Weakly Supervised Formula Learner for Solving Mathematical Problems

Yuxuan Wu and **Hideki Nakayama**
The University of Tokyo
{wuyuxuan,nakayama}@nlab.ci.i.u-tokyo.ac.jp

Abstract

Mathematical reasoning task is a subset of the natural language question answering task. Existing work suggested solving this task with a two-phase approach, where the model first predicts formulas from questions and then calculates answers from such formulas. This approach achieved desirable performance in existing work. However, its reliance on annotated formulas as intermediate labels throughout its training limited its application. In this work, we put forward the idea to enable models to learn optimal formulas autonomously. We proposed Weakly Supervised Formula Learner, a learning framework that drives the formula exploration with weak supervision from the final answers to mathematical problems. Our experiments are conducted on two representative mathematical reasoning datasets MathQA and Math23K. On MathQA, our method outperformed baselines trained on complete yet imperfect formula annotations. On Math23K, our method outperformed other weakly supervised learning methods. [1]

1 Introduction

Mathematical reasoning is a task where mathematical problems are described in natural language or mathematical symbols. Such problems require values, expressions, or other mathematical representations as answers. A naive approach to solving this task is to treat it as an end-to-end token-by-token predicting problem from questions to answers. However, this approach showed a relatively poor generalization capacity on unseen numbers (Saxton et al., 2019). Another approach to solving mathematical reasoning task is to adopt a two-phase methodology. In the first phase, specific formulas are predicted for each question. In the second phase, such formulas are calculated under predefined rules to produce the final answers. This approach has been widely applied in recent work and has achieved desirable results in many representative mathematical reasoning datasets (Wang et al., 2017; Amini et al., 2019). However, this two-phase solution leads to a reliance on annotated formulas as indispensable labels for training the formula predictor in the first phase. This reliance further results in two major weaknesses. Firstly, ground-truth formula annotations are not necessarily prepared for every mathematical problem and dataset. This makes it impossible to extend this solution to datasets without these annotations. Secondly, the learning process can be misled when there is noise in the formula annotations. In consideration of this, we are motivated to propose a new learning framework for solving mathematical problems that is not dependent on formula annotations. On the whole, we followed the principle of the two-phase methodology and implemented the two phases with what we call `PolicyNet` and `ActTaker`. Figure 1 is an overview of our learning framework. Our main contributions in this work can be summarized as follows:

- We established a new mechanism to learn formulas with weak supervision from final answers, which outperforms existing weakly supervised learning methods.
- We enabled models to explore reasonable formulas autonomously through a heuristic search in the space of possible formulas.
- We verified that the formulas learned with weak supervision can be more beneficial to the question answering than complete yet imperfect formula annotations.

2 Related Work

2.1 Mathematical Reasoning

In recent years, various datasets have been published to study the capacity of machine learning

[1]The software is available at https://github.com/evan-ak/wsfl.

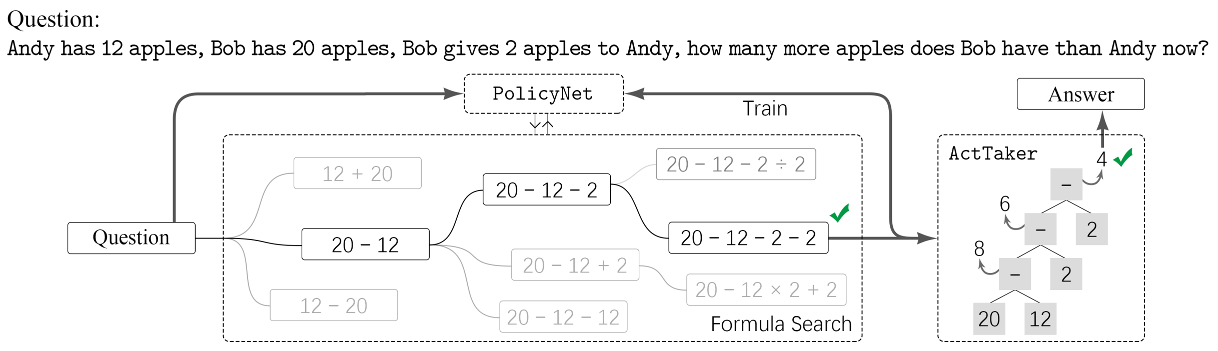

Figure 1: To get rid of the dependence on annotated formulas, our learning framework conducts a search process to explore optimal formulas. Such formulas are then fed back to the formula predictor for its training.

models in solving mathematical problems and performing quantitative reasoning. Math23K (Wang et al., 2017) is a dataset crawled from a couple of online education websites consisting of 23,162 problems with formula annotations. MathQA (Amini et al., 2019) is a dataset collected from another former dataset named AQuA (Ling et al., 2017) consisting of 37,200 problems with formula annotations. Mathematics (Saxton et al., 2019) is a complex large-scale dataset in which questions are generated in a broad range of areas including algebra, arithmetic, and calculus.

2.2 Mathematical Problem Solvers

End-to-end and formula-based methods are the two common methodologies for solving mathematical problems. The essential difference between them is that they either produce final answers directly or adopt formulas as intermediate labels.

Concretely, end-to-end methods simply regard both questions and answers as sequences of alphabets, digits, and symbols, and conduct a sequence-to-sequence prediction (Saxton et al., 2019). The application of these methods is not restricted by the absence of formulas. However, the lack of the concept of complete numbers forces them to receive and predict rational numbers digit by digit, which leads to a relatively weak generalization capacity.

On the other hand, formula-based methods employ what are called formulas or equations as intermediate labels for solving mathematical problems. Most elementary applied numerical mathematical problems can be solved by building equations with unknowns and solving the equations to acquire the answers. This generated the idea of first letting the model predict such equations and then solving the equations in a rule-based manner. This idea was first implemented by Wang et al. (2017) and then improved in later work (Wang et al., 2018; Xie and Sun, 2019; Zhang et al., 2020; Chen et al., 2020a). Faced with the problem of the reliance on formula annotations, Hong et al. (2021) proposed a fixing mechanism to learn formulas through error propagation and formula correction.

2.3 Semantic Parsing

Semantic parsing is the task of translating natural language utterances into machine-understandable logical form (Kamath and Das, 2019). Recent studies on solving mathematical problems have also benefited from semantic parsing by automatically synthesizing formulas from questions (Koncel-Kedziorski et al., 2015; Shi et al., 2015; Hopkins et al., 2017). However, considering that there is no guarantee that semantic parsing necessarily provide valid formulas for every question, the invalid formulas become noise if they are fed to following learning processes without correction. As a result, weakly supervised formula learning remains meaningful and valuable as long as perfect formula annotations are not prepared. In view of this, semantic parsing is considered an approach that works in parallel with weakly supervised formula learning methods for solving mathematical problems.

2.4 Neural Module Networks

Neural Module Networks (NMNs) are another relevant existing approach with a similar two-phase methodology (Andreas et al., 2016b). For solving visual question answering tasks, NMNs first predict programs from the questions and then compute with modules to acquire the final answers. Later work also succeeded in applying NMNs to solve discrete reasoning problems (Yi et al., 2018). Faced with the similar difficulty in training a two-phase model, existing work either utilized rein-

Question : `Andy has 12 apples, Bob has 20 apples, Bob gives 2 apples to Andy, how many more apples does Bob have than Andy now?`
Equation : `20 - 12 - 2 - 2`

q : `Andy has <N0> apples, Bob has <N1> apples, Bob gives <N2> apples to Andy, how many more apples does Bob have than Andy now?`
num : `(12, 20, 2)`
a : `4`
f : `(-, -, -, N_0, <End>, N_1, <End>, N_2, <End>, N_2, <End>)`

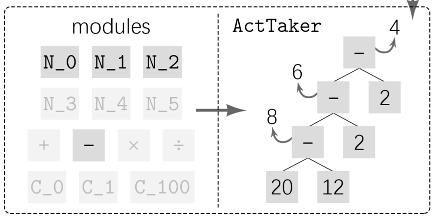

Figure 2: The preprocessing and the symbols applied in our work.

forcement learning (Andreas et al., 2016a; Johnson et al., 2017) or developed specific heuristic learning algorithms (Wu and Nakayama, 2020).

3 Weakly Supervised Formula Learner

3.1 Problem Definition

To achieve a formal representation of the mathematical problems in given datasets, we first performed some preprocessing on the question–answer pair of each problem. As shown in Figure 2, given the raw text of a question, the numbers that appear in the text are extracted as num. On the other hand, the numbers in the original text are replaced with special tokens $\langle$`Nx`$\rangle$, where `x` is the index of each number. We refer to the questions with these replaced tokens as "templates", which can also be simply denoted by q. Note that multiple questions can share the same template if they only differ in numbers. In this case, they are combined into a single template q. We let $\{num\}$ denote the set of numbers num extracted from the questions corresponding to the same template, and $\{a\}$ denote the set of answers a to these questions. Note that num in $\{num\}$ and a in $\{a\}$ should be kept paired. After preprocessing, the data visible to the following procedures should be triplets of $(q, \{num\}, \{a\})$.

Although the ground-truth equations for solving corresponding problems are annotated in some datasets and considered part of the training data in supervised learning methods, they are not visible to our learning framework, which performs weakly supervised learning. Instead, we let f denote the formula we used. Each formula is made up of a sequence of tokens where each of the tokens is the name of a module or an $\langle$`End`$\rangle$ sign. This sequence provides a preorder traversal of the tree of desired modules where $\langle$`End`$\rangle$ marks the leaf nodes. Details of the modules are presented in Section 3.5.

3.2 General Learning Process

Generally, our proposed learning framework is made up of `PolicyNet` and `ActTaker`. As shown by Equations 1 and 2, `PolicyNet` takes the question template q as the input and predicts the formula f. `ActTaker` takes the formula f and the set of numbers $\{num\}$ as inputs and calculates a set of answers $\{\hat{a}\}$ corresponding to each num.

$$f = \texttt{PolicyNet}(q) \tag{1}$$

$$\{\hat{a}\} = \texttt{ActTaker}(f,\ \{num\}) \tag{2}$$

With these two models, Algorithm 1 shows the general learning process of our Weakly Supervised Formula Learner. Here, $\mathbb{D}$ denotes the original dataset holding triplets of training data $(q, \{num\}, \{a\})$. $\mathbb{L}$ denotes a dictionary initialized to be empty for storing the optimal formulas found through the learning process. After `PolicyNet`, `ActTaker`, and $\mathbb{L}$ get initialized, the learning process is composed of numerous basic loops. Within each loop, at first, a triplet of training data is sampled from the dataset $\mathbb{D}$. Then, a search process is conducted to try to find the optimal formula for solving the given question template q. The behavior of this `Search` function is presented in detail in Section 3.3. After the optimal formula f and its accuracy $accu_f$ on q are obtained, they are used to update the dictionary $\mathbb{L}$. Concretely, if f is not `None`, and then if no f has been recorded for q or $accu_f$ exceeded the previously recorded accuracy, f and $accu_f$ will be recorded for q in $\mathbb{L}$. Finally, `PolicyNet` is

Algorithm 1 General Learning Process

```
1: PolicyNet, ActTaker ← Init()
2: $\mathbb{L}$ ← {}
3: for $loop$ in range($max\_loop$) do
4:     $(q, \{num\}, \{a\})$ ← Sample($\mathbb{D}$)
5:     $f$, $accu_f$ ← Search(PolicyNet, ActTaker, $q$, $\{num\}$, $\{a\}$)
6:     $\mathbb{L}$.update($q$: ($f$, $accu_f$))
7:     PolicyNet.train($\mathbb{L}$)
8: end for
```

trained with the q and f sampled from $\mathbb{L}$. These procedures are repeated until max_loop is reached. By the time of inference, this process is no longer required, and the answer can be acquired directly through Equations 1 and 2.

3.3 Formula Search

As shown by line 5 of Algorithm 1, we conduct a `Search` procedure in each loop to explore optimal formulas. For this procedure, we basically followed the Graph-based Heuristic Search algorithm proposed by Wu and Nakayama (2020) and adapted it to meet the needs of solving mathematical problems. The process of this procedure is presented by Algorithm 2.

Graph In this algorithm, $\mathcal{G}$ denotes a graph used to store the formulas under exploration with each of its nodes representing a unique formula. This graph is maintained under the following two rules:

- Each of its nodes n_f represents a unique formula f.
- There is an edge between two nodes if and only if the edit distance between the formulas they represent is one.

Moreover, each node n_f maintains a score $n_f.score$. Intuitively, this score indicates how desirable a formula is for solving the current given question. Wu and Nakayama (2020) suggested binding this score to the question answering accuracy. However, in mathematical problems, unreasonable formulas are highly likely to lead to scattered answers and thus result in zero accuracies. These scores can no longer provide enough guidance in a heuristic search if most of them degenerate to zero. In view of this, we modified this score to the average of two factors. Among them, the one is the standardized likelihood of the formula given by `PolicyNet`, and the other is an index related to the actual question answering effectiveness of the formula. In this section, we employ a concise implementation for the latter factor, which is simply the question answering accuracy. We will present another delicately designed way to scale this factor in Section 3.4. In general, $n_f.score$ is defined by Equations 3 to 6.

$$n_f.score = \frac{1}{2}[\, p(f|q,\theta_P) + f.accu\,] \quad (3)$$

$$p(f|q,\theta_P) = \frac{1}{L}\sum_{i=1}^{L} p(f_i|f_{1:i-1},q,\theta_P) \quad (4)$$

$$f.accu = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{|\{\hat{a}\}_i - \{a\}_i| < 10^{-3}} \quad (5)$$

$$\{\hat{a}\} = \texttt{ActTaker}(f,\ \{num\}) \quad (6)$$

Here, θ_P denotes the parameters of `PolicyNet`. L denotes the length of the current formula. $p(f_i|f_{1:i-1},q,\theta_P)$ denotes the likelihood of producing the i^{th} token in the formula given `PolicyNet`, the question template q, and the previous tokens $f_{1:i-1}$. N denotes the length of sets $\{num\}$ and $\{a\}$ corresponding to template q. $\mathbf{1}_{|\{\hat{a}\}_i - \{a\}_i| < 10^{-3}}$ returns one if the difference between the i^{th} result in $\{\hat{a}\}$ and the i^{th} answer in $\{a\}$ is less than an acceptable floating-point error bound 10^{-3}.

Graph Initialization As for the initialization of the graph as shown in Line 2 of Algorithm 2, $\mathcal{G}$ is initialized with at most three nodes. The formulas of these nodes are as follows:

- `(N_0,`$\langle$`End`$\rangle$`)`, which is the shortest legal formula;
- the formula predicted by `PolicyNet` given q with maximum likelihood;
- the formula recorded for q_c, where q_c is the previously solved template recorded in $\mathbb{L}$ that the current q is semantically closest to.

For the last clause, we determine the semantic distance between two templates by calculating the Euclidean distance between their sentence vectors embedded by the encoder of `PolicyNet`. With $E_P(\cdot)$ denoting the encoder of `PolicyNet`, the decision of q_c can be expressed by Equation 7.

$$q_c = \arg\min_{q^* \in \mathbb{L}} \|E_P(q^*) - E_P(q)\|_2 \quad (7)$$

Algorithm 2 Formula Search

```
1: func Search(PolicyNet, ActTaker, $q$, $\{num\}$, $\{a\}$)
2:     $\mathcal{G} \leftarrow$ Init()
3:     for $iter$ in range($max\_iter$) do
4:         $f_{exp} \leftarrow$ Sample($\mathcal{G}$)
5:         $f_{exp}.accu \leftarrow$ Accuracy(ActTaker ($f_{exp}$, $\{num\}$), $\{a\}$)
6:         $\mathcal{G}$.update(Mutate($f_{exp}$))
7:     end for
8:     $f_{best} \leftarrow \arg\max_{f\in\mathcal{G}} f.accu$
9:     return $f_{best}$, $f_{best}.accu$
```

Formula Sampling As shown in Line 4 of Algorithm 2, in every iteration of the search, we first select a formula f_{exp} from $\mathcal{G}$ as the formula to explore. For the `Sample` function, we also followed the mechanism proposed by Wu and Nakayama (2020). Concretely, there is an $Expectation$ value defined on every node n of $\mathcal{G}$ as presented by Equation 8. In every iteration of the search, the node with the greatest $Expectation$ value among unexplored nodes is selected as the node to be explored.

$$n.Exp = \sum_{d=0}^{3} w_d * max\{n^*.score \mid n^* \in \mathcal{G}, distance(n^*, n) \leqslant d\} \quad (8)$$

$$w = [0.5,\ 0.25,\ 0.15,\ 0.1] \quad (9)$$

Formula Examination As shown in Line 5 of Algorithm 2, the selected formula f_{exp} is examined by `ActTaker` to obtain its accuracy given $\{num\}$ and $\{a\}$. The calculation of the accuracy basically follows Equations 5 and 6. In addition, the examination by `ActTaker` may not necessarily succeed because f_{exp} is not guaranteed to be semantically legal and illegal calculations like division by zero may be encountered. In such cases, corresponding $\{\hat{a}\}_i$ is considered invalid and $|\{\hat{a}\}_i - \{a\}_i|$ is considered infinite.

Formula Mutation As shown in Line 6 of Algorithm 2, mutations are generated from f_{exp} to expand the graph $\mathcal{G}$. Here, insertion, deletion, and substitution are the three operations for generating mutations. Respectively, they insert new modules into a formula, delete existing modules from a formula, and substitute existing modules with other modules in a formula. The newly generated formulas are then added to $\mathcal{G}$ if they do not yet exist in $\mathcal{G}$. The relevant edges should also be added to $\mathcal{G}$ to keep $\mathcal{G}$ conforming to its definition and features.

When all the search iterations are finished, the formula that achieved the highest accuracy is returned together with its accuracy as the result of this formula search. If none of the formulas achieved non-zero accuracy, this function returns `None`.

3.4 Difference-Based Formula Scoring

In Section 3.3, we have presented an elementary practice for determining $n_f.score$, the score of each formula, with Equations 3 to 6. In this section, we further discuss Difference-Based Scoring (DBS), another advanced approach to determining this score based on the difference between answers acquired from formulas and ground-truth answers. This difference-based scoring technique has also been suggested and verified in recent work represented by Petersen et al. (2021).

$$n_f.score = p(f|q, \theta_P) + \beta * f.diffscore \quad (10)$$

$$f.diffscore = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{1 + 10\ln\left(|\{\hat{a}\}_i - \{a\}_i| + 1\right)} \quad (11)$$

With DBS, $n_f.score$ is calculated through Equations 10 and 11. In Equation 10, β is a hyperparameter that scales the contribution of $f.diffscore$ in $n_f.score$ with default value 1.0. In Equation 11, for each specific i, if $\{\hat{a}\}_i$ is equal to $\{a\}_i$, the corresponding term returns one, the same value as Equation 5. Otherwise, it returns a score negatively correlated with the difference between $\{\hat{a}\}_i$ and $\{a\}_i$. The 10 in the denominator is an empirical coefficient that ensures the following logarithmic difference would not contribute too much to the whole score. With this score, formula search tends to explore formula nodes surrounding the nodes that lead to answers close to ground-truth answers.

3.5 Modules

Modules are the basic calculating units for solving each mathematical problem. During computation, the modules in the tree specified by a formula are calculated recursively from the leaves to the root to acquire the final answer. The three types of modules we adopted are `Number`, `Operation`, and `Constant`.

Number The `Number` modules, which are denoted by `N_x`, are employed to establish references to the numbers extracted from questions. These modules need no input and return a number. Here, `x` is the index of the number that is referred to. This index starts from 0. For example, `N_1` returns the second number in num. An error is raised if `x` exceeds the number of numbers in num.

Operation The `Operation` modules, which are denoted simply by their symbols, are employed to conduct specific mathematical calculations. These modules need a specific number of numeric inputs (commonly two) and return the calculation result as a number. For example, the + module takes two rational numbers `a` and `b` as inputs and returns the rational number `(a+b)`. An error is raised if the calculation is illegal such as division by zero.

	Accuracy
Seq2prog (Amini et al., 2019)	51.9%
Seq2prog+cat (Amini et al., 2019)	54.2%
LSTM2TP (Chen et al., 2020a)	54.6%
TP-N2F (Chen et al., 2020a)	55.9%
Ours w/o DBS	59.5%
Ours w/ DBS	**60.1%**

Table 1: The option selecting accuracy achieved by our learning framework and baselines on MathQA.

Constant The `Constant` modules, which are denoted by `C_x`, are employed to generate constant numbers. These modules need no input and return a number. Here, `x` is the specific rational number that is referred to. For example, `C_100` returns the integer 100.

4 Experiments

Our experiments in this work are conducted on two representative mathematical reasoning datasets, MathQA (Amini et al., 2019) and Math23K (Wang et al., 2017). We report and discuss our findings on them in Section 4.1 and 4.2, respectively.

4.1 MathQA

Experimental setup To prepare the training data, we followed the preprocessing procedure presented in Section 3.1 to transform the original questions and answers into triplets of $(q, \{num\}, \{a\})$. For `PolicyNet`, the encoder is a two-layer Bidirectional LSTM (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997) with hidden state size 256. The decoder is a two-layer LSTM with hidden state size 512. The input embedding size of both of them is 300. For their training, we adopted the Adam optimizer (Kingma and Ba, 2015) with learning rate 0.001. As indicated by line 7 of Algorithm 1, `PolicyNet` is trained continuously in every learning loop. Here, the batch size for sampling training data from $\mathbb{L}$ is 64. Within each loop, `PolicyNet` is trained on 500 batches. For the comparison with baselines trained in a fully-supervised manner, we utilized part of the formula annotations to pretrain `PolicyNet`. The pretraining data, which is organized as tuples of (q, f), is filtered from the training set to meet the following two requirements. First, the formula should be able to solve the question and achieve non-zero accuracy. Second, the formula should be made up of only the four fundamental arithmetic calculations.

	Accuracy
Ours / raw annotation	52.4%
Ours / REINFORCE	56.5%
Ours	60.1%

Table 2: The performance of our models trained with raw formula annotations and formula exploration.

For `ActTaker`, we adopted twenty `Number` modules, `N_0` to `N_19`, four `Operation` modules, +, −, ×, and ÷, and four `Constant` modules, `C_1`, `C_2`, `C_3`, and `C_100`.

Evaluation metric For evaluation, MathQA provides five options for each question where the correct option is annotated. To select an option, we calculate the differences between the answer acquired by our models and each option, and select the option for which the difference is minimal. If an error is raised through the computation of `ActTaker`, we randomly select one of the options. The final accuracy we report is the accuracy of option selection. Note that this metric is different from the metric of measuring the formula matching accuracy, which is adopted by some existing work (Chen et al., 2020b) and tolerates the inherent noise in formula annotations. The results analyzed under these two metrics are not directly comparable, while we choose the former to study the influence on learning brought by noisy annotations.

Results Table 1 shows the accuracy achieved by our learning framework and baselines on MathQA. It is shown that our proposed method outperforms all the baselines on this dataset. We also conducted an ablation study on whether to employ the DBS discussed in Section 3.4 or not. The result confirmed the performance improvement made by DBS. Compared with the baselines, we only adopt a simple LSTM for the formula inference, which appears to be naive in contrast to the delicately crafted models adopted in existing work. We attribute our success to the autonomous formula exploring capacity of our learning framework.

In our investigation of the annotated formulas provided by MathQA, some noise was found. This means that part of the annotated formulas cannot solve the corresponding questions correctly. Including this noise in the training labels results in the degradation of performance in existing work. However, our learning framework is capable of removing this noise and finding valid formulas for the

question index	14328 (train)
question text	12.5 % of 192 = 50 % of ? **(Answer: 48)**
annotated formula	(((50 × 192) ÷ 100) × 12.5) ÷ 100 **(wrong)**
our formula (raw)	(÷, ×, N_0, ⟨End⟩, N_1, ⟨End⟩, N_2, ⟨End⟩)
(flatten)	(12.5 × 192) ÷ 50
question index	1752 (train)
question text	how many multiples of 4 are there between 8 and 160 ? **(Answer: 37)**
annotated formula	((160 − 8) ÷ 4) + 1 **(wrong)**
our formula (raw)	(−, ÷, −, N_2, ⟨End⟩, N_1, ⟨End⟩, N_0, ⟨End⟩, C_1, ⟨End⟩)
(flatten)	((160 − 8) ÷ 4) − 1
question index	3999 (train)
question text	the average of first 25 natural numbers is ? **(Answer: 13)**
annotated formula	25 + 1 **(wrong)**
our formula (raw)	(+, ÷, N_0, ⟨End⟩, C_2, ⟨End⟩, ÷, C_1, ⟨End⟩, C_2, ⟨End⟩)
(flatten)	(25 ÷ 2) + (1 ÷ 2)
question index	23604 (train)
question text	the telephone company wants to add an area code composed of 2 letters to every phone number . in order to do so , the company chose a special sign language containing 324 different signs . if the company used 322 of the signs fully and two remained unused , how many additional area codes can be created if the company uses all 324 signs ? **(Answer: 1292)**
annotated formula	322 × (324 − 322) × (324 − 322) **(wrong)**
our formula (raw)	(+, ×, N_0, ⟨End⟩, N_1, ⟨End⟩, ×, N_0, ⟨End⟩, N_2, ⟨End⟩)
(flatten)	(2 × 322) + (2 × 324)
question index	285 (test)
question text	today jim is twice as old as fred , and sam is 4 years younger than fred . 4 years ago jim was 8 times as old as sam . how old is jim now ? **(Answer: 20)**
annotated formula	(((8 × 8) − 4) ÷ (8 − 2)) × 2
our formula (raw)	(×, +, N_0, ⟨End⟩, ÷, +, ×, N_0, ⟨End⟩, C_2, ⟨End⟩, −, ×, N_1, ⟨End⟩, N_2, ⟨End⟩, N_1, ⟨End⟩, −, N_0, ⟨End⟩, N_1, ⟨End⟩, C_2, ⟨End⟩) **(wrong)**
(flatten)	(4 + ((4 × 2) + ((4 × 8) − 4)) ÷ (4 − 4)) × 2 **(wrong)**

Table 3: Examples of five questions together with annotated formulas and the formulas discovered by our learning framework. The first four examples show the cases that the annotated formulas are invalid and lead to incorrect answers. However, our learning framework discovered valid formulas for these questions instead. The last example shows a case of failure in the test. In this case, an incorrect formula is predicted by our models and an error is raised through the computations of the modules because of the occurrence of division by zero. In cases like this, formulas cannot produce valid answers even though they are semantically acceptable.

corresponding questions afresh through the search with weak supervision from the answers. This enables our learning framework to achieve higher accuracy in this task. Table 3 provides a case study for this issue.

To strengthen this idea, we also compared the performance of our models trained in different conditions and report the results in Table 2. Here, "raw annotation" refers to the models trained merely on the raw formula annotations. "REINFORCE" refers to the models trained with raw annotations and have REINFORCE (Williams, 1992) implemented to enable a preliminary formula exploration. We restricted the maximum number of attempts on formulas for both REINFORCE and our learning framework to the same bound 10^8. Compared with the learning on raw annotations, although REINFORCE improved the performance to some extent, our method showed a more powerful formula exploring capacity. This comparison verified the advantage of autonomous formula exploration and the superiority of our learning framework over the naive reinforcement learning method.

4.2 Math23K

Experimental setup For Math23K, the data preprocessing and the configuration of PolicyNet are the same as MathQA. Nevertheless, we provided no pretraining data to PolicyNet in this experiment to make a fair comparison with weakly supervised learning baselines. This means that the formula search completely starts from scratch. In addition, we also used the formulas discovered in our formula search to train the GTS model (Xie and Sun, 2019) with its default settings to compare with existing work. For ActTaker, we adopted six Number modules, N_0 to N_5, four Operation modules +, −, ×, and ÷, and two Constant modules, C_1 and C_100.

	Accuracy
Supervised learning approaches	
DNS (Wang et al., 2017)	58.1%
GTS (Xie and Sun, 2019)	74.3%
G2TL (Zhang et al., 2020)	**75.5%**
Weakly supervised learning approaches	
Seq2seq + REINFORCE	12.1%
Seq2seq + MAPO (Hong et al., 2021)	10.7%
Seq2seq + LBF (Hong et al., 2021)	44.7%
Seq2seq + Ours w/o DBS	51.2%
Seq2seq + Ours w DBS	52.4%
GTS + REINFORCE	14.0%
GTS + MAPO (Hong et al., 2021)	20.8%
GTS + LBF (Hong et al., 2021)	59.4%
GTS + Ours w/o DBS	59.8%
GTS + Ours w DBS	**59.9%**

Table 4: The accuracy achieved on Math23K by fully supervised and weakly supervised learning methods under five-fold cross-validation.

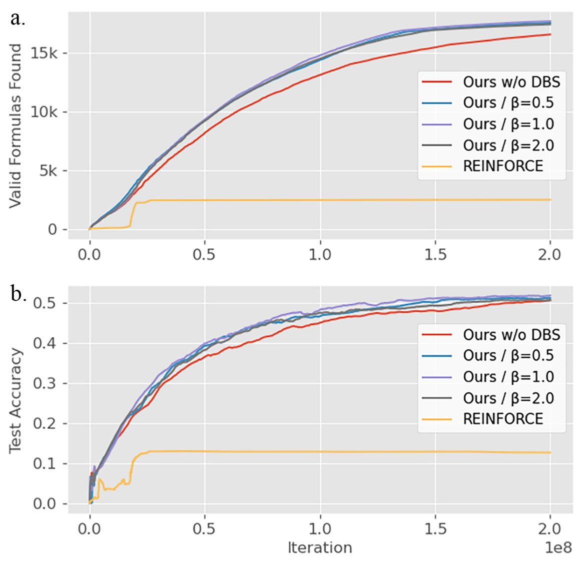

Figure 3: The learning processes of different weakly supervised learning methods measured by two metrics: a. the number of valid formulas found in the training set, b. the accuracy achieved on the test set.

Results Table 4 shows the accuracy achieved by both supervised learning methods and weakly supervised learning methods on Math23K. Although supervised learning methods remain the state-of-the-art approach on this task owing to the natural strength of utilizing ground-truth formula supervision, our learning framework outperformed all the existing weakly supervised approaches on both two inference models LSTM and GTS. In weakly supervised learning environment, learning engines are expected to acquire knowledge on formulas from scratch. Naive reinforcement learning methods represented by REINFORCE and MAPO (Liang et al., 2018) are inadequate in performing this job because their exploring capacity is mainly powered by the estimation made by the inference model, which can hardly deal with some complicated formula modification such as inserting an operator into a formula while keeping the rest of the formula intact. Compared with the learning-by-fixing (LBF) mechanism proposed by Hong et al. (2021), we attribute the superiority of our method to the capacity of managing formula exploration in a broader search space. The fixing mechanism of LBF mainly focuses on the 1-step fix, which assumes that only one symbol in the reasoning tree should be substituted. However, our heuristic formula exploration based on the formula graph and formula score can sample candidate formulas according to the observation on various formulas that are likely to be valid and thus organize a broader heuristic formula exploration.

Moreover, we analyzed the learning processes of different weakly supervised learning methods together with our method with different hyperparameter β in DBS. The result is shown in Figure 3. First of all, it can be noticed that REINFORCE only managed to discover a small number of formulas within the learning in the training set. This led to its poor performance on the test set. Compared with REINFORCE, our method showed a much more powerful formula exploring capacity and higher exploring efficiency. Furthermore, though the ablation study on our method, it is shown that DBS makes the learning process converge faster, discover more valid formulas in the training set, and achieve slightly higher accuracy on the test set. However, the difference brought by β is not quite obvious. From this result, we concluded that the existence of the difference-based score contributed to the heuristic search, but the search process is not very sensitive to its scale.

Further Discussion Although our proposed method has shown remarkable formula exploring capacity as a weakly supervised leaning approach, we are still alert to the gap of performance between our method and supervised learning methods. Generally, this gap can be ascribed to two causes. On the one hand, the size of the space of possible formulas, which is also the size of the search space, can be approximately up to 10^{20}. Such a search space is so huge that some complex ground-truth formulas cannot be guaranteed to be found through a heuristic search by nature. On the other hand, through the search process, incorrect formulas may result in correct answers by accident. For example, for the question “Find the sum of 2 and 2.”, a valid formula could be `N_0+N_1`, but formulas such as `N_0+N_0` and `N_0×N_1` also result in correct answers by accident. These errors can be reduced by providing multiple pairs of num and a to each q, which is also one of our motivations for proposing the template combining process in preprocessing. However, such errors still cannot be entirely avoided for templates exclusively owned by unique questions. Anyway, considering that these are some common challenges faced by all weakly supervised learning approaches, we leave the solution to these problems to future work.

5 Conclusion

This work discussed the issue of formula annotation dependence in existing work on solving mathematical problems. To deal with this issue, we proposed a new learning framework, Weakly Supervised Formula Learner. This framework established a mechanism to learn formulas with weak supervision from final answers and enabled a heuristic formula search in the space of possible formulas.

In the experiments, our learning framework showed remarkable formula exploring capacity on both MathQA and Math23K datasets. Particularly, on MathQA, we illustrated that our models trained with formulas discovered in formula exploration outperformed baselines trained with complete yet imperfect formula annotations. On Math23K, our learning framework showed more powerful formula exploring capacity than existing weakly supervised learning methods. In view of this evidence, we consider our proposed learning framework a valid and advanced approach for solving mathematical problems with weak supervision from their answers.

Acknowledgement

This work was supported by Institute of AI and Beyond of the University of Tokyo, JSPS KAKENHI Grant Number JP19H04166, and the commissioned research (No. 225) by National Institute of Information and Communications Technology (NICT), Japan.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016a. Learning to compose neural networks for question answering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1545–1554.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016b. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48.

Kezhen Chen, Qiuyuan Huang, Hamid Palangi, Paul Smolensky, Kenneth D Forbus, and Jianfeng Gao. 2020a. Mapping natural-language problems to formal-language solutions using structured neural representations. In *International Conference on Machine Learning*, pages 1566–1575.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V Le. 2020b. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Yining Hong, Qing Li, Daniel Ciao, Siyuan Huang, and Song-Chun Zhu. 2021. Learning by fixing: Solving math word problems with weak supervision. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(6):4959–4967.

Mark Hopkins, Cristian Petrescu-Prahova, Roie Levin, Ronan Le Bras, Alvaro Herrasti, and Vidur Joshi. 2017. Beyond sentential semantic parsing: Tackling the math SAT with a cascade of tree transducers. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 795–804.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2989–2998.

Aishwarya Kamath and Rajarshi Das. 2019. A survey on semantic parsing. In *Automated Knowledge Base Construction (AKBC)*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc Le, and Ni Lao. 2018. Memory augmented policy optimization for program synthesis and semantic parsing. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 10015–10027.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167.

Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. 2021. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1132–1142.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018. Translating a math word problem to a expression tree. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1064–1069.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854.

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

Yuxuan Wu and Hideki Nakayama. 2020. Graph-based heuristic search for module selection procedure in neural module network. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *IJCAI*, pages 5299–5305.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. 2018. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 1039–1050.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937.