Learning To Use Formulas To Solve Simple Arithmetic Problems

Arindam Mitra  
Arizona State University  
amitra7@asu.edu

Chitta Baral  
Arizona State University  
chitta@asu.edu

Abstract

Solving simple arithmetic word problems is one of the challenges in Natural Language Understanding. This paper presents a novel method to learn to use formulas to solve simple arithmetic word problems. Our system analyzes each of the sentences to identify the variables and their attributes; and automatically maps this information into a higher level representation. It then uses that representation to recognize the presence of a formula along with its associated variables. An equation is then generated from the formal description of the formula. In the training phase, it learns to score the <formula, variables> pair from the systematically generated higher level representation. It is able to solve 86.07% of the problems in a corpus of standard primary school test questions and beats the state-of-the-art by a margin of 8.07%.

1 Introduction

Developing algorithms to solve math word problems (Table 1) have been an interest of NLP researches for a long time (Feigenbaum and Feldman, 1963). It is an interesting topic of study from the point of view of natural language understanding and reasoning for several reasons. First, it incorporates rigorous standards of accurate comprehension. Second, we know a good representation to solve the word problems, namely algebraic equations. Finally, the evaluation is straightforward and the problems can be collected easily.

In the recent years several challenges have been proposed for natural language understanding. This includes the Winograd Schema challenge for commonsense reasoning (Levesque, 2011), Story Comprehension Challenge (Richardson et al., 2013), Facebook bAbI task (Weston et al., 2015), Semantic Textual Similarity (Agirre et al., 2012) and Textual Entailment (Bowman et al., 2015; Dagan et al., 2010). The study of word math problems is also an important problem as quantitative reasoning is inextricably related to human life. Clark & Etzioni (Clark, 2015; Clark and Etzioni, 2016) discuss various properties of math word (and science) problems emphasizing elementary school science and math tests as a driver for AI.

Researchers at Allen AI Institute have published two standard datasets as part of the Project Euclid\footnote{http://allenai.org/euclid.html} for future endeavors in this regard. One of them contains simple addition-subtraction arithmetic problems (Hosseini et al., 2014) and the other contains general arithmetic problems (Koncel-Kedziorski et al., 2015). In this research, we focus on the former one, namely the AddSub dataset.

| Formula | Associated variables |
|---------|----------------------|
| part-whole | whole: x, parts: {42, 47} |

Equation: \(x = 42 + 47\)

Table 1: Solving a word problem using part-whole

Broadly speaking, common to the existing approaches (Kushman et al., 2014; Hosseini et al., 2014; Zhou et al., 2015; Shi et al., 2015; Roy and Roth, 2015) is the task of grounding, that takes as input a word problem in the natural language and represent it into a formal language, such as, a system of equations, expression trees or states (Hosseini et al., 2014), from which the answer can be easily computed. In this work, we divide this task of grounding into two parts as follows:
In the first step, the system learns to connect the assertions in a word problem to abstract mathematical concepts or formulas. In the second step, it maps that formula into an algebraic equation. Examples of such formulas in the arithmetic domain include part whole which says, ‘the whole is equal to the sum of its parts’, or the Unitary Method that is used to solve problems like ‘A man walks seven miles in two hours. What is his average speed?’.

Consider the problem in Table 1. If the system can determine it is a ‘part whole’ problem where the unknown quantity X plays the role of whole and its parts are 42 and 47, it can easily express the relation as $X = 42 + 47$. The translation of a formula to an equation requires only the knowledge of the formula and can be formally encoded. Thus, we are interested in the question, ‘how can an agent learn to apply the formulas for the word problems?’ Solving a word problem in general, requires several such applications in series or parallel, generating multiple equations. However, in this research, we restrict the problems to be of single equations which requires only one application.

Our system currently considers three mathematical concepts: 1) the concept of part whole, 2) the concept of change and 3) the concept of comparison. These concepts are sufficient to solve the arithmetic word problems in AddSub. Table 2 illustrates each of these three concepts with examples. The part whole problems deal with the part whole relationships and ask for either the part or the whole. The change problems make use of the relationship between the new value of a quantity and its original value after the occurrence of a series of increase or decrease. The question then asks for either the initial value of the quantity or the final value of the quantity or the change. In case of comparison problems, the equation can be visualized as a comparison between two quantities and the question typically looks for either the larger quantity or the smaller quantity or the difference. While the equations are simple, the problems describe a wide variety of scenarios and the system needs to make sense of multiple sentences without a priori restrictions on the syntax or the vocabulary to solve the problem.

Training has been done in supervised fashion. For each example problem, we specify the formula that should be applied to generate the appropriate equation and the relevant variables. The system then learns to apply the formulas for new problems. It achieves an accuracy of 86.07% on the AddSub corpus containing 395 word arithmetic problems with a margin of 8.07% with the current state-of-the-art (Roy and Roth, 2015).

Our contributions are three-fold: (a) We model the application of a formula and present a novel method to learn to apply a formula; (b) We annotate the publicly available AddSub corpus with the correct formula and its associated variables; and

### Table 2: Examples of Add-Sub Word Problems

| Change            |                     |
|-------------------|---------------------|
| **RESULT UNKNOWN**| Mary had 18 baseball cards, and 8 were torn. Fred gave Mary 26 new baseball cards. Mary bought 40 baseball cards. How many baseball cards does Mary have now? |
| **CHANGE UNKNOWN**| There were 28 bales of hay in the barn. Tim stacked bales in the barn today. There are now 54 bales of hay in the barn. How many bales did he store in the barn? |
| **START UNKNOWN** | Sam’s dog had puppies and 8 had spots. He gave 2 to his friends. He now has 6 puppies. How many puppies did he have to start with? |

| Part Whole        |                     |
|-------------------|---------------------|
| **TOTAL SET UNKNOWN** | Tom went to 4 hockey games this year, but missed 7. He went to 9 games last year. How many hockey games did Tom go to in all? |
| **PART UNKNOWN**  | Sara’s high school played 12 basketball games this year. The team won most of their games. They were defeated during 4 games. How many games did they win? |

| Comparison        |                     |
|-------------------|---------------------|
| **DIFFERENCE UNKNOWN** | Last year, egg producers in Douglas County produced 1416 eggs. This year, those same farms produced 4636 eggs. How many more eggs did the farms produce this year? |
| **LARGE QUANTITY UNKNOWN** | Bill has 9 marbles. Jim has 7 more marbles than Bill. How many marbles does Jim have? |
| **SMALL QUANTITY UNKNOWN** | Bill has 9 marbles. He has 7 more marbles than Jim. How many marbles does Jim have? |
We model each formula as a template that has predefined slots and can be mapped to an equation. The target equation \( E \) where the target equation \( E \) is generated as follows:

\[
val_{\text{start}} + \sum_{g \in \text{gains}} val_g = \sum_{l \in \text{losses}} val_l + val_{\text{end}}
\]

Comparison The comparison concept has three slots namely the large quantity, the small quantity and their difference. An instantiation of the comparison concept is mapped to the following equation: \( \text{large} = \text{small} + \text{difference} \).

2.2 The Space of Possible Applications

Consider the problem in Table 1. Even though the correct application is an instance of part whole formula with \( \text{whole} = x \) and the \( \text{parts} \) being \( \{42, 47\} \), there are many other possible applications, such as, partWhole(whole=47, parts=x,42), change(start=47, losses={x}, gains={}, end = 42), comparison(large=47, small=x, difference=42). Note that, comparison(large=47, small=38, difference=42) is not a valid application since none of the associated variable is an unknown. Let \( A_P \) be the set of all possible applications to the problem \( P \). The following lemma characterizes the size of \( A_P \) as a function of the number of variables in \( P \).

Lemma 2.2.1. Let \( P \in \mathcal{P}^1_{\text{addsub}} \) be an arithmetic word problem with \( n \) variables (|\( V_P \)| = \( n \)), then the following are true:

1. The number of possible applications of part whole formula to the problem \( P \), \( N_{\text{partwhole}} \) is \((n + 1)2^{n-2} + 1\).
2. The number of possible applications of change formula to the problem \( P \), \( N_{\text{change}} \) is \(3^n-3(2n^2 + 6n + 1) - 2n + 1\).
3. The number of possible applications of comparison formula to the problem \( P \), \( N_{\text{comparison}} \) is \(3(n-1)(n-2)\).
4. The number of all possible applications to the problem $P$ is $N_{\text{partwhole}} + N_{\text{change}} + N_{\text{comparison}}$.

Proof of lemma 2.2.1 is provided in the Appendix. The total number of applications for problems having 3, 6, 7, 8 number of variables are 47, 3,105, 11,755, 43,699 respectively. Addition–Subtraction arithmetic problems hardly contain more than 6 variables. So, the number of possible applications is not intractable in practice. The total number of applications increases rapidly mainly due to the change concept. Since, the template involves two sets there is a $3^{n-3}$ factor present in the formula of $N_{\text{change}}$. However, any application of change concept with gains and losses slots containing a collection of variables can be broken down into multiple instances of change concept where the gains and losses slots accepts only a single variable by introducing more intermediate unknown variables. Since, for any formula that does not have a slot that accepts a set, the number of application is polynomial in the number of variables, there is a possibility to reduce the application space. We plan to explore this possibility in our future work. For the part whole concept, even though there is a exponential term involved, it is practically tractable (for $n = 10$, $N_{\text{partwhole}} = 2,817$). In practice, we believe that there will hardly be any part whole application involving more than 10 variables. For formulas that are used for other categories of word math problems (algebraic or arithmetic), such as the unitary method, formulas for ratio, percentage, time-distance and rate of interest, none of them have any slot that accepts sets of variables. Thus, further increase in the space of possible applications will be polynomial.

2.3 Probabilistic Model

For each problem $P$ there are different possible applications $y \in A_P$, however not all of them are meaningful. To capture the semantics of the word problem to discriminate between competing applications we use the log-linear model, which has a feature function $\phi$ and parameter vector $\theta \in \mathbb{R}^d$. The feature function $\phi : H \rightarrow \mathbb{R}^d$ takes as input a problem $P$ and a possible application $y$ and maps it to a $d$-dimensional real vector (feature vector) that aims to capture the important information required to discriminate between competing applications. Here, the set $H$ is defined as

\[
\{(P, y) : P \in \mathcal{P}_{\text{addsub}} \land y \in A_P\},
\]

to accommodate the dependency of the possible applications on the problem instance. Given the definition of the feature function $\phi$ and the parameter vector $\theta$, the probability of an application $y$ given a problem $P$ is defined as,

\[
p(y|P; \theta) = \frac{e^{\theta \cdot \phi(P, y)}}{\sum_{y' \in A_P} e^{\theta \cdot \phi(P, y')}}
\]

Here, . denotes dot product. Section 3 defines the feature function. Assuming that the parameter $\theta$ is known, the function $f$ that computes the correct application is defined as,

\[
f(P) = \arg \max_{y \in A_P} p(y|P; \theta)
\]

2.4 Parameter Estimation

To learn the function $f$, we need to estimate the parameter vector $\theta$. For that, we assume access to $n$ training examples, \{$(P_i, y_i^* : i = 1 \ldots n$\}, each containing a word problem $P_i$ and the correct application $y_i^*$ for the problem $P_i$. We estimate $\theta$ by minimizing the negative of the conditional log-likelihood of the data:

\[
O(\theta) = -\sum_{i=1}^n \log p(y_i^*|P_i; \theta) = -\sum_{i=1}^n [\theta \cdot \phi(P_i, y_i^*) - \log \sum_{y \in A_{P_i}} e^{\theta \cdot \phi(P_i, y)}]
\]

We use stochastic gradient descent to optimize the parameters. The gradient of the objective function is given by:

\[
\frac{\nabla O}{\nabla \theta} = -\sum_{i=1}^n [\phi(P_i, y_i^*) - \sum_{y \in A_{P_i}} p(y|P_i; \theta) \times \phi(P_i, y)]
\]

Note that, even though the space of possible applications vary with the problem $P_i$, the gradient for the example containing the problem $P_i$ can be easily computed.

3 Feature Function $\phi$

A formula captures the relationship between variables in an compact way which is sufficient to generate an appropriate equation. In a word problem, those relations are hidden in the assertions
of the story. The goal of the feature function is thus to gather enough information from the story so the underlying mathematical relation between the variables can be discovered. The feature function thus needs to be aware of the mathematical relations so that it knows what information it needs to find. It should also be “familiar” with the word problem language so that it can extract the information from the text. In this research, the feature function has access to machine readable dictionaries such as WordNet (Miller, 1995), ConceptNet (Liu and Singh, 2004) which captures inter word relationships such as hypernymy, synonymy, antonymy etc, and syntactic and dependency parsers that help to extract the subj, verb, object, preposition and temporal information from the sentences in the text. Given these resources, the feature function first computes a list of attributes for each variables. Then, for each application $y$ it uses that information, to compute if some aspects of the expected relationship described in $y$ is satisfied by the variables in $y$.

Let the first $b$ dimensions of the feature vector contain part whole related features, the next $c$ dimensions are for change related features and the remaining $d$ features are for comparison concept. Then the feature vector for a problem $P$ and an application of a formula $y$ is computed in the following way:

\[
\text{Data: } \text{A word problem } P, \text{ an application } y \\
\text{Result: } d\text{-dimensional feature vector, } f_v \\
\text{Initialize } f_v := 0 \\
\text{if } y \text{ is instance of part whole then} \\
\quad \text{compute } f_v[1:b] \\
\text{end} \\
\text{if } y \text{ is instance of change then} \\
\quad \text{compute } f_v[b + 1:b + c] \\
\text{end} \\
\text{if } y \text{ is instance of comparison then} \\
\quad \text{compute } f_v[b + c + 1:b + c + d] \\
\text{end} \\
\text{Algorithm 1: Skeleton of the feature function } \phi
\]

The rest of the section is organized as follows. We first describe the attributes of the variables that are computed from the text. Then, we define a list of boolean variables which computes semantic relations between the attributes of each pair of variables. Finally, we present the complete definition of the feature function using the description of the attributes and the boolean variables.

3.1 Attributes of Variables

For each occurrence of a number in the text a variable is created with the attribute value referring to that numeric value. An unknown variable is created corresponding to the question. A special attribute type denotes the kind of object the variable refers to. Table 3 shows several examples of the type attribute. It plays an important role in identifying irrelevant numbers while answering the question.

| Text                        | Type    |
|-----------------------------|---------|
| John had 70 seashells       | seashells|
| 70 seashells and 8 were broken | seashells|
| 61 male and 78 female salmon | male, salmon|
| 35 pears and 27 apples      | pear    |

Table 3: Example of type for highlighted variables.

The other attributes of a variable captures it’s linguistic context to surrogte the meaning of the variable. This includes the verb attribute i.e. the verb attached to the variable, and attributes corresponding to Stanford dependency relations (De Marneffe and Manning, 2008), such as nsubj, tmod, prep in, that spans from either the words in associated verb or words in the type. These attributes were computed using Stanford Core NLP (Manning et al., 2014). For the sentence, “John found 70 seashells on the beach.” the attributes of the variable are the following: \{value: \{70\}, verb: \{found\}, nsubj: \{John\}, prep on: \{beach\}\}.

3.2 Cross Attribute Relations

Once the variables are created and their attributes are extracted, our system computes a set of boolean variables, each denoting whether the attribute $a_1$ of the variable $v_1$ has the same value as the attribute $a_2$ of the variable $v_2$. The value of each attribute is a set of words, consequently set equality is used to calculate attribute equality. Two words are considered equal if their lemma matches.

Four more boolean variables are computed for each pair of variables based on the attribute type and they are defined as follows:

**subType**: Variable $v_1$ is a subType of variable $v_2$ if $v_2.type \subseteq v_1.type$ or their type consists of a single word and there exists the IsA relation between them in ConceptNet (Speer and Havasi, 2013; Liu and Singh, 2004).
disjointType is true if \( v_1.\text{type} \cap v_2.\text{type} = \phi \)

intersectingType is true if \( v_1 \) is neither a subType of \( v_2 \) nor is disjointType nor equal.

We further compute some more variables by utilizing several relations that exist between words:

antonym: For every pair of variables \( v_1 \) and \( v_2 \), we compute an antonym variable that is true if there exists a pair of words in \( (v_1.\text{verb}) \cup v_1.\text{adj} \times (v_2.\text{verb}) \cup v_2.\text{adj} \) that are antonym to each other in WordNet irrespective of their part of speech tag.

relatedVerbs: The verbs of two variables are related if there exists a RelatedTo relation in ConceptNet between them.

subjConsume: The \( nsubj \) of \( v_1 \) consumes the \( nsubj \) of \( v_2 \) if the former refers to a group and the later is a part of that group. For example, in the problem ‘Joan grew 29 carrots and 14 watermelons. Jessica grew 11 carrots. How many carrots did they grow in all?’, the \( nsubj \) of the unknown variable consumes others. This is computed using Stanford co-reference resolution. For the situation where there is a variable with \( nsubj \) as ‘they’ and it does not refer to any entity, the \( subjConsume \) variable is assumed to be implicitly true for any variable having a \( nsubj \) of type person.

3.3 Features: Part Whole

The part whole features look for some combinations of the boolean variables and the presence of some cue words (e.g. ‘all’) in the attribute list. These features capture the underlying reasonings that can affect the decision of applying a part whole concept. We describe the conditions which when satisfied activate the features. If active, the value of a feature is the number of variables associated with the application \( y \) and 0 otherwise. This is also true for change and comparision features also. Part whole features are computed only when the \( y \) is an instance of the formula \( \text{part whole} \). The same applies for change and comparision features.

Generic Word Cue This feature is activated if \( y.\text{whole} \) has a word in its attributes that belongs to the “total words set” containing the followings words “all”, “total”, “overall”, “altogether”, “together” and “combine”; and none of the variables in parts are marked with these words.

ISA Type Cue is active if all the part variables are subType of the whole.

Type-Verb Cue is active if the type and verb attributes of \( v_{\text{whole}} \) matches that of all the variables in the part slot of \( y \).

Type-Individual Group Cue is active if the variable \( v_{\text{whole}}.\text{subjConsume} \) each part variable \( v_p \) in \( y \) and their type matches.

Type-Verb-Tmod Cue is active if the variable in the slot \( \text{whole} \) is the unknown and for each part variable \( v_p \), their \( \text{verb} \), \( \text{type} \) and \( \text{tmod} \) (time modifier of the verb) attributes match.

Type-SubType-Verb Cue is active if the variable in the slot \( \text{whole} \) is either the unknown or marked with a word in “total words set” and for all parts \( v_p \), their \( \text{verb} \) matches and one of the type or subType boolean variable is true.

Type-SubType-Related Verb Cue is similar to Type-SubType-Verb Cue however relaxes the \( \text{verb} \) match conditions to related verb match. This is helpful in problems like ‘Mary went to the mall. She spent $13.04 on a shirt and $12.27 on a jacket. She went to 2 shops. In total, how much money did Mary spend on clothing?’.

Type-Loose Verb Cue ConceptNet does not contain all relations between verbs. For example, according to ConceptNet ‘buy’ and ‘spend’ are related however there is no relation in ConceptNet between ‘purchase’ and ‘spend’. To handle these situations, we use this feature which is similar to the previous one. The difference is that it assumes that the verbs of part-whole variable pairs are related if all verbs associated with the parts are same, even though there is no relation in ConceptNet.

Type-Verb-Prep Cue is active if type and verb matches. The whole does not have a “preposition” but parts have and they are different.

Other Cues There are also features that add \( nsubj \) match criteria to the above ones. The \( prior \) feature for \( \text{part whole} \) is that the whole if not unknown, is smaller than the sum of the parts. There is one more feature that is active if the two part variables are antonym to each other; one of type or subType should be true.

3.4 Features: Change

The change features are computed from a set of 10 simple indicator variables, which are computed in the following way:
Start Cue is active if the verb associated with the variable in start slot has one of the following possessive verbs: {‘call for’, ‘be’, ‘contain’, ‘remain’, ‘want’, ‘has’, ‘have’, ‘hold’, ...}; the type and nsubj of start variable match with the end variable and the tense of the end does not precede the start. The list of ‘possessive verbs’ is automatically constructed by adding all the verbs associated with the start and the end slot variables in annotated corpus.

Start Explicit Cue is active if one of following words, “started with”, “initially”, “begining”, “originally” appear in the context of the start variable and the type of start and end variables match.

Start prior is active if the verb associated with the variable in start slot is a member of the set ‘possessive verbs’ and the variable appears in first sentence.

Start Default Cue is active if start variable has a “possessive verb” with past tense.

End Cue is active if the verb associated with the variable in slot end has a possessive verb with the tense of the verb not preceding the tense of the start, in case the start is not missing. The type and nsubj should match with either the start or the gains in case the start is missing.

End Prior is true if v_end has a possessive verb and an unknown quantity and at least one of v_end or v_start does not have a nsubj attribute.

Gain Cue is active if for all variables in the gains slot, the type matches with either v_end or v_start and one of the following is true: 1) the nsubj of the variable matches with v_end or v_start and the verb implies gain (such as ‘find’) and 2) the nsubj of the variable does not match with v_end or v_start and the verb implies losing (e.g. spend). The set of gain and loss verbs are collected from the annotated corpus by following the above procedure.

Gain Prior is true if problem contains only three variable, with v_start < v_end and the only variable in the gain slot, associated with non-possessive is the unknown.

Loss Cue & Loss prior are designed in a fashion similar to the Gain cue and Gain Prior.

Let’s say bad\_gains denotes that none of the gain prior or gain cue is active even though the gain slots is not empty. bad\_losses is defined similarly and let bad = bad\_gains ∨ bad\_losses. Then the change features are computed from these boolean indicators using logical operators and, or, not. Table 4 shows some of the change features.

| Feature                                                                                                                                   |
|------------------------------------------------------------------------------------------------------------------------------------------|
| ![bad ∧ gain\_cure ∧ start\_default ∧ end\_cure](image)                                                                                     |
| ![bad\_!\_gains ∨ loss\_cure ∧ start\_default ∧ end\_cure](image)                                                                            |
| ![bad ∧ (gain\_cure ∨ loss\_cure) ∧ start\_cure\_!\_default ∧ end\_cure](image)                                                            |
| ![bad \!\_cure\_gains ∧ (gain\_cure ∨ loss\_cure) ∧ start\_prior \!\_end\_prior](image)                                               |
| ![bad ∧ (gain\_cure ∨ loss\_cure) ∧ (start\_prior ∨ start\_cure\_!\_default ∧ end\_prior](image) |

Table 4: Activation criteria of some change related features.

3.5 Features: Comparison

The features for the “compare” concept are relatively straight forward.

Difference Unknown Que If the application y states that the unknown quantity is the difference between the larger and smaller quantity, it is natural to see if the variable in the difference slot is marked with a comparative adjective or comparative adverb. The prior is that the value of the large quantity must be bigger than the small one. Another two features add the type and subject matching criteria along with the previous ones.

Large & Small Unknown Que These features can be active only when the variable in the large or small slot is unknown. To detect if the referent is bigger or smaller, it is important to know the meaning of the comparative words such as ‘less’ and ‘longer’. Since, the corpus contains only 33 comparison problems we collect these comparative words from web which are then divided into two categories. With these categories, the features are designed in a fashion similar to change features that looks for type, subject matches.

3.6 Handling Arbitrary Number of Variables

This approach can handle arbitrary number of variables. To see that consider the problem, ‘Sally found 9 seashells , Tom found 7 seashells , and Jessica found 5 seashells on the beach . How many seashells did they find together ?’. Let us say the feature vector contains only the ‘Type-Individual Group Cue’ feature and the weight
of that feature is 1. Consider the two following applications: 
\( y_1 = \text{partWhole}(x, \{9, 7\}) \) and 
\( y_2 = \text{partWhole}(x, \{9, 7, 5\}) \). For both \( y_1 \) and \( y_2 \) the ‘Type-Individual Group Cue’ feature is active since the subject of the unknown \( x \) refers to a group that contains the subject of all part variables in \( y_1 \) and \( y_2 \) and their type matches. However, as mentioned in section 3.3, when active, the value of a feature is the number of variables associated with the application. Thus 
\( \frac{p(y_2; P, B)}{p(y_1; P, B)} = \frac{e^4}{e^3} = e \). Thus, \( y_2 \) is more probable than \( y_1 \).

4 Related Works

Researchers in early years have studied math word problem in a constrained domain by either limiting the input sentences to fixed set of patterns (Bobrow, 1964b; Bobrow, 1964a; Hinsley et al., 1977) or by directly operating on a propositional representation instead of a natural language text (Kintsch and Greeno, 1985; Fletcher, 1985). Mukherjee and Garain (2008) survey these works.

Among the recent algorithms, the most general ones are the work in (Kushman et al., 2014; Zhou et al., 2015). Both algorithms tries to map a word math problem to a ‘system template’ that contains a set of ‘equation templates’ such as \( ax + by = c \). These ‘system templates’ are collected from the training data. They implicitly assume that these templates will reoccur in the new examples which is a major drawback of these algorithms. Also, Koncel-Kedziorksi et al. (2015) show that the work of Kushman et al. (2014) heavily relies on the overlap between train and test data and when this overlap is reduced the system performs poorly.

Work of (Koncel-Kedziorksi et al., 2015; Roy and Roth, 2015) on the other hand try to map the math word problem to an expression tree. Even though, these algorithms can handle all the four arithmetic operators they cannot solve problems that require more than one equation. Moreover, experiments show that our system is much more robust to diversity in the problem types between training and test data for the problems it handles.

The system ARIS in (Hosseini et al., 2014) solves the addition-subtraction problems by categorizing the verbs into seven categories such as ‘positive transfer’, ‘loss’ etc. It represents the information in a problem as a state and then updates a step according to the category of a verb as the story progresses. Both ARIS and our system share the property that they give some explanation behind the equation they create. However, the verb categorization approach of ARIS can only solve a subset of addition-subtraction problems (see error analysis in (Hosseini et al., 2014)); whereas the usage of formulas to model the word problem world, gives our system the ability to accommodate other math word problems as well.

5 Experimental Evaluation

5.1 Dataset

The AddSub dataset consist of a total of 395 addition-subtraction arithmetic problems for third, fourth, and fifth graders. The dataset is divided into three diverse set MA1, MA2, IXL containing 134, 140 and 121 problems respectively. As mentioned in (Hosseini et al., 2014), the problems in MA2 have more irrelevant information compared to the other two datasets, and IXL includes more information gaps.

5.2 Result

Hosseini et al. (2014) evaluate their system using 3-fold cross validation. We follow that same procedure. Table 5 shows the accuracy of our system on each dataset (when trained on the other two datasets). Table 6 shows the distribution of the part whole, change, comparison problems and the accuracy on recognizing the correct formula.

|       | MA1 | IXL | MA2 | Avg |
|-------|-----|-----|-----|-----|
| ARIS  | 83.6| 75.0| 74.4| 77.7|
| KAZB  | 89.6| 51.1| 51.2| 64.0|
| ALGES | -   | -   | -   | 77.0|
| Roy & Roth | - | - | - | 78.0|
| Majority | 45.5 | 71.4 | 23.7 | 48.9|
| Our System | 96.27 | 82.14 | 79.33 | 86.07|

Table 5: Comparision with ARIS, KAZB (Kushman et al., 2014), ALGES (Koncel-Kedziorksi et al., 2015) and the state of the art Roy & Roth on the accuracy of solving arithmetic problems.

As we can see in Table 6 only IXL contains problems of type ‘comparison’. So, to study the accuracy on the compare formula we uniformly distribute the 33 examples over the 3 datasets. Doing that results in only two errors in the recognition of a compare formula and also increases the overall accuracy of solving arithmetic problems to 90.38%.
5.3 Error Analysis

An equation that can be generated from a change or comparision formula can also be generated by a part whole formula. Four such errors happened for the change problems and out of the 33 compare problems, 18 were solved by part whole. Also, there are 3 problems that require two applications. One example of such problem is, “There are 48 erasers in the drawer and 30 erasers on the desk. Alyssa placed 39 erasers and 45 rulers on the desk. How many erasers are now there in total ?”. To solve this we need to first combine the two numbers 48 and 30 to find the total number of erasers she initially had. This requires the knowledge of ‘part-whole’. Now, that sum of 48 and 30, 39 and x can be connected together using the ‘change’ formula. With respect to ‘solving’ arithmetic problems, we find the following categories as the major source of errors:

Problem Representation: Solving problems in this category require involved representation. Consider the problem, ‘Sally paid $12.32 total for peaches, after a ‘3 dollar’ coupon, and $11.54 for cherries. In total, how much money did Sally spend?’ Since the associated verb for the variable 3 dollar is ‘pay’, our system incorrectly thinks that Sally did spend it.

Information Gap: Often, information that is critical to solve a problem is not present in the text. E.g. Last year, 90171 people were born in a country, and 16320 people immigrated to it. How many new people began living in the country last year ?. To correctly solve this problem, it is important to know that both the event ‘born’ and ‘immigration’ imply the ‘began living’ event, however that information is missing in the text. Another example is the problem, ‘Keith spent $6.51 on a rabbit toy, $5.79 on pet food, and a cage cost him $12.51. He found a dollar bill on the ground. What was the total cost of Keith’s purchases?’ It is important to know here that if a cage cost Keith $12.51 then Keith has spent $12.51 for cage.

Modals: Consider the question ‘Jason went to 11 football games this month. He went to 17 games last month, and plans to go to 16 games next month. How many games will he attend in all ?’. To solve this question one needs to understand the meaning of the verb “plan” and “will”. If we replace “will” in the question by “did” the answer will be different. Currently our algorithm cannot solve this problem and we need to either use a better representation or more powerful learning algorithm to be able to answer correctly. Another interesting example of this kind is the following: “For his car, Mike spent $118.54 on speakers and $106.33 on new tires. Mike wanted 3 CD’s for $4.58 but decided not to. In total, how much did Mike spend on car parts ?”.

Incomplete IsA Knowledge: For the problem “Tom bought a skateboard for $9.46, and spent $9.56 on marbles. Tom also spent $14.50 on shorts. In total, how much did Tom spend on toys ?”, it is important to know that ‘skateboard’ and ‘marbles’ are toys but ‘shorts’ are not. However such knowledge is not always present in ConceptNet which results in error.

Parser Issue: Error in dependency parsing is another source of error. Since the attribute values are computed from the dependency parse tree, a wrong assignment (mostly for verbs) often makes the entity irrelevant to the computation.

6 Conclusion

Solving math word problems often requires explicit modeling of the word. In this research, we use well-known math formulas to model the word problem and develop algorithm that learns to map the assertions in the story to the correct formula. Our future plan is to apply this model to general arithmetic problems which require multiple applications of formulas.

7 Acknowledgement

We thank NSF for the DataNet Federation Consortium grant OCI-0940841 and ONR for their grant N00014-13-1-0334 for partially supporting this research.

| Type         | MA1 correct | MA2 correct | IXL correct |
|--------------|-------------|-------------|-------------|
| part whole   | 59          | 59          | 89          | 81          | 51          | 40          |
| change       | 74          | 70          | 18          | 15          | 68          | 56          |
| compare      | 0           | 0           | 33          | 0           | 0           | 0           |

Table 6: Accuracy on recognizing the correct application. None of the MA1 and MA2 dataset contains “compare” problems so the cross validation accuracy on “IXL” for “compare” problems is 0.
References

Eneko Agirre, Mona Diab, Daniel Cer, and Aitor Gonzalez-Agirre. 2012. Semeval-2012 task 6: A pilot on semantic textual similarity. In Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation, pages 385–393. Association for Computational Linguistics.

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

Daniel G. Bobrow. 1964b. A question-answering system for high school algebra word problems. In Proceedings of the October 27-29, 1964, Fall Joint Computer Conference, Part I, AFIPS ’64 (Fall, part 1), pages 591–614, New York, NY, USA. ACM.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. arXiv preprint arXiv:1508.05326.

Peter Clark and Oren Etzioni. 2016. My computer is an honor student but how intelligent is it? standardized tests as a measure of ai. AI Magazine.(To appear).

Peter Clark. 2015. Elementary school science and math tests as a driver for ai: Take the aristo challenge! In AAAI, pages 4019–4021.

Ido Dagan, Bill Dolan, Bernardo Magnini, and Dan Roth. 2010. Recognizing textual entailment: Rational, evaluation and approaches—erratum. Natural Language Engineering, 16(01):105–105.

Marie-Catherine De Marneffe and Christopher D Manning. 2008. Stanford typed dependencies manual. Technical report, Technical report, Stanford University.

Edward A. Feigenbaum and Julian Feldman. 1963. Computers and thought.

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

Dan A. Hinsley, John R. Hayes, and Herbert A. Simon. 1977. From words to equations: Meaning and representation in algebra word problems. Cognitive processes in comprehension, 329.

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), pages 523–533.

Walter Kintsch and James G. Greeno. 1985. Understanding and solving word arithmetic problems. Psychological review, 92(1):109.

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. Association for Computational Linguistics.

Hector J. Levesque. 2011. The winograd schema challenge.

Hugo Liu and Push Singh. 2004. Conceptneta practical commonsense reasoning tool-kit. BT technology journal, 22(4):211–226.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In ACL (System Demonstrations), pages 55–60.

George A. Miller. 1995. Wordnet: a lexical database for english. Communications of the ACM, 38(11):39–41.

Anirban Mukherjee and Utpal Garain. 2008. A review of methods for automatic understanding of natural language mathematical problems. Artificial Intelligence Review, 29(2):93–122.

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. 2013. Mctest: A challenge dataset for the open-domain machine comprehension of text. In EMNLP, volume 1, page 2.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. 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 the Conference on Empirical Methods in Natural Language Processing (EMNLP), Lisbon, Portugal.

Robert Speer and Catherine Havasi. 2013. Conceptnet 5: A large semantic network for relational knowledge. In The Peoples Web Meets NLP, pages 161–176. Springer.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. arXiv preprint arXiv:1502.05698.

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, pages 817–822.