Semantically-Aligned Universal Tree-Structured Solver for Math Word Problems

Jinghui Qin[1], **Lihui Lin**[2], **Xiaodan Liang**[1,2,*], **Rumin Zhang**[2], **Liang Lin**[1,2]
[1] Sun Yat-sen University
[2] Dark Matter AI Inc.
qinjingh@mail2.sysu.edu.cn, linlh23@mail2.sysu.edu.cn,
xdliang328@gmail.com, rm_zhang@foxmail.com,
linliang@ieee.org

Abstract

A practical automatic textual math word problems (MWPs) solver should be able to solve various textual MWPs while most existing works only focused on one-unknown linear MWPs. Herein, we propose a simple but efficient method called Universal Expression Tree (UET) to make the first attempt to represent the equations of various MWPs uniformly. Then a semantically-aligned universal tree-structured solver (SAU-Solver) based on an encoder-decoder framework is proposed to resolve multiple types of MWPs in a unified model, benefiting from our UET representation. Our SAU-Solver generates a universal expression tree explicitly by deciding which symbol to generate according to the generated symbols' semantic meanings like human solving MWPs. Besides, our SAU-Solver also includes a novel subtree-level semantically-aligned regularization to further enforce the semantic constraints and rationality of the generated expression tree by aligning with the contextual information. Finally, to validate the universality of our solver and extend the research boundary of MWPs, we introduce a new challenging **H**ybrid **M**ath **W**ord **P**roblems dataset (HMWP), consisting of three types of MWPs. Experimental results on several MWPs datasets show that our model can solve universal types of MWPs and outperforms several state-of-the-art models[1].

1 Introduction

Math word problems (MWPs) solving aims to automatically answer a math word problem by understanding the textual description of the problem and reasoning out the underlying answer. A typical MWP is a short story that describes a partial state of the world and poses a question about an unknown quantity or multiple unknown quantities. Thus, a machine should have the ability of natural language understanding and reasoning. To solve an MWP, the relevant quantities need to be identified from the text, and the correct operators and their computation order among these quantities need to be determined.

[*]Corresponding Author

[1]The code and the new HMWP dataset are available at https://github.com/QinJinghui/SAU-Solver.

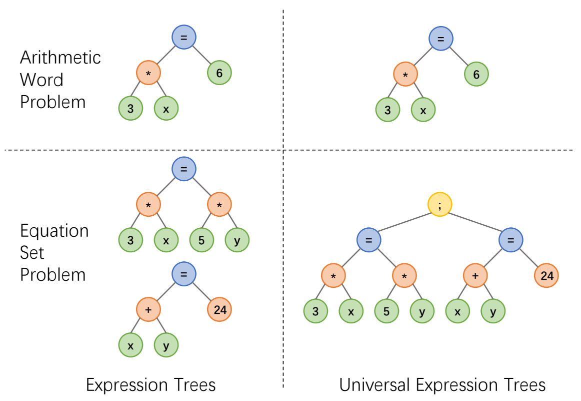


Figure 1: Universal Expression Trees (UET). In our UET representation, multiple expression trees underlying a MWP will be integrated as an universal expression tree (UET) via symbol extension. UET can enable a solver to handle multiple types of MWPs in an unified manner like a single expression tree of an equation.

Many traditional methods (Yuhui et al., 2010; Kushman et al., 2014; Shi et al., 2015) have been proposed to address this problem, but they relied on tedious hand-crafted features and template annotation, which required extensive human efforts and knowledge. Recently, deep learning has opened a new direction towards automatic MWPs solving (Wang et al., 2017; Huang et al., 2018; Wang et al., 2018b, 2019; Xie and Sun, 2019; Chiang and Chen, 2019). Most of deep learning-based methods try to train an end-to-end neural network to automatically learn the mapping function between problems and their corresponding equations. However, there are some limitations hindering them from being applied in real-world applications. First, although seq2seq model (Wang et al., 2017) can be applied to solve various MWPs, it suffers from fake numbers generation and mispositioned numbers generation due to all data share the same target vocabulary without problem-specific constraints. Second, some advanced methods (Wang et al., 2018b, 2019; Xie and Sun, 2019) only target at arithmetic word problems without any unknown or with one unknown that do not need to model the unknowns underlying in MWPs, which prevent them from generalizing to various MWPs, such as equation set problems. Thus, their methods can only handle arithmetic problems with no more than one unknown. Besides, they also lack an efficient equation representation mechanism to handle those MWPs with multiple unknowns and multiple equations, such as equation set problems. Finally, though some methods (Wang et al., 2017; Huang et al., 2018; Chiang and Chen, 2019) can handle multiple types of MWPs, they neither generate next symbol by taking full advantage of the generated symbols like a human nor consider the semantic transformation between equations in a problem, resulting in poor performance on the multiple-unknown MWPs, such as the MWPs involving equation set.

To address the above issues, we propose a simple yet efficient method called Universal Expression Tree (UET) to make the first attempt to represent the equations of various MWPs uniformly like the expression tree of one-unknown linear word problems with considering unknowns. Specifically, as shown in Fig. 1, UET integrates all expression trees underlying in an MWP into an ensemble expression tree via math operator symbol extension so that the grounded equations of various MWPs can be handled in a unified manner as handling one-unknown linear MWPs. Thus, it can significantly reduce the difficulty of modeling equations of various MWPs.

Then, we propose a semantically-aligned universal tree-structured solver (SAU-Solver), which is based on our UET representation and an Encoder-Decoder framework, to solve multiple types of MWPs in a unified manner with a single model. In our SAU-Solver, the encoder is designed to understand the semantics of MWPs and extract number semantic representation while the tree-structured decoder is designed to generate the next symbol based on the problem-specific target vocabulary in a semantically-aligned manner by taking full advantage of the semantic meanings of the generated expression tree like a human uses problem's contextual information and all tokens written to reason next token for solving MWPs. The problem-specific target vocabulary can help our solver to mitigate the problem of fake numbers generation as much as possible.

Besides, to further enforce the semantic constraints and rationality of the generated expression tree, we also propose a subtree-level semantically-aligned regularization to further improve subtree-level semantic representation by aligning with the contextual information of a problem, which can improve answer accuracy effectively.

Finally, to validate the universality of our solver and push the research boundary of MWPs to math real-word applications better, we introduce a new challenging **H**ybrid **M**ath **W**ord **P**roblems dataset (HMWP), consisting of one-unknown linear word problems, one-unknown non-linear word problems, and equation set problems with two unknowns. Experimental results on HWMP, ALG514, Math23K, and Dolphin18K-Manual show the universality and superiority of our approach compared with several state-of-the-art methods.

2 Related Works

Numerous methods have been proposed to attack the MWPs task, ranging from rule-based methods (Bakman, 2007; Yuhui et al., 2010), statistical machine learning methods (Kushman et al., 2014; Zhou et al., 2015; Mitra and Baral, 2016; Huang et al., 2016; Roy and Roth, 2018),semantic parsing methods (Shi et al., 2015; Koncelkedziorski et al., 2015; Huang et al., 2017), and deep learning methods (Ling et al., 2017; Wang et al., 2017, 2018b; Huang et al., 2018; Wang et al., 2018a; Xie and Sun, 2019; Wang et al., 2019). Due to space limitations, we only review some recent advances on deep leaning-based methods. (Wang et al., 2017) made the first attempt to generate expression templates using Seq2Seq model. Seq2seq method has achieved promising results, but it suffers from generating spurious numbers, predicting numbers at wrong positions, or equation duplication problem (Huang et al., 2018; Wang et al., 2018a). To address them, (Huang et al., 2018) proposed to add a copy-and-alignment mechanism to the standard Seq2Seq model. (Wang et al., 2018a) proposed equation normalization to normalize the duplicated equations by considering the uniqueness of an expression tree.

Different from seq2seq-based works, (Xie and Sun, 2019) proposed a tree-structured decoder to generate an expression tree inspired by the goal-driven problem-solving mechanism. (Wang et al., 2019) proposed a two-stage template-based solution based on a recursive neural network for math expression construction. However, they do not model the unknowns underlying in MWPs, resulting in only handling one-unknown linear word problems. Besides, they also lack an efficient mechanism to handle those MWPs with multiple unknowns and multiple equations, such as equation set problems. Therefore, their solution can not solve other types of MWPs that are more challenging due to larger search space, such as equation set problems, non-linear equation problems, etc. (Chiang and Chen, 2019) is a general equation generator that generates expression via the stack, but they did not consider the semantic transformation between equations in a problem, resulting in poor performance on the multiple-unknown MWPs, such as equation set problems.

3 The design of SAU-Solver

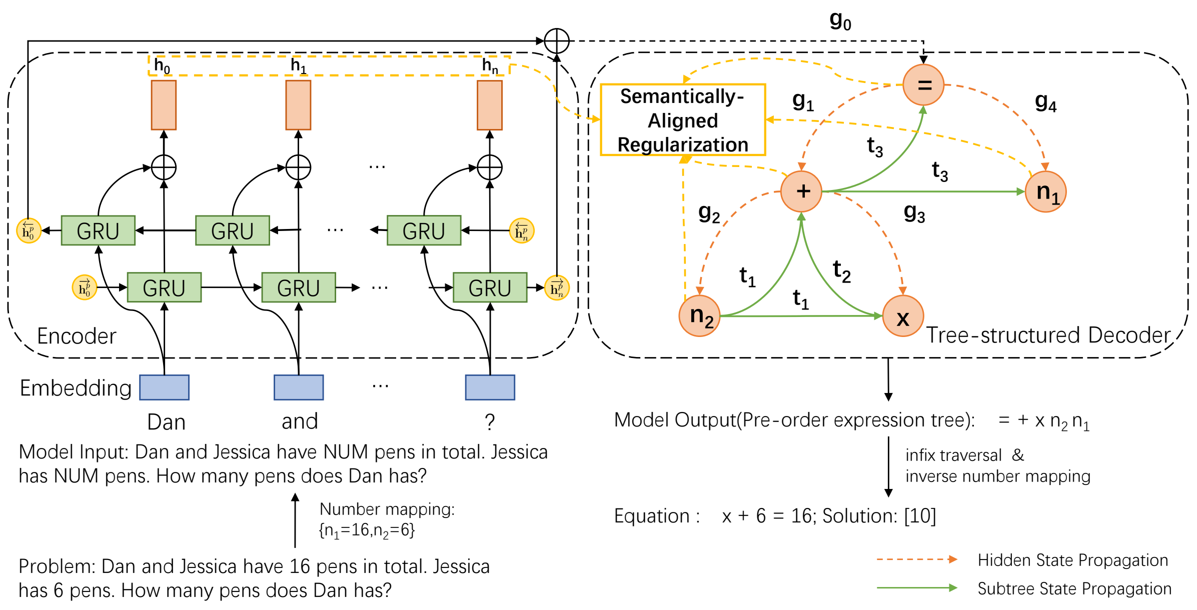
3.1 Universal Expression Tree (UET)

The primary type of textual MWPs can be divided into two groups: arithmetic word problems and equation set problems. For a universal MWPs solver, it is highly demanded to represent various equations of various MWPs in a unified manner so that the solver can generate equations efficiently. Although most of the existing works can handle one-unknown linear word problems well, it is more challenging and harder for current methods to handle the equation set MWPs with multiple unknowns well since they not only do not model the unknowns in the MWPs but also lack of an efficient equations representation mechanism to make their decoder generate required equations efficiently. To handle the above issue, an intuitive way is treating the equation set as a forest of expression trees and all trees are processed iteratively in a certain order. Although this is an effective way to handle equations set problems, it increases the difficulty of equation generation since the model needs to reason out the number of equations before starting equation generation and the prediction error will influence equation generation greatly. Besides, it is also challenging to take full advantage of the context information from the problem and the generated trees. Another way is that we can deploy Seq2Seq-based architecture to handle various equations in infix order like in previous works (Wang et al., 2017; Huang et al., 2018), but there are some limitations, such as generating invalid expression, generating spurious numbers, and generating numbers at wrong positions.

To overcome the above issues and maintain simplicity, we propose a new equation representation called Universal Expression Tree (UET) to make the first attempt to represent the equations of various MWPs uniformly. Specially, we extend the math operator symbol table by introducing a new operator ; as the lowest priority operator to integrate one or more expression trees into a universal expression tree, as shown in Fig. 1. With UET, a solver can handle the underlying equations of various textual MWPs easier in a unified manner like the way on arithmetic word problems. Although our UET is simple, it provides an efficient, concise, and uniform way to utilize the context information from the problem and treat the semantic transformation between equations as simple as treating the semantic transformation between subtrees in an equation.

3.2 SAU-Solver

Based on our proposed UET representation, we design a universal tree-structured solver to generate a universal expression tree explicitly according to the problem context and explicitly model the relationships among unknown variables, quantities, math operations, and constants in a tree-structured way, as shown in Fig. 2. Our solver consists of a Bi-GRU-based problem encoder and an explicit tree-structured equation decoder. When a problem is entered, our model first encodes each word of the problem to generate the problem's contextual representation $\mathbf{g}_0$ by our problem encoder. Then, the $\mathbf{g}_0$ will be used as the initial hidden state by our tree-structured equation decoder to guide the equation generation in prefix order with two intertwined processes: top-down tree-structured decoding and bottom-up subtree semantic transformation. With the help of top-down tree-structured decoding and bottom-up subtree semantic transformation, SAU-Solver can generate the next symbol by taking full advantage of generated symbols in a semantically-aligned manner like human solving MWPs. Finally, we apply infix traversal and inverse number mapping to generate the corresponding human-readable



g0
Semantically-Aligned Regularization
g1
g4
t3
t3
g2
g3
t1
t2
t1
Tree-structured Decoder
Encoder
Embedding
Dan
and
?
Model Input: Dan and Jessica have NUM pens in total. Jessica has NUM pens. How many pens does Dan has?
Number mapping: {n1=16,n2=6}
Problem: Dan and Jessica have 16 pens in total. Jessica has 6 pens. How many pens does Dan has?
Model Output(Pre-order expression tree): = + x n2 n1
infix traversal & inverse number mapping
Equation : x + 6 = 16; Solution: [10]
Hidden State Propagation
Subtree State Propagation

Figure 2: An overview of our SAU-Solver. When a problem preprocessed by number mapping and replacement is entered, our problem encoder encodes the problem text as context representation. Then our equation decoder generates an expression tree explicitly in pre-order traversal for the problem according to the context representation. Finally, infix traversal and inverse number mapping are applied to generate the corresponding equation.

equation that can be computed by SymPy[2], which is a python library for symbolic mathematics.

3.2.1 Problem Encoder

Bidirectional Gated Recurrent Unit (BiGRU) (Cho et al., 2014) is an efficient method to encode sequential information. Formally, given an input math word problem sentence $P = \{x_t\}_{t=1}^{n}$, we first embed each word into a vector $\mathbf{x}_t$. Then these embeddings are fed into a two-layer BiGRU from beginning to end and from end to beginning to model the problem sequence:

$$\begin{aligned}\overrightarrow{\mathbf{h}_t^p} &= GRU(\overrightarrow{\mathbf{h}_{t-1}^p}, \mathbf{x}_t)\\ \overleftarrow{\mathbf{h}_t^p} &= GRU(\overleftarrow{\mathbf{h}_{t+1}^p}, \mathbf{x}_t)\\ \mathbf{h}_t^p &= \overrightarrow{\mathbf{h}_t^p} + \overleftarrow{\mathbf{h}_t^p}\end{aligned} \tag{1}$$

where $GRU(\cdot,\cdot)$ represents the function of a two-layer GRU. $\mathbf{h}_t^p$ is the sum of the hidden states $\overrightarrow{\mathbf{h}_t^p}$ and $\overleftarrow{\mathbf{h}_t^p}$, which are from both forward and backward GRUs. These representation vectors are then fed into our tree-structured equation decoder for ensemble expression tree generation. Besides, we also construct the hidden state $\mathbf{g}_0$ as the initial hidden state of our equation decoder:

$$\mathbf{g}_0^p = \overrightarrow{\mathbf{h}_n^p} + \overleftarrow{\mathbf{h}_0^p} \tag{2}$$

where $\overrightarrow{\mathbf{h}_n^p}$ and $\overleftarrow{\mathbf{h}_0^p}$ are the hidden states of forward sequence and backward sequence respectively.

3.2.2 Equation Decoder

For decoding, inspired by previous works (Xie and Sun, 2019; Chiang and Chen, 2019), we build a semantically-aligned tree decoder to decide which symbol to generate by taking full advantage of the semantic meanings of the generated symbols with two intertwined processes: top-down tree-structured decoding and bottom-up subtree semantic transformation. Our decoder takes tree-based information $\mathbf{g}_{parent}$ (left node) or $(\mathbf{g}_{parent}, \mathbf{t}_l)$ (right node) as the input and maintains two auxiliary stacks G and T to enforce semantically-aligned decoding procedure. The stack G maintains the hidden states generated from the parent node while the stack T helps the model decide which symbol to generate by maintaining subtree semantic information of generated symbols. Benefiting from UET, our decoder can automatically end the decoding procedure without any special token. If the predicted token y_t is an operator, then we generate two children hidden states $\mathbf{g}_l$ and $\mathbf{g}_r$ according to the current node embedding $\mathbf{n}$ of y_t, and push them into the stack G to maintain the state transition among nodes and be used to predict token and its node embedding. Besides, we also push the token embedding $\mathbf{e}(y_t|P)$ of y_t into the stack T so that

[2]https://www.sympy.org/

we can maintain subtree semantic information of generated symbols after right child node generation. If the predicted token y_t is not an operator, we check the size of the stack T to judge whether the current node is a right node. If the current node is a right node, we transform the embedding of parent node op, left sibling node l and current node $\mathbf{e}(y_t|P)$ to a subtree semantic representation $\mathbf{t}$, which represents the semantic meanings of generated symbols for current subtree and is used to help the right node generation of the upper subtree. In this way, our equation decoder can decode out an equation as a human writes out an equation according to the problem description.

Token Embedding. For a problem P, its target vocabulary V^{tar} consists of 4 parts: math operators V_{op}, unknowns V_u, constants V_{con} that are those common-sense numerical values occurred in the target expression but not in the problem text (e.g. a chick has 2 legs.), and the numbers n_p occurred in P. For each token y in V^{tar}, its token embedding $\mathbf{e}(y|P)$ is defined as:

$$\mathbf{e}(y|P) = \begin{cases} \mathbf{M}_{op}(y) & \text{if } y \in V_{op} \\ \mathbf{M}_{u}(y) & \text{if } y \in V_{u} \\ \mathbf{M}_{con}(y) & \text{if } y \in V_{con} \\ \mathbf{h}^p_{loc}(y,P) & \text{if } y \in n_P \end{cases} \quad (3)$$

where $\mathbf{M}_{op}$, $\mathbf{M}_u$, and $\mathbf{M}_{con}$ are three trainable word embedding matrices independent of the specific problem. However, for a numeric value in n_P, we take the corresponding hidden state $\mathbf{h}^p_{loc}$ from encoder as its token embedding, where $loc(y,P)$ is the index position of numeric value y in P.

Gating Mechanism and Attention Mechanism. To better flow important information and ignore useless information, we apply a gating mechanism to generate node state $\mathbf{n}$ which will be used for predicting the output and generating child hidden states $\mathbf{g}_l$ and $\mathbf{g}_r$ for descendant nodes if the output of the current node is a math operator:

$$\begin{aligned} q &= \sigma\left(\mathbf{W}_q I\right) \\ Q &= \tanh\left(\mathbf{W}_Q I\right) \\ O &= q \odot Q \end{aligned} \quad (4)$$

where O can be a left node state $\mathbf{n}_l$, a right node state, a left child hidden state $\mathbf{g}_l$, or a right child hidden state $\mathbf{g}_r$. For $\mathbf{n}_l$, I is $\mathbf{g}_l$ generated by the parent node. For $\mathbf{n}_r$, I is $[\mathbf{g}_r, \mathbf{t}_l]$ which is the concatenation of the hidden state $\mathbf{g}_r$ generated by the parent node and the subtree semantic embedding $\mathbf{t}_l$ of left sibling. For $\mathbf{g}_l$ and $\mathbf{g}_r$, I is $[\mathbf{n}, \mathbf{c}, \mathbf{e}(y_t|P)]$ which is the concatenation of the current node state $\mathbf{n}$, the contextual vector $\mathbf{c}$ aggregating relevant information of the problem as a weighted representation of the input tokens by attention mechanism, and the token embedding $\mathbf{e}(y_t|P)$ of the predicted token y_t.

For better predicting a token y_t by utilizing contextual information, we deploy an attention mechanism to aggregate relevant information from the input vectors. Formally, given current node state $\mathbf{n}$ and the encoder outputs $\{\mathbf{h}^p_t\}^n_{t=1}$, we calculate the contextual vector $\mathbf{c}$ as follows:

$$\mathbf{c} = \sum_s \frac{\exp\left(\mathbf{V}_a \tanh\left(\mathbf{W}_a\left[\mathbf{n}, \mathbf{h}^p_s\right]\right)\right)}{\sum_i \exp\left(\mathbf{V}_a \tanh\left(\mathbf{W}_a\left[\mathbf{n}, \mathbf{h}^p_i\right]\right)\right)} \mathbf{h}^p_s \quad (5)$$

Based on the contextual vector $\mathbf{c}$ and current node state $\mathbf{n}$, we can predict the token y_t as follows:

$$y = \arg\max \frac{\exp(\mathbf{s}(y|\mathbf{n}, \mathbf{c}, P))}{\sum_i \exp\left(\mathbf{s}\left(y_i|\mathbf{n}, \mathbf{c}, P\right)\right)} \quad (6)$$

where

$$\mathbf{s}(y|\mathbf{n}, \mathbf{c}, P) = \mathbf{V}_n \tanh\left(\mathbf{W}_s[\mathbf{n}, \mathbf{c}, \mathbf{e}(y|P)]\right) \quad (7)$$

Subtree Semantic Transformation. Although our decoder decodes a universal expression tree in the prefix, to help our model to generate the next symbol in a semantically-aligned manner by taking full advantage of the semantic meanings of the generated expression tree, we design a recursive neural network to transform the semantic representations of the current node and its two child subtrees $\mathbf{t}_l$ and $\mathbf{t}_r$ into a high-level embedding $\mathbf{t}$ in a bottom-up manner. Formally, let t be a subtree, and y denotes the predicted token of the root node of the subtree. If y is a math operator, which means that the current subtree t must have two child subtrees $\mathbf{t}_l$ and $\mathbf{t}_r$, the high-level embedding $\mathbf{t}$ should fuse the semantic information from the operator token y, the left child subtree $\mathbf{t}_l$ and the right child subtree $\mathbf{t}_r$ as follows:

$$\begin{aligned} g_t &= \sigma\left(\mathbf{W}_{gt}\left[\mathbf{t}_l, \mathbf{t}_r, \mathbf{e}(\hat{y}|P)\right]\right) \\ C_t &= \tanh\left(\mathbf{W}_{ct}\left[\mathbf{t}_l, \mathbf{t}_r, \mathbf{e}(\hat{y}|P)\right]\right) \\ \mathbf{t} &= g_t \odot C_t \end{aligned} \quad (8)$$

Otherwise, $\mathbf{t}$ is the embedding $\mathbf{e}(y|P)$ of the predicted token y because y is a numeric value, an unknown variable, or a constant quantity and the recursion stops.

3.2.3 Semantically-Aligned Regularization

When a subtree t is produced by our model, this means that we have a computable unit. The semantics of this computable unit should be consistent with the problem text P. To achieve this goal, we propose a subtree-level semantically-aligned regularization to help train a better model with higher performance. For each subtree embedding $\mathbf{t}$ and encoder outputs $\{\mathbf{h}_1^P, \mathbf{h}_1^P, \cdots, \mathbf{h}_n^P\}$, we first apply an attention function to compute a semantically-aligned vector $\mathbf{a}$ as Equation(5), then we use a two-layer feed-forward neural network with tanh activation to transform $\mathbf{t}$ and $\mathbf{a}$ into same semantic space respectively. The procedure can be formulated as:

$$\begin{aligned} \mathbf{e}_{sa} &= \mathbf{W}_{e2} \tanh\left(\mathbf{W}_{e1}\mathbf{a}\right) \\ \mathbf{d}_{sa} &= \mathbf{W}_{d2} \tanh\left(\mathbf{W}_{d1}\mathbf{t}\right) \end{aligned} \tag{9}$$

where $\mathbf{W}_{e1}$, $\mathbf{W}_{e2}$, $\mathbf{W}_{d1}$, and $\mathbf{W}_{d2}$ are trainable parameter matrices. With the vectors $\mathbf{e}_{sa}$ and $\mathbf{d}_{sa}$ Let m be the number of subtrees in a universal expression tree, we can regularize our model by minimizing the following loss:

$$\mathcal{L}_{sa}(T|P) = \frac{1}{m}\sum_{i=1}^{m} \|\mathbf{d}_{sa} - \mathbf{e}_{sa}\|_2 \tag{10}$$

3.2.4 Training Objective

Given the training dataset **D**=$\{(P^i,T^1), (P^2,T^2), \cdots,(P^N,T^N)\}$, where T^i is the universal expression tree of problem P^i, we minimize the following loss function:

$$\mathcal{L}(T|P) = \sum_{(P,T)\in\mathbf{D}} [-\log p(T|P) + \lambda * \mathcal{L}_{sa}(T|P)] \tag{11}$$

where

$$p(T|P) = \prod_{t=1}^{m} \text{prob}(y_t|\mathbf{g}_t, \mathbf{c}_t, P) \tag{12}$$

where m denotes the size of T, and $\mathbf{g}_t$ and $\mathbf{c}_t$ are the hidden state vector and its contextual vector at the t-th node. We set λ as 0.01 empirically.

3.3 Discussion

The methods most relevant to our method are GTS (Xie and Sun, 2019) and StackDecoder (Chiang and Chen, 2019). However, compared with them, our method is different from them as follows. First, our method applies a universal expression tree to represent the diverse equations underlying different MWPs uniformly, which match real-word MWPs better than GTS and StackDecoder which either can only handle single-var linear MWPs without considering unknowns or can handle equations set problem iteratively. Second, we introduce subtree-level semantically-aligned regularization for better enforcing the semantic constraints and rationality of generated expression tree during training, leading to higher answer accuracy, as illustrated in Table 2.

4 Hybrid Math Word Problem Dataset

Most public datasets for automatic MWPs solving either are quite small such as Alg514 (Kushman et al., 2014), DRAW-1K (Upadhyay and Chang, 2017), MaWPS (Koncel-Kedziorski et al., 2016) or exist some incorrect labels such as Dolphin18K (Huang et al., 2016). An exception is the Math23K dataset which contains 23161 problems labeled well with structured equations and answers. However, it only contains one-unknown linear MWPs, which is not sufficient to validate the ability of a math solver about solving multiple types of MWPs. Therefore, we introduce a new high-quality MWPs dataset, called HMWP, in which each sample is extracted from a Chinese K12 math word problem bank, to validate the universality of math word problem solvers and push the research boundary of MWPs to match real-world scenes better. Our dataset contains three types of MWPs: arithmetic word problems, equations set problems, and non-linear equation problems. There are 5491 MWPs, including 2955 one-unknown-variable linear MWPs, 1636 two-unknown-variable linear MWPs, and 900 one-unknown-variable non-linear MWPs. It should be noticed that our dataset is sufficient for validating the universality of math word problem solvers since these problems can cover most cases about MWPs. We labeled our data with structured equations and answers as Math23K (Wang et al., 2017). The data statistics of our dataset and several publicly available datasets are shown in Table 1. From the statistics, we can see that the #AVG EL (average equation length), #Avg PN (average number of quantities occurred in problems and their corresponding equations), and #Avg Ops (average numbers of operators in equations) are the largest among the serval publicly available datasets. (Xie and Sun, 2019) showed the higher these values, the more difficult it is. Therefore, our dataset is more challenging for MWPs solvers.

Dataset	# Problems	# Templates	# Sentences	# Words	# Avg EL	# Avg SNI	# Avg Constants	# Avg Ops	Problem types
Alg514	514	28	1.62k	19.3k	9.67	3.54	0.44	5.69	algebra, linear
Dolphin1878	1,878	1,183	3.30k	41.4k	8.18	2.58	0.63	4.97	linear + nonlinear
DRAW-1K	1,000	230	6.23k	81.5k	9.985	3.386	0.747	5.852	algebra, linear
MaWPS	2373	-	2373	73.3k	4.55	2.31	0.26	1.78	algebra, linear
Math23K	23,161	2,187	70.1k	822k	5.55	3.0	0.28	2.28	algebra, linear
Dolphin18k	18,460	5,871	49.9k	604k	9.19	3.15	1.09	4.96	linear + nonlinear
HMWP	5470	2779	9.56k	342k	**10.73**	3.42	**1.35**	**5.96**	linear + nonlinear

Table 1: Statistics of our dataset and several publicly available datasets. Avg EL, Avg SNI, Avg Constants, and Avg Ops represent average equation length, average number of quantities occurred in problems and their corresponding equations, average numbers of constants only occurred in equations, and average numbers of operators in equations, respectively. The higher these values, the more difficult it is. This has been shown in (Xie and Sun, 2019).

5 Experiments

5.1 Experimental Setup and Training Details

Datasets, Baselines, and Evaluation metric. We conduct experiments on four datasets, such as HMWP, Alg514 (Kushman et al., 2014), Math23K (Wang et al., 2017) and Dolphin18K-Manual (Huang et al., 2016). The data statistics of four datasets are shown in Table 1. The main state-of-the-art learning-based methods to be compared are as follows: **Seq2Seq-attn w/ SNI** (Wang et al., 2017) is a universal solver based on the seq2seq model with significant number identification(SNI). **GTS** (Xie and Sun, 2019) is a goal-driven tree-structured MWP solver only for one-unknown-variable non-linear MWPs. **StackDecoder** (Chiang and Chen, 2019) is a semantically-aligned MWPs solver. **SAU-Solver w/o SSAR** and **SAU-Solver** are two universal tree-structured solvers proposed in this paper without and with subtree semantically-aligned regularization. Following our baselines, we use *answer accuracy* as the evaluation metric: if the calculated value of the predicted expression tree equals to the true answer, it is thought of correct since the predicted expression is equivalent to the target expression.

Model	HMWP	ALG514	Math23K	Dolphin18K Manual
Seq2Seq-attn w/ SNI	23.2%	16.1%	58.1%	5.9%
GTS	-	-	73.9%	-
StackDecoder	27.4%	28.86%	66.0%	9.8%
SAU-Solver w/o SSAR (ours)	44.40%	55.44%	74.53%	11.02%
SAU-Solver (ours)	**44.83%**	**57.39%**	**74.84%**	**11.41%**

Table 2: Model comparison on answer accuracy via 5-fold cross-validation. “-” means either the code is not released or the model is not suitable on those datasets.

Implementation Details. We use PyTorch[3] to implement our model on Linux with NVIDIA RTX 2080Ti. All the words with less than five occurrences are converted into a special token UNK. We set the dimensionality of word embedding and the size of all hidden states for other layers as 128 and 512, respectively. But for HMWP and Dolphin18K-Manual, we set the size of all hidden states for other layers as 384 since the memory consumption exceeds the capacity of NVIDIA RTX 2080Ti. Our model is trained by ADAM optimizor (Kingma and Ba, 2015) with β_1 = 0.9, β_2 =0.999, and $\epsilon = 10^{-8}$. The mini-batch size is set to 32. The initial learning rate is set to 10^{-3} and then decreases to half every 20 epochs. To prevent overfitting, we set the dropout probability as 0.5 and weight decay as $1e^{-5}$. Finally, we set beam size as 5 in beam search to generate expression trees.

[3]http://pytorch.org

	linear (One-VAR)	linear (Two-VAR)	non-linear (One-VAR)	All
# Num	1944	1614	1912	5470
# Avg EL	10.50	12.10	9.83	10.73
# Avg SNI	3.59	3.59	3.12	3.42
# Avg Constants	1.21	1.41	1.45	1.35
# Avg Ops	5.70	7.10	5.26	5.96
Correct number (Retrieval-Jaccard)	222	348	618	1188
Accuracy (Retrieval-Jaccard)	11.42%	21.56%	32.32%	21.72%
Correct number (Seq2Seq-attn w/SNI)	244	312	711	1267
Accuracy (Seq2Seq-attn w/SNI)	12.55%	19.33%	37.19%	23.2%
Correct number (SAU-Solver (ours))	**593**	**673**	**1186**	**2452**
Accuracy (SAU-Solver (ours))	**30.50%**	**41.70%**	**62.03%**	**44.83%**

Table 3: The data statistics and performance on different subset of HMWP.

5.2 Results and Analyses

Answer Accuracy. We conduct 5-fold cross-validation to evaluate the performances of baselines and our models on all four datasets. The results are shown in Table 2. Several observations can be made from the results in Table 2 as follows:

First, our SAU-Solver has achieved significantly better than the baselines on four datasets. It proves that our model is feasible for solving multiple types

Case 1: 鸡兔同笼， 上数 有 NUM(n_0 [20]) 个头， 下数 有 NUM(n_1 [50]) 条腿， 可知 鸡 数量 为 多少？ (An unknown number of rabbits and chickens were locked in a cage, counting from the top, there were NUM(n_0 [20]) heads, counting from the bottom, there were NUM(n_1 [50]) feet. How many chickens were locked in this cage?)		
Seq2Seq: (x-n_1)/n_0=(x-n_1)/n_2; (**error**)	**SAU-Solver w/o SSAR:** n_0+x+4.0*x=n_1; (**correct**)	**SAU-Solver:** 2.0*x+4.0*(n_0-x)=n_1; (**correct**)
Case 2: NUM(n_0 [1]) 艘 轮船 航行 于 A 、 B NUM(n_1 [2]) 个 码头 之间， 顺水 需 NUM(n_2 [5]) 小时， 逆水 需 NUM(n_3 [7]) 小时， 已知 水流 速度 为 每 小时 NUM (n_4 [5]) 千 米， 则 A 、 B 之间 距离 为 多少 千 米？ (NUM(n_0 [1]) boat sailing between NUM(n_1 [2]) docks, it takes NUM(n_2 [5]) hours to sail from A to B downstream, while NUM(n_3 [7]) hours sailing upstream. Knowing the velocity of the water flow is 5 km/h, what is the distance between A and B?)		
Seq2Seq: x/(n_2+n_1)+n_1=x-/n_2; (**error**)	**SAU-Solver w/o SSAR:** x/n_2-n_4=x/n_3+n_4; (**correct**)	**SAU-Solver:** x/n_2-n_4=x/n_3+n_4; (**correct**)
Case 3: 整理 NUM(n_0 [1]) 批 图书， 如果 由 NUM(n_1 [1]) 个 人 单独 做，要 花 NUM(n_2 [60]) 小时． 现在 由 一部分 人 用 NUM(N_3 [1]) 小时 整理，随后 增加 NUM(n_4[15]) 人 和 他们 一起 又 做 了 NUM(n_5 [2]) 小时，恰好 完成 整理 工作． 假设 每个 人 的 工作效率 相同， 那么 先 安排 整理 的 人员 有 多少 人？ (Given NUM(n_0 [1]) stack of books, NUM(n_1 [1]) student can sort them in NUM(n_2 [60]) hours. In the first NUM(N_3 [1]) hours, there were several students sorting books, later, NUM(n_4[15]) more students joined them, and they finished the job in another NUM(n_5 [2]) hours together. If each student is as efficient as the others, how many students were working at the beginning?		
Seq2Seq: n_1*(x/n_2)+n_5*(x+n_4)/n_2=1.0; (**error**)	**SAU-Solver w/o SSAR:** x/n_2+n_5*(x+n_4)/n_2=1.0; (**correct**)	**SAU-Solver:** x/n_2+n_5*(x+n_4)/n_2=1.0; (**correct**)
Case 4: 某 农场 老板 准备 建造 NUM(n_0 [1]) 个 矩形 羊圈， 他 打算 让 矩形 羊圈 的 NUM(n_1 [1]) 面 完全 靠 墙， 墙 可 利用 的 长度 为 NUM(n_2 [25]) m， 另外 NUM(n_3 [1]) 面 用 长度 为 NUM(n_4 [50]) m 的 篱笆 围成 (篱笆 正好 要 全部 用 完， 且 不 考虑 接头 的 部分)， 若要 使 矩形 羊圈 的 面积 为 NUM(n_5 [300]) m ^ NUM(n_6 [2])， 求 垂直于 墙 的 边 长． (A farm owner plans to build a rectangle sheepfold, with NUM(n_1 [1]) side against the wall. The wall is 25 meters long, and he used NUM(n_3 [1]) NUM(n_4 [50])-meter-long fence to build the rest of the sheepfold (the fence should be exactly used up, neglecting the joining part). If the area of the sheepfold is NUM(n_5 [300]) m ^ NUM(n_6 [2]), find the length of the side vertical to the wall.		
Seq2Seq: x*(n_3-2.0*x)=n_4; (**error**)	**SAU-Solver w/o SSAR:** (n_2-2.0*x)*(n_4-2.0*x)= n_5; (**error**)	**SAU-Solver:** x*(n_4-2.0*x)= n_5; (**correct**)

Table 4: Typical cases. Note that the results are represented as infix traversal of expression trees which is more readable than prefix traversal.

of MWPs. It also proves that our model is more general and more effective than other state-of-the-art models on the real-word scenario that need to solve multiple types of MWPs with a unified solver.

Second, with our subtree-level semantically-aligned regularization on training procedure, our SAU-Solver has gained additional absolute 0.43% accuracy on HMWP, absolute 1.95% accuracy on ALG514, absolute 0.31% accuracy on Math23k, and absolute 0.39% accuracy on Dolphin18k-Manual. This shows that subtree-level semantically-aligned regularization is helpful for improving subtree semantic embedding, resulting in improving expression tree generation, especially for the generation of the right child node. Although StackDecoder can be a universal math word problem solver via simple operator extension, the performances on HMWP, ALG514, and Dolphin18k-Manual are very poor, since it generates expression trees independently and only considers the semantic-aligned transformation in an expression tree. Different from it, our SAU-Solver generates multiple expression trees as a universal expression tree and conducts subtree-level semantic-aligned transformation for subsequent tree node generation in our universal expression tree. In this way, we can deliver the semantic information of the previous expression tree to help the generation of the current expression tree. Therefore we can achieve better performance than StackDecoder.

Overall, our model is more general and effective than other state-of-the-art models on multiple MWPs and outperforms the compared state-of-the-art models by a large margin on answer accuracy.

Performance on different types of MWPs. We drill down to analyse the performance of **Retrieval-Jaccard**, **Seq2seq-attn w/SNI**, and **SAU-Solver** on different types of MWPs in HMWP. The data statistics and performance results are shown in Table 3. We can observe that our model outperforms the other two models by a large margin on all subsets. Intuitively, the longer the expression length is, the more complex the mathematical relationship of the problem is, and the more difficult it is. And the average expression length of our dataset is much longer than Math23K according to the data statistics of Table 3 and Table 1. Therefore, we can observe that the accuracy of our model on linear (One-VAR) is lower than Math23K in Table 2.

Expression Tree Sizes	Math23K			HMWP		
	Correct	Error	Acc(%)	Correct	Error	Acc(%)
3-	729	168	81.27%	0	0	0%
5	1872	435	81.14%	3	1	75.00%
7	620	291	68.06%	32	25	56.14%
9	147	143	50.69%	159	69	69.74%
11	66	74	47.14%	102	111	47.89%
13+	20	66	23.26%	197	395	33.28%

Table 5: Accuracy of different expression tree size.

5.3 Error Analysis

In Table 5, we show the results of how the accuracy changes as the expression tree size becomes larger. We can observe that as the expression tree size becomes larger, our model's performance becomes lower. This shows that although our model can handle various equations in a unified manner, it still has limitations at predicting long equations since longer equations often match with more complex MWPs which are more difficult to solve. Thus, our model still has room for improvement in reasoning, inference, and semantic understanding. Besides, compared with performances on Math23K which has only a few examples with complex templates, our model achieves significant improvement on the subset of HMWP with expression tree size 13+. This shows that constructing datasets with abundant complex examples can improve the model's ability to handle complex problems.

5.4 Case Study

Further, we conduct a case analysis and provide four cases in Table 4, which shows the effectiveness of our approach. Our analyses are summarized as follows. From **Case 1**, Seq2Seq generates a spurious number n_2 not in problem text while both SAU-Solver w/o SSAR and SAU-Solver predict correctly owning to the problem-specific target vocabulary. Besides, although both SAU-Solver w/o SSAR and SAU-Solver can generate correct an equation, the equation generated by our SAU-Solver is more semantically-aligned with a human than the equation generated by SAU-Solver. From **Case 2**, we can see that Seq2Seq generates an invalid expression containing consecutive operators while our models can guarantee the validity of expressions since they generate expression trees directly. From **Case 3**, we find it interesting that tree-based models can avoid generating redundant operations, such as "n_1*". From **Case 4**, we can see that SAU-Solver can prevent generating the similar subtree as its left sibling when the parent node is "*".

6 Conclusion

We propose an SAU-Solver, which is able to solve multiple types of MWPs, to generate the universal express tree explicitly in a semantically-aligned manner. Besides, we also propose a subtree-level semantically-aligned regularization to improve subtree semantic representation. Finally, we introduce a new MWPs datasets, called HMWP, to validate our solver's universality and push the research boundary of MWPs to math real-world applications better. Experimental results show the superiority of our approach.

Acknowledgements

We thank all anonymous reviewers for their constructive comments. This work was supported in part by National Key RD Program of China under Grant No. 2018AAA0100300, National Natural Science Foundation of China (NSFC) under Grant No.U19A2073 and No.61976233, Guangdong Province Basic and Applied Basic Research (Regional Joint Fund-Key) Grant No.2019B1515120039, Nature Science Foundation of Shenzhen Under Grant No. 2019191361, Zhijiang Lab's Open Fund (No. 2020AA3AB14), Sichuan Science and Technology Program (No. 2019YJ0190).

References

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. *Computing Research Repository*, arXiv:math/0701393.

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. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics.

Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. 2018. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 213–223. Association for Computational Linguistics.

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to solve math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 805–814. Association for Computational Linguistics.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. How well do computers solve math word problems? large-scale dataset construction and evaluation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 887–896. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *international conference on learning representations*.

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.

Rik Koncelkedziorski, 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.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 271–281.

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. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Subhro Roy and Dan Roth. 2018. Mapping to declarative knowledge for word problem solving. *Transactions of the Association for Computational Linguistics*, 6:159–172.

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. Association for Computational Linguistics.

Shyam Upadhyay and Mingwei Chang. 2017. Annotating derivations: A new evaluation strategy and dataset for algebra word problems. In *15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 494–504.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. 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. Association for Computational Linguistics.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018b. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5545–5552.

Lei Wang, Dongxiang Zhang, Zhang Jipeng, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Thirty-Third AAAI Conference on Artificial Intelligence*, pages 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. Association for Computational Linguistics.

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, IJCAI-19*, pages 5299–5305. International Joint Conferences on Artificial Intelligence Organization.

Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang Ronghuai. 2010. Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems. In *International Workshop on Education Technology and Computer Science*, volume 2, pages 476–479.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 817–822. Association for Computational Linguistics.