Natural Language Processing for Solving Arithmetic Word Problems

M. Uday Theja[1][CS14B044] and S. Sai Teja Reddy[2][CS14B051]

[1] Indian Institue of Technology Madras, Chennai 600036, Tamil Nadu, India
uday@cse.iitm.ac.in
[2] Indian Institue of Technology Madras, Chennai 600036, Tamil Nadu, India
saiteja@cse.iitm.ac.in

Abstract. In this project we implement a system which can solve simple arithmetic word problems. The system is given an arithmetic word problem in natural language as input. A temporal state representation is extracted from the question and equations are built from it. These equations are solved and the answer is displayed. The Natural Language Processing techniques used to extract the temporal state representation are described in detail in this paper.

Keywords: Arithmetic word problems · Natural language processing · Verb categorization · Dependency parsing.

1 Introduction

In this project we aim to solve simple arithmetic word problems given in natural language involving addition/subtraction. Designing such an algorithm is not a trivial problem as will be discussed later in the paper. The word problem given to us can be visualized as a sequence of partial world states,i.e. each sentence provides partial information about the world state usually in a time sequenced manner. We extract this information and use it to build equations which are solved to get the final answer.The input and output to our system are as illustrated below,

Input: `Mike had 34 peaches at his roadside fruit dish. He went to the orchard and picked peaches to stock up.There are now 86 peaches. How many did he pick?`

Output: `52`

The system makes use of verb categorization to understand what each action does. To illustrate, in the given example, the verb `picked` implies that the number of peaches in the orchard has decreased while the number of peaches with Mike has increased. This might seem trivial to understand but it must be noted that the system has no such world knowledge. The steps used by the system are described in detail in later sections.

For evaluation purposes, the datasets provided by (Hosseini et al., 2014) are used.

2 Arithmetic Problem Representation

We split the problem text into fragments which are simplified sentences. The information in these fragments is captured through world states and transitions between world states. The world state is represented as a tuple <E,C,R> consisting of Entities E, Containers C and Relations R.

For example consider the problem text, `"Joan found 70 blue seashells and 30 red seashells on the beach . She gave few blue seashells to Sam. She has 27 blue seashell left . How many blue seashells did she give to Sam ?"`

2.1 Entities

An *entity* is the object in the problem text whose quantity is being observed or is changing throughout the problem. In the above example, `seashell` is the entity.

2.2 Attributes

An entity is further differentiated from other entities using *attributes* which are adjectives or noun modifiers of an entity. In the above example, there are 2 types of seashells, `blue seashell` and `red seashell`

2.3 Containers

A *container* is generally the mention in the text possessing the entity or a location containing the entity. In the above example, `Joan` and `Sam` are the containers.

2.4 Quantities

Every container-entity pair is associated with a numerical *quantity*. This quantity can either be a known number or a variable or an arithmetic expression and is denoted as `N(container,entity)`. In the above example, for the first fragment, `N(Joan,seashell[blue]) = 70` and `(Joan,seashell[red]) = 30`.

3 Architecture

The system solves problems using the following steps:

- Preprocessing sentences
- Extracting state representation
- Building and solving equations

These steps are elaborated upon in the following section.

4 Preprocessing sentences

In this step the system breaks the given word problem into constituent *simple sentences*. A *simple sentence* has a single entity, atmost two containers and a single verb. This decomposition is done using the following steps:

4.1 Resolving Conjunctions

Sentences containing `and`,`but`,`if` and `then` are split into constituent sentences,

- The sentence is split at the conjunction into two parts Part1 and Part2
- Each part is assumed to have four subparts - preverb(PV), verb(V), after-verb(AV) and preposition phrase(PP).
- Any subpart from Part2 which is empty is populated with the corresponding subpart from Part1 and vice versa.
- The four subparts are concatenated into a sentence. The original sentence is replaced by the two new sentences.

Consider `There are 11 rulers and 34 crayons in the drawer`

```
[
    { 'Part1': "There are 11 rulers",'PV': "There", 'V': "are",
    ↪  'AV': "11 rulers", 'PP': "" } ,
    { 'Part2': "34 crayons in the drawer" , 'PV': "", 'V': "",
    ↪  'AV': "34 crayons", 'PP': "in the drawer" }
]
```

After populating,

```
[
    { 'Part1': "There are 11 rulers", 'PV': "There", 'V': "are",
    ↪  'AV': "11 rulers", 'PP': "in the drawer" } ,
    { 'Part2': "34 crayons in the drawer", 'PV': "There", 'V':
    ↪  "are", 'AV': "34 crayons", 'PP': "in the drawer" }
]
```

The original sentence `There are 11 rulers and 34 crayons in the drawer` is replaced by `There are 11 rulers in the drawer.There are 34 crayons in the drawer.`

4.2 Preprocessing Currencies

Whenever we see text of the type `$300` we replace it with `300 dollars`. This is done so that the dependency parser correctly parses the sentence.

4.3 Coreference Resolution

This step is used to replace pronouns with their referring entities.
For example,
`Joan found 70 seashells on the beach. She gave few seashells to Sam.`
is replaced by `Joan found 70 seashells on the beach. Joan gave few seashells to Sam.`
One additional heuristic used is that `They` and `Their` are ignored. This is because coreference resolution for these pronouns almost always fails.

Preprocessing example

Input: `"Joan found 70 blue seashells and 30 red seashells on the beach . She gave few blue seashells to Sam. She has 27 blue seashell left . How many blue seashells did she give to Sam ?"`

Output: `"Joan found 70 blue seashells on the beach. Joan found 30 red seashells on the beach . Joan gave few blue seashells to Sam. Joan has 27 blue seashell left . How many blue seashells did Joan give to Sam ?"`

5 Verb Categorization

We have previously mentioned that our individual sentence fragments are preprocessed such that there is one entity,one verb and atmost two containers. We notice that the verb performs some kind of transaction between the two containers in terms of the entity. Based on the transaction, we identify 7 categories for verbs,

- ***Observation*** quantity of entity is intitialized by this verb. For example, `Sam had 100 dollars.`
- ***Positive*** quantity of entity is only increased in the first container by this verb. For example, `Sam found 100 dollars.`
- ***Negative*** quantity of entity is only decreases in the first container by this verb. For example, `Sam lost 100 dollars.`
- ***Positive Transfer*** quantity of entity is transferred from second container to first container by this verb. For example, `Sam borrowed 100 dollars from Jack.`
- ***Negative Transfer*** quantity of entity is transferred from first container to second by this verb. For example, `Sam gave 100 dollars to Jack.`
- ***Construct*** quantity of entity is increased in both containers by this verb. For example, `The farmer produced 100kg rice for the village.`
- ***Destroy*** quantity of entity is decreased in both containers by this verb. For example, `The gardener cut the weeds in the park.`

For generating a corpus of category labelled verbs, we first take the set of 119 manually categorized verbs provided by (Hosseini et al., 2014). For each verb we take the synsets of it from WordNet and label their lemmas with the same category as the original verb. Using this method we were able a generate a corpus of 1250 labelled verbs.

6 Extracting state representation

6.1 Identifying entities and containers

After preprocessing the text, we obtain a sequence of sentence fragments $< w_1, ..., w_T, w_x >$. Each sentence in the output of the preprocessed text is a fragment. Every w_t has a an entity e_t , its attribute a_t, quantity num_t, verb v_t and atmost two containers c_{t1} and c_{t2}. w_x is the fragment containing the question sentence.

Entites are the objects whose quantity is observed or is changing throughout the problem. So we first extract set h of all the nouns which are dependant on a number. We then extract all the noun phrases and define e_t as the set of NP headed by a noun in h. From the previous example,

```
{
    'h' : { "seashell" } ,
    'e_t' : ["70 blue seashell","30 red seashell","few blue
    ↪  seashell","27 blue seashell"]
}
```

Attributes are the adjective or noun modifiers of the entities. These are extracted from the dependency parse tree. If not found, they are assigned the previously observed values. From the previous example,

```
[
    { 'e_t' : "70 blue seashell" , 'a_t' : "blue" } ,
    { 'e_t' : "30 red seashell" , 'a_t' : "red" }
]
```

Containers Each entity is associated with one or more containers. There are atmost 2 containers c_{t1} and c_{t2} in a fragment. c_{t1} is generally the subject of the fragment but if w_t contains *There is/are* and preposition like *in* then we choose the adverb of place or the location depicted by the preposition. c_{t2} is generally the object of the verb. To handle this missing information in problem text, we use the **Circumscription Assumption (McCarthy, 1980)**. The circumscription assumption states that unless things are changed explicitly, they are assumed to be be the same. So, for a e_t if the c_t cannot be identified, then it is set to previously observed c_t for the previous e_t with the same head word h. From the previous example,

```
{
    'w_1' : "Joan found 70 blue seashells on the beach." ,
    'c_t1': "Joan", 'c_t2': "beach"
}
```

Quantities are the dependent number of e_t if exists. Bare numbers which are present in *numbers* and not associated with any e_t yet are associated with the nearest entity in the text. All other num_t are set to unknown and are assigned variables. Thus, quantites can either be a known number or a variable or an arithmetic expression and is denoted as `N(container,entity)`. From the previous example,

```
{
    'w_1' : "Joan found 70 blue seashells on the beach." ,
    'N("Joan",{"E":"seashell","A":"blue"})' : '70'
}
```

Pseudo Code

- Extract all *numbers* and noun phrases (NP).
- $h \leftarrow$ set of all nouns which are dependent on a number.
- $e_t \leftarrow$ all NP which are headed by a noun in h.
- $num_t \leftarrow$ the dependent number of e_t if exists. Bare numbers which are present in *numbers* and not associated with any e_t yet are associated with the nearest entity in the text. All other num_t are set to unknown.
- $a_t \leftarrow$ adjective and noun modifiers of e_t. If not found, are assigned the previously observed attribute for that e_t.
- $v_t \leftarrow$ the verb with the shortest path to e_t in the dependency parse tree.
- c_t1 , $c_t2 \leftarrow$ the containers for entity e_t. If not found, set to previous values by circumscription assumption

6.2 Verb Categorization

For each fragment w_t, the verb v_t is categorized into one of the 7 possible categories as mentioned in section 5. We first lemmatize the verb v_t to get the base form of the verb and then fetch its verb category. In the above example, the verbs identified and their categories are :

```
{
    "found" : "POSITIVE",
    "gave" : "NEGATIVE TRANSFER",
    "has" : "OBSERVATION"
}
```

6.3 State Progression

Now for each fragment w_t we have the required data about e_t, a_t, num_t, c_{t1}, c_{t2} and also the verb category for the verb v_t. Using this data, for each fragment in $< w_1, ..., w_T >$, we build corresponding states $< s_1, ..., s_T >$ and $s_0 \leftarrow null$. For building state s_{t+1}, we first copy all the contents of state s_t and then create or append a new container-entity if previously not present , of the form **{**c_t **:** [**{**e_t **:** , a_t **:** , num_t **:** **}**]**}** else if it is already present then we update the existing contents num_t using the verb category of v_t, for container-entity pairs `(`c_{t1}`,`e_t`)` and `(`c_{t2}`,`e_t`)` and quantity num_t as follows :

- ***Observation :*** `N(`c_{t1}`,`e_t`) =` num_t
- ***Positive :*** `N(`c_{t1}`,`e_t`) +=` num_t
- ***Negative :*** `N(`c_{t1}`,`e_t`) -=` num_t
- ***Positive Transfer :*** `N(`c_{t1}`,`e_t`) +=` num_t and `N(`c_{t2}`,`e_t`) -=` num_t
- ***Negative Transfer :*** `N(`c_{t1}`,`e_t`) -=` num_t and `N(`c_{t2}`,`e_t`) +=` num_t
- ***Construct :*** `N(`c_{t1}`,`e_t`) +=` num_t and `N(`c_{t2}`,`e_t`) +=` num_t
- ***Destroy :*** `N(`c_{t1}`,`e_t`) -=` num_t and `N(`c_{t2}`,`e_t`) -=` num_t

When we are creating a new entry in the state and the verb category is not *Observation*, then its quantity $N(c, e)$ is initialized with an unknown before performing the operation.
In the above example, the states built are :

```
[
    {   'joan': [{'A': 'blue', 'E': 'seashell', 'N': 'J0+70'}]  }
    {
        'sam': [{'A': 'blue', 'E': 'seashell', 'N': 'J00+$0'}] ,
        'joan': [{'A': 'blue', 'E': 'seashell', 'N': 'J0+70-$0'}]
    }
    {
        'sam': [{'A': 'blue', 'E': 'seashell', 'N': 'J00+$0'}] ,
        'joan': [{'A': 'blue', 'E': 'seashell', 'N': '27'}]
    }
]
```

7 Building and Solving Equations

Once we get the world states we must now build relevant equations based on the question asked, w_x and solve for the answer. For finding the solution, we only consider the states and fragments which consist of the entity e_x mentioned in the question sentence w_x. For instance, in the above example e_x `= blue seashell` thus we can ignore the states and fragments with e_t `= red seashell`. By progressing through all the relevant states, an equation is formed when num_t of state s_i consists of an unknown and num_t of state s_{i+1} is a known number and all such equations are solved to find the values of the unknowns.

Once the equations are solved and the values of unknowns are used, we need to pick the variables which contribute to the final answer as follows :

- If the verb category of v_x is *Observation* and w_x contains `"begin"` or `"start"` then the answer is the num_t of the first occurrence of e_t in the states.
- Else we attempt to pick the fragments whose verb v_t is similar to v_x and has the same container pairs and add the variable introduced in that fragment to our answer.
- Else we attempt to pick the fragments whose verb v_t is opposite to v_x and has the opposite container pairs and add the variable introduced in that fragment to our answer.
- Else, finally, we pick the final value of the entity e_x in the final state s_T and assign its quantity num_t to the answer.

While evaluating the answer, if the irrelevant initializer values are set to `0`.
In the above example, the equations formed are:

```
[ 'J0+70-$0=27' ]
```

and the solutions obtained are:

```
{   'J0': '0', '$0': '43'   }
```

8 Evaluation and Results

The performance of our system is evaluated on the dataset provided by (Hosseini et al., 2014). The performance is compared against two other systems. Thay are WolframAlpha's Full Results API and Roy Roth Illinois' interface.Wolfram Alpha's system expects simple and straightforward queries.Hence, this system can be considered as a Baseline for evaluation.

Roy-Roth Ilinois	95.03%
Our System	46.81%
Wolfram Alpha	4.25%

9 Error Analysis

Some of the cases where our model fails to solve the problem text are:

Resolving Conjunctions: Consider the sentence, `"Jason has 43 blue and 16 red marbles"`. The output of our preprocessing step gives `"Jason has 43 blue. Jason has 16 red marbles."` and so the first fragment is missing the entity `"marble"`. Our algorithm expects each fragment to have an entity and so fails in this case.

Decimal Numbers: Consider the sentence, `"Joan purchased a basketball game for $ 5.20 , and a racing game for $ 4.23 ."`. When we split this into component sentences we see that the system doesnot understand the difference between a full stop and a decimal point. Hence it also splits the sentence at the decimal point leading to unexpected results.

Irrelevant Sentences: Consider the problem text, `"Joan decided to sell all of her old books . She gathered up 33 books to sell . She sold 26 books in a yard sale . How many books does Joan now have ?"`. Here `"Joan decided to sell all of her old books ."` is irrelevant. But the system categorizes `sell` as Negative Transfer and creates a state for it. THis leads to an incorrect final answer.

World Knowledge: Consider the sentence, `"Sara 's high school played 12 basketball games this year. The team won most of their games ."`. Here, `"Sara's high school"` is detected as a container in first fragment and `"team"` is detected as a separate container from second fragment. But the system should also understand that both refer to the same container.

Coreference Resolution: Consider the problem text, `"Jason had 49 quarters in his bank . His dad gave him 25 quarters . How many quarters does he have now ? "`. Here, there is inherent ambiguity in the question. The `he` can refer to either `Jason` or `His dad`.

Verb Categorization: If the category of the detected verb is incorrect, the quantities in the states are identified incorrectly which ultimately leads to an incorrect answer.

Container Detection: Consider the problem text, `"Melanie picked 4 plums . Dan picked 9 plums and Sally picked 3 plums from the plum tree . How many plums were picked in total ?"`. Here, in the last fragment, `"How many plums were picked in total ?"` we detect no container which actually refers to `"Melanie + Dan + Sally"`

Attribute Detection: Consider the problem text, `"Mike has 35 books in his library . He bought several books at a yard sale over the weekend . He now has 56 books in his library . How many books did he buy at the yard sale ?"` . Our system recognizes `several` as an attribute. Once we see this attribute we assign it to all further instances of this entity with no attribute. Therefore two different entities are created, `several books` and `books` when in fact they are one and the same.

Verb Similarity: Consider the problem text, `"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 ?"`. Here, the lemmatized verb in all the fragments except the question fragment w_x is `"go"` whereas the lemmatized verb in the question fragment w_x is `"attend"` and thus it is difficult to mark fragments which are associated with w_x.

10 Discussions and Conclusion

In this project we implement a system to solve arithmetic word problems. The system builds a temporal state representation from the given problem and uses it to answer the question asked.
It is difficult to implement an algorithm which solves all different kinds of problems. We generalize the algorithm to cover as many models of word problems as possible. More rules can be added to extend the system to other word problems. For verb categorization we use the baseline approach i.e. using WordNet for categorization. Instead, training a machine learning model with the features extracted from the sentences to categorize verbs will improve accuracy.Another way to improve accuracy would be to use better dependency parsing and coreference resolution.

References

1. Mohammad Javad Hosseini, Hannaneh Hajishirzi,Oren Etzioni, and Nate Kushman 2014. Learning to Solve Arithmetic Word Problems with Verb Categorization. Proceedings of the Conference on Empirical Methods in Natural Language Processing.
2. Sowmya S Sundaram, Deepak Khemani, Natural Language Processing for Solving Simple Word Problems
3. Christopher D. Manning, Surdeanu, Mihai, Bauer, John, Finkel, Jenny, Bethard, Steven J., and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60
4. WolframAlpha Full Results API https://api.wolframalpha.com/v2/query?
5. Roy Roth Illinois' Web Interface http://cogcomp.org/page/demo_view/Math