Understanding Mathematical Relations in Text

Abstract

Identifying mathematical relations expressed in text is essential to understanding a broad range of natural language text from election reports, to financial news, to sport commentaries to mathematical word problems. The task necessitates necessitates identifying relevant entities in the text as well as mapping the text to a corresponding mathematical equation involving these variables. This paper focuses on identifying and understanding mathematical relations described within a single sentence. We introduce an equation parse for sentences, which naturally handles composition and nesting of mathematical expressions, and directly generates the necessary equation expressed in the text. Our structured prediction formulation jointly grounds variables and generates an equation parse for a sentence. We demonstrate that extracting relations via equation parses improves accuracy over template based approaches.

1 Introduction

Understanding text often involves reasoning with respect to quantities mentioned in it. Understanding a news article statement that *"Emanuel's campaign contributions total three times those of his opponents put together"* requires identifying relevant entities, the mathematical relations expressed among them in text, and how to compose them. Similarly, an elementary school math word problem such as: *"The difference of twice a number and 3 is five more than thrice another number."* requires realizing that we deal with the relations between two numbers, knowing the meaning of *"difference"* and composing the right equation – *"five"* needs to be added only after three is multiplied with the number.

The first step in this process involves identifying relevant variables and *grounding* them in text, that is, detecting a span of text which describes what the variable represents. Grounding is necessary to support reasoning, as well as analysis across multiple sentences. However, the text often does not mention *"variables"* explicitly. For example, the sentence *"Flying with the wind , a bird was able to make 150 kilometers per hour."* describes a mathematical relation between the speed of bird and the speed of wind, without mentioning "speed" explicitly. There is also ambiguity regarding mention choice while grounding variables. For example, in the following sentence-equation pair

Example 1	
Sentence	*City Rentals rent an intermediate-size car for 18.95 dollars plus 0.21 per mile.*
Equation	$V_1 = 18.95 + 0.21 * V_2$

the variable V_1 could be grounded to *"City Rentals"*, *"rent"*, or *"intermediate-size car"*. The later part of the text might mention another rental company, making *"City Rentals"* the prime entity of this sentence. Again, the text might go on to discuss *"City Rental full size"* cars, making *"intermediate cars"* more important.

In this paper, we introduce the *Equation Parsing* task: given a sentence describing a mathematical relation between entities, the goal is to map it to an equation representing the relation, and to determine what the variables the text refers to. We propose a novel approach to address the problem of equation parsing, involving the identification and grounding of variables and predicting the corresponding equation.

Example 2	
Sentence	*If 10 is added to two numbers, the first one will be 5 more than thrice the second.*
Equation	$V_1 + 10 = 5 + 3 * (V_2 + 10)$
Grounding	V_1 = {"the first one", "two numbers"} V_2 = {"the second", "two numbers"}

We propose a novel approach to address the problem of equation parsing, involving the identification and grounding of variables and predicting the corresponding equation. First, we detect and normalize all quantities in the sentence, and prune the irrelevant quantities, which do not take part in the relation. Next we develop a structured prediction model to jointly predict variable grounding and the corresponding equation. As seen in example 2, there are often multiple spans of text which can be valid grounding of a variable. Here, V_1 can be correctly grounded to both *"the first one"* or *"both numbers"*. Since exact segmentation of mentions can be ambiguous, we ground variables to tokens in the sentence, which are either nouns, verbs or adjectives. A grounding is said to be valid if the system grounds to a token within a gold grounded span. Since there can be multiple valid groundings for a problem, we model the best grounding to be latent, and let the model figure out which grounding is best for it, to generate equations.

Our inference algorithm relies heavily on the idea of an equation parse of a sentence. The parse can effectively capture compositionality, can directly generate the necessary equation from text, and it provides a way to explore the space of all equations efficiently. Our model also leverages a small knowledge base of mathematical concepts as a source of domain knowledge.

We develop and annotate datasets for evaluation and show that our method can handle the equation generation and grounding task quite well.

The next section presents some related work on mathematical reasoning and automatic problem solvers. We then present our equation parse representation, describe our algorithm to generate the parse from text, and conclude with experimental evaluation.

2 Related Work

From the NLP perspective, our work is related to two main areas of research : semantic parsing and automatic math word problem solving.

Automatic Math Word Problem Solvers Mathematical relation extraction is an essential component of math word problem solvers. However, most problem solvers restrict the space of equations that can be generated, either by using equation templates or making assumptions on the problem type. (Kushman et al., 2014) solves simultaneous equation math word problems, by inducing equation templates and aligning numbers mentioned in the problem to template slots. Unlike our work, this approach does not ground variables and cannot handle compositional expressions. Moreover, our Equation Parsing approach explores the entire space of equations, and does not assume that the equation templates for the test data has been seen in the training examples.

Other approaches to automatic math problem solving (Roy et al., 2015; Hosseini et al., 2014) focus on simpler elementary school problems, and hence has a few number of possible output equations. (Roy et al., 2015) focuses on questions which can be answered by choosing two numbers from the problem, and applying one of four basic operations on it. (Hosseini et al., 2014) looks at addition and subtraction problems, which they model as state transitions. Both these approaches search a small space of equations, and can handle only simple syntax.

Semantic Parsing There has been a lot of work in mapping natural language text to formal meaning representation. Some (Zettlemoyer and Collins, 2005; Ge and Mooney, 2006) assume access to manually annotated formal representation for every sentence, whereas others (Eisenstein et al., 2009; Kate and Mooney, 2007; Goldwasser and Roth, 2011; Clarke et al., 2010) reduce the amount of supervision using domain knowledge. Reinforcement learning has also been used (Branavan et al., 2009; Branavan et al., 2010; Vogel and Jurafsky, 2010) to automatically understand and execute instructions.

Some of these approaches depend on a CCG-like parse of the entire sentence, and all restrict their domain to simple short sentences. In constrast, our equation parse is only constructed on quantities mentioned in text, and token spans which represent variable entities for the output equation. This allows us to efficiently parse longer compositional sentences.

From the ML perspective, our work builds on earlier work in constrained latent structured prediction (Chang et al., 2010; Samdani et al., 2012; Björkelund and Kuhn, 2014), and we provide details on it in Sec.4.3.1.

3 Equation Representation

In this section, we introduce an equation parse for a sentence. An equation parse of a sentence S is a 3-tuple (T, E, M), where T represents a set of *triggers* extracted from S, E represents an *equation tree* formed with the set T as leaves, and M represents the *modifiers* along with the manner in which they modify a variable. We now describe these terms in detail.

Trigger Given a sentence S mentioning a mathematical relation, a trigger can either be a *quantity trigger* expressed in S, or *variable trigger* which is a span of text in S corresponding to a variable. A *quantity trigger* is a tuple (q, s), where q is the numeric value of the quantity mentioned in text, and s is the span of text from the sentence S which includes the quantity. A *variable trigger* is a tuple (l, s), where l represents the name of the variable, and s, as before, represents the span of text representing the variable. For example, for the sentence in Fig 1, the spans "10", "5", and "thrice" generate quantity triggers, whereas "the first one" and "the second" generate variable triggers, with labels V_1 and V_2 respectively. The same span of text can generate multiple triggers. For example, the span "two numbers" represent a quantity, but can be a valid grounding for both the variables in text. Therefore, it generates 3 triggers - (2.0, "two numbers"), (V_1, "two numbers"), and (V_2, "two numbers").

Equation Tree An equation tree is a binary tree whose leaves represent triggers, and internal nodes (except the root) are labeled with one of the following operations – *addition, subtraction, multiplication, division*. In addition, for nodes which are labeled with subtraction or division, we maintain a separate variable to determine order of its children. The root of the tree is always labeled with the operation *equal*. In Fig 1, the equation tree is constructed on the triggers generated by "the first one", "5", "thrice", and "the second".

An equation tree also gives a natural representation for an equation. Each node n in an equation tree represents an expression $\textsc{Expr}(n)$, and the label of the parent node determines how the expressions of its children are to be composed to construct its own expression. Let us denote the operation label for a non-leaf node n to be $\odot(n)$, where $\odot(n) \in \{+, -, *, /, =\}$ and the order of a node n's children by $\textsc{Order}(n)$, which takes values lr, rl. For trigger node n, the expression $\textsc{Expr}(n)$ represents the variable label, if n is a variable trigger, and the numeric value of the quantity, if it is a quantity trigger. Finally, we use $lc(n)$ and $rc(n)$ to represent the left and right child of node n, respectively. The generation of the mathematical expression that corresponds to the parse tree can be done recursively as follows:

For all non-leaf node n of an equation tree, we have

$$\textsc{Expr}(n) = \begin{cases} \textsc{Expr}(lc(n)) \odot(n)\, \textsc{Expr}(rc(n)) & \text{if } \odot(n) \in \{+, *, =\} \\ \textsc{Expr}(lc(n)) \odot(n)\, \textsc{Expr}(rc(n)) & \text{if } \odot(n) \in \{-, /\} \wedge \\ & \textsc{Order}(n) = lr \\ \textsc{Expr}(rc(n)) \odot(n)\, \textsc{Expr}(lc(n)) & \text{if } \odot(n) \in \{-, /\} \wedge \\ & \textsc{Order}(n) = rl \end{cases} \tag{1}$$

Referring to the equation tree in 1, the node marked "*" represents $3 * V_2$, the node marked "+" represents $5 + 3 * V_2$ and, finally the root represents $V_1 = 5 + 3 * V_2$.

The leaves in an equation tree form an ordered list of triggers, that is, the order of the leaves used in an equation tree is the same as the order of the corresponding triggers in the text. Hence equation trees implicitly provide a constituent-parse-like structure for the sentence. The underlying assumption we make when constructing equation trees is that if there is a need to perform an operation between two

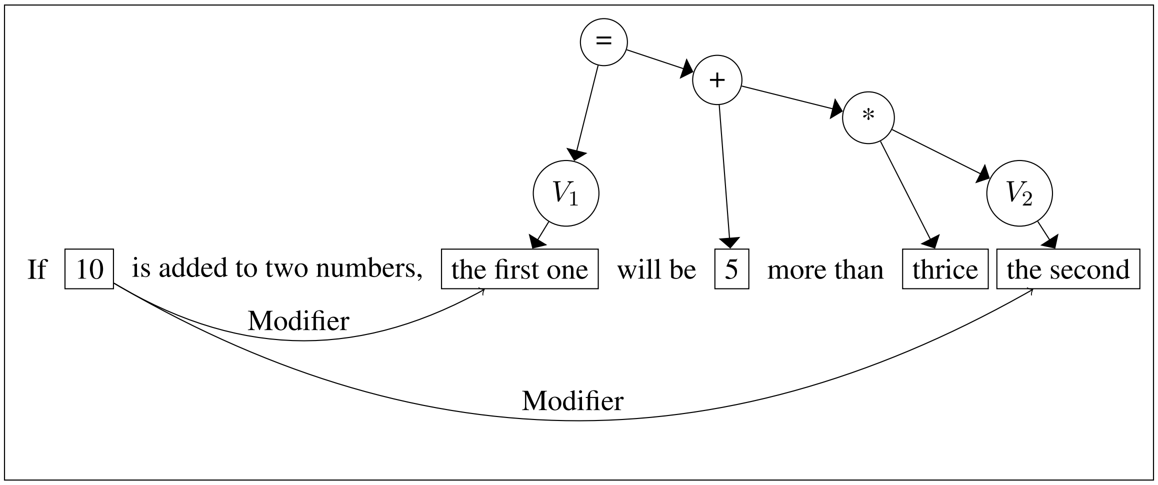

Figure 1: Example of an Equation parse

expressions, then those expressions are mentioned consecutively in the text; that is, no other expression from the equation is mentioned between them. For example, in Fig 1, "thrice" and "the second" are found next to each other, "5" and "thrice the second" do not have any trigger mentioned in the span between them.

Although we found that the "consecutive triggers" assumption described above holds in a large fraction of the texts we studied, there are cases when it is not supported. Certain numbers in the text modify both variables in the same manner, and they might not be mentioned in the vicinity of grounded variable spans. In our running example, the number "10" needs to be added to both variables, but 10 and V_2 clearly do not satisfy the consecutive trigger assumption. In order to support such variable modification, we introduce the concept of modifiers, which we define next.

Modifiers Modifiers are quantities or expressions mentioned in text, which modify both variables of an equation in a similar way. For example, in "When you double two numbers, their sum is 30", the term "double" determines that both variables needs to be multiplied by 2. Similarly, in "In 5 years, John will be twice as old as his brother.", "10" needs to be added to the ages of both John and his brother, before we can use the relation expressed in the later part of the text. We also associate a label with each modifier to know how it is changing the variables in the sentence. For example, "double" in the first example was a multiplicative modifier, whereas "10" in the second example is an additive modifier. In our problem, we restrict modifiers to comprise only single quantity triggers and assume that a modifier modifies all its relevant variables in the same way. We believe that these assumptions do not restrict our coverage. There are six possible ways it can modify the variables in the problem: addition, subtraction (both orders), multiplication and division (both orders).

Given an equation parse of a sentence, the equation represented by it is the expression generated by the root of T (following Equation 1), followed by a modification of the variables by the modifier M.

4 Predicting Equations

We now describe the algorithm for identifying the terms and expressions of the equation from a sentence, and how we can construct a complete equation using them. We use the following two modules in a pipeline:

1. **Quantity Detector** : This module looks at a quantity and decides whether it is relevant to the equation formed from the sentence. Also, it determines whether the quantity acts as a modifier in a sentence, and how it modifies the corresponding variables.

2. **Equation Parse Generator** : This module jointly grounds variables and generates an equation parse.

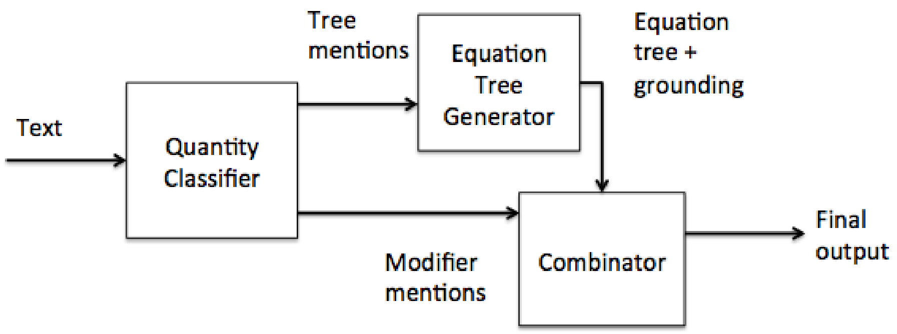

Figure 2: Pipeline of our system

We discuss the details of these modules in the following subsections.

4.1 Quantities Extraction and Normalization

The natural first step towards identifying the terms participating in the equation is to find the quantities mentioned in the sentence. We use the Illinois Quantifier package (Roy et al., 2015) to detect all the quantity mentions. The quantifier extracts mention spans from the text, where each span identifies a quantity. Each such span is represented as a triple (v, u, c) where v denotes the value, u denotes the unit and c denotes the change in the quantity (if any).

The quantity mentions identified fall into one of the following three categories:

1. **Tree Mention** - Mentions that are part of the final equation tree. In Fig 1, "5" is a tree mention.

2. **Spurious Mentions** - Quantities which are not used in the equation. In Fig 1, the quantity "two numbers" is a spurious mention, as the magnitude 2 plays no role in the final equation. This means that the quantity triggers arising from this mention will not be part of the equation tree. Variable triggers generated using the same span can still be relevant to the equation tree.

3. **Modifier Mentions** - Quantities which play the role of modifiers. In Fig 1, "10" is a modifier mention, as it modifies both the variables in the resulting equation.

4.2 Quantity Classifier

We use a multiclass classifier trained on a small number of examples from our annotated data set, to determine the quantity type. Moreover, for the case of modifier mentions, there is a need to determine also how the modifier changes the variables in the problem. Since the number of outcomes is small, we predict these properties jointly with the quantity type. Specifically, we perform a multicalss classification to classify each quantity to one of Tree mention, Spurious mention, or a modifier conjoined with a modification manner.

The features used for this module are as follows :

1. **Lexical features** : Unigrams and bigrams generated from a window around the quantity.

2. **POS Tags** : Part of speech of tags of neighborhood tokens, conjoined with lexical features.

3. **Quantity Features** : Units of the quantity, whether number associated with quantity is one or two.

4.3 Equation Tree Generator

This module receives the Tree mentions detected by the Quantity classifier, and predicts an equation tree along with grounding of the variables. We formulate this problem as a structured prediction task. The input structure x consists of the sentence S, as well as the tree mentions detected by the Quantity classifier. As we pointed out earlier grounding of variables might be ambiguous, since multiple groundings might be valid for any given variable. Therefore, we model the grounding of variables as a latent variable h, and our system gets to decide which of the valid groundings it will use. The output structure y now comprises of an equation tree, whose leaves are formed by the union of the tree mentions in x, and the grounded variable mentions h.

We learn a scoring function to decide, given x, the best equation tree y^* and the best grounding of variables h^*. We assume that our scoring function is of the following form :

$$f_w(y, h) = w^T \phi(x, y, h)$$

where $\phi(x, y, h)$ is a feature vector extracted from x, h, y. The prediction (y^*, h^*) is then chosen as follows:

$$(y^*, h^*) = \arg \max_{(y,h) \in \mathcal{H} \times \mathcal{Y}} f_w(y, h)$$

where $\mathcal{H}$ represents the set of all possible groundings of the variables, and $\mathcal{Y}$ represents the set of all possible parse trees.

4.3.1 Learning

We assume access to N training examples of the form : $(x_1, H_1, y_1), (x_2, H_2, y_2), \ldots, (x_N, H_N, y_N)$, where each H_i is a set of valid groundings possible for the variables in the equation tree y_i. This is different from most latent variable structured learning problems in that the domain of the hidden variable is explicitly constrained by H_i. We use a modified latent structural SVM to learn the weight vector w. The details of the algorithm are in Algorithm 1.

Algorithm 1 Constrained Latent Structural SVM

```
Input: Training data $T = \{(x_1, H_1, y_1), (x_2, H_2, y_2), \ldots, (x_N, H_N, y_N)\}$
Output: Trained weight vector $w$
 1: $w \leftarrow w_0$
 2: repeat
 3:     $T' \leftarrow \emptyset$
 4:     for all $(x_i, y_i, H_i) \in T$ do
 5:         $h_i^* \leftarrow \arg\max_{h \in H_i} w^T \phi(x_i, y_i, h)$
 6:         $T' \leftarrow T \cup \{x_i, (y_i, h_i^*)\}$
 7:     end for
 8:     Update $w$ by running standard Structural SVM algorithm on $T'$
 9: until convergence
10: return $w$
```

The distinction from standard latent structural SVM is in the latent variable completion problem, stated in line 5 of Algorithm 1. In order to get the best latent variable assignment h_i^* for input tuple (x_i, y_i, H_i), we search only inside H_i, instead of all possible values of h_i. A similar formulation can be found in (Zettlemoyer and Collins, 2007; Björkelund and Kuhn, 2014), but in both the cases, the set H_i can be computed deterministically from y_i. In our case, we depend on explicit annotations for H_i.

4.3.2 Inference

We need to solve two inference algorithms to run the latent SSVM algorithm. The first is the label completion problem, where we need to compute the best grounding h^* given gold input x_i and gold equation tree y_i. We also have access to the set of valid groundings H_i, which reduces the search space significantly, and allows us to do exhaustive search. Specifically, we look at all nouns, verbs and adjectives, which are valid groundings of the variables used in y_i, and choose the one with the best score.

The second inference algorithm is the one used in the standard structural SVM algorithm. Here we need to compute

$$(y^*, h^*) = \arg \max_{(y,h) \in \mathcal{Y} \times \mathcal{H}} w^T \phi(x, y, h)$$

Given an input sentence and the corresponding tree mentions, we need to find the best grounding of the variables h^* as well as the best equation tree y^*. Here we do not have access to candidate gold groundings, and exhaustive search will entail looking at exponential number of possibilities. Therefore, we use a beam search procedure to approximately find the best grounding and equation tree. First we enumerate all possible groundings for both single variable and two variable problems, and initialize our beam with top-k best groundings. Once the groundings are available, we know what the leaves of the equation tree are. Hence for each grounding in our beam, we now employ a CKY algorithm to construct the trees in a bottom up fashion. Since the independence assumptions needed for CKY are not valid here, we continue to use beam search, maintaining the best k parses at each cell of the CKY table.

4.3.3 Features

The features used for this module aim to capture the characteristics of variable grounding and the equation tree generation. They can be broadly categorized into two groups:

1. **Variable Grounding Features**
   (a) **Lexical Features** We use, as features, tokens to which variables have been grounded. Certain words and phrases are frequently associated with variables, such as *"both numbers"*, *"one of them"*, and this helps us capture that.
   (b) **Neighborhood Features** Unigrams and bigrams extracted from a window around tokens, which are grounded to variables. Often, units of numbers represent variables, and having a neighboring number can be a strong indication for being a valid grounding. For example, if a math problem mentions "4 apples", it is likely that some property (count, price, etc.) of *"apples"* will be a variable in the problem.

2. **Equation Tree Features** : We take advantage of the constituent-parse-like structure of an equation parse, to define expressive features. For any node n of an equation parse tree, we define Left(n) to be the position of the leftmost trigger in the sentence, which lies in the subtree spanned by node n. Likewise, we define, Right(n) to be the rightmost trigger in the subtree of n. The node n can be seen as representing the segment of text spanning from Left(n) to Right(n), which we define to be Span(n).
   (a) **Span Features** : We conjoin the operation of a node, with the tokens lying between the spans of its children. We observe that when nodes n_1 and n_2 are being combined in an equation parse to form n, the operation at n is mostly defined by the text in between Span(n_1) and Span(n_2). For example, *"The stock prices increased 10$ to reach 340$"*, the term "increased" lies between "stock prices" and "340$".
   (b) **Left Lexical Features** : We add lexical features from the left of Left(n), to capture math terms like "sum", "difference", etc. For example, in *"The difference of twice a number and another number is 5."*, the term "difference" dictates the operation between "twice a number" and "another number".

   In addition, we add features based on quantity units, and whether a smaller number is being subtracted from a greater number.

4.4 Combinator

This module receives the modifiers extracted by the Quantity Classifier, the equation tree generated by Equation Tree Generator, and changes the variables in the equation tree, according to the type of the modifier. It is a rule based system to combine the output of the other modules to generate the final equation.

5 Experimental Results

In this section, we describe our dataset, and evaluate our system's performance on grounding variables and generating equations from text. We also compare our results against a template based baseline.

Dataset We created a new dataset consisting of 400 sentences extracted from algebra word problems and financial news headlines. For algebra word problems, we used a recently released MIT dataset (Kushman et al., 2014), and two high school mathematics textbooks, Elementary Algebra (College of Redwoods) (Red,) and Beginning and Intermediate Algebra (Tyler Wallace)(Wal,). Financial news headlines from The Latest News feed of MarketWatch (Mar,), over the month of February, 2014, was used a source of financial news headlines. Some statistics of the dataset are provided in Table 1

Source	#Sentences
MIT	250
Redwoods	25
Wallace	50
MarketWatch	75

Table 1: Statistics of dataset

Sentences were chosen such that each sentence has information describing exactly one mathematical relation and all quantities needed in the equation should be mentioned in the sentence. Each sentence was annotated with the final equation. For each variable in the equation, we annotated spans of text which best describe what the variable represents. Maximal span of tokens which describe or refer to the variable in the sentence were annotated. For example, in the sentence, *"City Rentals rent an intermediate-size car for 18.95 dollars plus 0.21 per mile."*, the phrase *"City Rentals rent an intermediate-size car"* was annotated. We also allow discontiguous spans, as in Example 2, we annotate both *"two numbers"* and *"the second"* as representing variable V_2. The inter-annotator agreement was 0.76.

Quantity Classifier Table 2 shows the performance of the Quantity Classifier in classifying quantity mentions in text as Tree Mention, Spurious Mention or Modifier conjoined with modification manner. We report two accuracies, one is the fraction of quantities it classified correctly (Per quantity accuracy), and second is the fraction of problems in which it correctly classified all the mentioned quantities (Per problem accuracy). The accuracy was reported using 5-fold cross-validation.

Type	Accuracy (%)
Per Quantity	93.64
Per Problem	87.88

Table 2: Statistics of dataset

Equation Tree Generator In this section, we evaluate the performance of the equation tree generator, which jointly grounds variables and generates equation trees. We compute two scores - Equation Accuracy, which is the fraction of equations it correctly predicted, irrespective of the grounding, and Equation+Grounding Accuracy, which is the fraction of sentences for which it correctly grounded the variables and generated the correct equation.

We compare our system EQTREEGEN against a template based baseline algorithm TEMPLATE, similar to the one used in (Kushman et al., 2014). TEMPLATE is identical to our system, except that instead of using equation trees to perform inference, it learns to align numbers and strings to slots of equation templates. The beam search in TEMPLATE iteratively aligns slots in templates to tokens or numbers from the text. We used the same feature set for both the methods. We further augmented the feature set of TEMPLATE with a bias term for the template used, and unigrams and bigrams conjoined with the template used.

We also evaluate the performance of the equation tree generation systems based on gold and predicted Tree mentions. Overall, we evaluate the performance of four systems :

1. GOLD+TEMPLATE : Uses gold Tree mentions with TEMPLATE.

2. GOLD+EQTREEGEN : Uses gold Tree mentions with EQTREEGEN.

3. PREDICT+TEMPLATE : Uses predicted Tree mentions with TEMPLATE.

4. PREDICT+EQTREEGEN : Uses predicted Tree mentions with EQTREEGEN.

System	Equation (%)	Grounding+Equation (%)
GOLD+TEMPLATE	54.5	14.5
GOLD+EQTREEGEN	74.6	66.4
PREDICT+TEMPLATE	49.5	14.25
PREDICT+EQTREEGEN	71.0	60.75

Table 3: Statistics of dataset

Fig 3 shows the performance of TEMPLATE and EQTREEGEN on predicting correct groundings of variables and the corresponding equation, with gold and predicted Tree Mentions. TEMPLATE can predict equations reasonably well, but fails to capture the understanding of what the variables represent.

Qualitative Analysis Since the quantity classifier takes decisions independently for each quantity in a sentence, it makes mistakes where decisions of neighboring quantities affect each other. For example, it does not detect that the number "20" is spurious, for the relation in *"A bank teller has 54 5-dollar and 20-dollar bills in her cash drawer"*. The decision for the quantity "20-dollar bills" could have been inferred from the decision for "5-dollar", since they are used in a similar context. The equation tree generator find difficulty extracting relations from sentences like "There are 5 more boys than girls". Although the term "more" is frequently used with addition, here "5" needs to be subtracted from the number of boys, to get the number of girls.

6 Conclusion

In this paper, we investigate methods to identify and understand mathematical relations expressed in text. We introduce the equation parsing task, which involves generating an equation from a sentence, as well as identifying what the variables represent. Our method leverages a new structure – an equation parse, which allows for an efficient search over the space of all equations. We develop a structured prediction formulation, which leverages equation parse based inference techniques, and jointly grounds variables and generates equations. Experimental evaluation suggests that predicting relations with equation parses performs considerably better than baselines which use template matching techniques.

To achieve these results we made several assumptions. The key one is that we only deal with expressions that are within a sentence. This should be lifted in future work. We also made a few assumptions on the way quantities are presented in the sentence (e.g., modifiers act in the same way on all variables, etc.) We believe that these assumption do not restrict our work but we intend to better quantify it in future.

References

Anders Björkelund and Jonas Kuhn. 2014. Learning structured perceptrons for coreference resolution with latent antecedents and non-local features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 47–57, Baltimore, Maryland, June. Association for Computational Linguistics.

S.R.K. Branavan, H. Chen, L. Zettlemoyer, and R. Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 82–90, Suntec, Singapore, August. Association for Computational Linguistics.

S. R. K. Branavan, Luke S. Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1268–1277, Stroudsburg, PA, USA. Association for Computational Linguistics.

M. Chang, D. Goldwasser, D. Roth, and V. Srikumar. 2010. Discriminative learning over constrained latent representations. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 6.

J. Clarke, D. Goldwasser, M. Chang, and D. Roth. 2010. Driving semantic parsing from the world's response. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, 7.

J. Eisenstein, J. Clarke, D. Goldwasser, and D. Roth. 2009. Reading to learn: Constructing features from semantic abstracts. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Singapore.

R. Ge and R. Mooney. 2006. Discriminative reranking for semantic parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 263–270, Sydney, Australia, July. Association for Computational Linguistics.

D. Goldwasser and D. Roth. 2011. Learning from natural instructions. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*.

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, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 523–533.

Rohit J. Kate and Raymond J. Mooney. 2007. Learning language semantics from ambiguous supervision. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 1*, AAAI'07, pages 895–900. AAAI Press.

N. Kushman, L. Zettlemoyer, R. Barzilay, and Y. Artzi. 2014. Learning to automatically solve algebra word problems. In *ACL (1)*, pages 271–281.

Marketwatch. [Online; accessed 20-February-2015].

S. Roy, T. Vieira, and D. Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics*, 3.

R. Samdani, M. Chang, and D. Roth. 2012. Unified expectation maximization. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 6.

Adam Vogel and Dan Jurafsky. 2010. Learning to follow navigational directions. In *Proceedings of the 48th*

Annual Meeting of the Association for Computational Linguistics, ACL '10, pages 806–814, Stroudsburg, PA, USA. Association for Computational Linguistics.

L. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*.

L. Zettlemoyer and M. Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Conference on Empirical Methods for Natural Language Processing - Conference on Computational Natural Language Learning*, pages 678–687.