`Quantity Tagger`: A Latent-Variable Sequence Labeling Approach to Solving Addition-Subtraction Word Problems

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

Abstract

An arithmetic word problem typically includes a textual description containing several constant quantities. The key to solving the problem is to reveal the underlying mathematical relations (such as addition and subtraction) among quantities, and then generate equations to find solutions. This work presents a novel approach, *Quantity Tagger*, that automatically discovers such hidden relations by tagging each quantity with a *sign* corresponding to one type of mathematical operation. For each quantity, we assume there exists a latent, variable-sized *quantity span* surrounding the quantity token in the text, which conveys information useful for determining its sign. Empirical results show that our method achieves 5 and 8 points of accuracy gains on two datasets respectively, compared to prior approaches.

1 Introduction

Teaching machines to automatically solve arithmetic word problems, exemplified by two problems in Figure 1, is a long-standing Artificial Intelligence (AI) task (Bobrow, 1964; Mukherjee and Garain, 2008). Recent research (Hosseini et al., 2014; Kushman et al., 2014; Roy and Roth, 2015; Wang et al., 2017, 2018b,a) focused on designing algorithms to automatically solve arithmetic word problems. One line of prior works designed rules (Mukherjee and Garain, 2008; Hosseini et al., 2014) or templates (Kushman et al., 2014; Zhou et al., 2015; Mitra and Baral, 2016) to map problems to expressions, where rules or templates are collected from training data.

However, it would be non-trivial and expensive to acquire a general set of rules or templates. Furthermore, such approaches typically require additional annotations. The *addition-subtraction* problems, which constitute the most fundamental class of arithmetic word problems, have been the focus for many previous works (Hosseini et al., 2014; Mitra and Baral, 2016). We also focus on this important task in this work. Our key observation is that essentially solving such a class of problems can be tackled from a sequence labeling perspective. This motivates us to build a novel sequence labeling approach, namely *Quantity Tagger*. The approach tags each quantity in the text with a label that indicates a specific mathematical operation.

Problem 1: *A worker at a medical lab is studying blood samples.* **2** *samples contained a total of* **7341** *blood cells. The first sample contained* **4221** *blood cells. How many blood cells were in the second sample?*
Prediction: $(\mathbb{0})\times\mathbf{2}+(+\mathbb{1})\times\mathbf{7341}+(-\mathbb{1})\times\mathbf{4221}+(-\mathbb{1})\times x=0$
Equation: $7341-4221-x=0$
Solution: $x=3120$

Problem 2: *There are* **22** *walnut trees currently in the park. Park workers will plant walnut trees today. When the workers are finished there will be* **55** *walnut trees in the park. How many walnut trees did the workers plant today?*
Prediction: $(+\mathbb{1})\times\mathbf{22}+(-\mathbb{1})\times\mathbf{55}+(+\mathbb{1})\times x=0$
Equation: $22-55+x=0$
Solution: $x=33$

Figure 1: Two examples of arithmetic word problems described in English with answers.

Taking Problem 1 from Figure 1 as an example, three constant quantities "2","7341" and "4221" sequentially appear in the problem text. We further introduce an unknown quantity x corresponding to the question sentence. From the problem description, one can form an equation "$7341-4221-x=0$", based on which we can obtain the solution to x. This equation is mathematically equivalent to "$\mathbb{0}\times 2+(+\mathbb{1})\times 7341+(-\mathbb{1})\times 4221+(-\mathbb{1})\times x=0$" where "$\mathbb{0},+\mathbb{1},-\mathbb{1},-\mathbb{1}$" are *sign*s associated with the quantities "$2, 7341, 4221, x$".

Solving arithmetic word problem can thus be casted as a sequence labeling problem where we assign every quantity appearing in the problem text a sign (in the form of a *tag*) from the set $\{+\mathbb{1},\mathbb{0},-\mathbb{1}\}$. We further assume there exists a latent *quantity span* that needs to be learned – a sequence of words surrounding each quantity, based

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 5246–5251
Florence, Italy, July 28 - August 2, 2019.

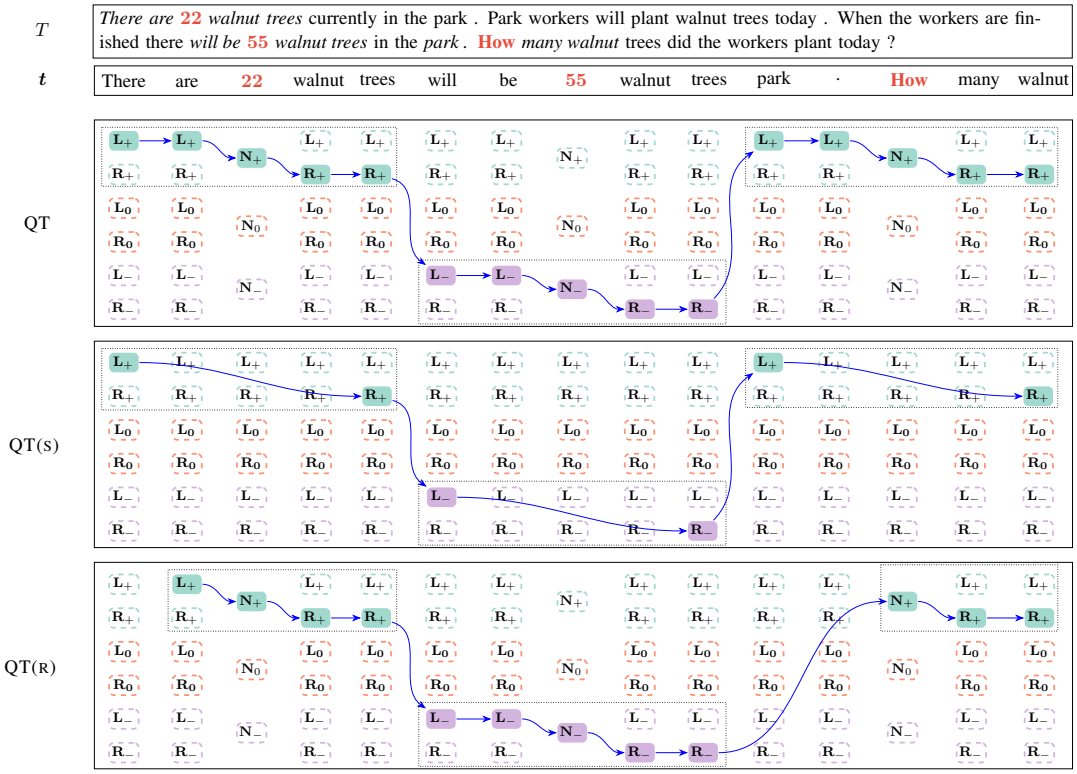

Figure 2: Illustrations of assumptions made by QT, QT(S) and QT(R), with possible paths (selected nodes are highlighted) built for the token sequence $\boldsymbol{t}$ (J=3), consisting of words from the original problem text T.

on which tagging decisions could be made.

We demonstrate through experiments on benchmark data that, despite its relatively simple assumptions involved, our novel sequence labeling approach is able to yield significantly better results than various state-of-the-art models. To the best of our knowledge, this is the first work that tackles the problem from a sequence labeling perspective. Our code is publicly available at https://github.com/zoezou2015/quantity_tagger.

2 Our Approach

2.1 A Tagging Problem

We define $\mathbf{Q} = (q_1, q_2, \ldots, q_i, x, q_{i+1}, \cdots q_m)$ ($0<i<m$, $m\geq 2$ in arithmetic word problems) as an ordered quantity sequence for a problem text T, where $q_i \in \mathbf{Q}$ represents a *constant quantity* appearing in T, and x stands for the *unknown quantity* assigned to the question sentence. $\mathbf{Q}$ maintains the same order as the quantities appearing in T. The goal is to construct a valid math equation E. This research investigates such a problem by sequentially tagging each quantity $q \in \mathbf{Q}$ with the most likely *sign* from set $\mathcal{S} = \{+\mathbb{1}, \mathbb{0}, -\mathbb{1}\}$, where "$+(-)\mathbb{1}$" means a quantity is positively (negatively) related to the question, i.e., the sign of the quantity should be +(-) when forming part of the equation; "$\mathbb{0}$" means a quantity is irrelevant to the question and should be ignored.

Given a specific prediction of the signs to the quantities, we can form an equation as follows:

$$\sum_{q_i \in \mathbf{Q}/\{x\}} s_i q_i + s_x x = 0 \qquad (1)$$

where $s_i \in \{+\mathbb{1}, \mathbb{0}, -\mathbb{1}\}$ is the sign for the i-th constant quantity q_i, and $s_x \in \{+\mathbb{1}, -\mathbb{1}\}$ is the sign for x. The solution can be easily obtained.

2.2 Quantity Tagger

Our primary assumption is that, for each quantity, there exists an implicit *quantity span* that resides in the problem text and can convey relevant information useful for determining the signs of the quantities. The quantity span of a quantity is essentially a contiguous token sequence from the problem text that consists of the quantity itself and some surrounding word tokens.

Formally, our model needs to learn how to sequentially assign each *quantity* $q \in \mathbf{Q}$ its optimal sign $s \in \mathcal{S}$. This is a sequence labeling problem (Lample et al., 2016; Zou and Lu, 2019). Common sequence labeling tasks, such as NER and POS tagging, mainly consider one sentence at a

time, and tag each token in the sentence. However, our tagging problem typically involves multiple sentences where relatively unimportant information may be potentially included. For instance, the second sentence of Problem 2 in Figure 1, "*Park workers will plant walnut trees today*" describes background knowledge of the problem, but such information may not be useful for solving problems, yet even obstructive.

For each quantity $q \in \mathbf{Q}$, we first consider a token window consisting of q and $J-1$ surrounding tokens located immediately to the left and right of q. This gives us a window of word tokens in the size of $2J-1$. Next, such token windows for all quantities in $\mathbf{Q}$ are merged to form a new token sequence, denoted as $\boldsymbol{t}$. Note that $\boldsymbol{t}$ is formed by concatenating token subsequences taken from T and is in the length of n ($1 \leq n \leq N$, where N is the length of T). We assume the quantity spans are defined over such a token sequence $\boldsymbol{t}$ (rather than T), which we believe convey most relevant information for determining the signs for the quantities. Exemplified by Problem 2 in Figure 1, we show an example token sequence $\boldsymbol{t}$ with $J=3$ in Figure 2.

To capture quantity span information, we design 9 different labels with different semantics: $\mathcal{H}$=$\{\mathbf{L}_+, \mathbf{L_0}, \mathbf{L}_-; \mathbf{N}_+, \mathbf{N_0}, \mathbf{N}_-; \mathbf{R}_+, \mathbf{R_0}, \mathbf{R}_-\}$.

- The $\mathbf{N}$ nodes are used to indicate that the current token is a quantity.
- The $\mathbf{L}$ ($\mathbf{R}$) nodes are used to indicate that the current token appears within a quantity span of a given quantity but to the left (right) of the quantity.

The subscripts "$+$", "$\mathbf{0}$", and "$-$" are used to denote the sign ($+\mathbb{1}$, $\mathbb{0}$ and $-\mathbb{1}$ respectively) associated with the quantities (and quantity spans).

All quantities are explicitly given in the problem text. Therefore, the $\mathbf{N}$ node is used to tag a word token if and only if the token represents a quantity. Otherwise, $\mathbf{L}$ and $\mathbf{R}$ nodes are considered. Furthermore, the unknown quantity is always relevant to the problem. We thus tag it with either $\mathbf{N}_+$ or $\mathbf{N}_-$, while three types of $\mathbf{N}$ nodes are for all constant quantities. As illustrated in Figure 2, only one node from $\mathcal{H}$ will be selected at each position. Sequentially connecting all such nodes will form a single path that reveals information about quantity spans selected for all quantities.

Following CRF (Lafferty et al., 2001), we formulate our method as a log-linear model with latent variables. Formally, given the problem text T, let $\boldsymbol{t} = (t_1, t_2, \ldots, t_n)$ be a token sequence as defined above, $\boldsymbol{y}$ be the corresponding label sequence, and $\boldsymbol{h}$ be a latent variable that provides specific quantity span information for the $(\boldsymbol{t}, \boldsymbol{y})$ tuple, we define:

$$p(\boldsymbol{y}|\boldsymbol{t}) = \frac{\sum_{\mathbf{h}} \exp(\mathbf{w}^T \mathbf{f}(\boldsymbol{t}, \boldsymbol{y}, \boldsymbol{h}))}{\sum_{\boldsymbol{y'}, \boldsymbol{h'}} \exp(\mathbf{w}^T \mathbf{f}(\boldsymbol{t}, \boldsymbol{y'}, \boldsymbol{h'}))} \quad (2)$$

where $\mathbf{w}$ is the feature weight vector, i.e., model parameters, and $\mathbf{f}$ is the feature vector defined over the triple $(\boldsymbol{t}, \boldsymbol{y}, \boldsymbol{h})$, $\mathbf{f}(t, \boldsymbol{y}, \boldsymbol{h})$ returns a list of discrete features (refer to supplementary materials).

During training, we would like to minimize the negative log-likelihood of the training set:

$$\mathcal{L}(\mathbf{w}) = \sum_i \log \sum_{\boldsymbol{y'}, \boldsymbol{h'}} \exp\left(\mathbf{w}_f^T \mathbf{f}(\boldsymbol{t}^{(i)}, \boldsymbol{y'}, \boldsymbol{h'})\right) - \sum_i \log \sum_{\boldsymbol{h}} \exp\left(\mathbf{w}_f^T \mathbf{f}(\boldsymbol{t}^{(i)}, \boldsymbol{y}^{(i)}, \boldsymbol{h})\right) \quad (3)$$

where the $(\boldsymbol{t}^{(i)}, \boldsymbol{y}^{(i)})$ is the i-th training instance. The standard gradient-based methods can be used to optimize the above objective, such as L-BFGS (Liu and Nocedal, 1989). Gradients of the above function is given by:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_k} = \sum_i \mathbf{E}_{p(\boldsymbol{y'}, \boldsymbol{h}|\boldsymbol{t}^{(i)})}[f_k(\boldsymbol{t}^{(i)}, \boldsymbol{y'}, \boldsymbol{h})] - \sum_i \mathbf{E}_{p(\boldsymbol{h}|\boldsymbol{t}^{(i)}, \boldsymbol{y}^{(i)})}[f_k(\boldsymbol{t}^{(i)}, \boldsymbol{y}^{(i)}, \boldsymbol{h})] \quad (4)$$

where $\mathbf{E}_p[\cdot]$ is the expectation under distribution p.

We can construct a lattice representation on top of the nodes shown in Figure 2. The representation compactly encodes exponentially many paths, where each path corresponds to one possible label sequence. Note that there exists a topological ordering amongst all nodes. This allows us to apply a generalized forward-backward algorithm to perform exact marginal inference so as to calculate both objective and expectation values efficiently (Li and Lu, 2017; Zou and Lu, 2018). The MAP inference procedure can be done analogously, which is called during the decoding time.

2.3 Model Variants

We further consider two variants of our model. **Semi-Markov Variant**: Our first variant, namely QT(S), employs the semi-Markov assumption (Sarawagi and Cohen, 2005), where $\mathbf{N}$ nodes are removed. Different from QT which makes the first-order Markov assumption, QT(S) assumes $\mathbf{L}$

Model	AddSub	AS_CN
Hosseini et al. (2014)	77.70	-
Kushman et al. (2014)	64.00	-
Koncel-Kedziorski et al. (2015)	77.00	-
Roy and Roth (2015)	78.00	47.57
Zhou et al. (2015)	53.14	51.48
Mitra and Baral (2016)	86.07	-
Roy and Roth (2017)	60.99	47.71
Wang et al. (2017)	-	20.64
Wang et al. (2018b)	78.50	-
QT(FIX)	87.73	53.19
QT	**90.79**	58.72
QT(S)	87.30	54.81
QT(R)	88.69	**59.10**
QT(-EF)	60.44	56.53
QT(S-EF)	63.49	52.62
QT(R-EF)	67.52	57.48

Table 1: Accuracy (%) on AddSub and AS_CN. -EF: without external features.

and **R** nodes are used to indicate the left and right boundaries of a quantity span respectively. Thus the model constructs edges (where non-Markovian features can be defined) by directly connecting the exactly first **L** and the last **R** nodes of a span.

Relaxed Variant: One assumption made by QT is: each word in $\boldsymbol{t}$ strictly belongs to a certain quantity span. The variant QT(R) relaxes such a constraint. In this variant, some tokens in $\boldsymbol{t}$ may not belong to any quantity spans. Considering the example shown in Figure 2, the token "*There*" in $\boldsymbol{t}$ may not belong to any spans.

3 Experiments

We conduct experiments on two datasets, AddSub (Hosseini et al., 2014), consisting of 395 addition-subtraction problems in English, and AS_CN with 1,049 addition-subtraction problems in Chinese (Wang et al., 2017). For all of our experiments, we use the L-BFGS algorithm (Liu and Nocedal, 1989) for learning model parameters with $\ell 2$ regularization coefficient of 0.01. To tune the hyperparameter J, we randomly select 80% instances of the training set for training and the rest 20% for development. We tune J on the development set.

3.1 Analysis

Following standard evaluation procedures used in previous works (Hosseini et al., 2014; Mitra and Baral, 2016), we conduct 3-fold cross validation on AddSub and AS_CN, and report accuracies in Table 1. We make comparisons with a list of recent works[1] and two baselines. Another is QT(FIX)

[1]Results on AS_CN are obtained by running publicly released systems.

Model	AddSub					AS_CN				
	$A_{S.S.}$	$A_{M.S.}$	F_+	$F_{\mathbb{0}}$	F_-	$A_{S.S.}$	$A_{M.S.}$	F_+	$F_{\mathbb{0}}$	F_-
QT	**89.5**	**97.3**	**96.0**	**86.4**	**96.5**	56.9	60.3	85.5	62.2	85.0
QT(S)	86.5	91.2	95.0	82.8	95.6	53.6	56.3	85.3	**62.9**	84.3
QT(R)	87.5	92.6	95.4	82.5	96.0	**57.03**	**60.9**	**86.5**	62.9	**85.6**

Table 2: Accuracies on two types of problems and $F1$ scores for three types of signs of quantities. $A_{S.S.}$: accuracy of single-step problems (%) ; $A_{M.S.}$ accuracy of multi-step problems (%) ; $F_{+(-/\mathbb{0})}$: $F1$ score of sign "$+\mathbb{1}(-\mathbb{1}/\mathbb{0})$" (%).

where the quantity span for each quantity is a fixed-size token window. All of our proposed models consistently outperform previous research efforts. These figures confirm the capability of our approach to provide more promising solutions to addition-subtraction problems. We do not require any additional annotations which can be expensive, while annotations like variable-word alignments and formulas are necessary for works of (Kushman et al., 2014; Mitra and Baral, 2016).

To investigate the power of features extracted by external tools, such as ConceptNet (Liu and Singh, 2004) and Stanford CoreNLP tool (Manning et al., 2014), we conduct additional experiments on the afore-mentioned datasets, where we call such features *external features* (see supplementary material), indicated as "-EF". It is expected that the performance drops because such features are necessary for capturing evidence across sentences. Especially, for the AddSub dataset, it affects a lot. As discussed before (Hosseini et al., 2014; Mitra and Baral, 2016), there exists lots of irrelevant information and information gaps in AddSub. We thus can infer the external features support our approach to be capable of bridging information gaps and recognizing irrelevant information for solving arithmetic problems. Poor performance shows challenges to solve such problems in Chinese.

Which of our variants works the best? We observe that models with variable-sized quantity spans, namely QT, QT(S) and QT(R), generally perform better than QT(FIX) where the quantity spans are fixed token windows. This shows the effectiveness of introducing the quantity span as a latent variable. QT obtains the highest average accuracy on the AddSub and QT(R) outperforms other two variants on the AS_CN.

How does our approach perform on different types of problems? We divide problems into two categories: single-step and multi-step problems. The equation of a single-step problem contains at most two constant quantities tagged with either "$+\mathbb{1}$" or "$-\mathbb{1}$", while the equation for a multi-

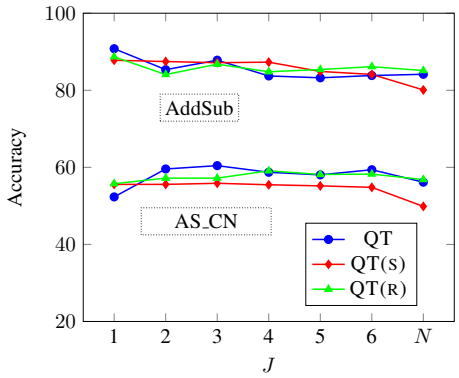

Figure 3: Effects of J on three models (QT, QT(S) and QT(R)) evaluated on AddSub and AS_CN.

step problem has more than two constant quantities with signs of "$+\mathbb{1}$" or "$-\mathbb{1}$". We report accuracy and $F1$ score in Table 2. According to empirical results illustrated in Table 2, our approach is able to give more accurate answers to multi-step problems, while the accuracy of single-step problems is lower. On the other hand, three models have similar patterns in terms of performance for three types of signs. The $F1$ scores for signs of "$+\mathbb{1}$" and "$-\mathbb{1}$" are higher than scores of "$\mathbb{0}$". After examining outputs, we found that problem texts of single-step problems often contain more than two constant quantities, among which only two of them are supposed to be labeled as "$+\mathbb{1}$" or "$-\mathbb{1}$" and the rest should be tagged as "$\mathbb{0}$". However, incorrectly labeling an irrelevant quantity with "$+\mathbb{1}$" or "$-\mathbb{1}$" leads to wrong solutions to single-step problems. This also reveals that one main challenge for automatically solving arithmetic word problems is to recognize the irrelevant quantities. Failures in identifying irrelevant information may due to implicit information of problem text or the external tool issues.

Does J really matter? We further investigate the effects of J on the three proposed models. Figure 3 plots how performance varies with J ($J \in \{1, 2, 3, 4, 5, 6, N\}$[2]) on datasets AddSub (above) and AS_CN (below). On AddSub, three models have similar patterns that performance tends to be worse with a larger J. As for the AS_CN dataset, three models achieve relatively higher accuracies with $J \in \{2, 3, 6\}$ compared to other scenarios. Interestingly, it seems that QT and QT(R) performs better than the semi-Markov variant QT(S). We tracked outputs from three models and found that QT(S) made more mistakes in predictions for

[2]All tokens in the problem text are considered as the selected token window for a quantity when $J = N$.

Model		AddSub			AS_CN		
		P.	*R.*	*F.*	*P.*	*R.*	*F.*
QT	+	**95.21**$*$	**96.70**$*$	**95.95**$*$	81.86	89.54	85.53
	0	**88.88**†	83.96	**86.35**†	75.83	52.74	62.21
	–	96.65	**96.38**$\star$	**96.51**$\star$	88.17	82.04	84.99
QT(S)	+	93.97	96.01	94.98	80.74	**90.30**$*$	85.26
	0	81.06	**84.50**†	82.75	75.00	**54.22**†	**62.94**†
	–	96.97	94.18	95.55	**88.66**$\star$	80.25	84.25
QT(R)	+	94.37	96.42	95.38	**83.79**$*$	89.39	**86.50**$*$
	0	80.55	**84.50**†	82.48	**78.02**†	52.72	62.92
	–	**97.48**$\star$	94.65	96.04	86.67	**84.61**$\star$	**85.63**$\star$

Table 3: Results for three types of signs for quantities predicted by three models. *P.*: Precision (%), *R.*: Recall (%), *F.*: $F1$ score (%); Highest scores are in **bold** and we use $*$, † and $\star$ to distinguish different sign types.

unknown. The fact that models with $J = N$ perform do not perform well confirms our assumption that taking token windows into account rather than the whole text is reasonable and effective.

Evaluation on different types of signs: We investigate the capability of proposed approach to predict three types of signs ($\{+\mathbb{1}, \mathbb{0}, -\mathbb{1}\}$), as illustrated in Table 3. Three models have similar patterns on two datasets. Predictions of "$+\mathbb{1}$" and "$-\mathbb{1}$" are more promising, compared to "$\mathbb{0}$". This reveals that one main challenge for automatically solving arithmetic word problems is to recognize the irrelevant information that should be labeled with "$\mathbb{0}$". Like what we discussed, failure on detecting irrelevant knowledge could be resulted from inevitably errors introduced by external resources and the lack of presence of crucial information in problem text.

Error Analysis The leading sources of errors can be categorized into three types: 1) The description of the problem is incomplete and implicit, which is challenging for machine to understand. 2) Failing in recognizing relevant quantities caused missing quantities or introducing irrelevant information. 3) Incomplete information or errors from external tools, such as ConceptNet (Liu and Singh, 2004) and Standford CoreNLP tool (Manning et al., 2014), are another source of errors leading to wrong predictions, which are inevitable.

4 Conclusion and Future Work

This work proposes the *Quantity Tagger* that regards solving addition-subtraction problem as a sequence labeling task by introducing the quantity span for each quantity. Despite its simplicity, it yields better performance. In the future, we would also like to investigate better models that are capable to address general arithmetic word problems, including addition, subtraction, multiplication and division.

Acknowledgments

We would like to thank the three anonymous reviewers for their thoughtful and constructive comments. We would also like to thank Yan Wang for his help on this work. This work is supported by Singapore Ministry of Education Academic Research Fund (AcRF) Tier 2 Project MOE2017-T2-1-156, and is partially supported by SUTD project PIE-SGP-AI-2018-01.

References

Daniel G Bobrow. 1964. A question-answering system for high school algebra word problems. In *Proc. of AFIPS*.

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

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *TACL*, 3:585–597.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proc. of ACL*.

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

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

Hao Li and Wei Lu. 2017. Learning latent sentiment scopes for entity-level sentiment analysis. In *Proc. of AAAI*.

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

Hugo Liu and Push Singh. 2004. ConceptNet-a practical commonsense reasoning tool-kit. *BT technology journal*, 22.

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

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

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

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

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

Sunita Sarawagi and William W Cohen. 2005. Semi-Markov conditional random fields for information extraction. In *Proc. of NIPS*.

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

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

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

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *Proc. of EMNLP*.

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

Yanyan Zou and Wei Lu. 2019. Joint detection and location of english puns. In *Proc. of NAACL*.