A Neural Semantic Parser for Math Problems Incorporating Multi-Sentence Information

RUIYONG SUN, Fudan University, China
YIJIA ZHAO, Fudan University, China
QI ZHANG, Fudan University, China
KEYU DING*, iFLYTEK CO.,LTD, China
SHIJIN WANG, iFLYTEK CO.,LTD, China
CUI WEI, Information Center of Ministry of Science and Technology, China

In this paper, we studied the problem of parsing a math problem into logical forms. It is an essential pre-processing step for automatically solving math problem. Most of the existing studies on semantic parsing mainly focused on the single sentence level. However, for parsing math problem, we need to incorporate information from multiple sentences into consideration. To achieve the task, we formulated the task as a machine translation problem and extended the sequence to sequence model with a novel two-encoder architecture and a word level selective mechanism. For training and evaluating the proposed method, we constructed a large-scale dataset. Experimental results showed that the proposed two-encoder architecture and word level selective mechanism could bring significant improvement. The proposed method can achieve better performance than the state-of-the-art methods.

CCS Concepts: • **Computing methodologies → Artificial intelligence**; **Natural language processing**; Language resources;

Additional Key Words and Phrases: Semantic parsing, math problem solving, multi-sentence, selective mechanism

ACM Reference Format:
Ruiyong Sun, Yijia Zhao, Qi Zhang, Keyu Ding, Shijin Wang, and Cui Wei. 2019. A Neural Semantic Parser for Math Problems Incorporating Multi-Sentence Information. *ACM Trans. Web* 9, 4, Article 15 (January 2019), 16 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

We studied the problem of parsing a math problem into logical forms. It is an essential pre-processing step for automatically solving math problems [17]. Most of the recent studies on semantic parsing can be roughly divided into three categories: grammatical-rules models [17, 35], syntactic-parsing models [8], and sequence-to-sequence models [2]. Grammatical-rules models and syntactic-parsing models relied on manually designed features or rules which could render models representation-specific or domain-specific. In contrast to the above two categories of models, sequence-to-sequence

*This is the corresponding author

Authors' addresses: Ruiyong Sun, Fudan University, China, rysun16@fudan.edu.cn; Yijia Zhao, Fudan University, China, zhaoyj16@fudan.edu.cn; Qi Zhang, Fudan University, China, qz@fudan.edu.cn; Keyu Ding, iFLYTEK CO.,LTD, 666 Wangjiang West Road, Hefei, 230088, China, kyding@iflytek.com; Shijin Wang, iFLYTEK CO.,LTD, China, sjwang3@iflytek.com; Cui Wei, Information Center of Ministry of Science and Technology, China, cuiw@most.cn.

1559-1131/2019/1-ART15 $15.00
https://doi.org/0000001.0000001

Problems:

Define a function $f(x) = \lg\left(x + \frac{a}{x} - 2\right)$ where a is a constant greater than 0. Find the domain D of the function; When $a \epsilon (1,4)$,Find the minimum value of the function at $[2, +\infty)$. For all $x \in [2, +\infty), f(x) > 0$ always holds true, try to determine the range of value a .

Logical forms :

Function $(f(x), lg(x + \frac{a}{x} - 2))$; GreaterThan$(a, 0)$; Constant(a);

$\exists\ what, f(x): Equals(what, DomainOfFunction(D, f(x)))$;

$\exists\ what, a, f(x): BelongTo(a, (1,4)) \wedge Equals\left(what, MinimalValueOfFunction(f(x), [2, +\infty))\right)$;

$\forall$ x $\exists\ f(x), a: BelongTo(x, [2, +\infty)) \wedge GreaterThan(f(x), 0)$; $\exists\ what, a: Equals(what, ValueRangeOfConstant(a))$;

Fig. 1. An example of math problem and its logical forms. The upper part is a math problem with four sentences, and the lower part is the logical forms of the four sentences.

models directly predict the logical forms of the natural language input by used an encoder-decoder architecture which do not contain any manually designed features.

However, sequence-to-sequence models have only performed evaluations at the single sentence level datasets, while math problems usually have more than one sentence and we need to incorporate information from multiple sentences into consideration. Our experiments on math problems showed that neural sequence-to-sequence models usually had bad performances if an entire math problem was treated as a long sentence. However, splitting a math problem into multiple single sentences, and treating this task at the single sentence level can cause the loss of some important information in the context sentences. As shown in Figure 1, the word "function" in the second sentence represents "$f(x)$" in the first sentence.

In this paper, we introduce a novel sequence-to-sequence model with two encoders that parses the math problems at the single sentence level, but intelligently introduces the important information from the entire math problem. First, input math problem x is split into multiple single sentences $s_1, s_2, ..., s_M$. The sentences in the math question are forward related, and $s_1, s_2, ..., s_{i-1}$ are defined as *context sentences* of the i-th sentence s_i. Then, for each single sentence, one encoder RNN maps its words into hidden state vectors, and another encoder does the same thing to its context sentences to obtain the semantically related words from these sentences. A decoder based on a pointer-generator network [20] takes the state vectors of these two encoders as inputs and generates the logical forms of the input sentence. To reduce the interference from unrelated words in the context sentences, we introduce a word-level selective mechanism in the second encoder to help our model determine the importance of each source-side word. Our word-level selective mechanism constructs a second-level representation of each source-side word by using a selective gate network [37], which enhances the model by extracting semantic information from the source sentence.

To evaluate our model, we constructed a labeled dataset based on the Chinese National College Entrance Examination (like the SAT in the US), which contains over 15,000 math problems with complex logical structures. Experimental results on our dataset showed that our model improved the sentence-level accuracy of the best baseline mode from 63.76% to 69.83%, and the logical form-level accuracy from 65.09% to 70.70%.

The main contributions of this paper can be summarized as follows.

- We investigated multi-sentence-level neural semantic parsing, which has been relatively unexplored in past research and is more challenging than the single-sentence level.
- We present novel extensions to the neural encoder-decoder architecture and propose a word-level selective mechanism to achieve this task.

- Extensive experiments on our large-scale dataset demonstrated that our model based on a two encoder architecture outperformed several well-known statistical methods and neural models that performed well on the task of semantic parsing.

2 RELATED WORK

The Semantic parsing of natural language has been an long AI challenge [31] and has received significant attention. Most of the early studies [3, 5, 11, 19, 26, 32, 33] relied on manually designed features and predefined rules, which made their models task-specific or representation-specific.

Recent work has shown that a neural attention model can be used for semantic parsing with very good effects [7, 10, 12, 14, 23, 24]. Most of these studies used an encoder RNN to map the input words of the input sequence into hidden state vectors, and then used a decoder RNN with these state vectors as inputs and generated the logical forms of the input word by word. However, these models were trained on a single-sentence corpus, and compared with the math problems in our dataset, these sentences are very short, which makes it easier to align semantically related words using the attention mechanism. Our goal was to propose a novel neural attention model that can perform well at multi-sentence-level semantic parsing tasks.

The automation of math problem solving has also been studied for a long time and dates back to Feigenbaum et al. [4]. Many previous studies on math problem solving tended to directly map the math problem text to equations or templates by using statistical learning approaches or a neural sequense to sequence model and reported promising results [9, 13, 16, 18, 28, 29, 36]. Other works focused on symbolic approaches [15, 22], where the math problem text was first transformed into specific structures by some means. Then, equations or templates were derived from these structures.

Recently, a few studies [8, 17, 21] have given attention to the semantic information in math problems, which proved to be important for math problem solving [17, 21]. However, the dataset used to evaluate their model was quite small. The conclusions obtained from these datasets may not be representative. Comparatively, our work made great efforts to focus on the semantic information, and our model was evaluated on a much larger dataset.

3 PROBLEM STATEMENT

The semantic parsing of a math problem can be treated as a sequence-to-sequence task. Given an input sequence of words of a math problem $x_1, x_2, ..., x_N \epsilon\ v^{(in)}$, where $v^{(in)}$ is the input vocabulary, the goal is to train a model that can learn its logical forms $y_1, y_2, ..., y_J \epsilon\ v^{(out)}$, where $v^{(out)}$ is the the output vocabulary.

3.1 Neural Sequence to Sequence Modeling

A neural sequence to sequence model relies on an encoder-decoder framework, and both the encoder and decoder are implemented with Recurrent Neural Networks (RNNs). In such a framework, the encoder RNN maps the input words of a math problem $x = (x_1, x_2, ..., x_N)$ into hidden state vectors $h_1, h_2, ..., h_N$ and the decoder RNN takes these state vectors as inputs and generates the logical forms of the input math problem $y = (y_1, y_2, ..., y_J)$ word by word. The target token y_j in output sequence y is predicted as follows:

$$P(y|x) = \prod_{j=1}^{J} P(y_j | y_{<j}, x) \tag{1}$$

where J is the length of the output sequence, and $y_{<j}$ is the previous output.

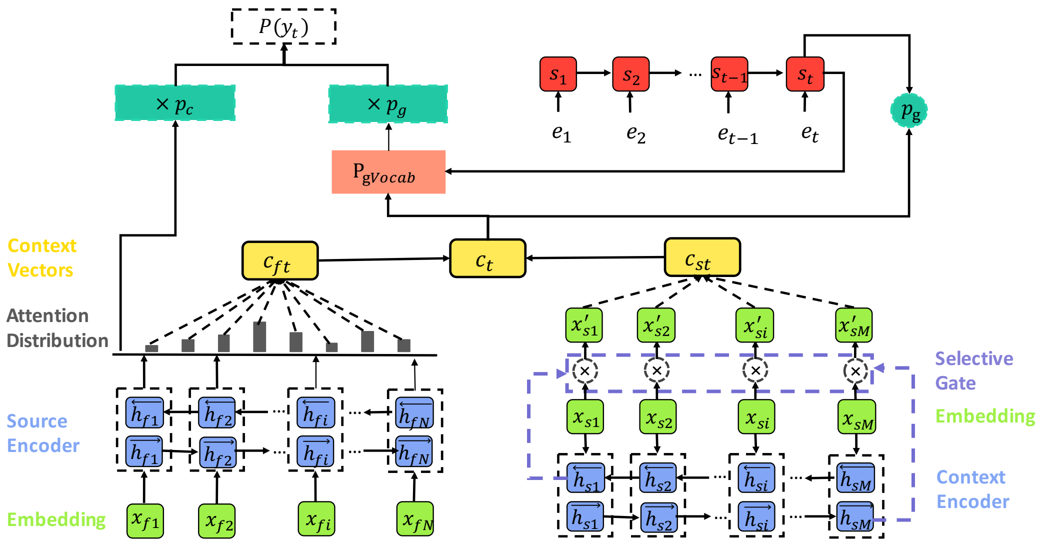

Fig. 2. Overview of our semantic parsing model. Our model contains two encoder. The source encoder and the context encoder based on a selective gate maps the source sentence and its context sentences to hidden state vectors, respectively.The decoder takes these state vectors as inputs and generates the logical forms of the source sentence.For simplicity, we omit some links for computing the attention score (see section ***Approach*** for more details).

This task can be treated at the single-sentence level by taking each sentence of a math problem as an input and generating its logical forms. However, this will cause the lose of some important information in the context sentences.

3.2 Two-Encoder Architecture and word-level selective mechanism

Compared to the encoder-decoder framework, our two-encoder architecture treats this task at the single sentence level, but provides an additional encoder to introduce the important information of the entire math problem. We also introduce a word-level selective mechanism in this encoder to help it determine the importance of each source-side word in the context sentences, which was shown to be effective in reducing the interference of unrelated words. Our model is described in detail in next section.

4 APPROACH

In this section, we describe the two-encoder architecture, word-level selective mechanism, joint attention mechanism, and pointer generator network-based decoder, which are shown in Figure 2.

4.1 Word Representation

We get our word embedding by using an embedding matrix $E \in R^{d \times V}$ where d is the embedding dimension, and V is the words vocabulary size. A word embedding vector x_i is represented by $x_i = Ew_i$ where w_i is the i-th word in a sentence.

4.2 Two-Encoder Architecture

Our model parses the math problems at the single sentence level, but uses an additional encoder to introduce the important information of the entire math problem. In detail, one encoder maps a source sentence to hidden state vectors and the other encoder maps its context sentences to hidden state vectors. Both of our two encoders are implemented using a single-layer bidirectional LSTM network. The sequence of words of a math problem is fed into the LSTM network word by word at each time step. Let h_t, c_t define the hidden state vectors at a time step. They are computed in the LSTM as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1}) \tag{2}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1}) \tag{3}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1}) \tag{4}$$

$$\widetilde{c}_t = tanh(W_c x_t + U_c h_{t-1}) \tag{5}$$

$$c_t = f_t \odot \widetilde{c}_{t-1} + i_t \odot \widetilde{c}_t \tag{6}$$

$$h_t = o_t \odot tanh(c_t) \tag{7}$$

where x_t is an embedding vector of the input word, the variables U and W are weight matrices, σ is a sigmoid activation function, and $\odot$ represents element-wise multiplication.

There is a forward LSTM and a backward LSTM in the bidirectional LSTM, which are respectively used to obtain the forward hidden state $(\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_N})$ and backward hidden state $(\overleftarrow{h_1}, \overleftarrow{h_2}, \ldots, \overleftarrow{h_N})$. Then, the forward and backward hidden states are concatenated to obtain the final hidden states, i.e., $h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}]$. At this point, we obtain the hidden state vectors $(h_{f1}, h_{f2}, ..., h_{fN})$ of the source sentence and the hidden state vectors $(h_{s1}, h_{s2}, ..., h_{sM})$ of its context sentences.

4.3 Word Level Selective Mechanism

To obtain the semantically related words from the context sentences, the second encoder of our model maps the context sentences of the source sentence into hidden state vectors $(h_{s1}, h_{s2}, ..., h_{sM})$. But for the source sentence, only a few words in its context statement are relevant. Thus, we need to keep the highlights and remove the unrelated information. Zhou et al. [37] proposed a selective mechanism that uses a gate network to map the hidden state vectors $(h_{s1}, h_{s2}, ..., h_{sM})$ into a second-level representation $(h'_{s1}, h'_{s2}, ..., h'_{sM})$ that can determine the importance of each word in a context sentence before the decoder. In this section, we propose a word-level selective mechanism to do the same thing at the word embedding layer and obtain better performance.

In detail, our word-level selective mechanism maps the embeddings $(x_{s1}, x_{s2}, ..., x_{sM})$ of each word in a context sentence into a second-level representation $(x'_{s1}, x'_{s2}, ..., x'_{sM})$. First, we concatenate the last forward hidden states $\overrightarrow{h_{sM}}$ and backward hidden states $\overleftarrow{h_{s1}}$ of the second encoder as the context sentence representation S:

$$S = [\overleftarrow{h_{s1}}; \overrightarrow{h_{sM}}] \tag{8}$$

Then, the embedding selective gate vector $esGate_{si}$ and second-level embedding representation x'_{si} of the inputs in each time step i are computed as follows:

$$esGate_{si} = \sigma(W_s x_{si} + U_s S + b) \tag{9}$$

$$x'_{si} = x_{si} \odot esGate_{si} \tag{10}$$

where W_s and U_s are the weight matrices, and b is the bias vector.

4.4 Joint Attention Mechanism

We extend the attention mechanism [1] in our model to jointly learn to align in two encoders. For the first encoder, the context vectors c_{ft} at each time step t are calculated as follows:

$$c_{ft} = \sum_{i=1}^{I} \alpha_{ti}^{f} h_{fi} \tag{11}$$

Here, α_{ti}^{f} denotes the attention distribution and is calculated as follows:

$$e_{ti}^{f} = v \cdot tanh(W_e h_{fi} + U_e s_t + b_e) \tag{12}$$

$$\alpha_{ti}^{f} = \frac{exp(e_{ti}^{f})}{\sum_{i'=1}^{I} exp(e_{ti'}^{f})} \tag{13}$$

where J is the length of the target sequence s_t is the decoder hidden state at time step t and W, U, and v are the weight matrices. We use similar equations to obtain the context vectors c_{st} for the second encoder at each time steps t, except that we use the second-level embedding representation $x_{si}^{'}$ instead of the hidden states h_{si}.

Then, we concatenate the context vectors (c_{ft}, c_{st}) of the two encoders as the final context vector c_t:

$$c_t = [c_{ft}; c_{st}] \tag{14}$$

4.5 Semantic Decoder

Because math problem sentences and their logical form representations share many of the same subsequences, our decoder uses a pointer-generator network [20] to jointly copy and generate the logical form tokens. Briefly, our decoder obtains the target tokens by calculating four important elements: the generation probability $p_g \in [0, 1]$, copy probability $p_c \in [0, 1]$, vocabulary generation distribution $P_g(y_t)$, and vocabulary copy distribution $P_c(y_t)$. The final vocabulary distribution $P(y_t|y_{<t}, x)$ at time step t is calculated as follows:

$$P(y_t|y_{<t}, x) = p_g P_g(y_t) + p_c P_c(y_t) \tag{15}$$

In detail, $P_g(y_t)$ is the generation probability distribution of all the words in the vocabulary and is calculated as follows:

$$P_g(y_t) = softmax(U(W[s_t, c_t] + b_w) + b_u) \tag{16}$$

where U, W, and b are learnable parameters; s_t is the decoder hidden state at time step t; and c_t is the context vector of the problem obtained using equation (14). The generation probability p_g for each time step is calculated as follows:

$$p_g = \sigma(w_{fc}^{T} c_{ft} + w_{sc}^{T} c_{st} + w_{e}^{T} e_t + w_{s}^{T} s_t + b) \tag{17}$$

where w_* are vectors, b is a scalar, e_t is the decoder input at time step t, and c_{ft} and c_{st} are the context vectors of the two encoders (as seen in equation (14)). The copy probability p_c for each time step is calculated as follows:

$$p_c = 1 - p_g \tag{18}$$

$P_c(y_t)$ is the copy probability distribution of the words that appear in the sentence that fed to the first encoder. In other words, when obtaining the logical form tokens of a sentence, we tend to make the decoder only copy words from this sentence and not all of the words in the entire math problem. In this case, $P_c(y_t)$ is obtained by sampling from the attention distribution of the first encoder α^f_{ti}:

$$P_c(y_t) = \sum_{i:w_i=w} \alpha^f_{ti} \tag{19}$$

Math problems contain many equations, if the equations are generated word by word, some incorrect equations may be generated, such as: "$a + +b$". How to ensure that the output equations are mathematically correct is a key point. In this task, the equations in the math problems needs to be copied into their logical forms without any change, such as the equation $lg(x + \frac{a}{x} - 2)$ in the Figure 1. So we treat each equation as a single word. In that way, the decoder will generate a equation at one time step.

In addition, for each input sequence, we add all the words in this sequence to the vocabulary to produce out-of-vocabulary (OOV) words.

4.6 Objective Function

During training, our goal is maximize the likelihood of the generated logical forms given the input sequence. Therefore, we chose the negative log-likelihood loss function:

$$loss = -\frac{1}{D} logP(y|x) \tag{20}$$

where D denotes the set of all math problem-logical form training pairs, and $P(y|x)$ is computed as shown in Equation 1.

5 EXPERIMENTS

We compared our model with several popular semantic parsing models using our dataset. The experimental results show that both the two-encoder architecture and word-level selective mechanism can bring significant improvement.

5.1 Dataset

Motivation Many previous studies on semantic parsing have focused on question-answer pairs, because there are several popular public datasets. However, each question in those datasets contains only one context-independent sentence, which is not in line with the characteristics of natural language. A few recent works [17, 21] began to pay attention to math problems, and performed semantic parsing on the multi-sentence level. However, the datasets they used have scale limitations and are not sufficient to train the models based on neural networks. Moreover, the conclusions based on those datasets may not be representative. Because of the limitations of the previous datasets, we constructed a large multi-sentence-level dataset that contains math problems collected from Chinese National College Entrance Examination (as the SAT in the US) papers.

Data Description and Division Following Seo et al. [21], all 15,546 of the math problems in our dataset were labeled using the basic constructions of first-order logic. Because their dataset contained only geometry problems, we made some extensions to cover all the required *predicates* (such as "GreaterThan") and *entities* (such as "vectors") that appeared in our dataset. For all experiments, we randomly split the dataset into training set(3/5), development set(1/5) and test set(1/5).

Data Collection We aim to construct a large and diverse math problem collection that can fully verify our model's ability to understand math problem texts. To achieve this goal, we choose to collect math problem from Chinese National College Entrance Examination papers. Because problems in these papers covers multiple entrance exam math knowledge points includes functions,

Dataset	# Problems	Mathematical symbol			Problems Types
		propositional	predicate	others	
Train	9369	$\wedge\ \vee$ $\neg$ $\rightarrow$ $\Leftrightarrow$	$\forall$ $\exists$ $\exists!$	$= \neq \approx$ $\cup \cap$ $\perp \parallel \simeq$ $< > \leqslant \geqslant$	functions (power,exponential, logarithmic,trigonometric); inequalities; complex number; aggregate; sequence; vector; geometry (solid, plane); coordinate system
Test	3078				
Dev	3099				

Table 1. Dataset statistics ('propositional' for the propositional logic symbols; 'predicate' for the predicate logic symbols;)

Datasets	# problems	avg.sents	avg.words	uniq.words	atoms
JOBs [34]	640	1.00	9.38	391	4.63
GEOQUERY[34]	880	1.00	8.56	284	4.25
GEOMETRY [21]	119	1.74	23.64	202	11.00
UNIV [17]	394	3.74	70.59	365	10.34
Ours	15546	5.22	25.06	23233	9.78

Table 2. Statistics of our dataset and other semantic parsing data

sequences, inequalities, complex number, geometry and other key points. Besides, our dataset contains almost all of the mathematical logics($\forall$,$\exists$,$\nexists$ etc.).

We collected over 2,000 sets of math examination paper for the last ten years through crowd-sourcing. In detail, the people we hire is given WORD format papers. They are required to convert the papers into XML formats according to the gold standard we set. For example, mathematical symbols and formulas in math problems need to be marked in latex format. In addition, another person's confirmation is necessary to ensure high-quality dataset. In this way, we collect over 15k math problems.

Table 1 shows the statistics of the datasets. The first column lists the number of problems. The remaining columns in Table 1 reveal the logical complexities of the problems. 'propositional' list some of the propositional logic symbols in the problems. The columns for the 'predicate' and 'others' list some predicate logic symbols and other important symbols in the math problems. In conclusion, our dataset covers a wider variety of mathematical symbols.

We compare our dataset to several benchmark datasets listed by Matsuzaki et al. (2017). The statistics of these dataset are listed in Table 2. The first column lists the number of problems. The next two provide the average number of sentences and words in a problem. The fourth column list the number of unique words in the whole datasets. The column for 'atoms' shows that the average number of predicates in each problem. We can see our dataset is much larger than other benchmark dataset.

Labeling with Semantic Language To represent the problem clearly, commonly and simply, we develop a semantic language following the basic constructions of first-order logic, but extending it by covering all the required predicates and objects in our dataset. We had two experienced people to confirm the quality of the data, and the simple kappa coefficient of their assessment results had to be greater than 0.9. According to the statistics, our semantic language has more than 500 different objects (e.g. circle, rectangle) and 2600 different predicate functions (e.g. GreaterThan).

5.2 Baseline

We compared our model with several well-known statistical methods and neural models that performed well on the task of semantic parsing.

WASP Wong and Mooney [30] proposed a novel statistical model that maps natural language sentences to their formal language representations by using statistical machine translation techniques. There are two core parts in their model: a word alignment model and a parsing model. The word alignment model is used for lexical acquisition, and the parsing mode is a syntax-based statistical machine translation model.

Seq2Tree Dong and Lapata [2] proposed a sequence to tree model based on recurrent neural networks, which has been proved to be effective on the task of semantic parsing.

Seq2Seq Sutskever et al. [25] proposed a novel Seq2Seq model based on an encoder-decoder framework, which has been attracting increasing attention. We implemented it with a bidirectional LSTM encoder and a two-layer LSTM decoder.

S2S+ATT Bahdanau et al. [1] introduced an attention mechanism to the Seq2Seq model to dynamically generate a context vector, which helped to find important inputs for a target word being generated. We implemented it as one of our baseline models.

S2S+COPY In several tasks, some output tokens are the same as some of the tokens in the input sequence. To deal with this problem, Gu et al. [6] incorporated a copying mechanism in the Seq2Seq model to locate a certain sub-sequence of the input sentence and place it into the output sequence. We implemented it as one of our baseline models.

S2S+PO-GEN See et al. [20] proposed a pointer-generator network architecture with coverage to jointly copy and generate a target from the input sequence and achieved abstractive state-of-the-art results. We implemented it as one of our baseline models.

5.3 Experimental Settings

For all the baseline models, we used same meaning representations and the same training-test split for our dataset. For our model and all the baseline models based on nearal networks, we followed See et al. [20] and set the word embedding size to 128, and all of the LSTM hidden state sizes to 256. For the hyper-parameter settings, the dropout rate was selected from {0.2, 0.3, 0.4, 0.5} and the batch size was selected from {32, 64, 96, 128 }. For the pointer-generator network, the source and target sequences shared a vocabulary of more than 20K words. We performed an evaluation in each epoch, and the models stopped training when the evaluation indicators of the validation set did not improve after 10 consecutive evaluations.

5.4 Implementation Details

We designed two levels of comparative experiments for all the baseline models. For *sentence-level experiments*, we split each math problem into multiple single sentences, and used each single sentence as the input of each model. For *problem-level experiments*, we treated each math problem as a long sentence, and used the entire math problem as the input to keep the information of the math problem.

For our model and all the baseline models, we reported the *sentence-level match accuracy* of test dataset as evaluation metric. The *sentence-level match accuracy* was defined as the proportion of the input sentences that were correctly parsed to their gold standard logical forms. That is, the logical

		Number of sentences					
		<=2	3	4	5	>=5	Overall
problems level	WAPS [30]	9.19	12.54	15.87	21.05	17.79	17.55
	Seq2Tree [2]	12.74	19.21	21.35	26.35	19.73	20.48
	Seq2Seq [25]	11.74	16.51	18.26	25.75	20.92	20.91
	S2S + ATT [1]	26.02	38.25	42.98	57.03	51.76	50.27
	S2S + COPY [6]	29.12	44.29	49.43	62.26	57.12	55.55
	S2S + PO-GEN [20]	29.79	40.95	47.75	59.32	55.11	53.51
sentences level	WAPS [30]	6.76	16.19	17.42	37.71	34.46	31.98
	Seq2Tree [2]	8.08	20.32	23.03	50.04	38.43	37.26
	Seq2Seq [25]	7.75	18.10	22.05	45.41	39.14	36.83
	S2S + ATT [1]	24.58	41.90	49.30	62.71	64.76	60.58
	S2S + COPY [6]	24.25	40.63	48.60	70.83	67.00	63.39
	S2S + PO-GEN [20]	29.24	50.63	55.76	65.83	67.13	63.64
Our Model		**34.88**	**54.60**	**59.27**	**72.89**	**73.22**	**69.67**

Table 3. Evaluation results on problem-level experiments and sentence-level experiments.

forms generated by the model are the same in both type and quantity as the gold standard logical forms. But the order of all logical forms of a sentence is not important. If a model's output logical forms satisfies this conditions,we think the output of this model is correct, otherwise it is an error output. For example, as shown in Figure 1, the sentence "Define a function $f(x) = lg(x + \frac{a}{x} - 2)$,where a is a constant greater than 0" correspond to three logical forms "Function $(f(x), lg(x+\frac{a}{x}-2))$; GreaterThan(a, 0); Constant(a);". The above sentence can also correspond to the sequence of logical forms "Constant(a); GreaterThan(a, 0); Function $(f(x), lg(x + \frac{a}{x} - 2))$;".

5.5 Main Results

We first evaluated the performance of our model and the baseline models on our dataset, and then performed a set of parametric experiments on our model and the best performing baseline models to show that the good performance of our model did not benefit from the introduction of more parameters.

Table 3 shows problem-level experiments and the single sentence-level experiments. We group the results according to the number of sentences included in the math problem, and our model performs best in every group. In addition, the performance of the baseline models at two levels of experimentation show that although these models lost contextual information when running at the single sentence-level, they still achieves better accuracy than at the problem-level. This is because most of the words in different sentences are not semantically related, and the unrelated words in different sentences will cause bad interference if we treat the entire math problem as a input. In comparison, our model parsed the math problems at the single sentence level, but introduced the important information of the entire math problem by using an additional encoder. As shown in Table 3, our model outperformed the best performing baseline model by up to 6.03% at the sentence-level accuracy.

Compared to the neural baseline models, our model contains an additional encoder that introduces more parameters. This increase in parameters could increase the expression ability of the neural network and produce better performance. We evaluated our model and the best performing neural baseline models of the two levels experiments on different parameter dimensions. In detail, we

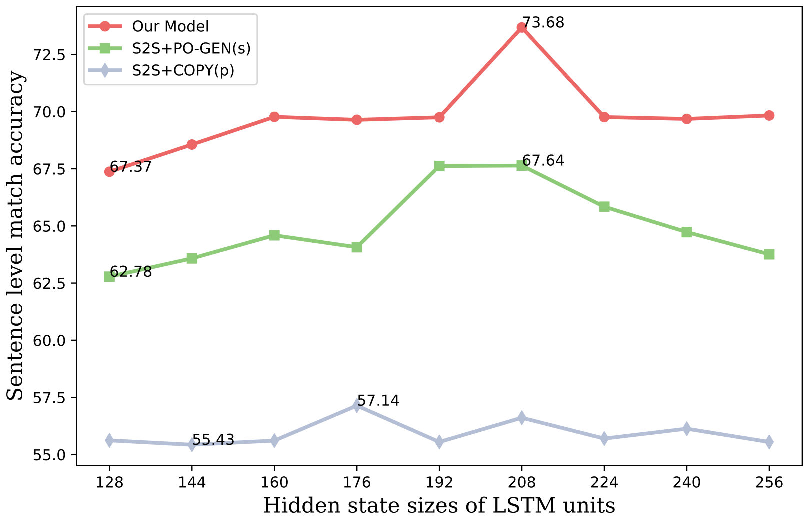

Fig. 3. Results of our model and the best performing baseline models influenced by different hidden state sizes of LSTM units. “p” and “s” represent the problem-level and the single sentence-level experiments respectively.

Inputs :

s1:	已知	双曲线	C :	formula	的	右准线	与	两	渐近线	交于	A B	两	点
	yizhi	shuangquxian	C :	formula	de	youzhunxian	yu	liang	jianjinxian	jiaoyu	A B	liang	dian

(Given a hyperbola C, its right directrix intersects with its two asymptotes at points A and B.)

s2:	其	右焦点	为	F
	qi	youjiaodian	wei	F

(and its right focus point is F)

Model	**Outputs of s2**
Golden :	RightfocusOfHyperbola (C , F) ;
S2S+COPY(p) :	None
S2S+PO-GEN(s) :	RightfocusOfHyperbola (rs_a , F) ;
Our Model :	RightfocusOfHyperbola (C , F) ;

Fig. 4. Example outputs from our model and the best baseline models. “p” and “s” represent the problem-level and the sentence-level experiments respectively.

set the hidden state sizes of the LSTM units to {128, 144, 160, 176, 192, 208, 224, 240, 256}. Figure 3 shows the sentence-level accuracy at each parameter dimension. We found that the best hidden state sizes for our model and baseline models were not the biggest LSTM hidden state size (256). And continuing to increase parameters would result in degraded model performance when the parameter is close to 256. Our model achieved the highest accuracy of 73.68% when the LSTM hidden state size was set to 208. It is worth noting that our model with the smallest LSTM hidden state size (128) had a similar accuracy with the best performance baseline model (Setting LSTM hidden state size to 208). These results showed that the good performance of our model did not benefit from the introduction of more parameters.

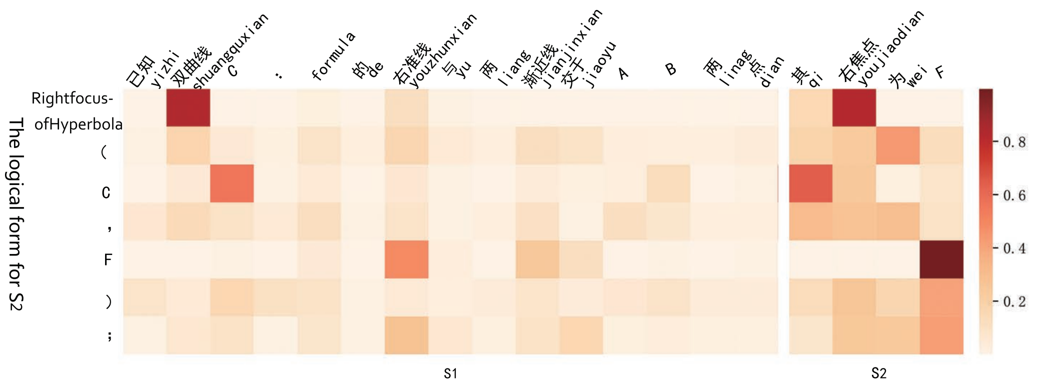

Fig. 5. Attention score matrices of two adjacent sentences in a math problem that computed by our model. Darker color represents higher attention score.

5.6 Effects of two-encoder architecture and word-level selective mechanism

We did additional experiments of our model and the best performing neural baseline models of the two levels experiments and analyses to demonstrate the effectiveness of our two-encoder architecture and word-level selective mechanism.

Figure 4 shows two adjacent sentences in a math problem of our dataset, the gold standard logical forms of the second sentence and the outputs of the second sentence that were generated by our model, the best problem level baseline model "S2S+COPY" and the best single sentence-level baseline model "S2S+PO-GEN". It is obvious that the word "qi" (its) in the second sentence means "shuangquxian" (hyperbola) in the first sentence. Thus, the parser needs to obtain information from the first sentence when parsing the second sentence. In theory, both our model and the problem-level baseline models could achieve this goal. However, in actuality, our model generated the correct logical forms of the second sentence, but the best problem-level baseline model did not generate any logical forms related to the second sentence. Obviously, the problem-level baseline model did not assign the alignment probability to each word very well. While the sentence-level baseline model could not obtain information from the first sentence, it generated the token "rs_a", instead of the "C" in the first sentence. The token "rs_a" is a collective term for pronouns in our dataset.

Figure 5 shows the alignment matrix produced by our model when parsing the second sentence to logical forms. The tokens above the picture are the words of the two sentences, and the tokens on the left are the logical forms of the second sentence generated by our model. Each cell in the alignment matrix corresponds to α_{ti}, which is computed by Equation 13. It can be seen that the semantically related words in different sentences are assigned higher alignment probabilities by our model, such as the alignments between "shuangquxian" (hyperbola) in the first sentence, "youjiaodian" (right focus point) in the second sentence, and predicate "RightfocusOfHyperbola" in the logical forms. These results showed that although our model parses the math problems at the single sentence-level, it does successfully introduce important information from the context sentences.

To verify the effectiveness of our word level selective mechanism, we used the following three settings:

1) Using a hidden state level selective mechanism (H_SEL) [37]

Models	**Accuracy**	
	one-encoder	two-encoder
Our Model w/o SEL	64.74	64.73
Our Model w/ H_SEL	67.23	66.64
Our Model w/ W_SEL	**69.83**	**69.70**

Table 4. Evaluation results of our model on three types of selective mechanism settings. "w/o" and "w/" represent "without" and "with" respectively.

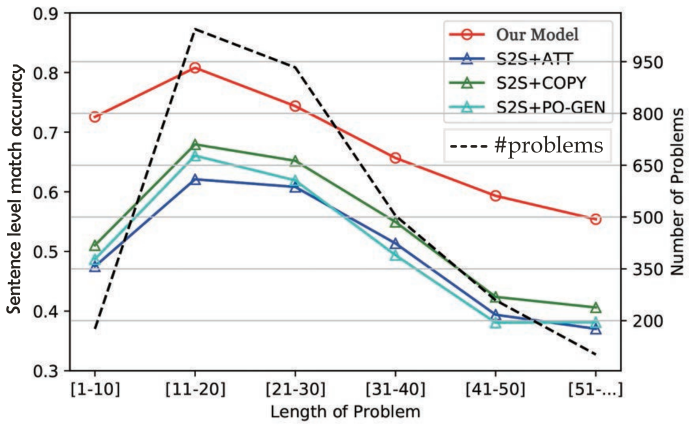

Fig. 6. Sentence-level accuracy of each model in different length of math problems. The black dash line shows the number of math problems in each group.

2) Using our word-level selective mechanism (W_SEL)

3) Not using either of the two selective mechanisms, but simply using the original embedding of each word for the context sentence.

And for the setting of H_SEL and W_SEL, we choose to use them in both two encoder and only the second encoder. Table 4 lists the results of our model with these three settings. Our word-level selective mechanism performed the best and outperformed the hidden state-level selective mechanism by up to 2.60% at the sentence level accuracy. Besides, as shown in Table 4, it is better to use the selection mechanism only for the second model. We made some error analysis for this situation and found that using selective mechanism in both encoder may lead our model ignore some important information and make "under-translation" problem[27] even worse.

These experiments showed that both our two-encoder architecture and word-level selective mechanism could bring significant improvement.

5.7 Effects of Problem Length

The sequence to sequence model may face "under-translation" and "over-translation" problems when the length of input sequence is long enough[27] . So we did did additional experiments on our model and the baseline models based on sequence to sequence framework to show the effects of problem length.

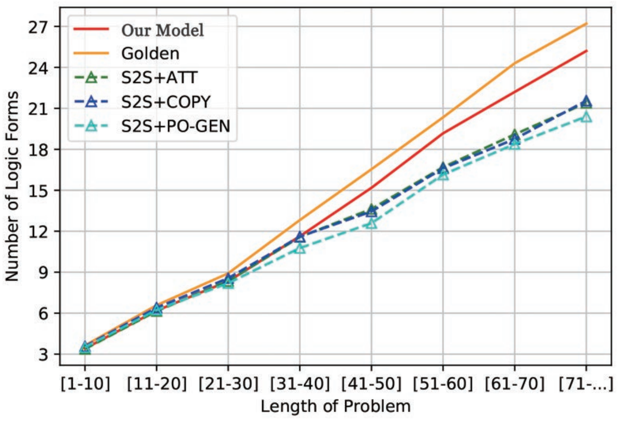

Fig. 7. Number of logical forms generated by each model in different length of math problems.

We group sentences of similar lengths together and compute sentence-level precision for each group. As shown in Figure 6, our model outperforms all baseline models on each group. Moreover, the performance of all models decreases when the length of input problem increases. One main reason is probably related to the size of each group. We count the number of math problems in each group and mark it in the Figure 6, the result show that the performance of all models is positively correlated with the size of each group.

It is noteworthy that when the length of math problem exceeds 40, the performance of all baseline models deteriorates rapidly and our model is significantly superior to them. What's more is that the baseline models are similar to our model in terms of precision in logical form-level, but they lag far behind our model in terms of recall rate. Due to the above two observations, we suppose that when the math problem is long enough, these baseline models may have under-translation problems. To verify this assumptions, we calculate the average number of valid logical forms obtained by each models and golden output. The results in Figure 7 shown that baseline models gets less valid logical forms then our model when the length of math problem exceeds 40. Our model works better on long math problems and can better deal with the under-translation problem.

5.8 Error Analysis

Finally, we analyzed the wrong results of our model and summarized the most common cause of errors below.

Under-Mapping Although our model can better deal with the under-mapping problem than our baseline models, it still cannot be completely avoided. Some source words may be ignored in the decoding process. For example, in some cases our model may get the logical form “Angle(B)” instead of the gloden output “InteriorAngle(B)”. Taking the alignment history into consideration may be a good way to solve this problem [27].

Brackets Not Paired We don't have a good strategy to deal with the problem of "brackets not paired", although this problem does not occur often. For example, in some cases our model may get the output “StringOfCircle(C, AC”, but “StringOfCircle(C, AC’)’ is the gloden output. Adding some rules in our model may be an effective solution [29].

6 CONCLUSION

In this study, we focused on multi-sentence-level semantic parsing and proposed a novel two-encoder architecture sequence to sequence model to automatically map math problems to their logical forms. In addition, we designed a word-level selective mechanism that determines the importance of each source-side word at the embedding layer. A large dataset was constructed for model training and empirical evaluation. The experimental results showed that both the two-encoder architecture and word-level selective mechanism could bring significant improvement.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers for their helpful comments.

This work was partially funded by National Natural Science Foundation of China (No. 61751201, 61532011), and STCSM (No.16JC1420401,17JC1420200).

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Li Dong and Mirella Lapata. 2016. Language to Logical Form with Neural Attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 33–43.

[3] Jacob Eisenstein, James Clarke, Dan Goldwasser, and Dan Roth. 2009. Reading to learn: Constructing features from semantic abstracts. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*. Association for Computational Linguistics, 958–967.

[4] Edward A Feigenbaum, Julian Feldman, et al. 1963. *Computers and thought*. New York.

[5] Ruifang Ge and Raymond J Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the ninth conference on computational natural language learning*. Association for Computational Linguistics, 9–16.

[6] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 1631–1640.

[7] Jonathan Herzig and Jonathan Berant. 2017. Neural Semantic Parsing over Multiple Knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Vol. 2. 623–628.

[8] Mark Hopkins, Cristian Petrescu-Prahova, Roie Levin, Ronan Le Bras, Alvaro Herrasti, and Vidur Joshi. 2017. Beyond sentential semantic parsing: Tackling the math sat with a cascade of tree transducers. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 795–804.

[9] Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. How well do computers solve math word problems? Large-scale Dataset construction and evaluation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 887–896.

[10] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 963–973.

[11] Rohit J Kate and Raymond J Mooney. 2007. Learning language semantics from ambiguous supervision. In *AAAI*, Vol. 7. 895–900.

[12] Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 1516–1526.

[13] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 271–281.

[14] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 23–33.

[15] Christian Liguda and Thies Pfeiffer. 2012. Modeling math word problems with augmented semantic networks. In *International Conference on Application of Natural Language to Information Systems*. Springer, 247–252.

[16] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 158–167.

[17] Takuya Matsuzaki, Takumi Ito, Hidenao Iwane, Hirokazu Anai, and Noriko H Arai. 2017. Semantic parsing of pre-university math problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 2131–2141.

[18] Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 2144–2153.

[19] Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association of Computational Linguistics* 2, 1 (2014), 377–392.

[20] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 1073–1083.

[21] Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 1466–1476.

[22] Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 1132–1142.

[23] Yu Su and Xifeng Yan. 2017. Cross-domain Semantic Parsing via Paraphrasing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 1235–1246.

[24] Raymond Hendy Susanto and Wei Lu. 2017. Neural Architectures for Multilingual Semantic Parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Vol. 2. 38–44.

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

[26] Lappoon R Tang and Raymond J Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*. Springer, 466–477.

[27] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling Coverage for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 76–85.

[28] Shyam Upadhyay, Ming-Wei Chang, Kai-Wei Chang, and Wen-tau Yih. 2016. Learning from explicit and implicit supervision jointly for algebra word problems. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 297–306.

[29] Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep Neural Solver for Math Word Problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 845–854.

[30] Yuk Wah Wong and Raymond J Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*. Association for Computational Linguistics, 439–446.

[31] William A Woods. 1973. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the June 4-8, 1973, national computer conference and exposition*. ACM, 441–450.

[32] John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*. 1050–1055.

[33] Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

[34] Luke S Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 658–666.

[35] Yuchen Zhang, Panupong Pasupat, and Percy Liang. 2017. Macro Grammars and Holistic Triggering for Efficient Semantic Parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 1214–1223.

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

[37] Qingyu Zhou, Nan Yang, Furu Wei, and Ming Zhou. 2017. Selective Encoding for Abstractive Sentence Summarization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 1095–1104.

accepted January 2019