Generate & Rank: A Multi-task Framework for Math Word Problems

Jianhao Shen[1†], Yichun Yin[2], Lin Li[3], Lifeng Shang[2],
Xin Jiang[2], Ming Zhang[1*], Qun Liu[2]

[1]Department of Computer Science, School of EECS, Peking University
[2]Huawei Noah's Ark Lab
[3]Huawei HiSilicon
{jhshen, mzhang_cs}@pku.edu.cn
{yinyichun, lilin29, shang.lifeng, jiang.xin, qun.liu}@huawei.com

Abstract

Math word problem (MWP) is a challenging and critical task in natural language processing. Many recent studies formalize MWP as a generation task and have adopted sequence-to-sequence models to transform problem descriptions to mathematical expressions. However, mathematical expressions are prone to minor mistakes while the generation objective does not explicitly handle such mistakes. To address this limitation, we devise a new ranking task for MWP and propose Generate & Rank, a multi-task framework based on a generative pre-trained language model. By joint training with generation and ranking, the model learns from its own mistakes and is able to distinguish between correct and incorrect expressions. Meanwhile, we perform tree-based disturbance specially designed for MWP and an online update to boost the ranker. We demonstrate the effectiveness of our proposed method on the benchmark and the results show that our method consistently outperforms baselines in all datasets. Particularly, in the classical Math23k, our method is **7%** (78.4% → **85.4%**) higher than the state-of-the-art[1].

1 Introduction

Solving math word problems (MWP) (Bobrow, 1964) is an important and fundamental task in natural language processing (NLP), which requires to provide a solution expression given a mathematical problem description, as illustrated in Table 1. Many recent studies formalize MWP as a generation task and commonly adopt LSTM-based sequence-to-sequence (Seq2Seq) models (Wang et al., 2017, 2018b; Xie and Sun, 2019), where problem texts are source sequences, mathematical expressions are target sequences and the model learns the mapping

Original MWP	
Problem	A project is completed in 25 days by 12 workers. If it takes 20 days to complete, how many workers will it take?
Solution	*25 * 12 / 20*
Number-mapped MWP	
Problem	A project is completed in *NUM0* days by *NUM1* workers. If it takes *NUM2* days to complete, how many workers will it take?
Solution	*NUM0 * NUM1 / NUM2*

Table 1: An example of MWP, where numbers are usually mapped to special tokens, such as *Num0/1/2*.

from source texts to target expressions. These studies have proposed numerous advanced techniques to improve the MWP solver, but their performance is still unsatisfactory yet.

We argue that it is not sufficient to model MWP as only a generation task, because there is a significant difference between mathematical expressions and natural language sequences: one minor mistake in a mathematical expression will change the whole semantic thus lead to a wrong answer, whereas natural language is more robust to such minor mistakes. The objective function of the generation task is to maximize generation likelihood on ground-truth expressions, which does not have an explicit strategy to make the model learn to distinguish between ground-truth and expressions that have minor mistakes. In addition, previous works (Liu et al., 2019a; Xie and Sun, 2019; Zhang et al., 2020) find that the performance of generation models degrades fast as the expression gets longer.

To handle the above problems, we propose Generate & Rank, a multi-task framework for MWP, which introduces a new ranker to explicitly distinguish between correct and incorrect expressions. Specifically, our framework includes two modules: a generator and a ranker. The former is designed to generate candidate expressions given a problem text and the latter aims to rank the candidate

expressions. They are built based on an encoder-decoder model and are jointly trained with generation loss and ranking loss. In this work, we build our model based on BART (Lewis et al., 2020), a widely used pre-trained language model that achieves SOTA performance on various sequence-to-sequence tasks (Ahmad et al., 2021; Liu et al., 2020). During multi-task training, expressions produced by the generator are used to construct an expression bank and train the ranker, in which way the model can learn from its own mistakes. To construct more informative candidates for the ranker, we specially design tree-based disturbance for MWP. We also introduce an online update mechanism to generate a new set of candidate expressions at each training epoch. The overall training procedure is in an iterative manner, in which the ranker and generator continue to enhance each other.

To evaluate the effectiveness of the proposed model, we conduct extensive experiments on the datasets of Math23K (Wang et al., 2017) and MAWPS (Koncel-Kedziorski et al., 2016). The results show that our model outperforms typical baselines. Particularly, we obtain an improvement of 7% in the Math23K dataset that is extensively studied. Moreover, we do ablation study and model analysis, which shows that (1) joint training improves the performance of the generator and ranker over separate training; (2) both strategies of constructing candidate expressions and online updating are important to the success of the ranker. We also find that with the ranker, our model achieves a large improvement in generation of long expressions.

The contributions of our work are two-fold: (1) We propose Generate & Rank, a new multi-task framework to train a pre-trained language model for math word problem solving. To construct informative candidate expressions for the ranker, we propose two effective generation methods and also introduce an online update strategy. (2) Experiments show that our proposed model consistently outperforms the state-of-the-art models and achieves a significant improvement on the Math23K dataset.

2 Preliminaries

2.1 Math Word Problem

A math word problem P is a sequence of word tokens and numeric values, which typically describes a partial quantitative state of a world and some updates or relationships among quantities, then asks a question about an unknown quantity. The solution S to the question is a mathematical expression that consists of math operators and numbers. In solving a math word problem, we usually do not care about the specific number of a quantity, so the numbers in problems and solution expressions are mapped to special tokens *NUM#i* according to their orders in the problem text. Table 1 gives an example of an original math word problem and the corresponding number-mapped problem.

2.2 BART

BART is a widely-used pre-trained language model. It follows a standard encoder-decoder structure using Transformer layers (Vaswani et al., 2017) and is pre-trained with text denoising tasks. The pre-trained BART can be fine-tuned for tasks of sequence classification and generation.

Transformer-based Encoder-Decoder. BART uses an encoder-decoder structure that is the mainstream architecture for sequence-to-sequence tasks. The encoder adopts the bidirectional self-attention to map an input sequence of tokens $P = (x_1, x_2, \ldots, x_n)$ to a sequence of continuous representations $\mathbf{R} = (\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_n})$. The BART encoder is composed of multiple Transformer layers, each consists of a multi-head self-attention (MHA) module and a fully connected feed-forward (FFN) module. We denote the mapping function of the BART encoder as follows:

$$(\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_n}) = \text{BART}_{\text{Enc}}(x_1, x_2, \ldots, x_n) \quad (1)$$

The BART decoder also consists of multiple Transformer layers. Besides MHA and FFN modules, the decoder layer adds another multi-head attention over the output of the encoder. The decoder takes in one token s_i at a time, and gives an output state based on the output of the encoder and previous tokens in the decoder input. This output state is then fed into a linear transformation followed by a softmax function to get the predicted next-token probabilities. This one-step decoding process is denoted as follows:

$$P(*) = \text{softmax}(\mathbf{d}_i \mathbf{W} + \mathbf{b}) \quad (2)$$

$$\mathbf{d}_i = \text{BART}_{\text{Dec}}(\mathbf{R}; s_0, s_1, \ldots, s_{i-1}), \quad (3)$$

where s_0 is a special *[bos]* token indicating the start of decoding, and $\mathbf{R}$ is the output of encoder.

BART Pre-training. BART is pre-trained by the tasks of recovering a corrupted document to orig-
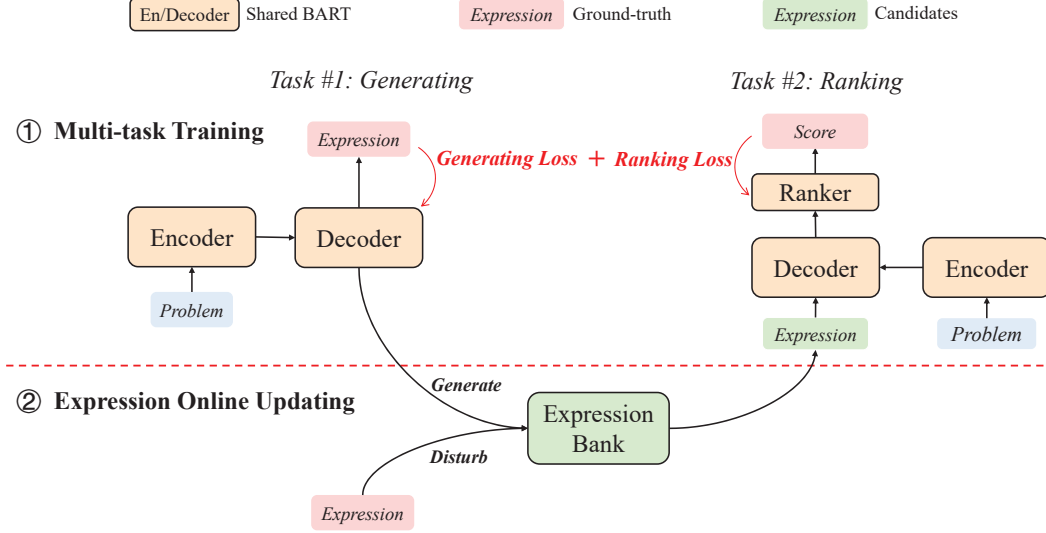
Figure 1: Our proposed Generate & Rank framework for BART-based MWP solver. The model consists of a generator and a ranker. They share BART encoder and decoder, and are jointly trained with generating loss and ranking loss. We construct an expression bank for training the ranker with expressions produced by the generator and ones obtained by tree-based disturbance. The expression bank is updated every epoch so that the model can constantly learn from new informative examples.

inal one. The input to BART is corrupted in two ways: (1) a number of text spans are replaced with a single *[MASK]* token; (2) sentences in a document are shuffled in a random order. The objective of BART pre-training is to minimize the cross-entropy loss between the decoder's generation probabilities and the ground-truth of original document.

3 Methodology

We propose Generate & Rank, a BART-based multi-task framework for math word problems. Our model consists of a generator and a ranker, which share a BART model and are jointly trained with a generating task and ranking task. The objective of generating is to generate expressions given a math word problem. We also add a ranking task so that the model can select a correct expression from a set of candidates. We construct an expression bank to provide training examples for the ranker. Figure 1 shows our proposed framework and we introduce details for each task and the whole framework in the following sections.

3.1 Multi-task Training

Task #1: Generating. We first formulate the math word problem as a sequence-to-sequence task, in which BART is trained to generate solution expressions given a math word problem. Following the fine-tuning strategy of BART (Lewis et al., 2020), we take problem text, a sequence of tokens

$P = (x_1, x_2, \ldots, x_n)$, as input to BART encoder, and minimize negative log-likelihood of the solution expression $S = (s_1, s_2, \ldots, s_m)$,

$$J_{\text{GEN}} = \frac{1}{|\mathcal{D}|} \sum_{(P,S)\in\mathcal{D}} -\log \Pr(S|P), \quad (4)$$

where the conditional probability is decomposed in an auto-regressive way as:

$$\Pr(S|P) = \prod_{i=1}^{m} \Pr(s_i|P, S_{j<i}) \quad (5)$$

$$\Pr(*|P, S_{j<i}) = \text{softmax}(\mathbf{d}_i \mathbf{W} + \mathbf{b}) \quad (6)$$

$$\mathbf{d}_i = \text{BART}_{\text{Dec}}(\mathbf{R}; S_{j<i}) \quad (7)$$

$$\mathbf{R} = \text{BART}_{\text{Enc}}(P). \quad (8)$$

Additionally, we add two special tokens $s_1 =$*[bos]* and $s_m =$*[eos]* to indicate the start and end symbols of decoding sequences.

Task #2: Ranking. Through generating, we obtain many candidate solution expressions. To decide which expression is a correct solution to the problem, we propose a ranking task which is essentially a task of sequence pair classification. Given pairs of problems and candidate expressions, the ranker chooses the expression with highest ranking score as the final solution to the problem. Specifically, we add an MLP classifier on top of the final layer hidden state of the last decoder token. The last

decoder token is always a special *[eos]* token and its corresponding hidden state can attend to all token representations of problem text and expression. Same as the generation task, we feed the problem text into the encoder and expression into the decoder, obtaining sequence representations. The last decoder representation is then taken as input to the classifier for ranking score prediction:

$$\Pr(\cdot|P, S) = \text{softmax}(\mathbf{d}'_{m+1}) \tag{9}$$

$$\mathbf{d}'_{m+1} = \tanh(\mathbf{d}_{m+1}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{10}$$

$$\mathbf{d}_{m+1} = \text{BART}_{\text{Dec}}(\mathbf{R}; S), \tag{11}$$

where $\mathbf{R}$ is the output of the encoder, S is the expression token sequence, $\mathbf{d}_{m+1}$ is the decoder representation of the last token, and $\mathbf{W}_{1|2}$ and $\mathbf{b}_{1|2}$ are trainable parameters. The training objective of the ranker is cross-entropy between classifier output and correct labels,

$$J_{\text{RANK}} = -\frac{1}{|\mathcal{D}^+ \cup \mathcal{D}^-|}\Bigg[\sum_{(P,S)\in\mathcal{D}^+} \log\Pr(1|P, S)$$
$$+ \sum_{(P,S)\in\mathcal{D}^-} \log\Pr(0|P, S)\Bigg] \tag{12}$$

where $\mathcal{D}^+$ and $\mathcal{D}^-$ are sets of positive and negative examples, respectively. We introduce how to generate negative examples in the next section.

Optimization Objective. We train the model on the joint loss of two tasks together:

$$J = J_{\text{GEN}} + J_{\text{RANK}}. \tag{13}$$

and the two modules share BART parameters.

3.2 Expression Bank

By definition, any expression that does not equal the ground-truth can serve as a negative example, but we cannot use all of them due to limited computational resources. To train the ranker efficiently, we use two different strategies, namely model-based generation and tree-based disturbance, to construct an expression bank for ranker training.

Model-based Generation. The first strategy is to produce new expressions with the generator. Specifically, given a problem, we use beam search with the generator to produce top-K expressions. Each expression is labeled as positive or negative depending on whether its calculation result equals the result of ground-truth.

Tree-based Disturbance. Our second way to construct new expressions is adding disturbance to ground-truth expressions. We design four kinds of disturbances which are illustrated in Figure 2. The ground-truth expression is first transformed to an abstract syntax tree (AST) (Liu et al., 2019a). Then we disturb tree nodes or sub-structures to produce new expressions in four ways: **a) Expand.** A leaf node is expanded into a sub-tree with a new operation and a number. **b) Edit.** A node is randomly changed to another while keeping the expression valid (i.e., a number node will be changed to another number, and an operator node to another operator). **c) Delete.** Delete a leaf node and replace its father with its sibling node. **d) Swap.** Swap the left and right children of an operation node.

We use the above methods to construct the expression bank. Since new expressions may also be correct (for example, swapping two operands of addition or multiplication), we compare the numerical results of newly obtained expressions with that of the ground-truth, and add them to positive or negative samples depending on the comparison. Then both positive and negative pairs are sampled from this expression bank for the multi-task training. In order to make the model learn with more informative examples, we do an online update for expression bank, which means that we use new expressions obtained by model-based generation and tree-based disturbance at each training epoch.
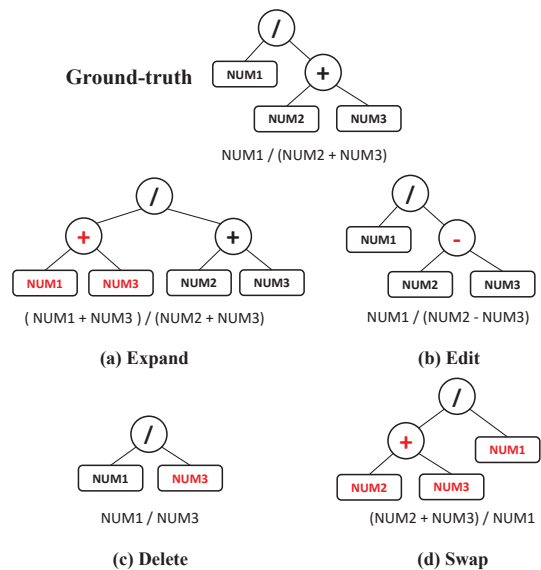

Figure 2: Overview of tree-based disturbance.

Algorithm 1 Training Algorithm

Input: MWP Dataset $\mathcal{D} = \{(P, S)\}$
Parameter: Pre-trained BART encoder and decoder parameters θ_e and θ_d, random initialized ranker θ_v, beam size K, epoch number M

1: // Fine-tune the generator
2: **for** $epoch = 1$ to M **do**
3:     Fine-tuning BART encoder θ_e and decoder θ_d on $\mathcal{D}$ with generation loss Eq. (4).
4: **end for**
5: // Construct expression bank
6: $\mathcal{D}^+ \leftarrow \mathcal{D}, \mathcal{D}^- \leftarrow \{\}$
7: **for** $(P, S) \in \mathcal{D}$ **do**
8:     Generate top-K expressions $\{\bar{S}_i\}$ for problem P with beam search
9:     Get new expressions $\{\bar{S}_i'\}$ by adding tree-based disturbance to S
10:     $\{\bar{S}_i\} \leftarrow \{\bar{S}_i\} \cup \{\bar{S}_i'\}$
11:     **for** $\bar{S} \in \{\bar{S}_i\}$ **do**
12:         **if** result of $\bar{S}$ equals result of S **then**
13:             $\mathcal{D}^+ \leftarrow \mathcal{D}^+ \cup \{(P, \bar{S})\}$
14:         **else**
15:             $\mathcal{D}^- \leftarrow \mathcal{D}^- \cup \{(P, \bar{S})\}$
16:         **end if**
17:     **end for**
18: **end for**
19: // Joint training
20: **for** $epoch = 1$ to M **do**
21:     Train θ_e, θ_d, θ_v w.r.t. the joint loss Eq.(13) on $\mathcal{D}^+$ and $\mathcal{D}^-$
22:     Repeat lines 6-18 to reconstruct expression bank
23: **end for**

3.3 Training Procedure

The training procedure includes multi-task training and expression online updating. We first fine-tune the pre-trained BART for the generation task (J_{GEN} in Eq. 4). After that, we use the fine-tuned BART and tree-based disturbance to generate expressions as the training samples for the ranker. Then we do the joint training of generation and ranking. This process is performed in an iterative manner and the two modules (i.e., generator and ranker) continue to enhance each other. Meanwhile, training examples for ranking are updated after each epoch. We summarize the overall training procedure in Algorithm 1.

3.4 Model Inference

We perform a two-stage model inference, namely generation and ranking. Specifically, given a new problem text sequence P, we first pass it to the encoder to get the problem representation $\mathbf{R}$. Then we perform the beam search to generate top-K expressions. These generated expressions are used as candidate solutions for the ranker. All expressions are passed to the ranker and that with the highest score is selected as the final result.

4 Experiment

4.1 Experimental Setup

Datasets. We conduct the experiments on two commonly-used datasets: Math23K (Wang et al., 2017) and MAWPS (Koncel-Kedziorski et al., 2016). Math23K is a large-scale Chinese dataset that contains 23,162 math word problems and their corresponding expression solutions. MAWPS is a English dataset containing 2,373 problems. All the problems are one-unknown-variable linear problems and can be solved with a single expression.

Baselines. We compare our model with the following baselines including the state-of-the-art models: DNS (Wang et al., 2017) uses a vanilla Seq2Seq model to generate expressions. Math-EN (Wang et al., 2018b) uses the equation normalization to avoid equation duplication problem. T-RNN (Wang et al., 2019b) applies recursive neural networks to model the tree structures of expressions. S-Aligned (Chiang and Chen, 2019) tracks the semantic meanings of operands with a stack during decoding. Group-ATT (Li et al., 2019) leverages the attention mechanism to enrich problem representation. Both AST-Dec (Liu et al., 2019a) and GTS (Xie and Sun, 2019) develop a tree-based decoder to generate expressions. Graph2Tree (Zhang et al., 2020) proposes to build a quantity cell graph and a comparison graph to better capture the quantity relationships of the problem. Multi-E/D (Shen and Jin, 2020) is an ensemble model which combines multiple encoders and decoders.

Implementation Details. We use the PyTorch[2] implementations and pre-trained language models provided by the Transformers library[3]. Since the Math23K dataset is a Chinese dataset and officially released BART is only for English, we switch to

---

[2] https://pytorch.org/
[3] https://github.com/huggingface/transformers

2273

mBART25 (Liu et al., 2020), which is a multilingual BART for 25 languages including Chinese. For the MAWPS dataset, we also use mBART25. We optimize our model with AdamW (Loshchilov and Hutter, 2019). The training hyperparameters are set as follows. We set the batch size to 128, the learning rate to 5e-5 and the warm-up ratio to 0.1. The weight decay is set to 0.01. The number of epochs M for fine-tuning and multi-task training are set to 50. We set beam size K to 10 in beam search and expression bank size to 20 unless otherwise stated. All experiments are carried out on NVIDIA Tesla V100. We use 8 GPUs for training and 1 for testing. For our proposed framework, the training time is 1.5 hours for one epoch and testing time is 15 minutes for the whole test set.

Evaluation Metric. Both MAWPS and Math23K are evaluated with a metric of "solution accuracy", that is, the expression is considered as correct if it induces the same number as the ground-truth. For the Math23K dataset, some baselines are evaluated using the public available test set while others use the results of 5-fold cross-validation. We report our results on both settings. For the MAWPS dataset, models are evaluated with 5-fold cross-validation.

4.2 Results and Analysis

Evaluation results of our model and baselines are summarized in Table 2. We observe that: (1) direct fine-tuning of mBART already outperforms the state-of-the-art models on Math23K, which shows the powerful generation ability of mBART. (2) on MAWPS, mBART outperforms most Seq2Seq baselines but is worse than GTS and Graph2Tree. These two models leverage tree structure of expressions during decoding which is critical for math word problem solving. We believe that pre-trained language models would achieve a better performance if combined with structure information, and we leave it as a future work[4]. (3) Generate & Rank framework further improves mBART and achieves new state-of-the-art results. In particular, Generate & Rank outperforms mBART baselines by more than **4%** in all the evaluation settings and also outperforms the previous best models by **7%** on Math23K[†], **7.4%** on 5-fold cross-validation Math23K[‡]. The improvement over pre-trained mBART demonstrates the effectiveness of

---

[4]One may think that the sequence decoder might not always generate valid expressions. However, we check all expressions generated by mBART and find that 99.9% are valid.

our multi-task training framework.

Model	Math23K[†]	Math23K[‡]	MAWPS[‡]
DNS	-	58.1	59.5
Math-EN	66.7	-	69.2
T-RNN	66.9	-	66.8
S-Aligned	-	65.8	-
Group-ATT	69.5	66.9	76.1
AST-Dec	69.0	-	-
GTS	75.6	74.3	82.6
Graph2Tree	77.4	75.5	83.7
Multi-E/D	78.4	76.9	-
mBART	80.8	80.0	80.1
Generate & Rank	**85.4**	**84.3**	**84.0**

Table 2: Solution accuracy on MAWPS and Math23K. † refers to the result of test set and ‡ denotes the result of 5-fold cross-validation. "-" means that the results are not reported in the original papers.

4.3 Ablation Study and Model Analysis

To better understand our model, we further conduct ablation study on Math23K to show how the proposed components affect performance.

4.3.1 Effect of Joint Training

To investigate the effect of joint training, we introduce the baseline of two-stage training (i.e., w/o Joint), which means we first train the generator, then train the ranker, and the modules are trained independently. We also study the effect of joint training on generation and perform comparison between mBART and our generator (i.e., w/o Ranker). The results are listed in Table 3. We can see that the joint training brings 2.2% improvement compared with the two-stage training and 2.6% for the generator compared with the mBART trained alone, suggesting that the joint training of generator and ranker benefits each other. Besides, the joint training is more space efficient since we only need to save one unified model rather than two.

Model	Acc
Generate & Rank	**85.4**
w/o Joint	83.2
w/o Ranker	83.4
w/o both (mBART)	80.8

Table 3: Effect of joint training.

4.3.2 Effect of Expression Bank Strategy

We investigate the effect of different strategies to construct the expression bank. Here we choose a random sampling strategy as our baseline, where

the set of expressions that appeared in the training data is sampled as the expression bank. We evaluate different strategies with and without online updating and summarize the results in Table 4.

Strategy	Online	w/o Online
Random Sample	75.2	69.7
Model	84.2	83.2
Model+Tree	**85.4**	83.1

Table 4: Accuracy for different expression bank strategies. The expression bank size is 20 for all settings.

We can see that our strategies outperform the random sampling strategy. Since the ground-truth can not be accessed during model inference, we cannot use the tree-based disturbance to generate candidate expressions as in the training phase. This discrepancy between training and inference leads to poor performance if we only use tree-based disturbance to construct the expression bank. However, combining the tree-based disturbance and model-based generation strategies, we can obtain better results than the only model-based generation, which gives evidence that the tree-based disturbance contains some informative examples that the generator does not cover and it is possible to improve the performance based on the human knowledge of math expression.

We can also see that strategies have a performance drop without online updating. We conjecture that without online updating the ranker may tend to memorize existing negative expressions thus generalize poorly on new problems. As for strategies with model-based generation, there is another possible reason: the generator keeps updating during multi-task training, so the previously generated expressions are no longer good samples of the current model, and newly generated expressions are more informative. To summarize, both strategies of constructing the expressions bank and online updating play an important role in the success of the ranker.

4.3.3 Impact of Expression Bank Size

We further analyze the impact of expression bank size on the ranker and results are shown in Figure 3. If the model-based generation is used, performance reaches the best at expression bank size 20. This suggests that the expression bank size should not be too small nor too large. One possible reason may be that the generated expressions cannot cover

#Op	Pro	AST-Dec	G2T	mBART	Generate & Rank
1	17.3	82.7	85.5	90.2	90.8 (+0.6)
2	52.2	74.5	83.7	88.1	90.2 (+2.1)
3	19.1	59.9	71.7	71.2	79.1 (+7.9)
4	6.6	42.4	51.5	53.0	63.6 (+10.6)
5	3.4	44.1	38.2	41.2	58.8 (+17.6)
6	0.9	55.6	55.6	55.6	88.8 (+33.2)

Table 5: Accuracy for increasing length of expressions. #Op is the number of operations in expressions. Pro denotes proportion of expressions with different lengths.

possible mistakes when the expression bank is too small, and when the expression bank is too large, low-quality expressions may be generated and hinder ranker training. Tree-based disturbance has a similar trend and the best bank size is 10.
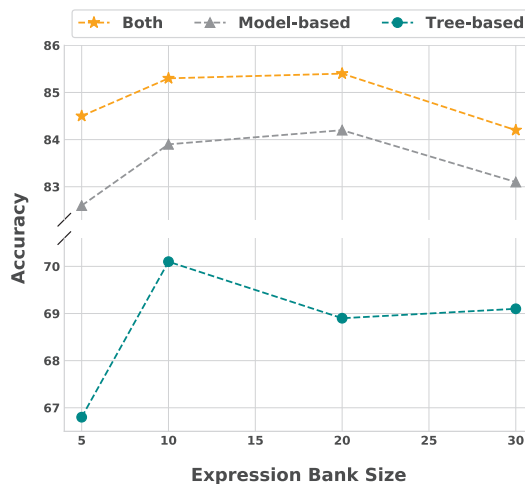

Figure 3: Accuracy with different expression bank sizes from 5 to 30.

4.3.4 Model Analysis

In Table 5, we list how the model accuracy changes with respect to the number of operations in expressions. We do not discuss the case of 6 operators since it has too few examples and high variance. For expressions less than 6 operators, all models perform worse when the expression gets longer. This is as expected since longer expressions require more steps of reasoning and have less data to train. In addition, we also observe that Generate & Rank training has larger improvement over fine-tuned mBART on longer expressions. This implies that our model is more suitable to handle complex problems and expressions.

Following Liu et al. (2019a), we also examine the performance of our model in different domains. The domain of each problem is defined by whether it contains any keywords of this domain and we

use the same keyword list as Liu et al. (2019a). Table 6 shows the results. We observe the similar pattern that the fine-tuned mBART has limitations in geometry which requires external knowledge such as formulas for the circumference and area of a circle. Interestingly, our proposed model mainly improves on these domains. This suggests that the ranking task may be a better choice to learn and use mathematical knowledge than generating.

Domain	Pro	mBART	Generate & Rank
Distance & Speed	11.8	83.9	83.9
Tracing	2.7	85.2	85.2
Engineering	5.8	86.2	**87.9**
Interval	0.6	66.7	66.7
Circle Geometry	1.9	73.7	**78.9**
Plane Geometry	1.2	75.0	**83.3**
Profit	1.1	72.7	72.7
Solid Geometry	1.6	81.3	**87.5**
Interest Rate	0.9	100.0	100.0
Production	0.4	100.0	100.0

Table 6: Accuracy for different problem domains. Pro denotes the proportion of each domain in the test data. Note that the sum of proportion is not 100% since there are problems not belonging to any specified domain.

5 Related Work

5.1 Math Word Problem

Rule-based methods. Early approaches on math word problems mainly craft rules and templates for pattern matching (Bobrow, 1964; Slagle, 1965; Fletcher, 1985; Bakman, 2007). These methods rely heavily on manual design and can only solve a limited scope of problems.

Parsing-based methods. Later on, researchers use statistical methods to solve MWP and achieve a great performance improvement. One line of research focuses on semantic parsing, which leverages traditional machine learning techniques to identify entities, quantities, and operators from the problem text. Roy et al. (2015) proposes three types of classifiers to identify different elements of problems. ARIS (Hosseini et al., 2014) splits the problem into fragments and updates a logic template named *state* by verb categorization. Other works (Sundaram and Khemani, 2015; Mitra and Baral, 2016; Liang et al., 2016) follow a similar process with different templates and annotations.

Two-stage methods. Another research line first obtains an expression template then maps numbers to the template slots. Kushman et al. (2014) train a classifier to select from a set of pre-defined templates. Roy and Roth (2015) propose to construct candidate expressions in a bottom-up manner and train a global scoring function to guide the beam search process. ALGES (Koncel-Kedziorski et al., 2015) converts the process of searching valid expressions to an integer linear programming problem and adopts a different scoring function. Unit-Dep (Roy and Roth, 2017) proposes Unit Dependency Graph to enhance the scoring function.

Deep learning methods. Recently, deep learning models have become prevailing methods for math word problems. DNS (Wang et al., 2017) is the first to apply vanilla RNN-based models to MWP. Math-EN (Wang et al., 2018b) introduces equation normalization and compares three Seq2Seq models on MWP solving. Group-ATT (Li et al., 2019) uses multi-head attention to capture different aspects of features. Some works also leverage tree structures and graph information to improve performance (Wang et al., 2019b; Chiang and Chen, 2019; Liu et al., 2019a; Xie and Sun, 2019; Zhang et al., 2020). Shen and Jin (2020) propose a model of multi-encoders and multi-decoders.

5.2 Pre-trained Language Model

Pre-trained language models have obtained state-of-the-art results in many NLP benchmarks (Wang et al., 2018a, 2019a). These models are usually based on Transformer layers (Vaswani et al., 2017) and trained on large corpus with self-supervised tasks. According to their architectures, pre-trained language models can be categorized into three types: encoder-only, decoder-only and encoder-decoder models. BERT (Devlin et al., 2019) is an encoder-only model which firstly proposes masked token prediction and next sentence prediction to train a language representation model. Following this, many other models are proposed like RoBERTa (Liu et al., 2019b) and SpanBERT (Joshi et al., 2020). Decoder-only models are typically auto-regressive models trained to estimate the probability distribution of a text corpus, including GPT2 (Radford et al., 2019), GPT3 (Brown et al., 2020) and XLNet (Yang et al., 2019). Encoder-decoder models like BART (Lewis et al., 2020) and T5 (Raffel et al., 2020) use the encoder-decoder architecture and are trained on sequence-to-sequence tasks such as text denoising and translation.

6 Conclusion and Future Work

We propose Generate & Rank, a new multi-task framework for math word problems. Specifically, our model has a generator and a ranker which enhance each other with joint training. We also use tree-based disturbance and online update to further improve the performance. The experimental results on the benchmark show that our work consistently outperforms baselines in all datasets. In future work, we will explore the generation and ranking framework to other tasks like summarization and translation.

Acknowledgements

References

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*.

Yefim Bakman. 2007. Robust Understanding of Word Problems with Extraneous Information. *arXiv:math/0701393*.

Daniel G. Bobrow. 1964. Natural Language Input for a Computer Problem Solving System.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*.

Ting-Rui Chiang and Yun-Nung Chen. 2019. Semantically-Aligned Equation Generation for Solving and Reasoning Math Word Problems. 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 2656–2668.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 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 4171–4186.

Charles R. Fletcher. 1985. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, 17(5):565–571.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to Solve Arithmetic Word Problems with Verb Categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Transactions of the Association for Computational Linguistics*, 8(0):64–77.

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.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A Math Word Problem Repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to Automatically Solve Algebra Word Problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Jierui Li, Lei Wang, Jipeng Zhang, Yan Wang, Bing Tian Dai, and Dongxiang Zhang. 2019. Modeling Intra-Relation in Math Word Problems with Different Functional Multi-Head Attentions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6162–6167.

Chao-Chun Liang, Kuang-Yi Hsu, Chien-Tsung Huang, Chung-Min Li, Shen-Yu Miao, and Keh-Yih Su. 2016.

A tag-based statistical English math word problem solver with understanding, reasoning and explanation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 4254–4255.

Qianying Liu, Wenyv Guan, Sujian Li, and Daisuke Kawahara. 2019a. Tree-structured Decoding for Solving Math Word Problems. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2370–2379.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual Denoising Pre-training for Neural Machine Translation. *Transactions of the Association for Computational Linguistics*, 8(0):726–742.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019*.

Arindam Mitra and Chitta Baral. 2016. Learning To Use Formulas To Solve Simple Arithmetic Problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2153.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Subhro Roy and Dan Roth. 2015. Solving General Arithmetic Word Problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752.

Subhro Roy and Dan Roth. 2017. Unit Dependency Graph and Its Application to Arithmetic Word Problem Solving. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1).

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

Yibin Shen and Cheqing Jin. 2020. Solving Math Word Problems with Multi-Encoders and Multi-Decoders. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2924–2934.

James R. Slagle. 1965. Experiments with a deductive question-answering program. *Communications of the ACM*, 8(12):792–798.

Sowmya S Sundaram and Deepak Khemani. 2015. Natural Language Processing for Solving Simple Word Problems. In *Proceedings of the 12th International Conference on Natural Language Processing*, pages 394–402.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, pages 3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018a. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018b. 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.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019b. Template-Based Math Word Problem Solvers with Recursive Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7144–7151.

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.

Zhipeng Xie and Shichao Sun. 2019. A Goal-Driven Tree-Structured Neural Model for Math Word Problems. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 5299–5305.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019.

Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32, pages 5754–5764.

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.