Semantically-Aligned Equation Generation for Solving and Reasoning Math Word Problems

Ting-Rui Chiang **Yun-Nung Chen**
National Taiwan University, Taipei, Taiwan
r07922052@csie.ntu.edu.tw y.v.chen@ieee.org

Abstract

Solving math word problems is a challenging task that requires accurate natural language understanding to bridge natural language texts and math expressions. Motivated by the intuition about how human generates the equations given the problem texts, this paper presents a neural approach to automatically solve math word problems by operating symbols according to their semantic meanings in texts. This paper views the process of generating equations as a bridge between the semantic world and the symbolic world, where the proposed neural math solver is based on an encoder-decoder framework. In the proposed model, the encoder is designed to understand the semantics of problems, and the decoder focuses on tracking semantic meanings of the generated symbols and then deciding which symbol to generate next. The preliminary experiments are conducted in a benchmark dataset Math23K, and our model significantly outperforms both the state-of-the-art single model and the best non-retrieval-based model over about 10% accuracy, demonstrating the effectiveness of bridging the symbolic and semantic worlds from math word problems.[1]

1 Introduction

Automatically solving math word problems has been an interesting research topic and also been viewed as a way of evaluating machines' ability (Mandal and Naskar, 2019). For human, writing down an equation that solves a math word problem requires the ability of reading comprehension, reasoning, and sometimes real world understanding. Specifically, to solve a math word problem, we first need to know the goal of the given problem, then understand the semantic meaning of each numerical number in the problem, perform reasoning based on the comprehension in the previous step, and finally decide what to write in the equation.

Most prior work about solving math word problems relied on hand-crafted features, which required more human knowledge. Because those features are often in the lexical level, it is not clear whether machines really understand the math problems. Also, most prior work evaluated their approaches on relatively small datasets, and the capability of generalization is concerned.

This paper considers the reasoning procedure when writing down the associated equation given a problem. Figure 1 illustrates the problem solving process. The illustration shows that human actually assigns the semantic meaning to each number when manipulating symbols, including operands (numbers) and operators ($+ - \times \div$). Also, we believe that the semantic meaning of operands can help us decide which operator to use. For example, the summation of "*price of one pen*" and "*number of pens Tom bought*" is meaningless; therefore the addition would not be chosen.

Following the observation above, this paper proposes a novel encoder decoder model, where the encoder extracts semantic meanings of numbers in the problem, and the decoder is equipped with a stack that facilitates tracking the semantic meanings of operands. The contributions of this paper are 4-fold:

- This paper is the first work that models semantic meanings of operands and operators for math word problems.
- This paper proposes an end-to-end neural math solver with a novel decoding process that utilizes the stack to generate associated equations.

[1]The source code is available at https://github.com/MiuLab/E2EMathSolver.

Proceedings of NAACL-HLT 2019, pages 2656–2668
Minneapolis, Minnesota, June 2 - June 7, 2019.

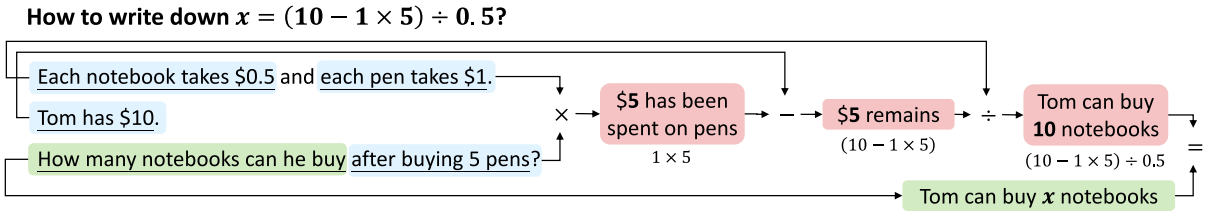

Figure 1: The solving process of the math word problem "*Each notebok takes $0.5 and each pen takes $1. Tom has $10. How many notebook can he buy after buying 5 pens?*" and the associated equation is $x = (10 - 1 \times 5) \div 0.5$. The associated equation is $x = (10 - 1 \times 5) \div 0.5$.

- This paper achieves the state-of-the-art performance on the large benchmark dataset Math23K.
- This paper is capable of providing interpretation and reasoning for the math word problem solving procedure.

2 Related Work

There is a lot of prior work that utilized hand-crafted features, such as POS tags, paths in the dependency trees, keywords, etc., to allow the model to focus on the quantities in the problems (Kushman et al., 2014; Hosseini et al., 2014; Roy et al., 2015; Roy and Roth, 2015; Koncel-Kedziorski et al., 2015; Roy et al., 2016; Upadhyay et al., 2016; Upadhyay and Chang, 2017; Roy and Roth, 2018; Wang et al., 2018). Recently, Mehta et al.; Wang et al.; Ling et al. attempted at learning models without predefined features. Following the recent trend, the proposed end-to-end model in this paper does not need any hand-crafted features.

Kushman et al. first extracted templates about math expressions from the training answers, and then trained models to select templates and map quantities in the problem to the slots in the template. Such two-stage approach has been tried and achieved good results (Upadhyay and Chang, 2017). The prior work highly relied on human knowledge, where they parsed problems into equations by choosing the expression tree with the highest score calculated by an operator classifier, working on a hand-crafted "trigger list" containing quantities and noun phrases in the problem, or utilizing features extracted from text spans (Roy et al., 2015, 2016; Koncel-Kedziorski et al., 2015). Shi et al. defined a Dolphin language to connect math word problems and logical forms, and generated rules to parse math word problems. Upadhyay et al. parsed math word problems without explicit equation annotations. Roy and Roth classified math word problems into 4 types and used rules to decide the operators accordingly. Wang et al. trained the parser using reinforcement learning with hand-crafted features. Hosseini et al. modeled the problem text as transition of world states, and the equation is generated as the world states changing. Our work uses a similar intuition, but hand-crafted features are not required and our model can be trained in an end-to-end manner. Some end-to-end approaches have been proposed, such as generating equations directly via a seq2seq model (Wang et al., 2017). Ling et al. tried to generate solutions along with its rationals with a seq2seq-like model for better interpretability.

This paper belongs to the end-to-end category, but different from the previous work; we are the first approach that generates equations with stack actions, which facilitate us to simulate the way how human solves problems. Furthermore, the proposed approach is the first model that is more interpretable and provides reasoning steps without the need of rational annotations.

3 End-to-End Neural Math Solver

Our approach composes of two parts, an encoder and a decoder, where the process of solving math word problems is viewed as transforming multiple text spans from the problems into the target information the problems ask for. In the example shown in Figure 1, all numbers in the problem are attached with the associated semantics. Motivated by the observation, we design an encoder to extract the semantic representation of each number in the problem text. Considering that human usually manipulates those numbers and operators (such as addition, subtraction, etc.) based on their semantics for problem solving, a decoder is designed to construct the equation, where the semantics is aligned with the representations extracted by the encoder. The idea of the proposed model

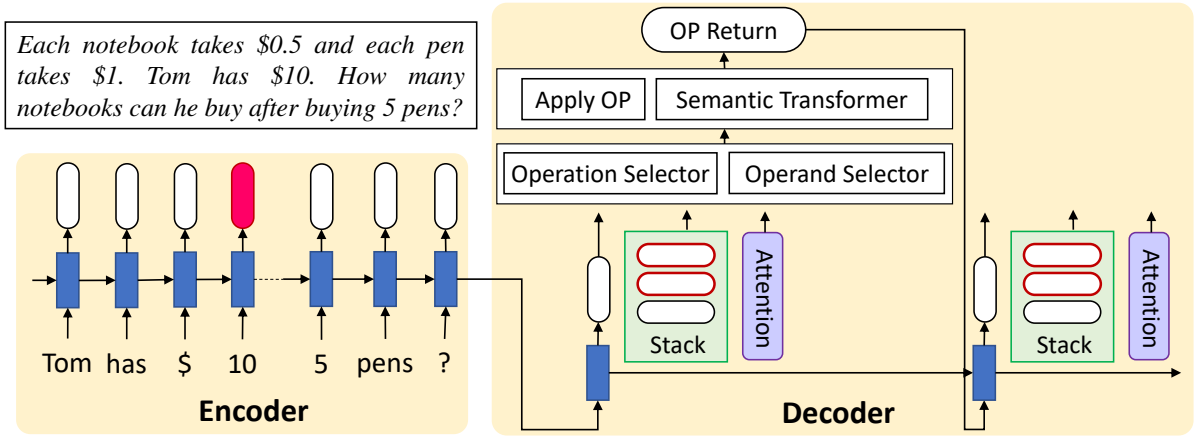

Figure 2: The encoder-decoder model architecture of the proposed neural solver machine.

is to imitate the human reasoning process for solving math word problems. The model architecture is illustrated in Figure 2.

3.1 Encoder

The encoder aims to extract the semantic representation of each constant needed for solving problems. However, the needed constants may come from either the given problem texts or domain knowledge, so we detail these two procedures as follows.

3.1.1 Constant Representation Extraction

For each math word problem, we are given a passage consisting of words $\{w_t^P\}_{t=1}^m$, whose word embeddings are $\{e_t^P\}_{t=1}^m$. The problem text includes some numbers, which we refer as constants. The positions of constants in the problem text are denoted as $\{p_i\}_{i=1}^n$. In order to capture the semantic representation of each constant by considering its contexts, a bidirectional long short-term memory (BLSTM) is adopted as the encoder (Hochreiter and Schmidhuber, 1997):

$$h_t^E, c_t^E = \text{BLSTM}(h_{t-1}^E, c_{t-1}^E, e_t^P), \quad (1)$$

and then for the i-th constant in the problem, its semantic representation e_i^c is modeled by the corresponding BLSTM output vector:

$$e_i^c = h_{p_i}^E. \quad (2)$$

3.1.2 External Constant Leveraging

External constants, including 1 and π, are leveraged, because they are required to solve a math word problem, but not mentioned in the problem text. Due to their absence from the problem text, we cannot extract their semantic meanings by BLSTM in (2). Instead, we model their semantic representation e^π, e^1 as parts of the model parameters. They are randomly initialized and are learned during model training.

3.2 Decoder

The decoder aims at constructing the equation that can solve the given problem. We generate the equation by applying stack actions on a stack to mimic the way how human understands an equation. Human knows the semantic meaning of each term in the equation, even compositing of operands and operators like the term "$(10-1\times5)$" in Figure 1. Then what operator to apply on a pair operands can be chosen based on their semantic meanings accordingly. Hence we design our model to generate the equation in a postfix manner: a operator is chosen base on the semantic representations of two operands the operator is going to apply to. Note that the operands a operator can apply to can be any results generated previously. That is the reason why we use "stack" as our data structure in order to keep track of the operands a operator is going to apply to. The stack contains both symbolic and semantic representations of operands, denoted as

$$S = [(v_{l_t}^S, e_{l_t}^S), (v_{l_t-1}^S, e_{l_t-1}^S), \cdots, (v_1^S, e_1^S)], \quad (3)$$

where v^S of each pair is the symbolic part, such as $x+1$, while e^S is the semantic representation, which is a vector. The components in the decoder are shown in the right part of Figure 2, each of which is detailed below.

3.3 Decoding State Features

At each decoding step, decisions are made based on features of the current state. At each step, fea-

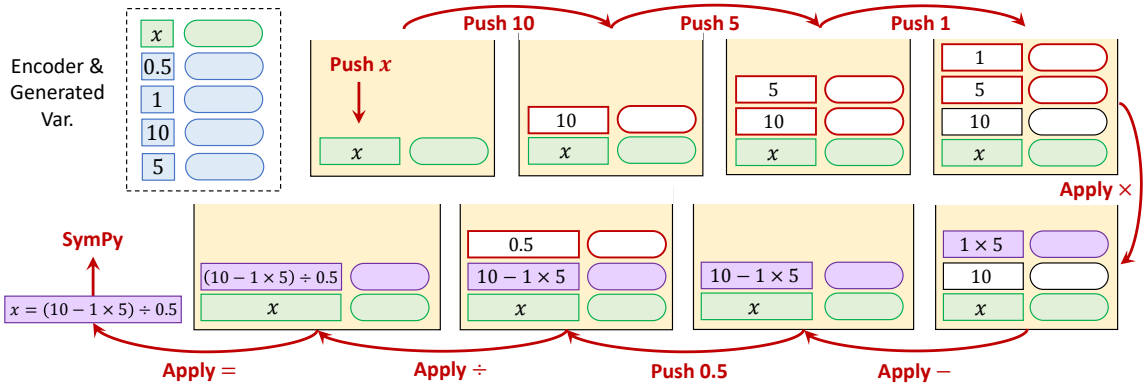

Figure 3: Illustration of the inference process. The purple round blocks denote the transformed semantics, while the green ones are generated by the variable generator.

tures r^{sa} and r^{opd} are extracted to select a stack action (section 3.3.2) and an operand to push (section 3.3.3). Specifically, the features are the gated concatenation of following vectors:

- h_t^D is the output of an LSTM, which encodes the history of applied actions:

$$h_t^D, c_t^D = \text{LSTM}(h_{t-1}^D, c_{t-1}^D, \text{res}_{t-1}), \quad (4)$$

  where res_{t-1} is the result from the previous stack action similar to the seq2seq model (Sutskever et al., 2014). For example, if the previous stack action o_{t-1} is "push", then res_{t-1} is the semantic representation pushed into the stack. If the previous stack action o_{t-1} is to apply an operator $\diamond$, then res_{t-1} is the semantic representation generated by $f_\diamond$.

- s_t is the stack status. It is crucial because some operators are only applicable to certain combinations of operand semantics, which is similar to the type system in programming languages. For example, operating multiplication is applicable to the combination of "*quantity of an item*" and "*price of an item*", while operating addition is not. Considering that all math operators supported here $(+, -, \times, \div)$ are binary operators, the semantic representations of the stack's top 2 elements at the time $t-1$ are considered:

$$s_t = [e_{l_t}^S; e_{l_t}^S]. \quad (5)$$

- q_t incorporates problem information in the decision. It is believed that the attention mechanism (Luong et al., 2015) can effectively capture dependency for longer distance. Thus, the attention mechanism over the encoding problem $h_1^E, h_2^E, \cdots$ is adopted:

$$q_t = \text{Attention}(h_t^D, \{h_i^E\}_{i=1}^m), \quad (6)$$

where the attention function in this paper is defined as a function with learnable parameters w, W, b:

$$\text{Attention}(u, \{v_i\}_{i=1}^m) = \sum_{i=1}^m \alpha_i h_i, \quad (7)$$

$$\alpha_i = \frac{\exp(s_i)}{\sum_{l=1}^m \exp(s_i)}, \quad (8)$$

$$s_i = w^\mathsf{T} \tanh(W^T[u; v_i] + b). \quad (9)$$

In order to model the dynamic features for different decoding steps, features in r_t^{sa} is gated as follows:

$$r_t^{sa} = [g_{t,1}^{sa} \cdot h_t^D; g_{t,2}^{sa} \cdot s_t; g_{t,3}^{sa} \cdot q_t], \quad (10)$$

$$g_t^{sa} = \sigma(W^{sa} \cdot [h_t^D; s_t; q_t]), \quad (11)$$

where σ is a sigmoid function and W^{sa} is a learned gating parameter. r_t^{opd} is defined similarly, but with a different learned gating parameter W^{opd}.

3.3.1 Stack Action Selector

The stack action selector is to select an stack action at each decoding step (section 3.3.2) until the unknowns are solved. The probability of choosing action a at the decoding step t is calculated with a network NN constituted of one hidden layer and ReLU as the activation function:

$$\begin{aligned} &P(Y_t \mid \{y_i\}_{i=1}^{t-1}, \{w_i\}_{i=1}^m) \quad (12) \\ &\quad = \text{StackActionSelector}(r_t^{sa}) \\ &\quad = \text{softmax}(\text{NN}(r_t^{sa})), \end{aligned}$$

where r_t^{sa} is decoding state features as defined in section 3.3.

3.3.2 Stack Actions

The available stack actions are listed below:

- **Variable generation**: The semantic representation of an unknown variable x is generated dynamically as the first action in the decoding process. Note that this procedure provides the flexibility of solving problems with more than one unknown variables. The decoder module can decide how many unknown variables are required to solve the problem, and the semantic representation of the unknown variable is generated with an attention mechanism:

$$e^x = \text{Attention}(h_t^D, \{h_i^E\}_{i=1}^m). \quad (13)$$

- **Push**: This stack action pushes the operand chosen by the operand selector (section 3.3.3). Both the symbolic representation v_* and semantic representation e_* of the chosen operand would be pushed to the stack S in (3). Then the stack state becomes

$$S = [(v_*^S, e_*^S), (v_{l_t}^S, e_{l_t}^S), \cdots, (v_1^S, e_1^S)]. \quad (14)$$

- **Operator $\diamond$ application ($\diamond \in \{+, -, \times, \div\}$)**: One stack action pops two elements from the top of the stack, which contains two pairs, (v_i, e_i) and (v_j, e_j), and then the associated symbolic operator, $v_k = v_i \diamond v_j$, is recorded. Also, a semantic transformation function $f_\diamond$ for that operator is invoked, which generates the semantic representation of v_k by transforming semantic representations of v_i and v_j to $e_k = f_\diamond(e_i, e_j)$. Therefore, after an operator is applied to the stack specified in (3), the stack state becomes

$$\begin{aligned} S = &[(v_{l_t}^S \diamond v_{l_t-1}^S, f_\diamond(e_{l_t}^S, e_{l_t-1}^S)), \\ &(v_{l_t-2}^S, e_{l_t-2}^S), \cdots, (v_1^S, e_1^S)]. \end{aligned} \quad (15)$$

- **Equal application**: When the equal application is chosen, it implies that an equation is completed. This stack action pops 2 tuples from the stack, $(v_i, e_i), (v_j, e_j)$, and then $v_i = v_j$ is recorded. If one of them is an unknown variable, the problem is solved. Therefore, after an OP is applied to the stack specified in (3), the stack state becomes

$$S = [(v_{l_t-2}^S, e_{l_t-2}^S), \cdots, (v_1^S, e_1^S)]. \quad (16)$$

3.3.3 Operand Selector

When the stack action selector has decided to push an operand, the operand selector aims at choosing which operand to push. The operand candidates e include constants provided in the problem text whose semantic representations are $e_1^c, e_2^c, \cdots, e_n^c$, unknown variable whose semantic representation is e^x, and two external constants 1 and π whose semantic representations are e^1, e^π:

$$e = [e_1^c, e_2^c, \cdots, e_n^c, e^1, e^\pi, e^x]. \quad (17)$$

An operand has both symbolic and semantic representations, but the selection focuses on its semantic meaning; this procedure is the same as what human does when solving math word problems.

Inspired by addressing mechanisms of neural Turing machine (NTM) (Graves et al., 2014), the probability of choosing the i-th operand candidate is the attention weights of r_t over the semantic representations of the operand candidates as in (8):

$$\begin{aligned} &P(Z_t \mid \{y_i\}_{i=1}^{t-1}, \{w_i\}_{i=1}^m) \quad (18) \\ &= \text{OperandSelector}(r_t^{opd}) \\ &= \text{AttentionWeight}(r_t^{opd}, \{e_i\}_{i=1}^m \cup \{e^1, e^\pi, e^x\}), \end{aligned}$$

and r_t^{opd} is defined in section 3.3.

3.3.4 Semantic Transformer

A semantic transformer is proposed to generate the semantic representation of a new symbol resulted from applying an operator, which provides the capability of interpretation and reasoning for the target task. The semantic transformer for an operator $\diamond \in \{+, -, \times, \div\}$ transforms semantic representations of two operands e_1, e_2 into

$$f_\diamond(e_1, e_2) = \tanh(U_\diamond \text{ReLU}(W_\diamond[e_1; e_2] + b_\diamond) + c_\diamond), \quad (19)$$

where $W_\diamond, U_\diamond, b_\diamond, c_\diamond$ are model parameters. Semantic transformers for different operators have different parameters in order to model different transformations.

3.4 Training

Both stack action selection and operand selection can be trained in a fully supervised way by giving problems and associated ground truth equations. Because our model generates the equation with stack actions, the equation is first transformed into its postfix representation. Let the postfix representation of the target equation be $y_1, \cdots y_t, \cdots, y_T$,

where y_t can be either an operator $(+, -, \times, \div, =)$ or a target operand. Then for each time step t, the loss can be computed as

$$L(y_t) = \begin{cases} L_1(\text{push\_op}) + L_2(y_t) & y_t \text{ is an operand} \\ L_1(y_t) & \text{otherwise} \end{cases},$$

where L_1 is the stack action selection loss and L_2 is the operand selection loss defined as

$$L_1(y_t) = -\log P(Y_t = y_t \mid \{o_i\}_{i=1}^{t-1}, \{w_i\}_{i=1}^{m}),$$
$$L_2(y_t) = -\log P(Z_t = y_t \mid r_t).$$

The objective of our training process is to minimize the total loss for the whole equation, $\sum_{t=1}^{T} L(y_t)$.

3.5 Inference

When performing inference, at each time step t, the stack action with the highest probability $P(Y_t|\{\tilde{y}_i\}_{i=1}^{t-1}, \{w_i\}_{i=1}^{m})$ is chosen. If the chosen stack action is "*push*", the operand with the highest probability $P(Z_t|\{\tilde{Y}_i\}_{i=1}^{t-1}, \{w_i\}_{i=1}^{m})$ is chosen. When the stack has less than 2 elements, the probability of applying operator $+, -, \times, \div, =$ would be masked out to prevent illegal stack actions, so all generated equations must be legal math expressions. The decoder decodes until the unknown variable can be solved. After the equations are generated, a Python package SymPy (Meurer et al., 2017) is used to solve the unknown variable. The inference procedure example is illustrated in Figure 3. The detailed algorithm can be found in Algorithm 1.

Algorithm 1 Training and Inference

function SOLVEPROBLEM(problem_text)
  $v \leftarrow$ ExtractConstants(problem_text)
    ▷ v is a list of constants in the problem.
  $h^E, h_0^D, c_0^D, E \leftarrow$ Encoder(problem_text)
  $S \leftarrow$ Stack()
  ret, loss, t, equations $\leftarrow$ padding, 0, 1, {}
  while not solvable(equations) **do**
    $h_t^D \leftarrow \text{LSTM}(h_{t-1}^D, c_{t-1}, \text{ret})$
    $s_t \leftarrow S.\text{get\_top2}()$
    $h^E \leftarrow \text{Attention}(h_{t-1}^D, h^E)$
    $r_t \leftarrow [h_t^D, s_t, h^E]$
    $p_{sa} \leftarrow \text{StackActionSelector}(r_t)$
    $p_{opd} \leftarrow \text{OperandSelector}(r_t)$
    if training **then**
      ▷ Target equation y is available when training.
      $Y_t \leftarrow y_t$
      if y_t is operand **then**
        loss $\leftarrow$ loss $+ L_1(\text{push}) + L_2(y_t)$
      else
        loss $\leftarrow$ loss $+ L_1(y_t)$
      end if
    else
      $Y_t \leftarrow \text{StackActionSelector}(r_t^{sa})$
      if $Y_t = \text{push}$ **then**
        $Z_t \leftarrow \text{OperandSelector}(r_t^{opd})$
      end if
    end if
    if $Y_t = \text{gen\_var}$ **then**
      $e^x \leftarrow \text{Attention}(h_t^D, h^E)$
      ret $\leftarrow e^x$
    else if $Y_t = \text{push}$ **then**
      $S.\text{push}(v_{Z_t}, e_{Z_t})$
      ret $\leftarrow e_{Z_t}$
    else if $Y_t \in \{+, -, \times, \div\}$ **then**
      $(v_a, e_a), (v_b, e_b) = S.\text{pop}(), S.\text{pop}()$
      $S.\text{push}(v_a Y_t v_b, f_{Y_t}(e_a, e_b))$
      ret $\leftarrow f_{Y_t}(e_a, e_b)$
    else if $Y_t = \text{equal}$ **then**
      $(v_a, e_a), (v_b, e_b) = S.\text{pop}(), S.\text{pop}()$
      equations = equations $\cup$ "$v_a = v_b$"
      ret $\leftarrow S.\text{top}()$
    end if
  end while
  return solve(equations)
end function

4 Experiments

To evaluate the performance of the proposed model, we conduct the experiments on the benchmark dataset and analyze the learned semantics.

4.1 Settings

The experiments are benchmarked on the dataset Math23k (Wang et al., 2017), which contains 23,162 math problems with annotated equations. Each problem can be solved by a single-unknown-variable equation and only uses operators $+, -, \times, \div$. Also, except π and 1, quantities in the equation can be found in the problem text. There are also other large scale datasets like Dolphin18K (Shi et al., 2015) and AQuA (Ling et al., 2017), containing 18,460 and 100,000 math word problems respectively. The reasons about not evaluating on these two datasets are 1) Dolphin18k contains some unlabeled math word problems and some incorrect labels, and 2) AQuA contains rational for solving the problems, but the equations in the rational are not formal (e.g. mixed with texts, using x to represent $\times$, etc.) and inconsistent. Therefore, the following experiments are performed and analyzed using Math23K, the only large scaled, good-quality dataset.

4.2 Results

The results are shown in Table 1. The retrieval-based methods compare problems in test data with problems in training data, and choose the most

Model		Accuracy
Retrieval	Jaccard	47.2%
	Cosine	23.8%
Classification	BLSTM	57.9%
	Self-Attention	56.8%
Generation	Seq2Seq w/ SNI	58.1%
	Proposed Word-Based	**65.3%**
	Proposed Char-Based	**65.8%**
Hybrid	Retrieval + Seq2Seq	64.7%

Table 1: 5-fold cross validation results on Math23K.

Model	Accuracy
Char-Based	65.8%
Word-Based	65.3%
Word-Based - Gate	64.1%
Word-Based - Gate - Attention	62.5%
Word-Based - Gate - Attention - Stack	60.1%
Word-Based - Semantic Transformer	64.1%
Word-Based - Semantic Representation	61.7%

Table 2: 5-fold cross validation results of ablation tests.

similar one's template to solve the problem (Kushman et al., 2014; Upadhyay and Chang, 2017). The classification-based models choose equation templates by a classifier trained on the training data. Their performance are reported in Robaidek et al.. The seq2seq and hybrid models are from Wang et al., where the former directly maps natural language into symbols in equations, and the latter one ensembles prediction from a seq2seq model and a retrieval-based model. The ensemble is the previous state-of-the-art results of Math23K.

Our proposed end-to-end model belongs to the generation category, and the single model performance achieved by our proposed model is new state-of-the-art ($> 65\%$) and even better than the hybrid model result (64.7%). In addition, we are the first to report character-based performance on this dataset, and the character-based results are slightly better than the word-based ones. Among the single model performance, our models obtain about more than 7% accuracy improvement compared to the previous best one (Wang et al., 2017). The performance of our character-based model also shows that our model is capable of learning the relatively accurate semantic representations without word boundaries and achieves better performance.

4.3 Ablation Test

To better understand the performance contributed by each proposed component, we perform a series of ablation tests by removing components one by one and then checking the performance by 5-fold cross validation. Table 2 shows the ablation results.

Char-Based v.s. Word-Based As reported above, using word-based model instead of character-based model only causes 0.5% performance drop. To fairly compare with prior word-based models, the following ablation tests are performed on the word-based approach.

Word-Based - Gate It uses r_t instead of r_t^{sa} and r_t^{opr} as the input of both $\mathrm{StackActionSelector}$ and $\mathrm{OperandSelector}$.

Word-Based - Gate - Attention Considering that the prior generation-based model (seq2seq) did not use any attention mechanism, we compare the models with and without the attention mechanism. Removing attention means excluding q_{t-1} in (11), so the input of both operator and operand selector becomes $r_t = [h_t^D; s_t]$. The result implies that our model is not better than previous models solely because of the attention.

Word-Based - Gate - Attention - Stack To check the effectiveness of the stack status (s_t in (11)), the experiments of removing the stack status from the input of both operator and operand selectors ($r_t = h_t^D$) are conducted. The results well justify our idea of choosing operators based on semantic meanings of operands.

Word-Based - Semantic Transformer To validate the effectiveness of the idea that views an operator as a semantic transformer, we modify the semantic transformer function of the operator $\diamond$ into $f_\diamond(e_1, e_2) = e_\diamond$, where $e_\diamond$ is a learnable parameter and is different for different operators. Therefore, $e_\diamond$ acts like the embedding of the operator $\diamond$, and the decoding process is more similar to a general seq2seq model. The results show that the semantic transformer in the original model encodes not only the last operator applied on the operands but other information that helps the selectors.

Word-Based - Semantic Representation To explicitly evaluate the effectiveness of operands' semantic representations, we rewrite semantic representation of the i-th operand in the problem texts

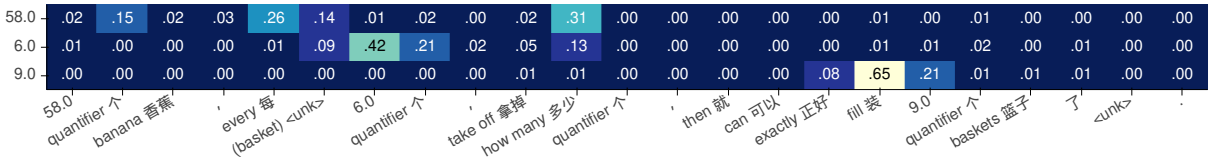

Figure 4: The self-attention map visualization of operands' semantic expressions for the problem "*There are 58 bananas. Each basket can contain 6 bananas. How many bananas are needed to be token off such that exactly 9 baskets are filled?*".

from (2) to $e_i^c = b_i^c$, where b_i^c is a parameter. Thus for every problem, the representation of the i-th operand is identical, even though their meanings in different problems may be different. This modification assumes that no semantic information is captured by b_i^c, which can merely represent a symbolic placeholder in an equation. Because the semantic transformer is to transform the semantic representations, applying this component is meaningless. Here the semantic transformer is also replaced with $f_\diamond(e_1, e_2) = e_\diamond$ as the setting of the previous ablation test. The results show that the model without using semantic representations of operands causes a significant accuracy drop of 3.5%. The main contribution of this paper about modeling semantic meanings of symbols is validated and well demonstrated here.

5 Qualitative Analysis

To further analyze whether the proposed model can provide interpretation and reasoning, we visualize the learned semantic representations of constants to check where the important cues are,

5.1 Constant Embedding Analysis

To better understand the information encoded in the semantic representations of constants in the problem, a self-attention is performed when their semantic representations are extracted by the encoder. Namely, we rewrite (2) as

$$e_i^c = \text{Attention}(h_{p_i}^E, \{h_t^E\}_{t=1}^m. \quad (20)$$

Then we check the trained self-attention map (α in the attention function) on the validation dataset.

For some problems, the self-attention that generates semantic representations of constants in the problem concentrates on the number's quantifier or unit, and sometimes it also focuses on informative verbs, such as "*gain*", "*get*", "*fill*", etc., in the sentence. For example, Figure 4 shows the attention weights for an example math word problem, where lighter colors indicate higher weights. The numbers "*58*" and "*6*" focus more on the quantifier-related words (e.g. "*every*" and "*how many*"), while "*9*" pays higher attention to the verb "*fill*". The results are consistent with those handcraft features for solving math word problems proposed by the prior research (Hosseini et al., 2014; Roy and Roth, 2015; Roy et al., 2015). Hence, we demonstrate that the automatically learned semantic representations indeed capture critical information that facilitates solving math word problems without providing human-crafted knowledge.

5.2 Decoding Process Visualization

We visualize the attention map (q_t in (6)) to see how the attention helps the decoding process. An example is shown in the top of Figure 5, where most attention focuses on the end of the sentence. Unlike the machine translation task, the attention shows the word-level alignment between source and target languages, solving math word problems requires high-level understanding due to the task complexity.

To further analyze the effectiveness of the proposed gating mechanisms for stack action and operand selection, the activation of gates g^{sa}, g^{opd} at each step of the decoding process is shown in the bottom of Figure 5. It shows that most of time, the gate activation is high, demonstrating that the proposed gating mechanisms play an important role during decoding. We also observe a common phenomenon that the activation g_2^{sa}, which controls how much attention the stack action selector puts on the stack state when deciding an stack action, is usually low until the last "operator application" stack action. For example, in the example of Figure 5, g_2^{sa} is less than 0.20 till the last argument selection stack action, and activates when deciding the *division operator application* ($\div$) and the *equal application* ($=$). It may result from the higher-level semantics of the operand $(6.75 - 2.75)$ on the stack when selecting the stack action *division operator application* ($\div$). In terms

Problem & Results
红花有60朵，黄花比红花多1/6朵，黄花有多少朵． (There are 60 red flowers. Yellow flowers are more than red ones by 1/6. How many yellow flowers are there?) *Generated Equation*: $60 + \frac{1}{6}$ *Correct Answer*: 70
火车 48 小时行驶 5920 千米，汽车 25 小时行驶 2250 千米，汽车平均每小时比火车每小时慢多少 千米？ (The train travels 5920 kilometers in hours, and the car travels 2250 kilometers in 25 hours. How many kilometers per hour is the car slower than the train?) *Generated Equation*: $2250 \div 25 - 5920 \div 48$ *Correct Answer*: $33\frac{1}{3}$
小红前面 5 人，后面 7 人，一共有多少人？ (There are 5 people in front of Little Red and 7 people behind. How many persons are there in total?) *Generated Equation*: $5 + 7$ *Correct Answer*: 13

Table 3: Randomly sampled incorrect predictions.

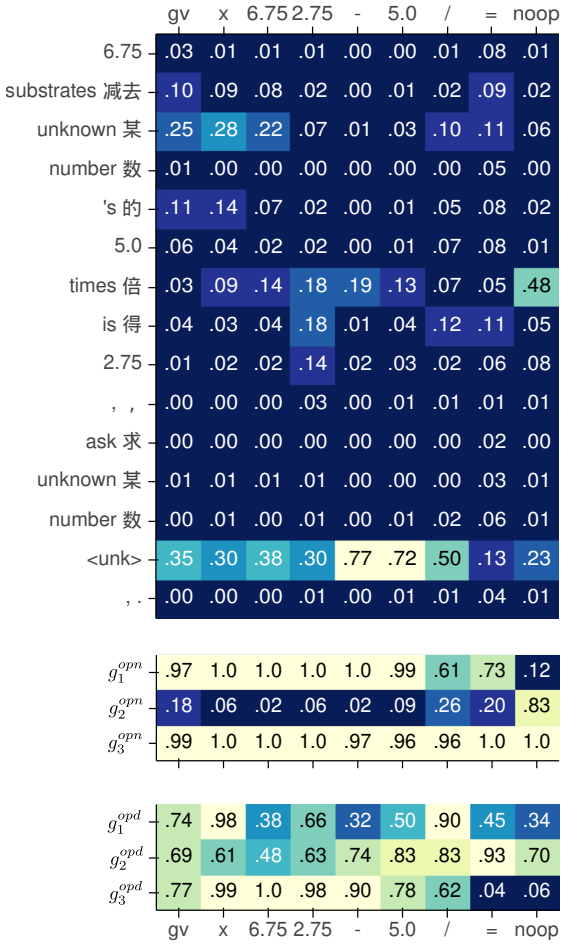

Figure 5: Word attention and gate activation (g^{sa} and g^{opd}) visualization when generating stack actions for the problem "*6.75 deducting 5 times of an unknown number is 2.75. What is the unknown number?*", where the associated equation is $x = (6.75 - 2.75) \div 5$. Note that g^{opd} is meaningful only when the t-th stack action is push_op.

of the activation of g^{opd}, we find that three features are important in most cases, demonstrating the effectiveness of the proposed mechanisms.

5.3 Error Analysis

We randomly sample some results predicted incorrectly by our model shown in Table 3. In the first example, the error is due to the language ambiguity, and such ambiguity cannot be resolved without considering the exact value of the number. From the second example, although our model identifies the problem as a comparison problem successfully, it handles the order of the operands incorrectly. For the third problem, it cannot be solved by using only the surface meaning but requires some common sense. Therefore, above phenomena show the difficulty of solving math word problems and the large room for improvement.

6 Conclusion

We propose an end-to-end neural math solver using an encoder-decoder framework that incorporates semantic representations of numbers in order to generate mathematical symbols for solving math word problems. The experiments show that the proposed model achieves the state-of-the-art performance on the benchmark dataset, and empirically demonstrate the effectiveness of each component in the model. In sum, the proposed neural math solver is designed based on how human performs reasoning when writing equations, providing better interpretation without the need of labeled rationals.

References

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 523–533.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *TACL*, 3:585–597.

Nate Kushman, Luke Zettlemoyer, Regina Barzilay, and Yoav Artzi. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014*, pages 271–281.

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, ACL 2017*, pages 158–167.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Sourav Mandal and Sudip Kumar Naskar. 2019. Solving arithmetic mathematical word problems: A review and recent advancements. In *Information Technology and Applied Mathematics*, pages 95–114. Springer.

Purvanshi Mehta, Pruthwik Mishra, Vinayak Athavale, Manish Shrivastava, and Dipti Misra Sharma. 2017. Deep neural network based system for solving arithmetic word problems. In *Proceedings of the IJCNLP 2017*, pages 65–68.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Benjamin Robaidek, Rik Koncel-Kedziorski, and Hannaneh Hajishirzi. 2018. Data-driven methods for solving algebra word problems. *CoRR*, abs/1804.10718.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1743–1752.

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

Subhro Roy, Shyam Upadhyay, and Dan Roth. 2016. Equation parsing : Mapping sentences to grounded equations. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1088–1097.

Subhro Roy, Tim Vieira, and Dan Roth. 2015. Reasoning about quantities in natural language. *TACL*, 3:1–13.

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, EMNLP 2015*, pages 1132–1142.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, pages 3104–3112.

Shyam Upadhyay and Ming-Wei Chang. 2017. Annotating derivations: A new evaluation strategy and dataset for algebra word problems. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 494–504.

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*, pages 297–306.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018. MathDQN: Solving arithmetic word problems via deep

reinforcement learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

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*, pages 845–854.

A Algorithm Detail

The training and inference procedures are shown in Algortihm 1.

B Hyperparameter Setup

The model is trained with the optimizer `adam` (Kingma and Ba, 2014), and the learning rate is set to 0.001. Pretrained embeddings using FastText (Joulin et al., 2016) are adopted. The hidden state size of LSTM used in the encoder and decoder is 256. The dimension of hidden layers in attention, semantic transformer and operand/stack action selector is 256. The dropout rate is set as 0.1 before inputting the decoder LSTM, before the stack action selector and after the hidden layer of the stack action selector and attention. The reported accuracy is the result of 5-fold cross-validation, same as Wang et al. for fair comparison.

C Error Analysis between Seq2Seq

We implement the seq2seq model as proposed by Wang et al. and compare the performance difference between our proposed model and the baseline seq2seq model. Table 4 shows the generated results seq2seq predicts correctly but our model predicts incorrectly. Table 5 show the results our model can predict correctly but seq2seq cannot.

Problem & Results
小红前面 5 人，后面 7 人，一共有多少人？ (There are 5 people in front of Little Red and 7 people behind. How many persons are there in total?) *Proposed Model*: $5 + 7$ *Seq2Seq Model*: $5 + 7 + 1$
两个数相差28，如果被减数减少3，减数增加5，那么它们的差=？ (The difference between two numbers is 28. If the minuend is reduced by 3, and the subtrahend is increased by 5, then their difference=?) *Proposed Model*: $(28 - 3) \div 5$ *Seq2Seq Model*: $28 - (3 + 5)$
机床厂第一车间有55人，第二车间有45人，每人每天平均生产261个零件，这两个 车间每天共生产多少个零件？ (There are 55 people in the first workshop of the machine tool factory and 45 people in the second workshop. Each person produces 261 small components per day in average. How many components do the two workshops produce every day in total?) *Proposed Model*: $(55 + 45) \div 261$ *Seq2Seq Model*: $(55 + 45) \times 261$
箭鱼游动时的速度是28米/秒，8秒可以游多少米？ (The swordfish swims at speed 28 meters/sec. How many meters can it swim in 8 seconds?) *Proposed Model*: $28 \div 8$ *Seq2Seq Model*: 28×8
水果店有梨子387千克，卖出205千克后，又运来一批，现在水果店共有梨子945千克．水果店又运来梨子多少千克？ (The fruit shop has 387 kilograms of pears . After selling 205 kilograms, some pears arrive. Now the fruit shop has 945 kilograms of pears in total. How many kilograms of pears does the fruit shop get?) *Proposed Model*: $945 \times (387 - 205)$ *Seq2Seq Model*: $945 - (387 - 205)$
王老师买排球用了40元，买篮球用的钱数是排球的3倍．王老师买球一共用了多少元？ (Teacher Wang spent 40 dollars buying volleyballs and 3 times of money for basketballs. How many dollars did Teacher Wang spend for the balls?) *Proposed Model*: $40 \div 3 + 40$ *Seq2Seq Model*: $40 + 40 \times 3$
筑路队修筑一条长1200米的公路，甲队单独修40天可以完成任务，乙队单独修30天可以完成任务．甲队每天修的比乙队少多少米？ (The road construction team built a road with a length of 1200 meters. Team A can complete the task in 40 days alone, and team B can complete the task in 30 days alone. How many meters does team A construct more than team B every day?) *Proposed Model*: $1200 \div 40 - 1200 \div 30$ *Seq2Seq Model*: $1200 \div 30 - 1200 \div 40$
一共1800本，我们六年级分得2/9，分给五年级的本数相当于六年级的4/5，五年级分得多少本？ (There are 1800 books in total. We sixth grade get 2/9. The number of books given to the fifth grade is equal to 4/5 of the number to the sixth grade. How many books does the fifth grade get?) *Proposed Model*: $1800 \times \frac{2}{9} \div \frac{4}{5}$ *Seq2Seq Model*: $1800 \times \frac{2}{9} \times \frac{4}{5}$
有一批布料，如果只做上衣可以做10件，如果只做裤子可以做15条，那么这批布料可以做几套这样的衣服？ (There is a batch of fabrics. If all is used for making shirts, 10 pieces can be made, and 15 pieces if used to make pants only. Then how many suits of such clothes can be made with this batch of fabric?) *Proposed Model*: $10 \times 1 \div 15$ *Seq2Seq Model*: $1 \div (1 \div 10 + 1 \div 15)$
贝贝的钱买一本5.9元笔记本差0.6元，他买一本4.8元的，剩下的钱正好买一只圆珠笔，这只圆珠笔多少钱？ (Beibei needs 0.6 dollars more to buy a notebook of 5.9 dollars. If he buys one of 4.8 dollars, the remaining money allows her to buy exactly one ball pen. How much is the ball pen?) *Proposed Model*: $5.9 + 0.6 - 4.8$ *Seq2Seq Model*: $5.9 - 0.6 - 4.8$

Table 4: Examples that Seq2Seq predicts correctly while our proposed model predicts incorrectly.

Problem & Results
医院里经常要给病人输入葡萄糖水，这种葡萄糖水是把葡萄糖和水按1：19配制的，根据这些信息，你能知道什么？ (In hospital, it is often necessary to give glucose injection to patient. This glucose water is prepared by mixing glucose and water at 1:19. Based on this information, what do you know?) *Proposed Model*: $1 \div (1 + 19.0)$ *Seq2Seq Model*: $1 \times (1 + 19.0)$
一根长2.45米的木桩打入河底，现在测得木桩水上部分长0.75米，水中长1.05米，求这根桩打在泥中的长度=多少米？ (A wooden pile of 2.45 meters long is hammered into the bottom of a river. Now the part above water is measured as 0.75 meters long, and the part in the water is measured as 1.05 meters long. How long is the part of the pile in the mud?) *Proposed Model*: $2.45 - 0.75 - 1.05$ *Seq2Seq Model*: $2.45 + 0.75 + 1.05$
李强6月份的生活费为255元，比计划节省了15%，节省了多少元． (Li Qiang's living expenses in June were 255 dollars, 15% savings over the plan. How much did he save?) *Proposed Model*: $(255.0 \div (1 - 0.15)) \times 0.15$ *Seq2Seq Model*: $0.15 = 6.0/(1 - 255.0) - 6.0$
小芳在计算一个数除以10时，将除号看成了乘号，结果得3.2，正确的结果应该=． (When Xiaofang calculates a number divided by 10 , the division sign is mistakenly treated as a multiplication sign, and the result is 3.2. The correct result should be = .) *Proposed Model*: $3 \div 10 \div 10$ *Seq2Seq Model*: $3.2 \div (1 + 10)$
24 + 91 的 2/13，所得的和再除 19/20，商 =？ (2/13 of 91 + 24, and the sum is divided by 19/20, quotient = ?) *Proposed Model*: $\frac{19}{20} \div (24 + 91 \times \frac{2}{13})$ *Seq2Seq Model*: $\frac{19}{20} \div (24 \times 91 - \frac{2}{13})$
1/3 + 0.25 =？ (1/3 + 0.25 = ?) *Proposed Model*: $\frac{1}{3} + 0.25$ *Seq2Seq Model*: $\frac{1}{3} \times 0.25$
商店运来鸡蛋和鸭蛋各7箱．鸡蛋每箱重26千克，鸭蛋每箱重31千克，商店一共运来的鸡蛋和鸭蛋共多少千克？ (The store shipped 7 boxes of eggs and duck eggs respectively. Eggs weigh 26 kilograms per box, duck eggs weigh 31 kilograms per box. How many kilograms of eggs and duck eggs are shipped from the store in total?) *Proposed Model*: $26 \times 7 + 31 \times 7$ *Seq2Seq Model*: $26 \times 7 + 31$
3.8 - 2.54 + 1.46 =？ (3.8 - 2.54 + 1.46 =) *Proposed Model*: $3.8 - 2.54 + 1.46$ *Seq2Seq Model*: $3.8 + 2.54 + 1.46$
有一池水，第一天放出200吨，第二天比第一天多放20%，第3天放了整池水的36%，正好全部放完．这池水共有多少吨？ (There was a pool of water, which released 200 tons of water in the first day, 20% more in the second day than the first day, and 36% of the whole pool on the third day. Then the water is gone. How many tons of water did this pool have?) *Proposed Model*: $(200.0 + 200.0 \times (1 + 0.2)) \div (1 - 0.36)$ *Seq2Seq Model*: $(200.0 + 0.2) \times 3.0 + 0.2 \times (1 - 0.36)$
16 的 5/12 比一个数的 7 倍多 2， 这个数 =？ (5/12 of 16 is more than 7 times of a number by 2. What is the number=?) *Proposed Model*: $(16 \times \frac{5}{12} - 2) \div 7$ *Seq2Seq Model*: $(16 \times \frac{5}{12} + 7) \div 2$

Table 5: Examples that Seq2Seq predicts incorrectly while our proposed model predicts correctly.