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 *H*ierarchical *M*ath *S*olver (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).

Problem	A rectangle is 4 cm wide, and its length is $\boxed{3}$ cm $\boxed{\text{longer}}$ than its $\boxed{\text{width}}$. What is the perimeter of the rectangle?
Answer	22
Expression	$2\times(4+3+4)$

Figure 1: An example of MWP.

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

*Corresponding Author.

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 “$2 \times (\mathrm{NUM} + \mathrm{NUM})$” 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 *H*ierarchical *M*ath *S*olver (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 (Fletcher 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).

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).

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: $P = \{p_1, p_2, \cdots, p_n\}$, 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: $E_P = \{y_1, y_2, \cdots, y_m\}$, 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 $\hat{E}_P = \{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_m\}$ towards the answer s_P for P.

Model Overview

Motivated by human MWP solving, we propose a novel *H*ierarchical *M*ath *S*olver (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 $\hat{E}_P$ 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 $\mathcal{C}_P = \{C_1, C_2, \cdots, C_m\}$ (red boxes in encoder in Figure 2) as units for local semantics, where $C_i = \{p_{i1}, p_{i2}, \cdots, p_{il}\}$ 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 $P = \{p_1, p_2, \cdots, p_n\}$, in word level the encoder aims to learn meaning representation $\boldsymbol{h}_s$ for each word p_s enriched with contextual information of sequential relations.

Specifically, the decoder first maps each word token p_s to an embedding vector $\boldsymbol{x}_s$ pre-trained with *word2vec* (Mikolov et al. 2013). Then the decoder learns the contextual representation $\boldsymbol{h}_s$ for word p_s (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 $\boldsymbol{h_s}$ can be formulated as follows:

$$\boldsymbol{h}_s^f = \mathrm{GRU}_f(\boldsymbol{h}_{s-1}^f, \boldsymbol{x}_s), \quad \boldsymbol{h}_s^b = \mathrm{GRU}_b(\boldsymbol{h}_{s+1}^b, \boldsymbol{x}_s), \tag{1}$$

$$\boldsymbol{h}_s = \boldsymbol{h}_s^f + \boldsymbol{h}_s^b. \tag{2}$$

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 $\boldsymbol{h}_s$ for each word p_s in word level and corresponding partition C_k for each clause, in clause level the encoder aims to model local clause semantics $\boldsymbol{c}_k^d$ 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 $T_k = \{t_{k1}, t_{k2}, \cdots, t_{kl}\}$ 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,

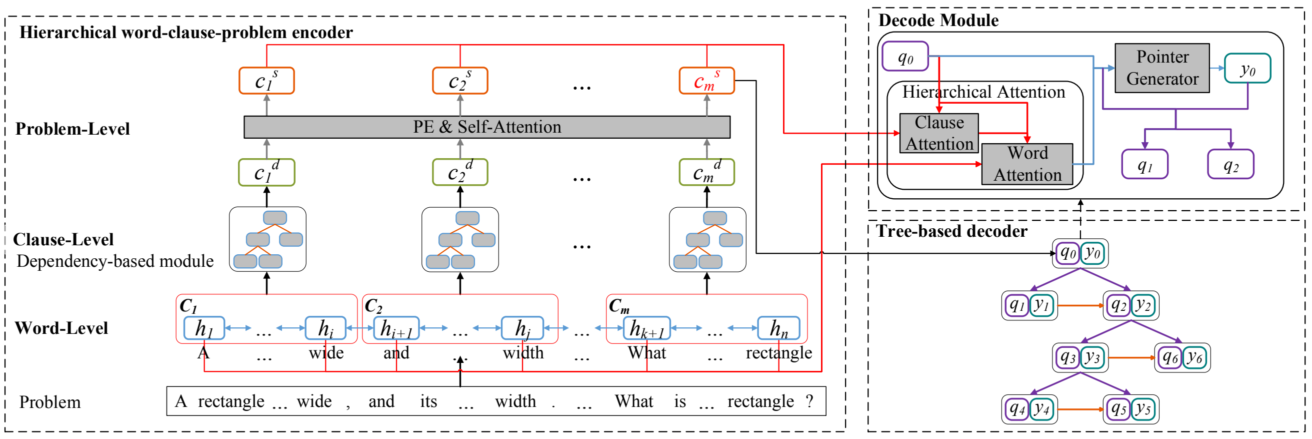

Figure 2: Overview of proposed HMS framework. Decoding process goes from y_0 to y_6 in a recursive manner.

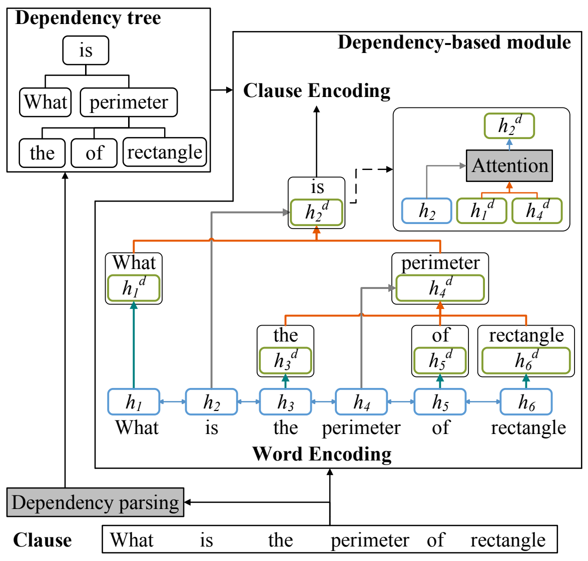

Figure 3: Dependency-based module for clause level. Encoding process goes on dependency tree in a bottom-up manner from leaves to root.

which is shown in Figure 3. Specifically, given dependency tree T_k (on the left) for the kth 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) $\boldsymbol{h}_l^d = \boldsymbol{h}_l$ for leaf node t_l (dark cyan arrows), where $\boldsymbol{h}_l$ 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 $\boldsymbol{h}_p$ for t_p is initialized with its word-level representation (gray arrows) and enriched representation $\boldsymbol{h}_c^d$ for t_c (green boxes for the children) has merged semantics from its children if any. Then the enriched representation $\boldsymbol{h}_p^d$ for t_p is generated by attention mechanism (orange arrows) as follows:

$$\mathrm{S}_{ac}(\boldsymbol{h}_p, \boldsymbol{h}_c^d) = \mathbf{w}_{cs}^{\mathrm{T}} \mathrm{ReLU}(\mathbf{W}_{ca}[\boldsymbol{h}_p, \boldsymbol{h}_c^d] + \boldsymbol{b}_{ca}), \quad (3)$$

$$w_c = \frac{\exp(\mathrm{S}_{ac}(\boldsymbol{h}_p, \boldsymbol{h}_c^d))}{\sum_i \exp(\mathrm{S}_{ac}(\boldsymbol{h}_p, \boldsymbol{h}_i^d))}, \quad (4)$$

$$\boldsymbol{h}_p^r = \sum_c w_c \boldsymbol{h}_c^d, \quad (5)$$

$$\boldsymbol{h}_p^d = \mathrm{ReLU}(\mathbf{W}_{co}[\boldsymbol{h}_p, \boldsymbol{h}_p^r] + \boldsymbol{b}_{co}). \quad (6)$$

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 $\boldsymbol{c}_k^d = \boldsymbol{h}_{root}^d$.

Inter-Clause Relation-Enhanced Problem Module. After modeling the local semantics for clause $\boldsymbol{c}_k^d$, 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 $\boldsymbol{c}_k^d$ (green boxes in Figure 2) for the kth clause C_k, the encoder first applies positional encoding with a learnable embedding PE to get position-enriched representation $\boldsymbol{c}_k^p$ with sequential relations between clauses as follows:

$$\boldsymbol{c}_k^p = \boldsymbol{c}_k^d + \mathrm{PE}(k). \tag{7}$$

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_k with representation $\boldsymbol{c}_k^p$ and each clause $\boldsymbol{c}_s^p$, the enhanced representation $\boldsymbol{c}_k^s$ (orange boxes) is generated as follows:

$$\mathrm{S}_{as}(\boldsymbol{c}_k^p, \boldsymbol{c}_s^p) = \mathbf{w}_{ss}^{\mathrm{T}}\mathrm{ReLU}(\mathbf{W}_{sa}[\boldsymbol{c}_k^p, \boldsymbol{c}_s^p] + \boldsymbol{b}_{sa}), \tag{8}$$

$$w_s = \frac{\exp(\mathrm{S}_{as}(\boldsymbol{c}_k^p, \boldsymbol{c}_s^p))}{\sum_i \exp(\mathrm{S}_{as}(\boldsymbol{c}_k^p, \boldsymbol{c}_i^p))}. \tag{9}$$

$$\boldsymbol{c}_k^r = \sum_s w_s \boldsymbol{c}_s^p, \tag{10}$$

$$\boldsymbol{c}_k^s = \mathrm{ReLU}(\mathbf{W}_{so}[\boldsymbol{c}_k^p, \boldsymbol{c}_k^r] + \boldsymbol{b}_{so}). \tag{11}$$

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 $\boldsymbol{h}^s = \boldsymbol{c}_m^s$ (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 $\hat{E}_P$ 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 $\boldsymbol{q}$ 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 $\boldsymbol{q}_0$ is initialized with the global representation of the problem $\boldsymbol{h}^s$ (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 $\hat{et}_p$ is an operator, the decoder adds two children to the node. The decoder first generates a new goal vector from that of $\hat{et}_p$ 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 $\boldsymbol{q}$ for node $\hat{et}$, the decoder first generates context vector $\boldsymbol{c}$ according to the goal vector with the hierarchical attention mechanism for local semantics exploitation (red arrows in decode module), and then predicts symbol $\hat{y}$ according to goal and context vectors with the pointer-generator network for knowledge inference (blue arrows).

Hierarchical Context Generation. Given goal vector $\boldsymbol{q}$, 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 $\boldsymbol{q}$ and each clause-level representation $\boldsymbol{c}_k^s$ as follows:

$$\mathrm{S}_c(\boldsymbol{q}, \boldsymbol{c}_k^s) = \mathbf{w}_{sc}^{\mathrm{T}}\mathrm{ReLU}(\mathbf{W}_{ac}[\boldsymbol{q}, \boldsymbol{c}_k^s] + \boldsymbol{b}_{ac}). \tag{12}$$

Then the decoder models relevance between the goal vector $\boldsymbol{q}$ and each word $\boldsymbol{h}_{k,t}$ in the *k*th clause as follows:

$$\mathrm{S}_w(\boldsymbol{q}, \boldsymbol{h}_{k,t}) = \mathbf{w}_{sw}^{\mathrm{T}}\mathrm{ReLU}(\mathbf{W}_{aw}[\boldsymbol{q}, \boldsymbol{h}_{k,t}] + \boldsymbol{b}_{aw}). \tag{13}$$

The context vector $\boldsymbol{c}$ is finally generated as follows:

$$\boldsymbol{c} = \sum_k w_k^c(\boldsymbol{c}_k^s + \sum_t w_{k,t}^w \cdot \boldsymbol{h}_{k,t}), \tag{14}$$

where

$$w_k^c = \frac{\exp(\mathrm{S}_c(\boldsymbol{q}, \boldsymbol{c}_k^s))}{\sum_i \exp(\mathrm{S}_c(\boldsymbol{q}, \boldsymbol{c}_i^s))}, \tag{15}$$

$$w_{k,t}^w = \frac{\exp(\mathrm{S}_w(\boldsymbol{q}, \boldsymbol{h}_{k,t}))}{\sum_i \exp(\mathrm{S}_w(\boldsymbol{q}, \boldsymbol{h}_{k,i}))}. \tag{16}$$

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 $\boldsymbol{q}$ and context vector $\boldsymbol{c}$ for the tree node, in symbol prediction, the decoder aims to generate symbol $\hat{y}$ 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 "$\times$"), 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:

$$P_{gen} = \sigma(\mathbf{w}_{pg}[\boldsymbol{q}, \boldsymbol{c}] + b_{pg}). \tag{17}$$

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

$$\boldsymbol{P}_p(Y_p) = \mathrm{Softmax}(\mathbf{w}_{ps}^{\mathrm{T}}\mathrm{ReLU}(\mathbf{W}_{pa}[\boldsymbol{q}, \boldsymbol{c}, \boldsymbol{h}_{Y_p}] + b_{pa})), \tag{18}$$

where $\boldsymbol{h}_{Y_p}$ denotes word-level representations for quantities Y_p. The probability distribution for external symbols Y_g inferred from knowledge is predicted as follows:

$$\boldsymbol{P}_g(Y_g) = \mathrm{Softmax}(\mathbf{w}_{gs}^{\mathrm{T}}\mathrm{ReLU}(\mathbf{W}_{ga}[\boldsymbol{q}, \boldsymbol{c}, \boldsymbol{e}_{Y_g}] + b_{ga})), \tag{19}$$

where $\boldsymbol{e}_{Y_g}$ denotes learnable embeddings of external symbols Y_g. Then the combined probability distribution for all symbols Y is calculated as follows:

$$\boldsymbol{P}_c(Y=y) = \begin{cases} (1-P_{gen}) * \boldsymbol{P}_p(y) & y \in Y_p \\ P_{gen} * \boldsymbol{P}_g(y) & y \in Y_g \end{cases} \tag{20}$$

The predicted symbol $\hat{y}$ is generated according to probability distribution $\boldsymbol{P}_c$. 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:

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

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.

Experimental Setup

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.

Baselines. We compare our proposed model with the following baseline models:

- **DNS** (Wang, Liu, and Shi 2017) applies seq2seq model to directly map input problems to output expressions.
- **Math-EN** (Wang et al. 2018b) addresses the problem of duplicated equations by equation normalization.

Dataset	Math23K	MAWPS
Num. Problems	23,162	2,373
Avg. problem length	29	30
Avg. Num. clauses	3.75	3.27
Avg. expr. length	5.55	3.87

Table 1: Statistics of datasets.

	Math23K	MAWPS
DNS	0.581	0.595
Math-EN	0.667	0.692
T-RNN	0.669	0.668
GROUP-ATT	0.695	0.761
GTS	0.743	0.786
HMS	**0.761**	**0.803**
HMS w/o hierarchy	0.750	0.788
HMS w/o dependency	0.755	0.791
HMS w/o pointer-generator	0.756	0.789

Table 2: Accuracy of HMS and baseline models.

- **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.

Implementation Details.[1] Our model is implemented using PyTorch. The dimension of word embedding vectors is 128 and the dimensions of all other hidden vectors are 512. The word embeddings are initialized with pre-trained word2vec vectors learned from the training dataset. Words with less than 5 occurrences are converted to a special token "UNK". Other parameters are initialized by Kaiming initialization (He et al. 2015). We apply dropout in our model with a probability of 0.5. We use Adam optimizer with an initial learning rate of 0.001 to optimize the loss function (Eq. 21). The learning rate is halved every 20 epochs. The mini-batch size is set to 64, and our model is trained for 80 epochs.

Since the Math23K dataset has provided a partition when published, we simply follow its original setup. For MAWPS, we apply the 5-fold cross-validation.

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,

[1]Our code is available at https://github.com/bigdata-ustc/hms.

Expr. length	3-	5	7	9	11	13+
# Instances	174	522	191	66	34	13

Table 3: Number of test instances over expression length.

Problem. length	19-	20-29	30-39	40-49	50+
# Instances	160	354	289	135	62

Table 4: Number of test instances over problem length.

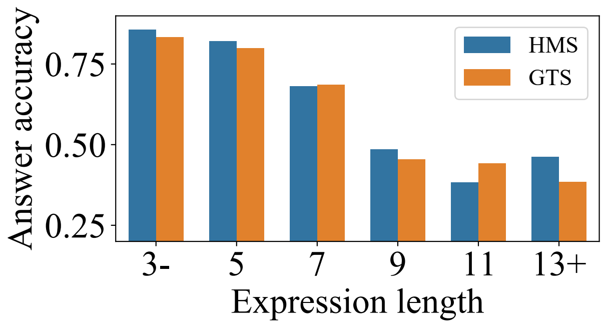

Figure 4: Accuracy over expression length.

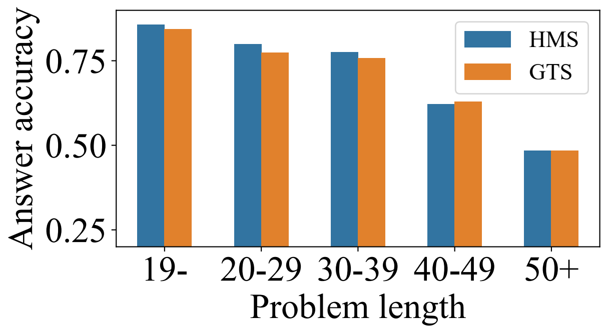

Figure 5: Accuracy over problem length.

DNS has the worst performance, which shows that the exploitation of related features in MWP (e.g., the tree structure of the expression and semantic relations in the problem) is of great importance in the task. Last, the accuracies in MAWPS are higher than those in Math23K, especially for models with higher performance. This is due to the higher difficulty of Math23K from longer expressions and problems and fewer instances for each expression template.

Model Discussion

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 in the following experiments for simplification. There are several observations. First, generally, the performance of each model degrades with the increasing length of expressions, which is reasonable due to increasing difficulty. Second, the difference between the performance of the two models also decreases with the increasing length of expressions. For high difficulty, both models do not work well, thus the difference is not obvious, which is reasonable. But exceptions are found for problems with very long expressions. The reason may be that the number of test instances of these groups is not enough, and the occasional factors have a great impact on the results, which may be not representative. Similar trends are observed in performance over different problem lengths that the accuracy and difference between the two models decrease with the increasing problem length and corresponding increasing difficulty.

Performance over Clause Number. To investigate the performance of models with the increasing number of clauses in the problem, similarly, we report the number of test instances over different clause number in Table 5 and corresponding accuracies compared with the GTS model in Figure 6. 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 models 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

# Clause	1	2	3	4	5	6	7	8+
# Instances	38	80	297	326	175	52	17	15

Table 5: Number of test instances over clause number.

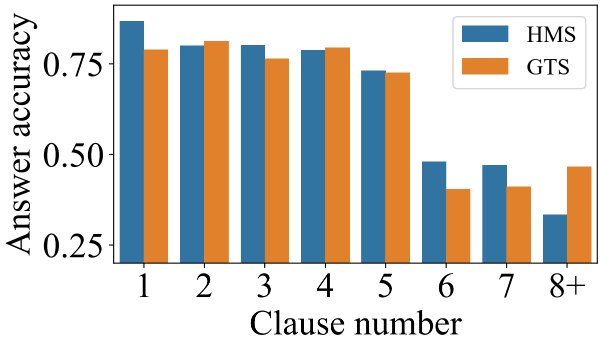

Figure 6: Accuracy over clause number.

Case 1: A car drives for $n1$ hours in the morning, and $n2$ hours at the same speed in the afternoon. If the car drives $n3$ km in the morning, how many km does it drive in the afternoon? **Our model:** $n3 \div n1 \times n2$ (**correct**) **GTS:** $n3 \div n2 \times n1$ (**wrong**)
Case 2: In January there are $n1$ sunny days, the number of rainy days is $n2$ times less than sunny days, and the number of cloudy days is $n3$ times more than rainy days. How many cloudy days are there in January? **Our model:** $n1 \times (1 - n2) \times (1 + n3)$ (**correct**) **GTS:** $n1 \times (1 - n2) + n1 \times n3$ (**wrong**)
Case 3: If we use squared tiles with a side length of $n1$ m to pave the ground with an area of $n2$ m^2, how many tiles are required? **Our model:** $n2 \div (n1 \times n1)$ (**correct**) **GTS:** $n2 \div n1$ (**wrong**)

Table 6: Typical cases generated by our model and GTS.

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 *H*ierarchical *M*ath *S*olver (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

Bastings, J.; Titov, I.; Aziz, W.; Marcheggiani, D.; and Sima'an, K. 2017. Graph Convolutional Encoders for Syntax-aware Neural Machine Translation. In *Proceedings of the EMNLP 2017*, 1957–1967.

Bobrow, D. G. 1964. Natural language input for a computer problem-solving system. *Semantic Information Processing* 146–226.

Feigenbaum, E. A.; Feldman, J.; et al. 1963. *Computers and thought*. New York McGraw-Hill.

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

Gu, J.; Lu, Z.; Li, H.; and Li, V. O. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *Proceedings of the ACL 2016*, 1631–1640.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034.

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

Huang, D.; Liu, J.; Lin, C.-Y.; and Yin, J. 2018. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, 213–223.

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

Huang, Z.; Liu, Q.; Chen, E.; Zhao, H.; Gao, M.; Wei, S.; Su, Y.; and Hu, G. 2017b. Question difficulty prediction for READING problems in standard tests. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 1352–1359.

Huang, Z.; Liu, Q.; Gao, W.; Wu, J.; Yin, Y.; Wang, H.; and Chen, E. 2020. Neural mathematical solver with enhanced formula structure. In *Proceedings of the 43rd International*

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

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

Koncel-Kedziorski, R.; Roy, S.; Amini, A.; Kushman, N.; and Hajishirzi, H. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1152–1157.

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

Li, J.; Wang, L.; Zhang, J.; Wang, Y.; Dai, B. T.; and Zhang, D. 2019. Modeling intra-relation in math word problems with different functional multi-head attentions. In *Proceedings of the ACL 2019*, 6162–6167.

Liu, Q.; Ge, Y.; Li, Z.; Chen, E.; and Xiong, H. 2011. Personalized travel package recommendation. In *2011 IEEE 11th International Conference on Data Mining*, 407–416. IEEE.

Manning, C. D.; Surdeanu, M.; Bauer, J.; Finkel, J. R.; Bethard, S.; and McClosky, D. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 55–60.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

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

See, A.; Liu, P. J.; and Manning, C. D. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the ACL 2017*, 1073–1083.

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

Sun, F.; Jiang, P.; Sun, H.; Pei, C.; Ou, W.; and Wang, X. 2018. Multi-source pointer network for product title summarization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 7–16.

Tesnière, L. 1959. *Éléments de syntaxe structurale*. Paris: Klincksieck.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. In *Advances in neural information processing systems*, 2692–2700.

Wang, H.; Chen, E.; Liu, Q.; Xu, T.; Du, D.; Su, W.; and Zhang, X. 2018a. A United Approach to Learning Sparse Attributed Network Embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*, 557–566. IEEE.

Wang, L.; Wang, Y.; Cai, D.; Zhang, D.; and Liu, X. 2018b. Translating a Math Word Problem to a Expression Tree. In *Proceedings of the EMNLP 2018*, 1064–1069.

Wang, L.; Zhang, D.; Gao, L.; Song, J.; Guo, L.; and Shen, H. T. 2018c. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

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 *Proceedings of the AAAI 2019*, volume 33, 7144–7151.

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

Wu, S.; Zhang, D.; Zhang, Z.; Yang, N.; Li, M.; and Zhou, M. 2018. Dependency-to-dependency neural machine translation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26(11): 2132–2141.

Xie, Z.; and Sun, S. 2019. A Goal-Driven Tree-Structured Neural Model for Math Word Problems. In *IJCAI*, 5299–5305.

Yang, B.; Wong, D. F.; Chao, L. S.; and Zhang, M. 2020. Improving tree-based neural machine translation with dynamic lexicalized dependency encoding. *Knowledge-Based Systems* 188: 105042.

Yin, Y.; Huang, Z.; Chen, E.; Liu, Q.; Zhang, F.; Xie, X.; and Hu, G. 2018. Transcribing content from structural images with spotlight mechanism. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2643–2652.

Yin, Y.; Liu, Q.; Huang, Z.; Chen, E.; Tong, W.; Wang, S.; and Su, Y. 2019. Quesnet: A unified representation for heterogeneous test questions. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1328–1336.

Yuhui, M.; Ying, Z.; Guangzuo, C.; Yun, R.; and Ronghuai, H. 2010. Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems. In *2010 Second International Workshop on Education Technology and Computer Science*, volume 2, 476–479. IEEE.

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. *IEEE transactions on pattern analysis and machine intelligence* .

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 *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 3928–3937.

Zhao, H.; Zhang, H.; Ge, Y.; Liu, Q.; Chen, E.; Li, H.; and Wu, L. 2017. Tracking the dynamics in crowdfunding. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 625–634.