Automatically Solving Algebra Word Problems with Structured Prediction Energy Networks

Nikhil Yadav
nikhilyadav@umass.edu

Arpit Jain
jain@umass.edu

Gota Gando
ggando@umass.edu

Krishna Prasad Sankaranarayanan
ksankaranara@umass.edu

Abstract

To solve algebra word problems automatically, typical approaches find a transformation from the given word problem into a set of equations which correctly represents the input word problem. To approach this task, we apply structured prediction energy networks (SPENs), which is an energy based model where we can inject the structural knowledge of the task. We compare our SPEN to a simple greedy baseline approach to determine the effectiveness of modeling structural dependencies in this problem.

1 Introduction

Algebra word problems in general describe some particular world situation and pose a question about it. In this project, we only consider the problems where we can use a system of equations to solve the questions. Figure 1 shows one example of such problems. In this case, we can express the problem as a set of algebraic equations that describe the mathematical relationship of entities in the problem: for example $m = 4 \times n$ and $m - 4 = 6 \times (n - 4)$ where m and n represent the age of Maria and Kate, respectively. To answer the question, we need to solve these equations and get the solution. However, we focus on finding a mapping from the word problem into an equation system in this project, as there are many automated solvers for this process.

Maria is now four times as old as Kate. Four years ago, Maria was six times as old as Kate. Find their ages now.

Figure 1: An example of algebra word problem.

1.1 Structured Prediction

If we consider each character of the output equations as a random variable, then this task can be considered as a structured prediction task, as every output random variable would be dependent on some other variables. Furthermore, as both the input and output are sequences, we can also interpret this task as a sequence-to-sequence task.

In many machine learning tasks, we try to predict $\mathbf{y}$ given an input $\mathbf{x}$. In some cases it is sufficient to use a discriminative function F to predict the output such as $\mathbf{y} = F(\mathbf{x})$. However, this model fails to model the structure of the output and the interactions among output components and may perform poor on more complex tasks. We can instead use *an energy function* E that both depends on $\mathbf{x}$ and $\mathbf{y}$ to model such structure and seek the optimal $\mathbf{y}$ that minimizes the energy:

$$\hat{\mathbf{y}} = argmin_{\mathbf{y}} E(\mathbf{y}, \mathbf{x}) \tag{1}$$

Energy function can be seen as a scoring function that returns how compatible $\mathbf{x}$ and $\mathbf{y}$ are, or how likely they occur together. One thing we need to note here is that there is no guarantee of the returning value of energy functions will be in the range of $\{0, 1\}$. One common way to address this problem and calibrate the energy is putting it through the Gibbs distribution.

$$P(Y|X) = \frac{\exp -\beta E(Y, X)}{\int_{y \in (Y)} \exp -\beta E(y, X)} \tag{2}$$

However, it should be noted that this transformation is only possible when computing the integral term $\int_{y \in (Y)} \exp -\beta E(y, X)$ is tractable.

2 Related Works

Solving word algebra problems automatically is a long-standing task in natural language processing (NLP). In this context, solving arithmetic word

problems is of specific interest. One key challenge of solving algebra word problems is the lack of fully annotated data. Solving these problems require reasoning and logic across sentence boundaries to find a system of equations that precisely models the described semantic and mathematical relationships.

(Kushman et al., 2014) presented a baseline approach to solving algebra word problems using varied supervision, including either full equations or just the final answers . In this two step algorithm, a template is first selected to match the overall structure of the equation system followed by which it is instantiated with numbers and nouns from the test. In this probabilistic inference mode, the probability of the answer is marginalized over the template selection and alignment.Following template induction wherein labeled equations are generalized to templates, inference is performed y beam search. Beam search provides a mapping to each template based on a canonical ordering. The results on primarily answer annotations (weakly supervised data) were 70%of the accuracies over the complete set of equations which clearly justified the benefits of supervision.

(Roy and Roth, 2016) proposed a novel method for solving multi-step arithmetic word problems without the assistance of predefined annotations and templates. The algorithm is based on multiple classification problems on the target arithmetic equation and then combining these results to obtain the solutions. Grounding is the task taking an input word problem in the natural language and representing it as a formal language such as a set of equations, expression trees or states. This paper achieves grounding as a single step process. On the other hand,(Mitra and Baral, 2016), split this up into two parts. In the first step, the system learns to connect the assertions in a word problems to formula and in the second step, maps the formula into an algebraic equation.

(Koncel-Kedziorski et al., 2015) formalizes solving algebraic word problems as that of generating and evaluating equation trees.Unlike in (Roy, 2017), wherein a system for reasoning about quantities is introduced for arithmetic word problems involving only two values from the text and an arithmetic operator, this method (ALGES) learns to solve complex problems with multiple operands where the space of possible solutions is larger. Quantified sets (Qsets) are used a building block for equation trees to model natural language text quantities. These Qsets are combined with operator to yield a semantically augmented equation tree. With respect to grounding, the Qset properties are extracted and then ordered based on semantic and textual constraints followed by which Qsets are combined with arithmetic operators in an equation tree representation. The inference is to find the most likely equation tree with minimum violation of hard and soft constraints. This is facilitated by calculating a likelihood score for each operand t and the Qsets over which it is operated. Comparison results between template based methods and ALGES show that although the template based methods are able to solve a wider range of problems than ALGES, ALGES fairs well fairs well even in systems with fewer repeated templates or less spurious lexical overlap between problems. In conclusion ALGES is a hybrid of template based and verb categorization methods.

(Upadhyay et al., 2016) demonstrates a state of the art approach for solving algebra word problems with little or no manual annotation of equations. In this novel structured-output learning) algorithm (MixedSP), both explicit (eg., equations) and implicit(eg.,solutions) supervision signals are leveraged jointly.In the case of using algorithms that learn from implicit supervision, the system does not model directly the relation between input x and solution z. Also, multiple combinations of templates and alignments could end up with the same solution making implicit supervision slow, intractable and noisy. To overcome this limitation, MixedSP uses a standard structured prediction update procedure to find the best scoring candidate structures among the templates and alignments. Th algorithm starts with just explicit signals since it leads to a better intermediate model which later allows exploring the output space more efficiently using the implicit signals. The results clearly demonstrate that a joint learning approach benefits from noisy implicit supervision. The accuracies are represented for Explicit, Implicit, Pseudo, Pseudo + Explicit and MixedSP. Mixed SP outperforms it's counterparts even without manual annotation.

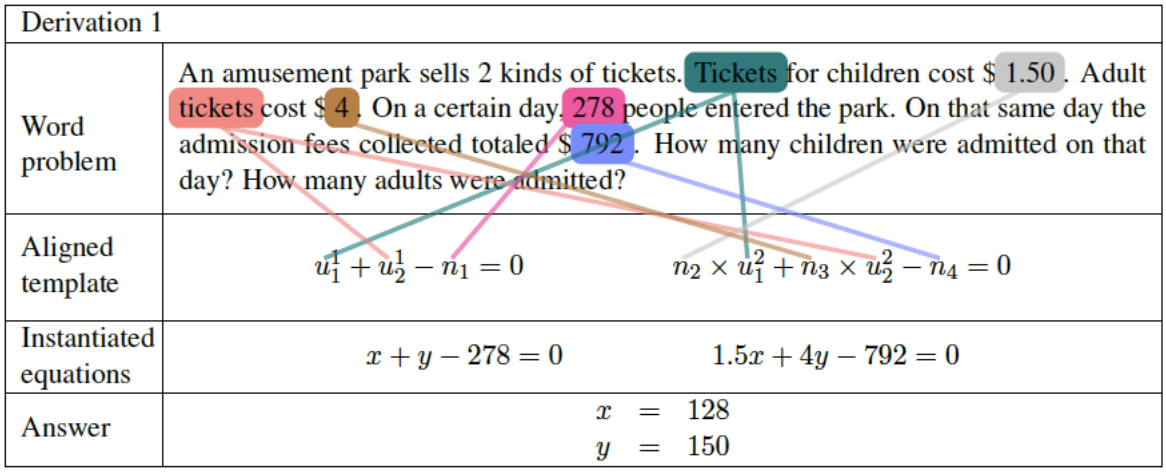

Figure 2: The structure of the baseline model. Cited from (Kushman et al., 2014)

3 Problem Definition

3.1 Template

Following (Kushman et al., 2014), we define the space of possible equations by a set of "templates". A template represents a structure for a set of equations for an algebra problem, consisting of slots for unknown variables and coefficients. As we express equations as a sequence of symbol indices, there can be many possible correct equations. Therefore, we assume that each text problem has only one correct set of equations, and express the structure with a template. Figure 2 shows an example of template selection and instantiation. In this case the template has u_i slots that represent unknown variables, and n_j slots for coefficients values. In this scheme, transformation of word problems to equations as a two step process. First, a template is selected to define the structure of the target equation system. Second, the selected template is instantiated with actual texts for unknowns and values for coefficients.

3.2 Derivation

In this section, we review the problem formulation described in (Kushman et al., 2014) and (Upadhyay et al., 2016). We denote the input word problem as $\mathbf{x}$, and the output as $\mathbf{y}$. We call the output as a derivation, which consists of a template T and an alignment A. Therefore we can express the output as $\mathbf{y} = (T, A)$. More specifically, template T is a set of equations $= \{t_0, ..., t_l\}$, and an alignment A is defined as a set of pairs (w, s), where where w is a token in $\mathbf{x}$ and s is a slot instance in T. In the methods we describe in the following sections, we use this joint structure for training the models. More precisely, this is a vector of the template index and values for the template's slots as shown in Figure 3.

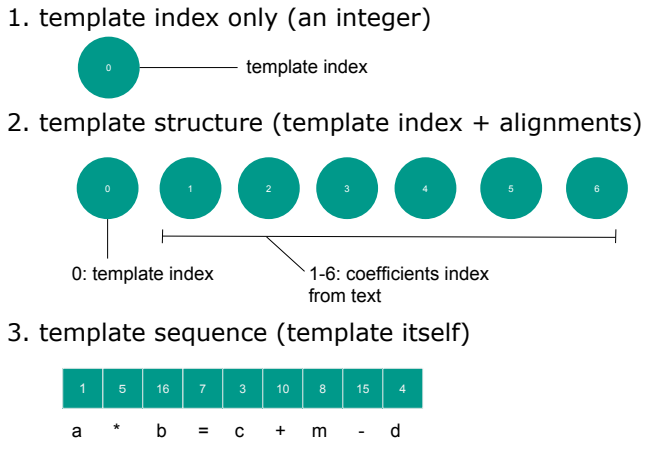

Figure 3: The structures of $\mathbf{y}$ we use in our experiments. 1. A structure only with the template index, 2. The structure with the template index and alignments (Derivation): rather than using the entire sequence to express equations, we use a template index and its slots to specify a set of equations. 3. Template sequence: with this structure we predict the template itself as a sequence.

3.3 Additional output structures

We also evaluate the SPEN with two extra structures: one is a structure only containing the template index, and the other one is the template sequence. In the latter setting, we predict the template itself as a sequence of mathematical sym-

bols, rather than a vector of the template index and alignments.

3.4 Datasets

We use two datasets; the ALG-514 dataset proposed by (Kushman et al., 2014) which contains 514 word problems, and the DRAW dataset provided by Microsoft that contains about 1000 independent examples. One issue we are currently aware of these datasets is that, while they include template structure of output equations for each problem, they do not provide the mapping of unknowns to text values. We plan to manually annotate such alignment information on these datasets so that we can fully make use of the template structure and its alignments.

4 Methods

4.1 Scoring function

4.1.1 Rules for the template index

4.1.2 Rules for the derivation

We first describe our manually constructed scoring function, which is used for both the baseline model and SPEN. This function takes in as a predicted derivation and the true derivation, and returns the score of how good the prediction is. We also pass the problem text and the solution (e.g. $x = 2, y = 3$) of the true equation to this function so that it can utilize the full information. It first compares the predicted template index with the true template index, and punishes significantly if the predicted index differs from the true index. This is because if the derivation uses a different template, two derivations would be completely different. If the template index matches, then the function examines how the coefficients are matching; for each coefficient slot, it computes the distance between the predicted token index and the true token index, and add a negative value to the score proportional to the distance. Finally, the function solves the predicted equations and checks if the solutions are matching the actual solutions of the true equations. If the solution does not match, it penalizes the final score. Often times, the alignments in the predicted derivation do not point to actual number words such as "two" or "1,000". In those cases we search for the nearest number word in the problem text, and build the predicted equation with the filled out coefficients.

4.1.3 Rules for the template sequence

As we are predicting a set of equations for the template sequence structure, we create a scoring function which penalizes invalid predictions. For example, a valid equation does not contain the equal symbol "=" nor operator symbols (e.g. "+", "-") at the very beginning.

4.2 Baseline model

To evaluate the performance of SPEN, We build a greedy baseline which determines the prediction for a word problem component by component. It first determines the component for template index by checking the scores from all the possible indices with the above mentioned scoring function. After fixing the template, it then proceeds with determining the value for each coefficient slot greedily.

4.3 SPEN

Structured Prediction Energy Networks (SPENs) (Belanger and McCallum, 2015)(Belanger et al., 2017) are one of the energy-based models (EBMs) that performs structured prediction, such as conditional random fields (CRFs), structured perceptrons, and structured SVMs (SSVMs). There are two main differences between SPEN and other energy-based models. First, SPENs can learn nonlinear energy functions, as it is parameterized by a deep neural network. Similar to other EBMs, SPEN also tries to predict y that minimizes the energy function. While EBMs listed above usually assume a restricted graphical structure such as a chain or a tree and consider a linear energy function, SPEN instead considers a general energy function and assumes the non-convexity. Then, it approximates the optimal $\mathbf{y}$ with gradient descent:

$$\bar{\mathbf{y}} = argmin_{\bar{\mathbf{y}}} E(\bar{\mathbf{y}}, \mathbf{x}) \tag{3}$$

$$\mathbf{y}^{t+1} = \mathbf{y}^{t} - \eta \frac{\partial E}{\partial \mathbf{y^t}} \tag{4}$$

Additionally, SPENs are much more efficient at the inference stage where the model predicts a candidate output $\bar{\mathbf{y}}$, as SPENs use gradient descent to approximate an optimal $\bar{\mathbf{y}}$ instead of using computationally expensive searching methods such as viterbi algorithm or beam search.

4.4 Architecture of SPEN

Although SPEN can take general energy functions, in this work we consider the energy function

consists of the global energy and the sum of local energies as follows:

$$E_x^{local}(\bar{y}) = \sum_{i=1}^{L} \bar{y}_i b_i^T F(x) \quad (5)$$

$$E_x^{global}(\bar{y}) = c_2^T g(C_1 \bar{y}) \quad (6)$$

where g is a non-linearity function, each b_i is a vector of parameters for each label, and the product $C_1\bar{y}$ is a set of learned affine (linear + bias) measurements of the output.

We give the same representation of y as described in the baseline section to this energy network.

4.5 Rank-based loss

The traditional SPEN framework uses the SSVM loss:

$$\sum_{\{x_i,y_i\}} \max_y \left[\Delta(y_i, y) - E_{x_i}(y) + E_{x_i}(y_i)\right]_+ \quad (7)$$

Here, $+[\cdot]$ denotes the max function. However, this loss requires the full annotation for derivations. As our dataset only contains partial annotations for alignments, we use a rank-based loss which is defined as below:

$$\min_{\mathbf{w}} \sum_{\mathbf{x}\in\mathcal{D}} \left[\alpha(V(\mathbf{y}_h,\mathbf{x}) - V(\mathbf{y}_l,\mathbf{x})) - E_{\mathbf{w}}(\mathbf{y}_h,\mathbf{x}) + E_{\mathbf{w}}(\mathbf{y}_l,\mathbf{x})\right]_+ \quad (8)$$

Where $\mathbf{y}_h$ is the predicted output and $\mathbf{y}_l$ is the true configuration. V is the value of the scoring function, which returns the goodness of $\mathbf{y}$ given a word problem $\mathbf{x}$. The intuition of this loss is that we would like the difference between energy values of the prediction and true $\mathbf{y}$ to be roughly the same as the difference between the scores, as depicted in Figure 4. In other words, we would like to approximate the scoring function with the energy networks. One big advantage of this approach is that, unlike the raw scoring function energy networks are differentiable and have less local optima. Therefore, we do not need to enumerate all the possible configurations for y and we can obtain the minimum y that minimizes the energy function automatically, as described in Section 4.3. Additionally, as the scoring function can be arbitrary, we can inject the domain knowledge for this task. For example, as each data sample contains the true solution pairs from equations, intuitively if the solutions from the predicted equations match with them, we can imagine that the prediction is more likely to be correct. We can implement a scoring function that gives higher score to such predictions, as described in Section 4.1.

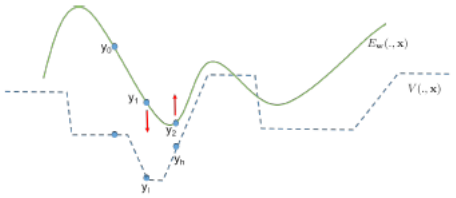

Figure 4: The illustration of the rank-based loss.

5 Experiments

5.1 Preliminary experiment

To check if our structure of the output $\mathbf{y}$ is actually learnable by a neural network, we first trained a small MLP model with a subset of the ALG-514 dataset. The network consists of one embedding layer, one dense layer and 7 softmax layers. We trained the network with a joint loss which is the sum of all the losses from each softmax layer. The embedding layer is initialized with the 50 dimensional Glove vector. We confirmed that the network successfully overfit to the first 10 examples of the dataset and achieved 1.0 accuracy and 0.0 loss value for the train set after 50 epochs. The illustration of the architecture of the network is shown in Figure 5.

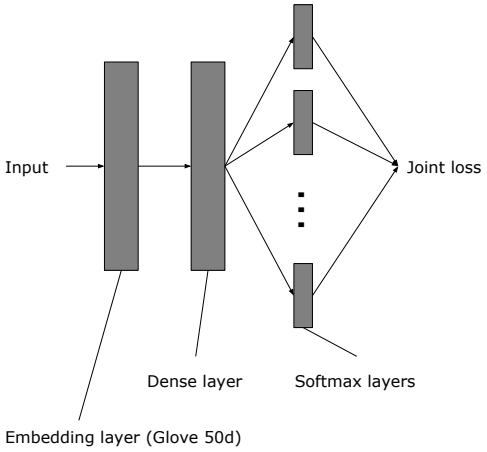

Figure 5: The architecture of the MLP model.

5.2 Experimental settings for SPEN

Next, we evaluated the performance of the baseline model and the SPEN with the full dataset of ALG-514. We split the data into the train set containing 412 examples and the validation set containing 102 examples. We show the architecture of the energy networks in Figure 6. The whole network is end-to-end; We first pass the input to

the pretrained bi-LSTM layer to extract useful features from the input. This is the equivalent to $F(\mathbf{x})$ in SPEN. The output of this layer will be passed into the global energy network and local energy network, and then we take the sum of the output values from those networks and compute the scalar energy value in the end. We implemented our scoring function using sympy, and we trained the SPEN with both the SSVM loss and the rank-based loss. On training SPEN, we first trained a neural network with a supervised sequence-to-sequence task with the dataset and fine-tuned the glove initialized embedding layer. Then, we initialized the embedding layer of SPEN with the pretrained deep neural network and trained the model with a rank-based loss. Hyperparameters used during the training are shown in Table 1. We tested the three output structure types as described in a previous section; template index, template index with alignments, and template sequence. We performed experiments for each of the output structure with the same architecture except the scoring function. In the following sections, we call the experiments for template index, tempalte alignments, template sequence as Experiment 1, Experiment2, and Experiment 3 respectively.

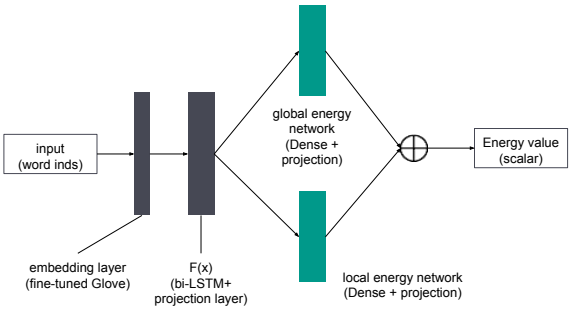

Figure 6: The architecture of the SPEN model used in the experiment.

Table 1: Hyperparameters of the model

Parameter name	**Value**
Learning rate	0.001
Embedding dim	50
Batch size	100
Inference iterations	50
Inference rate	0.1
Margin weight	100

6 Results and Discussion

Tables 2 - 4 show the accuracies for the second experiment. In the Experiment 2 and Experiment 3, both approaches of the SPEN outperformed the greedy baseline model, and the model trained with the rank-based loss achieved about a 4% higher accuracy. However, this result is very lower than the accuracy reported by (Kushman et al., 2014),

Table 2: Experimental results for Experiment 1 on the ALG-514 dataest. Each cell represents the accuracy of a method.

Method	**Accuracy**
MLP	0.631
SPEN (SSVM loss)	0.420
SPEN (Rank-based loss)	

Table 3: Results for Experiment 2.

Method	**Accuracy**
Baseline (greedy)	0.024
MLP	0.14
SPEN (SSVM loss)	0.05
SPEN (Rank-based loss)	0.15

Table 4: Results for Experiment 2.

Method	**Accuracy**
MLP	0.173
SPEN (SSVM loss)	0.055
SPEN (Rank-based loss)	0.368

7 Conclusion & Future Work

We have created a minimal pipeline to apply SPEN to algebra word problems, and evaluated SPEN in both supervised and unsupervised settings on the ALG-514 dataset. However, experimental results show that SPEN did not perform as well as we expected. We are suspecting that our scoring function is not sophisticated enough, and our SPEN architecture is not tuned sufficiently to fit to this task. In future, aside from tuning the model further, we are interested in adopting a structure which can scale to the number of types in equations, such as rationales proposed by (Ling et al., 2017).

References

David Belanger and Andrew McCallum. 2015. Structured prediction energy networks. *CoRR*, abs/1511.06350.

David Belanger, Bishan Yang, and Andrew McCallum. 2017. End-to-end learning for structured prediction energy networks. *arXiv preprint arXiv:1703.05667*.

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

Yann Lecun, Sumit Chopra, Raia Hadsell, Fu Jie Huang, G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar (eds. 2006. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *CoRR*, abs/1705.04146.

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *ACL*, volume 1, pages 2144–2153.

Subhro Roy. 2017. *Reasoning about quantities in natural language*. Ph.D. thesis, University of Illinois at Urbana-Champaign.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Shyam Upadhyay, Ming-Wei Chang, Kai-Wei Chang, and Wen-tau Yih. 2016. Learning from explicit and implicit supervision jointly for algebra word problems. In *EMNLP*.