Using Intermediate Representations to Solve Math Word Problems

Danqing Huang[1]*, **Jin-Ge Yao**[2], **Chin-Yew Lin**[2], **Qingyu Zhou**[3], **and Jian Yin**[1]
{huangdq2@mail2,issjyin@mail}.sysu.edu.cn
{Jinge.Yao,cyl}@microsoft.com
qyzhou@hit.edu.cn
[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]Microsoft Research [3] Harbin Institute of Technology

Abstract

To solve math word problems, previous statistical approaches attempt at learning a direct mapping from a problem description to its corresponding equation system. However, such mappings do not include the information of a few higher-order operations that cannot be explicitly represented in equations but are required to solve the problem. The gap between natural language and equations makes it difficult for a learned model to generalize from limited data. In this work we present an intermediate meaning representation scheme that tries to reduce this gap. We use a sequence-to-sequence model with a novel attention regularization term to generate the intermediate forms, then execute them to obtain the final answers. Since the intermediate forms are latent, we propose an iterative labeling framework for learning by leveraging supervision signals from both equations and answers. Our experiments show using intermediate forms outperforms directly predicting equations.

1 Introduction

There is a growing interest in math word problem solving (Kushman et al., 2014; Koncel-Kedziorski et al., 2015; Huang et al., 2017; Roy and Roth, 2018). It requires reasoning with respect to sets of numbers or variables, which is an essential capability in many other natural language understanding tasks. Consider the math problems shown in Table 1. To solve the problems, one needs to know how many numbers to be summed up (e.g. "2 numbers/3 numbers"), and the relation between variables ("the first/second number"). However, an equation system does not encode these information explicitly. For example, an equation represents "the sum of 2 numbers" as $(x + y)$ and "the sum of 3 numbers" as $(x + y + z)$. This makes it difficult to generalize to cases unseen from data (e.g. "the sum of 100 numbers").

1) **The sum of 2 numbers** is 18. **The first number** is 4 more than **the second number**. Find the two numbers. **Equations:** $x + y = 18, x = y + 4$
2) **The sum of 3 numbers** is 15. **The larger number** is 4 times **the smallest** and **the middle number** is 5. What are the numbers? **Equations:** $x + y + z = 15, x = 4 * z, y = 5$

Table 1: Math word problems. Equations have lost the information of *count, max, ordinal* operations.

This paper presents a new intermediate meaning representation scheme for solving math problems, aiming at closing the semantic gap between natural language and equations. To generate the intermediate forms, we adapt a sequence-to-sequence (seq2seq) network following recent work that tries to generate equations from problem descriptions for this task. Wang et al. (2017) have shown that seq2seq models have the power to generate equations of which problem types do not exist in training data. In this paper, we propose a new method which adds an extra meaning representation and generate an intermediate form as output. Additionally, we observe that the attention weights of the seq2seq model repetitively concentrates on numbers in the problem description. To address the issue, we further propose to use a form of attention regularization.

To train the model without explicit annotations of intermediate forms, we propose an iterative la-

*Work done while this author was an intern at Microsoft Research.

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 419–428
Melbourne, Australia, July 15 - 20, 2018.

beling framework to leverage signals from both equations and their solutions. We first derive possible intermediate forms with ambiguity using the gold-standard equation systems, and use these forms for training to get a pre-trained model. Then we iteratively refine the intermediate forms using the learned model and the signals from the gold-standard answers.

We conduct experiments on two publicly available math problem datasets. Our experimental results show that using the intermediate forms for training performs significantly better than directly mapping problems to equation systems. Furthermore, our iterative labeling framework creates better labeled data with intermediate forms for training, which leads to improved performance.

To summarize, our contributions include:

- We present a new intermediate meaning representation scheme for solving math problems.
- We design an iterative labeling framework to automatically augment training data with intermediate meaning representation.
- We propose using attention regularization in training to address the issue of incorrect attention in the seq2seq model.
- We verify the effectiveness of our proposed solutions by conducting experiments and analysis on real-world datasets.

2 Meaning Representation

In this section, we will compare meaning representations for solving math problems and introduce the proposed intermediate meaning representation.

2.1 Meaning Representations for Math Problem Solving

We first discuss two meaning representation schemes for math problem solving.

An **equation system** is a collection of one or more equations involving the same set of variables, which should be considered as highly abstractive symbolic representation.

The **Dolphin Language** is introduced by Shi et al. (2015). It contains about 35 math-related classes and over 200 math-related functions, with additional classes and functions automatically mined from Freebase.

Unfortunately, these representation schemes do not generalize well. Consider the two problems listed in Table 2. They belong to the same type of problems asking about the summation of consecutive integers. However, their meaning representations are very different in the Dolphin language and in equations. On one hand, the Dolphin language aligns too closely with natural utterances. Since the math problem descriptions are diverse in using various nouns and verbs, Dolphin language may represent the same type of problems differently. On the other hand, an equation system does not explicitly represent useful problem solving information such as "*number of variables*" and "*numbers are consecutive*"

2.2 Intermediate Meaning Representation

To bridge the semantic gap between the two meaning representations, we present a new intermediate meaning representation scheme for math problem solving. It consists of 6 *classes* and 23 *functions*. Here a *class* is the set of entities with the same semantic properties and can be inherited (e.g. $2 \in int$, $int \sqsubseteq num$). A *function* is comprised of a name, a list of arguments with corresponding types, and a return type. For example, there are two overloaded definitions for the function *math#sum* (Table 3). These forms can be constructed by recursively applying joint operations on functions with class type constraints. Our representation scheme attempts to borrow the explicit use of higher-order functions from the Dolphin language, while avoiding to be too specific. Meanwhile, the intermediate forms are not as concise as the equation systems (Table 2). We leave more detailed definitions to the supplement material due to space limit.

3 Problem Statement

Given a math word problem p, our goal is to predict its answer A_p. For each problem we have annotations of both the equation system E_p and the answer A_p available for training. The latent intermediate form will be denoted as LF_p.

We formulate math problem solving as a sequence prediction task, taking the sequence of words in a math problem as input and generating a sequence of tokens in its corresponding intermediate form as output. We then execute the intermediate form to obtain the final answer. We evaluate the task using answer accuracy on two publicly

Problem 1: Find three consecutive integers with a sum of 267.
Dolphin Language: vf.find(cat('integers'), count:3, adj.consecutive, (math#sum(pron.that, 267, det.a)))
Equation: $x + (x + 1) + (x + 2) = 267$
This work: math#consecutive(3), math#sum(cnt: 3) = 267

Problem 2: What are 5 consecutive numbers total 95?
Dolphin Language: wh.vf.math.total((cat('numbers'), count:5, pron.what, adj.consecutive), 95)
Equation: $x + (x + 1) + (x + 2) + (x + 3) + (x + 4) = 95$
This work: math#consecutive(5), math#sum(cnt: 5) = 95

Table 2: Different representations for math problems. Dolphin language is detailed ('all words'). Equation system is coarse that it represents many functions implicitly, such as "*count*", "*consecutive*".

Classes
int, float, num, unk, var, list
Functions
ret:int count($1:list): number of variables in $1
ret:var max($1:list): variable of max value in $1
ret:var math#product($1,$2:var): $1 times $2
ret:var math#sum($1:list): sum of variables in $1
ret:var math#sum(cnt:$1:int): sum of $1 unks
Example
Four times the sum of three and a number is 10.
-> math#product(4, math#sum(3, m))=10

Table 3: Examples of classes and functions in our intermediate representation. "ret" stands for return type. $1, $2 are arguments with its types.

available math word problem datasets[1]:

- **Number Word Problem** (NumWord) is created by Shi et al. (2015). It contains 1,878 number word problems (verbally expressed number problems, such as the examples in Table 1). Its linear subset (subset of problems that can be solved by linear equation systems) has 986 problems, only involving four basic operations $\{+, -, *, /\}$.

- **Dolphin18K** is created by Huang et al. (2016). It contains 18,711 math word problems collected from Yahoo! Answers[2]. Since it contains some problems without equations, we only use the subset of 10,644 problems which are paired with their equation systems.

[1]Other small datasets with 4 basic operations $\{+, -, *, /\}$ and only one unknown variable are considered as subsets of our datasets.

[2]https://answers.yahoo.com/

4 Model

In this section, we describe (1) the basic sequence-to-sequence model, and (2) attention regularization.

4.1 Sequence-to-Sequence RNN Model

Our baseline model is based on sequence-to-sequence learning (Sutskever et al., 2014) with attention (Bahdanau et al., 2015) and copy mechanism (Gulcehre et al., 2016; Gu et al., 2016).

Encoder: The encoder is implemented as a single-layer bidirectional RNN with gated recurrent units (GRUs). It reads words one-by-one from the input problem, producing a sequence of hidden states $h_i = [h_i^F, h_i^B]$ with:

$$h_i^F = GRU(\phi^{in}(x_i), h_{i-1}^F), \quad (1)$$

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

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

Decoder with Copying: At each decoding step j, the decoder receives the word embedding of the previous word, and an attention function is applied to attend over the input words as follows:

$$e_{ji} = v^T \tanh(W_h h_i + W_s s_j + b_{attn}), \quad (3)$$

$$a_{ji} = \frac{\exp(e_{ji})}{\sum_{i'=1}^{m} \exp(e_{ji'})}, \quad (4)$$

$$c_j = \sum_{i=1}^{m} a_{ji} h_i, \quad (5)$$

where s_j is the decoder hidden state. Intuitively, a_{ji} defines the probability distribution of attention over the input words. They are computed from the unnormalized attention scores e_{ji}. c_j is the context vector, which is the weighted sum of the encoder hidden states.

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

$$p_{gen} = \sigma(w_c^T c_j + w_s^T s_j + b_{ptr}), \quad (6)$$

where w_c, w_s and b_{ptr} are model parameters. Next, p_{gen} is used as a soft switch: with probability p_{gen} the model decides to generate from the decoder state. The probability distribution over all words in the vocabulary is:

$$P_{RNN} = \text{softmax}(W[s_j, c_j] + b); \quad (7)$$

with probability $1 - p_{gen}$ the model decides to directly copy an input word according to its attention weight. This leads to the final distribution of decoder state outputs:

$$P(w_j = w|\cdot) = p_{gen} P_{RNN}(w) + (1 - p_{gen}) a_{ji} \quad (8)$$

4.2 Attention Regularization

In preliminary experiments, we observed that the attention weights in the baseline model repetitively concentrate on the numbers in the math problem description (will be discussed in later sections with Figure 1(a)). To address this issue, we regularize the accumulative attention weights for each input token using a rectified linear unit (ReLU) layer, leading to the regularization term:

$$\text{AttReg} = \sum_i \text{ReLU}(\sum_{j=0}^{T} a_{ji} - 1), \quad (9)$$

where $\text{ReLU}(x) = \max(x, 0)$. This term penalizes the accumulated attention weights on specific locations if it exceeds 1. Adding this term to the primary loss to get the final objective function:

$$Loss = -\sum_i \log p(\mathbf{y}^i|\mathbf{x}^i; \theta) + \lambda * \text{AttReg} \quad (10)$$

where λ is a hyper-parameter that controls the contribution of attention regularization in the loss.

The format of our attention regularization term resembles the coverage mechanism used in neural machine translation (Tu et al., 2016; Cohn et al., 2016), which encourages the coverage or fertility control for input tokens.

5 Iterative Labeling

Since explicit annotations of our intermediate forms do not exist, we propose an iterative labeling framework for training.

5.1 Deriving Latent Forms From Equations

We use the annotated equation systems to derive possible latent forms. First we define some simple rules that map an expression to our intermediate form. For example, we use regular expressions to match numbers and unknown variables. Example rules are shown in Table 4 (see Section 2 of the Supplement Material for all rules).

Regex/Rules	Class/Function
`\-?[0-9\.]+`	num
`[a-z]`	unk
`<num>|<unk>`	var
`(<var>\+)+<var>`	math#sum($1:list)
`(<unk>\+)+<unk>` $1=count of unk	math#sum (cnt:$1:int)

Table 4: Example rules for deriving latent forms from equation system.

5.2 Ambiguity in Derivation

For one equation system, several latent form derivations are possible. Take the following math problem as an example:

> *Find **3** consecutive integers that **3** times the sum of the first and the third is 79.*

Given the annotation of its equation $3 * (x + (x + 2)) = 79$, there are two possible latent intermediate forms:
1) math#consecutive(3), math#product(3, math#sum(ordinal(1), ordinal(3)))=79
2) math#consecutive(3), math#product(3, math#sum(min(), max()))=79

There exist two types of ambiguities: a) **operator ambiguity**. $(x + 2)$ may correspond to the operator "*ordinal(3)*" or "*max()*"; b) **alignment ambiguity**. For each "3" in the intermediate form, it is unclear which "3" in the input to be copied. Therefore, we may derive multiple intermediate forms with spurious ones for a training problem.

We can see from Table 5 that both datasets we used have the issue of ambiguity, containing about 20% of problems with operator ambiguity and 10% of problems with alignment ambiguity.

5.3 Iterative Labeling

To address the issue of ambiguity, we perform an iterative procedure where we search for correct intermediate forms to refine the training data. The

Dataset	Ambiguous		Ambig. #LF
	oper	align	(per prob)
NumWord (Linear)	28.0%	10.2%	3.67
NumWord (All)	26.9%	9.5%	4.29
Dolphin18K	35.9%	9.6%	3.86

Table 5: Statistics of latent forms on two datasets. The percentage of problems with operator and alignment ambiguity is shown in the 2*nd* and 3*rd* columns respectively. We also show the average number of intermediate forms of problems with derivation ambiguity in the rightmost column.

intuition is that a better model will lead to more correct latent form outputs, and more correct latent forms in training data will lead to a better model.

Algorithm 1 Iterative Labeling

Require:
(1) Tuples of (math problem description, equation system, answer) $D_n = \{(p_i, E_{p_i}, A_{p_i})\}$
(2) Possible latent forms $P_{LF} = \{(p_0, LF^1_{p_0}), (p_0, LF^2_{p_0}), ..., (p_n, LF^m_{p_n})\}$
(3) Beam size B
(4) training iterations N_{iter}, pre-training iterations N_{pre}
Procedure:
for $iter = 1$ to N_{iter} **do**
  if $iter < N_{pre}$ **then**
    $\theta \leftarrow$ MLE with P_{LF}
  else
    for (p, LF) in P_{LF} **do**
      C = Decode B latent forms given p
      for j in 1...B **do**
        if Ans(C_j) is correct **then**
          $LF \Leftarrow C_j$
          break
    $\theta \leftarrow$ MLE with relabeled P_{LF}

Algorithm 1 describes our training procedure. As pre-training, we first update our model by maximum likelihood estimation (MLE) with all possible latent forms for N_{pre} iterations. Ambiguous and wrong latent forms may appear at this stage. This pre-training is to ensure faster convergence and a more stable model. After N_{pre} iterations, iterative labeling starts. We decode on each training instance with beam search. We declare C_j to be the *consistent form* in the beam if it can be executed to yield the correct answer. Therefore we can relabel the latent form LF with C_j for problem p and use the new pairs for training. If there is no consistent form in the beam, we keep it unchanged. With iterative labeling, we update our model by MLE with relabeled latent forms. There are two conditions of N_{pre} to consider:
(1) $N_{pre} = 0$, the training starts iterative labeling without pre-training.
(2) $N_{pre} = N_{iter}$, the training is pure MLE without iterative labeling.

6 Experiments

In this section, we compare our method against several strong baseline systems.

6.1 Experiment Setting

Following previous work, experiments are done in 5-fold cross validation: in each run, 20% is used for testing, 70% for training and 10% for validation.

Representation To make the task easier with less auxiliary nuisances (e.g. bracket pairs), we represent the intermediate forms in Polish notation. [3]

Implementation details The dimension of encoder hidden state, decoder hidden state and embeddings are 100 in NumWord, 512 in Dolphin18K. All model parameters are initialized randomly with Gaussian distribution. The hyper-parameter λ for the weight of attention regularization is set to 1.0 on NumWord and 0.4 on Dolphin18K. We use SGD optimizer with decaying learning rate initialized as 0.5. Dropout rate is set to 0.5. The stopping criterion for training is validation accuracy with the maximum number of iterations no more than 150. The vocabulary consists of words observed no less than N times in training set. We set $N = 1$ for NumWord and $N = 5$ for Dolphin18K. The beam size is set to 20 in the decoding stage. For iterative training, we first train a model for $N_{pre} = 50$ iterations for pre-training. We tune the hyper-parameters on a separate dev set.

We consider the following models for comparisons:

- **Wang et al. (2017):** a seq2seq model with attention mechanism. As preprocessing, it replaces numbers in the math problem with tokens $\{n_1, n_2, ...\}$. It generates **equation**

[3] https://en.wikipedia.org/wiki/Polish_notation

as output and recovers $\{n_1, n_2, ...\}$ to corresponding numbers in the post-processing.

- **Seq2Seq_Equ:** we implement a seq2seq model with attention and copy mechanism. Different from Wang et al. (2017), it has the ability to copy numbers from problem description.

- **Shi et al. (2015):** a rule-based system. It parses math problems into Dolphin language trees with predefined grammars and reasons across trees to get the equations with rules. We report numbers from their paper as the Dolphin language is not publicly available.

- **Huang et al. (2017):** the current state-of-the-art model on Dolphin18K. It is a feature-based model. It generates candidate equations and find the most probable equation by ranking with predefined features.

6.2 Results

Overall results are shown in Table 6. From the table, we can see that our final model (**Seq2Seq_LF+AttReg+Iter**) outperforms the neural-based baseline models (Wang et al. (2017)[4] and **Seq2Seq_Equ**). On Number word problem dataset, our model already outperforms the state-of-the-art feature-based model (Huang et al., 2017) by 40.8% and is comparable to the ruled-based model (Shi et al., 2015)[5].

Advantage of intermediate forms: From the first two rows, we can see that the seq2seq model which is trained to generate intermediate forms (**Seq2Seq_LF**) greatly outperforms the same model trained to generate equations (**Seq2Seq_Equ**). The use of intermediate forms helps more on NumWord than on Dolphin18K. This result is expected as the Dolphin18K dataset is more challenging, containing many other types of difficulties discussed in Section 6.3.

Effect of Attention Regularization: Attention regularization improves the seq2seq model on the two datasets as expected. Figure 1 shows an example. The attention regularization does meet the expectation: the alignments in Fig 1(b) are less concentrated on the numbers in the input and more importantly and alignments are more reasonable. For example, when generating "*math#product*" in the output, the attention is now correctly focused on the input token "*times*".

Effect of Iterative Labeling: We can see from Table 6 that iterative labeling clearly contributes to the accuracy increase on the two datasets. Now we compare the performance with and without pre-training in Table 7. When $N_{pre} = 0$ in Algorithm 1, the model starts iterative labeling from the first iteration without pre-training. We find that training with pre-training is substantially better, as the model without pre-training can be unstable and may generate misleading spurious candidate forms.

Next, we compare the performance with pure MLE training on NumWord (Linear) in Figure 2. The difference is that after 50 iterations of MLE training, iterative labeling would refine the latent forms of training data. In pure MLE training, the accuracy converges after 130 iterations. By using iterative labeling, the model achieves the accuracy of 61.6% at 110*th* iterations, which is faster to converge and leads to better performance.

Furthermore, to check whether iterative labeling actually resolves ambiguities in the intermediate forms of the training data, we manually sample 100 math problems with derivation ambiguity. **78%** of them are relabeled with correct latent forms as we have checked. From Table 8, we can see the latent form of one training problem is iteratively refined to the correct one.

6.3 Model Comparisons

To explore the generalization ability of the neural approach and better guide our future work, we compare the problems solved by our neural-based model with the rule-based model (Shi et al., 2015) and the feature-based model (Huang et al., 2017).

Neural-based v. Rule-based: On NumWord (ALL), **41.6%** of problems can be solved by both models. **15.5%** can only be solved by our neural model, while the rule-based model generates an empty or a wrong semantic tree due to the limitations of the predefined grammar. The neural model is more consistent with flexible word order and insertion of lexical items (e.g. rule-based model cannot handle the extra word 'whole' in "Find two consecutive **whole** numbers").

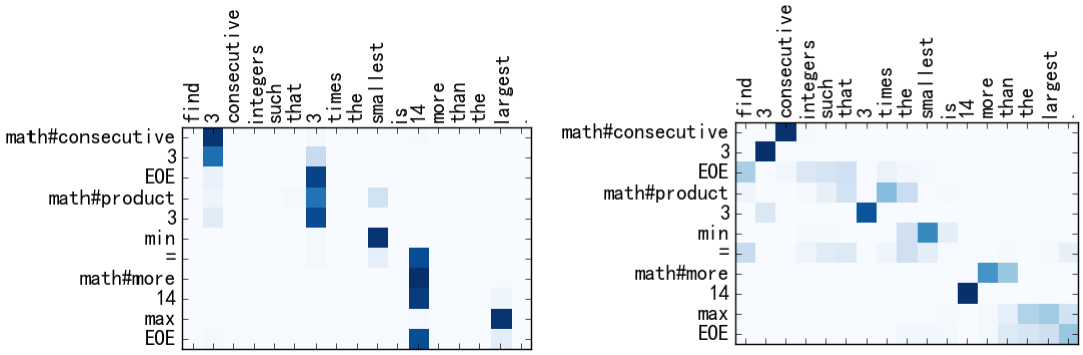
Neural-based v. Feature-based: On Dolphin18K, **9.2%** of problems can be solved by both models. **7.6%** can only be solved by our neural model, which indicates that the neural model

[4]We re-implement this since it is not publicly available.

[5]The system reports precision and recall. Since all the problems have answers, its recall equals to our accuracy.

Models	NumWord (Linear)	NumWord (ALL)	Dolphin18K (Linear)
Wang et al. (2017)	19.7%	14.6%	10.2%
Seq2Seq_Equ	26.8%	20.1%	13.1%
Seq2Seq_LF	50.8%	45.2%	13.9%
Seq2Seq_LF+AttReg	56.7%	54.0%	15.1%
Seq2Seq_LF+AttReg+Iter	61.6%	57.1%	16.8%
Shi et al. (2015)	63.6%	60.2%	n/a
Huang et al. (2017)	20.8%	n/a	28.4%

Table 6: Performances on two datasets. "**LF**" means that the model generates latent intermediate forms instead of equation systems. "**AttReg**" means attention regularization. "**Iter**" means iterative labeling. "**n/a**" means that the model does not run on the dataset.



(a) **seq2seq_LF** (b) **seq2seq_LF+AttReg**

Figure 1: Example alignments for one problem (darker color represents higher attention score).

	NumWord (Linear)	NumWord (ALL)	Dolphin18K (Linear)
-pre	58.1%	54.9%	14.9%
+pre	61.6%	57.1%	16.8%

Table 7: Performance with and without pre-training in iterative labeling.

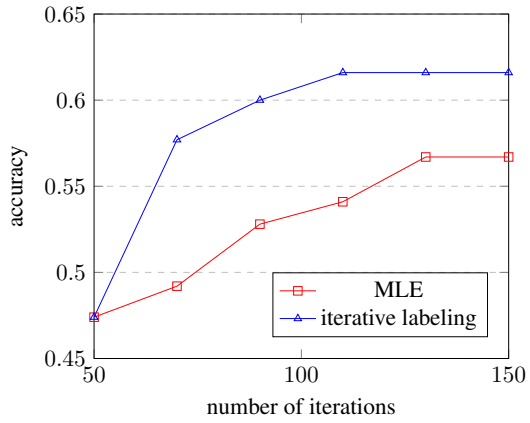


Figure 2: Accuracy with different iterations of training on NumWord (Linear).

can capture novel features that the feature-based model is missing.

While our neural model is complementary to the above mentioned models, we observe two main types of errors (more examples are shown in the supplementary material):

1. Natural language variations: Same type of problems can be described in different scenarios. The two problems: (1) "What is 10 minus 2?" and (2) "John has 10 apples. How many apples does John have after giving Mary 2 apples", lead to the same equation $x = 10 - 2$ but with very different descriptions. With limited size of data, we could not be expected to cover all possible ways to ask the same underlining math problems. Although the feature-based model has considered this with some features (e.g. POS Tag), the challenge is not well-addressed.

2. Nested operations: Some problems require multiple nested operations (e.g. "I think of a number, double it, add 3, multiply the answer by 3 and then add on the original number"). The rule-based model performs more consistently on this.

Training Problem:
*Find **2_0** consecutive integers which the first number is **2_1** more than **2_2** times the second number.*
Intermediate form in 1*st* iteration
(✗) math#consecutive(**2_0**), ordinal(1) = math#sum("**2_0**", math#product("**2_0**", "**max()**")
Intermediate form in 51*st* iteration
(✗) math#consecutive(**2_0**), ordinal(1) = math#sum(**2_1**, math#product("**2_0**", ordinal(2))
Intermediate form in 101*st* iteration
(✓) math#consecutive(**2_0**), ordinal(1) = math#sum(**2_1**, math#product(**2_2**, ordinal(2))

Table 8: Instance check of intermediate form for one math problem in several training iterations. **2_0** means the the first '2' in the input and so on. Tokens with quote marks mean that they are incorrect.

7 Related Work

Our work is related to two research areas: math word problem solving and semantic parsing.

7.1 Math Word Problem Solving

There are two major components in this task: (1) meaning representation; (2) learning framework.

Semantic Representation With the annotation of equation system, most approaches attempt at learning a direct mapping from math problem description to an equation system. There are other approaches considering an intermediate representation that bridges the semantic gap between natural language and equation system. Bakman (2007) defines a table of schema (e.g. Transfer-In-Place, Transfer-In-Ownership) with associated formulas in natural utterance. A math problem can be mapped into a list of schema instantiations, then converted to equations. Liguda and Pfeiffer (2012) use augmented semantic network to represent math problems, where nodes represent concepts of quantities and edges represent transition states. Shi et al. (2015) design a new meaning representation language called Dolphin Language (DOL) with over 200 math-related functions and more additional noun functions. With predefined rules, these approaches accept limited well-format input sentences. Inspired by these representations, our work describes a new formal language which is more compact and is effective in facilitating better machine learning performance.

Learning Framework In rule-based approaches (Bakman, 2007; Liguda and Pfeiffer, 2012; Shi et al., 2015), they map math problem description into structures with predefined grammars and rules.

Feature-based approaches contain two stages: (1) generate equation candidates; They either replace numbers of existing equations in the training data as new equations (Kushman et al., 2014; Zhou et al., 2015; Upadhyay et al., 2016), or enumerate possible combinations of math operators and numbers and variables (Koncel-Kedziorski et al., 2015), which leads to intractably huge search space. (2) predict equation with features. For example, Hosseini et al. (2014) design features to classify verbs to addition or subtraction. 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). Roy and Roth (2017) focus on the dependencies between number units. These approaches requires manual feature design and the features may be difficult to be generalized to other tasks.

Recently, there are a few works trying to build an end-to-end system with neural models. Ling et al. (2017) consider multiple-choice math problems and use a seq2seq model to generate rationale and the final choice (i.e. A, B, C, D). Wang et al. (2017) apply a seq2seq model to generate equations with the constraint of single unknown variable. Similarly, we use the seq2seq model but with novel attention regularization to address incorrect attention weights in the seq2seq model.

7.2 Semantic Parsing

Our work is also related to the classic settings of learning executable semantic parsers from indirect supervision (Clarke et al., 2010; Liang et al., 2011; Artzi and Zettlemoyer, 2011, 2013; Berant et al., 2013; Pasupat and Liang, 2016). Maximum marginal likelihood with beam search (Kwiatkowski et al., 2013; Pasupat and Liang, 2016; Ling et al., 2017) is traditionally used. It maximizes the marginal likelihood of all consistent logical forms being observed. Recently

reinforcement learning (Guu et al., 2017; Liang et al., 2017) has also been considered, which maximizes the expected reward over all possible logical forms. Different from them, we only consider one single consistent latent form per training instance by leveraging training signals from both the answer and the equation system, which should be more efficient for our task.

8 Conclusion

This paper presents an intermediate meaning representation scheme for math problem solving that bridges the semantic gap between natural language and equation systems. To generate intermediate forms, we propose a seq2seq model with novel attention regularization. Without explicit annotations of latent forms, we design an iterative labeling framework for training. Experimental result shows that using intermediate forms is more effective than directly using equations. Furthermore, our iterative labeling effectively resolves ambiguities and leads to better performances.

As shown in the error analysis, same types of problems can have different natural language expressions. In the future, we will focus on tackling this challenge. In addition, we plan to expand the coverage of our meaning representation to support more mathematic concepts.

Acknowledgments

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

References

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. In *Transactions of the Association for Computational Linguistics*.

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.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the worlds response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*.

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. 2016. Incorporating structural alignment biases into an attentional neural translation model. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

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*.

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*.

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*.

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*.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luku Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

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*.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

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*.

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*.

Panupong Pasupat and Percy Liang. 2016. Inferring logical forms from denotations. In *Proceedings of the 52nd Annual Meeting of 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 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, 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*.

Ilya Sutskever, Oriol Vinyals, and Quoc Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

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*.

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*.