Neural Math Word Problem Solver with Reinforcement Learning

Danqing Huang[1]*, **Jing Liu**[2]*, **Chin-Yew Lin**[3], **and Jian Yin**[1]
{huangdq2@mail2,issjyin@mail}.sysu.edu.cn
liujing46@baidu.com
cyl@microsoft.com
[1] The School of Data and Computer Science, Sun Yat-sen University.
Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou, P.R.China
[2]Baidu Inc. [3]Microsoft Research

Abstract

Sequence-to-sequence model has been applied to solve math word problems. The model takes math problem descriptions as input and generates equations as output. The advantage of sequence-to-sequence model requires no feature engineering and can generate equations that do not exist in training data. However, our experimental analysis reveals that this model suffers from two shortcomings: (1) generate spurious numbers; (2) generate numbers at wrong positions. In this paper, we propose incorporating copy and alignment mechanism to the sequence-to-sequence model (namely CASS) to address these shortcomings. To train our model, we apply reinforcement learning to directly optimize the solution accuracy. It overcomes the "train-test discrepancy" issue of maximum likelihood estimation, which uses the surrogate objective of maximizing equation likelihood during training while the evaluation metric is solution accuracy (non-differentiable) at test time. Furthermore, to explore the effectiveness of our neural model, we use our model output as a feature and incorporate it into the feature-based model. Experimental results show that (1) The copy and alignment mechanism is effective to address the two issues; (2) Reinforcement learning leads to better performance than maximum likelihood on this task; (3) Our neural model is complementary to the feature-based model and their combination significantly outperforms the state-of-the-art results.

1 Introduction

The task of math word problem solving aims to automatically solve a math problem by reading the text description of the problem and generating the answer. This task requires the machine to have the ability of natural language understanding and reasoning.

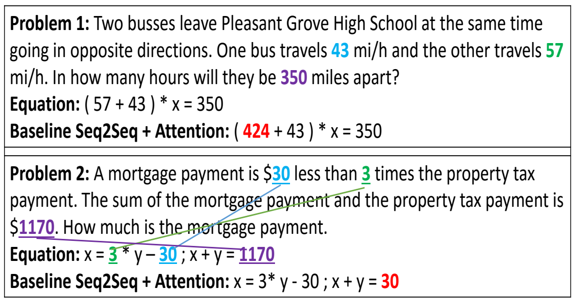
In the past years, most of the proposed methods heavily rely on predefined rules or feature engineering. On one hand, the rule-based approaches (Bakman, 2007; Liguda and Pfeiffer, 2012; Shi et al., 2015) predefine a structured representation and maps the problem description into the structure by rules. These approaches usually accept only well-formed input and are difficult to scale to other problem types. On the other hand, the feature-based statistical learning approaches (Kushman et al., 2014; Roy and Roth, 2018) generate equation candidates and find the most probable equation by using predefined features. These approaches have two major drawbacks: (1) Their ability of equation generation is weak. An equation is generated by either replacing the numbers of existing equations in the training data, or enumerating possible combinations of math operators, numbers and variables, which leads to intractably huge search space. (2) They need manually designed features that are specific to math word problems.

Recent attempts (Ling et al., 2017; Wang et al., 2017) use sequence-to-sequence (seq2seq) model for math word problem solving and they have shown promising results. Wang et al. (2017) apply a standard seq2seq model to generate equations. They have shown that seq2seq models have the power to generate equations of which the problem types do not exist in the training data.

*Work was done at Microsoft Research.

However, we have observed two major shortcomings of the existing seq2seq model with attention mechanism. First, the model may generate spurious numbers, which are detrimental for math problem solving. As shown in Figure 1, the model generates a number "424" in the equation for *Problem 1*, which does not exist in the problem description. Since some of the numbers are unavoidably rare or do not appear in the training data, the existing models have difficulty generalizing to the long tail numbers. Second, the model is prone to generate numbers at wrong positions. In *Problem 2*, although the equation template is correctly generated by the model, the last token in the equation should be aligned to "1170" instead of "30" in the problem description. These two behaviors mentioned above are undesirable and will lead to wrong solutions.

In this paper, we focus on addressing the above two issues in the existing model (Wang et al., 2017) by incorporating Copy and Alignment mechanism to the seq2seq model (namely CASS). (1) Copy means directly copying the numbers in the problem description to the equations. In this way, we can avoid generating spurious numbers that are not in the problem description. (2) Alignment means that there is alignment information between the numbers in the equations and the numbers in the problem description. The model could learn the alignment in a supervised way.


Problem 1: Two busses leave Pleasant Grove High School at the same time going in opposite directions. One bus travels 43 mi/h and the other travels 57 mi/h. In how many hours will they be 350 miles apart?
Equation: (57 + 43) * x = 350
Baseline Seq2Seq + Attention: (424 + 43) * x = 350

Problem 2: A mortgage payment is $30 less than 3 times the property tax payment. The sum of the mortgage payment and the property tax payment is $1170. How much is the mortgage payment.
Equation: x = 3 * y – 30 ; x + y = 1170
Baseline Seq2Seq + Attention: x = 3* y - 30 ; x + y = 30

Figure 1: The baseline model generates a spurious number "*424*" in problem 1 and align numbers wrongly in problem 2.

When training the model, we adopt the reinforcement learning technique, specifically policy gradient. Because maximum likelihood estimation (MLE) suffers from the issue of "train-test discrepancy". It means that MLE uses a surrogate objective of maximizing equation likelihood during training, while the evaluation metric of the task is solution accuracy, which is non-differentiable. Therefore we use policy gradient to directly optimize the solution accuracy, which is more capable for this task.

Furthermore, we observe that the neural model and the traditional feature-based model are complementary. To take the advantage of both approaches, we add the result of our neural model as a simple feature to the feature-based model (Huang et al., 2017) to create a combined model.

We test our model on three publicly available datasets. The experimental results show that the copy and alignment mechanism is effective. Reinforcement learning leads to better performance than MLE. When combining our neural model with the feature-based model, we achieve the state-of-the-art results on all publicly available datasets.

The contributions of this paper are as follows:

1) We incorporate copy and alignment mechanism that augment the standard seq2seq model to address two types of errors: generating spurious numbers and generating numbers in wrong positions.

2) We adopt the reinforcement learning to optimize the solution accuracy, which is more capable for this task and leads to better performance.

3) We propose a simple but effective way to combine the neural model with a traditional feature-based model. The combined model outperforms the state-of-the-art models.

2 Problem Statement and Datasets

Given a math word problem P, the goal is to predict its answer A_p. In the training phase, we have annotations of both equation system E_p and answer A_p for each problem. In the testing phase, we obtain the final answer by generating equation and executing it with a math solver. We evaluate the task using solution accuracy.

Equation template is a unique form of an equation system. For example, given an equation system $\{2 * x + 4 * y = 34, x + 4 = y\}$, we replace the numbers with four number tokens $\{n_1, n_2, n_3, n_4\}$ and generalize the equations as the following equation template $\{n_1 * x + n_2 * y = n_3, x + n_4 = y\}$.

We can see that an equation system includes one or more equations and it is a solution for a specific math word problems. In contrast, an equation template can correspond to several math problems. The number of templates in a dataset reflects the diversity of problem types. Specifically, we have a subset setting $T6$, which represents problems for which the associated template appeared equal to or more than six times in the subset. Note that T6 is a soft constraint of previous feature-based models.

We evaluate different models on three publicly available math word problem datasets[1].

- **Algebra 514** (Alg514) is created by Kushman et al. (2014). It contains 514 algebra word problems from Algebra.com. In the dataset, each template corresponds to at least 6 problems (T6 setting). It only contains 28 templates in total.

- **Number Word Problem** (NumWord) is created by Shi et al. (2015). It contains 2,871 number word problems (i.e., verbally expressed number problems) with 1,183 templates. The T6 subset contains 348 problems. One example problem is *"The sum of two numbers is 10. Their difference is 4. What are the two numbers?"*. We use its linear subset, which contains 986 problems. The vocabulary of this dataset is the smallest among the three, but contains more templates than Alg514.

- **Dolphin18K** (Dophin18K) is created by Huang et al. (2016). It contains 18,711 math word problems from Yahoo! Answers with 5,738 templates. It has much more problem types than the previous two datasets. This dataset is the most challenging of the three. We use its subset with equation annotation, which contains 10,644 problems. The T6 subset contains 6,827 problems.

3 Modeling

Math word problem solving can be formulated as a sequence prediction problem. We set x as the sequence of words in a math word problem description and we want to generate y, the sequence of tokens in an equation system. In this section, we describe (1) the baseline seq2seq model, (2) our two number-related mechanisms to address the issues in existing model.

3.1 Basic Sequence-to-Sequence Model

Following Wang et al. (2017), we first map numbers in the problem description to a list of number tokens $\{n_1,...,n_m\}$ and replace the corresponding numbers in the equation system with the number tokens. As shown in Figure 1, the problem description of *Problem 1* will be as follows after number mapping:
Two buses leave Pleasant Grove High School at the same time going in opposite directions. One bus travels n_1 mi/h and the other travels n_2 mi/h. In how many hours will they be n_3 miles apart?
And the equation system is normalized to: *(n_2 + n_1) * x* = n_3.

The baseline model of seq2seq with attention is based on the work by Bahdanau et al. (2015). It can be viewed as an encoder-decoder model. Basically, the encoder (that is implemented as a single-layer bidirectional GRU) reads the source tokens in problem description one-by-one and produces a sequence of hidden states $h_i = [h_i^F, h_i^B]$ with:

$$h_i^F = GRU(\phi^{in}(x_i), h_{i-1}^F) \tag{1}$$
$$h_i^B = GRU(\phi^{in}(x_i), h_{i+1}^B) \tag{2}$$

where ϕ^{in} maps each token x_i to a fixed-dimensional vector.

At each decoding step j, the decoder receives the embedding of the previous generated token, the current decoder hidden state and the context vector to produce the target vocabulary distribution as described in the following equation:

$$P_{vocab}(w_j) = softmax(U[\phi^{out}(y_{j-1}), c_j, s_j] + b) \tag{3}$$

[1] (Wang et al., 2017) create a dataset containing 23,161 Chinese algebra math problems which has not been public yet.

where s_j is the decoder state and $\phi^{out}(y_{j-1})$ is the previous output embedding. c_j is the context vector and we calculate it as follows:

$$e_{ji} = v^T tanh(W_h h_i + W_s s_j + b_{attn}) \tag{4}$$

$$a_{ji} = \frac{\exp(e_{ji})}{\sum_{i'=1}^{m} \exp(e_{ji'})} \tag{5}$$

$$c_j = \sum_{i=1}^{m} a_{ji} h_i \tag{6}$$

Intuitively, a_{ji} defines the probability distribution over the input tokens. They are computed from the unnormalized attention scores e_{ji}. c_j is the weighted sum of the encoder hidden states. Specifically, W_h, W_s, U, b_{attn}, and b are parameters of the model.

Once we obtain the decoding result, a post-processing step recovers all tokens n_i to their corresponding numbers in the problem description.

3.2 Copy Numbers

In the basic model, the output token y_j is chosen via a softmax over all words in the output vocabulary. However, this model has the problem of generating spurious number tokens. For example, *Problem 1* in Figure 1 only contains three numbers, replaced with tokens $\{n_1, n_2, n_3\}$. Since the output vocabulary contains other tokens, the model may generate a token "n_4", which cannot be recovered and results in generating wrong equations.

To address this problem, we incorporate an attention-based copy mechanism into the basic model similar to Jia and Liang (2016). In our case, we only copy numbers from the source problem.

At each decoding step j, the model has to decide whether to *generate a token* from target vocabulary or *copy a number* from the problem description. The generation probability p_{gen} is modeled by:

$$p_{gen} = \sigma(W_c c_j + W_s' s_j + W_y y_{j-1} + b_{gen}) \tag{7}$$

where W_c, W_s', W_y, and b_{gen} are parameters of the model and σ is the sigmoid function.

Then the output candidates are extended to the concatenation of the target vocabulary and the numbers in the math problem. We can obtain the output probability distribution:

$$P(w_j = w) = p_{gen} * P_{vocab}(w_j) + (1 - p_{gen}) * \frac{\sum_{i:w_i=w} a_{ji}}{\sum_{k:w_n} a_{jk}} \tag{8}$$

w_n is the set of numbers in the problem description. In this way, the model is capable of eliminating the spurious word error.

3.3 Align Numbers

In the decoding phase, the model often generates equations with numbers in wrong positions. *Problem 2* in Figure 1 is an example. The output equation structure is correct by the basic model. However, the numbers are aligned wrongly. When decoding the last token, the model should pay more attention to the source token "1170". Instead, the model wrongly aligned to the source token "30".

One characteristic of math word problems is that they have explicit alignment information between numbers in the problems and numbers in the equations. To improve the alignment in math problems, we use a supervised align mechanism to guide the training of our seq2seq model. Similar to Mi et al. (2016), the basic idea is minimize the cost of distance between the "true" alignment and the model predicted attention. We use the cross entropy loss function in the following:

$$\delta(a^i, \hat{a}^i) = -\sum_m \sum_n \hat{a}^i_{m,n} \times \log a^i_{m,n} \tag{9}$$

Please note that the disagreement only exists in numbers. The actual distribution $\hat{a}^i_{m,n}$ will be 1 if the *m*th token in the source is a number and equals to the *n*th tokens in the target sequence, otherwise it is 0.

4 Reinforcement Learning

As previously mentioned, MLE optimizes the surrogate objective of maximizing equation likelihood, while the evaluation metric of the task is solution accuracy. Besides, MLE assumes ground truth is provided at each timestep to predict the next token during training, which is not the case at test time. To remedy the discrepancy, we adopt the reinforcement learning technique, which directly optimizes the solution accuracy.

4.1 Policy Learning

The goal of the REINFORCE (Williams, 1992) is to find an agent that maximizes the task-level expected reward. We view our seq2seq model as a RL agent, which takes math problem description as input and then at each step, outputs a token y_j either by generating from the vocabulary or by copying numbers from the input problem. The agent follows a policy $\pi(y_j|\cdot)$, which we define as Equ 8.

The loss function and the gradient in the reinforcement learning are:

$$L_{RL} = -\sum_i \mathbb{E}_{p_\theta(\mathbf{y}^i|\mathbf{x}^i)}[R(\mathbf{x}^i, \mathbf{y}^i)] \tag{10}$$

$$\nabla_\theta L_{RL} = -\sum_i \mathbb{E}_{p_\theta(\mathbf{y}^i|\mathbf{x}^i)}[R(\mathbf{x}^i, \mathbf{y}^i)\nabla_\theta \log p_\theta(\mathbf{y}^i|\mathbf{x}^i)] \tag{11}$$

$$\approx -\sum_i \sum_{\mathbf{y}^i} p_\theta(\mathbf{y}^i|\mathbf{x}^i) R(\mathbf{x}^i, \mathbf{y}^i)\nabla_\theta \log p_\theta(\mathbf{y}^i|\mathbf{x}^i) \tag{12}$$

where $R(\mathbf{x}^i, \mathbf{y}^i)$ is the reward function. We define it as +1 if $\mathbf{y}^i$ yields to the correct solution, and -1 if $\mathbf{y}^i$ is not a valid equation or yields to a wrong solution.

Gradient Approximation It is often intractable to compute the gradient (Equ 11) because it involves taking an expectation over all possible equations. Therefore we sample from the model by using the top-k equations in the beam to approximate the gradient (Equ 12). Note that the gradient weights $p_\theta(\mathbf{y}^i|\mathbf{x}^i)$ of our sampling equations are renormalized to be summed up to 1.

In practice, the REINFORCE algorithm is unstable and converges slowly when the search space is large. Thus we pre-train our model based on maximum-likelihood for a few iterations before starting reinforcement learning.

4.2 Mixed Objective Function

To consider the alignment loss in Section 3.3, we define a mixed learning objective function:

$$L = L_{RL} + \lambda * \delta(a^i, \hat{a}^i) \tag{13}$$

where λ is a hyper-parameter that controls the magnitude of number alignment disagreement in the loss. In the pre-training step based on MLE, we replace the loss L_{RL} with the negative log likelihood:

$$L_{MLE} = -\sum_i \log p(\mathbf{y}^i|\mathbf{x}^i; \theta) \tag{14}$$

5 Model Ensemble

In this section, we observe and discuss that the neural model and the traditional feature-based model are complementary. To explore the advantage of both approaches, we combine two models by adding the result of our neural model as a feature to the feature-based model.

5.1 Feature-based Model

We use the state-of-the-art feature-based model (Huang et al., 2017) in our experiments. It contains two stages:

(1) Template retrieval. Candidate templates are derived from the training data. Given a problem p_i, they create a feature vector $f(p_i, t_j)$ for each candidate template t_j, and learn to rank the templates. They retrieve the top N templates.

(2) Equation ranking. Given the top N templates, they generate candidate equations with all possible number alignments. For example, there are two candidate equations given the template $x = n_1 - n_2$ and the numbers $\{3, 5\}$: $x = 3 - 5$ and $x = 5 - 3$. Similar to the previous stage, they create a feature vector $f(p_i, e_k)$ for each candidate equation e_k, and learn to rank the equations.

5.2 Generalization Ability of Neural Model

In Figure 2 we show the statistic of problems solved by our neural model and the feature-based model on Dolphin18K. 10.4% of problems can be solved correctly by both models, 18.0% can only be solved correctly by the feature-based model and 5.5% can only be solved correctly by our neural model.

To explore the generalization ability of our neural model, we further look into the 5.5% of problems that can only be solved correctly by our neural model (blue area in Figure 2). They can be summarized into two categories (examples are shown in Table 1):

(1) **Ability to generate new equation templates**. Among the 5.5% of problems, there are 32.3% for which the template does not exist in the training data. Note that the feature-based model can only retrieve candidate templates from the training data. It means that the neural model has the ability to generate new equation templates, similar to the observation in Wang et al. (2017).

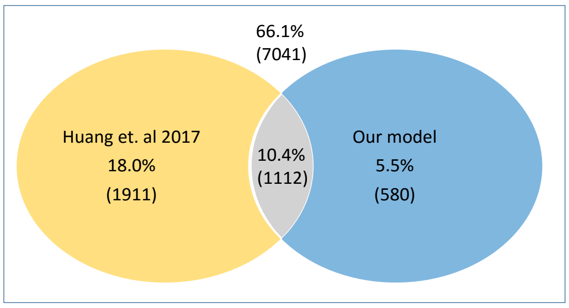


Figure 2: Statistic of problems solved by our neural model and feature-based model.

(2) **Ability to capture novel features**. For the remaining 67.7% of problems that can only be solved correctly by our neural model, their templates exist at least one time in the training data. The feature-based model should be able to retrieve the correct template and get the correct ranking for the equation, but it fails. This indicates that the neural model can capture novel features that the feature-based model is missing.

(1) **Problem**: Find 2 consecutive even integer such that 5 time the small integer be 10 more than 3 time the large integer. **Equation**: $5 * (2 * x) = 3 * (2 * x + 2) + 10$ (Our model generates the equation of which template does not exist in the training data)
(2) **Problem**: The area of a rectangular garden is 4472 ft$\hat{2}$. If the length of the garden is 86 feet, what is its width? **Equation**: $86 * x = 4472$ (its template exists in the training data)

Table 1: Example problems that are only solved correctly by our neural model.

5.3 Model Ensemble

In previous section, we observe that neural model is complementary to the feature-based approach when the templates are sparse or do not exist in the training data. Hence, it is intuitive to combine the neural model and feature-based approach to obtain better performance. To ensemble both models, We incorporate the neural model output in the two stages of the feature-based approach by (Huang et al., 2017):

(1) In the template retrieval stage, we add a feature of neural template. Given a problem, we derive the template t_{seq} of our neural output equation. For each candidate template t_i, if it is equivalent to t_{seq}, we set the value of the neural template feature to 1, otherwise set to 0.

(2) In the equation ranking stage, we add a feature of neural answer. Given a problem, we calculate the answer of our output equation a_{seq}. For each candidate equation e_j, if its answer a_j is equal to a_{seq}, we set the value of the neural answer feature to 1, otherwise set to 0.

6 Experiments

In this section, we test the performance of our model on three datasets. Furthermore, we conduct experiments to examine the effectiveness of our neural model as a feature in the hybrid model.

6.1 Implementation Details

Experiments are done in 5-fold cross-validation: in each run, 70% is used for training, 10% for validation, and 20% for testing. We report answer accuracy. The dimension of encoder hidden state, decoder hidden state and embeddings are 100 in NumWord and Alg514, 512 in Dolphin18K. All model parameters are initialized randomly with Gaussian distribution. The hyper-parameter λ for supervised attention of alignment is set to 1.0. We use SGD optimizer with decaying learning rate initialized as 0.5. Dropout rate is set to 0.5. The criterion for learning to stop is answer accuracy in validation set. The vocabulary consists of words observed in the training data more than or equal to N times. We set $N = 1$ for NumWord and $N = 5$ for the other two datasets. The beam size is set to 20 in the decoding stage. For reinforcement learning, we utilize a pre-training with maximum likelihood for 50 iterations. We set the beam size for sampling to 10. We tune all the hyper-parameters using a separate dev set.

6.2 Results

We implement the approach in Wang et al. (2017) as the baseline (**Seq2SeqAttn**). Its performance on Alg514 is 19.4%, versus 17.2% they reported.

Models	Alg514	NumWordT6	NumWordT1	Dolphin18KT6	Dolphin18KT1
Seq2SeqAttn (MLE)	19.4%	19.7%	11.0%	13.0%	10.2%
+Copy (MLE)	41.4%	59.9%	23.0%	20.2%	12.9%
+Copy+Align (MLE)	41.8%	60.4%	26.8%	21.0%	13.1%
+Copy+Align (RL)	44.5%	64.0%	29.2%	23.3%	15.9%
Huang et al. (2017)	81.6%	42.0%	20.8%	30.6%	28.4%
+ CASS^{RL} (hybrid)	**82.5%**	**65.8%**	**29.7%**	**33.2%**	**29.0%**

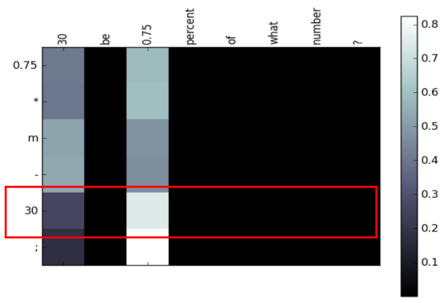
Table 2: Performances on three datasets. "*CASS*" means Seq2SeqAttn + Copy + Align.

From Table 2, we can see that our approach significantly improves the performance over the baseline. Especially on NumWord dataset, our model greatly exceeds the baseline with 44.3% increase on $T6$ and 18.2% increase on $T1$, and is already better than current state-of-the-art model (Huang et al., 2017).

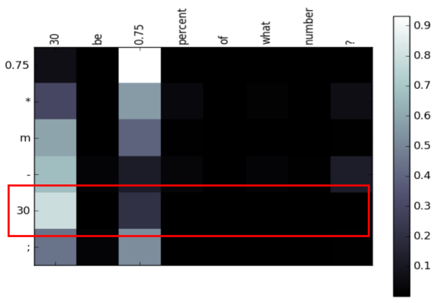
Copy impact The copy mechanism contributes greatly on all three datasets as shown in Table 2. It achieves a 40.2% accuracy on NumWordT6, even outperforms the feature-based models. In the most challenging dataset Dolphin18K among the three, with more diverse problems and larger problem size, our model still gets a 7.2% increase on Dolphin18KT6 and 2.7% increase on Dolphin18KT1.

Align impact From Table 2, we can see our model achieves a consistent improvements over all three datasets using number alignment. In the math problem in Figure 3a, for the target number "30", Seq2SeqAttn + Copy model has a higher attention weight of the source number "0.75" than the source number "30". Thus the model wrongly aligned the target number "30" to the source number "0.75". After incorporating the alignment mechanism, the target number "30" is now aligned correctly as shown in Figure 3b.

Reinforcement learning We can see RL performs substantially better than MLE. As the objective of RL is to optimize the solution accuracy, we further investigate the model's capability of considering multiple admissible equations. We compare the annotated equations with the equations generated by our model on Dolphin18KT1. Using RL training, there are **7.1%** of problems of which the predicted equations are different from the annotated ones but still yield to correct solutions, compared to 4.4% using MLE.



(a) **Seq2SeqAttn+Copy**



(b) **Seq2SeqAttn+Copy+Align**

Figure 3: Example alignments of (a) Seq2SeqAttn + Copy, (b) Seq2SeqAttn + Copy + Align.

6.3 Model Ensemble

We further test the performance of the hybrid model that combines our neural model into the feature-based model.

From Table 2, we can see that when our neural model is incorporated into the feature-based model, we achieve the state of the art. This indicates that the two models are complementary and the hybrid model takes advantages of both models.

Furthermore, we show detailed results when our neural model is incorporated in each of the two stages:

(1) Template retrieval. Same as Huang et al. (2017), we report Hit@N accuracy, which means the correct template for a problem is included in the top N list returned by the model. Table 3 shows the Hit@1/Hit@3 accuracy.

We can see that incorporating neural model helps increase template Hit@N accuracy, which means it can retrieve accurate templates for more problems. Except for Alg514, adding the neural feature drops Hit@3 by 1.3%. On Alg514, the feature-based model already outperforms our neural model. In addition, its data size is the smallest among the three datasets, which might be challenging for neural model to learn from. Therefore, adding neural feature on this dataset might bring much noise that leads to performance decrease.

Dataset	Huang et al. (2017)	+Neural
Alg514	62.8/80.9	60.3/79.6
NumWordT6	38.6/70.0	**38.9/72.9**
NumWordT1	20.0/35.1	20.4/35.1
Dolphin18KT6	27.3/39.7	**27.3/41.1**
Dolphin18KT1	17.5/26.3	**18.2/27.1**

Table 3: Accuracy (%) of template retrieval with top 1 / top 3 templates retrieved.

(2) Equation ranking (final accuracy). After retrieving the top N templates from the previous stage, we align numbers with template slots to generate candidate equations for ranking. The final results are shown in the last row of Table 2.

From the table, we can see that the neural feature is effective, achieving the state of the art on all three datasets. Especially, on NumWord dataset, incorporating neural model contributes to 23.8% accuracy increase on T6 and 8.9% increase on T1. Surprisingly, the performance on Alg514 is still improved, though the template retrieval accuracy in previous stage decreased. This indicates that the feature-based model has included some errors that caused by wrong number alignment in the second stage, and our neural feature eliminates this type of errors.

7 Related Work

Our work is related to three research areas: math word problem solving, the seq2seq model, and reinforcement learning for sequence generation.

7.1 Math Word Problem Solving

The approaches to solve math word problems can be divided into three categories: rule-based approach, feature-based approach, and neural-based approach.

Rule-based approaches (Bobrow, 1964a; Bobrow, 1964b; Bakman, 2007; Liguda and Pfeiffer, 2012; Shi et al., 2015) accept only well formed input sentences and map them into predefined structures by rules. These methods require strict constraints on both input text and math problem types.

Feature-based approaches design features and learn rankers to rank equation candidates to the math problems. Hosseini et al. (2014) design features to classify verbs to addition or subtraction. Kushman et al. (2014), Zhou et al. (2015), Upadhyay et al. (2016) use features such as dependency path between two numbers. Koncel-Kedziorski et al. (2015), Roy and Roth (2015) extract quantity information as features. Wang et al. (2018) extract features for quantity pairs and uses a reinforcement framework to construct an equation tree with the constraint of one unknown variable. Roy and Roth (2015), Roy et al. (2016) leverage the tree structure of equations. Mitra and Baral (2016), Roy and Roth (2018) design features for a few math concepts (e.g. Part-Whole, Comparison). Huang et al. (2017), Roy and Roth (2017) focus on the use of fine-grained expression and number units. These approaches requires manual feature design and it is difficult to generalizing the features to other problem types.

Recently, researchers try to build end-to-end neural models to solve math word problems. Ling et al. (2017) focus on multiple-choice problems. It takes a problem description as input and outputs the rationale and the final choice. Wang et al. (2017) apply a standard seq2seq model to generate equations under the constraint of one variable. However, the model is prone to generate numbers that do not exist in the problem or in wrong positions. Our model addresses these issues by incorporating copy and align.

7.2 Sequence-to-Sequence Models

Recent applications of seq2seq model in many areas have shown promising results.

Copy mechanism is proved effective in dealing with rare or unknown words. The main idea is to decide when and what to copy from the source words in the decoding phase. Jia and Liang (2016) use an attention-based copy mechanism to copy arguments from natural language query to logical form for the task of semantic parsing. Gulcehre et al. (2016) design a pointer network to select tokens in the input. Different from previous work, we consider copying only numbers from the problem description.

Some recent work have been proposed to improve the alignment for neural machine translation (Liu et al., 2016; Mi et al., 2016). They introduced a supervised attention mechanism to utilize the word alignment information between sentence pairs in the training data. They first obtain soft word alignments from conventional alignment models. Then they try minimize the distance between the soft word alignments and the word alignments based on model attention in the training procedure. In our math problems, the alignment information is explicit and can be directly obtained.

7.3 Reinforcement Learning for Sequence Generation

The classic REINFORCE algorithm (Williams, 1992) has been applied to solve a wide variety of tasks: machine translation (Norouzi et al., 2016), image captioning (Rennie et al., 2017), semantic parsing (Liang et al., 2017; Guu et al., 2017) and summarization (Paulus et al., 2018). Reinforcement learning is applied usually when the evaluation metric is non-differentiable, or there are multiple candidates that yields to the ground truth despite of token orders in target sequence. It trains an agent with a given environment to directly optimize the task evaluation metric (e.g., BLEU or ROUGE). We apply reinforcement learning in math problem solving for two reasons: (1) we evaluate our task with solution accuracy which is not directly optimized in maximum likelihood estimation; (2) there are multiple equations that yield to the correct solution which maximum likelihood estimation would ignore.

8 Conclusion

In this paper, we follow previous work that applies seq2seq model to solve math word problems. We augment the model with two mechanisms: copy and align. When training the model, we adopt the reinforcement learning to directly optimize the solution accuracy. Our model significantly improves the performance. To further explore the effectiveness of both neural approach and feature-based approach, we add our model output as a feature into the feature-based model. The combined model leverages advantages of both approaches and achieves the-state-of-the-art result.

In the future work, we will design more methods to combine the two models to better leverage each approach. Furthermore, for math problems, several modules are important, such as mathematical concept and commonsense knowledge, which we plan to incorporate into our model in the future.

Acknowledgements

Thanks to the anonymous reviewers for their helpful comments and suggestions. This work is supported by the National Natural Science Foundation of China (61472453, U1401256, U1501252, U1611264,U1711261,U1711262). Jian Yin is the corresponding author.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conferene on Learning Representation*.

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. http://arxiv.org/abs/math/0701393.

Daniel G. Bobrow. 1964a. Natural language input for a computer problem solving system. Technical report, Cambridge, MA, USA.

Daniel G. Bobrow. 1964b. Natural language input for a computer problem solving system. Ph.D. Thesis.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Kelvin Guu, Panupong Pasupat, Evan Zheran Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*.

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*, October.

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 52nd Annual Meeting of the Association for Computational Linguistics*.

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to solve math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Robin Jia and Percy Liang. 2016. Data recombinatin for neural semantic parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

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*.

Chen Liang, Jonathan Berant, Quoc Le, Kennet D.Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*.

Christian Liguda and Thies Pfeiffer. 2012. Modeling math word problems with augmented semantic networks. In *Natural Language Processing and Information Systems. International Conference on Applications of Natural Language to Information Systems (NLDB-2012)*, pages 247–252.

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 53rd Annual Meeting of the Association for Computational Linguistics*.

Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. 2016. Neural machine translation with supervised attention. In *Proceedings of the COLING 2016*.

Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. 2016. Supervised attentions for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Mohammad Norouzi, Samy Bengio, Zhifeng Chen, Navdeep Jaitly, Mike Schuster, Yonghui Wu, and Dale Schuurmans. 2016. Reward augmented maximum likelihood for neural structured prediction. In *Advances in Neural Information Processing Systems*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *Sixth International Conference on Learning Representations*.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *2017 Conference on Computer Vision and Pattern Recognition*.

Subhro Roy and Subhro Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752. The Association for Computational Linguistics.

Subhro Roy and Dan Roth. 2017. Unit dependency graph and its application to arithmetic word problem solving. In *Proceedings of the 2017 Conference on Association for the Advancement of Artificial Intelligence*.

Subhro Roy and Dan Roth. 2018. Mapping to declarative knowledge for word problem solving. In *Transactions of the Association for Computational Linguistic*.

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*.

Shuming Shi, Wang Yuehui, 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*.

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*.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural model for math word problem problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

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 *Thrity-Second AAAI Conference on Artificial Intelligence*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256.

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*.