Learn to Solve Algebra Word Problems Using Quadratic Programming

Lipu Zhou, Shuaixiang Dai, and Liwei Chen
Baidu Inc., Beijing, China
zhoulipu@outlook.com, {daishuaixiang, chenliwei}@baidu.com

Abstract

This paper presents a new algorithm to automatically solve algebra word problems. Our algorithm solves a word problem via analyzing a hypothesis space containing all possible equation systems generated by assigning the numbers in the word problem into a set of equation system templates extracted from the training data. To obtain a robust decision surface, we train a log-linear model to make the margin between the correct assignments and the false ones as large as possible. This results in a quadratic programming (QP) problem which can be efficiently solved. Experimental results show that our algorithm achieves 79.7% accuracy, about 10% higher than the state-of-the-art baseline (Kushman et al., 2014).

1 Introduction

An algebra word problem describes a mathematical problem which can be typically modeled by an equation system, as demonstrated in Figure 1. Seeking to automatically solve word problems is a classical AI problem (Bobrow, 1964). The word problem solver is traditionally created by the rule-based approach (Lev et al., 2004; Mukherjee and Garain, 2008; Matsuzaki et al., 2013). Recently, using machine learning techniques to construct the solver has become a new trend (Kushman et al., 2014; Hosseini et al., 2014; Amnueypornsakul and Bhat, 2014; Roy et al., 2015). This is based on the fact that word problems derived from the same mathematical problem share some common semantic and syntactic features due to the same underlying logic. Our method follows this trend.[1]

To solve a word problem, our algorithm analyzes all the possible ways to assign the numbers

	Our method	Kushman's method
Word problem	An **amusement park** sells 2 **kinds** of **tickets**. **Tickets** for **children** cost \$ 1.50. **Adult tickets** cost \$ 4. On a certain **day**, 278 **people** entered the **park**. On that same **day** the **admission fees** collected totaled \$ 792. How many **children** were admitted on that **day**? How many **adults** were admitted?	
Template	$u_1 + u_2 - n_1 = 0$ $n_2 \times u_1 + n_3 \times u_2 - n_4 = 0$	$u_1^1 + u_2^1 - n_1 = 0$ $n_2 \times u_1^2 + n_3 \times u_2^2 - n_4 = 0$
Assignment	$2, 1.5, 4, 278, 792$ $\Downarrow$ n_1, n_2, n_3, n_4	$2, 1.5, 4, 278, 792,$ *nouns* $\Downarrow$ $n_1, n_2, n_3, n_4, u_1^1, u_2^1, u_1^2, u_2^2$
Possible Assignment	$5 \times 4 \times 3 \times 2 = 120$	$17^4 \times 5 \times 4 \times 3 \times 2 = 10022520$

Figure 1: Comparison between our algorithm and (Kushman et al., 2014). Nouns are boldfaced.

in the word problem to a set of equation system templates. Kushman et al. (2014) also consider filling the equation system templates to generate the candidate equations. But Kushman's template contains number slots (e.g. n_1, n_2, n_3, n_4 in Figure 1) and unknown slots (e.g. u_1^1, u_1^2, u_2^1, u_2^2 in Figure 1). They separately consider assigning nouns into the unknown slots and numbers into the number slots, as demonstrated in Figure 1. As filling the unknown slots is closely related to the number slots assignment, we only consider assigning the number slots, and design effective features to describe the relationship between numbers and unknowns. This scheme significantly reduces the hypothesis space, as illustrated in Figure 1, which benefits the learning and inference processes.

We use a log-linear model to describe the template selection and number assignment. To learn the model parameters of such problem, maximizing the log-likelihood objective is generally adopted (Kwiatkowski et al., 2010; Kushman et al., 2014). The key difficulty of this method is that calculating the gradient of the objective function needs to sum over exponentially many samples. Thus, it is essential to approximate the gradient. For instance, Kushman et al. (2014) use

[1]Our code is available at http://pan.baidu.com/s/1dD336Sx

beam search to approximately calculate the gradient. This method can not exploit all the training samples. Thus the resulting model may be suboptimal. Motivated by the work (Taskar et al., 2005; Li, 2014), we adopt the max-margin objective. This results in a QP problem and opens the way toward an efficient learning algorithm (Koller and Friedman, 2009).

We evaluate our algorithm on the benchmark dataset provided by (Kushman et al., 2014). The experimental results show that our algorithm significantly outperforms the state-of-the-art baseline (Kushman et al., 2014).

2 Problem Formulation

Our word problem solver is constructed by training a log-linear model to find the correct mapping from a word problem to an equation.

Notations: Let $\mathcal{X}$ denote the set of training word problems, and $\mathcal{T}$ denote the set of equation system templates abstracted from $\mathcal{X}$ as (Kushman et al., 2014). x_i is the *i*-th word problem in $\mathcal{X}$. Assume T_j is the *j*-th equation system template in $\mathcal{T}$, and $\mathcal{N}_{T_j} = \left\{n^1_{T_j}, n^2_{T_j}, \cdots, n^m_{T_j}\right\}$ is the set of number slots of T_j, where m represents the size of $\mathcal{N}_{T_j}$. Denote the numbers in x_i by $\mathcal{N}_{x_i} = \left\{n^1_{x_i}, n^2_{x_i}, \cdots, n^l_{x_i}\right\}$, where l represents the size of $\mathcal{N}_{x_i}$. Assuming $l \geq m$, we further define π_{ijk} a sequence of m numbers chosen from $\mathcal{N}_{x_i}$ without repetition. Given π_{ijk}, we can map T_j to an equation system e_{ijk} by filling the number slots $\mathcal{N}_{T_j}$ of T_j sequently with the numbers in π_{ijk}. Solving e_{ijk}, we can obtain the corresponding solution s_{ijk}. To simplify the notation, we define $y_{ijk} = (T_j, \pi_{ijk}, e_{ijk}, s_{ijk})$ the k-th derivation give x_i and T_j, and let $\mathcal{Y}_i$ denote the set of all possible y_{ijk} given x_i and $\mathcal{T}$. Therefore, to correctly solve x_i is to find the correct $y_{ijk} \in \mathcal{Y}_i$.

Probabilistic Model: As (Kushman et al., 2014), we use the log-linear model to define the probability of $y_{ijk} \in \mathcal{Y}_i$ given x_i:

$$p(y_{ijk}|x_i;\theta) = \frac{e^{\theta\cdot\phi\left(x_i, y_{ijk}\right)}}{\sum\limits_{y'_{ijk}\in\mathcal{Y}_i} e^{\theta\cdot\phi\left(x_i, y'_{ijk}\right)}} \tag{1}$$

where θ is the parameter vector of the model, and $\phi\left(x_i, y_{ijk}\right)$ denotes the feature function. We adopt the max-margin objective (Vapnik, 2013) to directly learn the decision boundary for the correct derivations and the false ones.

3 Learning and Inference

3.1 Learning

Using (1), we obtain the difference between the log-probability of a correct derivation $y^c_{ijk} \in \mathcal{Y}_i$ and a false one $y^f_{ijl} \in \mathcal{Y}_i$ as:

$$\begin{aligned}&\ln P\left(y^c_{ijk}|x_i;\theta\right) - \ln P\left(y^f_{ijl}|x_i;\theta\right)\\ =&\theta\cdot\left(\phi\left(x_i, y^c_{ijk}\right) - \phi\left(x_i, y^f_{ijl}\right)\right)\end{aligned} \tag{2}$$

Note that the subtraction in (2) cancels the denominator of (1) which contains extensive computation. To decrease the generalization error of the learned model, we would like the minimal gap between the correct derivations and the false ones as large as possible. In practice, we may not find a decision hyperplane to perfectly separate the correct and the false derivations. Generally, this can be solved by introducing a slack variable $\xi_{ijkl} \geq 0$ (Bishop, 2006) for each constraint derived from (2). Define $\varphi\left(x_i, y^c_{ijk}, y^f_{ijl}\right) = \phi\left(x_i, y^c_{ijk}\right) - \phi\left(x_i, y^f_{ijl}\right)$. For $\forall\, x_i \in \mathcal{X}$, the resulting optimization problem is:

$$\begin{aligned}&\arg\min \frac{1}{2}\|\theta\|_2 + C\sum_{i,j,k,l}\xi_{ijkl} \\ &s.t.\ \theta\cdot\varphi\left(x_i, y^c_{ijk}, y^f_{ijl}\right) \geq 1-\xi_{ijkl},\ \xi_{ijkl}\geq 0\end{aligned} \tag{3}$$

The parameter C is used to balance the slack variable penalty and the margin. This is a QP problem and has been well studied (Platt, 1999; Fan et al., 2008).

According to the Karush-Kuhn-Tucker (KKT) condition, only a part of the constraints is active for the solution of (3) (Bishop, 2006). This leads to an efficient learning algorithm called constraint generation (Koller and Friedman, 2009; Felzenszwalb et al., 2010). Specifically, an initial model is trained by a randomly selected subset of the constraints. Next this model is used to check the constraints and at most N false deviations that are erroneously classified by this model are collected for each word problem. These constraints are then added to train a new model. This process repeats until converges. Our experimental results show that this process converges fast.

3.2 Inference

When we obtain the model parameter θ, the inference can be performed by finding the maximum

Single slot features
Relation between numbers and the question sentence.
Position of a number w.r.t a comparative word.
Context of a number.
Is one or two?
Is a multiplier?
Is between 0 and 1?
Slot pair features
Relation between two numbers.
Context similarity between two numbers.
Does there exist coreference relationship?
Are two numbers both multipliers?
Are two numbers in the same sentence or continuous sentences?
Information of raw path and dependency path between two numbers
One number is larger than another.
Solution features
Is integer solution?
Is positive solution?
Is between 0 and 1?

Table 1: Features used in our algorithm.

value of (1). This can be simplified by computing

$$\arg\max_{y_{ijk}\in\mathcal{Y}_i} \theta \cdot \phi(x_i, y_{ijk}) \qquad (4)$$

As we only consider assigning the number slots of the templates in $\mathcal{T}$, generally, the size of the possible assignments per word problem is bearable, as shown in the Table 2. Thus we simply evaluate all the $y_{ijk} \in \mathcal{Y}_i$. The one with the largest score is considered as the solution of x_i.

4 Features

A feature vector $\phi(x_i, y_{ijk})$ is calculated for each word problem x_i and derivation y_{ijk} pair. As Kushman (2014), a feature is associated with a signature related to the template of y_{ijk}. We extract three kinds of features, i.e., single slot features, slot pair features and solution features. Unless otherwise stated, single slot and slot pair features are associated with the slot and slot pair signature of the equation system template, respectively, and solution features are generated for the signature of the equation system template. Table 1 lists the features used in our algorithm. The detailed description is as follows.

4.1 Single Slot Features

To reduce the search space, we only consider the assignment of the number slots of the template. It seems that our algorithm will lose the information about the unknown. But such information can be recovered by the features that include the information of the question sentence. Specifically, we associate a number with all the nouns in the same sentence sorted by the length of the dependence path between them. For instance, [$, tickets, children] is the sorted noun list for 1.5 in Figure 1. Assume the n-th noun of the nouns associated to a given number is the first noun that appears in the question sentence. We quantify the relationship between a number and a queried entity by the reciprocal of n. For instance, in Figure 1, "children" appears in the question sentence, and it is the third noun associated to 1.5. So the value of this feature is $1/3$. A larger value of this feature means a number more likely relates to the queried entity. The maximum value of this feature is 1. Thus we introduce a feature to indicate whether this special case occurs. We also use a feature to indicate whether a number appears in the question sentence.

The comparative meaning is sensitive to both the comparative words and the position of a number relative to them. For example, "one number is 3 less than twice another" is different to "one number is 3 more than twice another", but equal to "twice a number is 3 more than another". To account for this, we use the comparative words coupled with the position of a number relative to them as features.

On the other hand, we use the lemma, part of speech (POS) tag and the dependence type related to the word within a widow [-5, +5] around a number as features. Besides, if the POS tag or the named entity tag of a number is not labeled as a general number, we also import these tags together with the first noun and the dependence type related to the number as features.

Additionally, the numbers 1 and 2 are usually used to indicate the number of variables, such as "the sum of two numbers". To capture such usage, we use a feature to denote whether a number is one or two as (Kushman et al., 2014). Since such usage appears in various kinds of word problems, this feature does not contain the slot signature. We also generate features to indicate whether a number belongs to (0, 1), and whether it is a multiplier, such as twice, triple.

4.2 Slot Pair Features

Assume n_1 and n_2 are two numbers in a word problem. Suppose NP_1 and NP_2 are the lists of nouns associated to n_1 and n_2 (described in section 4.1), respectively. We evaluate the relationship $r(n_1, n_2)$ between n_1 and n_2 by:

$$\max_{\substack{noun_1^i \in NP_1,\\ noun_2^j \in NP_2\\ s.t.\ noun_1^i = noun_2^j}} \left(\frac{2}{ord\left(noun_1^i\right) + ord\left(noun_2^j\right)} \right)$$

where $ord(\cdot)$ denotes the index of a noun in $NP_i\,(i = 1, 2)$, starting from 1. A larger $r(n_1, n_2)$ means n_1 and n_2 are more related. The maximum value of $r(n_1, n_2)$ is 1, which occurs when the first nouns of NP_1 and NP_2 are equal. We use a feature to indicate whether $r(n_1, n_2)$ is 1. This feature helps to import some basic rules of the arithmetic operation, e.g., the units of summands should be the same.

If two slots are symmetric in a template (e.g., n_2 and n_3 in Figure 1), the contexts around both numbers are generally similar. Assume CT_1 and CT_2 are two sets of certain tags within a window around n_1 and n_2, respectively. Then we calculate the contextual similarity between n_1 and n_2 by:

$$sim(ST_1, ST_2) = \frac{|ST_1 \cap ST_2|}{|ST_1 \cup ST_2|}$$

In this paper, the tags include the lemma, POS tag and dependence type, and the window size is 5.

Besides, we exploit features to denote whether there exists coreference relationship between any elements of the sentences where n_1 and n_2 locate, and whether two numbers are both multipliers. Finally, according to (Kushman et al., 2014), we generate features related to the raw path and dependence path between two numbers, and use the numeric relation between them as a feature to import some basic arithmetic rules, such as the positive summands are smaller than their sum. We also include features to indicate whether two numbers are in the same sentence or continuous sentences.

4.3 Solution Features

Many word problems are math problems about the real life. This background leads the solutions of many word problems have some special numerical properties, such as the positive and integer properties used by (Kushman et al., 2014). To capture such fact, we introduce a set of features to describe the solution properties.

5 Experiments

Dataset: The dataset used in our experiment is provided by (Kushman et al., 2014). Equivalent equation systme templates are automatically merged. The word problems are parsed by (Manning et al., 2014). The version of the parser is the same as (Kushman et al., 2014). The performance of our algorithm is evaluated by comparing each number of the correct answer with the calculated one, regardless of the ordering. We report the average accuracy of 5-fold cross-validation.

Learning: We use liblinear (Fan et al., 2008) to solve the QP problem. The parameter C in (3) is set to 0.01 in all the following experiments. We randomly select 300 false derivations of each word problem to form the initial training set. We add at most 300 false derivations for each word problem during the constraint generation step, and use 5-fold cross-validation to avoid overfitting. We stop iterating when the cross-validation error becomes worse or the training error converges or none new constraints are generated.

Supervision Level: We consider the learning with two different levels of supervision. In the first case, the learning is conducted by providing the equation and the correct answer of every training sample. In the second case, the correct answer is available for every training sample but without the equation. Instead, all the templates are given, but the correspondence between the template and the training sample is not available. During learning, the algorithm should evaluate every derivation of each template to find the true one.

Results: Table 2 lists the learning statistics for our algorithm and (Kushman et al., 2014). We can observe that the number of possible alignments per word problem of our algorithm is much smaller than (Kushman et al., 2014). However, the number of all the false alignments is still 80K. Using the constraint generation algorithm (Koller and Friedman, 2009), only 9K false alignments are used in the quadratic programming. We trained our model on a Intel i5-3210M CUP and 4G RAM laptop. Kushman's algorithm (2014) needs much more memory than our algorithm and can not run on a general laptop. Therefore, we tested their algorithm on a workstation with Intel E5-2620 CPU and 128G memory. As shown in Table 2, their algorithm takes more time than our algorithm.

Table 3 lists the accuracy of our algorithm and Kushman's algorithm (2014). It is clear that our

Mean negative samples	80K
Mean negative samples used in learning	9K
Mean time for feature extraction	22m
Mean training time	7.3m
Mean feature extraction and training time of (Kushman et al., 2014)	83m
# Alignments per problem of (Kushman et al., 2014)	4M
# Alignments per problem of our algorithm	1.9K

Table 2: Learning statistics.

Algorithm	Accuracy
Our algorithm fully supervised	**79.7%**
Our algorithm weakly supervised	72.3%
Kushman's algorithm (2014) fully supervised	68.7%

Table 3: Algorithm comparison.

Feature Ablation	Accuracy
Without single slot features	70.4%
Without slot pair features	69.3%
Without solution features	71.8%

Table 4: Ablation study for fully supervised data.

algorithm obtains better result. The result of the weakly supervised data is worse than the fully supervised one. But this result is still higher than Kushman's fully supervised result.

Table 4 gives the results of our algorithm with different feature ablations. We can find that all the features are helpful to get the correct solution and none of them dramatically surpasses the others.

Discuss: Although our algorithm gives a better result than (Kushman et al., 2014), there still exist two main problems that need to be further investigated, as demonstrated in Table 5. The first problem is caused by our feature for semantic representation. Our current lexicalized feature can not generalize well for the unseen words. For example, it is hard for our algorithm to relate the word "forfeits" to "minus", if it does not appear in the training corpus. The second problem is caused by the fact that our algorithm only considers the single noun as the entity of a word problem. Thus when the entity is a complicated noun phrase, our algorithm may fail.

Problem	Example
Lexicalized features can not generalize well for unseen words.	A woman is paid 20 dollars for each day she works and **forfeits** a 5 dollars for each day she is idle. At the end of 25 days she nets 450 dollars. How many days did she work?
Can not deal with complicated noun phrases.	**The probability that San Francisco plays in the next super bowl** is nine times **the probability that they do not play in the next super bowl**. **The probability that San Francisco plays in the next super bowl** plus **the probability that they do not play** is 1. What is **the probability that San Francisco plays in the next super bowl?**

Table 5: The problems of our algorithm.

6 Conclusion and Future work

In this paper, we present a new algorithm to learn to solve algebra word problems. To reduce the possible derivations, we only consider filling the number slots of the equation system templates, and design effective features to describe the relationship between numbers and unknowns. Additionally, we use the max-margin objective to train the log-linear model. This results in a QP problem that can be efficiently solved via the constraint generation algorithm. Experimental results show that our algorithm significantly outperforms the state-of-the-art baseline (Kushman et al., 2014).

Our future work will focus on studying the performance of applying nonlinear kernel function to the QP problem (3), and using the word embedding vector (Bengio et al., 2003; Mikolov et al., 2013) to replace current lexicalized features. Besides, we would like to compare our algorithm with the algorithms designed for specific word problems, such as (Hosseini et al., 2014).

7 Acknowledgments

This work is supported by the National Basic Research Program of China (973 program No. 2014CB340505). We would like to thank Hua Wu and the anonymous reviewers for their helpful comments that improved the work considerably.

References

Bussaba Amnueypornsakul and Suma Bhat. 2014. Machine-guided solution to mathematical word problems.

Yoshua Bengio, Rjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(6):1137–1155.

Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.

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

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874.

Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. 2010. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645.

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.

Daphne Koller and Nir Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. *ACL (1)*, pages 271–281.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1223–1233. Association for Computational Linguistics.

Iddo Lev, Bill MacCartney, Christopher D Manning, and Roger Levy. 2004. Solving logic puzzles: From robust processing to precise semantics. In *Proceedings of the 2nd Workshop on Text Meaning and Interpretation*, pages 9–16. Association for Computational Linguistics.

Hang Li. 2014. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 7(3):1–121.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Takuya Matsuzaki, Hidenao Iwane, Hirokazu Anai, and Noriko Arai. 2013. The complexity of math problems–linguistic, or computational. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 73–81.

Tomas Mikolov, Wen Tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space-word representations. *In HLT-NAACL.*

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.

John C. Platt. 1999. Fast training of support vector machines using sequential minimal optimization. *In B. Scho04lkopf, C. Burges and A. Smola (Eds.), Advances in kernel methods - Support vector learning.*

Subhro Roy, Tim Vieira, and Dan Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics*, 3:1–13.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd international conference on Machine learning*, pages 896–903. ACM.

Vladimir Vapnik. 2013. *The nature of statistical learning theory*. Springer Science & Business Media.