e_k^p, e_i^p))}{\sum_i \exp(S_{as}(e_k^p, e_i^p))},
\] (9)

\[
e_k^e = \sum_s w_s e_s^p,
\] (10)

\[
e_k^e = \text{ReLU}(W_{sa}[e_k^p, e_i^p] + b_{sa}).
\] (11)

The enhanced representation for each clause contains both local semantics within the clause and global semantics from other clauses in the problem. Therefore, we take the representation for the question clause, which serves as the central part with the goal of the problem and is usually the last one, as the global representation for the whole problem \( h^e = e_m \) (last orange box with red label).

**Tree-Based Expression Decoder**

After understanding the local details and the global goal of problem \( P \), in expression inference, to exploit tree structure of the expression, we propose a novel tree-based decoder to generate prefix expression \( E_P \) following the Goal-driven Tree-Structured (GTS) framework (Xie and Sun 2019). The expression generation is converted to the expression tree construction, the symbol is predicted according to the goal vector \( q \) given to each node, and the structure is predicted by goal decomposition in a top-down recursive manner.

Specifically, the goal vector (purple boxes in Figure 2) of the root node \( q_0 \) is initialized with the global representation of the problem \( h^e \) (black arrow between the encoder and decoder). The goal vectors for other nodes are generated by goal decomposition in a top-down recursive manner (purple arrows). If the predicted symbol (dark cyan boxes) for node \( e_l q \) is an operator, the decoder adds two children to the node. The decoder first generates a new goal vector from that of \( e_l q \) and makes prediction on the left child. After the whole left subtree is predicted, the decoder generates the goal vector and makes prediction on the right child with left subtree embedding (orange arrows in the decoder). Goal decomposition for one branch ends when the predicted symbol is not an operator (i.e., variables or constants). More details about goal decomposition are available in (Xie and Sun 2019).

Given goal vector \( q \) for node \( e_l \), the decoder first generates context vector \( c \) according to the goal vector with the hierarchical attention mechanism for local semantics exploitation (red arrows in decode module), and then predicts symbol \( y \) according to goal and context vectors with the pointer-generator network for knowledge inference (blue arrows).

**Hierarchical Context Generation.** Given goal vector \( q \), in context generation, the decoder aims to extract context representation related to the goal from semantic representations of the problem to assist symbol prediction. To enable the translation of logic in local semantics in each clause, it is required to enhance global problem semantics of goal with explicit local details from words and clauses.

To achieve the goal, we propose a hierarchical attention mechanism to explicitly exploit problem semantics from the word and clause levels. Specifically, the decoder first models relevance between the goal vector \( q \) and each clause-level representation \( e_k^c \) as follows:

\[
S_c(q, e_k^c) = W_{ac}^T \text{ReLU}(W_{ac}[q, e_k^c] + b_{ac}).
\] (12)

Then the decoder models relevance between the goal vector \( q \) and each word \( h_{k,t} \) in the \( k \)th clause as follows:

\[
S_w(q, h_{k,t}) = W_{sw}^T \text{ReLU}(W_{sw}[q, h_{k,t}] + b_{sw}).
\] (13)

The context vector \( c \) is finally generated as follows:

\[
c = \sum_k w_k^c (e_k^c + \sum_t w_{k,t}^w h_{k,t}),
\] (14)

where

\[
w_k^c = \frac{\exp(S_c(q, e_k^c))}{\sum_i \exp(S_c(q, e_i^c))},
\] (15)

\[
w_{k,t}^w = \frac{\exp(S_w(q, h_{k,t}))}{\sum_i \exp(S_w(q, h_{k,i}))}.
\] (16)

The context vector is explicitly incorporated with the word and clause level semantics, which helps symbol prediction by providing related local details to the global target semantics of the goal. The hierarchical manner helps query relevant words by reducing the influence of irrelevant words through searching unrelated clauses with smaller weights, which are less likely to contain relevant words.

**Symbol Prediction with Knowledge inference.** Given goal vector \( q \) and context vector \( c \) for the tree node, in symbol prediction, the decoder aims to generate symbol \( y \) from decoding target vocabulary. Expression generation requires the decoder to be able to copy existing variables from the problem (e.g., “3” and “4” in Figure 1), and infer extra knowledge (e.g., “2” and “x”), which is reflected as constants and operators from the external vocabulary.

To this end, we apply a pointer-generator network following (Huang et al. 2018) in symbol prediction. Specifically, the decoder first predicts the probability of symbol being extra knowledge from the external vocabulary as follows:

\[
P_{gen} = \sigma(w_{py}[q, c] + b_{py}).
\] (17)

Then the probability distribution for quantities \( Y_p \) copied from the problem is predicted as follows:

\[
P_p(Y_p) = \text{Softmax}(w_{ps}^T \text{ReLU}(W_{ps}[q, c, h_{Y_p}] + b_{ps})),
\] (18)

where \( h_{Y_p} \) denotes word-level representations for quantities \( Y_p \). The probability distribution for external symbols \( Y_g \) inferred from knowledge is predicted as follows:

\[
P_g(Y_g) = \text{Softmax}(w_{gs}^T \text{ReLU}(W_{gs}[q, c, e_{Y_g}] + b_{gs})).
\] (19)
where $e_Y$ denotes learnable embeddings of external symbols $Y_g$. Then the combined probability distribution for all symbols $Y$ is calculated as follows:

$$P_c(Y = y) = \begin{cases} (1 - P_{gen}) \cdot P_p(y) & \text{if } y \in Y_p \\ P_{gen} \cdot P_g(y) & \text{if } y \in Y_g \end{cases} \quad (20)$$

The predicted symbol $\hat{y}$ is generated according to probability distribution $P_c$. The prediction splits the search space into two parts and performs a weighted search on them with explicitly predicted probabilities (usually close to 0 or 1 if well-trained), which helps infer knowledge and copy information by reducing the confusion on the type of the symbol to predict and noise from another part of the search space.

The generated expression is composed of symbols predicted following the construction of the expression tree in a top-down manner, which can be viewed as a pre-order traverse on the whole predicted expression tree.

**Model Training**

For each problem $P$ in the training dataset and corresponding target expression $E_P = \{y_1, y_2, \cdots, y_m\}$, the loss $L$ can be defined as minimizing the sum of negative log-likelihood of probabilities of $m$ target symbols:

$$L = \sum_{t=1}^{m} -\log P_c(y_t | y_1, y_2, \cdots, y_{t-1}, P). \quad (21)$$

**Experimental Study**

In this section, we conduct experiments on two widely used datasets for the MWP task and compare our proposed HMS framework with baseline models for MWP.

**Datasets.** We use two widely used datasets for the MWP task in our experiments: Math23K and MAWPS.

- **Math23K** (Wang, Liu, and Shi 2017) is a dataset containing 23,162 Chinese math word problems for elementary school students with only one unknown variable.

- **MAWPS** (Roy and Roth 2017) combines several published math word problem datasets using MAWPS system (Koncel-Kedziorski et al. 2016). The dataset contains problems with one or more unknown variables. We select 2,373 problems with only one unknown variable and a target expression.

The statistics of the two datasets are shown in Table 1. The average clause number and expression length of Math23K are much longer than MAWPS, which shows that the Math23K dataset is more difficult than MAWPS. Math23K are much longer than MAWPS, which shows that the Math23K dataset is more difficult than MAWPS.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Math23K</th>
<th>MAWPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. Problems</td>
<td>23,162</td>
<td>2,373</td>
</tr>
<tr>
<td>Avg. problem length</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>Avg. Num. clauses</td>
<td>3.75</td>
<td>3.27</td>
</tr>
<tr>
<td>Avg. expr. length</td>
<td>5.55</td>
<td>3.87</td>
</tr>
</tbody>
</table>

Table 1: Statistics of datasets.

We use accuracy as the evaluation metric. The prediction is considered correct if the calculated value of the predicted expression equals the answer. Table 2 reports the accuracy of our proposed model and baseline models on the two datasets. There are several observations. First, our model outperforms all the baseline models, which demonstrates that HMS can effectively understand the problem by exploiting dependency and modeling semantics in different levels hierarchically, and thus enhance expression inference. Next, HMS w/o hierarchy outperforms all the baseline models, which demonstrates that HMS can effectively understand the problem by exploiting dependency and modeling semantics in different levels hierarchically, and thus enhance expression inference. Next, our model outperforms all the baseline models, which demonstrates that HMS can effectively understand the problem by exploiting dependency and modeling semantics in different levels hierarchically, and thus enhance expression inference.

**Overall Result**

We use accuracy as the evaluation metric. The prediction is considered correct if the calculated value of the predicted expression equals the answer. Table 2 reports the accuracy of our proposed model and baseline models on the two datasets. There are several observations. First, our model outperforms all the baseline models, which demonstrates that HMS can effectively understand the problem by exploiting dependency and modeling semantics in different levels hierarchically, and thus enhance expression inference. Next, we apply the 5-fold cross-validation.

- **T-RNN** (Wang et al. 2019) applies recursive neural network to predict the missing operators of the predicted expression template.

- **GROUP-ATT** (Li et al. 2019) uses several attention mechanisms to extract different features in the problem.

- **GTS** (Xie and Sun 2019) proposes a goal-driven tree-structured neural network to generate the expression.

*Our code is available at https://github.com/bigdata-ustc/hms.*
Ablation Study. To investigate the effect of each component in our model: hierarchical encoder, dependency-based module and pointer-generator network, we conduct ablation studies. The results are shown in Table 2. Specifically, “HMS w/o hierarchy” directly applies word-level encoding without the hierarchical encoder. “HMS w/o dependency” implements clause-level encoding with self-attention on word-level representations without dependency-based module. We observe that the accuracy of our model degrades when any component is missing, which means that all components are helpful and complementary in the problem understanding and expression generation.

Performance over Expression and Problem Length. To investigate the performance of our model with the increasing length of expressions and problems, we report the number of test instances over different expression and problem lengths in Table 3 and 4, and the accuracies of corresponding test instances compared with the GTS model in Figure 4 and 5 respectively. Since similar experiment results are observed on the two datasets, we only report results on Math23K. There are several observations. First, the performance of each model degrades with the increasing clause number due to increasing difficulty. Second, in group “6” and “7” where the performance drops sharply due to complex problems, our model outperforms GTS obviously, which proves that the hierarchical encoder is able to capture more semantics from complex problems. Last, for test instances with only 1 clause where the main difference between the two models is the dependency-based module, our model outperforms GTS and proves the effectiveness of the module.

Case Study. Further, we conduct case studies on expressions generated by our model and GTS and provide three cases in Table 6. For convenience, we convert the prefix expression generated by our model and GTS and provide three cases in Table 6. For convenience, we convert the prefix expression generated by our model and GTS to infix form. We can get the following observations from the cases: (1) In case 1, GTS confuses “n1” and “n2” with similar context but from different clauses, and our model avoids this problem by incorporating clause details “afternoon” and “morning” with each variable through the hierarchical attention mechanism; (2) In case 2, GTS confuses the relations between “sunny days”, “rainy days” and “cloudy days”, and our model addresses this problem and figures out the correct relations by modeling dependency between them; (3) In case 3, GTS confuses the type of the required symbol and mistakenly predicts problem variable “n1” in a node requiring an external symbol, and our

![Figure 4: Accuracy over expression length.](image1)

![Figure 5: Accuracy over problem length.](image2)

<table>
<thead>
<tr>
<th>Expr. length</th>
<th># Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-5 7 9 11 13+</td>
<td>174 522 191 66 34 13</td>
</tr>
</tbody>
</table>

Table 3: Number of test instances over expression length.

<table>
<thead>
<tr>
<th>Problem length</th>
<th># Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-20-29 30-39 40-49 50+</td>
<td>160 354 289 135 62</td>
</tr>
</tbody>
</table>

Table 4: Number of test instances over problem length.
model solves this problem by explicitly predicting that an external symbol is required and thus inferring “×” from extra knowledge with less noise from problem variables.

## Conclusion

In this paper, we proposed a novel Hierarchical Math Solver (HMS) to deeply exploit problem semantics for solving math word problems. We proposed a hierarchical word-clause-problem encoder and a dependency-based module to understand problem semantics, and a tree-based recursive decoder with hierarchical attention mechanism and pointer-generator network to generate expressions. Experimental results on the datasets and further discussions showed the effectiveness of our model and its components, especially our proposed hierarchical encoder and dependency-based module, which were able to model complex semantics in long problems and help expression generation. For further study, there exist many more relations in problems to investigate that may be helpful in solving MWP. Meanwhile, the exploitation of human knowledge is still rough and simple, where exists great room for improvement in future study.

## Acknowledgments

This research was partially supported by grants from the National Key Research and Development Program of China (Grant No. 2018YFC0832101), the National Natural Science Foundation of China (No.s U20A20229 and 61922073), the Foundation of State Key Laboratory of Cognitive Intelligence (No. iED2020-M004) and the Iflytek joint research program.

## References


ACM SIGIR Conference on Research and Development in Information Retrieval, 1729–1732.


