A Knowledge-Aware Sequence-to-Tree Network for Math Word Problem Solving

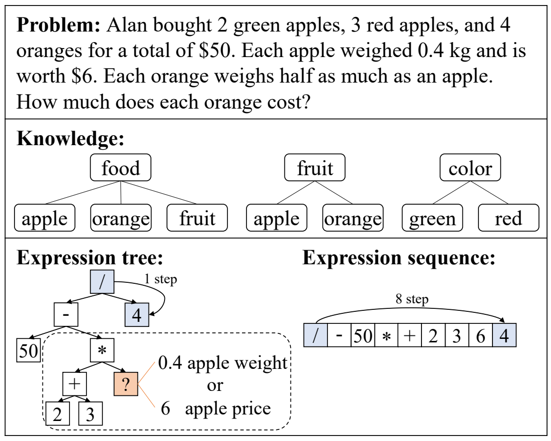
Qinzhuo Wu, Qi Zhang[*], Jinlan Fu, Xuanjing Huang
Shanghai Key Laboratory of Intelligent Information Processing,
School of Computer Science, Fudan University, Shanghai, China
(qzwu17,qz,fujl16,xjhuang)@fudan.edu.cn

Abstract

With the advancements in natural language processing tasks, math word problem solving has received increasing attention. Previous methods have achieved promising results but ignore background common-sense knowledge not directly provided by the problem. In addition, during generation, they focus on local features while neglecting global information. To incorporate external knowledge and global expression information, we propose a novel knowledge-aware sequence-to-tree (KA-S2T) network in which the entities in the problem sequences and their categories are modeled as an entity graph. Based on this entity graph, a graph attention network is used to capture knowledge-aware problem representations. Further, we use a tree-structured decoder with a state aggregation mechanism to capture the long-distance dependency and global expression information. Experimental results on the Math23K dataset revealed that the KA-S2T model can achieve better performance than previously reported best results.

1 Introduction

Math word problem solving has attracted increasing attention, and many math word problem solving systems have been developed. Early statistical learning methods (Feigenbaum et al., 1963; Fletcher, 1985; Bakman, 2007; Roy and Roth, 2016) extracted templates or features from problems and generated corresponding math expressions based on these templates or features. These methods require a large number of manually formulated features or can only be applied to small application problems in small areas. In recent years, many methods (Wang et al., 2017, 2018b; Xie and Sun, 2019) have been developed that apply neural networks to analyze math word problems, with promising results. These methods use end-to-end models to directly generate the corresponding math expressions from the problem text.



Problem: Alan bought 2 green apples, 3 red apples, and 4 oranges for a total of $50. Each apple weighed 0.4 kg and is worth $6. Each orange weighs half as much as an apple. How much does each orange cost?

Figure 1: An example of a math word problem. With external knowledge, a model can capture the relationships between the entities in the problem. With the global information of a generated expression tree, a model can capture information between long-distance nodes.

Although previous methods have achieved promising results, several problems remain that need to be addressed: 1) Background knowledge and common sense should be incorporated. For example, as shown in Figure 1, both apples and oranges are fruit. Humans are naturally aware of this common-sense information, but it is difficult for the model to learn this information from the problem text alone. 2) When generating expressions, sequence-to-sequence (Seq2Seq) methods tend to focus on local features and ignore global information. For example, the root node "/" of the expression tree in Figure 1 is directly adjacent to its right child "4", but they are eight steps apart in the pre-order expression sequence. Xie and Sun (2019) proposed a sequence-to-tree (Seq2Tree) method for generating an expression

[*] Corresponding author.

tree in pre-order based on the parent node and the left sibling tree of each node. However, global information is still not being considered in the generated expression tree.

To overcome these problems, we propose a novel knowledge-aware sequence-to-tree (KA-S2T) method for exploring how to better utilize external knowledge and capture the global expression information. The proposed model connects related entities and categories based on external knowledge bases to capture common-sense information and obtain better interaction between words. In addition, we designed a tree-structured decoder to capture the long-distance dependency and global expression information. KA-S2T updates all nodes in the generated expression at each time step, whereby the node state is updated by a recursive aggregation of its neighboring nodes. Through multiple iterations of aggregation, the model can use global information associated with the generated expression to generate the next node and thereby achieve better predictions.

The main contributions of this paper can be summarized as follows:

- We incorporate common-sense knowledge from external knowledge bases into math word problem solving tasks.
- We propose a tree-structured decoder for generating math expressions. To incorporate global information associated with generated partial expressions, we recursively aggregate the neighbors of each node in the expression at each time step.
- We conducted experiments on the Math23k dataset to verify the effectiveness of our KA-S2T model, and the results show that our model achieved better performance than previous methods.

2 Models

In this section, we define the problem and present our proposed KA-S2T model. As shown in Figure 2, we first use a bidirectional long short-term memory (LSTM) network to encode the math word problem sequences (Section 2.2). Then, we construct an entity graph based on external knowledge to model the relationships between different entities and categories in the problem (Section 2.3), and use a two-layer graph attention network (GAT) to obtain knowledge-aware problem representations (Section 2.4). Finally, we used a tree-structured decoder with a state aggregation mechanism to generate pre-order traversal math expression trees (Section 2.5).

2.1 Problem Definition

Consider the input sequence of a math word problem $\mathrm{X} = (x_1, x_2, \ldots, x_n)$. Our goal is to train a model that can generate its math expression $\mathrm{Y} = (y_1, y_2, \ldots, y_{n'})$. The task is to estimate a probability distribution in which $\mathbf{P}(\mathrm{Y}|\mathrm{X}) = \prod_{t=1}^{n'} \mathbf{P}(y_t|y_{<t}, \mathrm{X})$. Here, words generated in the math expression Y are either drawn from the input math word problem X, or a vocabulary V. Y, the pre-order sequence of a math expression tree, is executed to produce the answer to the problem X.

2.2 Bidirectional LSTM Encoder

The encoder transforms the words in math word problems into vector representations. We used a bidirectional LSTM (BiLSTM) (Hochreiter and Schmidhuber, 1997) network to encode each word x_i into a vector representation $\mathbf{h_i^{seq}}$:

$$\mathbf{h_i^{seq}} = \mathrm{BiLSTM}(\mathbf{e}(x_i), \mathbf{h_{i-1}^{seq}}) \in \mathrm{R}^{n \times 2d}, \quad (1)$$

where n and d are the size of the input sequence X and the BiLSTM hidden state, respectively. $\mathbf{e}(x_i)$ is the word embedding for word x_i in the problem. $\overrightarrow{\mathbf{h_i^{seq}}}$ and $\overleftarrow{\mathbf{h_i^{seq}}}$ are the BiLSTM hidden states generated by reading X in the forward and backward order, respectively. We define the final vector representation $\mathbf{h_i^{seq}}$ as the concatenation of the forward and backward hidden states, i.e., $\mathbf{h_i^{seq}} = [\overrightarrow{\mathbf{h_i^{seq}}} : \overleftarrow{\mathbf{h_i^{seq}}}]$.

2.3 Constructing Entity Graphs with External Knowledge

Each math word problem corresponds to an entity graph $\mathrm{G} = (\mathrm{N}, \mathrm{A})$, where N is a node list and A is the adjacent matrix of these nodes. The graphs are retrieved from external knowledge bases, with the words in the math word problem as nodes. If multiple words in X belong to the same category c in the knowledge base, we set category c as a node in the graph G and connect these words with their categories. For example, both "green" and "red" belong to the category "color".

To incorporate knowledge about phrases, if several phrases in X are combined with words belonging to the same category and the same words

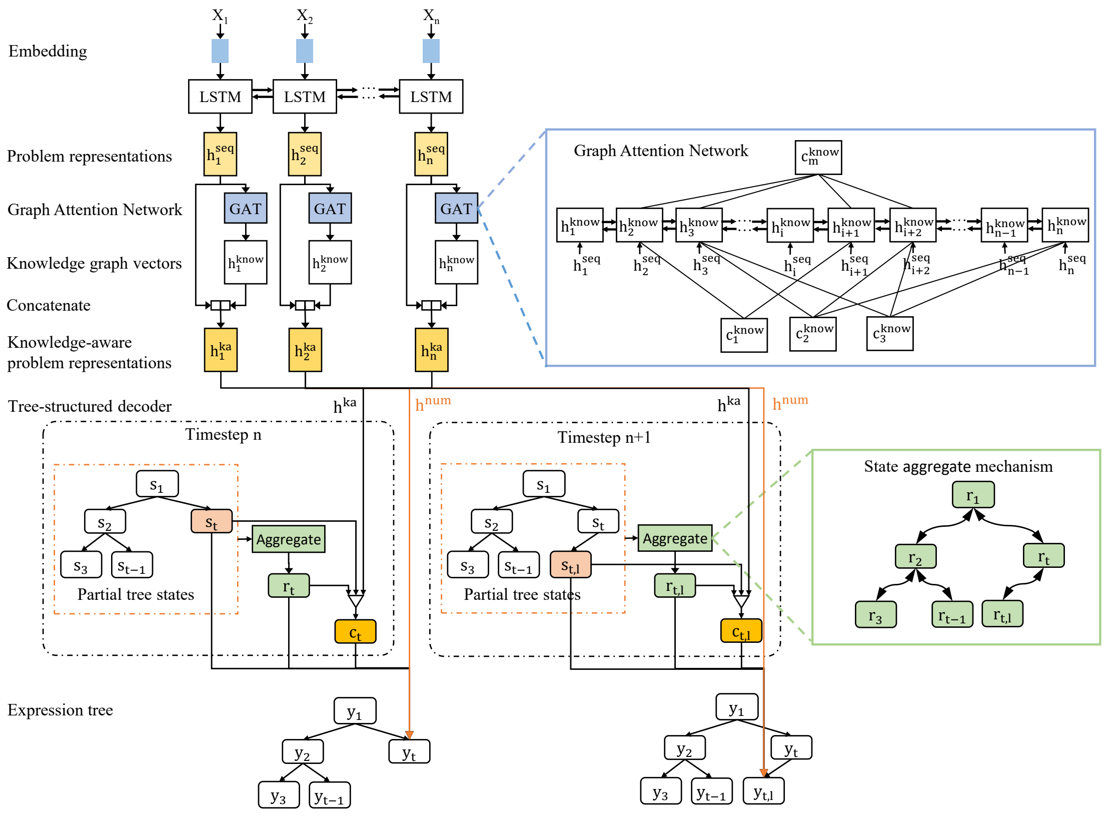


Figure 2: Main structure of our proposed KA-S2T model. At the top side of this figure shows an encoder consisting of a bidirectional LSTM network and a knowledge-aware graph attention network (see Section 2.2 and Section 2.4 for more details). The red line h_{num} indicates the representation of numbers in the problem, which are identified as $\{N1, N2, N3, \ldots\}$ according to their positions in the problem. Instead of generating these numbers directly from the output vocabulary, KA-S2T generates position identifiers that copy the numbers from the problem. The bottom of the figure shows a tree-structured decoder. At each time step, this decoder generates a current node state based on its parent node. Then, the decoder uses a state aggregation mechanism to obtain the context state r_t for each node in the partial expression tree, and generates context vector c_t based on encoder's hidden states. See Section 2.5 for more details.

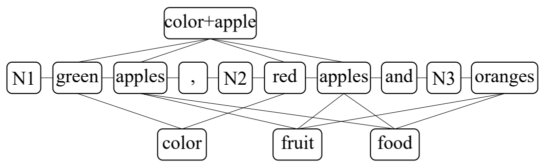


Figure 3: An example of an entity graph

in order, then we build a phrase category c' for these phrases and set c' as a node. As shown in Figure 3, "green apples" and "red apples" are combined in the same category words "green, red" and by the same word "apple". We build a phrase category "color+apple" for these two phrases. Then we connect this phrase category c' to the first and last words of its related phrase.

With n words from the problem and m categories from the knowledge base, an entity graph has a node list $\mathrm{N} = \{x_1, x_2, \ldots, x_n, c_1, \ldots, c_m\}$ with $n + m$ nodes.

2.4 Knowledge-aware Problem Representations

For an entity graph, we initialize category c with the average of the vector representations of words adjacent to c. For example, for the category c_1 "color" adjacent to the word x_2 "green" and the word x_6 "red", we initialize c_1 as $\mathbf{c_1} = \mathrm{avg}(\mathbf{h_2^{seq}}, \mathbf{h_6^{seq}})$. In this way, for the nodes in this entity graph, we have node initial vectors $\mathbf{h^{seq'}} = \{\mathbf{h_1^{seq}}, \ldots, \mathbf{h_n^{seq}}, \mathbf{c_1}, \ldots, \mathbf{c_m}\}$. Then, we use a two-layer GAT (Veličković et al., 2017) to obtain the hidden vectors $\mathbf{h^{know'}}$ of these nodes. The GAT functions are given as follows:

$$\mathbf{h_i^{know'}} = \underset{k=1,\ldots,K}{||} \sigma\Big(\sum_{\mathrm{A_{ij}}=1} \alpha_{ij} W_k \mathbf{h_j^{seq'}}\Big), \quad (2)$$

$$\alpha_{ij} = \frac{\exp(\mathrm{LRelu}(\mathrm{w}_\mathrm{s}^\mathrm{T}[\mathrm{W}_\mathrm{h}\mathbf{h}_\mathbf{i}^{\mathbf{seq}'}||\mathrm{W}_\mathrm{h}\mathbf{h}_\mathbf{j}^{\mathbf{seq}'}]))}{\sum\limits_{\mathrm{A}_\mathrm{ij}=1}\exp(\mathrm{LRelu}(\mathrm{w}_\mathrm{s}^\mathrm{T}[\mathrm{W}_\mathrm{h}\mathbf{h}_\mathbf{i}^{\mathbf{seq}'}||\mathrm{W}_\mathrm{h}\mathbf{h}_\mathbf{j}^{\mathbf{seq}'}]))},$$

where w_s^T, $\mathrm{W_h}$, $\mathrm{W_k}$ are trainable weight vector and matrices. $||$ is concatenation operation and LRelu is a LeakyRelu activation function (Xu et al., 2015). K is the number of heads in GAT and $\mathrm{A_{ij}} = 1$ means there are edge between the i-th and j-th node.

To represent n words in problem X, we simply select the first n vectors of $\mathbf{h}^{\mathbf{know}'} \in \mathbf{R}^{(n+m)\times 2d}$ as the knowledge graph vectors.

$$\mathbf{h}^{\mathbf{know}} = \mathbf{h}^{\mathbf{know}'}[0 : n]. \tag{3}$$

After concatenating the word vector $\mathbf{h}^{\mathbf{seq}}$ and the knowledge graph vectors $\mathbf{h}^{\mathbf{know}}$, the knowledge-aware problem representation $\mathbf{h}^{\mathbf{ka}}$ is obtained and fed to the tree-structured decoder.

$$\mathbf{h}^{\mathbf{ka}} = [\mathbf{h}^{\mathbf{seq}} : \mathbf{h}^{\mathbf{know}}]. \tag{4}$$

If there are n_{num} numbers in X, we may want to copy them directly from problem X rather than generating them from the vocabulary. We extract $\mathbf{h}^{\mathbf{num}} \in \mathrm{R}^{n_{num}\times 2d}$ from $\mathbf{h}^{\mathbf{ka}}$ based on the position of these numbers. $\mathbf{h}_\mathbf{i}^{\mathbf{num}}$ is the representation of the i-th number in the problem. We use $\mathbf{h}^{\mathbf{num}}$ to compute the distribution score of these numbers in Equation 6.

2.5 Tree-structured Decoder

In the KA-S2T tree-structured decoder, we generate pre-order expressions Y from top to bottom. At time step t, if the y_t we generate is an operator, this means it is an internal node and its left child $y_{t,l}$ and right child $y_{t,r}$ must still be generated. If y_t is a number, it is a leaf node. Once the children of all the internal nodes are generated, then the output expression sequence Y can be transformed into a complete executable expression tree.

The tree-structured decoder has three roles: 1) it attentively reads the knowledge-aware problem representation to obtain a context vector, and uses this vector to update the decoder's state; 2) it recursively aggregates the neighbors of each node in the generated partial expression to capture global information; and 3) it adaptively chooses a word from the vocabulary or copies a number from the math word problem for generation.

The decoder updates its state as follows:

$$\begin{aligned}\mathbf{s_{t,l}} &= \sigma(\mathrm{W_{left}}[\mathbf{s_t} : \mathbf{c_t} : \mathbf{r_t} : (\mathbf{e}(y_t)]),\\ \mathbf{s_{t,r}} &= \sigma(\mathrm{W_{right}}[\mathbf{s_t} : \mathbf{c_t} : \mathbf{r_t} : (\mathbf{e}(y_t)]),\end{aligned} \tag{5}$$

where $\mathrm{W_{left}}$ and $\mathrm{W_{right}}$ are the weight matrices and σ is a sigmoid function. $\mathbf{e}(y_t)$ is the embedding of t-th generated word y_t. $\mathbf{s_{t,l}}$ and $\mathbf{s_{t,r}}$ is the left child state and right child state of $\mathbf{s_t}$, respectively. For the root node y_1, it initializes $\mathbf{s_1}$ with the max pooling of $\mathbf{h}^{\mathbf{ka}}$. $\mathbf{c_t}$ is the context vector for the hidden states of the encoder. $\mathbf{r_t}$ is the context state for the partial expression generated at previous time steps.

Finally, the tree-structured decoder generates a word from vocabulary V or copies a number from the math word problem X with the following distributions:

$$\begin{aligned}\mathbf{P}_{\mathrm{gen}}(y_t) &= \mathrm{softmax}(\mathrm{W_g}[\mathbf{s_t}:\mathbf{c_t}:\mathbf{r_t}])\\ \mathbf{P}_{\mathrm{copy}}(y_t) &= \mathrm{softmax}(\mathrm{W_p}[\mathbf{s_t}:\mathbf{c_t}:\mathbf{r_t}:\mathbf{h}^{\mathbf{num}}])\\ \beta_t &= \sigma(\mathrm{W_z}[\mathbf{s_t}:\mathbf{c_t}:\mathbf{r_t}:\mathbf{h}^{\mathbf{num}}]),\\ \mathbf{P}(y_t|y_{<t},\mathrm{X}) &= \begin{cases}(1-\beta_t)\mathbf{P}_{\mathrm{gen}}(y_t)\\ \quad\beta_t\,\mathbf{P}_{\mathrm{copy}}(y_t)\end{cases},\end{aligned} \tag{6}$$

where $\mathrm{W_g}$, $\mathrm{W_p}$ are weight matrices. $\beta_t \in [0, 1]$ is a gate value to determine whether generate a word from vocabulary or copy a number from math word problem. $y_{<t}$ represents the words generated at earlier timesteps. The final distribution $\mathbf{P}(y_t|y_{<t},\mathrm{X})$ is a concatenation of generate distribution $\mathbf{P}_{\mathrm{gen}}(y_t)$ and copy distribution $\mathbf{P}_{\mathrm{copy}}(y_t)$.

Attention. We use attention mechanism (Bahdanau et al., 2014) to compute the context vector $\mathbf{c_t}$. Given the decoder state $\mathbf{s_t}$ and the expression context state $\mathbf{r_t}$, it first attends on the encoder's problem representation $\mathbf{h}^{\mathbf{ka}}$ to obtain $\mathbf{c_t}$, which is defined as below:

$$\begin{aligned}\alpha_{ti} &= \frac{\exp(\mathrm{W_e}\tanh(\mathrm{W_m}\mathbf{h}_\mathbf{i}^{\mathbf{ka}}+\mathrm{W_s}\mathbf{s_t}+\mathrm{W_r}\mathbf{r_t}))}{\sum_{j=1}^{n}\exp(\mathrm{W_e}\tanh(\mathrm{W_m}\mathbf{h}_\mathbf{j}^{\mathbf{ka}}+\mathrm{W_s}\mathbf{s_t}+\mathrm{W_r}\mathbf{r_t}))},\\ \mathbf{c_t} &= \sum_{i=1}^{n}\alpha_{ti}\mathbf{h}_\mathbf{i}^{\mathbf{ka}},\end{aligned} \tag{7}$$

where $\mathrm{W_e}$, $\mathrm{W_m}$ and $\mathrm{W_s}$ are the weight matrices. α_{ti} denotes the attention distribution on the knowledge-aware problem representations $\mathbf{h}^{\mathbf{ka}}$.

State Aggregation Mechanism: To incorporate the global information associated with the previously generated expression tree, the node state is recursively aggregated with its neighbor nodes in the expression tree at each time step. At time step t, all the generated nodes $(\mathbf{r_t})^0 = \{\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_t}\}$ are aggregated with a two-layer graph convolutional network (GCN) (Kipf and Welling, 2016). The aggregation functions are as follows:

$$\begin{aligned} \mathrm{D_{ii}} &= \sum\nolimits_{\mathrm{j=1}}^{\mathrm{n}} \mathrm{A}_{\mathrm{ij}}^{\mathrm{exp}}, \\ (\mathbf{r_t})^{\gamma+1} &= \sigma(\mathrm{D}^{-1}\mathrm{A}^{\mathrm{exp}}(\mathbf{r_t})^{\gamma}\mathrm{W_r}), \end{aligned} \tag{8}$$

where $\mathrm{W_r}$ is weight matrix. $\mathrm{A}^{\mathrm{exp}}$ is the adjacent matrix of the generated partial expression. If y_i is the child or parent of y_j or i = j, $\mathrm{A}_{\mathrm{ij}}^{\mathrm{exp}} = 1$. D means the number of adjacent nodes of each node plus 1. We use $\mathrm{D}^{-1}\mathrm{A}$ to normalize A. In this study, after two hops of GCN computation, we obtained the final context state $\mathbf{r_t}$ for each node in the partial expression.

2.6 Training

Given the training data $\mathrm{D} = (\mathrm{X}, \mathrm{Y})$, the objective function is to minimize the negative log likelihood:

$$\Delta(\mathrm{D}, \theta) = -\sum\nolimits_{\mathrm{i=1}}^{\mathrm{N_D}} \log \mathbf{P}(\mathrm{Y}|\mathrm{X}). \tag{9}$$

During training, for each question–answer pair (X, Y), we used the pre-order traversal of Y as the ground truth. The conditional probability is $\mathbf{P}(\mathrm{Y}|\mathrm{X})$:

$$\begin{aligned} \log \mathbf{P}(\mathrm{Y}|\mathrm{X}) = \prod_{t=1}^{n'} [\, &\mathbf{P}(y_t|y_{<t}, \mathrm{X}) \\ +\mathbf{P}(y_{t,l}|y_{<t}, \mathrm{X}) &+ \mathbf{P}(y_{t,r}|y_{<t}, \mathrm{X})] \end{aligned} \tag{10}$$

Here, $\mathbf{P}(y_{t,l}|y_{<t}, \mathrm{X})$ is an additional regular term about the child loss. At time step t, we only used the left child state $\mathbf{s_{t,l}}$ and right child state $\mathbf{s_{t,r}}$ to calculate the respective distribution scores $\mathbf{P}(y_{t,l}|y_{<t}, \mathrm{X})$ and $\mathbf{P}(y_{t,r}|y_{<t}, \mathrm{X})$, as shown in Equation 6. We expect that the distribution of node y_t is close to the ground truth, and that the distributions of its left and right children $y_{t,l}, y_{t,r}$ are also close to the ground truth.

3 Experiment

3.1 Dataset

We evaluated the proposed method on a large-scale dataset called Math23K, which was gathered by Wang et al. (2017) and contains 23,161 elementary school math word problems. Each problem was originally associated with an expression and answer. All problems in this dataset are described in Chinese and can be solved by a linear expression that contains only one unknown variable. We randomly split the dataset into a training set (80%) and testing set (20%).

Furthermore, we replaced all of the numbers in the problems with position tokens (e.g., N_1, N_2, N_3) in the preprocessing stage. After the model generated the expression, we replaced the position tokens in the expression with the numbers from the original problem, and executed this replaced expression to produce the answer.

We used Cilin (Mei, 1985) and Hownet (Dong et al., 2006) as our External Knowledge Source. Cilin is a Chinese synonym dictionary, where each word belongs to several different word groups. Hownet is a knowledge graph of Chinese words and concepts, where each word is labeled by several semantic units. We used these word groups and semantic units as our categories, and we set the max length of phrases in the phrase category to 3. We obtained 8,883 word-category pairs and 10,864 phrase category pairs.

3.2 Implementation Details

Our code was implemented with Pytorch [1]. We select the 4,000 words that appeared most frequently in the training data as the input vocabulary, and replaced the rest of the words in the problems with a token UNK. We set the dimension of hidden vectors d = 256. Both GCN and GAT have two layers. The number of heads K in GAT is 8. Model optimization was performed using an Adam optimizer (Kingma and Ba, 2014) with the learning rate set to 0.001. For the hyper-parameter setting, we set the dropout (Srivastava et al., 2014) rate to 0.5 and the batch size to 64. During training, it took 80 epochs to train the model. During decoding, the beam size was set to 5.

3.3 Baselines

To evaluate the performance of the proposed method, we compare it with the following baselines:

- **DNS (Wang et al., 2017):** This method has a two-layer GRU (Chung et al., 2014) encoder and a two-layer LSTM decoder. In addition, it uses a retrieval model to detect the problem that is most similar to the query problem from

[1]https://pytorch.org/

Models	Accuracy
DNS (Wang et al., 2017)	58.1%
DNS+Retrieval (Wang et al., 2017)	64.7%
Bi-LSTM (Wang et al., 2018a)	66.7%
ConvS2S (Wang et al., 2018a)	64.2%
Transformer (Wang et al., 2018a)	62.3%
Ensemble (Wang et al., 2018a)	68.4%
RecursiveNN (Wang et al., 2019)	68.7%
Tree-Decoder (Liu et al., 2019)	69.0%
GTS (Xie and Sun, 2019)	74.3%
KA-S2T (Our)	**76.3%**

Table 1: Answer accuracy of our model and other state-of-the-art models on Math23K dataset.

Models	Accuracy
GTS (Xie and Sun, 2019)	74.3%
KA-S2T w/o knowledge	75.5%
KA-S2T w/o multiple category	75.7%
KA-S2T w/o phrase category	76.0%
KA-S2T	76.3%

Table 2: Ablation study on reducing the amount of external knowledge incorporated into the model. “w/o phrase category” indicates the removal of knowledge about phrase categories from the KA-S2T model. “w/o multiple category” indicates that each entity in the knowledge base is connected to only one category that is most relevant to it.

the training set, and uses its expression as a template for the query problem. It combines the retrieval model with the DNS model to form a hybrid model “**DNS+Retrieval**”.

- **Ensemble (Wang et al., 2018a):** This ensemble model combines three types of Seq2Seq models: a bidirectional LSTM network (Wu et al., 2016), a convolutional Seq2Seq model (Gehring et al., 2017), and a transformer (Vaswani et al., 2017).

- **RecursiveNN (Wang et al., 2019):** A recursive neural network model first predicts the tree structure template using a Seq2Seq model, and then infers the expression based on the features extracted by a bidirectional LSTM and self-attention mechanism.

- **Tree-Decoder (Liu et al., 2019):** A Seq2Tree generative model with an auxiliary stack and a tree-structured decoder that generates the abstract syntax tree of the equation in a top-down manner. We call this method “**Tree-Decoder**”.

- **GTS (Xie and Sun, 2019):** A tree structured neural model that generates an expression tree in a goal-driven manner based on the parent node and the left sibling tree of each node. It uses top-down goal decomposition and bottom-up subtree embedding to directly predict the expression tree.

3.4 Result Analysis

To assess the overall performance of our KA-S2T model, we compared it with the performances of other state-of-the-art models on the Math23K dataset. Table 1 shows the accuracy of the results obtained by executing the generated expressions of these models, from which we can conclude the following:

1) The tree-structured decoder can improve the performance of most baselines. For example, the Seq2Tree baseline Tree-Decoder and GTS performed better than the best-performing Seq2Seq baseline RecursiveNN. This result demonstrates the effect of the tree-structured decoder.

2) The deep-learning models DNS and Ensemble did not perform as well as the RecursiveNN with attention mechanism. Tree-Decoder and GTS both have an attention structure, which proves that an attention mechanism can effectively capture the key features of a problem.

3) GTS performed the best of all the baselines, even better than the Tree-Decoder, which also has a Seq2Tree structure. The reason for this may be that GTS directly uses the states of the parent node and left sibling node to generate the current node. Tree-Decoder still sequentially generates expressions based on the last node state, and takes the sibling node and parent node states as additional features.

4) Finally, compared with GTS, the accuracy of KA-S2T was 2.0% better. We attribute the superior performance of KA-S2T to two properties: KA-S2T incorporates external knowledge, which can better capture the relationship between words. KA-S2T recursively aggregates the neighbors of each node in the partial expression, and thus better captures the global information associated with the currently generated expression tree.

3.5 Ablation Study

Effect of external knowledge: Table 2 shows the

Models	Accuracy
RecursiveNN (Wang et al., 2019)	68.7%
GTS (Xie and Sun, 2019)	74.3%
KA-S2T w/o child loss & state agg	72.9%
KA-S2T w/o state agg	73.5%
KA-S2T w/o child loss	75.2%
KA-S2T	75.5%

Table 3: Ablation study of different decoder structures. "w/o state agg" indicates that the model did not use the context state r_t to incorporate global expression information at each time step. "w/o child loss" indicates that the loss function did not use the additional regular terms defined in Equation 10 to introduce child loss. For a fair comparison, no external knowledge was used in KA-S2T and its variants.

results of ablation experiments conducted to reduce the amount of external knowledge incorporated into the model. We have the following observations:

1) Without external knowledge, the KA-S2T's answer accuracy would be reduced to 75.5%. However, "KA-S2T w/o knowledge" still outperforms the best-performing baseline GTS, which further verifies the effectiveness of our tree-structured decoder. We will analyze the effectiveness of these tree-structured decoder in the following section.

2) The use of multiple categories and phrase categories can improve accuracy by 0.2% and 0.5%, respectively. Their combination can provide further improvement in model performance. These results show that the external knowledge of the relationship between entities and categories enables the model to capture common-sense information and obtain better interaction between words.

Effect of KA-S2T tree-structured decoder: We designed several ablation experiments to measure the effect of our KA-S2T tree-structured decoder, the results of which are shown in Table 3. For a fair comparison, we used no external knowledge in these KA-S2T and variant models. From the table, we can see that:

1) The "KA-S2T w/o child loss & state agg" model, which can be regarded as a basic Seq2Tree model, achieved better accuracy than the best-performing Seq2Seq baseline RecursiveNN. The main difference between this model and RecursiveNN is that this model generates the current node state based on its parent node, and RecursiveNN generates the current node state based on the last node. This finding once again confirms the effectiveness of the Seq2Tree structure because fa-

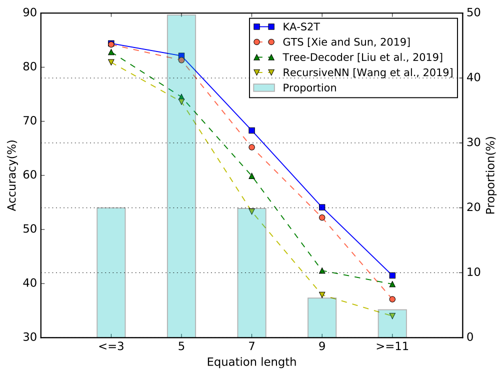


Figure 4: Model performance on expression trees of different lengths

cilitates the capture of long-distance dependencies.

2) The "KA-S2T w/o child loss" model outperformed the best-performing Seq2Tree baseline GTS, but the "KA-S2T w/o child loss & state agg" model did not perform as well as GTS. Compared with the basic Seq2Tree model, GTS uses bottom-up subtree embedding to introduce left sibling tree information. The state aggregation mechanism achieves further improvements, not only focusing on the left sibling subtree, but also incorporating the global information associated with the entire generated partial expression. This result proves the effectiveness of the state aggregation mechanism.

3) The KA-S2T model with child loss is better than the KS-S2T model without child loss, which proves that the child loss function obtains better performance.

Model performance on expression trees of different lengths: We compared the KA-S2T model with the other three best-performing state-of-the-art methods to investigate the performances of the models that have expression trees of different lengths. As shown in Figure 4, KA-S2T outperformed the other three state-of-the-art methods with respect to expression trees of different lengths, especially when the length of the expression tree was between 5 and 9. One possible explanation for this is that because the expression tree is complex, to achieve better performance, the model needs external knowledge and the global information associated with the expression. However, when the expression is too complex, the probability of the model producing correct results is too low, so the performance gap between the different models is not as obvious. These results further demonstrates the beneficial effect of incorporating

Problem 1: The library purchased N1 different types of books. Among them, there are N2 literary books and N3 encyclopedias. The number of science books is N4 more than N5 times the number of literary books. How many books are there in total?	
GTS: + + * N2 N5 N4 N2	**KA-S2T:** + + * N2 N5 N4 + N2 N3
Problem 2: A school spent N1 to buy N2 basketballs and N3 footballs. The price of each basketball is N4. How much does each basketball cost more than each football?	
GTS: - N4 / - N1 * N2 N4 N2	**KA-S2T:** - N4 / - N1 * N2 N4 N3

Table 4: Two examples of expressions generated by KA-S2T compared with GTS.

external knowledge and using state aggregation mechanism.

3.6 Case Study

Table 4 shows two examples generated by our KA-S2T model for comparison with GTS (Xie and Sun, 2019).

In Problem 1, without external knowledge, GTS does not know that encyclopedias are books much like literary books and scientific books, and therefore it generates incorrect results. By incorporating external knowledge, KA-S2T is able to obtain the relationship between these three types of books.

In Problem 2, there is a long distance between "N3" and the first two nodes [-, N4] of the expression tree. GTS does not realize that the current sub-expression tree indicates the price of each football and generates "N2" based on the nearest node "N4". Our proposed method can capture long-distance features and therefore generate correct results.

4 Related Work

Solving math word problems has long been a challenging task (Fletcher, 1985; Bakman, 2007; Roy and Roth, 2016) and has attracted the attention of many researchers. Some methods on math word problem solving incorporate extra features by manually crafting fine-grained templates or defining math concepts. Huang et al. (2017) formulated fine-grained templates and aligned numbers in math word problems to those candidate templates. Roy and Roth (2018) developed declarative rules to transform math concepts into expressions. These methods require manually formulated features and may be difficult to apply to math word problems in different domains. Recently, many studies have used deep learning methods to incorporate external knowledge from the knowledge base into many NLP tasks, such as dialogue systems (Zhong et al., 2019) and reading comprehension (Wang and Jiang, 2019; Qiu et al., 2019). These methods extend knowledge triples into natural language sequences or build multi-hop inference graphs based on relationships in the knowledge base, and have achieved promising results. In this paper, we model the entities in the problem and their categories as entity graphs and use graph attention to generate knowledge-aware problem representations.

Seq2Seq neural networks (Sutskever et al., 2014) have achieved promising results on math word problem solving. For instance, Wang et al. (2017) used a Seq2Seq model to generate math expressions. Wang et al. (2018b) incorporated reinforcement learning into the model to construct a math expression step by step. Zou and Lu (2019) used a data-driven approach to semantically parsing text into math expressions. Recently, tree-structured decoder was used to further improve the seq2seq framework. Xie and Sun (2019) propose a seq2tree model to generate expression tree in a goal-driven manner based on the parent node and left sibling tree of each node. Liu et al. (2019) propose a tree-structured decoding method with an auxiliary stack that generates the abstract syntax tree of the equation in a top-down manner. In this paper, we generated the pre-order math expression tree based on parent node state of each node and recursively aggregated neighbors of each node in the partial expression tree to incorporate global information.

5 Conclusion

In this study, we proposed a novel knowledge-aware sequence-to-tree model that can automatically solve math word problems. We used an entity graph to incorporate common sense knowledge from external knowledge bases into the proposed model. In addition, we proposed a tree-structured decoder with a state aggregation mechanism for generating math expressions. Our experimental results confirmed that our KA-S2T model outperformed other state-of-the-art models.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by China National Key R&D Program (No.2018YFB1005104, 2018YFC0831105, 2017YFB1002104), National Natural Science Foundation of China (No.61751201, 61976056, 61532011), Shanghai Municipal Science and Technology Major Project (No.2018SHZDZX01), Science and Technology Commission of Shanghai Municipality Grant (No.18DZ1201000, 17JC1420200).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. *arXiv preprint math/0701393*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Zhendong Dong, Qiang Dong, and Changling Hao. 2006. Hownet and the computation of meaning.

Edward A Feigenbaum, Julian Feldman, et al. 1963. *Computers and thought*, volume 7. McGraw-Hill New York.

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

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *ICML*, pages 1243–1252. JMLR. org.

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

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to solve math word problems. In *EMNLP*, pages 805–814.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Qianying Liu, Wenyv Guan, Sujian Li, and Daisuke Kawahara. 2019. Tree-structured decoding for solving math word problems. In *EMNLP-IJCNLP*, pages 2370–2379.

Jiaju Mei. 1985. *Tongyi ci cilin*. Shangai cishu chubanshe.

Delai Qiu, Yuanzhe Zhang, Xinwei Feng, Xiangwen Liao, Wenbin Jiang, Yajuan Lyu, Kang Liu, and Jun Zhao. 2019. Machine reading comprehension using structural knowledge graph-aware network. In *EMNLP-IJCNLP*, pages 5898–5903.

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

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Chao Wang and Hui Jiang. 2019. Explicit utilization of general knowledge in machine reading comprehension. In *ACL*, pages 2263–2272.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to a expression tree. In *EMNLP*, pages 1064–1069.

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 *AAAI*.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *AAAI*.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *EMNLP*, pages 845–854.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

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

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Peixiang Zhong, Di Wang, and Chunyan Miao. 2019. Knowledge-enriched transformer for emotion detection in textual conversations. In *EMNLP-IJCNLP*, pages 165–176.

Yanyan Zou and Wei Lu. 2019. Text2math: End-to-end parsing text into math expressions. In *EMNLP-IJCNLP*, pages 5330–5340.