Learning to Reason Deductively: Math Word Problem Solving as Complex Relation Extraction

Zhanming Jie[♥♦], **Jierui Li**[♣♦] and **Wei Lu**[♦]
[♥]ByteDance AI Lab, [♣]University of Texas at Austin
[♦]StatNLP Research Group, Singapore University of Technology and Design
allan@bytedance.com, jierui@cs.utexas.edu, luwei@sutd.edu.sg

Abstract

Solving math word problems requires deductive reasoning over the quantities in the text. Various recent research efforts mostly relied on sequence-to-sequence or sequence-to-tree models to generate mathematical expressions without explicitly performing relational reasoning between quantities in the given context. While empirically effective, such approaches typically do not provide explanations for the generated expressions. In this work, we view the task as a *complex relation extraction* problem, proposing a novel approach that presents explainable deductive reasoning steps to iteratively construct target expressions, where each step involves a primitive operation over two quantities defining their relation. Through extensive experiments on four benchmark datasets, we show that the proposed model significantly outperforms existing strong baselines. We further demonstrate that the deductive procedure not only presents more explainable steps but also enables us to make more accurate predictions on questions that require more complex reasoning.[1]

1 Introduction

Math word problem (MWP) solving (Bobrow, 1964) is a task of answering a mathematical question that is described in natural language. Solving MWP requires logical reasoning over the quantities presented in the context (Mukherjee and Garain, 2008) to compute the numerical answer. Various recent research efforts regarded the problem as a generation problem – typically, such models focus on generating the complete target mathematical expression, often represented in the form of a linear sequence or a tree structure (Xie and Sun, 2019).

Figure 1 (top) depicts a typical approach that attempts to generate the target expression in the form of a tree structure, which is adopted in recent research efforts (Xie and Sun, 2019; Zhang et al., 2020; Patel et al., 2021; Wu et al., 2021). Specifically, the output is an expression that can be obtained from such a generated structure. We note that, however, there are several limitations with such a structure generation approach. First, such a process typically involves a particular order when generating the structure. In the example, given the complexity of the problem, the decision of generating the addition ("+") operation as the very first step could be counter-intuitive and does not provide adequate explanations that show the reasoning process when being presented to a human learner. Furthermore, the resulting tree contains identical sub-trees ("$8 \times 3 + 3$") as highlighted in blue dashed boxes. Unless a certain specifically designed mechanism is introduced for reusing the already generated intermediate expression, the approach would need to repeat the same effort in its process for generating the same sub-expression.

Question: *In a division sum , the remainder is **8** and the divisor is **6** times the quotient and is obt-ained by adding **3** to the thrice of the remainder. What is the divident?*

Answer: 129.5 **Expr**: $((8 \times 3 + 3) \times (8 \times 3 + 3) \div 6) + 8$

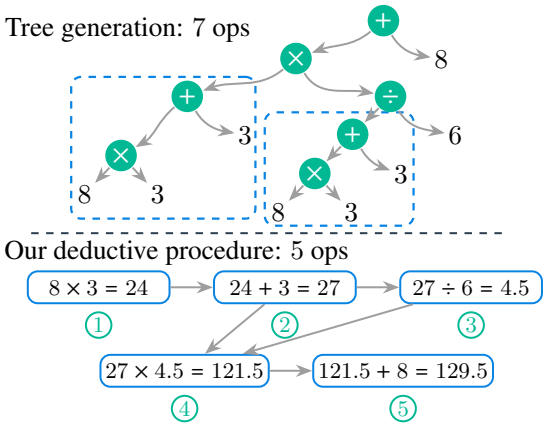

Figure 1: A MWP example taken from MathQA. Top: tree generation. Bottom: deductive procedure.

[1]Our code and data are released at https://github.com/allanj/Deductive-MWP.

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 5944 - 5955

Solving math problems generally requires deductive reasoning, which is also regarded as one of the important abilities in children's cognitive development (Piaget, 1952). In this work, we propose a novel approach that explicitly presents deductive reasoning steps. We make a key observation that MWP solving fundamentally can be viewed as a *complex relation extraction* problem – the task of identifying the complex relations among the quantities that appear in the given problem text. Each primitive arithmetic operation (such as *addition*, *subtraction*) essentially defines a different type of relation. Drawing on the success of some recent models for relation extraction in the literature (Zhong and Chen, 2021), our proposed approach involves a process that repeatedly performs relation extraction between two chosen quantities (including newly generated quantities).

As shown in Figure 1, our approach directly extracts the relation ("*multiplication*", or "×") between 8 and 3, which come from the contexts "*remainder is 8*" and "*thrice of the remainder*". In addition, it allows us to reuse the results from the intermediate expression in the fourth step. This process naturally yields a deductive reasoning procedure that iteratively derives new knowledge from existing ones. Designing such a complex relation extraction system presents several practical challenges. For example, some quantities may be irrelevant to the question while some others may need to be used multiple times. The model also needs to learn how to properly handle the new quantities that emerge from the intermediate expressions. Learning how to effectively search for the optimal sequence of operations (relations) and when to stop the deductive process is also important.

In this work, we tackle the above challenges and make the following major contributions:

- We formulate MWP solving as a complex relation extraction task, where we aim to repeatedly identify the basic relations between different quantities. To the best of our knowledge, this is the first effort that successfully tackles MWP solving from such a new perspective.
- Our model is able to automatically produce explainable steps that lead to the final answer, presenting a deductive reasoning process.
- Our experimental results on four standard datasets across two languages show that our model significantly outperforms existing strong baselines. We further show that the model performs better on problems with more complex equations than previous approaches.

2 Related Work

Early efforts focused on solving MWP using probabilistic models with handcrafted features (Liguda and Pfeiffer, 2012). Kushman et al. (2014) and Roy and Roth (2018) designed templates to find the alignments between the declarative language and equations. Most recent works solve the problem by using sequence or tree generation models. Wang et al. (2017) proposed the Math23k dataset and presented a sequence-to-sequence (seq2seq) approach to generate the mathematical expression (Chiang and Chen, 2019). Other approaches improve the seq2seq model with reinforcement learning (Huang et al., 2018), template-based methods (Wang et al., 2019), and group attention mechanism (Li et al., 2019). Xie and Sun (2019) proposed a goal-driven tree-structured (GTS) model to generate the expression tree. This sequence-to-tree approach significantly improved the performance over the traditional seq2seq approaches. Some follow-up works incorporated external knowledge such as syntactic dependency (Shen and Jin, 2020; Lin et al., 2021) or commonsense knowledge (Wu et al., 2020). Cao et al. (2021) modeled the equations as a directed acyclic graph to obtain the expression. Zhang et al. (2020) and Li et al. (2020) adopted a graph-to-tree approach to model the quantity relations using the graph convolutional networks (GCN) (Kipf and Welling, 2017). Applying pre-trained language models such as BERT (Devlin et al., 2019) was shown to significantly benefit the tree expression generation (Lan et al., 2021; Tan et al., 2021; Liang et al., 2021; Li et al., 2021; Shen et al., 2021).

Different from the tree-based generation models, our work is related to deductive systems (Shieber et al., 1995; Nederhof, 2003) where we aim to obtain step-by-step expressions. Recent efforts have also been working towards this direction. Ling et al. (2017) constructed a dataset to provide explanations for expressions at each step. Amini et al. (2019) created the MathQA dataset annotated with step-by-step operations. The annotations present the expression at each intermediate step during problem-solving. Our deductive process (Figure 1) attempts to automatically obtain the expression in an incremental, step-by-step manner.

Our approach is also related to relation extraction (RE) (Zelenko et al., 2003), a fundamental task in

$$\textbf{input:}\quad q \text{ in } \mathcal{Q}^{(0)}$$
$$\textbf{axiom:}\quad 0 : \langle q_1, \cdots, q_{|\mathcal{Q}^{(0)}|} \rangle$$
$$q_i \xrightarrow{op} q_j : \frac{t : \langle q_1, \cdots, q_{|\mathcal{Q}^{(t-1)}|} \rangle}{t+1 : \langle q_1, \cdots, q_{|\mathcal{Q}^{(t-1)}|} \mid q_{|\mathcal{Q}^{(t)}|} := e^{(t)}_{i,j,op} \rangle}$$

Figure 2: Our deductive system. t is the current step. $\langle \cdot \rangle$ denotes the quantity list.

the field of information extraction that is focused on identifying the relationships between a pair of entities. Recently, Zhong and Chen (2021) designed a simple and effective approach to directly model the relations on the span pair representations. In this work, we treat the operation between a pair of quantities as the relation at each step in our deductive reasoning process. Traditional methods (Liang et al., 2018) applied rule-based approaches to extract the mathematical relations.

MWP solving is typically regarded as one of the *system 2* tasks (Kahneman, 2011; Bengio et al., 2021), and our current approach to this problem is related to neural symbolic reasoning (Besold et al., 2017). We design differentiable modules (Andreas et al., 2016; Gupta et al., 2020) in our model (§3.2) to perform reasoning among the quantities.

3 Approach

The math word problem solving task can be defined as follows. Given a problem description $\mathcal{S} = \{w_1, w_2, \cdots, w_n\}$ that consists of a list of n words and $\mathcal{Q}_S = \{q_1, q_2, \cdots, q_m\}$, a list of m quantities that appear in $\mathcal{S}$, our task is to solve the problem and return the numerical answer. Ideally, the answer shall be computed through a mathematical reasoning process over a series of primitive mathematical operations (Amini et al., 2019) as shown in Figure 1. Such operations may include "+" (*addition*), "−" (*subtraction*), "×" (*multiplication*), "÷" (*division*), and "∗∗" (*exponentiation*).[2]

In our view, each of the primitive mathematical operations above can essentially be used for describing a specific *relation* between quantities. Fundamentally, solving a math word problem is a problem of *complex relation extraction*, which requires us to repeatedly identify the relations between quantities (including those appearing in the text and those intermediate ones created by relations). The overall solving procedure requires invoking a relation classification module at each step, yielding a deductive reasoning process.

In practice, some questions cannot be answered without relying on certain predefined constants (such as π and 1) that may not have appeared in the given problem description. We therefore also consider a set of constants $\mathcal{C} = \{c_1, c_2, \cdots, c_{|\mathcal{C}|}\}$. Such constants are also regarded as quantities (i.e., they would be regarded as $\{q_{m+1}, q_{m+2}, \ldots, q_{m+|\mathcal{C}|}\}$) which may play useful roles when forming the final answer expression.

3.1 A Deductive System

As shown in Figure 1, applying the mathematical relation (e.g., "+") between two quantities yields an intermediate expression e. In general, at step t, the resulting expression $e^{(t)}$ (after evaluation) becomes a *newly created quantity* that is added to the list of candidate quantities and is ready for participating in the remaining deductive reasoning process from step $t+1$ onward. This process can be mathematically denoted as follows:

- Initialization:

$$\mathcal{Q}^{(0)} = \mathcal{Q}_{\mathcal{S}} \cup \mathcal{C}$$

- At step t:

$$\begin{aligned} e^{(t)}_{i,j,op} &= q_i \xrightarrow{op} q_j \quad q_i, q_j \in \mathcal{Q}^{(t-1)} \\ \mathcal{Q}^{(t)} &= \mathcal{Q}^{(t-1)} \cup \{e^{(t)}_{i,j,op}\} \\ q_{|\mathcal{Q}^{(t)}|} &:= e^{(t)}_{i,j,op} \end{aligned}$$

where $e^{(t)}_{i,j,op}$ represents the expression after applying the relation op to the ordered pair (q_i, q_j). Following the standard deduction systems (Shieber et al., 1995; Nederhof, 2003), the reasoning process can be formulated in Figure 2. We start with an axiom with the list of quantities in $\mathcal{Q}^{(0)}$. The inference rule is $q_i \xrightarrow{op} q_j$ as described above to obtain the expression as a new quantity at step t.

3.2 Model Components

Reasoner Figure 3 shows the deductive reasoning procedure in our model for an example that involves 3 quantities. We first convert the quantities (e.g., 2,088) into a general quantity token "<*quant*>". We next adopt a pre-trained language model such as BERT (Devlin et al., 2019) or Roberta (Cui et al., 2019; Liu et al., 2019) to obtain the quantity representation $\boldsymbol{q}$ for each quantity q.

[2] While we consider binary operators, extending our approach to support unary or ternary operators is possible (§4.3).

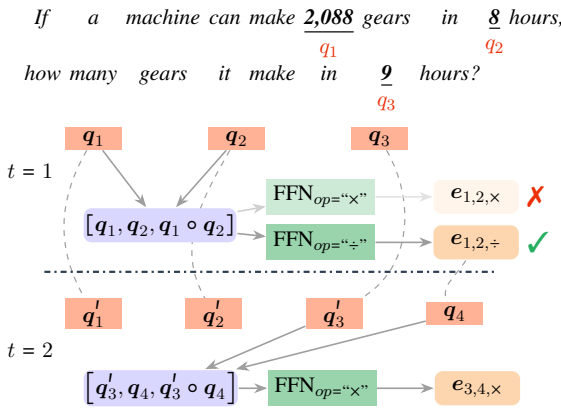

Figure 3: Model architecture for the deductive reasoner. We show the inference procedure to obtain the expression "$q_1 \div q_2 \times q_3$" for the example question.

Given the quantity representations, we consider all the possible quantity pairs, (q_i, q_j). Similar to Lee et al. (2017), we can obtain the representation of each pair by concatenating the two quantity representations and the element-wise product between them. As shown in Figure 3, we apply a non-linear feed-forward network (FFN) on top of the pair representation to get the representation of the newly created expression. The above procedure can be mathematically written as:

$$\boldsymbol{e}_{i,j,op} = \text{FFN}_{op}([\boldsymbol{q}_i, \boldsymbol{q}_j, \boldsymbol{q}_i \circ \boldsymbol{q}_j]), \quad i \leq j \quad (1)$$

where $\boldsymbol{e}_{i,j,op}$ is the representation of the intermediate expression e and op is the operation (e.g., "+", "−") applied to the ordered pair (q_i, q_j). FFN_{op} is an operation-specific network that gives the expression representation under the particular operation op. Note that we have the constraint $i \leq j$. As a result we also consider the "*reverse operation*" for division and subtraction (Roy and Roth, 2015).

As shown in Figure 3, the expression $\boldsymbol{e}_{1,2,\div}$ will be regarded as a new quantity with representation $\boldsymbol{q}_4$ at $t = 1$. In general, we can assign a score to a single reasoning step that yields the expression $e^{(t)}_{i,j,op}$ from q_i and q_j with operation op. Such a score can be calculated by summing over the scores defined over the representations of the two quantities and the score defined over the expression:

$$s(e^{(t)}_{i,j,op}) = s_q(\boldsymbol{q}_i) + s_q(\boldsymbol{q}_j) + s_e(\boldsymbol{e}_{i,j,op}) \quad (2)$$

where we have:

$$\begin{aligned} s_q(\boldsymbol{q}_i) &= \mathbf{w}_q \cdot \text{FFN}(\boldsymbol{q}_i) \\ s_e(\boldsymbol{e}_{i,j,op}) &= \mathbf{w}_e \cdot \boldsymbol{e}_{i,j,op} \end{aligned} \quad (3)$$

Rationalizer	Mechanism
Multi-head Self-Attention	Attention($Q = [\boldsymbol{q}_i, \boldsymbol{e}], K = [\boldsymbol{q}_i, \boldsymbol{e}], V = [\boldsymbol{q}_i, \boldsymbol{e}]$)
GRU cell	GRU_Cell(input = $\boldsymbol{q}_i$, previous hidden = $\boldsymbol{e}$)

Table 1: The mechanism in different rationalizers.

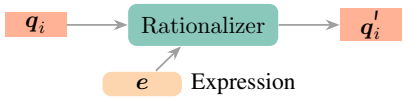

Figure 4: Rationalizing quantity representation.

where $s_q(\cdot)$ and $s_e(\cdot)$ are the scores assigned to the quantity and the expression, respectively, and $\mathbf{w}_q$ and $\mathbf{w}_e$ are the corresponding learnable parameters. Our goal is to find the optimal expression sequence $[e^{(1)}, e^{(2)}, \cdots, e^{(T)}]$ that enables us to compute the final numerical answer, where T is the total number of steps required for this deductive process.

Terminator Our model also has a mechanism that decides whether the deductive procedure is ready to terminate at any given time. We introduce a binary label τ, where 1 means the procedure stops here, and 0 otherwise. The final score of the expression e at time step t can be calculated as:

$$S(e^{(t)}_{i,j,op}, \tau) = s(e^{(t)}_{i,j,op}) + \mathbf{w}_\tau \cdot \text{FFN}(\boldsymbol{e}_{i,j,op}) \quad (4)$$

where $\mathbf{w}_\tau$ is the parameter vector for scoring the τ.

Rationalizer Once we obtain a new intermediate expression at step t, it is crucial to update the representations for the existing quantities. We call this step *rationalization* because it could potentially give us the rationale that explains an outcome (Lei et al., 2016). As shown in Figure 4, the intermediate expression $\boldsymbol{e}$ serves as the rationale that explains how the quantity changes from $\boldsymbol{q}$ to $\boldsymbol{q}'$. Without this step, there is a potential shortcoming for the model. That is, because if the quantity representations do not get updated as we continue the deductive reasoning process, those expressions that were initially highly ranked (say, at the first step) would always be preferred over those lowly ranked ones throughout the process.[3] We rationalize the quantity representation using the current intermediate expression $e^{(t)}$, so that the quantity is aware of the generated expressions when its representation gets updated. This procedure can be mathematically formulated as follows:

$$\boldsymbol{q}'_i = \text{Rationalizer}(\boldsymbol{q}_i, e^{(t)}) \quad \forall\ 1 \leq i \leq |\mathcal{Q}| \quad (5)$$

[3] See the supplementary material for more details on this.

Dataset	#Train	#Valid	#Test	Avg. Sent Len	#Const.	Lang.
MAWPS	1,589	199	199	30.3	17	English
Math23k	21,162	1,000	1,000	26.6	2	Chinese
MathQA†	16,191	2,411	1,605	39.6	24	English
SVAMP	3,138	-	1,000	34.7	17	English

Table 2: Dataset statistics. †: we follow Tan et al. (2021) to do preprocessing and obtain the subset.

Two well-known techniques we can adopt as rationalizers are *multi-head self-attention* (Vaswani et al., 2017) and a *gated recurrent unit* (GRU) (Cho et al., 2014) cell, which allow us to update the quantity representation, given the intermediate expression representation. Table 1 shows the mechanism in two different rationalizers. For the first approach, we essentially construct a sentence with two token representations – quantity $\boldsymbol{q}_i$ and the previous expression $\boldsymbol{e}$ – to perform self-attention. In the second approach, we use $\boldsymbol{q}_i$ as the input state and $\boldsymbol{e}$ as the previous hidden state in a GRU cell.

3.3 Training and Inference

Similar to training sequence-to-sequence models (Luong et al., 2015), we adopt the teacher-forcing strategy (Williams and Zipser, 1989) to guide the model with gold expressions during training. The loss[4] can be written as:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{t=1}^{T} \Big(\max_{(i,j,op)\in\mathcal{H}^{(t)},\tau} \Big[\mathcal{S}_{\boldsymbol{\theta}}(e^{(t)}_{i,j,op}, \tau) \Big] - \mathcal{S}_{\boldsymbol{\theta}}(e^{(t)}_{i^*,j^*,op^*}, \tau^*) \Big) + \lambda ||\boldsymbol{\theta}||^2 \quad (6)$$

where $\boldsymbol{\theta}$ includes all parameters in the deductive reasoner and $\mathcal{H}^{(t)}$ contains all the possible choices of quantity pairs and relations available at time step t. λ is the hyperparameter for the L_2 regularization term. The set $\mathcal{H}^{(t)}$ grows as new expressions are constructed and become new quantities during the deductive reasoning process. The overall loss is computed by summing over the loss at each time step (assuming totally T steps).

During inference, we set a maximum time step T_{max} and find the best expression e^* that has the highest score at each time step. Once we see $\tau = 1$ is chosen, we stop constructing new expressions and terminate the process. The overall expression (formed by the resulting expression sequence) will be used for computing the final numerical answer.

[4] Actually, one might have noticed that this loss comes with a trivial solution at $\boldsymbol{\theta} = \mathbf{0}$. In practice, however, our model and training process would prevent us from reaching such a degenerate solution with proper initialization (Goodfellow et al., 2016). This is similar to the training of a structured perceptron (Collins, 2002), where a similar situation is also involved.

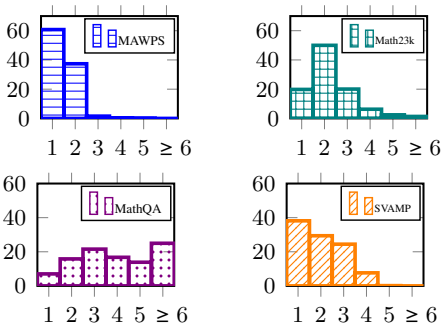

Figure 5: Percentage of questions with different operation count.

Declarative Constraints Our model repeatedly relies on existing quantities to construct new quantities, which results in a structure showing the deductive reasoning process. One advantage of such an approach is that it allows certain declarative knowledge to be conveniently incorporated. For example, as we can see in Equation 6, the default approach considers all the possible combinations among the quantities during the maximization step. We can easily impose constraints to avoid considering certain combinations. In practice, we found in certain datasets such as SVAMP, there does not exist any expression that involve operations applied to the same quantity (such as $9 + 9$ or 9×9, where 9 is from the same quantity in the text). Besides, we also observe that the intermediate results would not be negative. We can simply exclude such cases in the maximization process, effectively reducing the search space during both training and inference. We show that adding such declarative constraints can help improve the performance.

4 Experiments

Datasets We conduct experiments on four datasets across two different languages: MAWPS (Koncel-Kedziorski et al., 2016), Math23k (Wang et al., 2017), MathQA (Amini et al., 2019), and SVAMP (Patel et al., 2021). The dataset statistics can be found in Table 2. For MathQA[5], we follow Tan et al. (2021)[6] to

[5] The original MathQA (Amini et al., 2019) dataset contains a certain number of instances that have annotated equations which cannot lead to the correct numerical answer.

[6] Our dataset size is not exactly the same as Tan et al. (2021) as they included some instances that are wrongly annotated. We only kept the part that has correct annotations. We con-

	Model	Val Acc.
S2S	GroupAttn (Li et al., 2019)	76.1
	Transformer (Vaswani et al., 2017)	85.6
	BERT-BERT (Lan et al., 2021)	86.9
	Roberta-Roberta (Lan et al., 2021)	88.4
S2T/G2T	GTS (Xie and Sun, 2019)	82.6
	Graph2Tree (Zhang et al., 2020)	85.6
	Roberta-GTS (Patel et al., 2021)	88.5
	Roberta-Graph2Tree (Patel et al., 2021)	88.7
OURS	BERT-DEDUCTREASONER	91.2 (± 0.16)
	ROBERTA-DEDUCTREASONER	**92.0** (± 0.20)
	MBERT-DEDUCTREASONER	91.6 (± 0.13)
	XLM-R-DEDUCTREASONER	91.6 (± 0.11)

Table 3: 5-fold cross-validation results on MAWPS.

	Model	Val Acc. Test	Val Acc. 5-fold
S2S	GroupAttn (Li et al., 2019)	69.5	66.9
	mBERT-LSTM (Tan et al., 2021)	75.1	-
	BERT-BERT (Lan et al., 2021)	-	76.6
	Roberta-Roberta (Lan et al., 2021)	-	76.9
S2T/G2T	GTS (Xie and Sun, 2019)	75.6	74.3
	KA-S2T† (Wu et al., 2020)	76.3	-
	MultiE&D (Shen and Jin, 2020)	78.4	76.9
	Graph2Tree (Zhang et al., 2020)	77.4	75.5
	NeuralSymbolic (Qin et al., 2021)	-	75.7
	NUMS2T† (Wu et al., 2021)	78.1	-
	HMS (Lin et al., 2021)	76.1	-
	BERT-Tree (Li et al., 2021)	82.4	-
OURS	BERT-DEDUCTREASONER	84.5 (± 0.16)	82.6 (± 0.17)
	ROBERTA-DEDUCTREASONER	**85.1** (± 0.24)	**83.0** (± 0.23)
	MBERT-DEDUCTREASONER	84.3 (± 0.19)	82.5 (± 0.33)
	XLM-R-DEDUCTREASONER	84.0 (± 0.22)	82.0 (± 0.12)

Table 4: Results on Math23k. †: they used their own splits (so their results may not be directly comparable).

adapt the dataset to filter out some questions that are unsolvable. We consider the operations "*addition*", "*subtraction*", "*multiplication*", and "*division*" for MAWPS and SVAMP, and an extra "*exponentiation*" for MathQA and Math23k.

The number of operations involved in each question can be one of the indicators to help us gauge the difficulty of a dataset. Figure 5 shows the percentage distribution of the number of operations involved in each question. The MathQA dataset generally contains larger portions of questions that involve more operations, while 97% of the questions in MAWPS can be answered with only one or two operations. More than 60% of the instances in MathQA have three or more operations, which likely makes their problems harder to solve. Furthermore, MathQA (Amini et al., 2019) contains GRE questions in many domains including physics, geometry, probability, etc., while Math23k questions are from primary school. Different from other datasets, SVAMP (Patel et al., 2021)[7] is a challenging set that is manually created to evaluate a model's robustness. They applied variations over the instances sampled from MAWPS. Such variations could be: adding extra quantities, swapping the positions between noun phrases, etc.

Baselines The baseline approaches can be broadly categorized into sequence-to-sequence (S2S), sequence-to-tree (S2T) and graph-to-tree (G2T) models. **GroupAttn** (Li et al., 2019) designed several types of attention mechanisms such as question or quantity related attentions in the seq2seq model. Tan et al. (2021) uses multilingual BERT with an LSTM decoder (**mBERT-LSTM**). Lan et al. (2021) presented two seq2seq models that use BERT/Roberta as both encoder and decoder, namely, **BERT-BERT** and **Roberta-Roberta**. Sequence-to-tree models mainly use a tree-based decoder with GRU (**GTS**) (Xie and Sun, 2019) or BERT as the encoder (**BERT-Tree**) (Liang et al., 2021; Li et al., 2021). **NUMS2T** (Wu et al., 2020) and **NeuralSymbolic** (Qin et al., 2021) solver incorporate external knowledge in the S2T architectures. **Graph2Tree** (Zhang et al., 2020) models the quantity relations using GCN.

Training Details We adopt BERT (Devlin et al., 2019) and Roberta (Liu et al., 2019) for the English datasets. Chinese BERT and Chinese Roberta (Cui et al., 2019) are used for Math23k. We use the GRU cell as the rationalizer. We also conduct experiments with multilingual BERT and XLM-Roberta (Conneau et al., 2020). The pre-trained models are initialized from HuggingFace's Transformers (Wolf et al., 2020). We optimize the loss with the Adam optimizer (Kingma and Ba, 2014; Loshchilov and Hutter, 2019). We use a learning rate of $2e$-5 and a batch size of 30. The regularization coefficient λ is set to 0.01. We run our models with 5 random seeds and report the average results (with standard deviation). Following most previous works, we mainly report the value accuracy (percentage) in our experiments. In other words, a prediction is considered correct if the predicted expression leads to the same value as the gold expression. Following previous practice (Zhang et al., 2020; Tan et al., 2021; Patel et al., 2021), we report

firmed such information with the authors of Tan et al. (2021), and make our version of this dataset publicly available.

[7]There is no test split for this dataset. We strictly follow the experiment setting in Patel et al. (2021).

Model	Val Acc.
Graph2Tree (Zhang et al., 2020)	69.5
BERT-Tree (Li et al., 2021)	73.8
mBERT+LSTM (Tan et al., 2021)	77.1
BERT-DEDUCTREASONER	78.5 (± 0.07)
ROBERTA-DEDUCTREASONER	**78.6** (± 0.09)
MBERT-DEDUCTREASONER	78.2 (± 0.21)
XLM-R-DEDUCTREASONER	78.2 (± 0.11)

Table 5: Test accuracy comparison on MathQA.

	Model	Val Acc.
S2S	GroupAttn (Li et al., 2019)	21.5
	BERT-BERT (Lan et al., 2021)	24.8
	Roberta-Roberta (Lan et al., 2021)	30.3
S2T/G2T	GTS* (Xie and Sun, 2019)	30.8
	Graph2Tree (Zhang et al., 2020)	36.5
	BERT-Tree (Li et al., 2021)	32.4
	Roberta-GTS (Patel et al., 2021)	41.0
	Roberta-Graph2Tree (Patel et al., 2021)	43.8
OURS	BERT-DEDUCTREASONER	35.3 (± 0.04)
	+ *constraints*	42.3 (± 0.09)
	ROBERTA-DEDUCTREASONER	45.0 (± 0.10)
	+ *constraints*	**47.3** (± 0.20)
	MBERT-DEDUCTREASONER	36.1 (± 0.07)
	+ *constraints*	41.3 (± 0.08)
	XLM-R-DEDUCTREASONER	38.1 (± 0.08)
	+ *constraints*	44.6 (± 0.15)

Table 6: Test accuracy comparison on SVAMP.

5-fold cross-validation results on both MAWPS[8] and Math23k, and also report the test set performance for Math23k, MathQA and SVAMP.

4.1 Results

MAWPS and Math23k We first discuss the results on MAWPS and Math23k, two datasets that are commonly used in previous research. Table 3 and 4 show the main results of the proposed models with different pre-trained language models. We compare with previous works that have reported results on these datasets. Among all the encoders for our model DEDUCTREASONER, the Roberta encoder achieves the best performance. In addition, DEDUCTREASONER significantly outperforms all the baselines regardless of the choice of encoder. The performance on the best S2S model (Roberta-Roberta) is on par with the best S2T model (Roberta-Graph2Tree) on MAWPS. Overall, the accuracy of Roberta-based DEDUCTREASONER is more than 3 points higher than Roberta-Graph2Tree ($p < 0.001$)[9] on MAWPS, and more than 2 points higher than BERT-Tree ($p < 0.005$) on Math23k. The comparisons show that our deductive reasoner is robust across different languages and datasets of different sizes.

MathQA and SVAMP As mentioned before, MathQA and SVAMP are more challenging – the former consists of more complex questions and the latter consists of specifically designed challenging questions. Table 5 and 6 show the performance comparisons. We are able to outperform the best baseline mBERT-LSTM[10] by 1.5 points in accuracy on MathQA. Different from other three datasets, the performance between different language models shows larger gaps on SVAMP. As we can see from baselines and our models, the choice of encoder appear to be important for solving questions in SVAMP – the results on using Roberta as the encoder are particularly striking. Our best variant ROBERTA-DEDUCTREASONER achieves an accuracy score of 47.3 and is able to outperfrom the best baseline (Roberta-Graph2Tree) by 3.5 points ($p < 0.01$). By incorporating the constraints from our prior knowledge (as discussed in §3.3), we observe significant improvements for all variants – up to 7.0 points for our BERT-DEDUCTREASONER.

Overall, these results show that our model is more robust as compared to previous approaches on such challenging datasets.

Fine-grained Analysis We further perform fine-grained performance analysis based on questions with different numbers of operations. Table 7 shows the accuracy scores for questions that involve different numbers of operations. It also shows the equation accuracy on all datasets[11]. We compared our ROBERTA-DEDUCTREASONER with the best performing baselines in Table 3 (Roberta-Graph2Tree), 4 (BERT-Tree), 5 (mBERT+LSTM) and 6 (Roberta-Graph2Tree). On MAWPS and Math23k, our ROBERTA-DEDUCTREASONER model consistently yields higher results than baselines. On MathQA, our model also performs better on questions that involve 2, 3, and 4 operations. For the other more challenging dataset SVAMP, our model

[8] All previous efforts combine training/dev/test sets and perform 5-fold cross validation, which we follow.

[9] We conduct bootstrapping t-test to compare the results.

[10] We ran the their code on our adapted MathQA dataset.

[11] Equ Acc: we regard an equation as correct if and only if it matches with the reference equation (up to reordering of sub-expressions due to commutative operations, namley "+" and "×").

#Operation	MAWPS		Math23k		MathQA		SVAMP	
	Baseline	OURS	Baseline	OURS	Baseline	OURS	Baseline	OURS
1	88.2	**92.7**	91.3	**93.6**	**77.3**	**77.4**	**51.9**	**52.0**
2	91.3	**91.6**	89.3	**92.0**	81.3	**83.5**	17.8	**32.1**
3	-	-	74.5	**77.0**	81.9	**83.4**	-	-
4	-	-	59.1	**60.3**	79.3	**81.7**	-	-
>=5	-	-	56.5	**69.2**	**71.5**	**71.4**	-	-
Overall Performance								
Equ Acc.	80.8	88.6	71.2	79.0	74.0	74.0	40.9	45.0
Val Acc.	88.7	92.0	82.4	85.1	77.1	78.6	43.8	47.3

Table 7: Acc. under different number of operations.

	MAWPS	Math23k	MathQA	SVAMP
Unused	6.5%	8.2%	20.7%	44.5%
Accuracy (unused = 0)	93.6	87.1	81.4	63.6
Accuracy (unused ≥ 1)	100.0†	62.1	67.4	27.0

Table 8: Value accuracy with respect to the number of unused quantities. The second row shows the percentage of instances that have unused quantities. †: may not be representative as there are only 3 instances.

has comparable performance with the baseline on 1-step questions, but achieves significantly better results (+14.3 points) on questions that involve 2 steps. Such comparisons on MathQA and SVAMP show that our model has a robust reasoning capability on more complex questions.

We observe that all models (including ours and existing models) are achieving much lower accuracy scores on SVAMP, as compared to other datasets. We further investigate the reason for this. Patel et al. (2021) added irrelevant information such as extra quantities in the question to confuse the models. We quantify the effect by counting the percentage of instances which have quantities unused in the equations. As we can see in Table 8, SVAMP has the largest proportion (i.e., 44.5%) of instances whose gold equations do not fully utilize all the quantities in the problem text. The performance also significantly drops on those questions with more than one unused quantity on all datasets. The analysis suggests that our model still suffer from extra irrelevant information in the question and the performance is severely affected when such irrelevant information appears more frequently.

Effect of Rationalizer Table 9 shows the performance comparison with different rationalizers. As described in §3.2, the rationalizer is used to update the quantity representations at each step, so as to better "prepare them" for the subsequent reasoning process given the new context. We believe this step is crucial for achieving good performance, especially for complex MWP solving. As shown in Table 9, the performance drops by 7.3 points in value

Rationalizer	MAWPS		Math23k	
	Equ Acc.	Val Acc.	Equ Acc.	Val Acc.
NONE	88.4	91.8	71.5	77.8
Self-Attention	88.3	91.7	77.5	84.8
GRU unit	88.6	92.0	79.0	85.1

Table 9: Performance comparison on different rationalizer using the Roberta-base model.

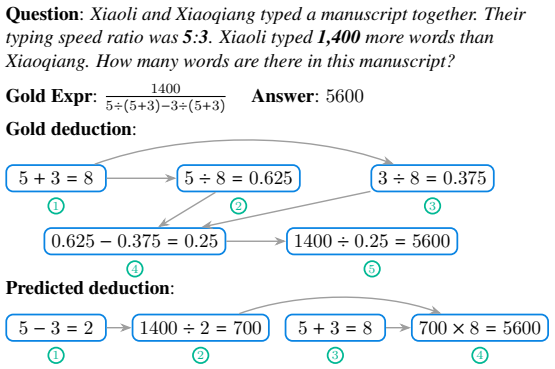

Figure 6: Deductive steps by our reasoner.

accuracy for Math23k without rationalization, confirming the importance of rationalization in solving more complex problems that involve more steps. As most of the questions in MAWPS involve only 1-step questions, the significance of using rationalizer is not fully revealed on this dataset.

It can be seen that using self-attention achieves worse performance than the GRU unit. We believe the lower performance by using multi-head attention as rationalizer may be attributed to two reasons. First, GRU comes with sophisticated internal gating mechanisms, which may allow richer representations for the quantities. Second, attention, often interpreted as a mechanism for measuring similarities (Katharopoulos et al., 2020), may be inherently biased when being used for updating quantity representations. This is because when measuring the similarity between quantities and a specific expression (Figure 4), those quantities that have just participated in the construction of the expression may receive a higher degree of similarity.

4.2 Case Studies

Explainability of Output Figure 6 presents an example prediction from Math23k. In this question, the gold deductive process first obtains the speed difference by "$5 \div (5+3) - 3 \div (5+3)$" and the final answer is 1400 divided by this difference. On the other hand, the predicted deductive process offers a slightly different understanding in speed difference. Assuming speed can be measured by some abstract "units", the predicted deductive

Question: *There are **255** apple trees in the orchard. Planting another **35** pear trees makes the number exactly the same as the apple trees. If every **20** pear trees are planted in a row, how many rows can be planted in total?*
Gold Expr: $(255 - 35) \div 20$ **Answer**: 11
Predicted Expr: $(255 + 35) \div 20$ **Predicted**: 14.5
Deductive Scores:

$255 + 35 = 260$ Prob.: 0.068 > $255 - 35 = 220$ Prob.: 0.062

Perturbed Question: *There are **255** apple trees in the orchard. **The number of pear trees are 35 fewer than the apple trees.** If every **20** pear trees are planted in a row, how many rows can be planted in total?*
$255 + 35 = 260$ **Prob.:** 0.061 < $255 - 35 = 220$ **Prob.:** 0.067

Figure 7: Question perturbation in deductive reasoning.

process first performs subtraction between 5 and 3, which gives us "2 units" of speed difference. Next, we can obtain the number of words associated with each speed unit ($1400 \div 2$). Finally, we can arrive at the total number of words by multiplying the number of words per unit (700) and the total number of units (8).[12] Through such an example we can see that our deductive reasoner is able to produce explainable steps to understand the answers.

Question Perturbation The model predictions also give us guidance to understand the errors. Figure 7 shows how we can perturb a question given the error prediction (taken from Math23k). As we can see, the first step is incorrectly predicted with the "+" relation between 255 and 35. Because the first step involves the two quantities in the first two sentences, where we can locate the possible cause for the error. The gold step has a probability of 0.062 which is somewhat lower than the incorrect prediction. We believe that the second sentence (marked in red) may convey semantics that can be challenging for the model to digest, resulting in the incorrect prediction. Thus, we perturb the second sentence to make it semantically more straightforward (marked below in blue). The probability for the sub-expression $225 - 35$ becomes higher after the purtubation, leading to a correct prediction (the "$-$" relation). Such an analysis demonstrates the strong interpretability of our deductive reasoner, and highlights the important connection between math word problem solving and reading comprehension, a topic that has been studied in educational psychology (Vilenius-Tuohimaa et al., 2008).

4.3 Practical Issues

We discuss some practical issues with the current model in this section. Similar to most previous research efforts (Li et al., 2019; Xie and Sun, 2019), our work needs to maintain a list of constants (e.g., 1 and π) as additional candidate quantities. However, a large number of quantities could lead to a large search space of expressions (i.e., $\mathcal{H}$). In practice, we could select some top-scoring quantities and build expressions on top of them (Lee et al., 2018). Another assumption of our model, as shown in Figure 3, is that only binary operators are considered. Actually, extending it to support unary or ternary operators can be straightforward. Handling unary operators would require the introduction of some unary rules, and a ternary operator can be defined as a composition of two binary operators.

Our current model performs the greedy search in the training and inference process, which could be improved with a beam search process. One challenge with designing the beam search algorithm is that the search space $\mathcal{H}^{(t)}$ is expanding at each step t (Equation 6). We empirically found the model tends to favor outputs that involve fewer reasoning steps. In fact, better understanding the behavior and effect of beam search in seq2seq models remains an active research topic (Cohen and Beck, 2019; Koehn and Knowles, 2017; Hokamp and Liu, 2017), and we believe how to perform effective beam search in our setup could be an interesting research question that is worth exploring further.

5 Conclusion and Future Work

We provide a new perspective to the task of MWP solving and argue that it can be fundamentally regarded as a complex relation extraction problem. Based on this observation, and motivated by the deductive reasoning process, we propose an end-to-end deductive reasoner to obtain the answer expression in a step-by-step manner. At each step, our model performs iterative mathematical relation extraction between quantities. Thorough experiments on four standard datasets demonstrate that our deductive reasoner is robust and able to yield new state-of-the-art performance. The model achieves particularly better performance for complex questions that involve a larger number of operations. It offers us the flexibility in interpreting the results, thanks to the deductive nature of our model.

Future directions that we would like to explore include how to effectively incorporate commonsense knowledge into the deductive reasoning process, and how to facilitate counterfactual reasoning (Richards and Sanderson, 1999).

[12] Interestingly, when we presented this question to 3 human solvers, 2 of them used the first approach and 1 of them arrived at the second approach.

Acknowledgements

We would like to thank the anonymous reviewers and our ARR action editor for their constructive comments, and Hang Li for helpful discussions and comments on this work. This work was done when Jierui Li was working as a research assistant at SUTD, and when Wei Lu was serving as a consultant at ByteDance AI Lab.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of NAACL*.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of CVPR*.

Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. 2021. Deep learning for ai. *Communications of the ACM*, 64(7):58–65.

Tarek R Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado Vieira Lima, et al. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*.

Daniel G Bobrow. 1964. Natural language input for a computer problem solving system.

Yixuan Cao, Feng Hong, Hongwei Li, and Ping Luo. 2021. A bottom-up dag structure extraction model for math word problems. In *Proceedings of AAAI*.

Ting-Rui Chiang and Yun-Nung Chen. 2019. Semantically-aligned equation generation for solving and reasoning math word problems. In *Proceedings of NAACL*.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*.

Eldan Cohen and Christopher Beck. 2019. Empirical analysis of beam search performance degradation in neural sequence models. In *Proceedings of ICML*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal Vishrav Chaudhary Guillaume Wenzek, Francisco Guzmán, Edouard Grave Myle Ott Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of ACL*.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. 2019. Pre-training with whole word masking for chinese bert. *arXiv preprint arXiv:1906.08101*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL*.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2020. Neural module networks for reasoning over text. In *Proceedings of ICLR*.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of ACL*.

Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. 2018. Neural math word problem solver with reinforcement learning. In *Proceedings of COLING*.

Daniel Kahneman. 2011. *Thinking, fast and slow*. Macmillan.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of ICML*.

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

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

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation, ACL*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of NAACL*.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of ACL*.

Yihuai Lan, Lei Wang, Qiyuan Zhang, Yunshi Lan, Bing Tian Dai, Yan Wang, Dongxiang Zhang, and Ee-Peng Lim. 2021. Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. *arXiv preprint arXiv:2109.00799*.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Proceedings of EMNLP*.

Kenton Lee, Luheng He, and Luke Zettlemoyer. 2018. Higher-order coreference resolution with coarse-to-fine inference. In *Proceedings of NAACL*.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. Rationalizing neural predictions. In *Proceedings of EMNLP*.

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

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. 2020. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. In *Proceedings of Findings of EMNLP*.

Zhongli Li, Wenxuan Zhang, Chao Yan, Qingyu Zhou, Chao Li, Hongzhi Liu, and Yunbo Cao. 2021. Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems. *arXiv preprint arXiv:2110.08464*.

Chao-Chun Liang, Yu-Shiang Wong, Yi-Chung Lin, and Keh-Yih Su. 2018. A meaning-based statistical english math word problem solver. In *Proceedings of NAACL*.

Zhenwen Liang, Jipeng Zhang, Jie Shao, and Xiangliang Zhang. 2021. Mwp-bert: A strong baseline for math word problems. *arXiv preprint arXiv:2107.13435*.

Christian Liguda and Thies Pfeiffer. 2012. Modeling math word problems with augmented semantic networks. In *International Conference on Application of Natural Language to Information Systems*.

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 *Proceedings of AAAI*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of ACL*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of ICLR*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*.

Anirban Mukherjee and Utpal Garain. 2008. A review of methods for automatic understanding of natural language mathematical problems. *Artificial Intelligence Review*, 29(2):93–122.

Mark-Jan Nederhof. 2003. Weighted deductive parsing and knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of NAACL*.

Jean Piaget. 1952. *Child's Conception of Number*. Routledge.

Jinghui Qin, Xiaodan Liang, Yining Hong, Jianheng Tang, and Liang Lin. 2021. Neural-symbolic solver for math word problems with auxiliary tasks. In *Proceedings of ACL-IJCNLP*.

Cassandra A Richards and Jennifer A Sanderson. 1999. The role of imagination in facilitating deductive reasoning in 2-, 3-and 4-year-olds. *Cognition*, 72(2):B1–B9.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of EMNLP*.

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

Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. 2021. Generate & rank: A multi-task framework for math word problems. In *Proceedings of Findings of EMNLP*.

Yibin Shen and Cheqing Jin. 2020. Solving math word problems with multi-encoders and multi-decoders. In *Proceedings of COLING*.

Stuart M Shieber, Yves Schabes, and Fernando CN Pereira. 1995. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1-2):3–36.

Minghuan Tan, Lei Wang, Lingxiao Jiang, and Jing Jiang. 2021. Investigating math word problems using pretrained multilingual language models. *arXiv preprint arXiv:2105.08928*.

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 *Proceedings of NeurIPS*.

Piia Maria Vilenius-Tuohimaa, Kaisa Aunola, and Jari-Erik Nurmi. 2008. The association between mathematical word problems and reading comprehension. *Educational Psychology*, 28(4):409–426.

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 *Proceedings of AAAI*.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of EMNLP*.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP: System Demonstrations*.

Qinzhuo Wu, Qi Zhang, Jinlan Fu, and Xuan-Jing Huang. 2020. A knowledge-aware sequence-to-tree network for math word problem solving. In *Proceedings of EMNLP*.

Qinzhuo Wu, Qi Zhang, Zhongyu Wei, and Xuan-Jing Huang. 2021. Math word problem solving with explicit numerical values. In *Proceedings of ACL-IJCNLP*.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *Proceedings of IJCAI*.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. 2003. Kernel methods for relation extraction. *Journal of machine learning research*, 3(Feb).

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

Zexuan Zhong and Danqi Chen. 2021. A frustratingly easy approach for entity and relation extraction. In *Proceedings of NAACL*.

A Importance of Rationalizer

We further elaborate on the importance of the rationalizer module in this section. As mentioned in §3.2, it is crucial for us to properly update the quantity representations, especially for questions that require more than 3 operations to solve (i.e., $t \geq 3$). Because if the quantity representations do not get updated as we continue the deductive reasoning process, those expressions that were initially highly ranked (say, at the first step) would always be preferred over those lowly ranked ones throughout the process.

We provide an example here to illustrate the scenario. Suppose our target expression is $(1 + 2) * (3 + 4)$, the first step is to predict:

$$e^{(1)} = 1 + 2 \tag{7}$$

In order to obtain the correct intermediate expression $1 + 2$ as $e^{(1)}$, the model has to give the highest score to this expression. Note that, the score of the expression $e_{1,2,+}$ also has to be larger than the score of $e_{3,4,+}$.

$$s(e^{(1)}_{1,2,+}) > s(e^{(1)}_{3,4,+}) \tag{8}$$

However, in order to reach the final target expression, in the next step, the model needs to construct the intermediate expression $3 + 4$. Without the rationalizer, the representations for the quantities are unchanged, so we would have:

$$s(e^{(2)}_{1,2,+}) = s(e^{(1)}_{1,2,+}) > s(e^{(1)}_{3,4,+}) = s(e^{(2)}_{3,4,+}) \tag{9}$$

From here we could see that the model would not be able to produce the intermediate expression $3 + 4$ in the second step (but would still prefer to generate another $1 + 2$). With the rationalizer in place, the above two equations in Equation 9 in general may not hold, which effectively prevents such an issue from happening.

B Additional Implementation Details

We implement our model with PyTorch and run all experiments using Tesla V100 GPU. The feed-forward network in our model is simply linear transformation followed by the ReLU activation. We also apply layer normalization and dropout in the feed-forward network. The hidden size in the feed-forward network is 768, which the is same as the hidden size used in BERT/Roberta.