Hierarchical Attention Decoder for Solving Math Word Problems

Anonymous ACL submission

Abstract

To answer math word problems (MWPs), models need to formalize equations from the source text of math problems. Recently, the tree-structured decoder has significantly improved model performance on this task by generating the target equation in a tree format. However, current decoders usually ignore the hierarchical relationships between tree nodes and their parents, which hinders further improvement. Thus, we propose a structure called hierarchical attention tree to aid the generation procedure of the decoder. As our decoder follows a graph-based encoder, our full model is therefore named as Graph to Hierarchical Attention Tree (G2HAT). We show a tree-structured decoder with hierarchical accumulative multi-head attention leads to significant performance improvement and reaches new state-of-the-art (SOTA) on both English MAWPS and Chinese Math23k MWP benchmarks. For further study, we also apply pre-trained language models for G2HAT, which even results in new higher performance.

1 Introduction

Math Word Problems (MWPs) require models to automatically give an answer to a math problem described by natural language text, which is called word context in a formal MWP. Presented in Figure 1, based on a word context, models have to infer an equation from it and calculate the final solution. Since the introduction of machine learning methods in the language processing (NLP) field, much effort has been spent designing features to train models to solve MWP (Kushman et al., 2014; Roy and Roth, 2018; Shi et al., 2015). However, these models suffer from low scalability, as they require hand-made features designed by humans.

In recent years, there has been a booming trend of the application of deep learning methods to MWP, among which seq2seq models apply an encoder to encode the word context into an intermediate representation for a decoder to sequentially generate parts of the equation. To take advantage of the tree-structured nature of equations for MWP, (Xie and Sun, 2019) introduces a tree-structured decoder that improves model performance significantly. Their tree-structured decoder generates a tree for prefix notation by generating tree nodes recursively while considering their parents and siblings.

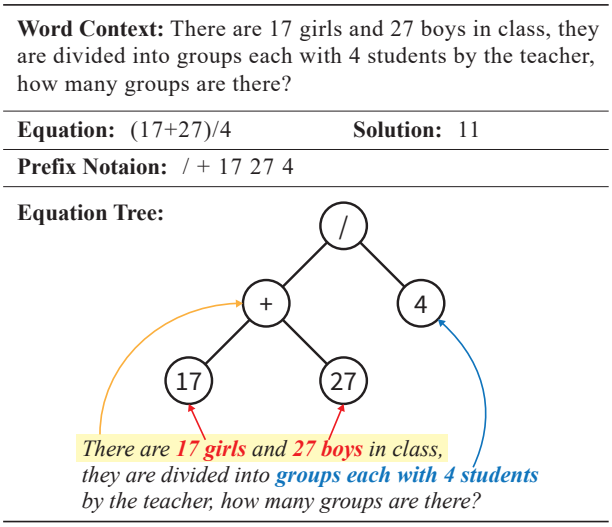

Figure 1: An example for MWP.

Efficient though the tree-structured decoder is, it ignores the hierarchical nature of nodes in the generated equation tree. Consider the equation tree from Figure 1 for example. To induce the + operator, the model should first pay attention to the span *"There are 17 girls and 27 boys in class,"* and then induce the 17 and 27 by attending to *"17 girls"* and *"27 boys"* inside the span.

From the example above, we obtain two conclusions. First, models have to attend to more words to decode a parent node than its children. A more comprehensive understanding of the word context is required for inducting nodes in a higher hierarchy. This property is named as **hierarchical attention decay** in an equation tree. Second, word context

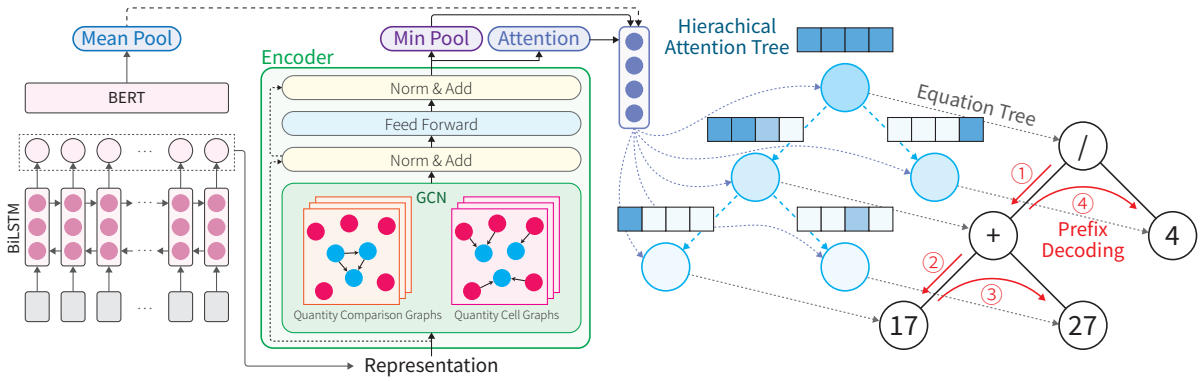

Figure 2: Our G2HAT model.

covered by children's attention is contained in context covered by parent's attention. We call this property **hierarchical attention succession** in an equation tree.

Previous tree-structured decoders fail to exploit these two key properties of parent-child relationships in the equation tree. They score attention based just on the input features of the current node for decoding without considering the attention score on its parent. We thus propose Hierarchical Attention Tree (HAT) decoder which is capable of implementing hierarchical attention on both hierarchical decay and hierarchical succession. Specifically, we allow nodes to succeed attention from their parents during decoding. Our full model lets the HAT decoder follow a graph-based encoder, thus we overall build a Graph to Hierarchical Attention Tree model to solve MWPs with an encoder-decoder framework.

We conduct experiments on two MWP benchmarks on English and Chinese, respectively, MAWPS and Math23k. Multi-head attention is added for better information extraction from the context with an attention mechanism. We show that our model outperforms the previous SOTA by 1.7 solution matching accuracy score on MAWPS and 2.6 on Math23k, which sets the new SOTA for both MWP benchmarks. With further improvement from the application of pre-trained language model, G2HAT even reaches a significant 3.3 score improvement on MAWPS and a 3.2 score improvement on Math23k.

2 Related Work

Math Word Problem Early works on machine learning for MWP concentrate on statistical models (Kushman et al., 2014; Hosseini et al., 2014; Mitra and Baral, 2016; Roy and Roth, 2018; Zou and Lu, 2019a) and semantic information (Shi et al., 2015; Koncel-Kedziorski et al., 2015; Roy and Roth, 2015; Huang et al., 2017; Zou and Lu, 2019b) for sequence construction and solution inference. (Zou and Lu, 2019a) models MWPs as path searching problems for token sequences and uses a model to solve this modified problem based on statistics. (Zou and Lu, 2019b) models MWPs as semantic parsing tasks and defines special operators to build a Text-Math Tree. The Text-Math Tree is interpreted into common equation to produce the final solution.

The encoder-decoder structure now dominates the realm of solving MWP with deep learning. An early work (Wang et al., 2018a) directly applies a Seq2Seq scenario to generate a sequence representing equation from the context. (Li et al., 2019a) adds multi-head attention to solve MWPs. Recently, a major improvement for MWPs is the introduction of the tree-structured decoder (Xie and Sun, 2019), which exploits the structural nature of MWPs and improves model performance significantly. For further exploration, (Hong et al., 2021) used a weakly supervised model that corrects the generation process by back-searching for faults from the result node. (Cao et al., 2021) introduces a DAG-structured decoder. (Lin et al., 2021) applies hierarchical attention for the encoder and uses sequential, additive clause attention and word attention are added sequential and (Zhang et al., 2020) implements a dependency-based graph encoder.

Hierarchical Modeling Hierarchical models are commonly used explicitly or implicitly for NLP tasks to facilitate multiple round classification (Aly et al., 2019) or inform models of cross-round relations among representations. For instance, (Fan

et al., 2018) applies a tree structure for story generation to explicitly represent story topics. (Chen et al., 2020) models entities hierarchically for model to enable the model's entity typing. Combining hierarchical models is a popular method for solving NLP tasks, and has been used in the hierarchical Transformer (Liu and Lapata, 2019) and hierarchical BERT (Zhang et al., 2019).

The tree-structured or graph-structured decoder is a typical style of hierarchical modeling. Aside from MWP, tree-structured decoder also leveraged for generating math expressions or markup from handwriting or images (Ma et al., 2019b; Zhang et al., 2021). Based on dependency parsing trees, the tree-structured decoder can also be applied for neural machine translation (Choshen and Abend, 2021). The tree-structured decoder performs much better than other models when generating structured data like expressions and codes (Wang et al., 2018c; Xie et al., 2021), as it captures the natural tree structure of sequences during generation.

For attention scoring, hierarchical models are generally used to score attention in different levels (Miculicich et al., 2018; Zhao et al., 2018; Wang et al., 2018b; Liu and Chen, 2019), such as the word level, the sentence level and the document level. Some others integrate multiple attention mechanisms for better representation (Luo et al., 2018; Ma et al., 2019a). Our model differs from those previous hierarchical attention-based model by modeling hierarchical attention for the tree structure. Our design enables attention succession and complement to further adapt hierarchical attention to trees, which is a rather more complex structure for modeling multi-level hierarchical attention.

Multi-head Attention First introduced with the Transformer (Vaswani et al., 2017) model, multi-head attention has been successfully applied in a large variety of NLP domains. (An et al., 2020) tries to interpret multi-head attention as Bayesian inference and (Iida et al., 2019) further develops multi-head attention into multi-hop attention for performance improvement on machine translation. Multi-head attention can also be used for knowledge reasoning tasks (Paul and Frank, 2020) and multi-modal training (Wang et al., 2020).

Recent work has shown that multi-head attention can be successfully applied for MWP solving. GROUP-ATT model (Li et al., 2019b) applies Transformer as the encoder for information extraction to solve MWPs. GROUP-ATT explicitly assigns self-attention to represent different types of attention. The result from GROUP-ATT has shown that all these attention mechanisms contribute to final performance improvement. Our study, instead, with the multi-head attention mechanism for the decoder and in a hierarchical tree structure, which differs from our previous study that only exploits multi-head attention in the encoder.

3 Method and Model

3.1 Graph Encoder

For the encoder, we follow (Zhang et al., 2020) to use a quantity graph to encode the word context of MWP by constructing a quantity graph based on it. Specifically, for a sentence $W = \{w_1, w_2, \cdots, w_n\}$, we first use rules to extract quantity tokens $Q = \{q_1, q_2, \cdots, q_m\}$ from it. Then a dependency parser is used to extract related non-quantity tokens and connect them to quantity tokens in the graph. A quantity token and its related tokens with edges between them are called a quantity cell, which is used when building quantity graphs for encoding in graph encoder.

With quantity cells extracted, we build two quantity graphs:

• **Quantity Cell Graph** is a combination of graphs of all quantity cells that are isolated from each other and allow related words to pass information to the quantity.

• **Quantity Comparison Graph** only contains quantity tokens as nodes. Edges are built based on partial ordering relations between quantity nodes. Specifically, a directed edge $e = (q_1, q_2)$ is built when $q_1 > q_2$.

Based on Quantity Cell Graphs $G^{Qcell} = \{G_i^{Qcell}\}$ where $i = 1, 2, \cdots, k_{Qcell}$ and Quantity Comparison Graphs $G^{Qcomp} = \{G_i^{Qcomp}\}$ where $i = 1, 2, \cdots, k_{Qcomp}$, A represents the adjecent matrix for a certain graph G where $A_{i,j} = 1$ notating there is an edge from i to j and $A_{i,j} = 0$ otherwise. The encoding procedure for MWP context is formulated as follows,

$$
\begin{aligned}
X &= \text{Embed}(W) \\
X &= \text{RNN}(X) \\
X &= \text{TransLayer}_i(X) \; for \; i = 1, 2, \cdots N \\
H &= \text{MinPooling}(X)
\end{aligned}
\tag{230}
$$

Here, the sentence is first embedded via an embedding layer before it is fed into a recurrent neural

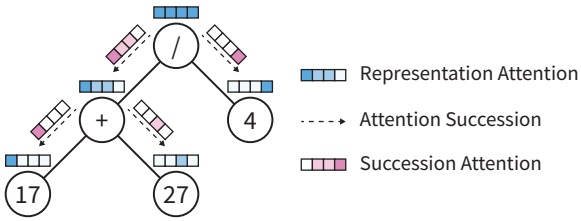

Figure 3: The procedure of attention succession from top to bottom in hierarchical attention tree.

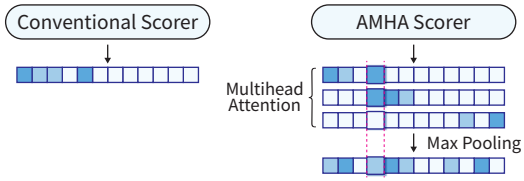

Figure 4: Comparison between the conventional attention scorer and our accumulative multi-head attention scorer.

network, such as a bidirectional long short-term memory (BiLSTM) network or a gated recurrent unit (GRU) network for contextual representations. Then the sentence is further processed by multiple Transformer layers that are defined as follows,

$$\begin{aligned}\text{TransLayer}(X) &= \text{GCN}(X) + \text{LN}(GCN(X)) \\ \text{GCN}(X) &= \text{FFN}(\underset{A\ for\ G}{||}(\text{LN}(AX) + X)) \\ G &= G^{Qcell} \cup G^{Qcomp}\end{aligned}$$

where LN refers to layer normalization. Finally, the output $Y \in \mathbb{R}^{n \times d}$ from graph encoder is pooled into hidden representation $H \in \mathbb{R}^{d}$ by token-level min pooling for decoder to process and generate the equation tree.

3.2 Hierarchical Attention

We give an elaborate description of our hierarchical attention mechanism in the HAT-structured Decoder in this section. Our decoder generates an equation tree following a hierarchical structure from top to bottom with attention succeeded from parent to child. In practice, we first generate the root operator node, then its left sub-tree and right sub-tree, each with part of its parent's attention on the encoded representation. This procedure recurses for each sub-tree until the sub-tree is a single quantity node.

As in Figure 3, the root operator node attends fully to the whole encoded representation. When we build the left and right sub-tree, we pass the representation down to them via an attention succession process. However, the sub-trees will only attend on part of the representation from its parent based on attention scores (succession attention) which are calculated during the succession process. Thus we can see how our hierarchical attention mechanism satisfies the two key properties for better equation tree construction.

• **Hierarchical Attention Decay** A child node in a hierarchical attention tree can only focus on part of the encoded representation that is attended by its parent. As a result, the attention will decay hierarchically during the succession from top to bottom.

• **Hierarchical Attention Succession** Likewise, the partial attention of child nodes will always be covered by their parents, which leads to succession relationships.

3.3 Accumulative Multi-head Succession Attention

To better exploit the hierarchical attention mechanism, we apply a novel attention mechanism which we call accumulative multi-head attention (AMHA) to score succession attention to pass representation attention. As shown in Figure 4, different from conventional attention scorer, the AMHA scorer will use a multi-head input to score multiple attention scores for all heads. Then, those attention scores are accumulated together via a max-pooling process.

$$\begin{aligned}A_i^k &= \frac{\exp(\text{Scorer}^k(X_i^k))}{\sum_{j=1}^{n}\exp(\text{Scorer}^k(X_j^k))} \\ A_i &= \max(A_i^1, A_i^2, \cdots, A_i^n)\end{aligned}$$

where $X_i^k \in$ refers to the hidden representation of the word in i-th position and k-th head. Each scorer passes X_i^k through a linear layer and then uses a softmax function to get attention A^k on k-th head. These attention scores are finally pooled via max-pooling to get our AMHA score.

AMHA is intuitively better than conventional attention mechanism for MWP solving due to the following two reasons:

• **Multiple Concentration** When constructing an equation tree, an operator node should not only contain information to induct its type, but it should also integrate information for its children, the nodes in its left sub-tree and right sub-tree. Thus, AMHA is a better vector for trees' information as different

heads are responsible to attend to different parts of the encoded representation.

• **Steady Attention Decay** For hierarchical attention tree building, attention succession is a key procedure for better performance. However, using a conventional attention mechanism will lead to an attention succession that decays too fast for performance improvement.

Here is an easy example: suppose there are two positions i and j for which a parent node wants to memorize for its children. This parent node will be encouraged to assign attention scores close to 1 for them. However, conventional attention mechanisms only allow attention scores with sums of 1, which means at least half of the attention will decay for either position. This will certainly hurt the efficiency of attention succession. AMHA can easily fix this problem by applying two heads for both positions for a parent node to succeed integrative attention for its children. This allows AMHA to preserve attention succession which is crucial for hierarchical attention tree building.

3.4 Complement Attention

To capture all necessary information for building trees and further improve the capacity of our model, we introduce another complementary attention mechanism to replenish attention scores that are dropped in previous rounds of attention succession. We once again apply AMHA for attention scoring. For a node during tree building, A^{parent} refers to the attention of the node's parent, and X refers to the input representation to induct succession and complement attentions.

$$SA = \text{AMHA}^{succession}(X)$$
$$CA = \text{AMHA}^{complement}(X)$$

We then use max pooling to get the final attention on the child node.

$$A_i^{child} = \max(SA_i * A_i^{parent}, CA_i)$$

Integrating complement attention can lead to further improvement compared to using only succession attention as complement attention allows child nodes to replenish their information to themselves without harming the hierarchical attention decay or succession properties. Also, the application of complement attention can prevent node attention from degrading to zero, which guarantees a slower attention decay for better tree building.

3.5 HAT-structured Decoder

Our HAT-structured decoder decodes the output H from encoder following the tree-structured decoder's procedure (Xie and Sun, 2019) with accumulative multi-head attention based attention succession process. Starting with a root representation q^{ROOT} initialized based on H and an all-ones root attention vector a^{ROOT}, we predict the label of the root node and generate the rest equation tree via four sub-modules of decoder, the Left Label and Attention Scorer (LLAS), Right Label and Attention Scorer (RLAS), Sub-Tree Encoder (STE) and Node Classifier (NC).

Here, LLS and RLS use AMHA to score succession attention. LLAS and RLAS are of gated structures to process input representations while NC applies a linear classifier for operator and constant label and pairwise classifier for pairs between the input representation and encoded quantity representations. These details are omitted in this paper since they are not contributions of this paper and can be found in detail in the implementation of tree-structured decoder of (Xie and Sun, 2019).

Step 1. Representation Attention Integration With current root representations and attention, we first use element-wise multiplication to integrate attention scores with encoded representations.

$$\hat{H}_i = H_i \times a_i^{ROOT}$$

Step 2. Left Sub-tree Generation LLAS generates the label representation, node representation, and succession attention for the left child node of the current root node. Then, NC is used to classify the label of the left child node.

$$e^l, q^l, SA^l, CA^l = \text{LLAS}(q^{ROOT}, \hat{H})$$
$$a_i^l = a_i^{ROOT} \times SA_i^l + CA_i^l$$
$$l^l = \text{NC}(e^l)$$

If e^l is classified to be an operator label, jump to **Step 1** with current root node representation set to q^l and root representation attention set to a^l to generate the left sub-tree. This procedure returns when the left sub-tree is completely constructed.

Step 3. Right Sub-tree Generation We first integrate representation of the left sub-tree with the root representation via STE. The label representation, node representation, and succession attention are then generated by RLAS for the right child

node of the current root node. The same NC for the left child node is applied to finally label the right child node.

$$
\begin{aligned}
t^l &= \text{STE}(e^l, q^l); \\
e^r, q^r, SA^r, CA^r &= \text{RLAS}(q^{ROOT}, \hat{H}, t^l); \\
a_i^r &= a_i^{ROOT} \times SA_i^r + CA_i^r; \\
l^r &= \text{NC}(e^r)
\end{aligned}
$$

Likewise, if e^r is classified to be an operator label, jump to **Step 1** with current root node representation set to q^r and root representation attention set to a^r to generate the right sub-tree. The procedure ends when the right sub-tree is completely constructed.

3.6 Pre-trained Language Model

For encoding word context, we first encode the sentence with the pre-trained language model (PLM) and then pool it into a context representation Q with mean pooling on the word level.

$$
P = \text{PLM}(W) \qquad Q = \frac{1}{n}\sum_{i}^{n}(P_i)
$$

Finally, Q from PLM and H from graph encoder are added together to get the final context representation.

$$
H' = H + Q
$$

4 Experiment

4.1 Experiment Setting

Dataset We conduct experiments on two MWP datasets, MAWPS and Math23K.

• **MAWPS** is an English MWP dataset with 2373 problems, where both template equation and final solution are provided for training and testing.

• **Math23K** is a Chinese MWP dataset with 23162 problems, where only equations for real quantities in the context are provided.

Model We use an embedding size of 128 dimensions and a hidden size of 512 dimensions for word representation. For RNNs, we use 4-layer LSTM or GRU for encoding. When calculating AMHA for succession attention, we use 4 attention heads to accumulate attention scorers. The beam size is 5 to search for the most likely equation to induct a more plausible solution.

Model	MAWPS	Math23K	
		Test	5-fold Valid
DNS	59.5	-	58.1
Math-EM	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	-
IRE	-	76.7	-
HMS	80.3	-	76.1
GTS	82.6†/78.6‡	75.6	74.3
G2T(Zhang et al., 2020)	83.7	77.4	75.5
G2T*	83.7	77.0	75.3
G2HAT(GRU)	85.2(↑1.5)	**80.0(↑2.6)**	76.6(↑1.3)
G2HAT(LSTM)	**85.4(↑1.7)**	79.2(↑1.8)	**77.4(↑2.1)**
G2HAT+BERT	**87.0(↑3.3)**	**80.9(↑3.5)**	**78.0(↑2.7)**

Table 1: Comparison of solution matching accuracy on MWP datasets. †: Result of GTS from G2T. ‡: Result of GTS from HMS. *: Re-implementation of previous model.

Pre-trained Language Model We apply BERT-base-cased and BERT-base-Chinese as PLM for MAWPS and Math23k respectively, we do not finetune the PLMs during training as the training datasets are rather small.

Training We use cross entropy loss for model optimization. The batch size for training is set to 64, and we use Adam optimizer (Kingma and Ba, 2015) with initial learning rate 10^{-3} and decay rate 10^{-5} for parameter updating. We train the model for 200 epochs in each experiment and save the model with the highest solution accuracy.

Model	MAWPS	Math23K	
		Test	5-fold Valid
G2T*	82.8	65.8	64.3
G2HAT(GRU)	84.3	**68.0**	65.6
G2HAT(LSTM)	**84.4**	67.3	**65.7**
G2HAT+BERT	**85.7**	**69.0**	**66.5**

Table 2: Comparison of equation matching accuracy on MWP datasets. *: Re-implementation of previous model.

4.2 Result

Tables 1 and 2 show the comparison between our proposed G2HAT model and previous SOTA models. We first re-implement the Graph2Tree model (Zhang et al., 2020), and find the result slightly lower than the reported result but still competitive for comparison. We then run our G2HAT on MWP datasets for GRU and LSTM encoders and find our AMHA-based HAT-structured decoder leads to significant improvement on both the English

and Chinese datasets. For solution matching accuracy, our model results in 1.7 an accuracy score improvement on the English MAWPS dataset and a 2.6 accuracy score improvement on the Chinese Math23K dataset.

The equation matching accuracy is evaluated as it reflects the capability for pattern discerning. The results in Table 2 can explain our model's source of improvement. Our G2HAT model boosts the baseline G2T model by 1.7 accuracy score on MAWPS and 2.6 accuracy score on Math23K. As our model enjoys both hierarchical attention decay and succession for hierarchical tree building, better equation trees are produced for higher solution matching accuracy. Moreover, results from 5-fold validation on Math23k show the robustness of our results, as G2HAT outperforms G2T by 1.9 accuracy score on solution matching and 1.4 accuracy score on equation matching.

For encoder choice, we can see LSTM outperforms GRU on MAWPS but performs worse on Math23k. We attribute this to the vocabulary difference. For MAWPS, the vocabulary is smaller, thus most words in the training dataset will still occur in the test dataset. A preciser encoder like LSTM will better capture details of training data. However, Math23k covers problems in all kinds of mathematics applications, resulting in a much large and unusual vocabulary, which makes LSTM easily overfitting, while GRU can resist such an overfitting risk with its simpler structure than LSTM.

After features from BERT are also incorporated for context representations, the improvement of G2HAT reaches 3.3 for the English MAWPS dataset and 3.5 for the Chinese Math23k dataset on solution matching. Correspondingly, respective improvements of 2.9 and 3.2 are achieved on MAWPS and Math23k datasets for equation matching, which proves the efficiency of BERT integration.

5 Analysis and Discussion

5.1 Ablation Study

Table 3 shows the results of the ablation study for our G2HAT model. We first run G2HAT with succession attention (SA) or complement attention (CA) removed to build equation trees. The drops in both results verify the contribution from both mechanisms to the model performance. The comparison also shows that succession attention makes a larger contribution than complement attention.

Model	MAWPS		Math23K	
	EM	SM	EM	SM
G2HAT	**84.4**	**85.4**	68.0	**80.0**
-CA	84.3	85.1	**68.8**	79.1
-SA	83.7	84.4	67.6	78.7
-AMHA	83.2	84.1	66.9	77.7
G2T	82.7	83.7	65.8	77.0

Table 3: Ablation studies on MWP datasets with only succession attention, CA: complement attention. SA: succession attention. G2T equals to G2HAT with all these mechanisms removed.

Moreover, G2HAT with only succeeding attention achieves a higher equation matching accuracy score but less solution matching accuracy than does the full model. This indicates that G2HAT with only succeeding attention better reconstructs equation trees in the way taught by the training dataset but is not flexible enough when facing new MWP problems.

We also trained the model without accumulative multi-head attention and unsurprisingly encounter a drop in performance, though there is still an improvement on the baseline G2T model. Thus we can see AMHA is a critical attention mechanism that supports the efficacy of our SA and CA procedure by avoiding a too fast attention decay which may hinder parent-child attention succession.

5.2 Attention Distribution Learning

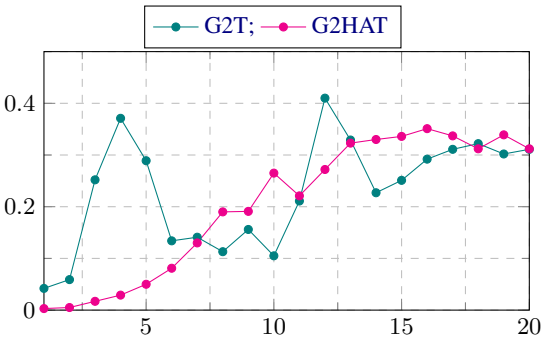

Figure 5: KL Divergence between Siblings vs. Training Epoch

Focusing on modeling hierarchical attention with attention succession and complementing, we next explore how our model learns to pass the attention score from parent nodes to child nodes. Here we experiment on MAWPS to evaluate the attention distribution learning of our G2HAT model, comparing with G2T baseline. We use KL divergence to represent the variation between two attention distri-

Q1: Tom went to 4 hockey games this year, but missed 7. He went to 9 games last year. How many hockey games did Tom go to in all?
G2T: 4 + 7 = 11 **G2HAT:** 4 + 9 = **13** **Gold:** 4 + 9 = **13**
Q2: Evelyn has 95 marbles. She gets 9 more from Henry. Later, Evelyn buys 6 cards at the store. How many marbles does Evelyn have in all?
G2T: 95 + 9 + 6 = 110 **G2HAT:** 95 + 9 = **104** **Gold:** 95 + 9 = **104**
Q3: Tom bought 40 tickets at the state fair. He spent 28 tickets at the 'dunk a clown' booth and decided to use the rest on rides. If each ride cost 4 tickets, how many rides could he go on?
G2T: 4 * (40 - 28) = 48 **G2HAT:** (40 - 28) / 4 = **3** **Gold:** (40 - 28) / 4 = **3**

Table 4: Case study for MWP solving capacity comparison between G2T and G2HAT.

butions on two sibling nodes. A good learning process for attention distribution should be in a steady increasing trend as sibling nodes should attend on related but still different information. As attention scores on deeper nodes become rather small due to hierarchical attention decay, we first normalize the attention distribution and then compute the KL divergence:

$$\hat{a}_i^{left} = \frac{a_i^{left}}{\frac{1}{n}\sum_{j=1}^{n}\hat{a}_j^{left}} \quad \hat{a}_i^{right} = \frac{a_i^{right}}{\frac{1}{n}\sum_{j=1}^{n}a_j^{right}}$$

$$\begin{aligned} D =& D_{KL}(\hat{a}^{left}||\hat{a}^{right}) + D_{KL}(\hat{a}^{right}||\hat{a}^{left}) \\ =& \sum_{i=1}^{n}\log(\hat{a}_i^{left})\log(\frac{\hat{a}_i^{left}}{\hat{a}_i^{right}}) + \\ & \sum_{i=1}^{n}\log(\hat{a}_i^{right})\log(\frac{\hat{a}_i^{right}}{\hat{a}_i^{left}}) \end{aligned}$$

From Figure 5, we can see the learning curve of KL divergence for the G2T baseline and our G2HAT model. We can see that, though both learning curves converge to about 0.3 KL divergence, the training of G2T model fluctuates a lot, which might hinder the decoder from correctly allocating attention to sibling nodes. Our G2HAT model, instead, keeps a rather steady learning curve thanks to two attention mechanisms, and consequently, better model performance when building equation trees.

5.3 Case Study

Table 4 shows our case study for our G2HAT model's performance solving MWPs in comparison with the G2T baseline. From the results for Question 1, G2HAT model successfully selects quantity 9, representing the games Tom went to last year to add to the number of games Tom went to this year, while G2T is misled by quantity 7, which represents the number of games that Tom missed. Thus, we can see that attention following a hierarchical tree structure can indeed help the model to focus more on relevant information since G2HAT model attends according to the selection of both parent and child.

In Question 2, G2T models add all the three quantities in question context together because the attention for word context representation is chosen independently when building each node. As a result, the seemingly relevant quantity 6 confuses the model as it appears to be the number of tokens necessary for counting all the marbles. Our G2HAT is immune to this problem as such irrelevant information is filtered during multiple rounds of hierarchical attention decay.

In Question 3, though G2T model solves a subproblem with the sub-tree *40 - 28*, it fails to understand the whole hierarchical structure of the equation tree and multiplies the result of this sub-tree by 4. In comparison, being aware of the hierarchical nature of the equation tree, our G2HAT model has learned to first construct a division relation and then build the subtraction sub-tree. Therefore we can conclude that the hierarchical nature of the equation tree can be better captured by a hierarchical structure motivated model than a model that scores attention on context each time independently.

6 Conclusion

In this paper, we propose a novel hierarchical attention tree-structured (HAT) decoder that satisfies the two intuitive properties for equation tree building. The HAT-structured decoder allows child nodes to succeed attention via a succession attention scorer. This scorer selects the child's parent's attention representation, enabling hierarchical attention decay and succession in the tree structure. For better attention succession, accumulative multi-head attention is further incorporated to avoid a too fast attention decay. Experiments on MWP benchmarks show that the use of our HAT decoder leads to a large performance margin between previous SOTA on equation matching accuracy, together with a significant improvement in solution matching accuracy.

References

Rami Aly, Steffen Remus, and Chris Biemann. 2019. Hierarchical multi-label classification of text with capsule networks. In *ACL*, pages 323–330.

Bang An, Jie Lyu, Zhenyi Wang, Chunyuan Li, Changwei Hu, Fei Tan, Ruiyi Zhang, Yifan Hu, and Changyou Chen. 2020. Repulsive attention: Rethinking multi-head attention as bayesian inference. In *EMNLP*, pages 236–255.

Yixuan Cao, Feng Hong, Hongwei Li, and Ping Luo. 2021. A bottom-up DAG structure extraction model for math word problems. In *AAAI*, pages 39–46.

Tongfei Chen, Yunmo Chen, and Benjamin Van Durme. 2020. Hierarchical entity typing via multi-level learning to rank. In *ACL*, pages 8465–8475.

Leshem Choshen and Omri Abend. 2021. Transition based graph decoder for neural machine translation. *CoRR*, abs/2101.12640.

Angela Fan, Mike Lewis, and Yann N. Dauphin. 2018. Hierarchical neural story generation. In *ACL*, pages 889–898.

Yining Hong, Qing Li, Daniel Ciao, Siyuan Huang, and Song-Chun Zhu. 2021. Learning by fixing: Solving math word problems with weak supervision. In *AAAI*, pages 4959–4967.

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.

Shohei Iida, Ryuichiro Kimura, Hongyi Cui, Po-Hsuan Hung, Takehito Utsuro, and Masaaki Nagata. 2019. Attention over heads: A multi-hop attention for neural machine translation. In *ACL*, pages 217–222.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

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

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. 2019a. Modeling intra-relation in math word problems with different functional multi-head attentions. In *ACL*, pages 6162–6167.

Jierui Li, Lei Wang, Jipeng Zhang, Yan Wang, Bing Tian Dai, and Dongxiang Zhang. 2019b. Modeling intra-relation in math word problems with different functional multi-head attentions. In *ACL*, pages 6162–6167.

Xin Lin, Zhenya Huang, Hongke Zhao, Enhong Chen, Qi Liu, Hao Wang, and Shijin Wang. 2021. HMS: A hierarchical solver with dependency-enhanced understanding for math word problem. In *AAAI*, pages 4232–4240.

Yang Liu and Mirella Lapata. 2019. Hierarchical transformers for multi-document summarization. In *ACL*, pages 5070–5081.

Zhengyuan Liu and Nancy Chen. 2019. Reading turn by turn: Hierarchical attention architecture for spoken dialogue comprehension. In *ACL*, pages 5460–5466.

Fuli Luo, Tianyu Liu, Zexue He, Qiaolin Xia, Zhifang Sui, and Baobao Chang. 2018. Leveraging gloss knowledge in neural word sense disambiguation by hierarchical co-attention. In *EMNLP*, pages 1402–1411.

Jing Ma, Wei Gao, Shafiq Joty, and Kam-Fai Wong. 2019a. Sentence-level evidence embedding for claim verification with hierarchical attention networks. In *ACL*, pages 2561–2571.

Zhiming Ma, Chun Yuan, Yangyang Cheng, and Xinrui Zhu. 2019b. Image-to-tree: A tree-structured decoder for image captioning. In *ICME*, pages 1294–1299.

Lesly Miculicich, Dhananjay Ram, Nikolaos Pappas, and James Henderson. 2018. Document-level neural machine translation with hierarchical attention networks. In *EMNLP*, pages 2947–2954.

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

Debjit Paul and Anette Frank. 2020. Social commonsense reasoning with multi-head knowledge attention. In *Findings of EMNLP*, pages 2969–2980.

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. *Trans. Assoc. Comput. 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 *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 *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to an expression tree. *CoRR*, abs/1811.05632.

Wei Wang, Ming Yan, and Chen Wu. 2018b. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. In *ACL*, pages 1705–1714.

Xinyi Wang, Hieu Pham, Pengcheng Yin, and Graham Neubig. 2018c. A tree-based decoder for neural machine translation. In *EMNLP*, pages 4772–4777.

Yue Wang, Jing Li, Michael R. Lyu, and Irwin King. 2020. Cross-media keyphrase prediction: A unified framework with multi-modality multi-head attention and image wordings. In *EMNLP*, pages 3311–3324.

Binbin Xie, Jinsong Su, Yubin Ge, Xiang Li, Jianwei Cui, Junfeng Yao, and Bin Wang. 2021. Improving tree-structured decoder training for code generation via mutual learning. In *AAAI*, pages 14121–14128.

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

Jianshu Zhang, Jun Du, Yongxin Yang, Yi-Zhe Song, and Lirong Dai. 2021. SRD: A tree structure based decoder for online handwritten mathematical expression recognition. *IEEE Trans. Multim.*, 23:2471–2480.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020. Graph-to-tree learning for solving math word problems. In *ACL*, pages 3928–3937.

Xingxing Zhang, Furu Wei, and Ming Zhou. 2019. HIBERT: document level pre-training of hierarchical bidirectional transformers for document summarization. In *ACL*, pages 5059–5069.

Yue Zhao, Xiaolong Jin, Yuanzhuo Wang, and Xueqi Cheng. 2018. Document embedding enhanced event detection with hierarchical and supervised attention. In *ACL*, pages 414–419.

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.