7-2020

Graph-to-tree learning for solving math word problems

Jipeng ZHANG
Singapore Management University, jpzhang@smu.edu.sg

Lei WANG
Singapore Management University, lei.wang.2019@phdcs.smu.edu.sg

Roy Ka-Wei LEE

Yi BIN

Yan WANG

See next page for additional authors

Citation

ZHANG, Jipeng; WANG, Lei; LEE, Roy Ka-Wei; BIN, Yi; WANG, Yan; SHAO, Jie; and LIM, Ee-peng. Graph-to-tree learning for solving math word problems. (2020). *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 3928-3937.
Available at: https://ink.library.smu.edu.sg/sis_research/5273

Graph-to-Tree Learning for Solving Math Word Problems

Jipeng Zhang[1,2,*], Lei Wang[2,*], Roy Ka-Wei Lee[3], Yi Bin[1], Yan Wang[4], Jie Shao[1,5,#], Ee-Peng Lim[2]

[1]Center for Future Media, University of Electronic Science and Technology of China
[2]School of Information Systems, Singapore Management University
[3]Department of Computer Science, University of Saskatchewan
[4]Tencent AI Lab
[5]Sichuan Artificial Intelligence Research Institute

zhangjipeng20@std.uestc.edu.cn, lei.wang.2019@phdcs.smu.edu.sg, roylee@cs.usask.ca, yi.bin@hotmail.com, bradenwang@tencent.com, shaojie@uestc.edu.cn, eplim@smu.edu.sg

Abstract

While the recent tree-based neural models have demonstrated promising results in generating solution expression for the math word problem (MWP), most of these models do not capture the relationships and order information among the quantities well. This results in poor quantity representations and incorrect solution expressions. In this paper, we propose *Graph2Tree*, a novel deep learning architecture that combines the merits of the graph-based encoder and tree-based decoder to generate better solution expressions. Included in our *Graph2Tree* framework are two graphs, namely the *Quantity Cell Graph* and *Quantity Comparison Graph*, which are designed to address limitations of existing methods by effectively representing the relationships and order information among the quantities in MWPs. We conduct extensive experiments on two available datasets. Our experiment results show that *Graph2Tree* outperforms the state-of-the-art baselines on two benchmark datasets significantly. We also discuss case studies and empirically examine *Graph2Tree*'s effectiveness in translating the MWP text into solution expressions[1].

1 Introduction

Math Word Problem (MWP), which involves automatically answering a mathematical question according to a textual description, is an important natural language understanding task that has been studied by researchers since the 1960s (Bobrow, 1964). A typical MWP is a short narrative that describes a problem and poses a question about an unknown quantity. Table 1 provides an example of a typical MWP where the reader is required to infer the revenue of a store after selling all the teddy bears. Earlier studies have attempted to perform the MWP task via statistical machine learning methods (Kushman et al., 2014; Hosseini et al., 2014; Mitra and Baral, 2016; Roy and Roth, 2018) and semantic parsing approaches (Shi et al., 2015; Koncel-Kedziorski et al., 2015; Roy and Roth, 2015; Huang et al., 2017). However, these methods are non-scalable as tremendous efforts are required to design suitable features and expression templates.

Problem: 348 teddy bears are sold for \$23 each. There are total 470 teddy bears in a store and the remaining teddy bears are sold for \$17 each. How much did the store earn after selling all the teddy bears?
Expression: $x = 348 \times 23 + (470 - 348) \times 17$
Solution: 10078

Table 1: A math word problem.

In recent years, deep learning-based models have been developed to solve MWPs. These deep learning methods are able to automate the learning of features and generalize well by returning new solution expressions that are unseen in the training datasets. Wang et al. (2017) proposed a large-scale MWP dataset and applied a vanilla sequence to sequence (seq2seq) model to translate the language text to a solution expression. Since then, many research efforts mainly focused on improving the generation of solution expressions. Some researchers have proposed seq2seq models to improve solution expression generation using implicit (Wang et al., 2018; Chiang and Chen, 2019) and explicit (Wang et al., 2019; Liu et al., 2019; Xie and Sun, 2019) tree structures. Improving the representation of quantity is a potential approach to achieve better solution expressions. For example, to get the correct solution expression for the problem described in Table 1, an ideal MWP model should be able to associate quantity, i.e., 348 teddy bears, with its price attribute of \$23, and understand the arithmetic order by deriving 122 remaining teddy bears, i.e.,

[*]Equal Contribution
[#]Corresponding Author
[1]Code could be found at https://github.com/2003pro/Graph2Tree

$470 - 348$, before associating the price attribute of \$17. The existing deep learning models are not effective in capturing such relationships and order information among the quantities in MWPs, thus resulting in an inaccurate representation of the final solution expressions.

To enrich the representation of a quantity, the relationships between the descriptive words associated with a quantity need to be modeled. However, such relationships cannot be effectively modeled using recurrent models, which are commonly used in the existing MWP deep learning methods. Inspired by the concept of Quantity Schema (Roy and Roth, 2015) and Qset (Koncel-Kedziorski et al., 2015), we design the *Quantity Cell Graph* to associate informatively descriptive words to quantity. We first extract associated nouns, verbs, adjectives, units, and rates that describe a quantity in the MWP text. Next, we construct a graph where the extracted descriptive words are represented as neighbor nodes directly linked to a quantity. Finally, a neural network model is used to learn enriched latent representations of the quantities based on the constructed *Quantity Cell Graph*.

The loss of quantities' numerical qualities in existing MWP methods can also result in poor quantity representations. Most of the existing MWP methods often replace quantities with special symbols (e.g., "n_1", "n_2", etc.) (Wang et al., 2017, 2018; Liu et al., 2019). The loss of quantities' numerical qualities could be problematic when generating solution expressions. Take the example in Table 1, without modeling the numerical qualities of quantities, an MWP method may learn a solution expression "$384 - 470$" which results in a negative number that is unlikely to occur in MWPs. To address this limitation, we introduce the *Quantity Comparison Graph*, which was inspired by a numerical machine reading comprehension model proposed by Ran et al. (2019). The intuition of *Quantity Comparison Graph* is to retain the numerical qualities of the quantity and leverage certain heuristics to represent the relationships among quantities in MWPs such that solution expressions reflect a more realistic arithmetic order.

Besides improving the quantity representation, we also aim to improve the solution expression generative process. For longer solution expressions in MWPs, as some quantities are repeatedly used in different arithmetic sub-solution expressions, the existing methods which utilized recurrent neural networks may not be able to learn the underlying reasoning process and arithmetic order. For example, in Table 1, the quantity 348 is being used in "$348 * 23$" and "$(470 - 348) * 17$". To address this limitation, we propose to use a graph encoder to guide the learning of representations of quantities and a tree decoder to explicitly model the multi-stage reasoning process.

Contribution. In this paper, we combine the above-proposed solutions and introduce the **Graph2Tree** solver to address the existing MWPs methods' limitations. The contributions of this paper are as follows:

- We construct the **Quantity Cell Graph** and **Quantity Comparison Graph** to enrich the quantity representations by capturing relationships between quantities and their attributes and retaining the quantities' numerical qualities.
- We propose the **Graph2Tree** to improve the learning of solution expressions' generation. The *Graph2Tree* model uses a graph transformer to learn the latent quantity representations from our proposed graphs, and a tree structure decoder to generate a solution expression tree. To the best of our knowledge, this is the first graph-to-tree model for MWPs.
- We conduct extensive experiments on two available large-scale MWPs datasets, and our results show that our proposed *Graph2Tree* model outperforms state-of-the-art baselines on MWP task.

2 Problem Formulation

We denote the text of the math word problem as P, where P is a sequence of word tokens and numeric values. We let $V_p = \{v_1, \cdots, v_m\}$ denote the word tokens in P and $n_P = \{n_1, \cdots, n_l\}$ denote the set of quantities in P. Our goal is to map P to a valid and correct mathematical expression E_p.

Solving MWPs requires an understanding of quantities in problem and their complex mathematical relationships. MWPs are often expressed in a linear textual sequence form, which is not ideal for learning the quantities' complex interactions. Thus, we propose to formulate the problem into graph form so that the relationships between quantities can be expressed more explicitly. The problem text P is transformed into graph $\mathcal{G}$ by augmenting the text sequences with other structural information like dependency parsing and POS tagging.

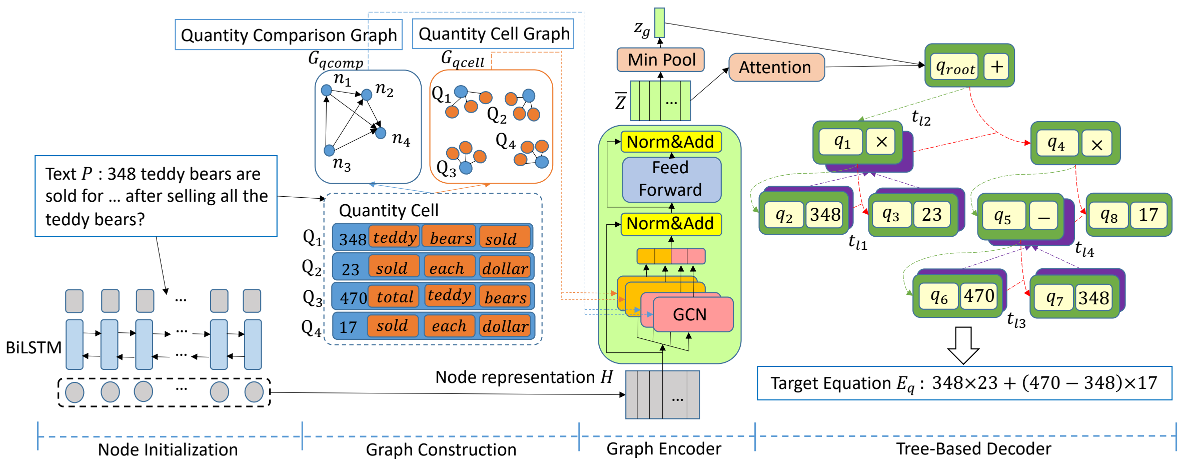

Figure 1: Overview of the proposed model. In order to initialize representation of text P, a BiLSTM is used to compute node representation H. Later, after extracting Quantity Cells from text P, we construct Quantity Comparison Graph and Quantity Cell Graph. With two graphs and H, we use the proposed graph transformer to get the internal representation. Finally, a tree-based decoder is implemented to generate the target euqation E_q.

The final mathematical expression E_p that we aim to construct can always be represented as a solution expression tree T. T may include constant quantities, operators and quantities in n_P. The set of constant quantities V_{con} contains some special values not appeared in text like $\pi, 1$. The set of math operators V_{op} contains $\{+, -, *, /\}$. Overall, the target vocabulary of P can be denoted as $V_{dec} = V_{op} \cup V_{con} \cup n_P$ (V_{dec} varies in different problems as n_P varies) . The goal of our *Graph2Tree* model here is to estimate the conditional probability $P(E_p|P)$, which can be transformed as $P(T|\mathcal{G}, V_{dec})$.

3 Methodology

Figure 1 shows our proposed *Graph2Tree* framework. *Graph2Tree* first encodes the MWP text input using BiLSTM and simultaneously constructs *Quantity Cell Graph* and *Quantity Comparison Graph*. The output of BiLSTM, word-level representations, are used as node representations. Together with the two constructed graphs, the node representations are input into a graph transformer to learn a graph representation of the MWP. The multiGCN component of the graph transformer is modified to learn the graph representation based on the *Quantity Cell Graph* and *Quantity Comparison Graph*. This enriches the final graph representation with quantities' relationship information and numerical qualities. Pooling is used to aggregate all nodes into a pool-based graph embedding vector as the graph transformer's output. Finally, the output graph representation and the updated node representations are used as input to a tree-structure decoder to infer the final solution expression tree.

3.1 Graph-Based Encoder

There have been some graph-based models (Sahu et al., 2019) intending to grab the complicated relations in text. The graph-based encoder in our *Graph2Tree* framework is inspired by the graph transformer model (Koncel-Kedziorski et al., 2016; Cai and Lam, 2019). We first discuss the initialization of node representations based on MWPs' input problem text. Next, we introduce the construction of the *Quantity Cell Graph* and *Quantity Comparison Graph*. Finally, we discuss the learning of graph representation using the graph transformer module.

3.1.1 Node Representation Initialization

To initialize the node representations, we first learn the word-level hidden state representations of the input MWP text using a BiLSTM neural network, $H = \{h_1, \cdots, h_N\} \in R^{N \times d}, N = m + l$. Here d denotes the dimension of hidden vectors, m represents the number of words, and l represents the number of quantities. The learned hidden state representations will be used as the input node representations for the graph encoder.

3.1.2 Quantity Cell

We refer all quantities n_P and words V_p from the problem as nodes in the graph. Next, we define a *quantity cell* as a subset of nodes in the graph that are associated with a quantity. Formally, each

MWP P is transformed into multiple *quantity cells* $QC = \{Q_1, Q_2, \cdots, Q_m\}$, where m is the number of quantities in P. Each quantity cell $Q_i \in QC$ contains a quantity token $\{n_i\}$ and the corresponding attributes $\{v_{1i}, \cdots, v_{qi}\}$. These *quantity cells* are sub-graph representations of quantity-related information in the MWPs. Dependency parsing, constituency parsing and POS tagging implemented with *stanford corenlp toolkit* (Manning et al., 2014) are used to extract and construct the *quantity cells*.

A *quantity cell* in an MWP P consists of the following properties:

- *Quantity.* The quantity numeric value.
- *Associated Nouns.* We consider the nouns related to the *Quantity* in the dependency parse tree. *Associated Nouns* are the nouns related by the *num*, *number* and *prep_of* relations.
- *Associated Adjectives. Associated Adjectives* are the adjectives related to *Quantity* or *Associated Nouns* with the *amod* relation, which is detected by the dependency parser.
- *Associated Verbs.* For each *Quantity*, we detect the related verbs, *Associated Verbs*, according to *nsubj* and *dobj* relations.
- *Units and Rates.* We detect the nouns related to *Associated Nouns* by *prep_of* as the *Unit*. The nouns related *Associated Nouns* which own the key words such as "each", "every" and "per" are regarded as *Rates*.

If the *quantity cell* detection process does not grab any attributes, we will use a window centered on *Quantity* to select neighboring words as the attributes of the *Quantity*. An example of the *quantity cell* is illustrated in the left part of Figure 1.

3.1.3 Quantity Graph Construction

From the quantity cells, we construct two graphs: *Quantity Cell Graph* and *Quantity Comparison Graph*. The goal of the *Quantity Cell Graph* is to associate informative descriptive words to quantity so as to enrich the quantity's representation. Similarly, the goal of the *Quantity Comparison Graph* is to retain the numerical qualities of the quantity and leverage heuristics to improve representations of the relationships among quantities. Formally, we define the construction of two graphs as follow:

- **Quantity Cell Graph** G_{qcell}**.** For each *Quantity Cell* $Q_i = \{n_i\} \cup \{v_{1i}, \cdots, v_{ni}\}$, the undirected edge e_{ij} between n_i and each $v_j \in \{v_{1i}, \cdots, v_{qi}\}$ will be added to the graph G_{qcell}.
- **Quantity Comparison Graph** G_{qcomp}**.** For two quantity nodes $n_i, n_j \in n_P$, a directed edge $e_{ij} = (n_i, n_j)$ pointing from n_i to n_j will be added to the graph G_{qcomp} if $n_i > n_j$. This heuristic constraint can prevent the subtracting a larger number from a smaller number, which results in a negative number.

We represent the two graphs using adjacency matrices. For graph G, an adjacency matrix $A \in R^{N \times N}$ is first initialized. If there exists an edge between the i-th and j-th nodes, we need to assign value 1 to corresponding position of the adjacency matrix $(i, j, A_{i,j})$ for this edge. Otherwise, 0 would be assigned. Thus, we compute the adjacency matrix A_{qcomp} for graph G_{qcomp} and A_{qcell} for G_{qcell}.

3.1.4 Graph Transformer

The inputs to the graph transfer module are adjacency matrices of multiple graphs $\{A_k\}_{k=1}^{K}$, $A_k \in \{A_{qcomp}, A_{qcell}\}$ and initial node embeddings H, where K is the number of graphs and each $A_k \in R^{N \times N}$ is the adjacency matrix for k-th graph. K graphs are used as we adopt a multi-head structure in our model and they are split evenly between *Quantity Cell Graphs* and *Quantity Comparison Graph*.

The graph transformer first utilizes graph convolution networks (GCNs) (Kipf and Welling, 2017) to learn the graph node features. For multiple graphs, we use a K-head graph convolution setup. This is similar to the transformer model proposed in Vaswani et al. (2017), where K separate graph convolution networks are used and concatenated before a residual connection is applied.

Specifically, a single GCN has its parameter $W_{gk} \in R^{d \times d_k}$, where $d_k = d/K$. Given an adjacency matrix A_k representing graph structure and a feature matrix X (in the beginning, X is set as H) meaning the input feature for all nodes, we define learning of GCN as follow:

$$GCN(A_k, X) = GConv_2(A_k, GConv_1(A_k, X)) \tag{1}$$

Here, the GCN contains 2 different graph convolution operations:

$$GConv(A_k, X) = relu(A_k X^T W_{gk}) \tag{2}$$

For each graphs $\{A_k\}_{k=1}^{K}$, we perform learning of GCN in parallel, yielding d_k-dimensional output values. The output values are concatenated and projected, resulting in the final values:

$$Z = \mathop{\Big\|}_{k=1}^{K} GCN(A_k, H) \tag{3}$$

Here, $\|$ denotes the concatenation of the K GCN heads.

Graph transformer then augments this K-head graph convolution network with a feed-forward network, layer-norm layer, and residual connection:

$$\hat{Z} = Z + LayerNorm(Z) \tag{4}$$
$$\overline{Z} = \hat{Z} + LayerNorm(FFN(\hat{Z})) \tag{5}$$

here, $FFN(x)$ is a two-layer feed-forward network with a $relu$ function between layers:

$$FFN(x) = max(0, xW_{f1} + b_{f1})W_{f2} + b_{f2} \tag{6}$$

The resulting node representations $\overline{Z}$ represent quantities, entities and relations. In order to learn the global context graph representation, we apply the element-wise min-pooling operation on all learned node representations. Finally, the global feature is fed into a fully connected neural network (FC) to generate the graph representation z_g:

$$z_g = FC(MinPool(\overline{Z})) \tag{7}$$

3.2 Tree-Based Decoder

Inspired by the the Goal-driven Tree Structure (GTS) (Xie and Sun, 2019), we build a tree-based decoder to construct the solution expressions. We set the quantity nodes to be the leaf nodes and each operator node must have two child nodes. As such, the specialized tree decoder generates an equation following the pre-order traversal ordering. As part of the tree construction process, the centermost operator is first produced, followed by the left child node. This process is repeated until the leaf node is produced. Subsequently, we generate the right child nodes recursively.

3.2.1 Tree Initialization

To start the above mentioned tree generation process, our model initializes the root node vector q_{root} according to the global context graph representation z_g. For each token y in the target vocabulary V_{dec} of P, the representation for a certain token $e(y|P)$ is defined as:

$$\begin{cases} e_{(y,op)} & if\ y \in V_{op} \\ e_{(y,con)} & if\ y \in V_{con} \\ \overline{z}^p_{loc(y,P)} & if\ y \in n_P \end{cases} \tag{8}$$

The expression trees in our decoder contain three kinds of nodes: operators, constant quantities, and quantities that appeared in P. Constant quantities and quantities in n_P are always set to be in leaf nodes position. Operators will always take up the positions of the non-leaf nodes. The quantities' representations in n_P are dependent on certain MWPs, i.e., y will take the corresponding $\overline{z}^p_{loc(y,P)}$ from $\overline{Z}$. The representations of operators and constant quantities are independent, i.e., their representations are obtained by 2 independent embedding matrices M_{op} and M_{con}.

3.2.2 Pre-Order Tree Generation

We adopt the pre-order traversal manner to construct the expression tree:

- **Step 1.** The generation starts with a derivation tree with only a root node q_{root}. We use attention module of GTS to encode the node embedding $\overline{Z}$ into global graph vector G_c:

$$G_c = \mathrm{GTS} - \mathrm{Attention}(q_{root}, \overline{Z}) \tag{9}$$

- **Step 2.** This tree decoder applies left sub-node generation module to the derivation in a top-down manner, generating new left child node q_l conditioned on the parent node q_p and global graph G_c. Note that the token $\hat{y}$ is predicted when generating the new node:

$$\begin{aligned} q_l &= \mathrm{GTS} - \mathrm{Left}(q_p, G_c) \\ \hat{y} &= \mathrm{GTS} - \mathrm{Predict}(q_l, G_c) \end{aligned} \tag{10}$$

  If the generated $\hat{y}$ is an operator, two empty child node positions are created and we will keep executing **Step 2**. This step works like decomposing the whole goal into multi-stage reasoning. If the generated $\hat{y}$ is a quantity (constant or from n_P), we will get into **Step 3**.

- **Step 3.** The tree decoder switches to use the right sub-node generation module and populate the empty right node position. At every decoding step, we use the left child node q_l, global graph vector G_c and a sub-tree embedding t_l as the input to the right generation module and generate the right child node q_r and the corresponding token $\hat{y}_r$:

$$\begin{aligned} q_r &= \mathrm{GTS} - \mathrm{Right}(q_l, G_c, t_l) \\ \hat{y}_r &= \mathrm{GTS} - \mathrm{Predict}(q_r, G_c) \end{aligned} \tag{11}$$

  The addition of the sub-tree embedding works similarly to incorporating a sub-tree copying mechanism. The additional sub-tree embedding t_l is computed by using sub-tree embedding component of GTS:

$$t_l = \mathrm{GTS} - \mathrm{SubTree}(\hat{y}_l, q_l) \tag{12}$$

If $\hat{y}_r$ is an operator, the next step should go back to **Step 2**. If $\hat{y}_r$ is a quantity, we will get into **Step 4**.

- **Step 4.** The model switches to backtracking to find the new empty right node position. If the model cannot find the new empty right node position, the generation is completed. If the empty right node position still exists, go back to **Step 2**.

3.3 Model Learning

For each problem-tree expression example, (p, T), the loss function $L(T, P)$ is defined as the a sum of the negative log-likeihoods of probabilities for predicting t-node token y_t. Formally, our training goal is to minimize the following loss function:

$$L(T,P) = \sum_{t=1}^{E} -\log prob(y_t|q_t, G_c, P) \quad (13)$$

where q_t is the goal vector, G_c is the global graph context, E is the number of tokens in T, and $prob$ is computed by distribution computation function in GTS.

4 Experiment

In this section, we compare our proposed *Graph2Tree* model with state-of-the-art baselines. We also conduct ablation study and analysis to investigate the effectiveness of various components of our model.

Datasets. Two commonly-used MWP datasets are used in our experiments: MAWPS (Koncel-Kedziorski et al., 2016) with 2,373 problems and Math23K (Wang et al., 2017) with 23,162 problems.

Baselines. We compare *Graph2Tree* to an extensive set of baselines and state-of-the-art models: **DNS** (Wang et al., 2017) uses a vanilla seq2seq model to generate expressions. **Math-EN** (Wang et al., 2018) benefits from an equation normalization to reduce target space. **T-RNN** (Wang et al., 2019) applies recursive neural networks over predicted tree-structure templates. **S-Aligned** (Chiang and Chen, 2019) designs the decoder with a stack to track the semantic meanings of operands. **GROUP-ATT** (Li et al., 2019) borrows the idea of multi-head attentions from Transformer (Vaswani et al., 2017). **AST-Dec** (Liu et al., 2019) creates an expression tree with a tree LSTM decoder. **GTS** (Xie and Sun, 2019) develops a tree structured neural networks in a goal-driven manner to generate expression trees. **IRE** (Sahu et al., 2019) is another baseline that was first proposed in relation extraction and has something in common with our method.

Implementation Details and Evaluation Metric. In the *Graph2Tree* model, we use a word embedding (not pre-trained) with 128 units, a one layer graph transformer with 4 GCNs, each of which has the dimension of the hidden state set to 128. The dimensions of the hidden state for all the other layers are set to 512. Our model is trained for 80 epochs. Mini-batch size and dropout rate are set to 64 and 0.5, respectively. For optimizer, we use Adam with learning rate set to 0.001, $\beta_1 = 0.94$ and $\beta_2 = 0.99$, and the learning rate will be halved every 20 epochs. Also, we use a beam size of 5 in beam search.

For the Math23K dataset, some methods are evaluated using 5-fold cross-validation, expressed in "Math23K*", and others are evaluated using the available test set (expressed as "Math23K"). We evaluate *Graph2Tree* on both settings. For the MAWPS dataset, the models are evaluated with 5-fold cross-validation. Following previous works, we use solution accuracy as the evaluation metric.

4.1 Overall Results

	MAWPS	Math23K	Math23K*
DNS	59.5	-	58.1
Math-EN	69.2	66.7	-
T-RNN	66.8	66.9	-
S-Aligned	-	-	65.8
GROUP-ATT	76.1	69.5	66.9
AST-Dec	-	69.0	-
GTS	82.6	75.6	74.3
IRE	-	76.7	-
Graph2Tree	**83.7**	**77.4**	**75.5**

Table 2: Solution accuracy of *Graph2Tree* and various baselines. Note that Math23K denotes results on public test set and Math23K* denotes 5-fold cross-validation.

Table 2 shows the solution accuracy of *Graph2Tree* and various baselines. We observe that *Graph2Tree* outperforms all baselines in the two MWP datasets. As the code for GTS is made available[2], we implemented GTS and tested it on all dataset settings. We also statistically test the improvement of *Graph2Tree* over the strongest baseline (i.e., GTS) and found that the improvement to be significant at 0.01 level using paired t-test. The superior performance of *Graph2Tree* demonstrates

[2]`https://github.com/ShichaoSun/math_seq2tree`

the importance of enriching quantity's representations in handling the MWP task.

4.2 Ablation Study and Parameter Analysis

To understand the effects of the various components and hyperparameters in our *Graph2Tree* model, we conduct ablation studies and parameter analysis on the Math23K dataset.

4.2.1 Effect of Quantity Graph

We investigate the effects of *Quantity Cell Graph* and *Quantity Comparison Graph* in our model. The results of our ablation study are shown in Table 3. We find that the *Graph2Tree* with both *Quantity Cell Graph* and *Quantity Comparison Graph* performs the best. We also observe that having either *Quantity Cell Graph* and *Quantity Comparison Graph* still outperforms the implementation without either graph (i.e., full-connected graph). More interestingly, we also noted that enriching the quantity representation with either graph would also outperform the baseline GTS model in this task, suggesting the importance of quantity representation in MWP task. From this study, we also infer that improving quantity representation, modeling the relationships among quantities, and retaining their numerical qualities help to achieve better results for the MWP task. Also, if two types of graphs are merged into an integrated graph, the performance drops. We postulate that a possible reason for the inferior performance may be due to the noise introduced by the integration of multiple graphs.

	Math23K
Graph2Tree	77.4
only Quantity Cell Graph	76.8
only Quantity Comparison Graph	76.9
only Full-Connected Graph	75.3
merge two graphs as single one	76.4

Table 3: Solution accuracy with various graph configurations in *Graph2Tree*.

4.2.2 Effect of Graph Number

The number of GCNs is a tuneable hyperparameter in our *Graph2Tree* model. Thus, we investigate the effect of the number of GCNs on our model's performance. We varied the number of GCNs from 2, 4, 8. Note that even numbers are used as the GCNs are split evenly to model the *Quantity Cell Graph* and *Quantity Comparison Graph*. Table 4 shows the study's results. We observe that the 4-GCN version achieves the best performance. A potential reason could be due to the optimal capacity of information aggregation is achieved using 4 GCNs over the two quantity graphs.

	Math23K
w/ 2 GCN	76.7
w/ 4 GCN	77.4
w/ 8 GCN	76.9

Table 4: Solution accuracy with varying number of GCNs.

4.2.3 Impact of Length of Expression

To investigate how well our *Graph2Tree* model performs with the increasing expression complexity as compared to state-of-the-art models using explicit tree decoders, we analyze the increasing number of operators in the test set. From the results shown in Table 5, we note that:

(1) Our proposed *Graph2tree* outperforms the other two models in most cases except that the number of operators equals to 5. In other cases with less than 5 operators, our model shown statistically significant improvements over other two models.

(2) All the models' performances follow an accuracy descending pattern when the length of expression becomes longer. This is intuitive as longer expressions often associate with more complex questions that are more difficult to solve and have fewer data for training.

#Op	Pro (%)	AST-Dec (%)	GTS (%)	Our (%)
1	17.3	82.7	84.9	**85.5**
2	52.2	74.5	80.6	**83.7**
3	19.1	59.9	70.7	**71.7**
4	6.6	42.4	50.0	**51.5**
5	3.4	**44.1**	38.2	38.2
6	0.9	**55.6**	44.4	**55.6**

Table 5: Accuracy for increasing length of templates. *#Op* is the number of operators in expressions. *Pro* denotes the proportion of MWPs for different expression lengths.

4.2.4 Impact of Numerical Comparison

One of the primary goals of our *Graph2Tree* model is to address the situation where the wrong arithmetic order leads to incorrect solution expression generation. We evaluate this aspect of our model by investigating how *Graph2Tree* has improved the arithmetic order errors. We first retrieve the MWPs with incorrectly predicted expressions. As we are interested in arithmetic order errors, we check that the incorrectly predicted expressions' length is equal to their corresponding ground truth

Case 1: The class organized students to climb the mountain. The female students were divided into 4 groups, and each group had 15 students. There were 76 male students in total. How many students joined climbing last week?	
GTS: $(15 + 76) * 4$; **(error)**	**Graph2Tree**: $15 * 4 + 76$;
Case 2: Lingling and Yaya are 200 meters apart. Lingling is in the front and runs 3 meters per second. Yaya is in the rear and runs 5 meters per second. They set off at the same time, running in the same direction. How long will it be before Yaya could catch up with Lingling?	
GTS: $200/(3 - 5)$; **(error)**	**Graph2Tree**: $200/(5 - 3)$;
Case 3: A bus and a truck departed from the two cities of A and B, which are 900 kilometers apart. They went in opposite directions. It takes 10 hours for the bus to travel from A to B, and 15 hours for the truck to travel from B to A. How many hours would it be before the bus, and the truck meet?	
GTS: $900/(900/10 + 1/15)$; **(error)**	**Graph2Tree**: $900/(900/10 + 900/15)$;

Table 6: Three examples of solving MWPs with our *Graph2Tree* model.

expressions' length. In total, we retrieved 103 incorrect predicted expressions for *Graph2Tree* and 119 for GTS. Next, we manually count the number of incorrectly predicted expression attributed to arithmetic order error among the initially retrieve set. We found that *Graph2Tree* has generated 7 expressions with arithmetic order error, while GTS has generated 27 arithmetic order error expressions. This suggests that *Graph2Tree* is able to significantly improve the arithmetic order in MWP task.

4.3 Case Study

Finally, we perform a case study on the solution expressions generated by GTS and *Graph2Tree*. Selected case studies are shown in Table 6. In Case 1, there are essential words, i.e., "each," "group," and "students" around the quantity "15", and "students" around the quantity "76". However, GTS predicts operator "+" between these two quantities with obviously different units as GTS is unable to model quantity representation effectively using BiLSTM. For the second case, we observe that GTS gives a wrong prediction "3 – 5" as GTS does not model quantities' numerical qualities. For the last case, this MWP requires models to have the ability to handle situation where quantities are repeatedly and frequently used. *Graph2Tree* is able to handle this situation better than the GTS model as our model encodes the MWP in richer graph representation. The three case studies demonstrate *Graph2Tree* model strengths in generating more accurate and realistic solution expressions for MWPs.

Besides, further analysis is performed on error cases. We found that our model, like other baselines, performed poorly in solving MWPs with long solution expressions. Answering these MWPs requires complex reasoning which opens the possibility for future works.

5 Related Work

5.1 Math Word Problems Solving

The earlier works on math word problems (MWPs) are mainly tested on small-scale datasets. These works can be broadly divided into statistical machine learning based (Kushman et al., 2014; Hosseini et al., 2014; Mitra and Baral, 2016; Roy and Roth, 2018; Zou and Lu, 2019a) and semantic parsing based (Shi et al., 2015; Koncel-Kedziorski et al., 2015; Roy and Roth, 2015; Huang et al., 2017; Zou and Lu, 2019b).

Recently, deep learning based models have become a new trend in solving math word problems. Wang et al. (2017) applied a vanilla seq2seq model to map the language text to an expression. Li et al. (2019) applied multi-head attention to model different types of MWP features. Both Wang et al. (2018) and Chiang and Chen (2019) proposed to generate expressions with the implicit tree structure. Huang et al. (2018) designed a new intermediate form to generate. Other models (Wang et al., 2019; Liu et al., 2019; Xie and Sun, 2019) have generated an expression tree explicitly to derive the final answer.

5.2 Graph Transformer

Transformer is a self-attention based neural network which has shown potential in tasks like neural machine translation (Vaswani et al., 2017) and language modeling (Devlin et al., 2019). However, there are only a fewer works which focus on extension of transformer to graph-structure data. In community of natural language processing, the first graph transformer was introduce in a knowledge-graph-to-text task (Koncel-Kedziorski et al., 2019), where a graph attention Network (Veličković et al., 2018) is used with a transformer style architecture. Another graph transformer (Cai and Lam, 2019) extends vanilla multi-head attention mechanism into relation-enhanced global attention mechanism. Our work aims to explore the adaptation of transformer in modeling multiple heterogeneous graph in parallel for the MWP task.

6 Conclusion

In this paper, we proposed a novel MWP solver, *Graph2Tree*, which improves the task performance by enriching the quantity representations in the problem. We conducted extensive experiments to evaluate our model against state-of-the-art baselines. Our experiments shown that *Graph2Tree* is able to outperform the baselines on the MWP task. For future work, we aim to consider more complex relationships among the quantities and other attributes to enrich quantity representations further. We will also explore adding heuristic in the tree-based decoder to guide and improve the generation of solution expression.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61832001 and No. 61672133), Sichuan Science and Technology Program (No. 2019YFG0535 and No. 2018GZDZX0032) and the National Research Foundation, Prime Ministers Office, Singapore under its International Research Centres in Singapore Funding Initiative.

References

D. Bobrow. 1964. Natural language input for a computer problem solving system. pages 146–226.

Deng Cai and Wai Lam. 2019. Graph transformer for graph-to-sequence learning. *CoRR*, abs/1911.07470.

Ting-Rui Chiang and Yun-Nung Chen. 2019. Semantically-aligned equation generation for solving and reasoning math word problems. In *NAACL-HLT*, 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 *NAACL-HLT*, pages 4171–4186.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, pages 523–533.

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.

Danqing Huang, Jin-Ge Yao, Chin-Yew Lin, Qingyu Zhou, and Jian Yin. 2018. Using intermediate representations to solve math word problems. In *ACL*, pages 419–428.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. 2019. Text generation from knowledge graphs with graph transformers. In *NAACL-HLT*, pages 2284–2293.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *TACL*, 3:585–597.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *NAACL*, pages 1152–1157.

Nate Kushman, Luke Zettlemoyer, Regina Barzilay, and Yoav Artzi. 2014. Learning to automatically solve algebra word problems. In *ACL*, pages 271–281.

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 *ACL*, pages 6162–6167.

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

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL*, pages 55–60.

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *ACL*.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. NumNet: Machine reading comprehension with numerical reasoning. In *EMNLP-IJCNLP*, pages 2474–2484.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *EMNLP*, pages 1743–1752.

Subhro Roy and Dan Roth. 2018. Mapping to declarative knowledge for word problem solving. *TACL*, 6:159–172.

Sunil Kumar Sahu, Fenia Christopoulou, Makoto Miwa, and Sophia Ananiadou. 2019. Inter-sentence relation extraction with document-level graph convolutional neural network. In *ACL*, pages 4309–4316.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *EMNLP*, pages 1132–1142.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR*.

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

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, 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.

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

Yanyan Zou and Wei Lu. 2019a. Quantity tagger: A latent-variable sequence labeling approach to solving addition-subtraction word problems. In *ACL*, pages 5246–5251.

Yanyan Zou and Wei Lu. 2019b. Text2Math: End-to-end parsing text into math expressions. In *EMNLP-IJCNLP*, pages 5326–5336.