Learning to Automatically Solve Algebra Word Problems

Nate Kushman[†], **Yoav Artzi**[‡], **Luke Zettlemoyer**[‡], **and Regina Barzilay**[†]

[†] Computer Science and Articial Intelligence Laboratory, Massachusetts Institute of Technology
{nkushman, regina}@csail.mit.edu
[‡] Computer Science & Engineering, University of Washington
{yoav, lsz}@cs.washington.edu

Abstract

We present an approach for automatically learning to solve algebra word problems. Our algorithm reasons across sentence boundaries to construct and solve a system of linear equations, while simultaneously recovering an alignment of the variables and numbers in these equations to the problem text. The learning algorithm uses varied supervision, including either full equations or just the final answers. We evaluate performance on a newly gathered corpus of algebra word problems, demonstrating that the system can correctly answer almost 70% of the questions in the dataset. This is, to our knowledge, the first learning result for this task.

1 Introduction

Algebra word problems concisely describe a world state and pose questions about it. The described state can be modeled with a system of equations whose solution specifies the questions' answers. For example, Figure 1 shows one such problem. The reader is asked to infer how many children and adults were admitted to an amusement park, based on constraints provided by ticket prices and overall sales. This paper studies the task of learning to automatically solve such problems given only the natural language.[1]

Solving these problems requires reasoning across sentence boundaries to find a system of equations that concisely models the described semantic relationships. For example, in Figure 1, the total ticket revenue computation in the second equation summarizes facts about ticket prices and total sales described in the second, third, and fifth sentences. Furthermore, the first equation models an implicit semantic relationship, namely that the children and adults admitted are non-intersecting subsets of the set of people who entered the park.

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?
Equations
$x + y = 278$ $1.5x + 4y = 792$
Solution
$x = 128$ $\quad y = 150$

Figure 1: An example algebra word problem. Our goal is to map a given problem to a set of equations representing its algebraic meaning, which are then solved to get the problem's answer.

Our model defines a joint log-linear distribution over full systems of equations and alignments between these equations and the text. The space of possible equations is defined by a set of equation templates, which we induce from the training examples, where each template has a set of slots. Number slots are filled by numbers from the text, and unknown slots are aligned to nouns. For example, the system in Figure 1 is generated by filling one such template with four specific numbers (1.5, 4, 278, and 792) and aligning two nouns ("Tickets" in "Tickets for children", and "tickets" in "Adult tickets"). These inferred correspondences are used to define cross-sentence features that provide global cues to the model. For instance, in our running example, the string

[1] The code and data for this work are available at http://groups.csail.mit.edu/rbg/code/wordprobs/.

pairs ("$1.50", "children") and ("$4","adults") both surround the word "cost," suggesting an output equation with a sum of two constant-variable products.

We consider learning with two different levels of supervision. In the first scenario, we assume access to each problem's numeric solution (see Figure 1) for most of the data, along with a small set of seed examples labeled with full equations. During learning, a solver evaluates competing hypotheses to drive the learning process. In the second scenario, we are provided with a full system of equations for each problem. In both cases, the available labeled equations (either the seed set, or the full set) are abstracted to provide the model's equation templates, while the slot filling and alignment decisions are latent variables whose settings are estimated by directly optimizing the marginal data log-likelihood.

The approach is evaluated on a new corpus of 514 algebra word problems and associated equation systems gathered from Algebra.com. Provided with full equations during training, our algorithm successfully solves over 69% of the word problems from our test set. Furthermore, we find the algorithm can robustly handle weak supervision, achieving more than 70% of the above performance when trained exclusively on answers.

2 Related Work

Our work is related to three main areas of research: situated semantic interpretation, information extraction, and automatic word problem solvers.

Situated Semantic Interpretation There is a large body of research on learning to map natural language to formal meaning representations, given varied forms of supervision. Reinforcement learning can be used to learn to read instructions and perform actions in an external world (Branavan et al., 2009; Branavan et al., 2010; Vogel and Jurafsky, 2010). Other approaches have relied on access to more costly annotated logical forms (Zelle and Mooney, 1996; Thompson and Mooney, 2003; Wong and Mooney, 2006; Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2010). These techniques have been generalized more recently to learn from sentences paired with indirect feedback from a controlled application. Examples include question answering (Clarke et al., 2010; Cai and Yates, 2013a; Cai and Yates, 2013b; Berant et al., 2013; Kwiatkowski et al., 2013), dialog systems (Artzi and Zettlemoyer, 2011), robot instruction (Chen and Mooney, 2011; Chen, 2012; Kim and Mooney, 2012; Matuszek et al., 2012; Artzi and Zettlemoyer, 2013), and program executions (Kushman and Barzilay, 2013; Lei et al., 2013). We focus on learning from varied supervision, including question answers and equation systems, both can be obtained reliably from annotators with no linguistic training and only basic math knowledge.

Nearly all of the above work processed single sentences in isolation. Techniques that consider multiple sentences typically do so in a serial fashion, processing each in turn with limited cross-sentence reasoning (Branavan et al., 2009; Zettlemoyer and Collins, 2009; Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013). We focus on analyzing multiple sentences simultaneously, as is necessary to generate the global semantic representations common in domains such as algebra word problems.

Information Extraction Our approach is related to work on template-based information extraction, where the goal is to identify instances of event templates in text and extract their slot fillers. Most work has focused on the supervised case, where the templates are manually defined and data is labeled with alignment information, e.g. (Grishman et al., 2005; Maslennikov and Chua, 2007; Ji and Grishman, 2008; Reichart and Barzilay, 2012). However, some recent work has studied the automatic induction of the set of possible templates from data (Chambers and Jurafsky, 2011; Ritter et al., 2012). In our approach, systems of equations are relatively easy to specify, providing a type of template structure, and the alignment of the slots in these templates to the text is modeled primarily with latent variables during learning. Additionally, mapping to a semantic representation that can be executed allows us to leverage weaker supervision during learning.

Automatic Word Problem Solvers Finally, there has been research on automatically solving various types of mathematical word problems. The dominant existing approach is to hand engineer rule-based systems to solve math problem in specific domains (Mukherjee and Garain, 2008; Lev et al., 2004). Our focus is on learning a model for the end-to-end task of solving word problems given only a training corpus of questions paired with equations or answers.

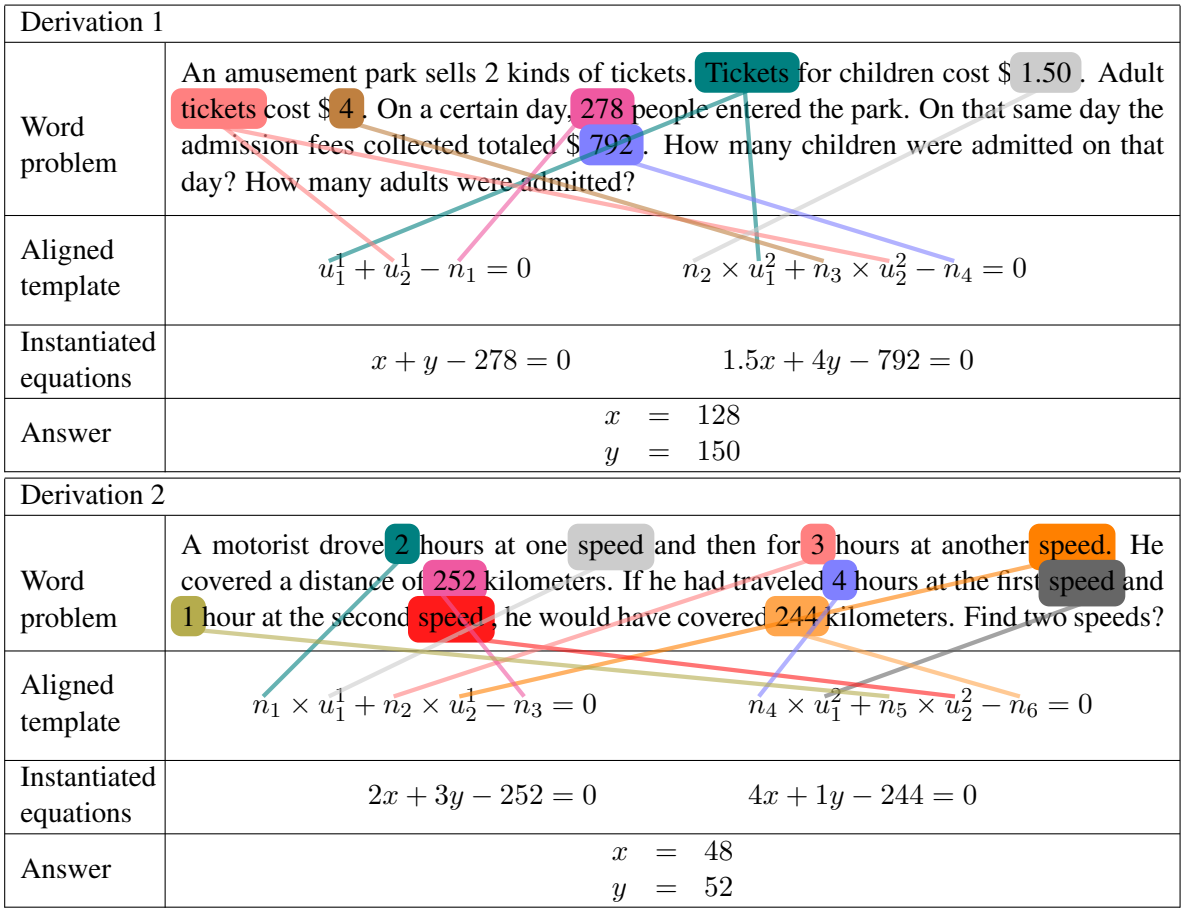

Figure 2: Two complete derivations for two different word problems. Derivation 1 shows an alignment where two instances of the same slot are aligned to the same word (e.g., u_1^1 and u_1^2 both are aligned to "Tickets"). Derivation 2 includes an alignment where four identical nouns are each aligned to different slot instances in the template (e.g., the first "speed" in the problem is aligned to u_1^1).

3 Mapping Word Problems to Equations

We define a two step process to map word problems to equations. First, a template is selected to define the overall structure of the equation system. Next, the template is instantiated with numbers and nouns from the text. During inference we consider these two steps jointly.

Figure 2 shows both steps for two derivations. The template dictates the form of the equations in the system and the type of slots in each: u slots represent unknowns and n slots are for numbers that must be filled from the text. In Derivation 1, the selected template has two unknown slots, u_1 and u_2, and four number slots, n_1 to n_4. Slots can be shared between equations, for example, the unknown slots u_1 and u_2 in the example appear in both equations. A slot may have different instances, for example u_1^1 and u_1^2 are the two instances of u_1 in the example.

We align each slot instance to a word in the problem. Each number slot n is aligned to a number, and each unknown slot u is aligned to a noun. For example, Derivation 1 aligns the number 278 to n_1, 1.50 to n_2, 4 to n_3, and 792 to n_4. It also aligns both instances of u_1 (e.g., u_1^1 and u_1^2) to "Tickets", and both instances of u_2 to "tickets". In contrast, in Derivation 2, instances of the same unknown slot (e.g. u_1^1 and u_1^2) are aligned to two different words in the problem (different occurrences of the word "speed"). This allows for a tighter mapping between the natural language and the system template, where the words aligned to the first equation in the template come from the first two sentences, and the words aligned to the second equation come from the third.

Given an alignment, the template can then be instantiated: each number slot n is replaced with the aligned number, and each unknown slot u with a variable. This output system of equations is then automatically solved to generate the final answer.

3.1 Derivations

Definitions Let $\mathcal{X}$ be the set of all word problems. A word problem $x \in \mathcal{X}$ is a sequence of k words $\langle w_1, \ldots w_k \rangle$. Also, define an *equation template* t to be a formula $A = B$, where A and B are expressions. An expression A is one of the following:

- A number constant f.
- A number slot n.
- An unknown slot u.
- An application of a mathematical relation R to two expressions (e.g., $n_1 \times u_1$).

We define a *system template* T to be a set of l equation templates $\{t_0, \ldots, t_l\}$. $\mathcal{T}$ is the set of all system templates. A slot may occur more than once in a system template, to allow variables to be reused in different equations. We denote a specific instance i of a slot, u for example, as u^i. For brevity, we omit the instance index when a slot appears only once. To capture a correspondence between the text of x and a template T, we define an alignment p to be a set of pairs (w, s), where w is a token in x and s is a slot instance in T.

Given the above definitions, an equation e can be constructed from a template t where each number slot n is replaced with a real number, each unknown slot u is replaced with a variable, and each number constant f is kept as is. We call the process of turning a template into an equation *template instantiation*. Similarly, an equation system E is a set of l equations $\{e_0, \ldots, e_l\}$, which can be constructed by instantiating each of the equation templates in a system template T. Finally, an answer a is a tuple of real numbers.

We define a derivation y from a word problem to an answer as a tuple (T, p, a), where T is the selected system template, p is an alignment between T and x, and a is the answer generated by instantiating T using x through p and solving the generated equations. Let $\mathcal{Y}$ be the set of all derivations.

The Space of Possible Derivations We aim to map each word problem x to an equation system E. The space of equation systems considered is defined by the set of possible system templates $\mathcal{T}$ and the words in the original problem x, that are available for filling slots. In practice, we generate $\mathcal{T}$ from the training data, as described in Section 4.1. Given a system template $T \in \mathcal{T}$, we create an alignment p between T and x. The set of possible alignment pairs is constrained as follows: each number slot $n \in T$ can be aligned to any number in the text, a number word can only be aligned to a single slot n, and must be aligned to all instances of that slot. Additionally, an unknown slot instance $u \in T$ can only be aligned to a noun word. A complete derivation's alignment pairs all slots in T with words in x.

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?

$$\begin{aligned} u_1^1 + u_2^1 - n_1 &= 0 \\ n_2 \times u_1^2 + n_3 \times u_2^2 - n_4 &= 0 \end{aligned}$$

Figure 3: The first example problem and selected system template from Figure 2 with all potential aligned words marked. Nouns (boldfaced) may be aligned to unknown slot instances u_i^j, and number words (highlighted) may be aligned to number slots n_i.

Figure 3 illustrates the space of possible alignments for the first problem and system template from Figure 2. Nouns (shown in boldface) can be aligned to any of the unknown slot instances in the selected template (u_1^1, u_1^2, u_2^1, and u_2^2 for the template selected). Numbers (highlighted) can be aligned to any of the number slots (n_1, n_2, n_3, and n_4 in the template).

3.2 Probabilistic Model

Due to the ambiguity in selecting the system template and alignment, there will be many possible derivations $y \in \mathcal{Y}$ for each word problem $x \in \mathcal{X}$. We discriminate between competing analyses using a log-linear model, which has a feature function $\phi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$ and a parameter vector $\theta \in \mathbb{R}^d$. The probability of a derivation y given a problem x is defined as:

$$p(y|x;\theta) = \frac{e^{\theta \cdot \phi(x,y)}}{\sum_{y' \in \mathcal{Y}} e^{\theta \cdot \phi(x,y')}}$$

Section 6 defines the full set of features used.

The inference problem at test time requires us to find the most likely answer a given a problem

x, assuming the parameters θ are known:

$$f(x) = \arg\max_a p(a|x;\theta)$$

Here, the probability of the answer is marginalized over template selection and alignment:

$$p(a|x;\theta) = \sum_{\substack{y \in \mathcal{Y} \\ \text{s.t. AN}(y)=a}} p(y|x;\theta) \quad (1)$$

where $\text{AN}(y)$ extracts the answer a out of derivation y. In this way, the distribution over derivations y is modeled as a latent variable. We use a beam search inference procedure to approximately compute Equation 1, as described in Section 5.

4 Learning

To learn our model, we need to induce the structure of system templates in $\mathcal{T}$ and estimate the model parameters θ.

4.1 Template Induction

It is possible to generate system templates $\mathcal{T}$ when provided access to a set of n training examples $\{(x_i, E_i) : i = 1, \ldots, n\}$, where x_i is a word problem and E_i is a set of equations. We generalize each E to a system template T by (a) replacing each variable with an unknown slot, and (b) replacing each number mentioned in the text with a number slot. Numbers not mentioned in the problem text remain in the template as constants. This allows us to solve problems that require numbers that are implied by the problem semantics rather than appearing directly in the text, such as the percent problem in Figure 4.

4.2 Parameter Estimation

For parameter estimation, we assume access to n training examples $\{(x_i, \mathcal{V}_i) : i = 1, \ldots, n\}$, each containing a word problem x_i and a validation function $\mathcal{V}_i$. The validation function $\mathcal{V} : \mathcal{Y} \rightarrow \{0, 1\}$ maps a derivation $y \in \mathcal{Y}$ to 1 if it is correct, or 0 otherwise.

We can vary the validation function to learn from different types of supervision. In Section 8, we will use validation functions that check whether the derivation y has either (1) the correct system of equations E, or (2) the correct answer a. Also, using different types of validation functions on different subsets of the data enables semi-supervised learning. This approach is related to Artzi and Zettlemoyer (2013).

Word problem
A chemist has a solution that is 18 % alcohol and one that is 50 % alcohol. He wants to make 80 liters of a 30 % solution. How many liters of the 18 % solution should he add? How many liters of the 30 % solution should he add?
Labeled equations
$18 \times 0.01 \times x + 50 \times 0.01 \times y = 30 \times 0.01 \times 80$ $x + y = 80$
Induced template system
$n_1 \times 0.01 \times u_1^1 + n_2 \times 0.01 \times u_2^1 = n_3 \times 0.01 \times n_4$ $u_1^2 + u_2^2 = n_5$

Figure 4: During template induction, we automatically detect the numbers in the problem (highlighted above) to generalize the labeled equations to templates. Numbers not present in the text are considered part of the induced template.

We estimate θ by maximizing the conditional log-likelihood of the data, marginalizing over all valid derivations:

$$O = \sum_i \sum_{\substack{y \in \mathcal{Y} \\ \text{s.t. } \mathcal{V}_i(y)=1}} \log p(y|x_i;\theta)$$

We use L-BFGS (Nocedal and Wright, 2006) to optimize the parameters. The gradient of the individual parameter θ_j is given by:

$$\frac{\partial O}{\partial \theta_j} = \sum_i E_{p(y|x_i,\mathcal{V}_i(y)=1;\theta)}\left[\phi_j(x_i, y)\right] - E_{p(y|x_i;\theta)}\left[\phi_j(x_i, y)\right] \quad (2)$$

Section 5 describes how we approximate the two terms of the gradient using beam search.

5 Inference

Computing the normalization constant for Equation 1 requires summing over all templates and all possible ways to instantiate them. This results in a search space exponential in the number of slots in the largest template in $\mathcal{T}$, the set of available system templates. Therefore, we approximate this computation using beam search. We initialize the beam with all templates in $\mathcal{T}$ and iteratively align slots from the templates in the beam to words in the problem text. For each template, the next slot

to be considered is selected according to a predefined canonicalized ordering for that template. After each iteration we prune the beam to keep the top-k partial derivations according to the model score. When pruning the beam, we allow at most l partial derivations for each template, to ensure that a small number of templates don't monopolize the beam. We continue this process until all templates in the beam are fully instantiated.

During learning we compute the second term in the gradient (Equation 2) using our beam search approximation. Depending on the available validation function $\mathcal{V}$ (as defined in Section 4.2), we can also accurately prune the beam for the computation of the first half of the gradient. Specifically, when assuming access to labeled equations, we can constrain the search to consider only partial hypotheses that could possibly be completed to produce the labeled equations.

6 Model Details

Template Canonicalization There are many syntactically different but semantically equivalent ways to express a given system of equations. For example, the phrase "John is 3 years older than Bill" can be written as $j = b + 3$ or $j - 3 = b$. To avoid such ambiguity, we canonicalize templates into a normal form representation. We perform this canonicalization by obtaining the symbolic solution for the unknown slots in terms of the number slots and constants using the mathematical solver Maxima (Maxima, 2014).

Slot Signature In a template like $s_1 + s_2 = s_3$, the slot s_1 is distinct from the slot s_2, but we would like them to share many of the features used in deciding their alignment. To facilitate this, we generate signatures for each slot and slot pair. The signature for a slot indicates the system of equations it appears in, the specific equation it is in, and the terms of the equation it is a part of. Pairwise slot signatures concatenate the signatures for the two slots as well as indicating which terms are shared. This allows, for example, n_2 and n_3 in Derivation 1 in Figure 2 to have the same signature, while the pairs $\langle n_2, u_1 \rangle$ and $\langle n_3, u_1 \rangle$ have different ones. To share features across templates, slot and slot-pair signatures are generated for both the full template, as well as for each of the constituent equations.

Features The features $\phi(x, y)$ are computed for a derivation y and problem x and cover all derivation decisions, including template and alignment selection. When required, we use standard tools to generate part-of-speech tags, lematizations, and dependency parses to compute features.[2] For each number word in y we also identify the closest noun in the dependency parse. For example, the noun for 278 in Derivation 1, Figure 2 would be "people." The features are calculated based on these nouns, rather than the number words.

Document level
Unigrams
Bigrams
Single slot
Has the same lemma as a question object
Is a question object
Is in a question sentence
Is equal to one or two (for numbers)
Word lemma X nearby constant
Slot pair
Dep. path contains: Word
Dep. path contains: Dep. Type
Dep. path contains: Word X Dep. Type
Are the same word instance
Have the same lemma
In the same sentence
In the same phrase
Connected by a preposition
Numbers are equal
One number is larger than the other
Equivalent relationship
Solution Features
Is solution all positive
Is solution all integer

Table 1: The features divided into categories.

We use four types of features: document level features, features that look at a single slot entry, features that look at pairs of slot entries, and features that look at the numeric solutions. Table 1 lists all the features used. Unless otherwise noted, when computing slot and slot pair features, a separate feature is generated for each of the signature types discussed earlier.

Document level features Oftentimes the natural language in x will contain words or phrases which are indicative of a certain template, but are not associated with any of the words aligned to slots in the template. For example, the word "chemist"

[2] In our experiments these are generated using the Stanford parser (de Marneffe et al., 2006)

might indicate a template like the one seen in Figure 4. We include features that connect each template with the unigrams and bigrams in the word problem. We also include an indicator feature for each system template, providing a bias for its use.

Single Slot Features The natural language x always contains one or more questions or commands indicating the queried quantities. For example, the first problem in Figure 2 asks "How many children were admitted on that day?" The queried quantities, the number of children in this case, must be represented by an unknown in the system of equations. We generate a set of features which look at both the word overlap and the noun phrase overlap between slot words and the objects of a question or command sentence. We also compute a feature indicating whether a slot is filled from a word in a question sentence. Additionally, algebra problems frequently use phrases such as "2 kinds of tickets" (e.g., Figure 2). These numbers do not typically appear in the equations. To account for this, we add a single feature indicating whether a number is one or two. Lastly, many templates contain constants which are identifiable from words used in nearby slots. For example, in Figure 4 the constant 0.01 is related to the use of "%" in the text. To capture such usage, we include a set of lexicalized features which concatenate the word lemma with nearby constants in the equation. These features do not include the slot signature.

Slot Pair Features The majority of features we compute account for relationships between slot words. This includes features that trigger for various equivalence relations between the words themselves, as well as features of the dependency path between them. We also include features that look at the numerical relationship of two numbers, where the numeric values of the unknowns are generated by solving the system of equations. This helps recognize that, for example, the total of a sum is typically larger than each of the (typically positive) summands.

Additionally, we also have a single feature looking at shared relationships between pairs of slots. For example, in Figure 2 the relationship between "tickets for children" and "$1.50" is "cost". Similarly the relationship between "Adult tickets" and "$4" is also "cost". Since the actual nature of this relationship is not important, this feature is not lexicalized, instead it is only triggered for the presence of equality. We consider two cases: subject-object relationships where the intervening verb is equal, and noun-to-preposition object relationships where the intervening preposition is equal.

# problems	514
# sentences	1616
# words	19357
Vocabulary size	2352
Mean words per problem	37
Mean sentences per problem	3.1
Mean nouns per problem	13.4
# unique equation systems	28
Mean slots per system	7
Mean derivations per problem	4M

Table 2: Dataset statistics.

Solution Features By grounding our semantics in math, we are able to include features which look at the final answer, a, to learn which answers are reasonable for the algebra problems we typically see. For example, the solution to many, but not all, of the problems involves the size of some set of objects which must be both positive and integer.

7 Experimental Setup

Dataset We collected a new dataset of algebra word problems from Algebra.com, a crowdsourced tutoring website. The questions were posted by students for members of the community to respond with solutions. Therefore, the problems are highly varied, and are taken from real problems given to students. We heuristically filtered the data to get only linear algebra questions which did not require any explicit background knowledge. From these we randomly chose a set of 1024 questions. As the questions are posted to a web forum, the posts often contained additional comments which were not part of the word problems and the solutions are embedded in long free-form natural language descriptions. To clean the data we asked Amazon Mechanical Turk workers to extract from the text: the algebra word problem itself, the solution equations, and the numeric answer. We manually verified both the equations and the numbers to ensure they were correct. To ensure each problem type is seen at least a few times in the training data, we removed the infrequent problem types. Specifically, we induced the system template from each equation system, as described in Section 4.1, and removed all problems for which the associated system template appeared

less than 6 times in the dataset. This left us with 514 problems. Table 2 provides the data statistics.

Forms of Supervision We consider both semi-supervised and supervised learning. In the semi-supervised scenario, we assume access to the numerical answers of all problems in the training corpus and to a small number of problems paired with full equation systems. To select which problems to annotate with equations, we identified the five most common types of questions in the data and annotated a randomly sampled question of each type. 5EQ+ANS uses this form of weak supervision. To show the benefit of using the weakly supervised data, we also provide results for a baseline scenario 5EQ, where the training data includes *only* the five seed questions annotated with equation systems. In the fully supervised scenario ALLEQ, we assume access to full equation systems for the entire training set.

Evaluation Protocol We run all our experiments using 5-fold cross-validation. Since our model generates a solution for every problem, we report only accuracy. We report two metrics: equation accuracy to measure how often the system generates the correct equation system, and answer accuracy to evaluate how often the generated numerical answer is correct. When comparing equations, we avoid spurious differences by canonicalizing the equation system, as described in Section 6. To compare answer tuples we disregard the ordering and require each number appearing in the reference answer to appear in the generated answer.

Parameters and Solver In our experiments we set k in our beam search algorithm (Section 5) to 200, and l to 20. We run the L-BFGS computation for 50 iterations. We regularize our learning objective using the L^2-norm and a λ value of 0.1. The set of mathematical relations supported by our implementation is $\{+, -, \times, /\}$.Our implementation uses the Gaussian Elimination function in the Efficient Java Matrix Library (EJML) (Abeles, 2014) to generate answers given a set of equations.

8 Results

8.1 Impact of Supervision

Table 3 summarizes the results. As expected, having access to the full system of equations (ALLEQ) at training time results in the best learned model, with nearly 69% accuracy. However, training from primarily answer annotations (5EQ+ANS) results in performance which is almost 70% of ALLEQ, demonstrating the value of weakly supervised data. In contrast, 5EQ, which cannot use this weak supervision, performs much worse.

	Equation accuracy	Answer accuracy
5EQ	20.4	20.8
5EQ+ANS	45.7	46.1
ALLEQ	66.1	68.7

Table 3: Cross-validation accuracy results for various forms of supervision.

	Equation accuracy	Answer accuracy	% of data
≤ 10	43.6	50.8	25.5
$11-15$	46.6	45.1	10.5
$16-20$	44.2	52.0	11.3
>20	85.7	86.1	52.7

Table 4: Performance on different template frequencies for ALLEQ.

8.2 Performance and Template Frequency

To better understand the results, we also measured equation accuracy as a function of the frequency of each equation template in the data set. Table 4 reports results for ALLEQ after grouping the problems into four different frequency bins. We can see that the system correctly answers more than 85% of the question types which occur frequently while still achieving more than 50% accuracy on those that occur relatively infrequently. We do not include template frequency results for 5EQ+ANS since in this setup our system is given only the top five most common templates. This limited set of templates covers only those questions in the >20 bin, or about 52% of the data. However, on this subset 5EQ+ANS performs very well, answering 88% of them correctly, which is approximately the same as the 86% achieved by ALLEQ. Thus while the weak supervision is not helpful in generating the space of possible equations, it is very helpful in learning to generate the correct answer when given an appropriate space of equations.

8.3 Ablation Analysis

Table 5 shows ablation results for each group of features. The results along the diagonal show the performance when a single group of features is ablated, while the off-diagonal numbers show the

	w/o pair	w/o document	w/o solution	w/o single
w/o pair	42.8	25.7	19.0	39.6
w/o document	–	63.8	50.4	57.6
w/o solution	–	–	63.6	62.0
w/o single	–	–	–	65.9

Table 5: Cross-validation accuracy results with different feature groups ablated for ALLEQ. Results are for answer accuracy which is 68.7% without any features ablated.

performance when two groups of features are ablated together. We can see that all of the features contribute to the overall performance, and that the pair features are the most important followed by the document and solution features. We also see that the pair features can compensate for the absence of other features. For example, the performance drops only slightly when either the document or solution features are removed in isolation. However, the drop is much more dramatic when they are removed along with the pair features.

8.4 Qualitative Error Analysis

We examined our system output on one fold of ALLEQ and identified two main classes of errors.

The first, accounting for approximately one-quarter of the cases, includes mistakes where more background or world knowledge might have helped. For example, Problem 1 in Figure 5 requires understanding the relation between the dimensions of a painting, and how this relation is maintained when the painting is printed, and Problem 2 relies on understanding concepts of commerce, including cost, sale price, and profit. While these relationships could be learned in our model with enough data, as it does for percentage problems (e.g., Figure 4), various outside resources, such as knowledge bases (e.g. Freebase) or distributional statistics from a large text corpus, might help us learn them with less training data.

The second category, which accounts for about half of the errors, includes mistakes that stem from compositional language. For example, the second sentence in Problem 3 in Figure 5 could generate the equation $2x - y = 5$, with the phrase "twice of one of them" generating the expression $2x$. Given the typical shallow nesting, it's possible to learn templates for these cases given enough data, and in the future it might also be possible to develop new, cross-sentence semantic parsers to enable better generalization from smaller datasets.

(1)	A painting is 10 inches tall and 15 inches wide. A print of the painting is 25 inches tall, how wide is the print in inches?
(2)	A textbook costs a bookstore 44 dollars, and the store sells it for 55 dollars. Find the amount of profit based on the selling price.
(3)	The sum of two numbers is 85. The difference of twice of one of them and the other one is 5. Find both numbers.
(4)	The difference between two numbers is 6. If you double both numbers, the sum is 36. Find the two numbers.

Figure 5: Examples of problems our system does not solve correctly.

9 Conclusion

We presented an approach for automatically learning to solve algebra word problems. Our algorithm constructs systems of equations, while aligning their variables and numbers to the problem text. Using a newly gathered corpus we measured the effects of various forms of weak supervision on performance. To the best of our knowledge, we present the first learning result for this task.

There are still many opportunities to improve the reported results, and extend the approach to related domains. We would like to develop techniques to learn compositional models of meaning for generating new equations. Furthermore, the general representation of mathematics lends itself to many different domains including geometry, physics, and chemistry. Eventually, we hope to extend the techniques to synthesize even more complex structures, such as computer programs, from natural language.

Acknowledgments

The authors acknowledge the support of Battelle Memorial Institute (PO#300662) and NSF (grant IIS-0835652). We thank Nicholas FitzGerald, the MIT NLP group, the UW NLP group and the ACL reviewers for their suggestions and comments. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the funding organizations.

References

Peter Abeles. 2014. Efficient java matrix library. https://code.google.com/p/efficient-java-matrix-library/.

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

S.R.K Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics.*

S.R.K Branavan, Luke Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics.*

Qingqing Cai and Alexander Yates. 2013a. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Qingqing Cai and Alexander Yates. 2013b. Semantic parsing freebase: Towards open-domain semantic parsing. In *Proceedings of the Joint Conference on Lexical and Computational Semantics.*

Nathanael Chambers and Dan Jurafsky. 2011. Template-based information extraction without the templates. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

David Chen and Raymond Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Conference on Artificial Intelligence.*

David Chen. 2012. Fast online lexicon learning for grounded language acquisition. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Conference on Computational Natural Language Learning.* Association for Computational Linguistics.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the Conference on Language Resources and Evaluation*.

Ralph Grishman, David Westbrook, and Adam Meyers. 2005. NYUs English ACE 2005 System Description. In *Proceedings of the Automatic Content Extraction Evaluation Workshop*.

Heng Ji and Ralph Grishman. 2008. Refining event extraction through cross-document inference. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Joohyun Kim and Raymond Mooney. 2012. Unsupervised pcfg induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Nate Kushman and Regina Barzilay. 2013. Using semantic unification to generate regular expressions from natural language. In *Proceeding of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*.

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 Conference on Empirical Methods on Natural Language Processing.*

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of Empirical Methods in Natural Language Processing*.

Tao Lei, Fan Long, Regina Barzilay, and Martin Rinard. 2013. From natural language specifications to program input parsers. In *Proceeding of the Association for Computational Linguistics.*

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

Mstislav Maslennikov and Tat-Seng Chua. 2007. A multi-resolution framework for information extraction from free text. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Cynthia Matuszek, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. 2012. A joint model of language and perception for grounded attribute learning. In *Proceedings of the International Conference on Machine Learning*.

Maxima. 2014. Maxima, a computer algebra system. version 5.32.1.

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

Jorge Nocedal and Stephen Wright. 2006. Numerical optimization, series in operations research and financial engineering. *Springer, New York*.

Roi Reichart and Regina Barzilay. 2012. Multi-event extraction guided by global constraints. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*.

Alan Ritter, Mausam, Oren Etzioni, and Sam Clark. 2012. Open domain event extraction from twitter. In *Proceedings of the Conference on Knowledge Discovery and Data Mining*.

Cynthia Thompson and Raymond Mooney. 2003. Acquiring word-meaning mappings for natural language interfaces. *Journal of Artificial Intelligence Research*, 18(1).

Adam Vogel and Dan Jurafsky. 2010. Learning to follow navigational directions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Annual Meeting of the North American Chapter of the Association of Computational Linguistics*. Association for Computational Linguistics.

John Zelle and Raymond Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Conference on Artificial Intelligence*.

Luke Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Luke Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing*.