Learning to Solve General Arithmetic Problems Using World Knowledge: A Planning Based Approach

Abstract

Automated solving of general arithmetic problems is one of the challenges in Natural Language Understanding. To solve such a problem one needs background knowledge (mathematical formulas, semantics of arithmetic operators, unit conversion knowledge) which is often missing in the text. This paper formulates the arithmetic problem solving as a planning problem and shows how an automated solver can incorporate such knowledge. Unlike the existing solvers which predict the equation in a single step, our system performs a sequence of actions until a solution can be computed or a limit on the maximum length of the sequence has been reached. A set of 13 actions and their effects have been defined. The execution condition of the actions and the order of execution is learned using conditional log-linear model. The system is evaluated on 4 standard datasets and achieves state-of-the-art performance on all of them.

Natural Language Understanding (NLU) includes being able to answer questions with respect to a given text and often needs knowledge beyond the given text. Its difficulties have led to the development of many NLU challenges, such as Winograd Schema challenge (Levesque 2011), Story Comprehension Challenge (Richardson, Burges, and Renshaw 2013; Rajpurkar et al. 2016), Science and Math question answering challenge (Clark 2015; Clark and Etzioni 2016), Facebook bAbl task (Weston et al. 2015), Semantic Textual Similarity (Agirre et al. 2012) and Textual Entailment (Bowman et al. 2015; Dagan et al. 2010). In this research our focus is on general arithmetic problems (Table 1).

General arithmetic problems contain a small narrative followed by a question (Table 1). To answer the question a student needs to know various concepts of arithmetic. For example, consider the problem 1 from Table 1. To solve this problem one needs to be familiar with the concept of 'unitary' formula which says if one item costs r unit and if you buy m number of items and the total cost incurred is t, then $t = m \times r$. In this problem, the quantity asked is the per item cost, the quantity 9 denotes the number of items (students) and 81 is the total cost. Once one figures out that the concept of unitary formula can be applied to this problem and finds out the related quantities, it is straight forward to generate the equation, which is $9 \times x = 81$. Here, applying the formula by recognizing its applicability and detecting all its associated quantities is considered to be an *action* (denoted as *unitary(m,r,t)*) and the task of forming the equation following the definition of the unitary formula is considered to be the *effect* of the action.

1. *There are 9 students in the class and 81 tickets. If the tickets are divided equally among the students, how many does each student get?*
2. *Ned bought 14 boxes of chocolate candy and gave 5 to his little brother. If each box has 6 pieces inside it, how many pieces did Ned still have?*
3. *Sam, Dan, Tom, and Keith each have 15 Pokemon cards. How many Pokemon cards do they have in all?*
4. *Nancy has saved 4900 cents from selling lemonade. How many dollars does Nancy have?*

Table 1: Sample General Arithmetic Problems

Contrary to the previous example, it is not always possible to "directly" apply a formula. Consider the problem 2 in Table 1. Even though one might recognize the applicability of the unitary formula to this problem, finding out the number of items (boxes) will require some additional effort. First, one needs to figure out that 14 is not the number of boxes Ned currently has. In fact the number of boxes Ned currently has is not directly provided in the text. Secondly, one needs to deduce that the number of boxes Ned initially had was 14 and that has been reduced by 5, so the current number of boxes in possession of Ned can be computed by subtracting 5 from 14, which is 9. Once this piece of information has been inferred from the text, one might finally conclude that *unitary(9,6,x)* holds and $9 \times 6 = x$. We refer to this action of inferring the current value of a quantity after a negative change has happened as *decrease*. Once one executes this action a new quantity with appropriate value and the metadata containing the action that created it, is added to the context.

Contrary to the *unitary* action which pertains to a for-

mula, *decrease* action corresponds to an arithmetic operator (here subtraction) and allows one to infer implied numeric information from the text. Another action which does the same but does not pertain to a formula or arithmetic operator is called *counting*. Students generally posses the skills of counting by the time they start their lessons on arithmetic and can use those whenever it is needed. An example of this scenario is the problem 3 from Table 1, where one needs to find the number of people by counting the individuals who have pokemon cards. Once the counting is done and the student has figured out that there are a total of 4 people who have pokenmon cards (a new numeric quantity), she can further proceed to deduce that *unitary(4,15,x)* holds and $4 \times 15 = x$.

In the previous examples (1-3), all the numeric information that is relevant to answer the question is specified, either directly or indirectly. However, sometimes some numeric information is not specified and it is expected that the student should know those numeric facts. One prominent example of this is the assumption of the knowledge of unit conversion rates (1 dollar = 100 cents, 1 dozen = 12 etc). In problem 4 from Table 1, it is up to the student to recognize that the unit conversion rate between dollar and cent is missing and is needed to answer the question. Here, the recognition of the need of a unit conversion knowledge is considered to be an *action* with the addition of the knowledge to the problem context being its *effect*. Once the information "1 dollar is equal to 100 cents" is added to the problem context, one can proceed further to conclude that *unitary(x,100,4500)* holds.

Planning is one of the 4 crucial steps in problem solving (Polya 1957). However, existing solvers have largely ignored this phase in their pursuit of classifying the equation in a single step. In this research, we propose a planing based approach for solving the general arithmetic problems, where at each step, the solver executes an action and as a result an equation is created or a numeric quantity either inferred from the text or containing some missing knowledge is added to the context. The goal condition is reached when an equation is created as all the problems in the datasets have only one unknown and can be solved by a single equation. However, the goal condition can be modified to allow it to generate sufficient number of equations. Deciding which actions to execute and when is learned from the annotated corpus.

The rest of the paper is organized as follows: in section 1, we formally define the task of solving arithmetic problems and describe the 13 actions that are used in this research; In section 2, we describe the machine learning model. Section 3 presents the feature function. In section 4, we describe the related works. In section 5, we present a detailed experimental evaluation of our system. Finally, section 6 concludes our paper. The code and the data will be made publicly available at *anonymous url*.

1 Problem Formulation

A single equation word arithmetic problem P is a sequence of k words $\langle w_1, ..., w_k \rangle$ and contains a set of *numeric quantities* $\mathbb{V}_P = \{v_0, v_1, ..., v_{n-1}, x\}$ where $v_0, v_1, ..., v_{n-1}$ are the *numbers* in P and x is the *unknown* whose value is the answer we seek (Koncel-Kedziorski et al. 2015). Each problem P can be solved by evaluating a valid mathematical equation E formed by combining elements of $\mathbb{V}_P^*$ using the binary operators from $\mathcal{O} = \{+, -, *, \div, =\}$. Here, the set $\mathbb{V}_P^*$ contains all the quantities from $\mathbb{V}_P$ and might contain some additional numeric quantities which are not directly specified in P but are required for solving P. Problem 3 and 4 from Table 1 are examples of this scenario.

Let $\mathcal{P}$ be the set of all such problems. We assume that each target equation E of $P \in \mathcal{P}$ is generated by performing a sequence of actions. Let l_P denote the length of the sequence and let A_P^t denotes the set of possible actions at time step t, where $1 \leq t \leq l_P$. The goal is then to find the correct sequence of actions $s_P^* \in A_P^1 \times A_P^2 \cdots \times A_P^{l_P}$ for the problem $P \in \mathcal{P}$. In the following subsections we describe the actions and the construction of the set A_P^t. Before that, we describe the data structure we have used to represent each numeric quantity in $\mathbb{V}_P$ which will be useful for explaining the working of the actions.

Quantity Schema For each occurrence of a number in the text, a quantity object is created with the attribute *value* referring to that numeric value. An unknown quantity is created corresponding to the question. A special attribute *type* denotes the kind of object the quantity refers to. Table 2 shows several examples of the *type* attribute. Another special attribute *rate* denotes the denominator of a rate type. For example, for the quantity "20 balls per box" or "each box contains 20 balls" the *type* is "ball" and the *rate* refers to "box".

Text	Type
John had **70** seashells	seashells
61 male and 78 female salmon	male, salmon

Table 2: Example of *type* for highlighted variables.

The other attributes in the quantity schema captures its linguistic context to surrogate its meaning. This includes the *verb* attribute i.e. the verb attached to the number, 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 played 10 matches in January." the attributes of the variable are the following: { **value**: {10}, **type**: {match}, **rate**: {}, **verb:** {*play*} , **nsubj :** {*John*}, **prep in:** {*January* }}. Let *Attr* denote the set of all attributes and $attr(q, a)$ be the value of the attribute a of quantity q. Also each quantity has some metadata containing all the information about the action that created it, if any. To save space, sometimes $a(q)$ is used to denote $attr(q, a)$.

Action pertaining to Formulas

We model each action pertaining to a formula as a template that has predefined slots. When executed these actions produce a single equation. Table 3 describes all these actions

and their effects. Each of these templates are instantiated by a subset of $\mathbb{V}_P$ that contains the *unknown*. Since there are different ways to fill the slots there are several possible actions for each formula and all of those are part of A_P^t. Note that $\mathbb{V}_P$ is updated by other actions and changes its content over time steps.

Action	Slots	Effect
PartWhole	• whole • parts	$val(whole) =$ $\sum_{q \in Parts} val(q)$
GainConcept	• start • gain • end	$val(end) =$ $val(start) +$ $val(gain)$
LossConcept	• start • loss • end	$val(end) =$ $val(start) -$ $val(loss)$
Comparison	• big • small • diff	$val(big) -$ $val(small) =$ $val(diff)$
Unitary	• multiplier • unit cost • total	$val(total) =$ $val(multiplier)$ $* val(unit\ cost)$

Table 3: This table shows all the actions pertaining to a formula. Each of these actions has fixed number of slots (column 2) and creates an equation when executed (column 3). All slots accept a single quantity except the *parts* slot which accepts a set of quantity.

Consider, the *unitary* action and the problem 1 from Table 1. Initially, the set $\mathbb{V}_P = \{9, 81, x\}$. Thus all of the following, *unitary(x,9,81)*, *unitary(x,81,9)*, *unitary(81,x,9)*, *unitary(81,9,x)*, *unitary(9,x,81)*, *unitary(9,81,x)* are part of A_P^1. It is important to note that these formulas are easily available from math books and are very small in number. Also, going forward the use of formulas will remain important for algebraic word problems. One who does not know that $I = \frac{p \times r \times t}{100}$ will probably fail to solve investment problems such as 'You invested $500 and received $650 after three years. What had been the interest rate?'.

Actions for Implied Information

Not all problems directly specify the information needed for applying a formula. As shown in example 2 & 3 (Table 1) some numeric information might be needed to be inferred from the text. To handle these situations, a total of 7 such actions are considered in this research (Table 5). Among those two (*Join*, *Increase*) correspond to the concept of *addition*, two (*Separate*, *Decrease*) correspond to *subtraction* and other three (*Multiply*, *Divide*, *Counting*) correspond to the concept of *multiplication, division & counting* respectively. Similar to the actions for *formulas*, these actions are also modeled as a template having predefined slots. However, when these actions are executed a quantity is added to $\mathbb{V}_P$. Table 5 describes these actions and the attribute of the resulting quantity. For the *count* action, all sets containing at least 2 nouns that either have a same NER class or appear with same dependency relation (e.g. nsubj, prep_in, tmod) are considered.

Action for Unit Change

This action *UnitConversionKnowldge* decides if it should use a unit conversion knowledge. For each possible unit conversion knowledge in our system's database, an action is defined to decide whether that particular knowledge should be used. All of those are part of A_P^t. When an action of unit conversion is executed at time point t a quantity with correct unit and value equal to the conversion rate is added to $\mathbb{V}_P$. For example, given problem 4 from Table 1, our system will first execute the unit change action from *dollar* to *cents*. As a result a *Quantity(value=100, type = cent, rate=dollar)* denoting '100 cents per dollar' will be added to $\mathbb{V}_P$.

The Size of A_P^t Let us say that the number of quantities in $\mathbb{V}_P$ at time point t is n. Most of the actions have two or three slots, thus those templates can be filled with $n(n-1)$ and $n(n-1)(n-2)$ ways respectively. The *part* slots in *Join* action accepts a set and thus can be filled with at most 2^n ways and similarly the *part whole* template can be filled with $n2^{n-1}$ ways. The number of unit change knowledge in systems database is 28 thus there are 28 possible unit change knowledge actions. The number of count actions depends on the word problem and is usually small.

2 Probabilistic Model

For each problem P there are different possible actions $y \in A_P^t$, however not all of them are meaningful. To capture the semantics of the word problem to discriminate between competing actions we use the log-linear model, which has a feature function ϕ and a parameter vector $\theta \in \mathbb{R}^d$. The feature function $\phi : H \rightarrow \mathbb{R}^d$ takes as input a problem P, the numeric quantities $\mathbb{V}_P$ 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. Given the definition of the feature function ϕ and the parameter vector θ, the probability of an application y given a problem P is defined as,

$$p(y|P, \mathbb{V}_P; \theta) = \frac{e^{\theta . \phi(P, \mathbb{V}_P, y)}}{\sum_{y' \in A_P^t} e^{\theta . \phi(P, \mathbb{V}_P, y')}}$$

Here, '.' denotes dot product. Section 3 defines the feature function. Assuming that the parameter θ is known, the function f that computes the correct action from A_P^t is defined as,

$$f(P, \mathbb{V}_P) = \arg \max_{y \in A_P^t} p(y|P, \mathbb{V}_P; \theta)$$

The algorithm that finds the sequence of actions $\hat{s}$ for the input P is then defined in Algorithm 1.

Parameter Estimation

To learn to find a sequence, we need to estimate the parameter vector θ. For that, we assume access to n labeled data. It is important to note that at some time point t, there may be multiple possible correct actions. For the problem, "Sara, Keith, Benny, and Alyssa each have 96 baseball cards. How many dozen baseball cards do they have in all?" one can first

Action	Slots	Effect ($\mathbb{V}_P = \mathbb{V}_P \cup \{q'\}$)	
Join	• parts	$val(q') = \sum_{q \in Parts} val(q)$	$attr(q', a) = \cup_{q \in parts} attr(q, a), \forall a \in Attr$ $type(q') = hypernym(\{type(q) : q \in parts\})$
Separate	• whole • part	$val(q') = val(whole) - val(part)$	$attr(q', a) = attr(whole, a), \forall a \in Attr$ $type(q') = type(q''),\ s.t, q'' \in V_P, type(whole) = hypernym(type(q''), type(part)))$
Increase	• start • gain	$val(q') = val(start) + val(gain)$	$attr(q', a) = attr(start, a), \forall a \in Attr$
Decrease	• start • gain	$val(q') = val(start) - val(loss)$	$attr(q', a) = attr(start, a), \forall a \in Attr$
Multiply	• multiplicand • multiplier	$val(q') = multiplier \times multipicand$	$attr(q', a) = attr(multiplicand, a), \forall a \in Attr$ $rate(q') = rate(multiplier)$
Divide	• dividend • divider	$val(q') = \frac{val(dividend)}{val(divider)}$	$attr(q', a) = attr(divident, a), \forall a \in Attr$ $rate(q') = type(divider) \setminus type(dividend)$
Count	• entities	$val(q') = size(entities)$	$type(q') = commonClass(entities)$

Table 4: This table shows all the actions for inferring implied information and their affects.

Problem	**Output**
1. *Amy had 4 music files and 21 video files on her flash drive. If she deleted 23 of the files, how many files were still on her flash drive?*	Join$(4, 21)$ LossConcept$(25, 23, x)$
2. *Benny bought a soft drink for 2 dollars and 5 candy bars. He spent a total of 27 dollars. How much did each candy bar cost?*	Multiply$(5, x)$ PartWhole$(27, \{2, 5x\})$
3. *A company invited 18 people to a luncheon, but 12 of them didn't show up. If the tables they had held 3 people each, how many tables do they need?*	Separate$(18, 12)$ Unitary$(x, 3, 6)$
4. *Oscar's bus ride to school is 0.75 of a mile and Charlie's bus ride is 0.25 of a mile. How much longer is Oscar's bus ride than Charlie's?*	Compare$(0.75, 0.25, x)$
5. *After eating at the restaurant, Sally, Sam and Alyssa decided to divide the bill evenly. If each person paid 45 dollars, what was the total of the bill?*	Count({Sally,Sam,Alyssa}) Unitary$(3, 45, x)$
6. *Mika had 20 stickers. She bought 26 stickers from a store in the mall and gave 6 of the stickers to her sister. Then Mika got 20 stickers for her birthday. How many stickers does Mika have now?*	Increase$(20, 26)$ Decrease$(46, 6)$ GainConcept$(52, 20, x)$
7. *Fred has 90 cents in his bank. How many dimes does Fred have ?*	UnitChangeKnowledge(cents,dime) Unitary$(90, 0.1, x)$

Table 5: This table shows several arithmetic problems and a sample sequence of actions that can be performed to obtain the answer.

decide to count the number of people or might decide to use the unit change knowledge for *dozen*. To handle this scenario, each labeled data contains a word problem P, the set V_P and a set Y^* containing the correct possible applications $\{y_i\}$. Let us say, I_{y_i} is the indicator variable for the action y_i and $X = \bigcup_{y_i \in Y*} y_i$. Then the event $X = true$ denotes that the machine has decided to take one of the correct actions. We aim to maximize the probability of this event. Since I_{y_i}s are disjoint events, $P(X) = \sum_{y \in Y*} P(y)$.

We estimate θ by minimizing the negative of the conditional log-likelihood of the data:

$$O(\theta) = -\sum_{i=1}^{n} log\ p(X_i^*|P_i; \theta)$$

$$= -\sum_{i=1}^{n} [log \sum_{y \in Y_i^*} e^{\theta.\phi(P_i, \mathbb{V}_{P_i}, y)} - log \sum_{y \in A_{P_i}} e^{\theta.\phi(P_i, \mathbb{V}_{P_i}, y)}]$$

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

$$\frac{\nabla O}{\nabla \theta} = -\sum_{i=1}^{n} [\sum_{y \in Y_i^*} (\frac{e^{\theta}.\phi(P_i, \mathbb{V}_{P_i}, y)}{\sum_{y' \in Y_i^*} e^{\theta.\phi(P_i, \mathbb{V}_{P_i}, y')}})$$

$$\times \phi(P_i, \mathbb{V}_{P_i}, y) - \sum_{y \in A_{P_i}} p(y|P_i; \theta) \times \phi(P_i, \mathbb{V}_{P_i}, y)]$$

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.

The training data contains a correct sequence of actions for each problem which is then used for parameter estimation. If the labeled sequence for a problem P is $\langle y_1, y_2, ..., y_n \rangle$, then n $\langle input, output \rangle$ pairs are created for parameter estimation in the following way:

Algorithm 1: FindSequence

Data: Problem instance P $\&\mathbb{V}_P$
Result: A sequence of actions, $\hat{s}$

$\hat{s} = \langle\rangle$;
do
  $action = f(P, \mathbb{V}_P)$;
  $t = t + 1$;
  $\hat{s} =< \hat{s}, action >$;
  if *action is not a Formula* **then**
    $\mathbb{V}_P = \mathbb{V}_P \bigcup \{action.result\}$;
while *action is not a Formula or t = limit*;

Algorithm 2: CreateIOPairs

Data: Problem instance P , $\mathbb{V}_P$ & the label $L = \langle y_1, y_2, ..., y_n \rangle$
Result: A set of input-output pairs

$\hat{s} = \{\}$;
for $t = 1$ *to* n **do**
  $executable\_actions = \{y \in L :$ y *pertains to counting* **or** *unitary knowledge* **or** *all the quantities associated with* $y \in V_P\}$;
  $\hat{s} = \hat{s} \cup (P\&\mathbb{V}_P, executable\_actions)$;
  if y_t *is not a Formula* **then**
    $\mathbb{V}_P = \mathbb{V}_P \bigcup \{y_t.result\}$;
  $t = t + 1$;
end

3 Feature Function

The goal of the feature function is to gather enough information from the story so that the competing actions can be discriminated. 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 a series of NLP tools such as tokenizer, lemmatizer, part-of-speech tagger, syntactic parser, dependency parser (Manning et al. 2014). We first define a list of boolean variables which compute semantic relations between the attributes of each pair of quantities. Then we present an overview of the features using the description of the attributes and the boolean variables.

Cross Attribute Relations

Similar to (Mitra and Baral 2016) our system computes a set of boolean variables from a pair of quantity schema, each denoting whether the attribute a_1 of the quantity v_1 has the same value as the attribute a_2 of the variable q_2. The value of each attribute is a set of words, consequently set equality is used for comparison. For the *type* attribute it computes three more variables which are *subType* (tests subset relationship), *disjointType* (tests disjointness) and *intersectingType* (true if the two sets intersect) (Mitra and Baral 2016). 3 more cross attribute relations (Mitra and Baral 2016) are created which are defined as follows:

antonym: It is true if there exists a pair of word in $(v_1.verb \bigcup v1.adjective) \times (v_2.verb \bigcup v2.adjective)$ that are antonyms to each other according to WordNet.

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

subjConsume: It is an asymmetric relation and is true if the subject of the quantity $v1$ is a superset of the subject of v_2.

Features: Unitary & Multiply

$a \wedge b \wedge subj(q_r) = type(q_m)$
$a \wedge b \wedge vmod(q_r) = dobj(q_m)$
$a \wedge b \wedge vmod(q_r) = xcomp(q_m)$
$a \wedge b \wedge rate(q_r) = type(q_m)$
$a \wedge b \wedge$ "$times$" $\in type(q_m) \wedge subj_{=}(q_m, q_t)$
$a \wedge b \wedge prep\_in(q_r) = type(q_m)$
$a \wedge b \wedge prep\_on(q_r) = type(q_m)$
$a \wedge b \wedge rate(q_r) \subset type(q_m)$
$a \wedge subType(q_r, q_m) \wedge rate(q_r) \subset type(q_m)$
$a \wedge b \wedge rate(q_r) = type(q_m) \wedge isCounted(q_m)$

Table 6: This table describes various features for *unitary*(q_m, q_r, q_t) action. Here $a = \neg sameType(q_m, q_t)$, $b = sameType(q_r, q_t)$. Predicate $isCounted(q_m)$ is *true* if it is created by a *count* action.

Each *unitary* feature is a conjunction over the cross attribute relations and attributes of the quantity schema. This is true for the features of other actions as well. Table 6 describes various features for *unitary* action. For each such feature there exists another feature which checks if there exists another quantity that is not a part of any slot of the unitary application and still satisfies the three necessary conditions if it replaces any of the three quantities. Another feature checks if none of these features is true. Features for *multiply* action are obtained by removing the conditions that contains q_t from the body of each feature in Table 6. These features for *multiply* actions are also computed for *unitary* actions to make the learning of *unitary* and *multiply* action dependent.

Features: LossConcept, GainConcept, Increase & Decrease

Knowing that the features are looking for an instance of these change actions gives a lot of information about what to look for. One important property of these actions is that the verbs associated with the quantities in a *change* action are special. For example, the quantity in the *start* slot describes how many objects were initially there, thus is normally associated with 'be' verbs or tagged with special keywords such as 'initially', 'beginning'. Similarly the quantities in the *gain* or *loss* slot are associated with verbs such as *found, lost, bought*. The set of verbs for each slot is collected from the training data. Given an action y, the change features thus check whether the verbs associated with each slots belong to their respective verb list. They additionally check for type matches within the variables, subject or preposition matches and the order of the *start, change, end* verbs (events) in the

text. The *Increase & decrease* features are obtained by dropping some conditions from the body of the *LossConcept & GainConcept* features.

Features: PartWhole, Join & Separate

Each *part whole* feature aims to look for three things: 1) whether the numeric quantities in *parts* slot posses good qualities to be part of some quantity, 2) whether the quantity in the *whole* slot possess suitable qualities to represent whole and 3) whether the quantities in parts are a good fit to be the parts of the given whole. Some of the conditions that appear in the body of the features are: 1) the type of parts are equal 2) the type of the parts are equal to the type of the whole 3) the type of parts are hyponym of whole 3) the verb of the parts are equal 4) the verbs of the parts are relatedVerb 5) the verb of whole matches with the parts 6) the whole has special keywords such as 'all', 'total' in its schema. Similar to the *Multiply* feature, *Join* features are obtained by deleting all conditions that use the attributes of the *whole*. A subset of *part whole* features is reused for *separate* action detection.

Features: Counting

One feature checks whether there is a quantity whose type attribute matches with that of the counted quantity. Others check for *verb*, *nsubj* match between the set being counted and the quantity with relevant type attribute. Most often these features will be true. However in some cases there might be a need to count and due to lack of information sufficient number of other features might not be active allowing these features to indicate that the count action should be executed.

Features: Unit Change Knowledge

For a given unit change action *unitChangeKnowledge(fromUnit, toUnit)*, one feature checks whether there exists quantities with type or rate being equal to that of the *fromUnit* and *toUnit*. If this condition is true another feature checks whether *toUnit* is equal to the type of the unknown. A third feature checks if there exists a quantity q in $\mathbb{V}_P$ for which $rate(q) \cup type(q) = \{fromUnit, toUnit\}$. Finally, a fourth feature checks if the number of quantities with type matching *fromUnit* is more than that of the *toUnit*.

Features: Comparison

To score the action of a *comparison* formula, it uses the following features: 1) whether all the associated quantities have the same type and 2) whether there are comparative adjectives or 'than' in the quantity schema of the *diff* quantity. In case it has a comparative word (e.g. shorter), the lemma of that word is used as a feature. It also considers a window of size k on the right side of that comparative word and computes the overlap with the other two quantities in the 'large' and 'small' slot.

4 Related Work

Developing algorithms to solve arithmetic word problems is a long standing challenge in NLP (Feigenbaum and Feldman 1963). Early years saw systems that solve the word problems in a constrained domain by either limiting the input sentences to a fixed set of patterns (Bobrow 1964b; 1964a; Hinsley, Hayes, and Simon 1977) or by directly operating on a propositional representation (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, Dai, and Chen 2015) which can solve both arithmetic and algebraic word problems. Both algorithms try to map a word math problem to one of n possible 'equation template's, such as $ax+b = c$, by filing the empty slots a, b, c with numbers from the text. These n templates are collected from the training data. For example, from the problem "There are 9 students in the class and 81 tickets. If the tickets are divided equally among the students, how many does each student get?" and the equation "$9 * x = 81$", it will get the following template : '$a * x = b$. They implicitly assume that these templates will reoccur in the new examples which is a major drawback of these algorithms. Also none of these algorithms properly handle the use of missing unit conversion knowledge or counting and the number of quantities that can appear in a new problem is bounded by the equation templates.

The closest to our work are the ones in (Koncel-Kedziorski et al. 2015; Roy and Roth 2015). Both the algorithms try to map the math word problem to a suitable expression tree. However, the expression trees considered contain only the numbers specifically mentioned in the text. Thus these algorithms fail to solve the problems that use information from outside such as the unit conversion knowledge or requires counting. Also, these algorithms do not allow the reuse of a number in the parse tree, which is often needed in math problems from higher grades.

Work of (Mitra and Baral 2016) is also close to our work in the sense that they have used formulas to solve addition-subtraction arithmetic problems. Their method takes as input a simple addition-subtraction arithmetic problem and tries to apply a formula to solve the problem in a single step. And because of this, their method cannot solve any of the problems in Table 1. In this work, we have proposed a more general model that can do multi step computation to solve a problem and can seamlessly incorporate missing knowledge if necessary. In each step it might take any of the 13 actions where only four of them pertains to formulas. The use of formulas allowed us to work with variable number of quantities in a equation and that property is utilized in this research.

Also there has been some work on very specific types of word problems (Hosseini et al. 2014; Shi et al. 2015). We do not discuss those here due to space limitation. Finally, our work is also related to *semantic parsing* (Zelle and Mooney 1996; Zettlemoyer and Collins 2012), a task of mapping sentences to formal expressions. However most of the semantic parsers process single sentences whereas arithmetic problem solving requires the entire narrative to be considered together.

5 Experimental Evaluation

Dataset & Results

We evaluate our system on 4 standard datasets.

AddSub Dataset This dataset released by (Hosseini et al. 2014) consists of a total of 395 addition-subtraction arithmetic problems for third, fourth, and fifth graders. They have reported 3-fold cross validation, where each fold contains problems collected from different websites. This helped them to test the robustness and we follow the same setting. To train our model, we have used the annotation provided in (Mitra and Baral 2016).

SingleEQ Dataset This dataset (Koncel-Kedziorski et al. 2015) contains a total of 508 *general* arithmetic problems requiring multiple steps. We have annotated the problems of this dataset manually to train our system. The authors have performed 5-cross validation and have reported the average. We follow the same setting.

IL Dataset This dataset (Roy, Vieira, and Roth 2015) contains a total of 562 arithmetic problems involving all the four arithmetic operators. Each problem from this dataset can be solved in a single step. Due to lack of suitable annotation we have used this dataset only as test data. When trained on SINGLEEQ dataset our system solves 422 problems giving an accuracy of 75.1%. The state-of-the-art performance (Roy and Roth 2015) on this dataset is 74% (average of 5 cross validation).

CommonCore Dataset This dataset (Roy and Roth 2015) contains a total of 600 multi-step arithmetic problems, 100 for each of the following types: 1) Addition followed by Subtraction 2) Subtraction followed by Addition 3) Addition followed by Multiplication 4) Addition followed by Division 5) Subtraction followed by Multiplication and 6) Subtraction followed by Division. We have annotated this dataset. On 6-fold cross validation our system achieved an accuracy of 53.5% which is 8.3% more than the state-of-the-art (Roy and Roth 2015) that obtained an accuracy of 45.2%.

METHOD	ADD SUB	SINGLE EQ
(Hosseini et al. 2014)	77.7	48.0
(Kushman et al. 2014)	64.0	67.0
(Koncel-Kedziorski et al. 2015)	77.0	72.0
(Roy and Roth 2015)	78.0	-
(Mitra and Baral 2016)	86.07	-
Our System	87.08	80.7

Table 7: Comparison with existing systems on the accuracy of solving arithmetic problems on the ADD SUB and SINGLE EQ datasets.

Table 7 compares the performance of our system on ADDSUB and SINGLEEQ dataset. There is an increase of 8.7% in the accuracy of solving problems in SINGLEEQ dataset. One important factor behind this improvement is that the existing systems cannot solve problems where the equation uses numbers that are not mentioned in the text.

Error Analysis

Among all the problems 38% of the error occurs in the application of *Unitary* formula. Our system uses a set of simple patterns to extract the *rate* attribute of a numeric quantity; however the extraction fails sometimes resulting in an error. A majority of the error (45%) occurs in the application of the *Part Whole* formula. There are several ways to describe a part whole relationship which presents rigorous challenges for part whole relation extraction. One example of a part whole problem which our system fails to solve is "*There were 3409 pieces of candy in a jar. If 145 pieces were red and the rest were blue, how many were blue?*". Since no quantity schema captures the information that "the rest were blue" it fails to identify the correct relationship. Also unit conversion rates are not the only types of missing information. To solve the problem, "*532 people are watching a movie in a theater. The theater has 750 seats. How many seats are empty in the theater?*", it is important to know that one person normally acquires one seat in a theater. Our system does not have this knowledge and fails to solve this problem. It is part of our future work to devise mechanisms which can fetch knowledge of this kind on the fly.

6 Conclusion

While a human being solves a math problem, she takes into account various missing knowledge which is necessary to characterize the solution. She can seamlessly solve a problem even if it needs a unit conversion and the unit conversion rate is missing. And the problems are also created keeping this phenomenon in mind. Math problems will often miss out information which is supposed to be part of every students' knowledge base. This is true for other domains as well. To solve a problem in classical mechanics, one needs to use Newton's formulas, which will not be provided with the problem. To solve a problem from Geography, one might need to use the fact that one degree change in longitude results in a difference of four minutes, which she is supposed to remember. Also, another important feature of the math problems is that to solve a problem one often needs involved thinking and needs to device a plan. Existing solvers for word math problems largely ignore these facts and aim to derive the equation in a single step. In this work, we have proposed a solver that generates a plan for solving a given arithmetic problem and can seamlessly use missing knowledge when needed. However, the errors show that, even though our system has attained state-of-the-art results, some more actions for adding missing knowledge and better representation of the problem is necessary. Our future work is to handle this issue.

References

Agirre, E.; Diab, M.; Cer, D.; and Gonzalez-Agirre, A. 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*, 385–393. Association for Computational Linguistics.

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

Bobrow, D. G. 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 I), 591–614. New York, NY, USA: ACM.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

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

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

Dagan, I.; Dolan, B.; Magnini, B.; and Roth, D. 2010. Recognizing textual entailment: Rational, evaluation and approaches–erratum. *Natural Language Engineering* 16(01):105–105.

De Marneffe, M.-C., and Manning, C. D. 2008. Stanford typed dependencies manual. Technical report, Technical report, Stanford University.

Feigenbaum, E. A., and Feldman, J. 1963. Computers and thought.

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

Hinsley, D. A.; Hayes, J. R.; and Simon, H. A. 1977. From words to equations: Meaning and representation in algebra word problems. *Cognitive processes in comprehension* 329.

Hosseini, M. J.; Hajishirzi, H.; Etzioni, O.; and Kushman, N. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 523–533.

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

Koncel-Kedziorski, R.; Hajishirzi, H.; Sabharwal, A.; Etzioni, O.; and Ang, S. D. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics* 3:585–597.

Kushman, N.; Artzi, Y.; Zettlemoyer, L.; and Barzilay, R. 2014. Learning to automatically solve algebra word problems. Association for Computational Linguistics.

Levesque, H. J. 2011. The winograd schema challenge.

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

Manning, C. D.; Surdeanu, M.; Bauer, J.; Finkel, J. R.; Bethard, S.; and McClosky, D. 2014. The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, 55–60.

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

Mitra, A., and Baral, C. 2016. Learning to use formulas to solve simple arithmetic problems. ACL.

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

Polya, G. 1957. *How to Solve it: A New Aspects of Mathematical Methods*. Prentice University Press.

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Richardson, M.; Burges, C. J.; and Renshaw, E. 2013. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*, volume 1, 2.

Roy, S., and Roth, D. 2015. Solving general arithmetic word problems. EMNLP.

Roy, S.; Vieira, T.; and Roth, D. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics* 3:1–13.

Shi, S.; Wang, Y.; Lin, C.-Y.; Liu, X.; and Rui, Y. 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*.

Weston, J.; Bordes, A.; Chopra, S.; and Mikolov, T. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

Zelle, J. M., and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, 1050–1055.

Zettlemoyer, L. S., and Collins, M. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*.

Zhou, L.; Dai, S.; and Chen, L. 2015. Learn to solve algebra word problems using quadratic programming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 817–822.