Learning by Fixing: Solving Math Word Problems with Weak Supervision

Yining Hong,[1] **Qing Li,**[1] **Daniel Ciao,**[1] **Siyuan Haung,**[1] **Song-Chun Zhu**[1]
[1] University of California, Los Angeles, USA.
yininghong@cs.ucla.edu, {liqing, danielciao, huangsiyuan}@ucla.edu, sczhu@stat.ucla.edu

Abstract

Previous neural solvers of math word problems (MWPs) are learned with full supervision and fail to generate diverse solutions. In this paper, we address this issue by introducing a *weakly-supervised* paradigm for learning MWPs. Our method only requires the annotations of the final answers and can generate various solutions for a single problem. To boost weakly-supervised learning, we propose a novel *learning-by-fixing* (LBF) framework, which corrects the misperceptions of the neural network via symbolic reasoning. Specifically, for an incorrect solution tree generated by the neural network, the *fixing* mechanism propagates the error from the root node to the leaf nodes and infers the most probable fix that can be executed to get the desired answer. To generate more diverse solutions, *tree regularization* is applied to guide the efficient shrinkage and exploration of the solution space, and a *memory buffer* is designed to track and save the discovered various fixes for each problem. Experimental results on the Math23K dataset show the proposed LBF framework significantly outperforms reinforcement learning baselines in weakly-supervised learning. Furthermore, it achieves comparable top-1 and much better top-3/5 answer accuracies than fully-supervised methods, demonstrating its strength in producing diverse solutions.

Introduction

Solving math word problems (MWPs) poses unique challenges for understanding natural-language problems and performing arithmetic reasoning over quantities with commonsense knowledge. As shown in Figure 1, a typical MWP consists of a short narrative describing a situation in the world and asking a question about an unknown quantity. To solve the MWP in Figure 1, a machine needs to extract key quantities from the text, such as "100 kilometers" and "2 hours", and understand the relationships between them. General mathematical knowledge like "distance = velocity $\times$ time" is then used to calculate the solution.

Researchers have recently focused on solving MWPs using neural-symbolic models (Ling et al. 2017; Wang, Liu, and Shi 2017; Huang et al. 2018; Wang et al. 2018; Xie and Sun 2019). These models usually consist of a neural perception module (*i.e.*, Seq2Seq or Seq2Tree) that maps the problem text into a solution expression or tree, and a symbolic module which executes the expression and generates the final answer. Training these models requires the full supervision of the solution expressions.

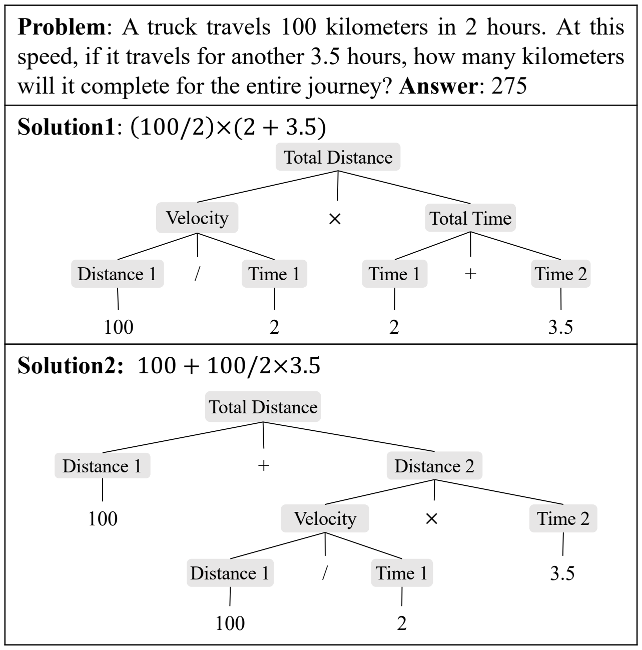

Problem: A truck travels 100 kilometers in 2 hours. At this speed, if it travels for another 3.5 hours, how many kilometers will it complete for the entire journey? **Answer**: 275
Solution1: (100/2)×(2 + 3.5)
Total Distance
Velocity
×
Total Time
Distance 1
/
Time 1
Time 1
+
Time 2
100
2
2
3.5
Solution2: 100 + 100/2×3.5
Total Distance
Distance 1
+
Distance 2
100
Velocity
×
Time 2
Distance 1
/
Time 1
3.5
100
2

Figure 1: Exemplar MWP with multiple solutions.

However, these fully-supervised approaches have three drawbacks. First, current MWP datasets only provide one solution for each problem, while there naturally exist multiple solutions that give different paths of solving the same problem. For instance, the problem in Figure 1 can be solved by "$(100/2) \times (2 + 3.5)$" if we first calculate the speed and then multiply it by the total time; alternatively, we can solve it using "$100 + 100/2 \times 3.5$" by summing the distances of the first and second parts of the journey. The models trained with full supervision on current datasets are forced to fit the given solution and cannot generate diverse solutions. Second, annotating the expressions for MWPs is time-consuming. However, a large amount of MWPs with their final answers can be mined effortlessly from the internet (*e.g.*, online forums). How to efficiently utilize these partially-labeled data without the supervision of expressions remains an open problem. Third, current supervised learning approaches suffer from the train-test discrepancy. The fully-supervised learning methods optimize expression accuracy rather than answer accuracy. However, the model is evaluated by the answer accuracy on the test set, causing a natural performance gap.

To address these issues, we propose to solve the MWPs with *weak supervision*, where only the problem texts and the final answers are required. By directly optimizing the answer accuracy rather than the expression accuracy, learning with weak supervision naturally addresses the train-test discrepancy. Our model consists of a tree-structured neural model similar to Xie and Sun (2019) to generate the solution tree and a symbolic execution module to calculate the answer. However, the symbolic execution module for arithmetic expressions is non-differentiable with respect to the answer accuracy, making it infeasible to use back-propagation to compute gradients. A straightforward approach is to employ policy gradient methods like REINFORCE (Williams 1992) to train the neural model. The policy gradient methods explore the solution space and update the policy based on generated solutions that happen to hit the correct answer. Since the solution space is large and incorrect solutions are abandoned with zero reward, these methods usually converge slowly or fail to converge.

To improve the efficiency of weakly-supervised learning, we propose a novel *fixing* mechanism to learn from incorrect predictions, which is inspired by the human ability to learn from failures via abductive reasoning (Magnani 2009; Zhou 2019a). The fixing mechanism propagates the error from the root node to the leaf nodes in the solution tree and finds the most probable *fix* that can generate the desired answer. The fixed solution tree is further used as a pseudo label to train the neural model. Figure 2 shows how the fixing mechanism corrects the wrong solution tree by tracing the error in a top-down manner.

Furthermore, we design two practical techniques to traverse the solution space and discover possible solutions efficiently. First, we observe a positive correlation between the number of quantities in the text and the size of the solution tree (the number of leaf nodes in the tree), and propose a *tree regularization* technique based on this observation to limit the range of possible tree sizes and shrink the solution space. Second, we adopt a *memory buffer* to track and save the discovered fixes for each problem with the fixing mechanism. All memory buffer solutions are used as pseudo labels to train the model, encouraging the model to generate more diverse solutions for a single problem.

In summary, by combining the fixing mechanism and the above two techniques, the proposed **learning-by-fixing** (LBF) method contains an exploring stage and a learning stage in each iteration, as shown in Figure 2. We utilize the fixing mechanism and tree regularization to correct wrong answers in the exploring stage and generate fixed expressions as pseudo labels. In the learning stage, we train the neural model using these pseudo labels.

We conduct comprehensive experiments on the Math23K dataset (Wang, Liu, and Shi 2017). The proposed LBF method significantly outperforms the reinforcement learning baselines in weakly-supervised learning and achieves comparable performance with several fully-supervised methods. Furthermore, our proposed method achieves much better top-3/5 answer accuracy than fully-supervised methods, illustrating its advantage in generating diverse solutions. The ablative experiments also demonstrate the efficacy of the designed algorithms, including the fixing mechanism, tree regularization, and memory buffer.

Related Work

Math Word Problems

Solving mathematical word problems (MWPs) is a challenging task in natural language processing. Previous studies range from traditional rule-based methods (Fletcher 1985; Bakman 2007; Yu-hui et al. 2010), statistical learning methods (Kushman et al. 2014; Zhou, Dai, and Chen 2015; Mitra and Baral 2016; Roy and Roth 2017; Huang et al. 2016), semantic-parsing methods (Shi et al. 2015; Koncel-Kedziorski et al. 2015; Huang et al. 2017) to recent deep learning methods (Ling et al. 2017; Wang, Liu, and Shi 2017; Huang et al. 2018; Robaidek, Koncel-Kedziorski, and Hajishirzi 2018; Wang et al. 2018, 2019; Chiang and Chen 2019; Xie and Sun 2019; Zhang et al. 2020).

In particular, Deep Neural Solver (DNS) (Wang, Liu, and Shi 2017) is a pioneering work that designs a Seq2seq model to solve MWPs and achieves promising results. Xie and Sun (2019) propose a tree-structured neural solver to generate the solution tree in a goal-driven manner. All these neural solvers learn the model with full supervision, where the ground-truth intermediate representations (e.g., expressions, programs) are given during training. To learn the solver with less supervision, Koncel-Kedziorski et al. (2015) use a discriminative model to solve MWPs in a weakly-supervised way. They utilize separate modules to extract features, construct expression trees, and score the likelihood, which is different from the current end-to-end neural solvers. Upadhyay et al. (2016), Zhou, Dai, and Chen (2015), and Kushman et al. (2014) use mixed supervision, where one dataset has only annotated equations, and the other has only final answers. However, for the set with final answers, they also depend on pre-defined equation templates. Chen et al. (2020) apply a neural-symbolic reader on MathQA, which is a large-scale dataset with fully-specified operational programs. They have access to the ground truth programs for a small fraction of training samples at the first iterations of training.

Unlike these methods, the proposed LBF method requires only the supervision of the final answer and generates diverse solutions by keeping a memory buffer. Notably, it addresses the sparse reward problem in policy gradient methods using a fixing mechanism that propagates error down a solution tree and finds the most probable fix.

Neural-Symbolic Learning for NLP

Neural-symbolic learning has been applied to solve NLP tasks with weak supervision, such as semantic parsing and program synthesis (Liang et al. 2016a; Guu et al. 2017; Liang et al. 2018; Agarwal et al. 2019). Like for MWP, they generate intermediate symbolic representations with a neural network and execute the intermediate representation with a symbolic reasoning module to get the final result. Typical approaches for such neural-symbolic models use policy gradient methods like REINFORCE since the symbolic execution module is non-differentiable. For example, Neural Symbolic Machines (Liang et al. 2016b) combines REINFORCE with a maximum-likelihood training process to find good programs. Guu et al. (2017) augment reinforcement learning with the maximum marginal likelihood so that probability is distributed evenly across consistent programs. Memory Augmented Policy Optimization (MAPO) (Liang et al. 2018) formulates its learning objective as an expectation over a memory buffer of high-reward samples and a separate expectation outside the buffer, which helps accelerate and stabilize policy gradient training. Meta Reward Learning (Agarwal et al. 2019) uses an auxiliary reward function to provide feedback beyond a binary success or failure. Since these methods can only learn from sparse successful samples, they suffer from cold start and inefficient exploration of large search spaces. Recently, Dai and Zhou (2017), Dai et al. (2019), and Zhou (2019b) introduce abductive learning, which states that human misperceptions can be corrected via abductive reasoning. In this paper, we follow the abductive learning method (Li et al. 2020) and propose a novel fixing mechanism to learn from negative samples, significantly accelerating and stabilizing the weakly-supervised learning process. We further design the tree regularization and memory buffer techniques to shrink and explore the solution space efficiently.

Weakly-Supervised MWPs

In this section, we define the weakly-supervised math word problems and describe the goal-driven tree model originated from Xie and Sun (2019). Then we introduce the proposed learning-by-fixing method, as also shown in Figure 2.

Problem Definition

A math word problem contains an input problem text P. A machine learning model with parameters θ needs to translate P into an intermediate expression T, which is executed to compute the final answer y. In fully-supervised learning, we learn from the ground truth expression T and the final answer y. The learning objective is to maximize the data likelihood $p(T, y|P; \theta) = p_\theta(T|P)p(y|T)$, where computing y given T is a deterministic process. In contrast, in the weakly-supervised setting, only P and y are observed, while T is hidden. In other words, the model is required to generate an unknown expression from the problem text. The expression is then executed to get the final answer.

Goal-driven Tree-Structured Model

A problem text P consists of words and numeric values. The model takes in problem text P and outputs a solution tree T. Let V^{num} denote the ordered list of numeric values in P according to their order in the problem text. Generally, T may contain constants $V^{con} = \{1, 2, \pi\}$, mathematical operators $V^{op} = \{+, -, \times, \div, \wedge\}$, and numeric values V^{num} from the problem text P. Therefore, the target vocabulary of P is denoted as $\Sigma = V^{op} \cup V^{con} \cup V^{num}$ and it varies between problems due to different V^{num}.

To generate the solution tree, we adopt the goal-driven tree-structured neural model (GTS) (Xie and Sun 2019), which first encodes the problem text into its goal and then recursively decomposes it into sub-goals in a top-down manner.

Problem Encoding. Each word of the problem text is encoded into a contextual representation. Specifically, for a problem $P = w_1 w_2 ... w_n$, each word w_i is first converted to a word embedding $\mathbf{w}_i$. Then the sequence of embeddings is inputted to a bi-directional GRU (Cho et al. 2014) to produce a contextual word representation: $\mathbf{h}_i = \overrightarrow{\mathbf{h}_i} + \overleftarrow{\mathbf{h}_i}$, where $\overrightarrow{\mathbf{h}_i}, \overleftarrow{\mathbf{h}_i}$ are the hidden states of the forward and backward GRUs at position i, respectively.

Solution Tree Generation. The tree generation process is designed as a preorder tree traversal (root-left-right). The root node of the solution tree is initialized with a goal vector $\mathbf{q}_0 = \overrightarrow{\mathbf{h}_n} + \overleftarrow{\mathbf{h}_0}$.

For a node with goal $\mathbf{q}$, we first derive a context vector $\mathbf{c}$ by an attention mechanism to summarize relevant information from the problem:

$$a_i = \text{softmax}(\mathbf{v}_a^\top \tanh(\mathbf{W}_a[\mathbf{q}, \mathbf{h}_i])) \tag{1}$$

$$\mathbf{c} = \sum_i a_i \mathbf{h}_i \tag{2}$$

where $\mathbf{v}_a$ and $\mathbf{W}_a$ are trainable parameters. Then the goal $\mathbf{q}$ and the context $\mathbf{c}$ are used to predict the token of this node from the target vocabulary Σ. The probability of token t is defined as:

$$s(t|\mathbf{q}, \mathbf{c}) = \mathbf{w}_n^\top \tanh(\mathbf{W}_s[\mathbf{q}, \mathbf{c}, \mathbf{e}(t)]) \tag{3}$$

$$p(t|\mathbf{q}, \mathbf{c}) = \text{softmax}(s(t|\mathbf{q}, \mathbf{c})) \tag{4}$$

where $\mathbf{e}(t)$ is the embedding of token t:

$$\mathbf{e}(t) = \begin{cases} \mathbf{M}_{op}(t) & \text{if } t \in V^{op} \\ \mathbf{M}_{con}(t) & \text{if } t \in V^{con} \\ \mathbf{h}_{loc(t,P)} & \text{if } t \in V^{num} \end{cases} \tag{5}$$

where $\mathbf{M}_{op}$ and $\mathbf{M}_{con}$ are two trainable embeddings for operators and constants, respectively. For a number token, its embedding is the corresponding hidden state $\mathbf{h}_{loc(t,P)}$ from the encoder, where $loc(t, P)$ is the index of t in the problem P. The predicted token $\hat{t}$ is:

$$\hat{t} = \arg\max_{t \in \Sigma} p(t|\mathbf{q}, \mathbf{c}) \tag{6}$$

If the predicted token is a number token or constant, the node is terminated and its goal is realized by the predicted token; otherwise, the predicted token is an operator and the current goal is decomposed into left and right sub-goals combined by the operator. Please refer to the *supplementary material* for more details about the goal decomposition process.

Answer Calculation. The generated solution tree is transformed into a reasoning tree $\hat{T}$ by creating auxiliary non-terminal nodes in place of the operator nodes to store the intermediate results, and the original operator nodes are attached as child nodes to the corresponding auxiliary nodes. Then the final answer $\hat{y}$ is calculated by executing $\hat{T}$ to the value of the root node in a bottom-up manner.

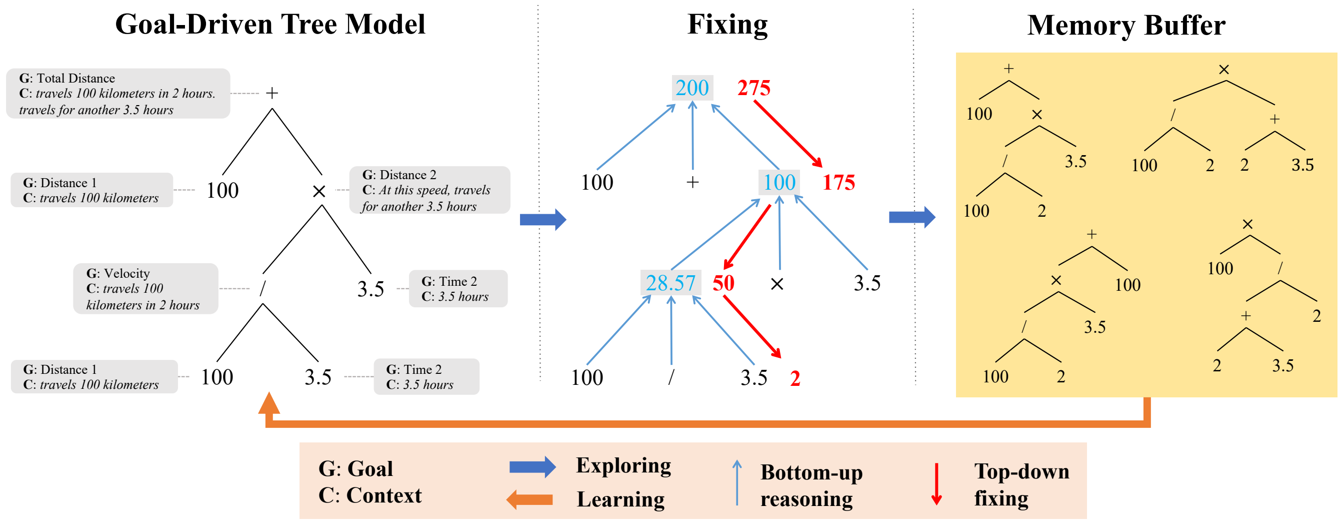


Figure 2: Overview of our proposed learning-by-fixing (LBF) method. It shows the process for learning the example in Figure 1. LBF works by iteratively exploring the solution space and learning the MWP solver. **Exploring**: the problem first goes through the GTS module and produces a tentative solution using tree regularization. Then the fixing mechanism diagnoses this solution by propagating the correct answer in a top-down manner. The fixed solution is then added to the memory buffer. **Learning**: all solutions in the memory buffer are used as pseudo labels to train the GTS module using a cross-entropy loss function.

Learning-by-Fixing

Fixing Mechanism Drawing inspiration from humans' ability to correct and learn from failures, we propose a fixing mechanism to correct the wrong solution trees via abductive reasoning following Li et al. (2020) and use the fixed solution trees as pseudo labels for training. Specifically, we find the most probable fix for the wrong prediction by back-tracking the reasoning tree and propagating the error from the root node into the leaf nodes in a top-down manner.

The key ingredient in the fixing mechanism is the 1-step fix (1-FIX) algorithm which assumes that only one symbol in the reasoning tree can be substituted. As shown by the 1-FIX function in Algorithm 1, the 1-step fix starts from the root node of the reasoning tree and gradually searches down to find a fix that makes the final output equal to the ground-truth. The search process is implemented with a priority queue, where each element is defined as a fix-tuple (A, α_A, p):

- A is the current visiting node.
- α_A is the expected value on this node, which means if the value of A is changed to α_A, $\hat{T}$ will execute to the ground-truth answer y.
- p is the visiting priority, which reflects the probability of changing the value of A.

In 1-FIX, error propagation through the solution tree is achieved by a *solve* function, which aims at computing the expected value of a child node from its parent's expected value. Supposing B is A's child node and α_A is the expected value of A, the $solve(B, A, \alpha_A)$ function works as following:

- If B is A's left or right child, we directly solve the equation $\alpha_B \bigoplus child_R(A) = \alpha_A$ or $child_L(A) \bigoplus \alpha_B = \alpha_A$ to get B's expected value α_B, where $\bigoplus$ denotes the operator.
- If B is an operator node, we try to replace B with all other operators and check whether the new expression can generate the correct answer. That is, $child_L(A)\ \alpha_B\ child_R(A) = \alpha_A$ where α_B is now an operator. If there is no α_B satisfying this equation, the solve function returns none.

Please refer to the *supplementary material* for the definition of the visiting priority as well as the illustrative example of the 1-FIX process.

To search the neighbors of $\hat{T}$ within multi-step distance, we extend the 1-step fix to multi-step by incorporating a RANDOMWALK function. As shown in Algorithm 1, if we find a fix by 1-FIX, we return this fix; otherwise, we randomly change one leaf node in the reasoning tree to another symbol within the same set (*e.g.*, operators V^{op}) based on the probability in Equation 4. This process will be repeated for certain iterations until it finds a fix for the solution.

Solution Space Exploration

Tree Regularization While Li et al. (2020) assumes the length of the intermediate representation is given, the expression length is unknown in weakly-supervised learning. Thus, the original solution space is infinite since the predicted token decides whether to continue the generation or stop. Therefore, it is critical to shrink the solution space, *i.e.*, control the size of the generated solution trees. If the size of the generated solution tree varies a lot from the target size, it would be challenging for the solution or its fix to hit the correct answer. Although the target size is unknown, we observe a positive correlation between the target size and the number of quantities in text. Regarding this observation as a tree size prior, we design a tree regularization algorithm to generate a solution tree with a target size and regularize the size in an empirical range. Denote the size of a solution tree Size(T) as the number of leaf nodes including quantities, constants, and operators. The prior range of Size(T) given the length of

Algorithm 1 Fixing Mechanism

```
1:  Input: reasoning tree $\hat{T}$, ground-truth answer $y$
2:  $T^{(0)} = \hat{T}$
3:  for $i \leftarrow 0$ to $m$ do
4:      $T^* = $ 1-FIX$(T^{(i)}, y)$
5:      if $T^* \neq \varnothing$ then
6:          return $T^*$
7:      else
8:          $T^{(i+1)} = $ RANDOMWALK$(T^{(i)})$
9:  return $\varnothing$
10:
11: function 1-FIX$(T, y)$
12: $q$ = PriorityQueue(), $S$ = the root node of $T$
13: $q.push(S, y, 1)$
14: while $(A, \alpha_A, p) = q.pop()$ do
15:     if $A \in \Sigma$ then
16:         $T^* = \hat{T}(A \rightarrow \alpha_A)$
17:         return $T^*$
18:     for $B \in child(A)$ do
19:         $\alpha_B = solve(B, A, \alpha_A)$
20:         if not ($B \in \Sigma$ and $\alpha_B \notin \Sigma$) then
21:             $q.push(B, \alpha_B, p(B \rightarrow \alpha_B))$
22: return $\varnothing$
```

the numeric value list len(V^{num}) is defined as:

$$\begin{aligned}&\text{Size}(T) \in [\text{minSize}(T), \text{maxSize}(T)]\\&\text{minSize}(T) = a_{min}\text{len}(V^{num}) + b_{min}\\&\text{maxSize}(T) = a_{max}\text{len}(V^{num}) + b_{max}\end{aligned} \tag{7}$$

where $a_{min}, b_{min}, a_{max}, b_{max}$ are the hyperparameters. The effect of these hyperparameters will be discussed in Table 2.

We further propose a *tree regularization* algorithm to decode a solution tree with a given size. To generate a tree of a given size l, we design two rules to produce a prefix-order expression during the preorder tree decoding:

1. The number of operators cannot be greater than $\lfloor l/2 \rfloor$.
2. Except the l-th position, the number of numeric values (quantities and constants) cannot be greater than the number of operators.

These two rules are inspired by the syntax of prefix notation (a.k.a, normal Polish notation) for mathematical expressions. The rules shrink the target vocabulary Σ in Equation 6 so that the tree generation can be stopped when it reaches the target size. Figure 3 shows illustrative examples of the tree regularization algorithm.

With tree regularization, we can search the possible fixes within a given range of tree size [minSize(T), maxSize(T)] for each problem.

Memory Buffer. We adopt a memory buffer to track and save the discovered fixes for each problem. The memory buffer enables us to seek multiple solutions for a single problem and use all of them as pseudo labels for training, which encourages diverse solutions. Formally, given a problem P and its buffer β, the learning objective is to minimize the negative log-likelihood of all fixed expressions in the buffer:

$$J(P, \beta) = -\sum_{T^* \in \beta} \log p(T^*|P) \tag{8}$$

$V^{num} = \{100, 2, 3.5\}$
$V^{op} = \{+, -, \times, \div, \wedge\}$
$V^{con} = \{1, 2, \pi\}$

Target size $l = 5$

×	②	V^{op}
÷	N/A	$V^{op} \cup V^{num} \cup V^{con}$
100	①	$V^{num} \cup V^{con}$
2	①	$V^{num} \cup V^{con}$
3.5	①	$V^{num} \cup V^{con}$

Prefix: × ÷ 100 2 3.5

Target size $l = 7$

×	②	V^{op}
÷	N/A	$V^{op} \cup V^{num} \cup V^{con}$
100	N/A	$V^{op} \cup V^{num} \cup V^{con}$
2	N/A	$V^{op} \cup V^{num} \cup V^{con}$
+	②	V^{op}
2	①	$V^{num} \cup V^{con}$
3.5	①	$V^{num} \cup V^{con}$

Prefix: × ÷ 100 2 + 2 3.5

Figure 3: Tree regularization for the problem in Figure 1 given different target sizes. The three columns are the generated tokens, the effective rules, and the target vocabularies shrunk by the rules, respectively.

Learning-by-Fixing Framework

The complete learning-by-fixing method is described in Algorithm 2. In the exploring state, we use the fixing mechanism and tree regularization to discover possible fixes for the wrong trees generated by the neural network, and put them into a buffer. In the learning stage, we train the model with all the solutions in the memory buffer by minimizing the loss function in Equation 8.

Algorithm 2 Learning-by-Fixing

```
1:  Input: training set $\mathcal{D} = \{(P_i, y_i)\}_{i=1}^N$
2:  memory buffer $\mathcal{B} = \{\beta_i\}_{i=1}^N$, the GTS model $\theta$
3:  for $P_i, y_i, \beta_i \in (\mathcal{D}, \mathcal{B})$ do
4:                                              ▷Exploring
5:      $\hat{T}_i$ = GTS $(P; \theta)$
6:      $T_i^* = m$-FIX$(\hat{T}_i, y_i)$
7:      if $T_i^* \neq \varnothing$ and $T_i^* \notin \beta_i$ then
8:          $\beta_i \leftarrow \beta_i \cup \{T_i^*\}$
9:                                              ▷Learning
10:     $\theta = \theta - \nabla_\theta J(P_i, \beta_i)$
```

Experimental Results

Experimental Setup

Dataset. We evaluate our proposed method on the Math23K dataset (Wang, Liu, and Shi 2017). It contains 23,161 math word problems annotated with solution expressions and answers. For the weakly-supervised setting, we only use the problems and final answers and discard the expressions. We do cross-validation following the setting of Xie and Sun (2019).

Evaluation Metric. We evaluate the model performance by answer accuracy, where the generated solution is considered correct if it executes to the ground-truth answer. Specifically, we report answer accuracies of all the top-$1/3/5$ predictions using beam search. It evaluates the model's ability to generate multiple possible solutions.

Models. We conduct experiments by comparing our methods with variants of weakly-supervised learning methods. Specifically, we experiment with two inference models: Seq2Seq with bidirectional Long Short Memory network (BiLSTM) (Wu et al. 2016) and GTS (Xie and Sun

2019), and train with four learning strategies: REINFORCE, MAPO (Liang et al. 2018), LBF, LBF-w/o-M (without memory buffer). MAPO is a state-of-the-art method in semantic parsing task that extends the REINFORCE with augmented memory. Both models are also trained with the tree regularization algorithm. We also compare with the fully-supervised learning methods to demonstrate our superiority in generating diverse solutions. In the ablative studies, we analyze the effect of the proposed tree regularization and the length of search steps in fixing mechanism.

Comparisons with state-of-the-art

Table 1 summarizes the answer accuracy of different weakly-supervised learning methods and the state-of-the-art fully-supervised approaches. The proposed learning-by-fixing framework significantly outperforms the policy gradient baselines like REINFORCE and MAPO, on both the Seq2seq and the GTS models. It demonstrates the strength of our proposed LBF method in weakly-supervised learning. The GTS-LBF-fully model is trained by initializing the memory buffer with all the ground-truth expressions. It demonstrates that by extending to the fully-supervised setting, our model maintains the top-1 accuracy while significantly improving solutions' diversity. We believe that learning MWPs with weak supervision is a promising direction. It requires fewer annotations and allows us to build larger datasets with less cost.

Model		Accuracy(%)
	Fully-Supervised	
Retrieval (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018)		47.2
Classification (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018)		57.9
LSTM (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018)		51.9
CNN (Robaidek, Koncel-Kedziorski, and Hajishirzi 2018)		42.3
DNS (Wang, Liu, and Shi 2017)		58.1
Seq2seqET (Wang et al. 2018)		66.7
Stack-Decoder (Chiang and Chen 2019)		65.8
T-RNN (Wang et al. 2019)		66.9
GTS (Xie and Sun 2019)		74.3
Graph2Tree (Zhang et al. 2020)		**74.8** [1]
GTS-LBF-fully		74.1
	Weakly-Supervised	
Seq2seq	REINFORCE	1.2
	MAPO	10.7
	LBF-w/o-M	44.7
	LBF	43.6
GTS	REINFORCE	15.8
	MAPO	20.8
	LBF-w/o-M	58.3
	LBF	**59.4**

Table 1: Answer accuracy on the Math23K dataset. We compare variants of models with our LBF method.

Convergence speed

Figure 4 shows the learning curves of different weakly-supervised learning methods for the GTS model. The proposed LBF method converges much faster and achieves higher accuracy compared with other methods. Both the REINFORCE and MAPO take a long time to start improving, which indicates the policy gradient methods suffer from the cold-start problem and need time to accumulate rewarding samples.

[1]We run the code using the same setting as GTS for three times and compute the average accuracy.

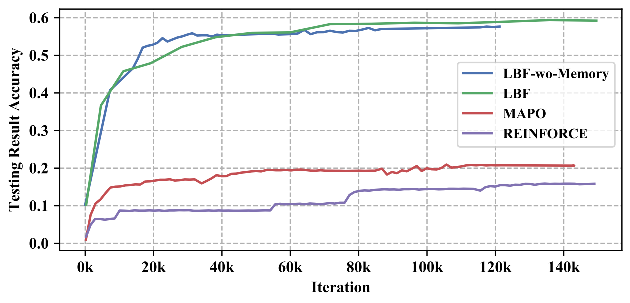


Figure 4: The learning curves of the GTS model using different weakly-supervised learning methods.

Diverse solutions with memory buffer

To evaluate the ability to generate diverse solutions, we report the answer accuracies of all the top-1/3/5 solutions on the test set using beam search, denoted as Acc@1/3/5, as shown in Table 2. In the weakly-supervised scenario, GTS-LBF achieves slightly better Acc@1 accuracy and much better Acc@3/5 accuracy than GTS-LBF-w/o-M. In the fully supervised scenario, GTS-LBF-fully achieves comparable top-1 accuracy and much better Acc@3/5 accuracy than the original GTS model. Particularly, GTS-LBF-fully outperforms GTS by 21% and 26% in terms of Acc@3/5 accuracy. It reveals the efficacy of the memory buffer in encouraging diverse solutions in both weakly-supervised learning and fully-supervised learning.

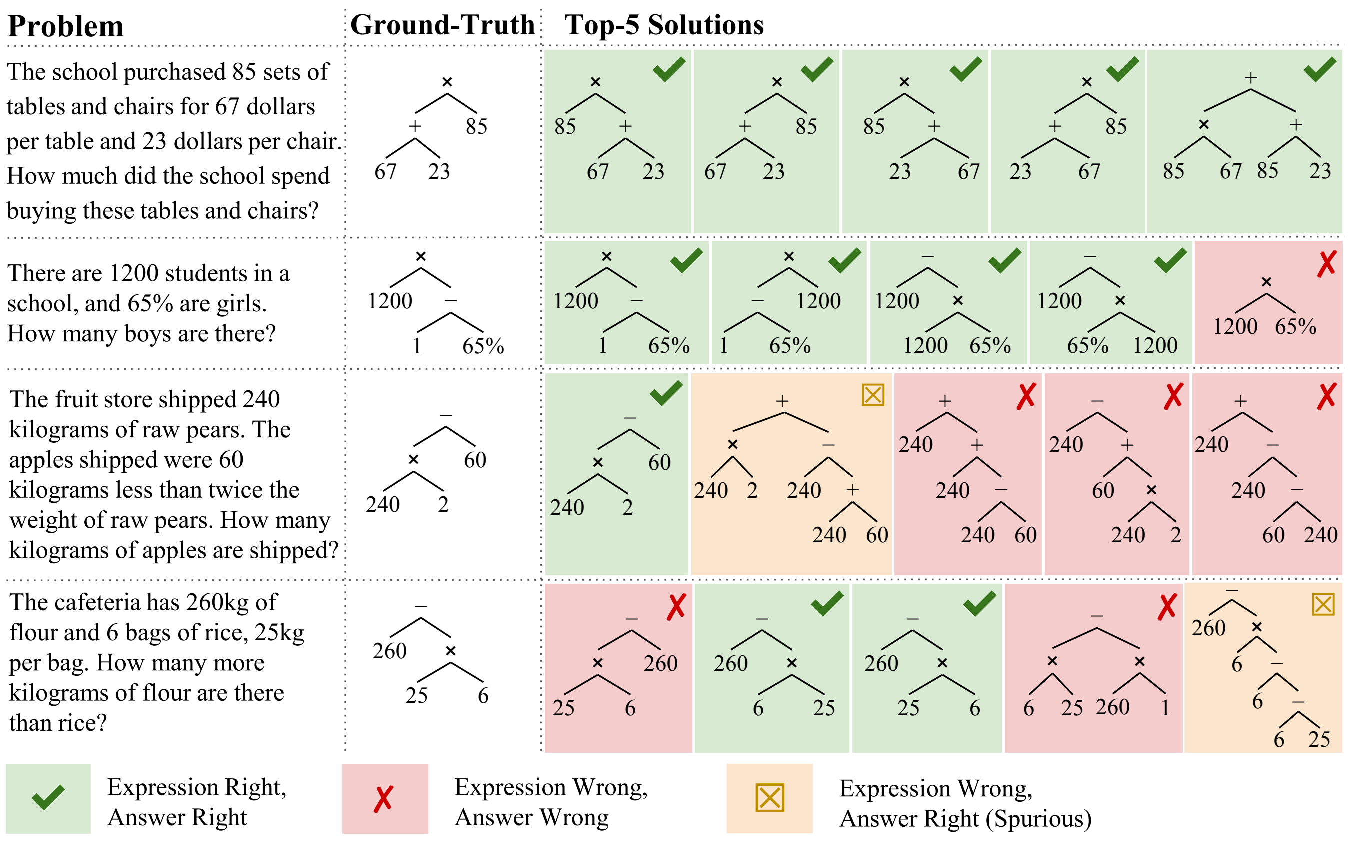
Model	Tree Size	Acc@1	Acc@3	Acc@5
	Fully Supervised			
GTS		**74.3**	42.2	30.0
GTS-LBF-fully		74.1	**63.4**	**56.3**
	Weakly Supervised			
GTS-LBF-w/o-M	[1,+∞)	∼0	∼0	∼0
	[2n-1,2n+1]	55.3	26.2	19.3
	[2n-1,2n+3]	58.3	27.7	20.3
	[2n-3,2n+5]	56.7	27.7	20.6
GTS-LBF	[1,+∞)	∼0	∼0	∼0
	[2n-1,2n+1]	56.7	45.3	39.1
	[2n-1,2n+3]	**59.4**	**49.6**	**45.2**
	[2n-3,2n+5]	57.6	49.3	45.2

Table 2: Answer accuracies of all the top-1/3/5 solutions decoded using beam search, denoted as Acc@1/3/5.

Qualitative analysis

We visualize several examples of the top-5 predictions of GTS-LBF in Figure 5. In the first example, the first solution generated by our model is to sum up the prices of a table and a chair first, and then multiply it by the number of pairs of tables and chairs. Our model can also produce another reasonable solution (the fifth column) by deriving the prices of tables and chairs separately and then summing them up.

One caveat for the multiple solutions is that some solutions have different solution trees but are equivalent by switching the order of numeric values or subtrees, as shown in the first four solutions of the first problem in Figure 5. In particular, multiplication and addition are commutative, and our


Problem
Ground-Truth
Top-5 Solutions
Expression Right, Answer Right
Expression Wrong, Answer Wrong
Expression Wrong, Answer Right (Spurious)

The school purchased 85 sets of tables and chairs for 67 dollars per table and 23 dollars per chair. How much did the school spend buying these tables and chairs?

There are 1200 students in a school, and 65% are girls. How many boys are there?

The fruit store shipped 240 kilograms of raw pears. The apples shipped were 60 kilograms less than twice the weight of raw pears. How many kilograms of apples are shipped?

The cafeteria has 260kg of flour and 6 bags of rice, 25kg per bag. How many more kilograms of flour are there than rice?

Figure 5: Qualitative results on the Math23K dataset. We visualize the solution trees generated by our method.

	Right	Wrong	Spurious
Acc@1	58.6	40.6	0.56
Acc@3	49.3	50.4	0.27
Acc@5	44.9	54.8	0.32

Table 3: Human evaluation on the generated solutions (%).

model learns and exploits this property to generate equivalent solutions with different tree structures.

The first solution to the fourth problem in Figure 5 is a typical error case of our model, which comes from the wrong prediction of the problem goal. Another type of failure in our model is the spurious solutions, which are correct but not meaningful answers, such as the second solution of the third problem in Figure 5. To test how frequent the spurious solutions appear, we randomly select five hundred examples from the test set, and ask three human annotators to determine whether each generated expression is right, wrong, or spurious. Table 3 provides the human evaluation results, and it shows that spurious solutions are rare in our model.

Ablative Analyses

Tree regularization. We test different choices of the hyperparameters defined by Equation 7 in tree regularization. As shown in Table 2, the model without tree regularization, *i.e.*, tree size $\in [1, +\infty)$, fails to converge and gets nearly 0 accuracy. The best range for the solution tree size is $[2n-1, 2n+3]$, where $n = \text{len}(V^{num})$. We provide an intuitive interpretation of this range: for a problem with n quantities, (1) $n-1$ operators are needed to connect n quantities, which leads to the lower bound of tree size to $2n-1$; (2) in certain cases, the constants or quantities are used more than once, leading to a rough upper bound of $2n+3$. Therefore, we use $[2n-1, 2n+3]$ as the default range in our implementations. Empirically, this range covers 88% of the lengths of the given ground-truth expressions in the Math23K dataset, providing an efficient prior for tree size.

Number of Search Steps Table 4 shows the comparison of various step lengths in the m-FIX algorithm. In most cases, increasing the step length improves the chances of correcting wrong solutions, thus improving the performance. We use 50 as the default step length in our method.

Models \ **Steps**	**1**	**10**	**50**	**100**
Seq2seq-LBF-w/o-M	41.9	43.4	44.7	**47.8**
Seq2seq-LBF	43.9	**45.7**	43.6	44.6
GTS-LBF-w/o-M	51.2	54.6	**58.3**	57.8
GTS-LBF	52.5	55.8	59.4	**59.6**

Table 4: The influence of search steps on accuracy (%).

Conclusion

In this work, we propose a weakly-supervised paradigm for learning MWPs and a novel learning-by-fixing framework to boost the learning. Our method endows the MWP learner with the capability of learning from wrong solutions, thus significantly improving the answer accuracy and learning efficiency. One future direction of the proposed model is to prevent generating equivalent or spurious solutions during training, possibly by making the generated solution trees more interpretable with semantic constraints.

Ethical Impact

The presented work should be categorized as research in the field of weakly-supervised learning and abductive reasoning. It can help teachers in school get various solutions of a math word problem. This work may also inspire new algorithmic, theoretical, and experimental investigation in neural-symbolic methods and NLP tasks.

References

Agarwal, R.; Liang, C.; Schuurmans, D.; and Norouzi, M. 2019. Learning to Generalize from Sparse and Underspecified Rewards. In *ICML*.

Bakman, Y. 2007. Robust Understanding of Word Problems with Extraneous Information.

Chen, X.; Liang, C.; Yu, A. W.; Zhou, D.; Song, D.; and Le, Q. V. 2020. Neural Symbolic Reader: Scalable Integration of Distributed and Symbolic Representations for Reading Comprehension. In *ICLR*.

Chiang, T.-R.; and Chen, Y.-N. 2019. Semantically-Aligned Equation Generation for Solving and Reasoning Math Word Problems. *ArXiv* abs/1811.00720.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP* .

Dai, W.-Z.; Xu, Q.; Yu, Y.; and Zhou, Z.-H. 2019. Bridging Machine Learning and Logical Reasoning by Abductive Learning. In *Advances in Neural Information Processing Systems*, 2811–2822.

Dai, W.-Z.; and Zhou, Z.-H. 2017. Combining logical abduction and statistical induction: Discovering written primitives with human knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Fletcher, C. R. 1985. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers* 17: 565–571.

Guu, K.; Pasupat, P.; Liu, E. Z.; and Liang, P. 2017. From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood. In *ACL*.

Huang, D.; Liu, J.; Lin, C.-Y.; and Yin, J. 2018. Neural Math Word Problem Solver with Reinforcement Learning. In *COLING*.

Huang, D.; Shi, S.; Lin, C.-Y.; and Yin, J. 2017. Learning Fine-Grained Expressions to Solve Math Word Problems. In *EMNLP*.

Huang, D.; Shi, S.; Lin, C.-Y.; Yin, J.; and Ma, W.-Y. 2016. How well do Computers Solve Math Word Problems? Large-Scale Dataset Construction and Evaluation. In *ACL*.

Koncel-Kedziorski, R.; Hajishirzi, H.; Sabharwal, A.; Etzioni, O.; and Ang, S. D. 2015. Parsing Algebraic Word Problems into Equations. *Transactions of the Association for Computational Linguistics* 3: 585–597.

Kushman, N.; Zettlemoyer, L.; Barzilay, R.; and Artzi, Y. 2014. Learning to Automatically Solve Algebra Word Problems. In *ACL*.

Li, Q.; Huang, S.; Hong, Y.; Chen, Y.; Wu, Y. N.; and Zhu, S.-C. 2020. Closed Loop Neural-Symbolic Learning via Integrating Neural Perception, Grammar Parsing, and Symbolic Reasoning. In *International Conference on Machine Learning (ICML)*.

Liang, C.; Berant, J.; Le, Q.; Forbus, K. D.; and Lao, N. 2016a. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *arXiv preprint arXiv:1611.00020* .

Liang, C.; Berant, J.; Le, Q. V.; Forbus, K. D.; and Lao, N. 2016b. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *ACL*.

Liang, C.; Norouzi, M.; Berant, J.; Le, Q. V.; and Lao, N. 2018. Memory Augmented Policy Optimization for Program Synthesis and Semantic Parsing. In *NeurIPS*.

Ling, W.; Yogatama, D.; Dyer, C.; and Blunsom, P. 2017. Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems. *ArXiv* abs/1705.04146.

Magnani, L. 2009. *Abductive cognition: The epistemological and eco-cognitive dimensions of hypothetical reasoning*, volume 3. Springer Science & Business Media.

Mitra, A.; and Baral, C. 2016. Learning To Use Formulas To Solve Simple Arithmetic Problems. In *ACL*.

Robaidek, B.; Koncel-Kedziorski, R.; and Hajishirzi, H. 2018. Data-Driven Methods for Solving Algebra Word Problems. *ArXiv* abs/1804.10718.

Roy, S.; and Roth, D. 2017. Unit Dependency Graph and Its Application to Arithmetic Word Problem Solving. In *AAAI*.

Shi, S.; Wang, Y.; Lin, C.-Y.; Liu, X.; and Rui, Y. 2015. Automatically Solving Number Word Problems by Semantic Parsing and Reasoning. In *EMNLP*.

Upadhyay, S.; Chang, M.-W.; Chang, K.-W.; and tau Yih, W. 2016. Learning from Explicit and Implicit Supervision Jointly For Algebra Word Problems. In *EMNLP*.

Wang, L.; Wang, Y.; Cai, D.; Zhang, D.; and Liu, X. 2018. Translating Math Word Problem to Expression Tree. In *EMNLP*.

Wang, L.; Zhang, D.; Zhang, J.; Xu, X.; Gao, L.; Dai, B. T.; and Shen, H. T. 2019. Template-Based Math Word Problem Solvers with Recursive Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence* 33(01): 7144–7151.

Wang, Y.; Liu, X.; and Shi, S. 2017. Deep Neural Solver for Math Word Problems. 845–854. Copenhagen, Denmark: Association for Computational Linguistics.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4): 229–256.

Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; Klingner, J.; Shah, A.; Johnson, M.; Liu, X.; Kaiser, L.; Gouws, S.; Kato, Y.; Kudo, T.; Kazawa, H.; Stevens, K.; Kurian, G.; Patil, N.; Wang, W.; Young, C.; Smith, J.; Riesa, J.; Rudnick, A.; Vinyals, O.; Corrado, G. S.; Hughes, M.; and Dean, J. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv* abs/1609.08144.

Xie, Z.; and Sun, S. 2019. A Goal-Driven Tree-Structured Neural Model for Math Word Problems. In *IJCAI*.

Yu-hui, M.; Ying, Z.; Guang-zuo, C.; Yun, R.; and Rong-huai, H. 2010. Frame-Based Calculus of Solving Arithmetic Multi-Step Addition and Subtraction Word Problems. *2010 Second International Workshop on Education Technology and Computer Science* 2: 476–479.

Zhang, J.; Wang, L.; Lee, R. K.-W.; Bin, Y.; Shao, J.; and Lim, E.-P. 2020. Graph-to-Tree Learning for Solving Math Word Problems. *ACL 2020* .

Zhou, L.; Dai, S.; and Chen, L. 2015. Learn to Solve Algebra Word Problems Using Quadratic Programming. In *EMNLP*.

Zhou, Z.-H. 2019a. Abductive learning: towards bridging machine learning and logical reasoning. *Science China Information Sciences* 62: 1–3.

Zhou, Z.-H. 2019b. Abductive learning: towards bridging machine learning and logical reasoning. *Science China Information Sciences* 62: 1–3.