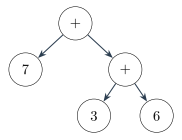
`Text2Math`: End-to-end Parsing Text into Math Expressions

Yanyan Zou and **Wei Lu**
StatNLP Research Group
Singapore University of Technology and Design
`yanyan_zou@mymail.sutd.edu.sg, luwei@sutd.edu.sg`

Abstract

We propose `Text2Math`, a model for semantically parsing text into math expressions. The model can be used to solve different math related problems including arithmetic word problems (Roy and Roth, 2017; Liang et al., 2018) and equation parsing problems (Roy et al., 2016). Unlike previous approaches, we tackle the problem from an end-to-end structured prediction perspective where our algorithm aims to predict the complete math expression at once as a tree structure, where minimal manual efforts are involved in the process. Empirical results on benchmark datasets demonstrate the efficacy of our approach.

Problem 1
Mike picked 7 apples. Nancy picked 3 apples and Keith picked 6 apples at the farm. In total, how many apples were picked?
Expression $(7 + (3 + 6))$



Answer 16

Problem 2
3 times one of the numbers is 11 less than 5 times the other.
Expression $(3 \times X_1) = (5 \times X_2) - 11$

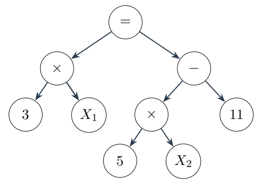


Figure 1: An example arithmetic word problem (top) and an example equation parsing problem (bottom) where the outputs can be represented as trees.

1 Introduction

Designing computer algorithms that can automatically solve math word problems is a challenge for the AI research community (Bobrow, 1964). Two representative tasks have been proposed and studied recently – solving arithmetic word problems (Wang et al., 2017; Roy and Roth, 2018; Zou and Lu, 2019b) and equation parsing (Roy et al., 2016), as illustrated in Figure 1. The former task focuses on mapping the input paragraph (which may involve multiple sentences) into a target math expression, from which an answer can be calculated. The latter task focuses on mapping a description (usually a single sentence) into a math equation that typically involves one or more unknowns. As we can observe from Figure 1, in both cases, the output can be represented as a tree structure.

Earlier approaches to solving arithmetic word problems focused on rule-based methods where hand-crafted rules have been used (Mukherjee and Garain, 2008; Hosseini et al., 2014). Recently, learning-based approaches based on statistical classifiers (Kushman et al., 2014; Roy and Roth, 2015; Roy et al., 2016; Liang et al., 2018) or neural networks (Wang et al., 2017, 2018b) have been used for making decisions in the expression[1] construction process. However, these models do not focus on predicting the target tree as a complete structure at once, but locally trained classifiers are often used and local decisions are then combined. Such local classifiers often make predictions on the choice of the underlying operator between two operands (e.g., numbers) appearing in the text in a particular order. As a result, special treatments of the non-commutative operators such as *subtraction* ($-$) and *division* ($\div$) are often involved, where the introduction of inverse operators is typically required[2]. Shi et al. (2015) tackled the problem from a structured prediction perspective, where a semantic parsing algorithm us-

[1]In this work, we use the term *expression* to refer to a math expression (for arithmetic word problem) or an equation (for equation parsing).

[2]For example, the inverse operator $-_i$ applied to two operands a followed by b is used to denote $b - a$.

ing context-free grammars (CFG) was used. However, their approach relies on semi-automatically generated rules and involves a manual step for interpreting the semantic representation they used.

While all these approaches focused on solving arithmetic word problems only, separate models have been developed for the task of equation parsing (Roy et al., 2016). It is not clear how easy each of these models specifically designed for one task can be adapted for the other task. Motivated by the observation that both problems involve mapping a text sequence to a tree structured representation, we propose `Text2Math` which regards both tasks as a class of structured prediction problems, and tackle them from a semantic parsing perspective. We make use of an end-to-end latent-variable approach to automatically produce the target math expression at once as a complete structure, where no prior knowledge on the operators (such as whether an operator is non-commutative) is required. Our model outperforms all baselines on two benchmark datasets. To the best of our knowledge, this is the first approach based on semantic parsing that tackles both arithmetic word problems and equation parsing with a single model. Our code is available at `http://statnlp.org/research/ta`.

2 Approach

2.1 Expression Tree

We first define tree representations for math expressions, which will then be regarded as the semantic representations used in the standard semantic parsing setup.

The nodes involved in the math expression trees can be classified into two categories, namely, *operator* and *quantity* nodes. Specifically, operator nodes are the tree nodes that define the types of operations involved in expressions. In this work we consider ADD (*addition*, $+$), SUB (*subtraction*, $-$), MUL (*multiplication*, $\times$) and DIV (*division*, $\div$). We also regard the equation sign ($=$) as an operation involved in math expressions and use EQU to denote it. We consider two types of quantity nodes: CON denoting constants, and VAR for unknown variables. Table 1 lists the above nodes. Each tree node comes with an *arity* which specifies the number of direct child nodes that should appear below the given node. For example, the operator node SUB with arity 2 is expecting two child nodes below it in the expression tree, while CON with arity 0 is supposed to be a leaf node. The two math expressions in Figure 1 can be equivalently represented by expression trees consisting of such nodes, as illustrated in Figure 2.

Category	Node	Interpretation	Arity
Operator	EQU	n_i EQU $n_j \Leftrightarrow (n_i = n_j)$	2
	ADD	n_i ADD $n_j \Leftrightarrow (n_i + n_j)$	2
	SUB	n_i SUB $n_j \Leftrightarrow (n_i - n_j)$	2
	MUL	n_i MUL $n_j \Leftrightarrow (n_i \times n_j)$	2
	DIV	n_i DIV $n_j \Leftrightarrow (n_i \div n_j)$	2
Quantity	CON	A constant	0
	VAR	A variable	0

Table 1: Expression tree nodes with interpretations, where n_i (n_j) refers to the first (second) operand.

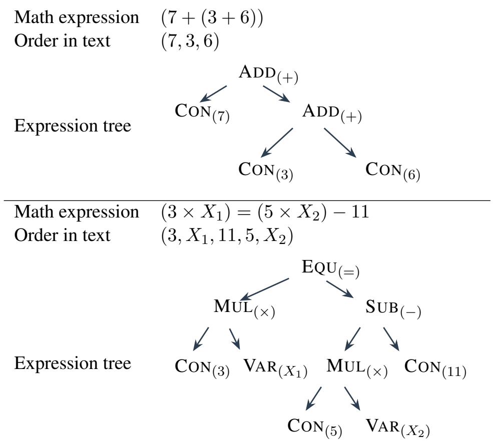


Figure 2: Expression trees for the two math expressions in Figure 1. "Order in text" refers to the order that the textual expressions of operands appear in the problem text. We use subscripts to indicate the actual semantic interpretations.

2.2 Latent Text-Math Tree

With the specifically designed expression trees for representing the math expressions, we will now be able to design a model for parsing the text into the expression tree. This is essentially a semantic parsing task. One of the key assumptions made by the various semantic parsing algorithms is the intermediate joint representation used for connecting the words and semantics (Wong and Mooney, 2006; Zettlemoyer and Collins, 2007; Lu et al., 2008; Artzi and Zettlemoyer, 2013b). In this work, we adopt an approach that is inspired by (Lu et al., 2008; Lu, 2014), which learns a latent joint representation for words and semantics in the form of *hybrid trees* where word-semantics correspondence information is captured. Specifically, we introduce a *text-math tree* representation that jointly encodes both text and the math expression tree.

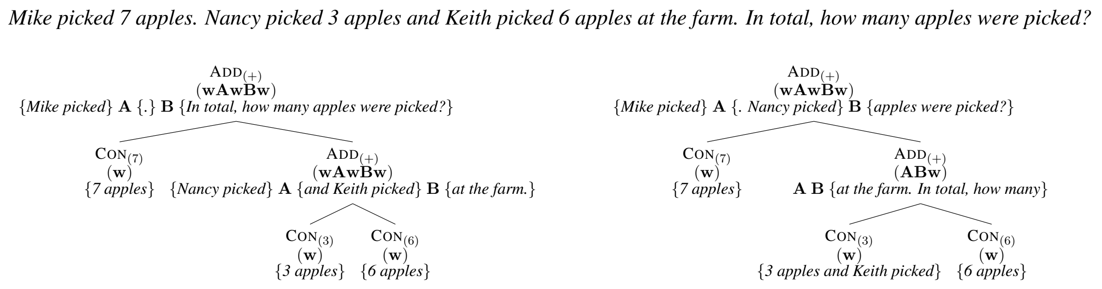

Mike picked 7 apples. Nancy picked 3 apples and Keith picked 6 apples at the farm. In total, how many apples were picked?
ADD(+)
(wAwBw)
{Mike picked} A {.} B {In total, how many apples were picked?}
CON(7)
(w)
{7 apples}
ADD(+)
(wAwBw)
{Nancy picked} A {and Keith picked} B {at the farm.}
CON(3)
(w)
{3 apples}
CON(6)
(w)
{6 apples}
ADD(+)
(wAwBw)
{Mike picked} A {. Nancy picked} B {apples were picked?}
CON(7)
(w)
{7 apples}
ADD(+)
(ABw)
A B {at the farm. In total, how many}
CON(3)
(w)
{3 apples and Keith picked}
CON(6)
(w)
{6 apples}

Figure 3: Example text-math trees for the arithmetic word problem example in Figure 1. The left tree captures the semantic correspondence well, while the right tree fails to capture the correct correspondence.

Such joint representations can be understood as a modified expression tree where each semantic node is now augmented with additional word information from the corresponding text.

From the joint representations we would be able to recover the semantic level correspondence information between words and math expressions. Possible joint representations of the two examples in Figure 1 are illustrated in Figure 3 and Figure 4 (left), respectively. Consider Problem 1 in Figure 1. We illustrate two possible text-math expression trees in Figure 3. Here each node in such joint representations is essentially a node in the original expression trees augmented with words from the problem text. For example, consider the left tree in Figure 3, the root node is an operator (the root node of the original expression tree) paired with discontiguous sequence of words "{*Mike picked*}...{.}...{*In total, how many apples were picked?*}" that appear in the problem text. This way, such a text-math tree is able to capture the semantic correspondence between words and basic units involved in the math expressions (i.e., operators, quantities). However, the text-math trees are not explicitly given during the training phase. For example, the right side of Figure 3 gives an alternative text-math tree that can also serve as a joint representation of both the text and the expression tree. Comparing both trees we may see the one on the left appears to be better at capturing the true semantic level correspondence between words and math expressions. Since there is no gold text-math tree explicitly given, we model it with a latent-variable approach.

Formally, given a text $\boldsymbol{x}$, paired with the expression $\boldsymbol{y}$ (or equivalently, the expression tree), we assume there exists a latent joint text-math representation in the form of text-math tree, that comprises exactly $\boldsymbol{x}$ and $\boldsymbol{y}$, denoted as $\boldsymbol{t}$. Each node is a word-semantics association $\langle x, y, p\rangle$ where x is a (possibly discontiguous) word sequence of $\boldsymbol{x}$ and y is an individual expression tree node from $\boldsymbol{y}$, and p is the word association pattern that is used to specify how words interact with the expression tree (further details will be provided in Sec. 2.3). Intuitively, such a joint text-math representation should precisely contain the exact information associated with the text and its corresponding math expression and nothing else. We will defer the discussion on how to exactly construct such joint representations until Sec. 2.3.

The training corpus provides both the problem text $\boldsymbol{x}$ and its math expression, which we represent with an expression tree $\boldsymbol{y}$. The joint representation $\boldsymbol{t}$ is not available in the training data, which we model as a latent variable. The conditional random fields (CRF) (Lafferty et al., 2001) has been successfully applied to many tasks in the NLP community (Lample et al., 2016; Zou and Lu, 2018, 2019a). In this work, we also apply CRF to model the conditional probability of the latent variable $\boldsymbol{t}$ and output expression $\boldsymbol{y}$, conditioned on the input $\boldsymbol{x}$. The objective is defined as follows:

$$P_{\Lambda,\Theta}(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{t}\in\mathcal{T}(\boldsymbol{x},\boldsymbol{y})} P_{\Lambda,\Theta}(\boldsymbol{y},\boldsymbol{t}|\boldsymbol{x})$$
$$= \frac{\sum_{\boldsymbol{t}\in\mathcal{T}(\boldsymbol{x},\boldsymbol{y})} e^{[\Lambda\cdot\Phi(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})+G_\Theta(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})]}}{\sum_{\boldsymbol{y}',\boldsymbol{t}'\in\mathcal{T}(\boldsymbol{x},\boldsymbol{y}')} e^{[\Lambda\cdot\Phi(\boldsymbol{x},\boldsymbol{y}',\boldsymbol{t}')+G_\Theta(\boldsymbol{x},\boldsymbol{y}',\boldsymbol{t}')]}} \quad (1)$$

where $\Phi(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})$ returns a list of discrete features defined over the tuple $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})$, Λ is the feature weight vector, G_Θ is a neural scoring function parameterized by Θ and $\mathcal{T}(\boldsymbol{x},\boldsymbol{y})$ is a set of possible joint representations (i.e., text-math trees) for the pair $(\boldsymbol{x},\boldsymbol{y})$.

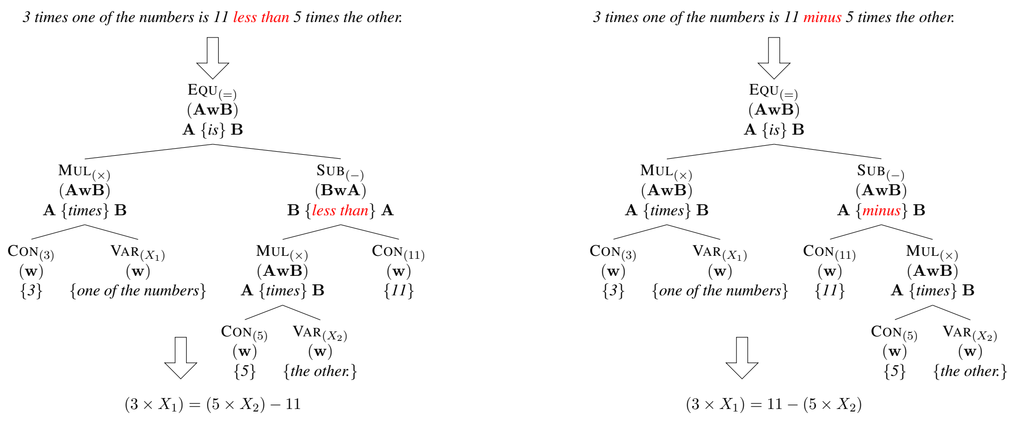


Figure 4: Left: example text-math tree for the equation parsing example in Figure 1, where the word association pattern $\mathbf{BwA}$ in the example is used for modeling reordering. Right: another example for a slightly different instance where the reordering is not required.

2.3 Inference

One challenge associated with the inference procedure in both training and decoding is how to handle the large space of latent structures defined by $\mathcal{T}(\boldsymbol{x}, \boldsymbol{y})$ and $\mathcal{T}(\boldsymbol{x})$. Without any constraints, searching or calculation that involves all possible structures within this space may be intractable. We therefore introduce some assumptions on the set of allowable structures, such that tractable inference can be applied to such structures.

We first introduce three symbols $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{w}$. The symbol $\mathbf{A}$ refers to a placeholder for the left sub-tree (rooted by the left child node), and similarly $\mathbf{B}$ is a placeholder for the right sub-tree. The symbol $\mathbf{w}$ refers to a contiguous sequence of (1 or more) words. We will then use these three symbols to define the set of *word association patterns*, which are used to specify how the words interact with the sub-trees of the current node. Specifically, for expression tree nodes with arity 0 (i.e., quantity nodes), only one pattern $\mathbf{w}$ is allowed to be attached to them, indicating that a contiguous word sequence from a given problem text is associated with such expression tree nodes. As for expression tree nodes with arity 2 (i.e., operator nodes), we define 16 allowable patterns denoted as $\{[\mathbf{w}]\mathbf{A}[\mathbf{w}]\mathbf{B}[\mathbf{w}], [\mathbf{w}]\mathbf{B}[\mathbf{w}]\mathbf{A}[\mathbf{w}]\}$, where [] denotes optional. Based on such word association patterns, we will be able to define a set of possible text-math trees for a particular text-expression pair that we regard as valid.

Before we formally define what is a valid text-math tree, let us look at an example in Figure 3 which shows two valid trees. First of all, we can verify that, if we exclude the words from both trees, we arrive at the math expression that corresponds to the text. Second, we can also recover the text information from such a joint representation. Let us look at the right tree in Figure 3. Consider the right sub-tree of the complete tree rooted by the node $\langle x, y, p\rangle = \langle\{$*at the farm. In total, how many*$\}$, ADD, $\mathbf{ABw}\rangle$. If we replace the placeholders $\mathbf{A}$ and $\mathbf{B}$ with the word sequences associated with its left and right sub-trees, respectively, we will arrive at the word sequence "*3 apples and Keith picked 6 apples at the farm. In total, how many*". Recursively performing such a rewriting procedure in a bottom-up manner, we will end up with a word sequence which is exactly the original input text as illustrated at the top of Figure 3.

Based on the above discussion, we can define $\mathcal{T}(\boldsymbol{x}, \boldsymbol{y})$ as a set that consists of the valid trees:

Definition 2.1 *For a given text $\boldsymbol{x}$ and an expression tree $\boldsymbol{y}$, a **valid text-math tree** satisfies the following two properties: 1) the semantics portion of the tree gives exactly $\boldsymbol{y}$, and 2) the text obtained through the recursive rewriting procedure discussed above gives exactly $\boldsymbol{x}$*[3].

Given the definition of the valid text-math trees, we will be able to use a bottom-up procedure to construct the set $\mathcal{T}(\boldsymbol{x}, \boldsymbol{y})$. Similarly, we will be able to construct the set $\mathcal{T}(\boldsymbol{x})$ by considering a forest-structured semantic representation that encodes all possible expression trees following (Lu, 2015). One nice property associated with considering only such joint representations is that there are known algorithms that can be used for performing efficient inference. Indeed, the resulting text-math trees are similar to the *hybrid tree* representations used in (Lu et al., 2008; Lu,

[3]We regard words that appear at different positions in $\boldsymbol{x}$ as distinct words, regardless of their string forms.

2014)[4], where dynamic programming based inference algorithms have been developed. Such algorithms allow $\mathcal{O}(n^3 m)$ time complexity for inference where n is the text length and m is the number of grammar rules[5] associated with the latent text-math trees.

We note that some prior systems (Roy and Roth, 2017, 2018) require extra inverse operators – inverse subtraction "$-_r$" and inverse division "$\div_r$" to handle the scenarios where the order of quantities appearing in the text is not consistent with the order that they appear in the expression. Exemplified by the example in the left of Figure 4, by introducing two operators $-_r$ and $\div_r$ to take their operands in a reverse order, the equation on the left is represented as "$(3 \times X_1) = 11 -_r (5 \times X_2)$". However, we do not need such two inverse operators. The two group patterns $[\mathbf{w}]\mathbf{A}[\mathbf{w}]\mathbf{B}[\mathbf{w}]$ and $[\mathbf{w}]\mathbf{B}[\mathbf{w}]\mathbf{A}[\mathbf{w}]$ are capable to capture both orders. A pattern from the first group handles the order that is consistent with the problem text, while a pattern from the second group is able to capture reordering of operands below an operator. Exemplified by Figure 4, reordering is required for the first example, but not for the second, though their texts only differ slightly. Unlike the second example, instead of using the pattern $\mathbf{AwB}$, the first joint representation adopts the pattern $\mathbf{BwA}$ for the SUB expression node. Thus, our model is able to work without the underlying knowledge on whether an operator is commutative or not.

2.4 Features

Discrete Features. The feature function $\Phi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t})$ is defined over each node $\langle x, y, p\rangle$ in the joint tree as well as the complete expression tree $\boldsymbol{y}$. For each node $\langle x, y, p\rangle$, we extract word n-gram, the word association pattern, and POS tags for words (Manning et al., 2014). The knowledge that whether a number is relevant to the question (if available in the annotated data) is also taken as a binary feature. To assess the quality of the structure associated with the expression tree (i.e., features defined over $\boldsymbol{y}$), we extract parent-child relational information (y_a, y_b) from $\boldsymbol{y}$, where y_a is the parent of y_b, as features. Following previous works (Roy et al., 2016; Liang et al., 2018), we also consider incorporating a lexicon in our model so as to make a fair comparison with such works, although we would like to stress that our model does not strictly require such lexicons for learning. More details are in supplementary material.

Neural Features. We design neural features over the pair of the L-sized window surrounding the target word x_i in $\boldsymbol{x}$ and an expression tree node y_j. The network takes as input the contiguous word sequence $(x_{i-L}, \ldots, x_i, \ldots, x_{i+L})$, whose distributed representation is a simple concatenation of embeddings of each word. The hidden layer applies an affine transformation with an element-wise nonlinear activation function, like tanh and ReLU. The final output layer contains as many nodes as there are expression tree nodes in the training set. The output is a score vector that gives a score for the input word sequence $(x_{i-L}, \ldots, x_i, \ldots, x_{i+L})$ and an expression tree node y_j. The neural scoring function is defined as follows:

$$G_\Theta(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t}) = \sum_{(x,y)\in\mathcal{W}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})} c(x, y, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t}) \times \psi(x, y)$$

where $\mathcal{W}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t})$ is the set of (x, y) pairs extracted from $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t})$, c returns the number of occurrences and $\psi(x, y)$ is a score of the target word x with L-sized windows and the expression tree node y, returned by the neural network. We regard L as a hyperparameter.

2.5 Algorithms

Given the complete training set, the log-likelihood is calculated as:

$$\begin{aligned}\mathcal{L}(\Lambda, \Theta) &= \sum_i \log P_{\Lambda,\Theta}(\boldsymbol{y}_i|\boldsymbol{x}_i) \\ &= \sum_i \log \sum_{\boldsymbol{t}\in\mathcal{T}(\boldsymbol{x}_i,\boldsymbol{y}_i)} P_{\Lambda,\Theta}(\boldsymbol{y}_i, \boldsymbol{t}|\boldsymbol{x}_i) \qquad (2)\end{aligned}$$

where $(\boldsymbol{x}_i,\boldsymbol{y}_i)$ refers to i-th instance in the training set. The additional L_2 regularization term can be introduced to avoid over-fitting. Here, we omit it for brevity.

The goal is to find optimal model parameters, i.e., Λ and Θ, which maximize the objective. We first consider the computation of gradients for Λ. Assuming $\Lambda = \langle\lambda_1, \lambda_2, \ldots, \lambda_N\rangle$, to learn the optimal feature weight values, we can calculate the

[4]They need to handle semantic nodes with arity 1 (which requires special constraints for properly defining $\mathcal{T}(\boldsymbol{x})$ (Lu, 2015)), and their semantic nodes are also assumed to convey semantic type information for guiding the expression tree construction process, while we do not need to consider them.

[5]The grammars are related to the word association patterns. The possible latent text-math trees are constructed based on such grammar rules.

gradient for each λ_k in Λ as:

$$\frac{\partial \mathcal{L}(\Lambda,\Theta)}{\partial \lambda_k} = \sum_i \sum_{\boldsymbol{t}} \boldsymbol{E}_{P_{\Lambda,\Theta}(\boldsymbol{t}|\boldsymbol{x}_i,\boldsymbol{y}_i)}[\phi_k(\boldsymbol{x}_i,\boldsymbol{y}_i,\boldsymbol{t})] - \sum_i \sum_{\boldsymbol{y},\boldsymbol{t}} \boldsymbol{E}_{P_{\Lambda,\Theta}(\boldsymbol{y},\boldsymbol{t}|\boldsymbol{x}_i)}[\phi_k(\boldsymbol{x}_i,\boldsymbol{y},\boldsymbol{t})] \quad (3)$$

where $\phi_k(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})$ is the number of occurrences for the k-th feature extracted from $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})$.

We then compute the gradient for the neural network parameters Θ. For an input word window x and a semantic unit y, the gradient is defined as:

$$\frac{\partial \mathcal{L}(\Lambda,\Theta)}{\partial \psi(x,y)} = \sum_i \sum_{\boldsymbol{t}} \boldsymbol{E}_{P_{\Lambda,\Theta}(\boldsymbol{t}|\boldsymbol{x}_i,\boldsymbol{y}_i)}[c(x,y,\boldsymbol{x}_i,\boldsymbol{y}_i,\boldsymbol{t})] - \sum_i \sum_{\boldsymbol{y},\boldsymbol{t}} \boldsymbol{E}_{P_{\Lambda,\Theta}(\boldsymbol{y},\boldsymbol{t}|\boldsymbol{x}_i)}[c(x,y,\boldsymbol{x}_i,\boldsymbol{y},\boldsymbol{t})] \quad (4)$$

The gradients (3,4) can be efficiently calculated by applying a generalized forward-backward algorithm, which allows us to conduct exact inference using the dynamic programming algorithm described in (Lu, 2014). Next, standard methods like gradient descent, L-BFGS (Liu and Nocedal, 1989) can be used to find optimal values for model parameters.

During decoding, the optimal equation tree $\boldsymbol{y}^*$ for a new input $\boldsymbol{x}$ can be calculated by:

$$\begin{aligned}\boldsymbol{y}^* &= \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) \\ &= \arg\max_{\boldsymbol{y}} \sum_{\boldsymbol{t}\in\mathcal{T}(\boldsymbol{x},\boldsymbol{y})} e^{F_{\Lambda,\Theta}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})} \\ &\approx \arg\max_{\boldsymbol{y},\boldsymbol{t}\in\mathcal{T}(\boldsymbol{x},\boldsymbol{y})} e^{F_{\Lambda,\Theta}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})} \quad (5)\end{aligned}$$

where $\mathcal{T}(\boldsymbol{x},\boldsymbol{y})$ refers to the set of all possible text-math trees that contain $\boldsymbol{x}$ and $\boldsymbol{y}$.

Instead of directly computing the summation over all possible latent text-math structures, we essentially replace the $\sum$ by the $\max$ operation inside the $\arg\max$. In other words, we first find the latent text-math tree $\boldsymbol{t}^*$ which yields the highest score and contains the input text $\boldsymbol{x}$. Then, the optimal expression tree $\boldsymbol{y}^*$ can be automatically extracted from $\boldsymbol{t}^*$.

An efficient dynamic programming based inference algorithm similar to the work of Lu (2014) was leveraged to find the optimal latent structure $\boldsymbol{t}^*$. We then obtain the optimal expression tree $\boldsymbol{y}^*$ from $\boldsymbol{t}^*$, which is the output of our system for the input problem text $\boldsymbol{x}$.

2.6 Comparisons with Roy et al. (2016)

It is worth noting that Roy et al. (2016) also proposed a system that maps text into an equation tree. Unlike this work that maps math problem texts into math expressions in an end-to-end fashion, Roy et al. (2016) designed three classifiers which sequentially make local decisions, namely identifying relevant numbers, recognizing possible variables and producing equations. They also require extra inverse operators to handle the non-commutative operation issues, which is not necessary for our model. Generating equations via a sequence of local classification decisions may propagate errors and even limit the ability to wholisticly understand the underlying semantics of problem texts which is important for predicting correct mathematical operations. In this work, we regard equation parsing problem as a structure prediction task that allows to parse the text to equations from a semantic parsing perspective. Moreover, `Text2Math` is capable to handle both tasks of equation parsing and arithmetic word problems, while the system of Roy et al. (2016) is specific to equation parsing.

3 Experiments

Datasets. Following prior works (Roy and Roth, 2015; Liang et al., 2018), we focus on two commonly-used benchmark datasets for arithmetic word problems, AI2 (Hosseini et al., 2014) and IL (Roy and Roth, 2015). We consider mathematical relations among numbers and calculate numerical values of the predicted expressions. For equation parsing, we also evaluate our model on the data released by (Roy et al., 2016). A predicted equation is regarded as a correct one if it is mathematically equivalent to the gold equation.

3.1 Empirical Results

Arithmetic Word Problem. Following previous work (Liang et al., 2018), we conduct 3-fold and 5-fold cross-validation on AI2 and IL, respectively, and report the accuracy scores, as shown in Table 2. Our method achieves competitive results on AI2 and IL. Overall, it performs better than previous systems in terms of average scores.

Ablation tests have been done to investigate the effectiveness of different components, such as POS tags, the lexicon and number relevance, as indicated by "-POS", "-LEX", "-ID" in Table 2. By eliminating POS tag features, we achieve

System		AI2	IL	Average
∗Liang et al. (2018) (Statistical)		81.5	81.0	81.25
∗Liang et al. (2018) (DNN)		69.8	70.6	70.20
∗Roy and Roth (2017)		76.2	71.0	73.60
Roy and Roth (2015)		78.0	73.9	75.95
Koncel-Kedziorski et al. (2015)		52.4	72.9	62.65
Roy et al. (2015)		-	52.7	-
Kushman et al. (2014)		64.0	73.7	68.85
Hosseini et al. (2014)		77.7	-	-
NON-NEURAL	Text2Math	85.8	80.4	83.10
	-POS	**86.0**	**81.0**	**83.50**
	-LEX	76.8	69.4	73.10
	-ID	75.4	78.1	76.75
NEURAL	$L = 0$	84.8	79.7	82.25
	$L = 1$	84.5	80.3	82.40
	$L = 2$	85.5	80.3	82.90
	$L = 3$	86.2	80.9	83.55
	$L = 4$	85.5	80.0	82.75
	$L = 5$	85.2	**81.4**	83.30
	$L = 6$	**86.5**	81.0	**83.75**

Table 2: **Arithmetic Word Problem**: Accuracy (%) on the two benchmark datasets. (-POS, -LEX, -ID mean that the model excludes POS tags feature, lexicon, or the number relevance feature.) ∗ indicates model uses prior knowledge, such as lexcion and inference rules.

new state-of-the-art results on two datasets, which shows POS tag features do not appear to be helpful in this case. Without using lexicon, the performance drops a lot as expected, but the results are still comparable with most previous systems. These figures demonstrate the effectiveness of the lexicon. It is worth noting that the work of Liang et al. (2018) that achieve previous state-of-the-art results leverage inference rules during the inference phase. Their approach can be regarded as a different way of using the lexicon similar to ours. We also consider the effects of neural features (see Sec. 2.4) with different window sizes $L \in \{0, 1, 2, 3, 4, 5, 6\}$. According to empirical results, a larger window size tends to give better results. One possible reason is that an arithmetic word problem often consists of several sentences, where a large word window is required to capture mathematical semantics.

Equation Parsing. We compare our model with previous work (Roy et al., 2016) on the equation parsing dataset, as shown in Table 3. Our method yields competitive results. Unlike the work of Roy et al. (2016), annotations of unknown variables are not required in our model. As reported in (Roy et al., 2016), they trained SPF (Artzi and Zettlemoyer, 2013a), a publicly available semantic parser, with sentence-equation pairs and a seed lexicon for mathematical terms. But it only obtained 3.1% accuracy. The result taken from (Roy et al., 2016) shows that it might be difficult for such a semantic parser in handling the equation

System		Equation
∗Roy et al. (2016) (Pipeline)		71.3
∗Roy et al. (2016) (Joint)		60.9
∗ SPF (Artzi and Zettlemoyer, 2013a)		3.1
NON-NEURAL	Text2Math	71.4
	-POS	69.1
	-LEX	71.4
	+G	**73.2**
	+G-LEX	**73.2**
NEURAL	$L = 0$	71.4
	$L = 1$	71.9
	$L = 2$	73.8
	$L = 3$	73.5
	$L = 4$	**74.5**
	$L = 5$	74.0
	$L = 6$	73.2

Table 3: **Equation Parsing**: Accuracy (%) on equation parsing dataset. -POS: without POS tag features; -LEX: without Lexicon; +G: with gold identification for numbers. ∗ indicates model uses lexicon. Result of SPF (Artzi and Zettlemoyer, 2013a) is taken from Roy et al. (2016).

parsing task even with a high precision lexicon. One possible reason is that mapping text into a math equation is essentially a structure prediction problem. Our model is capable to make guaranteed decisions from a structure prediction perspective. Different from arithmetic word problems, where numbers are explicitly given in the form of digits, some texts from equation parsing corpus describe numbers in string forms. Hence, a structured predictor is used to identify the numbers in the sentence, which achieves 95.3% accuracy. The identifications of numbers are taken as features. We also consider the gold label of numbers, indicated by (+G). The performance improves a lot, which shows that the accurate identification of numbers is necessary to in order to obtain a good performance. By removing POS tag features, there is a slight drop in accuracy. On the other hand, it is worth noting that even without the high precision lexicon, our model can still achieve new state-of-the-art accuracy in this task, while the previous work (Roy et al., 2016) always requires a high precision lexicon to boost performance. Incorporating neural features leads to new state-of-the-art accuracy of 74.5% when $L = 4$.

Expression Construction. In arithmetic word problems, the expression consists of several numbers only, exemplified by Problem 1 in Figure 1. In practice, an unknown variable X, representing the goal that the problem aims to calculate, can be appended to the expression to form an equation. We further investigate two constructions: appending the unknown variable X to the beginning or to the end of an expression. Results are listed in the first block of Table 4. For instance,

Variants	AI2	IL	Average
`Text2Math`	85.8	80.4	83.10
Prefix X	84.3	81.0	82.65
Suffix X	84.3	**81.2**	82.75
`Text2Math` + Inverse	**86.3**	80.1	**83.20**
Prefix X + Inverse	84.0	79.9	80.45
Suffix X + Inverse	84.0	80.5	82.25

Table 4: Performance of different constructions for expression of arithmetic word problems and the effects of incorporating inverse operators.

two new constructions of the running example are $X = (29 + (16 + 20))$ as indicated by "Prefix X", and $(29 + (16 + 20)) = X$ reported as "Suffix X". It is interesting that including an X and its position influences the performance. Overall, excluding X works the best which is adopted in this work.

Inverse Operators. As we discussed in Sec. 2.3, one distinct advantage of our approach, as compared to others, is that we do not need inverse operators, such as "$-_r$" and "$\div_r$". Our designed word association patterns are capable to handle the reordering issue. Here, we consider model variants by introducing two inverse expression tree nodes, SUB_r and DIV_r, to represent "$-_r$" and "$\div_r$", respectively. Empirical results, reported in the second block in Table 4, show that `Text2Math` (without including inverse operators) can obtain comparative results compared to the model variants with inverse operators. These results confirm that our model does not require additional knowledge of the semantics of operands, which is a unique property of our approach.

3.2 Qualitative Analysis

Output Comparisons. Equation parsing is more challenging than arithmetic word problems, since it requires generating unknown variables mapped to phrases residing in the text. We analyze output of this task to investigate the source that leads to better performance. Comparing predictions made by Pipeline (Roy et al., 2016) and our approach, we found that `Text2Math` can better capture the meaning of the problem text. We illustrate two examples in Table 5. The Pipeline approach fails to capture the meaning of "*rises to 36% from 3.4%*" which implies subtraction of two numbers, while our model is capable to capture such knowledge. In the second example, Pipeline misunderstands the meaning of "*five more than three*", although it seems correct in a local context. However, an equation should be mapped from the complete sentence that captures mathematical relations in a global perspective. Our model holds such a capability and makes more guaranteed predictions, which proves the efficacy of solving math problems from a structure prediction perspective.

Input:	*Japan January jobless rate rises to* 3.6% *from* 3.4%.
Gold:	$X_1 = 0.036 - 0.034$
Pipeline:	$X_1 \times 0.036 = 0.034$
`Text2Math`:	$X_1 = 0.036 - 0.034$
Input:	*The number of baseball cards he has is five more than three times the number of football cards.*
Gold:	$X_1 = 5 + (3 \times X_2)$
Pipeline:	$X_1 \times X_2 = 5 - 3$
`Text2Math`:	$X_1 = 5 + (3 \times X_2)$

Table 5: Comparison between predictions made by the previous state-of-the-art system (Roy et al., 2016) (denoted as Pipeline) and `Text2Math`.

Robustness. To further investigate the property of our model, we studied outputs. We found that our method is able to conduct self-correction. Exemplified by Example 3 in Table 6, considering the sentence *"Germany's DAX opens 0.7% lower at 18,842."* with annotated equation $X_1 + (0.007 \times X_1) = 18,842$, the prediction made by our method is $X_1 - (0.007 \times X_1) = 18,842$. It can be seen that the prediction made by our method is supposed to be the correct one, while the annotation is actually wrong. To make a fair comparison with previous works, we did not count such cases as correct during evaluation, which implies that accuracy reported in Table 3 is in fact higher.

Error Analysis. For arithmetic word problem, it is interesting that the operand of two operands should be addition/subtraction (multiplication/division), while the prediction is subtraction/addition (division/multiplication). Consider Example 4 and 5 in Table 6. Descriptions of such two problems share many words, such as *each*, *how many*, *there are*, etc. Slight difference in problem descriptions may lead to different results, which makes it a challenge.

As for equation parsing, the work of Roy et al. (2016) requires annotations on which phrases should be mapped to unknowns during the training phase. However, such supervised knowledge is not required for our method. In our setup, we did not make hard constraint that each prediction must contain one or two variables. Therefore, missing or redundant variables appearing in the predicted equations are one of the major error sources. Example 6 and 7 from Table 6 illustrate such cases. On the other hand, lack of professional background information also leads to miss-

Example 3:	*Germany's DAX opens 0.7% lower at 10,842.*
Gold:	$X_1 + (0.007 \times X_1) = 10842$
`Text2Math`:	$X_1 - (0.007 \times X_1) = 10842$
Example 4:	*Each child has 5 bottle caps. If there are 9 children, how many bottle caps are there in total?*
Gold:	5×9
`Text2Math`:	$5 \div 9$
Example 5:	*The school is planning a field trip. There are 14 students and 2 seats on each school bus. How many buses are needed to take the trip?*
Gold:	$14 \div 2$
`Text2Math`:	14×2
Example 6:	*530 pesos can buy 4 kilograms of fish and 2 kilograms of pork.*
Gold:	$530 = (4 \times X_1) + (2 \times X_2)$
`Text2Math`:	$530 \times X_3 = (4 \times X_1) + (2 \times X_2)$
Example 7:	*Flying with the wind , a bird was able to make 150 kilometers per hour.*
Gold:	$X_1 + X_2 = 150$
`Text2Math`:	$X_1 = 150$

Table 6: Examples with wrong predictions. Gold denotes the annotated correct equations and `Text2Math` refers to output equations generated by our method.

ing variables. Consider Example 6. Without world knowledge, it might be difficult for the algorithm to recognize that "*Flying with the wind*" implies the speed of the wind which should be considered as a variable of the equation.

4 Related Work

Math Word Problems. Mukherjee and Garain (2008) surveyed related approaches to this task in literature. Hosseini et al. (2014); Mitra and Baral (2016) solved the task by categorizing verbs or problems. The first method that can handle general arithmetic problems with multiple steps was proposed by Roy and Roth (2015), which was further extended by introducing (Roy and Roth, 2017, 2018). Zou and Lu (2019b,c) is the first work that proposed a sequence labelling approach to solving arithmetic word problems, which focuses on addition-subtraction word problems. Other systems include semantic parsing based approaches (Liang et al., 2018) and neural methods (Wang et al., 2017, 2018a,b). Unlike arithmetic word problems, the goal of algebra word problems is to map the text to an equation set (Kushman et al., 2014; Shi et al., 2015). Other types of problems have also been investigated, including probability problems (Dries et al., 2017), logic puzzle problems (Mitra and Baral, 2015; Chesani et al., 2017) and geometry problems (Seo et al., 2014, 2015). Besides the benchmark datasets used in this work, other popular datasets include Dolphin18K (Shi et al., 2015) and AQuA (Ling et al., 2017) for algebra word problems which are not the focus in this work. Roy et al. (2016) first proposed the Equation Parsing task and designed a pipeline method with three structured predictors.

Semantic Parsing. Another line of related works is semantic parsing (Wong and Mooney, 2006; Zettlemoyer and Collins, 2007; Kwiatkowksi et al., 2010; Liang et al., 2011; Dong and Lapata, 2018; Zou and Lu, 2018), which aims to map sentences into logic forms, including CCG-based lambda calculus expressions (Zettlemoyer and Collins, 2007; Kwiatkowksi et al., 2010; Artzi and Zettlemoyer, 2013b; Dong and Lapata, 2016), FunQL (Kate et al., 2005; Wong and Mooney, 2006; Jones et al., 2012), lambda-DCS (Liang et al., 2011; Berant et al., 2013; Jia and Liang, 2016), graph queries (Harris et al., 2013; Holzschuher and Peinl, 2013) and SQL (Yin et al., 2015; Sun et al., 2018). In this work, we adopt a text-math semantic representation encoding words and the expression tree.

5 Conclusion

In this work, we propose a unified structured prediction approach, `Text2Math`, to solving both arithmetic word problems and equation parsing tasks. We leverage a novel joint representation to automatically learn the correspondence between words and math expressions which reflects semantic closeness. Different from many existing models, `Text2Math` is agnostic of the semantics of operands and learns to map from text to math expressions in an end-to-end manner based on a data-driven approach. Experiments demonstrate the efficacy of our model. In the future, we would like to investigate how such an approach can be applied to more complicated math word problems, like algebra word problems where a problem usually maps to an equation set. Another interesting direction is to investigate how to incorporate world knowledge into the graph-based approach to boost the performance.

Acknowledgments

We would like to thank the three anonymous reviewers for their thoughtful and constructive comments. This work is supported by Singapore Ministry of Education Academic Research Fund (AcRF) Tier 2 Project MOE2017-T2-1-156.

References

Yoav Artzi and Luke Zettlemoyer. 2013a. Uw spf: The university of washington semantic parsing framework. *arXiv preprint arXiv:1311.3011*.

Yoav Artzi and Luke Zettlemoyer. 2013b. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of EMNLP*.

Daniel G Bobrow. 1964. Natural language input for a computer problem solving system. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC.

Federico Chesani, Paola Mello, and Michela Milano. 2017. Solving mathematical puzzles: A challenging competition for ai. *AI Magazine*, 38.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of ACL*.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of ACL*.

Anton Dries, Angelika Kimmig, Jesse Davis, Vaishak Belle, and Luc De Raedt. 2017. Solving probability problems in natural language. In *Proceedings of IJCAI*.

Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. 2013. Sparql 1.1 query language. *W3C Recommendation*, 21(10):778.

Florian Holzschuher and René Peinl. 2013. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of EMNLP*.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of ACL*.

Bevan Jones, Mark Johnson, and Sharon Goldwater. 2012. Semantic parsing with bayesian tree transducers. In *Proceedings of ACL*.

Rohit J Kate, Yuk Wah Wong, and Raymond J Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of AAAI*.

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 ACL*.

Tom Kwiatkowksi, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of EMNLP*.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of NAACL-HLT*.

Chao-Chun Liang, Yu-Shiang Wong, Yi-Chung Lin, and Keh-Yih Su. 2018. A meaning-based statistical english math word problem solver. In *Proceedings of NAACL*.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of ACL*.

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 ACL*.

Dong C Liu and Jorge Nocedal. 1989. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45.

Wei Lu. 2014. Semantic parsing with relaxed hybrid trees. In *Proceedings of EMNLP*.

Wei Lu. 2015. Constrained semantic forests for improved discriminative semantic parsing. In *Proceedings of ACL*.

Wei Lu, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. 2008. A generative model for parsing natural language to meaning representations. In *Proceedings of EMNLP*.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of ACL*.

Arindam Mitra and Chitta Baral. 2015. Learning to automatically solve logic grid puzzles. In *Proceedings of EMNLP*.

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

Anirban Mukherjee and Utpal Garain. 2008. A review of methods for automatic understanding of natural language mathematical problems. *Artificial Intelligence Review*, 29(2).

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of EMNLP*.

Subhro Roy and Dan Roth. 2017. Unit dependency graph and its application to arithmetic word problem solving. In *Proceedings of AAAI*.

Subhro Roy and Dan Roth. 2018. Mapping to declarative knowledge for word problem solving. *Transactions of the Association of Computational Linguistics*, 6:159–172.

Subhro Roy, Shyam Upadhyay, and Dan Roth. 2016. Equation parsing: Mapping sentences to grounded equations. In *Proceedings of EMNLP*.

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

Min Joon Seo, Hannaneh Hajishirzi, Ali Farhadi, and Oren Etzioni. 2014. Diagram understanding in geometry questions. In *Proceedings of AAAI*.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of EMNLP*.

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

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware sql generation. In *Proceedings of ACL*.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to an expression tree. In *Proceedings of EMNLP*.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018b. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Proceedings of AAAI*.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of EMNLP*.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of NAACL*.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2015. Neural enquirer: Learning to query tables with natural language. *arXiv preprint arXiv:1512.00965*.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of EMNLP-CoNLL*.

Yanyan Zou and Wei Lu. 2018. Learning cross-lingual distributed logical representations for semantic parsing. In *Proceedings of ACL*.

Yanyan Zou and Wei Lu. 2019a. Joint detection and location of english puns. In *Proceedings of ACL*.

Yanyan Zou and Wei Lu. 2019b. Quantity tagger: A latent-variable sequence labeling approach tosolving addition-subtraction word problems. In *Proceedings of ACL*.

Yanyan Zou and Wei Lu. 2019c. Supplementary material for quantity tagger: A latent-variable sequence labeling approach tosolving addition-subtraction word problems. In *Proceedings of ACL*.