Deep Neural Solver for Math Word Problems

Yan Wang **Xiaojiang Liu** **Shuming Shi**
Tencent AI Lab
{brandenwang, kieranliu, shumingshi}@tencent.com

Abstract

This paper presents a deep neural solver to automatically solve math word problems. In contrast to previous statistical learning approaches, we directly translate math word problems to equation templates using a recurrent neural network (RNN) model, without sophisticated feature engineering. We further design a hybrid model that combines the RNN model and a similarity-based retrieval model to achieve additional performance improvement. Experiments conducted on a large dataset show that the RNN model and the hybrid model significantly outperform state-of-the-art statistical learning methods for math word problem solving.

1 Introduction

Developing computer models to automatically solve math word problems has been an interest of NLP researchers since 1963 Feigenbaum et al. (1963); Bobrow (1964); Briars and Larkin (1984); Fletcher (1985). Recently, machine learning techniques Kushman et al. (2014); Amnueypornsakul and Bhat (2014); Zhou et al. (2015); Mitra and Baral (2016) and semantic parsing methods Shi et al. (2015); Koncel-Kedziorski et al. (2015) are proposed to tackle this problem and promising results are reported on some datasets. Although progress has been made in this task, performance of state-of-the-art techniques is still quite low on large datasets having diverse problem types Huang et al. (2016).

A typical math word problems are shown in Table 1. The reader is asked to infer how many pens Dan and Jessica have, based on constraints provided. Given the success of deep neural networks (DNN) on many NLP tasks (like POS tagging, syntactic parsing, and machine translation), it may be interesting to study whether DNN could also help math word problem solving. In this paper, we propose a recurrent neural network (RNN) model for automatic math word problem solving. It is a sequence to sequence (seq2seq) model that transforms natural language sentences in math word problems to mathematical equations. Experiments conducted on a large dataset show that the RNN model significantly outperforms state-of-the-art statistical learning approaches.

Problem: Dan has 2 pens, Jessica has 4 pens. How many pens do they have in total ?
Equation: x = 4+2
Solution: 6

Table 1: A math word problem

Since it has been demonstrated Huang et al. (2016) that a simple similarity based method performs as well as more sophisticated statistical learning approaches on large datasets, we implement a similarity-based retrieval model and compare with our seq2seq model. We observe that although seq2seq performs better on average, the retrieval model is able to correctly solve many problems for which RNN generates wrong results. We also find that the accuracy of the retrieval model positively correlate with the maximal similarity score between the target problem and the problems in training data: the larger the similarity score, the higher the average accuracy is.

Inspired by these observations, we design a hybrid model which combines the seq2seq model and the retrieval model. In the hybrid model, the retrieval model is chosen if the maximal similarity score returned by the retrieval model is larger than a threshold, otherwise the seq2seq model is selected to solve the problem. Experiments on our

dataset show that, by introducing the hybrid model, the accuracy increases from 58.1% to 64.7%.

Our contributions are as follows:

1) To the best of our knowledge, this is the first work of using DNN technology for automatic math word problem solving.

2) We propose a hybrid model where a seq2seq model and a similarity-based retrieval model are combined to achieve further performance improvement.

3) A large dataset is constructed for facilitating the study of automatic math problem solving.[1]

The remaining part of this paper is organized as follows: After analyzing related work in Section 2, we formalize the problem and introduce our dataset in Section 3. We present our RNN-based seq2seq model in Section 4, and the hybrid model in Section 5. Then experimental results are shown and analyzed in Section 6. Finally we conclude the paper in Section 7.

2 Related work

2.1 Math Word Problems Solving

Previous work on automatic math word problem solving falls into two categories: symbolic approaches and statistical learning approaches.

In 1964, STUDENT Bobrow (1964) handles algebraic problems by two steps: first, they transform natural language sentences into kernel sentences using a small set of transformation patterns. Then the kernel sentences are transformed to mathematical expressions by pattern matching. A similar approach is also used to solve English rate problems Charniak (1968, 1969). Liguda and Pfeiffer Liguda and Pfeiffer (2012) propose modeling math word problems with augmented semantic networks. In addition, Addition/subtraction problems are studied most Briars and Larkin (1984); Dellarosa (1986); Bakman (2007); Yuhui et al. (2010); Roy et al. (2015).

In 2015, Shi et.al Shi et al. (2015) propose a system SigmaDolphin which automatically solves math word problems by semantic parsing and reasoning. In the same year, Koncel et.al Koncel-Kedziorski et al. (2015) also formalizes the problem of solving multi-sentence algebraic word problems as that of generating and scoring equation trees.

[1]We plan to make the dataset publicly available when the paper is published

Since 2014, statistical learning based approaches are proposed to solve the math word problems. Hosseini et al. Hosseini et al. (2014) deal with the open-domain aspect of algebraic word problems by learning verb categorization from training data. Kushman et al. Kushman et al. (2014) proposed a equation template system to solve a wide range of algebra word problems. Zhou et al. Zhou et al. (2015) further extends this method by adopting the max-margin objective, which results in higher accuracy and lower time cost. In addition, Roy and Roth Roy et al. (2015); Roy and Roth (2016) tries to handle arithmetic problems with multiple steps and operations without depending on additional annotations or predefined templates. Mitra et al. Mitra and Baral (2016) presents a novel method to learn to use formulas to solve simple addition-subtraction arithmetic problems.

As reported in 2016 Huang et al. (2016), state-of-the-art approaches have extremely low performance on a big and highly diverse data set (18,000+ problems). In contrast to these approaches, we study the feasibility of applying deep learning to the task of math word problem solving.

2.2 Sequence to Sequence (seq2seq) Learning

With the framework of seq2seq learning Sutskever et al. (2014); Wiseman and Rush (2016), recent advances in neural machine translation (NMT) Bahdanau et al. (2014); Cho et al. (2014) and neural responding machine (NRM) Shang et al. (2015) have demonstrated the power of recurrent neural networks (RNNs) at capturing and translating natural language semantics. The NMT and NRM models are purely data-driven and directly learn to converse from end-to-end conversational corpora.

Recently, the task of translating natural language queries into regular expressions is explored by using a seq2seq model Locascio et al. (2016), which achieves a performance gain of 19.6% over previous state-of-the-art models. To our knowledge, we are the first to apply seq2seq model to the task of math word problem solving.

3 Problem Formulation and Dataset

3.1 Problem Formulation

A math word problem P is a word sequence W_p and contains a set of variables $V_p = \{v_1, \ldots, v_m, x_1, \ldots, x_k\}$ where $v_1, \ldots, v_m$ are known numbers in P and $x_1, \ldots, x_k$ are variables

Problem: Dan has 5 pens and 3 pencils, Jessica has 4 more pens and 2 less pencils than him. How many pens and pencils does Jessica have in total?
Equation: x = 5 + 4 +3 -2
Solution: 10

Table 2: A math word problem

whose values are unknown. A problem P can be solved by a mathematical equation E_p formed by V_p and mathematical operators.

In math word problems, different equations may belong to a same equation template. For example, equation $x = (9 * 3) + 7$ and equation $x = (4 * 5) + 2$ share the same equation template $x = (n_1 * n_2) + n_3$. To decrease the diversity of equations, we map each equation to an equation template T_p through a number mapping M_p. The number mapping process can be defined as:

Definition 1 *Number mapping: For a problem P with m known numbers, a number mapping M_p maps the numbers in problem P to a list of number tokens $\{n_1, \ldots, n_m\}$ by their order in the problem text.*

Definition 2 *Equation template: A general form of equations. For a problem P with equation E_p and number mapping M_p, its equation template is obtained by mapping numbers in E_p to a list of number tokens $\{n_1, \ldots, n_m\}$ according to M_p.*

Take the problem in Table 2 as an example, first we can obtain a number mapping from the problem:

$$M : \{n_1 = 5; \quad n_2 = 3; \quad n_3 = 4; \quad n_4 = 2; \}$$

and then the given equation can be expressed as an equation template:

$$x = n_1 + n_3 + n_2 - n_4$$

After number mapping, the problem in Table 2 can be mapped to:

"Dan have n_1 pens and n_2 pencils, Jessica have n_3 more pens and n_4 less pencils than him. How many pens and pencils do Jessica have in total?"

We solve math word problems by generating equation templates through a seq2seq model. The input of the seq2seq model is the sequence W_P after number mapping, and the output is an equation template T_P. The equation E_P can be obtained by applying the corresponding number mapping M_P to T_P.

3.2 Constructing a Large Dataset

Most public datasets for automatic math word problem solving are quite small and contains limited types of problems. The most frequently used Alg514 (Kushman et al., 2014) dataset contains only 514 linear algebra problems with 28 equation templates. There are 1,000 problems in the newly constructed DRAW-1K (Shyam and Ming-Wei, 2017) dataset. Dophin1878 (Shi et al., 2015) includes 1,878 number word problems. An exception is the Dolphin18K dataset (Huang et al., 2016) which contains 18,000+ problems. However, this dataset has not been made publicly available so far.

Since DNN-based approaches typically need large training data, we have to build a large dataset of labeled math word problems. We crawl over 60,000 Chinese math word problems from a couple of online education web sites. All of them are real math word problems for elementary school students. We focus on one-unknown-variable linear math word problems in this paper. For other problem types, we would like to leave as future work. Please pay attention that the solutions to the problems are in natural language, and we have to extract equation systems and structured answers from the solution text. We implement a rule-based extraction method for this purpose, which achieves very high precision and medium recall. That is, most equations and structured answers extracted by our method are correct, and many problems are dropped from the dataset. As a result, we get dataset Math23k which contains 23,161 problems labeled with structured equations and answers. Please refer to Table 3 for some statistics of the dataset and a comparison with other public datasets.

4 Deep Neural Solver

In this section, we propose a *RNN-based seq2seq model* to translate problem text to math equations. Since not all numbers in problem text may be useful for solving the problem, we propose, in Section 4.2, a *significant number identification* model to distinguish whether a number in a problem should appear in the corresponding equations.

4.1 RNN based Seq2seq Model

Figure 1 shows our RNN-based seq2seq model for transforming problem text to a math equation, using the problem in Table 2 as an example. The in-

dataset	# problems	# templates	# sentences	# words	problem types
Alg514	514	28	1.62k	19.3k	algebra, linear
Dolphin1878	1,878	1,183	3.30k	41.4k	number word problems
DRAW-1K	1,000	Unknown	6.23k	81.5k	algebra, linear, one-VAR
Math23K	23,161	2,187	70.1k	822k	algebra, linear, one-VAR

Table 3: Statistics of our dataset and several publicly available datasets

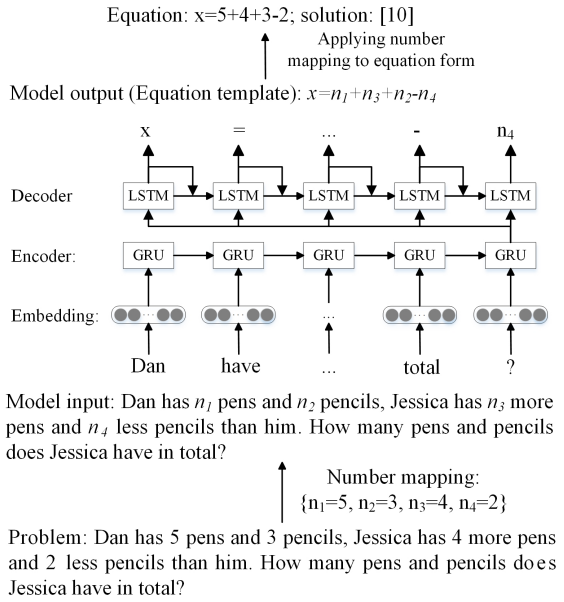

Figure 1: The seq2seq model

put sequence W is the problem after number mapping:

"Dan has n_1 pens and n_2 pencils, Jessica has n_3 more pens and n_4 less pencils than him. How many pens and pencils does Jessica have in total?"

The output sequence $R = \{r_1, \ldots, r_s\}$ is the equation template:

$$x = n_1 + n_3 + n_2 - n_4$$

The gated recurrent units (GRU) (Chung et al., 2014) and long short-memory (LSTM) (Hochreiter and Schmidhuber, 1997) cells are used for encoding and decoding, respectively. The reason why we use GRU as the encoder instead of LSTM is that the GRU has less parameters and less likely to be overfitted on small dataset. Four fundamental operational stages of GRU are as follows:

$$\begin{aligned}
z_t &= \sigma(W^{(z)}x_t + U^z h_{t-1}) && \text{(Update gate)} \\
r_t &= \sigma(W^{(r)}x_t + U^r h_{t-1}) && \text{(Reset gate)} \\
\hat{h_t} &= tanh(r_t \odot U h_{t-1} + W x_t) && \text{(New memory)} \\
h_t &= (1 - z_t) \odot \hat{h_t} + z_t \odot h_{t-1} && \text{(Hidden state)}
\end{aligned} \tag{1}$$

where σ represents the sigmoid function and $\odot$ is an element-wise multiplication. The input x_t is a word w_t along with previously generated character r_{t-1} . The variables U and W are weight matrices for each gate.

The fundamental operational stages of LSTM are as follows:

$$\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^i h_{t-1}) && \text{(Input gate)} \\
f_t &= \sigma(W^{(f)}x_t + U^f h_{t-1}) && \text{(Forget gate)} \\
o_t &= \sigma(W^{(o)}x_t + U^o h_{t-1}) && \text{(Output gate)} \\
\tilde{c_t} &= tanh(W^{(c)}x_t + U^{(c)} h_{t-1}) && \text{(New memory)} \\
c_t &= f_t \odot \tilde{c}_{t-1} + i_t \odot \tilde{c_t} && \text{(Final memory)} \\
h_t &= o_t \odot tanh(c_t) && \text{(Hidden state)}
\end{aligned} \tag{2}$$

where the input x_t is a word w_t along with previously generated character r_{t-1} .

Then, we redesigned the activation function of the seq2seq model, which is different from vanilla seq2seq models. If we directly generate equation templates by a softmax function, some incorrect equations may be generated, such as: "$x = n_1 + + * n_2$" and "$x = (n_1 * n_2$". To ensure that the output equations are mathematically correct, we need to find out which characters are illegal according to previously generated characters. This is done by five predefined rules like:

- Rule 1: If r_{t-1} in $\{+, -, *, /\}$, then r_t will not in $\{+, -, *, /,), =\}$;
- Rule 2: If r_{t-1} is a number, then r_t will not be a number and not in $\{(, =\}$;

- Rule 3: If r_{t-1} is "=", then r_t will not in $\{+, -, *, /, =,)\}$;
- Rule 4: If r_{t-1} is "(", then r_t will not in $\{(,), +, -, *, /, =\}$;
- Rule 5: If r_{t-1} is ")", then r_t will not be a number and not in $\{(,)\}$;

A binary vector ρ_t can be generated depends on r_{t-1} and these rules. Each position in ρ_t is corresponding to a character in the output vocabulary, where "1" represents that the character is mathematically correct, and "0" indicates mathematically incorrect. Thus, the output probability distribution at each time-step t can be calculated as:

$$P(\hat{r_t}|h_t) = \frac{\rho_t \odot e^{h_t^T W^s}}{\sum \rho_t \odot e^{h_t^T W^s}} \quad (3)$$

where h_t is the output of LSTM decoder, and W^s is the weight matrix. The probability of mathematically incorrect characters will be 0.

Our model is five layers deep, with a word embedding layer, a two-layer GRU as encoder and a two-layer LSTM as decoder. Both the encoder and decoder contain 512 nodes. We perform standard dropout during training (Srivastava et al., 2014) after GRU and LSTM layer with dropout probability equal to 0.5. We train for 80 epochs, utilizing a mini-batch size of 256 and a learning-rate of 0.01.

4.2 Significant Number Identification (SNI)

In a math word problem, not all numbers appear in the equation for solving the problem. An example is shown in Table 4, where the number *"1"* in *"1 day, 1 girl"* and number *"2"* in *"She has 2 types of"* should not be used in equation construction. We say a number is *significant* if the number should be included in the equation to the problem; otherwise it is *insignificant*. For the problem in Table 4, significant numbers are 9, 3, and 5, while 1 and 2 are insignificant numbers. Identifying significant and insignificant numbers are important for constructing correct equations. For this purpose, we build a LSTM-based binary classification model to determine whether a number in a piece of problem text is significant.

The training data for SNI model are extracted from the math word problems. Each number and its context in problems is a training instance of SNI. An instance will be labelled"True" if the number is *significant*, otherwise it will be labelled "False". The structure of SNI model is shown in Figure 2. By using single layer LSTMs with 128 nodes and a symmetric window of length 3, our model achieves 99.1% accuracy. Table 4 is an example of number mapping with and without SNI.

Problem: 1 day, 1 girl was organizing her book case making sure each of the shelves had exactly 9 books on it. She has 2 types of books - mystery books and picture books. If she has 3 shelves of mystery books and 5 shelves of picture books, how many books did she have in total?
Number mapping: $n_1 = 1; n_2 = 1; n_3 = 9; n_4 = 2; n_5 = 3; n_6 = 5$
Equation template: $x = n_5 * n_3 + n_6 * n_3$
Number mapping with SNI: $n_1 = 9; n_2 = 3; n_3 = 5$
Equation template with SNI: $x = n_2 * n_1 + n_3 * n_1$
Problem after number mapping and SNI: 1 day, 1 girl was organizing her book case making sure each of the shelves had exactly n_1 books on it. She has 2 types of books -mystery books and picture books. If she has n_2 shelves of mystery books and n_3 shelves of picture books, how many books did she have in total?

Table 4: Significant number identification (SNI) example

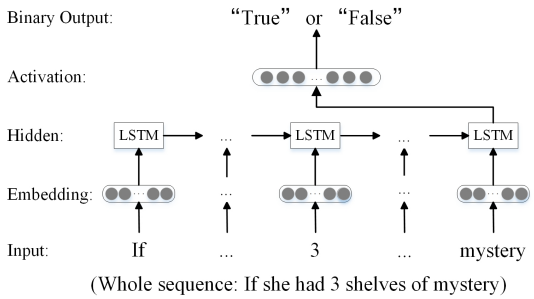

Figure 2: The significant number identification model

5 Hybrid Model

To compare the performance of our deep neural solver and traditional statistical learning methods, we implement a similarity-based retrieval model (refer to Section 5.1 for more details).

The Venn diagram in Figure 3 shows the relationship between the problems solved by the re-

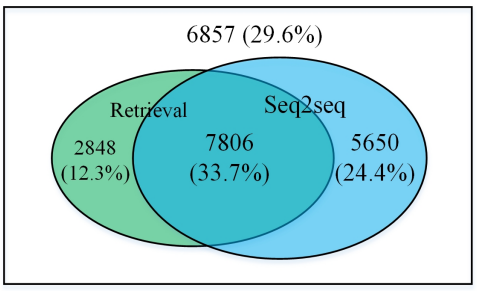

Figure 3: **Green area**: problems correctly solved by the retrieval model; **Blue area**: problems correctly solved by the seq2seq model; **Overlapped area**: problems correctly solved by both models; **White area**: problems that both models fail to solve

trieval model and those solved by the seq2seq model. We can see that although seq2seq performs better on average, the retrieval model is able to correctly solve many problems that seq2seq cannot solve. If we can combine the two models properly to build a hybrid model, more problems may get solved.

In this section, we first give some details about the retrieval model in Section 5.1, then the hybrid model is introduced in Section 5.2.

5.1 Retrieval Model

The retrieval model solves problems by calculating the lexical similarity between the testing problem and each problem in the training data, and then the equation template of the most similar problem is applied to the testing problem. Each problem is modeled as a vector of word TF-IDF scores $W = [w_{1,d}, w_{2,d}, \ldots, w_{N,d}]^T$, where

$$w_{t,d} = tf_{t,d} * \frac{|D|}{|d \in D | t \in d|} \quad (4)$$

and $tf_{t,d}$ is the word frequency of word t in problem d; $|D|$ is the total number of problems in dataset; $|d \in D | t \in d|$ is the number of documents containing the word t.

The similarity between the testing problem P_T and another problem Q can be calculated by the Jaccard similarity between their corresponding vectors:

$$J(P_T, Q) = \frac{|P_T \cap Q|}{|P_T \cup Q|} = \frac{|P_T \cap Q|}{|P_T| + |Q| - |P_T \cap Q|} \quad (5)$$

The retrieval model will choose training problem Q_1 that have the maximal similarity with P_T and use the equation template T of Q_1 as the template of problem P_T.

An important and interesting observation about the retrieval model is the relation between the maximal similarity and solution accuracy. Figure 4 shows the results of only considering the problems for those the maximal similarity returned by retrieval model is above a threshold θ (in other words, we skip a problem if its corresponding maximal similarity is below the threshold). It is clear that the larger the similarity score, the higher the average accuracy is. In our hybrid model, we make use of this property to combine the seq2seq model and the retrieval model.

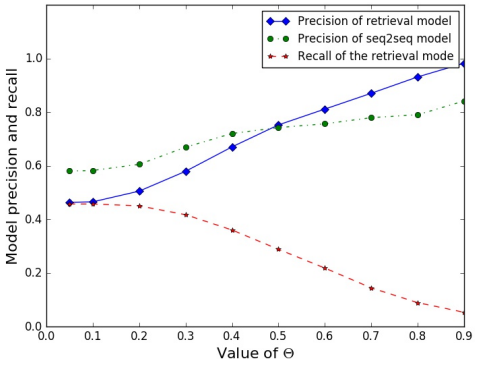

Figure 4: Precision and recall of the retrieval model, and the precision of the seq2seq model w.r.t. different similarity threshold (θ) values

5.2 Hybrid Model

Our hybrid model combines the retrieval model and the seq2seq model by setting a hyperparameter θ as the threshold of similarity. In algorithm 1, if the Jaccard similarity between testing problem P_T and the retrieved problem Q_1 is higher than θ, the model will choose the equation template T of Q_1 as the equation template of problem P_T. Otherwise an equation template will be generated by a seq2seq model. As shown in Figure 4, the retrieval model has a higher precision than the seq2seq model when we set a high threshold.

6 Experiments

In this section, we conduct experiments on two datasets to examine the performance of the proposed models. Our main experimental result is to show a significant improvement over the baseline

Algorithm 1 Hybrid model

Input: Q: problems in training data;
P_T: testing problem;
θ: pre-defined threshold of similarity

Output: Problem solution

1: Get equation templates and number mappings for training problems Q and testing problem P_T.
2: Number identification: identify significant numbers
3: Retrieval:
choose problem Q_1 from Q that has the maximal Jaccard similarity with P_T
4: **if** $J(P_T, Q_1) > \theta$ **then**
5: Apply the retrieval model: select equation template T of Q_1
6: **else**
7: Apply the seq2seq model: $T = seq2seq(P_T)$
8: **end if**
9: Applying number mappings of P_T to T and calculating final solution

	Math23K	Alg514
ZDC	42.1%	**79.7%**
Retrieval model w/o SNI	46.0%	70.1%
Retrieval model w/ SNI	47.2%	70.1%
Seq2seq model w/o SNI	53.7%	17.2%
Seq2seq model w/ SNI	58.1%	16.1%
Hybrid model w/o SNI	61.1%	70.1%
Hybrid model w/ SNI	**64.7%**	70.1%

Table 5: Model comparison (average accuracy of 5-fold cross validation)

	ZDC	R	R(S)	Seq	Seq(S)	H
R(S)	$\gg$	$>$				
Seq	$\gg$	$\gg$	$\gg$			
Seq(S)	$\gg$	$\gg$	$\gg$	$\gg$		
H	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	
H(S)	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$

Table 6: Result of significance test. The meaning of abbreviations in this table is as follows: R: retrieval model w/o SNI; R(S): retrieval model w/ SNI; Seq: seq2seq model w/o SNI; Seq(S): seq2seq model w/ SNI; H: hybrid model w/o SNI; H(S): hybrid model w/ SNI

method on the proposed Math23K dataset. We further show that the baseline method cannot solve problems with new equation templates. In contrast, the proposed seq2seq model is quite robust on problems with new equation templates (refer to Table 7).

6.1 Experimental Setup

Datasets: As introduced in Section 3.2, we collected a dataset called Math23K which contains 23161 math word problems labeled with equation templates and answers. All these problems are linear algebra questions with only one variable. There are 2187 equation templates in the dataset. In addition, we also evaluate our method on a public dataset Alg514 (Kushman et al., 2014).

Baseline: We compare our proposed methods with two baselines. The first baseline is the retrieval model introduced in Section 5.1. The second one is ZDC (Zhou et al., 2015), which is an improved version of KAZB (Kushman et al., 2014). It maps a problem to one equation template defined in the training set by reasoning across problem sentences. It reports an accuracy of 79.7% on the Alg514 dataset. The Stanford parser is adopted in ZDC to parse all math word problems to Stanford coreNLP output formats. [2]

6.2 Experimental Results

Each approach is evaluated on each dataset via 5-fold cross-validation: In each run, 4 folds are used for training and 1 fold is used for testing. Evaluation results are summarized in Table 5. First, to test the effectiveness of *significant number identification* (SNI), model performance before and after the application of SNI are compared. Then, the performance of the hybrid model, seq2seq model, and retrieval model are examined on two datasets respectively.

To check whether the performance improvements are significant enough, we conduct statistical significance study upon pairs of methods. Table 6 shows the results of sign test, where the symbol $>$ indicates that the method in the row significantly (with p value < 0.05) improves the performance of the method in the column, and the symbol $\gg$ indicates that the performance improvement is extremely significant (with p value < 0.01).

Several observations can be made from the re-

[2] We also try to run KAZB on our dataset, but fail on our workstation (2 12-core E5-2650 CPU, 128G RAM, 4 K80 GPUs) due to large memory consumption.

sults. First, the seq2seq model significantly outperforms state-of-the-art statistical learning methods (ZDC and the retrieval model). Second, by combining the retrieval model and the seq2seq model using a simple mechanism, our hybrid model achieves significant performance gain with respect to the seq2seq model. Third, the SNI module can effectively improve model accuracy. The accuracy of the hybrid model and seq2seq model gains approximately 4% increase after number identification. Please pay attention that on the small dataset of Alg514, the seq2seq model behaves much worse than others. This is not surprising, because deep neural networks typically need large training data.

Figure 5 shows the performance of different models on various scales of training data. As expected, the seq2seq model performs very well on big datasets, but poorly on small datasets.

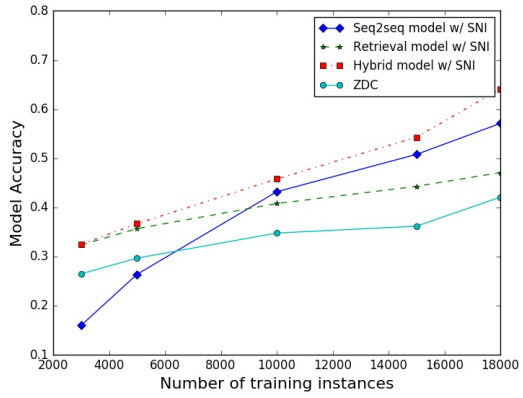

Figure 5: Performance of different models versus the size of training set

Ability to Generate New Equation Templates: please note that many problems in Math23K can be solved using the same equation template. For example, a problem which corresponds to the equation $x = (9 * 3) + 7$ and a different problem that maps to $x = (4 * 5) + 2$ share the same equation template.

One nice property of the seq2seq model is its ability of generating new equation templates. Most previous statistical learning methods (with a few exceptions) for math word problem solving are only able to select an equation template from those in the training data. In other words, they cannot generate new templates. To test the performance of the seq2seq model in generating new templates,

	Math23K
ZDC	15.1%
Retrieval model w/o SNI	26.1%
Retrieval model w/ NI	29.2%
Seq2seq model w/o SNI	40.3%
Seq2seq model w/ SNI	**47.5%**
Hybrid model w/o SNI	40.3%
Hybrid model w/ SNI	**47.7%**

Table 7: Experimental results of non-overlapping templates between training data and test data

we make a new split of our dataset between training data and test data, to ensure that the training data and the test data do not share overlapped templates. As a result, we get a training set with 19, 024 problems and 1, 802 equation templates, and a testing set with 4, 137 problems and 315 equation templates.

Experimental results on the new training set and test set are shown is shown in Table 7. By comparing Table 5 and Table 7, it is clear that the gap between the seq2seq model and the baselines becomes larger in the new settings. It is because the seq2seq model can effectively generate new equation templates for new problems, instead of selecting equation templates from the training set.

Although ZDC and the retrieval model cannot generate new templates, their accuracy is not zero in the new settings. That is because one problem can be solved by multiple equation templates: Although one problem is labeled with template T_1 in the test set, it may also be solved by another template T_2 in the training set.

6.3 Discussion

Compare to most previous statistical learning methods for math problem solving, our proposed seq2seq model and hybrid model have the following advantages: 1) They have higher accuracy on large training data. On the Math23K dataset, the hybrid model achieves at least 22% higher accuracy than the baselines. 2) They have the ability of generating new templates (i.e., templates that are not in the training data. 3) They do not rely on sophisticated feature engineering.

7 Conclusion

We have proposed an RNN-based seq2seq model to automatically solve math word problems. This model directly transforms problem text to a math

equation template. This is the first work of applying deep learning technologies to math word problem solving. In addition, we have designed a hybrid model which combines the seq2seq model and a retrieval model to further improve performance. A large dataset has been constructed for model training and empirical evaluation. Experimental results show that both the seq2seq model and the hybrid model significantly outperform state-of-the-art statistical learning methods in math word problem solving.

The output of our seq2seq model is a single equation containing one unknown variable. Therefore our approach is only applicable to the problems whose solution involves one linear equation of one unknown variable. As future work, we plan to extend our model to be able to generate equation systems and nonlinear equations.

References

Bussaba Amnueypornsakul and Suma Bhat. 2014. Machine-guided solution to mathematical word problems. In *PACLIC*. pages 111–119.

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

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. *arXiv preprint math/0701393* .

Daniel G Bobrow. 1964. Natural language input for a computer problem solving system .

Diane J Briars and Jill H Larkin. 1984. An integrated model of skill in solving elementary word problems. *Cognition and instruction* 1(3):245–296.

Eugene Charniak. 1968. *CALCULUS WORD PROBLEMS*. Ph.D. thesis, Massachusetts Institute of Technology.

Eugene Charniak. 1969. Computer solution of calculus word problems. In *Proceedings of the 1st international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., pages 303–316.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* .

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* .

Denise Dellarosa. 1986. A computer simulation of childrens arithmetic word-problem solving. *Behavior Research Methods, Instruments, & Computers* 18(2):147–154.

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

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

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 *EMNLP*. pages 523–533.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. How well do computers solve math word probl ems? large-scale dataset construction and evaluation. *Proceedings of the 2016 North American Chapter of the ACL (NAACL HLT)* .

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, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. Association for Computational Linguistics.

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, pages 247–252.

Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. 2016. Neural generation of regular expressions from natural language with minimal domain knowledge. *arXiv preprint arXiv:1608.03000* .

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. ACL.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413* .

Subhro Roy, Tim Vieira, and Dan Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics* 3:1–13.

Lifeng Shang, Zhengdong Lu, and Hang Li. 2015. Neural responding machine for short-text conversation. *arXiv preprint arXiv:1503.02364* .

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *EMNLP*. pages 1132–1142.

Upadhyay Shyam and Chang Ming-Wei. 2017. Annotating derivations: A new evaluation strategy and dataset for algebra word problems. In *EACL*. pages 494–504.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

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

Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960* .

Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang Ronghuai. 2010. Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems. In *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*. IEEE, volume 2, pages 476–479.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *EMNLP*. pages 817–822.