Solving Arithmetic Word Problems by Scoring Equations with Recursive Neural Networks

Klim Zaporojets[a,*], Giannis Bekoulis[b,**], Johannes Deleu[a], Thomas Demeester[a], Chris Develder[a]

[a] *Ghent University – imec, IDLab, Dept. of Information Technology (INTEC), Technologiepark Zwijnaarde 15, 9052 Ghent, Belgium*
[b] *Vrije Universiteit Brussel – imec, Dept. of Electronics and Informatics (ETRO), Pleinlaan 9, 1050 Brussels, Belgium*

Abstract

Solving arithmetic word problems is a cornerstone task in assessing language understanding and reasoning capabilities in NLP systems. Recent works use automatic extraction and ranking of candidate solution equations providing the answer to arithmetic word problems. In this work, we explore novel approaches to score such candidate solution equations using tree-structured recursive neural network (Tree-RNN) configurations. The advantage of this Tree-RNN approach over using more established sequential representations, is that it can naturally capture the structure of the equations. Our proposed method consists of transforming the mathematical expression of the equation into an expression tree. Further, we encode this tree into a Tree-RNN by using different Tree-LSTM architectures. Experimental results show that our proposed method (i) improves overall performance with more than 3% accuracy points compared to previous state-of-the-art, and with over 15% points on a subset of problems that require more complex reasoning, and (ii) outperforms sequential LSTMs by 4% accuracy points on such more complex problems.

Keywords: arithmetic word problems, recursive neural networks, information extraction, natural language processing

1. Introduction

Natural language understanding often requires the ability to comprehend and reason with expressions involving numbers. This has produced a recent rise in interest to

*Corresponding author

**The work presented in the paper was performed while dr. Bekoulis was with Ghent University – imec, IDLab, Department of Information Technology.

Email addresses: klim.zaporojets@ugent.be (Klim Zaporojets), gbekouli@etrovub.be (Giannis Bekoulis), johannes.deleu@ugent.be (Johannes Deleu), thomas.demeester@ugent.be (Thomas Demeester), chris.develder@ugent.be (Chris Develder)

build applications to automatically solve math word problems (Kushman et al., 2014; Koncel-Kedziorski et al., 2015; Mitra & Baral, 2016; Wang et al., 2018b; Zhang et al., 2019). These math problems consist of a textual description comprising numbers with a question that will guide the reasoning process to get the numerical solution (see Fig. 1 for an example). This is a complex task because of (i) the large output space of the possible equations representing a given math problem, and (ii) reasoning required to understand the problem.

The research community has focused in solving mainly two types of mathematical word problems: *arithmetic word problems* (Hosseini et al., 2014; Mitra & Baral, 2016; Wang et al., 2017; Li et al., 2019; Chiang & Chen, 2019) and *algebraic word problems* (Kushman et al., 2014; Shi et al., 2015; Ling et al., 2017; Amini et al., 2019). Arithmetic word problems can be solved using basic mathematical operations ($+, -, \times, \div$) and involve a single unknown variable. Algebraic word problems, on the other hand, involve more complex operators such as square root, exponential and logarithm with multiple unknown variables. In this work, we focus on solving *arithmetic word problems* such as the one illustrated in Fig. 1. This figure illustrates (a) *arithmetic word problem* statement, (b) the arithmetical formula of the *solution* to the problem, and (c) the *expression tree* representation of the solution formula where the leaves are connected to quantities and internal nodes represent operations.

The main idea of this paper is to explore the use of tree-based Recursive Neural Networks (Tree-RNNs) to encode and score the expression tree (illustrated in Fig. 1(c) that represents a candidate arithmetic expression of a specific arithmetic word problem). This contrasts with predominantly sequential neural representations (Wang et al., 2017, 2018a; Chiang & Chen, 2019) that encode the problem statement from left to right or vice versa. By using Tree-RNN architectures, we can naturally embed the equation inside a tree structure such that the link structure directly reflects the various mathematical operations between operands selected from the sequential textual input. We hypothesize that this structured approach can efficiently capture the semantic representations of the candidate equations to solve more complex arithmetic problems involving multiple and/or non-commutative operators. To test our results, we use the recently introduced SingleEQ dataset (Koncel-Kedziorski et al., 2015). It contains a collection of 508 arithmetic word problems with varying degrees of complexity. This allows us to track the performance of the evaluated systems on subsets that require

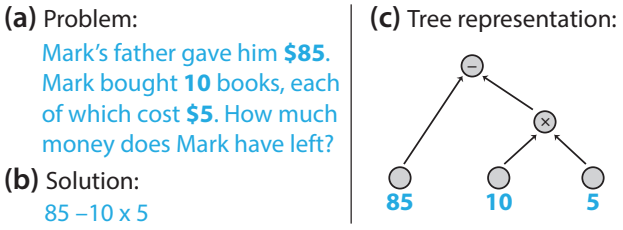

Figure 1: An example of arithmetic word problem from the SingleEQ dataset. It illustrates the (a) *an arithmetic word problem* statement, (b) the respective *solution* formula, and (c) the *expression tree* representing the solution.

different reasoning capabilities. More concretely, we subdivide the initial dataset into different subsets of varying reasoning complexity (i.e., based on the number of operators, commutative (symmetric) or non-commutative (asymmetric) operations), to investigate whether the performance of the proposed architecture remains consistent across problems of increasing complexity.

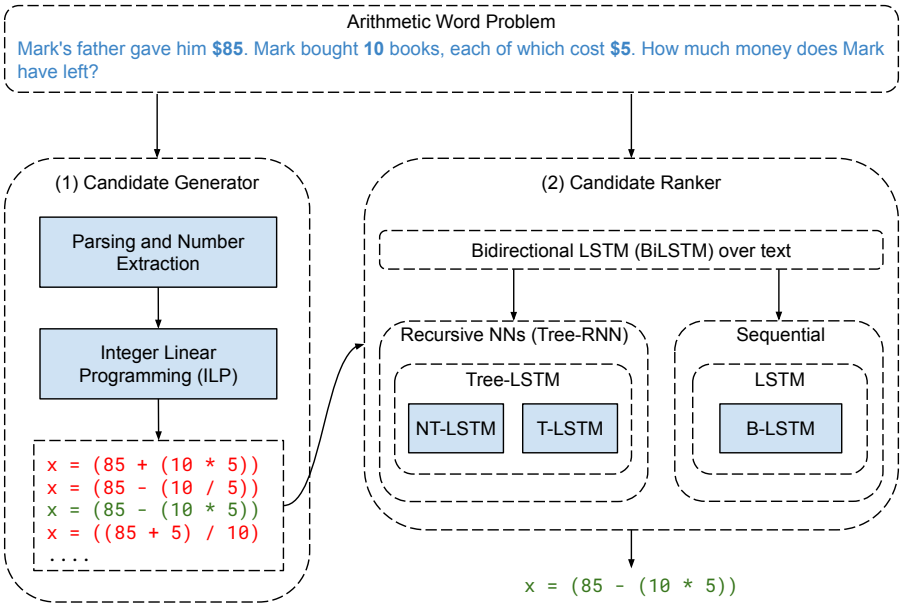

Figure 2: High-level conceptual view of the arithmetic word problem architecture used throughout the paper. It consists of two main components: (1) *candidate generator* responsible for generating candidate equations to solve a particular *arithmetic word problem*, and (2) *candidate ranker*, for selecting the best candidate from the list provided by *candidate generator*, using the models NT-LSTM, T-LSTM, or B-LSTM.

Figure 2 provides a high-level conceptual view of the interconnection between the main components of our proposed system. The processing flow consists of two main steps. In the first step, we use the *candidate generator* to generate a list of potential candidate equations for solving a particular *arithmetic word problem*. To achieve this, we employ the Integer Linear Programming (ILP) constraint optimization component proposed by Koncel-Kedziorski et al. (2015) (see Section 3.1). In the second step, the candidate equations are ranked by the *candidate ranker*, and the equation with the highest score is chosen as the solution to the processed *arithmetic word problem* (see Section 3.2). In this paper, we focus on this second step by exploring the impact of structural Tree-RNN-based and sequential Long Short Term Memory-based (LSTM; Hochreiter & Schmidhuber (1997)) candidate equation encoding methods. More specifically, we define two Tree-RNN models inspired by the work of Tai et al. (2015) on Tree-LSTM models: (i) T-LSTM (Child-Sum Tree-LSTM), and (ii) NT-LSTM (N-ary Tree-LSTM). In the rest of the manuscript we refer to the general tree-structured architecture of these models as Tree-LSTM. The main difference

between the two is that, while in T-LSTM the child node representations are summed up, in NT-LSTM they are concatenated. Unlike the representation used in Tai et al. (2015), where the input is given by the word embeddings, our Tree-LSTM models also take as input the operation embeddings (in inner nodes) that represent each of the arithmetic operators ($-$, $+$, $\div$, $\times$). This allows our architecture to distinguish between different operators that are contained in a particular expression tree. We show that NT-LSTM is more suitable to deal with equations that involve non-commutative operators because this architecture is able to capture the order of the operands. We also compare our Tree-LSTM models with a sequential LSTM model which we call B-LSTM. All the models (T-LSTM, NT-LSTM, and B-LSTM) take as input the contextualized representation of the numbers in text produced by a bidirectional LSTM layer (BiLSTM) (see Section 3.2 for details). After conducting a thorough multi-fold experimentation phase involving multiple random weight re-initializations in order to ensure the validity of our results, we will show that the main added value of our Tree-LSTM-based models compared to state-of-the-art methods lays in an increased performance for more complex arithmetic word problems.

More concretely, our contribution is three-fold: (i) we propose using Tree-LSTMs for solving arithmetic word problems, to embed structural information of the equation, (ii) we compare it against a strong neural baseline model (B-LSTM) that relies on sequential LSTMs, and (iii) we perform an extensive experimental study on the SingleEQ dataset, showing that our Tree-LSTM model achieves an overall accuracy improvement of 3%, including an increase $>$15% for more complex problems (i.e., requiring multiple and non-commutative operations), compared to previous state-of-the-art results.

2. Related work

Over the last few years, there has been an increasing interest in building systems to solve *arithmetic word problems*. The adopted approaches can be grouped in three main categories: (i) Rule-based systems, (ii) Statistical systems, and (iii) Neural network systems.

Rule-based systems: The first attempts to solve arithmetic problems date back to the 1960s with the work by Bobrow (1964), who proposed and implemented STUDENT, a rule-based parsing system to extract numbers and operations between them by using pattern matching techniques. Charniak (1968, 1969) extended STUDENT by including basic coreference resolution and capability to work with rate expressions (e.g., "kms per hour"). On the other hand, Fletcher (1985) designed and implemented a system that given a propositional representation of a math problem[1], applies a set of rules to calculate the final solution. The disadvantage of this system is that it needs a parsed propositional representation of a problem as input and cannot operate directly on raw text. This issue was tackled by Bakman (2007), who developed a schema-based system

[1]With propositions such as *GIVE Y X P9*, where entity Y gives to entity X the object defined in P9. This proposition in particular can be linked to the first sentence of example in Fig. 1: "Mark's father gave him $85", where Y represents "Mark's father", X represents "him" which is coreferenced to "Mark", and P9 represents "$85" that are being given.

that consisted of six main reasoning schemas, each one with slots to fill in. After instantiating the schemas for a particular math problem using lexical verb-based rules, the system could derive the corresponding mathematical equation to solve the problem.

The main disadvantages of such rule-based approaches are that they (i) rely on hard-coded lexico-grammar rules, and (ii) lack an integrated view of the problem to be solved, extracting operations one by one. We address these issues by proposing a model that integrates the mathematical representation of a problem in a single structured expression tree. This way, we are able to capture the operator-operator and number-operator relations involved in a particular mathematical expression in a unified manner. Furthermore, we avoid the use of lexico-grammar hard-coded rules (e.g., the use of pattern-based matching) when connecting numbers with the operators, replacing them by composition-semantic representations that link the arithmetic operations with parameters (numbers or other operations) in a recursive tree. Consequently, our solution is more generalizable by not depending on explicit hand-crafted logic.

Statistical systems: Recently, there has been a shift towards statistical feature-driven systems that automatically produce models by capturing patterns present in arithmetic word problem datasets. For example, Hosseini et al. (2014) presented an inductive model that links specific lexicon-based features (e.g., verb categories) to equation operators. The mathematical solution to the problem is built sequentially using state transitions related to operators that are triggered by different verb categories found in the problem statement. On the other hand, Mitra & Baral (2016) connected carefully designed features to equation templates in order to solve specific problem types. While these techniques produced competitive results, they were limited to addition ($+$) and subtraction ($-$) operations on a very narrow problem set domain. In order to solve more diverse types of problems that also involve multiplication and division operators, the community shifted towards more integrated approaches involving tree structure representations. Koncel-Kedziorski et al. (2015) proposed to rank candidate expression trees by training jointly a *local* model to link spans of text with operator tree nodes, and a *global* model that is used to score the consistency of an entire tree. The list of candidates to these two models is generated by an ILP constraint optimization component that, given a set of extracted numbers from a arithmetic word problem text as input, produces a set of candidate solution equations. Conversely, Roy & Roth (2015, 2017) introduced the concept of *monotonic expression tree* to generate candidates. It defines a set of conditions (e.g., two division and subtraction nodes cannot be connected to each other) that considerably restricts the expression tree search space. The authors propose to score the resulting monotonic expression trees jointly by summing up the scores of different classifiers related to a specific expression tree (e.g., the mathematical operator between two numbers in the tree, whether a particular number is related to a rate such as "kms per hour", etc). Recently, the same authors (Roy & Roth, 2018) included additional latent declarative rules (e.g., $[\text{Verb1} \in \text{HAVE}] \wedge [\text{Verb2} \in \text{GIVE}] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \implies \text{Subtraction}$) to link textual expression patterns (derived from preliminary dependency parsing) to specific operations. While these statistical approaches rely on tree structures to evaluate the mathematical expressions, on one hand, they require high manual effort to engineer the features and, on the other hand, it is hard to scale the features to capture operations between more than two numbers. This makes it challenging to apply such models to

more complex equations that involve multiple operators. We tackle this problem by defining a single Tree-RNN structure that evaluates an entire mathematical expression at once. This is done by recursively combining the information from the child nodes in the expression tree and then using a backpropagation mechanism to correspondingly adjust the weights of our model. Furthermore, our equation ranking architecture does not depend on hand-crafted features and parsing-dependent rules, making it more effective in generalizing across different domains.

Neural network systems: Recently, as in all sub-domains of natural language processing, neural network architectures have been applied to tackle math word problems. The first contribution was made by Wang et al. (2017), who introduced a model trained to map problem statements to equation templates. Their model was expanded upon by Huang et al. (2018), who introduced an attention-based copy mechanism for tokens representing numbers. They used a reinforcement learning setting, where positive rewards were assigned when the predicted mathematical expression resulted in a correct answer. Recently, Chiang & Chen (2019) used stack structures inside a sequential encoder-decoder setting where the encoder captures the semantics of a math word problem in a vector that is used by decoder to generate the equation to solve the problem. Moreover, Wang et al. (2018b) proposed the use of Q-Networks in order to generate expression trees, by giving positive reward whenever the operator between two numbers is correct. The aforementioned studies, while showing promising results, were not designed to naturally capture the structural form of mathematical expressions when multiple operators are involved (e.g., $1 + (2/3)$ vs. $(1 + 2)/3$). We propose encoding equations with Tree-LSTMs (Tai et al., 2015), which are recursive neural sequence models, thus allowing to naturally reflect the execution order of operations in an expression tree by recursively combining the children nodes' semantic representations.

Table 1 compares our approach (the use of Tree-LSTM-based T-LSTM and NT-LSTM models) with the rest of the methods described in this Section. The main difference of our architecture is that we explore the impact of using tree-based neural encoding (i.e., by means of Tree-LSTM models). We hypothesize that this approach allows to better capture the arithmetic equation structure than the currently predominant neural sequential models (Wang et al., 2017, 2018a; Chiang & Chen, 2019). Furthermore, the independence from feature-based and rule-based methods makes our solution more generalizable. This is because our model does not depend on hand-crafted rules or features to capture the patterns of a particular dataset. This aspect will be explored further when comparing the performance of our model to the current feature-based state-of-the-art system (Koncel-Kedziorski et al., 2015) in Section 5.

Tree-RNN models (Socher et al., 2011) have been shown to perform better for modeling data on tasks that have an inherently hierarchical structure. For example, Socher et al. (2011) proposed to use recursive models in order to model the compositional structure of scene images (e.g., a scene image of a house can be split in composing regions such as doors, windows, walls, etc.). The authors show that a Tree-RNN-based architecture outperforms previous methods in prediction of hierarchical structure of scene images and in scene image classification. Later, Socher et al. (2013) also showed how recursive structures can be used to encode the inherently hierarchical phrase structural grammar (e.g., the sentence "riding a bike" can be decomposed in the verb "riding" and the noun phrase "a bike", which itself can be decomposed into

Method	Rules	Features	N-Nets	Tree-Based Representation	Tree-Based Encoding
Bobrow (1964)	✓	–	–	–	–
Charniak (1968, 1969)	✓	–	–	–	–
Fletcher (1985)	✓	–	–	–	–
Bakman (2007)	✓	–	–	–	–
Hosseini et al. (2014)	–	✓	–	–	–
Koncel-Kedziorski et al. (2015)	✓	✓	–	✓	–
Mitra & Baral (2016)	–	✓	–	–	–
Roy & Roth (2015, 2017)	–	✓	–	✓	–
Wang et al. (2017)	–	–	✓	–	–
Roy & Roth (2018)	✓	✓	–	✓	–
Huang et al. (2018)	–	✓	✓	–	–
Wang et al. (2018b)	–	✓	✓	✓	–
Chiang & Chen (2019)	–	–	✓	–	–
Li et al. (2019)	–	–	✓	–	–
Our Approach (T-LSTM & NT-LSTM)	–	–	✓	✓	✓

Table 1: Comparison of the various architectures explored in related work. We focus on the following five characteristics: (i) *Rules* indicates whether a rule-based approach is used or not, (ii) *Features* specifies whether the architecture relies on manually engineered features, (iii) *N-Nets* indicates whether artificial neural networks are used or not, (iv) *Tree-Based Representation* groups the models that incorporate information coming from tree structures (e.g., by using trees for feature engineering), and (v) *Tree-Based Encoding* indicates whether the tree structures are used as encoders in a neural network model. The ✓ indicates the presence of a particular characteristic.

determiner "a" and the noun "bike"). This way, the authors achieved state-of-the-art performance in grammatical parsing of the sentences. More recently, Tai et al. (2015) and Chen et al. (2017) showed how encoding the syntactic parsing trees of the sentence with Tree-LSTM models can improve the performance in tasks such as sentiment classification and semantic relatedness (e.g., natural language inference). Similarly, we propose to take advantage of the inherently hierarchical representation of mathematical expression trees by encoding them using Tree-LSTM architectures. Our experiments demonstrate that this representation can be helpful in capturing the semantic relations between operators needed in order to solve more complex arithmetic problems consisting of multiple and/or non-commutative operations.

3. Proposed Architecture

Shortly stated, our task at hand is to identify the correct arithmetic equation, corresponding to an arithmetic problem expressed in natural language text. We follow a two-step approach similar to the work of Koncel-Kedziorski et al. (2015), which formalizes solving multi-sentence arithmetic word problems as (i) the generation and (ii) ranking of expression trees. The first step consists of generating candidate equations using the ILP optimization solver proposed in Koncel-Kedziorski et al. (2015) (*candidate generator* component in Fig. 2). The second step ranks these candidates and selects the top ranked one as the final answer to the arithmetic word problem (*candidate ranker* component in Fig. 2). We use the rest of this section to provide more insights into the *candidate generator* component in Section 3.1, and to describe in detail our proposed *candidate ranker* model in Section 3.2.

3.1. Candidate Generator

This component is responsible for generating possible candidate equations to solve a given arithmetic word problem. A straightforward solution would be to perform an exhaustive search on all the possible arithmetic expression trees given n extracted numbers from a particular problem. However, the resulting search space would grow exponentially with n, which makes this approach not scalable. In order to deal with this exponential growth in the number of candidates, we re-use the Integer Linear Programming (ILP) solver proposed by Koncel-Kedziorski et al. (2015). This solver takes as input the extracted numeric quantities with extra attributes derived from syntactic parsing[2], and generates the most promising candidate equations using two types of constraints:

1. *Hard Constraints:* such as the maximum equation length and syntactic validity of equations (e.g., only one unknown allowed, no division by 0, etc.). As a post-processing step, the ILP solver also removes the arithmetic expressions that produce negative or fractional results.

2. *Soft Constraints:* these constraints assign additional weight to candidate equations whose related entity types (extracted from dependency parse tree) are consistent. For example, in the problem of Fig. 1, the sum (85 + 5) will be prioritized over the sum (5 + 10), because both 85 and 5 refer to the same entity type ("$"), while 10 refers to entity type "books".

To provide a fair comparison between the *candidate ranker* model of ALGES proposed by Koncel-Kedziorski et al. (2015) and our approach (see Section 3.2), we use both the same constraint configuration, and also consider only the top 100 equations produced by the candidate generator. As in ALGES, we report the coverage as *ILP Coverage* in our results section (see Section 5). Additionally, we include in our result tables the performance of the *ILP Naive* approach, which consists of selecting the highest scored candidate by the ILP solver. This score allows us to estimate the impact of the *candidate ranker* component.

3.2. Candidate Ranker

Our proposed candidate ranker model architecture is sketched in Fig. 3 and comprises: (i) a word embedding layer, (ii) a bidirectional LSTM layer (BiLSTM) over the text, and (iii) an additional layer that encodes the equation, using either BiLSTM (B-LSTM model) or Tree-LSTM (T-LSTM and NT-LSTM models) based approaches, detailed below.

The input to our model is a **sequence of tokens** of length N, $W = \{w_1, ..., w_N\}$ of the arithmetic word problem, which we pass through an **embedding layer** to obtain embedded representations $X = \{x_1, ..., x_N\}$ where $x_t \in \mathbb{R}^{d_1}$. We adopt a BiLSTM to obtain **contextual representations** of the tokens. The following is the formal representation of the first LSTM (Hochreiter & Schmidhuber, 1997) layer used to produce the representation referred to as "*BiLSTM over text*" in Fig. 3:

[2]Stanford Dependency Parser in CoreNLP 3.4 is used.

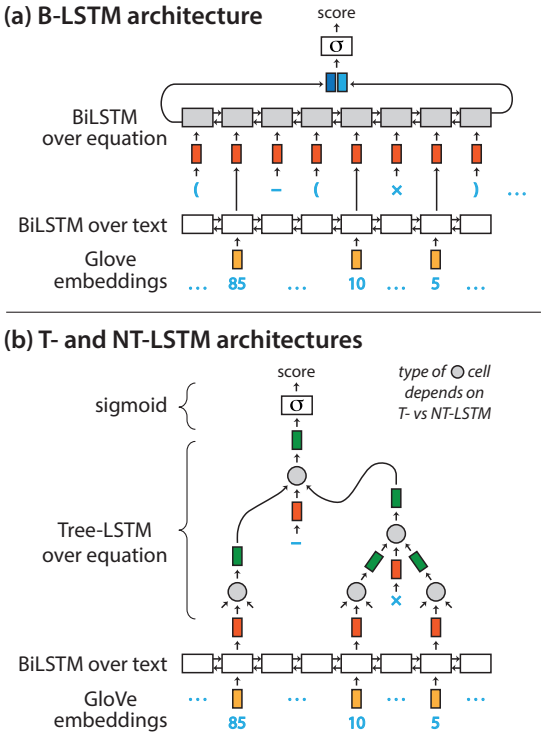

Figure 3: Models for scoring equations, taking the text and the equation from Fig. 1 to score (e.g., $85 - (10 \times 5)$) as input: (i) a word embedding layer at the bottom, (ii) a BiLSTM layer over the text, and (iii) a top layer that encodes the equation. For the latter we consider either **(a)** a sequential BiLSTM (B-LSTM architecture), or **(b)** a structured Tree-LSTM (T-LSTM and NT-LSTM architectures).

$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right) \tag{1}$$
$$o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right) \tag{2}$$
$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \tag{3}$$
$$u_t = \tanh\left(W_u x_t + U_u h_{t-1} + b_u\right) \tag{4}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot u_t \tag{5}$$
$$h_t = o_t \odot \tanh(c_t) \tag{6}$$

where $t \in \{1, ..., N\}$ represents a particular recursive execution time step and $h_t \in \mathbb{R}^{d_2}$ is the LSTM hidden state. The advantage of using the LSTM-based structure instead of a simpler recursive formulation, such as $h_t = \tanh(Wx_t + Uh_{t-1} + b)$, is that an LSTM model avoids the problems of exploding or vanishing gradients during the training process discussed in Hochreiter & Schmidhuber (1997); Bengio et al. (1994). This is achieved by using additional weight matrices and *gates* σ in Eqs. 1–3 in order to regulate the amount of information from previous execution steps h_{t-1} and current

input x_t that affect the current state h_t.[3] More concretely, W_i, W_f, W_o, $W_c \in \mathbb{R}^{d_2 \times d_1}$ and U_i, U_f, U_o, $U_c \in \mathbb{R}^{d_2 \times d_2}$ are the weight matrices related to different LSTM gates, and b_i, b_f, b_o, $b_c \in \mathbb{R}^{d_2}$ are the respective biases. In our experiments we initialize x_t with GloVe word embeddings (Pennington et al., 2014) and keep them static during training. These *GloVe embeddings* are depicted at the bottom of graphs **(a)** and **(b)** in Fig. 3. In order to obtain the BiLSTM representation (*"BiLSTM over text"* in Fig. 3), we run two LSTMs in different directions and concatenate the respective hidden states. This results in N hidden state representations $H = \{h_1^{(b)}, ..., h_N^{(b)}\}$ where $h_i^{(b)} \in \mathbb{R}^{d_3}$ and $d_3 = 2 \cdot d_2$. Using the input in H, we propose two different models to encode the candidate equations referred to as **(a)** and **(b)** in Fig. 3, and explained below:

(a) Sequential B-LSTM: We perform an in-order traversal of the expression tree to obtain a sequential representation of the equation (e.g., $(85 - (10 \times 5))$) that is encoded using a second BiLSTM (see *"BiLSTM over equation"* in Fig. 3**(a)**). We use as input the hidden state representations H calculated above for the numbers and (trainable) embeddings $O = \{o_-, o_+, o_\div, o_\times, o_(, o_)\}$ for the operators $(-, +, \div, \times)$ and opening/closing parentheses. More formally, the input to BiLSTM is represented by $X^E = \{x_1^e, ..., x_K^e\}$ where $x_t^e \in \{H \cup O\}$, $x_t^e \in \mathbb{R}^{d_3}$ and K is the number of tokens in the equation, including parentheses and operations. E.g., the equation $(85 - (10 \times 5))$ contains 9 tokens. In terms of the formal notation of LSTM in Eqs. 1–6, each x_t^e corresponds to input vector x_t. In order to obtain a score for ranking the equation, we concatenate the last (left and right) hidden states of the BiLSTM producing a vector of dimensionality d_4, and then apply a linear transformation followed by a *sigmoid* function.

(b) Tree-LSTM: We base our implementation on the Tree-LSTM architecture proposed by Tai et al. (2015). This architecture is based on the LSTM formulation described in Eqs. 1–6, but instead of being linearly linked, the input to a particular LSTM cell can come from different child step LSTM executions. More formally, we can describe the T-LSTM structure as follows:

[3] For a more detailed description of the LSTM architecture please refer to Hochreiter & Schmidhuber (1997).

$$\tilde{h}_t = \sum_{k \in \{L,R\}} h_{t-1}^k \tag{7}$$

$$i_t = \sigma\left(W_i x_t + U_i \tilde{h}_t + b_i\right) \tag{8}$$

$$o_t = \sigma\left(W_o x_t + U_o \tilde{h}_t + b_o\right) \tag{9}$$

$$f_t^k = \sigma\left(W_f x_t + U_f h_{t-1}^k + b_f\right) \tag{10}$$

$$u_t = \tanh\left(W_u x_t + U_u \tilde{h}_t + b_u\right) \tag{11}$$

$$c_t = i_t \odot u_t + \sum_{k \in \{L,R\}} f_t^k \odot c_{t-1}^k \tag{12}$$

$$h_t = o_t \odot \tanh\left(c_t\right) \tag{13}$$

where $\{L, R\}$ is the set that consists of left (L) and right (R) child nodes for the current execution node at step t. More specifically, a particular execution step t corresponds to the respective arithmetic operation in the expression tree (see Fig. 1(c)). This step takes as input the cell (c) and hidden (h) states of previous execution step ($t-1$) for each of the child nodes ($\{L, R\}$) that correspond to left and right operands in the expression tree. This execution process is recursive: each of the execution steps produces as output a hidden state h_t (Eq. 13) which is used by the parent execution step recursively in Eq. 7 either as left (h_{t-1}^L) or right (h_{t-1}^R) child. Additionally, a *cell state* c_t is passed across the execution steps, and contains a summarized historic information of the tree traversal[4] operations performed so far. Similarly as with LSTM, a *forget* gate f_t^k, *input* (i_t) and *update* (u_t) gates are used to determine which historic information is kept (forget gate) and which new information is added (input/update gates) to the cell state. W_i, W_o, W_f, $W_u \in \mathbb{R}^{d_4 \times d_3}$ together with U_i, U_o, U_f, $U_u \in \mathbb{R}^{d_4 \times d_4}$ are the weight matrices that transform the inputs $x_t \in \mathbb{R}^{d_3}$, the current hidden state $\tilde{h}_t \in \mathbb{R}^{d_4}$ and the children's hidden states $h_{t-1}^k \in \mathbb{R}^{d_4}$, by means of the Tree-LSTM gate representations. As depicted in Fig. 3**(b)**, the inputs x_t to the leaf nodes are the hidden state representations in H (coming from *"BiLSTM over text"* in Fig. 3**(b)**) on the positions where the numbers occur in the problem statement. The input x_t to the inner nodes, on the other hand, are one of the randomly initialized operation embeddings $O = \{o_-, o_+, o_\div, o_\times\}$ depending on the operation represented by the node. This contrasts with the original setup proposed in Tai et al. (2015) where the input x_t always comes from the word representation in the sentence. By using a separate operation embeddings set O as input, we expect our model to be able to capture a semantic representation for each of the different operations $o \in O$. The Tree-LSTM model finally outputs the hidden state for the root of the expression tree (i.e., the last executed operation), which is then passed through a sigmoid to deliver the score for a particular candidate arithmetic expression.

While T-LSTM allows to encode the equation information in a tree structure, it

[4]Post-order traversal is used, since it reflects the order of operator execution in an arithmetic equation to obtain the final result.

is symmetric in its child nodes. This is because the hidden states of the children are first summed up in Eq. 7 before applying the linear transformation and the gate activation functions. This could be problematic for non-commutative operations ($-$ and $\div$) where the result depends on the order of the operands. The reason for this is that Eq. 7 is commutative with respect to child nodes. Thus, given two child nodes $k \in \{L, R\}$ we have that $\tilde{h}_t = h^L_{t-1} + h^R_{t-1} = h^R_{t-1} + h^L_{t-1}$. As a consequence, the affine transformations U_i, U_o, and U_u in Eqs. 8, 9 and 11 cannot capture the order of the states of the input nodes. Furthermore, since there is only one weight matrix U_f for both h^L_{t-1} and h^R_{t-1} in Eq. 10, it can not apply a different affine transformation for left and right child nodes. This makes the T-LSTM model indifferent to the order of the arguments of the operations in a particular expression tree. Therefore, we introduce a second model, called NT-LSTM, that uses distinct weight matrices to transform each of the children's hidden states. More formally, the gate definition in NT-LSTM is as follows:

$$i_t = \sigma\left(W_i x_t + \sum_{k\in\{L,R\}} U^k_i h^k_{t-1} + b_i\right) \tag{14}$$

$$o_t = \sigma\left(W_o x_t + \sum_{k\in\{L,R\}} U^k_o h^k_{t-1} + b_o\right) \tag{15}$$

$$f^k_t = \sigma\left(W_f x_t + \sum_{l\in\{L,R\}} U^{kl}_f h^l_{t-1} + b_f\right) \tag{16}$$

$$u_t = \tanh\left(W_u x_t + \sum_{k\in\{L,R\}} U^k_u h^k_{t-1} + b_u\right) \tag{17}$$

$$c_t = i_t \odot u_t + \sum_{k\in\{L,R\}} f^k_t \odot c^k_{t-1} \tag{18}$$

$$h_t = o_t \odot \tanh\left(c_t\right) \tag{19}$$

where, similarly as for T-LSTM, $\{L, R\}$ is the set of child nodes. By introducing different weights U for each of the child node states h^k_{t-1}, we make sure that the model can differentiate between the order of the operands. This is because now each of the affine transformations $U^{(l)}_i$, $U^{(l)}_o$ and $U^{(l)}_t$ is different for each input child hidden state h^l_{t-1} in Eqs. 14, 15 and 17. Similarly, each of the children's ($k \in \{L, R\}$) forget gates f^k_t contains now two affine transformations U^{kl}_f ($l \in \{L, R\}$), one for each child. This way, the model can prioritize (components of f^k_t close to 1) or inhibit (components of f^k_t close to 0) separately the input of a particular child k based on the state of another child l ($k \neq l$). This can be useful when the state of one of the operands (e.g., influenced by the words that surround a particular number in text) has a strong indication of some operation, while the state of the other has very little evidence. As we will show in Section 5, the use of NT-LSTM makes a big difference compared to the performance of T-LSTM for equations involving non-commutative operations.

4. Experimental setup

We evaluate the proposed models (code publicly available[5]) on the SingleEQ dataset introduced by Koncel-Kedziorski et al. (2015). SingleEQ consists of 1,117 sentences and 15,292 words, and includes 508 arithmetic problems of varying complexity (i.e., equations with single or multiple operators). Each of the word problems is mapped to a single correct equation with one unknown. These equations include one or more of the following operators: multiplication ($\times$), division ($\div$), subtraction ($-$), and addition ($+$). The data was gathered from the following grade-school websites: `http://math-aids.com`, `http://k5learning.com`, and `http://ixl.com` as well as from a subset of problems from Kushman et al. (2014). To obtain results comparable to previous work, we perform 5-fold cross-validation using the original splits defined in Koncel-Kedziorski et al. (2015). Similar to the work of Koncel-Kedziorski et al. (2015) and Wang et al. (2018b), we report performance using the overall accuracy metric. The training/testing process is run for 5 different splits, in each one a separate fold is left as test set. This way, our results are reported on the whole SingleEQ dataset by concatenating the predictions of *test* folds across the splits. In total, we train 25 models with different seeds (5 for each split) and report average and standard deviation in Tables 4–5 and 7 in Section 5. Furthermore, we tune the neural net hyperparameters independently for each of the splits on the validation set that consists of 20% randomly selected arithmetic problems in each of the train folds. Due to limited resources that prevented us to perform a complete grid search, we conduct the hyperparameter tuning in steps. More specifically, in each step we perform a grid search on two hyperparameters that we identified as most correlated with each other. Table 2 summarizes our hyperparameter search space for each of the sequential tuning steps. Besides the usual hyperparameters (i.e., learning rate, batch size and dropout) tuning, we also adjust the dimensionalities d_3 (Dim LSTM) of the first BiLSTM layer (indicated as "*BiLSTM over text*" in Fig. 3), and d_4 (Dim Encoder) of either the sequential BiLSTM ("*BiLSTM over equation*" in Fig. 3) or the tree-based NT-LSTM models' encoder layers ("*Tree-LSTM*" in Fig. 3). The best hyperparameters are chosen after training for 75 epochs for each of the cross-validation splits independently.

Step	Hyperparameters				
	Learning Rate	Batch Size	Dim LSTM	Dim Encoder	Dropout
1	{3e−4, 1e−4}	{64, 128}	-	-	-
2	-	-	{256, 512}	-	{0.3, 0.4}
3	-	-	-	{256, 512}	{0.3, 0.4}

Table 2: The range of the hyperparameter search space for each of the hyperparameter tuning steps for each of the cross-validation splits of SingleEQ dataset.

Furthermore, we partition the dataset into several subsets to investigate the effect of varying problem complexity on the models' performances. These different subsets are characterized in Table 3. We form three main categories: (i) **Full:** the whole dataset is

[5] `https://github.com/klimzaporojets/arithmetic-word-problems`

Subset	Equation types	# Problems
Full	All operators	508
Single	Single operator	390
Multi	Multiple operators	118
$\text{Single}_{\text{sym}}$	Single symmetric operators	208
$\text{Multi}_{\text{sym}}$	Multiple symmetric operators	68
$\text{Single}_{\text{asym}}$	Single asymmetric operators	182
$\text{Multi}_{\text{asym}}$	Multiple asymmetric operators	50

Table 3: The defined subsets of the SingleEQ dataset with varying degrees of complexity.

included in this setting, (ii) **Complexity:** two subsets (i.e., Single, Multi) are formed based on the number of operators in the solution's equation, and (iii) **Symmetry:** four main subsets, namely $\text{Single}_{\text{sym}}$, $\text{Single}_{\text{asym}}$, $\text{Multi}_{\text{sym}}$, and $\text{Multi}_{\text{asym}}$ are formed to indicate whether the solution's equation contains single/multiple symmetric ($\times$ and $+$) or asymmetric ($\div$ and $-$) operations.

We hypothesize that our Tree-LSTM models will exhibit stronger performance on subsets involving multiple and/or non-commutative operations (Multi, $\text{Multi}_{\text{sym}}$, $\text{Multi}_{\text{asym}}$), since they should be able to better capture the semantic relationships between operator nodes encoded in a tree structure. We also expect a significant difference between T-LSTM and NT-LSTM architectures on subsets involving non-commutative operations ($\text{Single}_{\text{asym}}$ and $\text{Multi}_{\text{asym}}$). By using different weight matrices to transform each of the children's states (see Eqs. 14–17 of the NT-LSTM in Section 3.2 for more details), the NT-LSTM model should be able to capture the order of the operands and link the resulting structural information of a particular non-commutative mathematical expression to the semantic representation of the problem statement.

We obtain the top-100 equation-trees using the ILP solver of Koncel-Kedziorski et al. (2015), which we rank using scores provided by our proposed model (see Section 3.2). Training of our model is performed using the Adam optimizer (Kingma & Ba, 2015). As a bottom token representation layer, we use pre-trained 100-dimensional ($d_1 = 100$) GloVe embeddings (Pennington et al., 2014)[6] which we keep static during the training process.

5. Results

In this section, we evaluate the performance of our proposed models on the SingleEQ dataset. Besides the performance on the full dataset, we are particularly interested in evaluating how each architecture behaves when evaluated on arithmetic problems of varying complexity. We assume that the problems become more complex (i) as the number of needed mathematical operators grows, and (ii) when the used operators are non-commutative (asymmetric). We hypothesize that our structured Tree-LSTM-based

[6] https://nlp.stanford.edu/projects/glove/

Model	Features	Trees	Accuracy (%)
Hosseini et al. (2014)	✓	✗	48.00
Wang et al. (2018b)	✓	✓	52.96
Roy & Roth (2015)	✓	✓	66.38
Roy & Roth (2017)	✓	✓	72.25
ALGES	✓	✓	72.39
ILP Coverage	-	-	91.34
ILP Naive	-	-	52.56
B-LSTM	✗	✗	74.88±0.64
T-LSTM	✗	✓	74.88±1.06
NT-LSTM	✗	✓	**75.47±0.62**

Table 4: Accuracy attained by the proposed and state-of-the-art methods on the *Full* SingleEQ dataset. The ✓ and ✗ symbols indicate whether or not a model adopts hand-crafted features ('Features') or tree-structured encoding of the equations ('Trees'). The best result is typeset in **bold**.

approach is better suited to solve the aforementioned complex problems. In order to demonstrate this, we perform an extensive evaluation (Tables 4–5 and 7) of our models on subsets of different degree of complexity as defined in Table 3. Furthermore, in all of the result tables we include the potential maximum accuracy that can be achieved when using the candidates from the ILP *candidate generator* (ILP Coverage). This allows us to estimate how much improvement can still be achieved by *candidate ranker*. Conversely, in order to evaluate the impact of *candidate ranker* models, we also report the accuracy achieved when picking the top-weighted candidate by ILP solver (ILP Naive).

Comparison on the Full dataset: Table 4 shows the results of the evaluated systems on the Full SingleEQ dataset. The proposed models are the (i) B-LSTM, (ii) T-LSTM, and (iii) NT-LSTM as presented in Section 3.2. Clearly, all newly proposed architectures outperform previous methods. Concretely, our methods are able to outperform strong baselines on the task, reporting an accuracy improvement of more than 3% without relying on hand-crafted features (Hosseini et al., 2014; Koncel-Kedziorski et al., 2015; Roy & Roth, 2015, 2017). As detailed later on in this section (see analysis of Table 5 and Table 7), most of this improvement with respect to the current state-of-the-art (Koncel-Kedziorski et al., 2015) comes from an increased performance on the more complex arithmetic word problems that involve non-commutative and multiple operations. This supports our original hypothesis that tree-based architectures are superior in representing mathematical operations between operands, specially when the mathematical expressions involve multiple operations. The hand-crafted features, used in previous works, are usually related to terms indicating specific operations and thus if they are not detected in the data, the system cannot generalize well on out-of-domain mathematical descriptions. This also applies to recent neural-based methods (see, e.g., Wang et al. (2018b)) where explicitly defined features are encoded in the neural structure. Furthermore, in order to ensure the validity of the differences between our proposed approaches, we carry out a bootstrap significance analysis (Efron & Tibshirani, 1994) by sampling with replacement the results of B-LSTM, T-LSTM, and NT-LSTM

Model	Complexity		Symmetric		Asymmetric	
	Single	Multi	$\text{Single}_{\text{sym}}$	$\text{Multi}_{\text{sym}}$	$\text{Single}_{\text{asym}}$	$\text{Multi}_{\text{asym}}$
ILP Coverage	93.33	84.75	94.71	83.82	91.76	86.00
ILP Naive	56.41	39.83	53.85	69.12	59.34	0.00
ALGES	77.69‡	54.70‡	**89.90**	72.06	63.74‡	30.64‡
B-LSTM	79.59±0.72	59.32±2.34	80.87±0.64‡	69.12±2.08‡	78.13±1.36	**46.00±4.38**
T-LSTM	79.59±1.24	59.32±1.61	81.35±0.98‡	**72.35±1.44**	77.58±2.72	41.60±2.33⋆
NT-LSTM	**80.21±0.95**	**59.83±1.75**	81.35±1.44‡	71.17±2.20	**78.90±2.13**	44.40±4.96

Table 5: Comparison of the proposed methods with the state-of-the-art on the SingleEQ dataset in terms of accuracy. **Bold** font indicates the best results for each subset of SingleEQ (see Table 3). The markers ⋆, †, ‡ respectively indicate the achieved bootstrap significance levels α <0.1, <0.05 and <0.01 with respect to the best performing model in each of the subsets.

models 10,000 times. We compare the performance with respect to the NT-LSTM model in Table 4. We observe that, while our NT-LSTM model seems to outperform T-LSTM and B-LSTM models, this difference in performance is not significant.

Comparison for different problem complexity: Table 5 compares our models with ALGES (Koncel-Kedziorski et al., 2015) (i.e., the best performing state-of-the-art model of Table 4), for subsets of different complexity levels (defined in Table 3). We use bootstrap significance testing to estimate the degree of certainty between the lower performing models and the best performing one in each of the subsets. We indicate significant differences with p-values below the 1%, 5%, and 10% level (respectively denoted with ‡, †, and ⋆) in order to identify models performing significantly different from the best performing model in each of the subsets.

We observe that our newly proposed models do not significantly differ among each other for solving problems involving single (Single, $\text{Single}_{\text{sym}}$, and $\text{Single}_{\text{asym}}$ subsets) operations. Conversely, on the problem subset requiring multiple commutative operations in their solution ($\text{Multi}_{\text{sym}}$), our tree-based T-LSTM significantly outperforms the sequential B-LSTM model, suggesting a potential benefit in using tree-based models to solve the problems involving multiple operations. For the subset involving multiple non-commutative operations ($\text{Multi}_{\text{asym}}$) the B-LSTM and NT-LSTM models outperform the T-LSTM model, indicating a potential limitation of the latter in dealing with non-commutative operations, due to its symmetrical structure in its child nodes (a single weight matrix is used on the sum of children's states $\tilde{h}_t$ as described in Section 3.2). We were surprised by an overall good performance of our sequential B-LSTM model, specially on $\text{Multi}_{\text{asym}}$ subset, where it performs on par with the potentially more expressive NT-LSTM model. This fact also motivated us to explore the robustness of our models against additional asymmetric noise (see further analysis in the next paragraphs corresponding to the results in Table 7).

The results in Table 5 further show that the feature-based ALGES model has competitive performance on problems requiring single and/or non-commutative operators in the solution equations. In fact, it significantly outperforms all our models on the $\text{Single}_{\text{sym}}$ dataset and is only marginally outperformed by our tree-based T-LSTM model on $\text{Multi}_{\text{sym}}$. This suggests that the feature-based ALGES is able to explicitly capture symmetric operations by focusing on carefully engineered features. However, we observe a large drop in performance of ALGES on problems that require non-

Candidates	Metric	Subsets						
		Full	Single	Multi	$\text{Single}_{\text{sym}}$	$\text{Multi}_{\text{sym}}$	$\text{Single}_{\text{asym}}$	$\text{Multi}_{\text{asym}}$
ILP	Correct	2.53	1.44	6.13	1.89	7.72	0.92	3.96
	Incorrect	12	2.9	42.08	2.48	28.43	3.38	60.64
ILP + Asym	Correct	2.41	1.44	5.62	1.89	7.66	0.92	2.84
	Incorrect	15.08	4.06	51.5	3.57	35.43	4.62	73.36
	Δ Correct	-4.74%	0.00%	-8.32%	0.00%	-0.78%	0.00%	-28.28%
	Δ Incorrect	25.67%	40.00%	22.39%	43.95%	24.62%	36.69%	20.98%

Table 6: This table illustrates the difference in average number of *Correct* and *Incorrect* candidate equations per problem between the original *ILP* candidate generation process and the one obtained by adding noisy equations with asymmetric operators (*ILP + Asym*).

commutative (asymmetric) operations to be solved. This is showcased by a difference of more than 15% accuracy points on $\text{Single}_{\text{asym}}$ and $\text{Multi}_{\text{asym}}$ subsets in Table 5. This validates our initial intuition that feature-based models fall short to capture the reasoning necessary to address problems that require more complex (non-commutative and multiple) operators.

Robustness against asymmetric noise: The results analyzed so far are based on scoring the candidates generated by the ILP component introduced in Koncel-Kedziorski et al. (2015). However, this component already significantly reduces the number of incorrect candidates, particularly those involving asymmetric operators (e.g., by removing candidate equations that produce negative or fractional results as described in Section 3.1). In order to evaluate the robustness of the proposed models, we train and evaluate them on a noisy asymmetric candidate set where we add all possible permutations to the equations involving non-commutative operators. For example, if a particular candidate equation is $x = 8/2$, we would also add $x = 2/8$ to the candidate set. Table 6 shows the statistics of the noisy dataset (ILP + Asym) with respective deltas that indicate the percentage points (%) of increase/decrease in the average number of correct/incorrect candidate equations per problem with respect to the original ILP-generated candidate set. We observe a significant increase in the number of incorrect candidates for all subsets, as well as a drop in average number of correct equations for the subsets involving asymmetric operations (Multi and $\text{Multi}_{\text{asym}}$). This is because, similarly as in the original *ILP* setup, we only consider the first 100 generated candidates, which in *ILP + Asym* include more incorrect equations, leaving many correct ones out. This results in a lower correct/incorrect ratio that makes it more challenging for the evaluated models to find the right mathematical expression to solve a particular problem. Table 7 compares our models with the best performing state-of-the-art model (i.e., ALGES) on candidates generated in the *ILP + Asym* setting. Compared to the results presented in Table 5, we observe a sharp decrease in performance of the ALGES model on subsets involving multiple operations (Multi, $\text{Multi}_{\text{sym}}$ and $\text{Multi}_{\text{asym}}$). This demonstrates once more the weakness of this feature-based model in capturing the reasoning necessary to distinguish the order of the operands involved in equations containing multiple and non-commutative operators. Furthermore, we observe that the sequential B-LSTM model is now significantly outperformed by the tree-based NT-LSTM on subsets involving multiple operations to be solved (Multi, $\text{Multi}_{\text{sym}}$ and $\text{Multi}_{\text{asym}}$).

Model	Full	Complexity		Symmetric		Asymmetric	
		Single	Multi	$\text{Single}_{\text{sym}}$	$\text{Multi}_{\text{sym}}$	$\text{Single}_{\text{asym}}$	$\text{Multi}_{\text{asym}}$
ILP Coverage	91.14	93.33	83.90	94.71	83.82	91.76	84.00
ILP Naive	52.56	56.41	39.83	53.85	69.12	59.34	0.00
ALGES	68.44‡	75.90†	43.59‡	**85.58**	61.76‡	64.83‡	18.36‡
B-LSTM	72.99±1.14	**78.21±0.97**	55.76±2.10‡	83.36±1.20†	71.76±3.40‡	72.30±2.37	34.00±2.19*
T-LSTM	57.95±1.34‡	61.69±1.49‡	45.59±1.25‡	80.58±2.44‡	72.65±2.20‡	40.11±0.92‡	8.80±0.98‡
NT-LSTM	**73.19±0.93**	76.97±1.02†	**60.67±1.15**	80.76±2.37‡	**76.47±0.93**	**72.63±1.61**	**39.20±2.40**

Table 7: Comparison of the proposed methods with the state-of-the-art model (i.e., ALGES) on the SingleEQ dataset in terms of accuracy evaluated on candidate equations generated using *ILP + Asym* procedure (see Table 6). **Bold** font indicates the best results for each subset of SingleEQ (see Table 3). The markers $\star$, †, ‡ respectively indicate the achieved bootstrap significance levels α <0.1, <0.05 and <0.01 with respect to the best performing model in each of the subsets.

This again supports our initial hypothesis that tree-structured approach is better suited to capture more complex reasoning which is necessary to solve arithmetic problems. In the *ILP + Asym* candidate generation setting this is even more important because of the additional noise introduced with the incorrect candidates that involve multiple and asymmetric operations. Conversely, for arithmetic problems involving single operations to be solved (Single, $\text{Single}_{\text{sym}}$, and $\text{Single}_{\text{asym}}$ subsets), the B-LSTM model shows a competitive performance, surpassing the tree-based NT-LSTM model on problems requiring single commutative operations ($\text{Single}_{\text{sym}}$). Additionally, we observe an important drop in performance of T-LSTM model which is mainly influenced by low accuracy scores on asymmetric subsets ($\text{Single}_{\text{asym}}$ and $\text{Multi}_{\text{asym}}$). This is in line with our initial intuition that by using a single weight matrices U_i, U_o, U_f, U_u to transform either the sum of the children's states $\tilde{h}_t$ (see Eqs. 7–9 and 11) or the individual children states h_k (Eq. 10), the T-LSTM model is unable to distinguish the order of the operands involved in asymmetric equations. This difference is less evident in Table 5 because most of the incorrect candidates involving non-commutative operations are already filtered out by the ILP component. However, in our *ILP + Asym* candidate generation setup, we make sure that for each candidate involving non-commutative operation, we also include noisy candidates with all the possible asymmetric permutations. This makes it necessary not only to detect the right operation, but also to distinguish the order of the operands, where the T-LSTM model fails. Finally, we observe that overall (on Full dataset) our tree-based NT-LSTM model exhibits less variance among the different bootstrap results, compared to the sequential B-LSTM model. This indicates that NT-LSTM model is less susceptible to different seed initialization during the training process, making it more robust than other proposed models (T-LSTM and B-LSTM).

Error Analysis: In order to understand our system's weaknesses, we manually analyzed the errors that it consistently makes across different training seed instances. We grouped them into three main categories represented in Table 8: *complex reasoning*, *parsing and counting*, and *world knowledge* errors. We observe that more than half (57%) of our system's errors are due to problems requiring *complex reasoning* while the numbers have been correctly extracted from the text. This reflects the results from Tables 5 and 7 that show lower performance of our models on problems requiring multiple and/or non-commutative operations. As future work to alleviate this type of problems we can complement the tree-structures using additional information such as

Type	Problem Text	NT-LSTM
Complex reasoning (57%)	Seth bought 20 cartons of ice cream and 2 cartons of yogurt. Each carton of ice cream cost $6 and each carton of yogurt cost $1. How much more did Seth spend on ice cream than on yogurt?	$20/2 - 1 \times 6$
Parsing and counting (22%)	Jane's dad brought home 24 marble potatoes. If Jane's mom made potato salad for lunch and served an equal amount of potatoes to Jane, herself and her husband, how many potatoes did each of them have?	n/a
World Knowledge (21%)	Bert runs 2 miles every day. How many miles will Bert run in 3 weeks?	3×2
	The sum of three consecutive odd numbers is 69. What is the smallest of the three numbers?	n/a

Table 8: Examples of problems where our NT-LSTM model fails.

the entities inside the sentence. For instance, in the first example illustrated in Table 8, if the system would know that "ice cream" from the second sentence represents the same concept as in the first one, it would be easier to link numbers 6 and 20. A second consistent type of error is related to *parsing and counting*. It mainly happens when there are several entities involved in a problem statement and the system has to count them correctly. For instance, in the second example presented in Table 8, our current system is unable to produce the correct candidate mathematical expression since it can only extract the number 24 from text. Further work in improving aspects related to parsing and entity identification in the problem statement should significantly reduce this kind of mistakes. Finally, the *world knowledge* related errors account for 21% of the total mistakes. Most of these errors are due to the fact that the system is unable to capture the units correctly (i.e., there are 7 days in a week, or one dime equals 0.1 dollars). However, as in the second example, some of the problems require a more advanced conceptual world understanding, such as the notion of odd numbers. Future work can be directed towards methods that are able to capture and represent this kind of world knowledge.

Limitations of the current state-of-the-art: We performed an empirical study on the predicted results to understand better where our proposed model outperforms the current state-of-the art model, ALGES (Koncel-Kedziorski et al., 2015). Table 9 illustrates some examples of the problems where our model gets consistently correct predictions on different training initialization weights (Section 4). Most of the gains came from improving on problems requiring multiple and/or asymmetric operations, corroborating our previous findings.

Strengths of the current state-of-the-art and limitations of our approach: Tables 5 and 7 illustrate that in the case of single symmetric operations ($\text{Single}_{\text{sym}}$), the

Problem Text	ALGES	NT-LSTM
Nancy bought 615 crayons that came in packs of 15. How many packs of crayons did Nancy buy?	$615 - 15$	$615/15$
Carrie's mom gave her $91 to go shopping. She bought a sweater for $24, a T-shirt for $6, and a pair of shoes for $11. How much money does Carrie have left?	$91 + 24 + 6 + 11$	$91 - (24 + 6 + 11)$
Melanie had 19 dimes in her bank. Her dad gave her 39 dimes and her mother gave her 25 dimes. How many dimes does Melanie have now ?	$19 - 39 + 25$	$19 + 39 + 25$
On Saturday, Sara spent $10.62 each on 2 tickets to a movie theater. Sara also rented a movie for $1.59, and bought a movie for $13.95. How much money in total did Sara spend on movies?	$10.62{+}2{\times}1.59{+}13.95$	$10.62{\times}2{+}13.95{+}1.59$

Table 9: Examples of problems that NT-LSTM provides a correct solution, but current state-of-the-art ALGES (Koncel-Kedziorski et al., 2015) fails.

ALGES method outperforms the proposed architectures (i.e., B-LSTM, T-LSTM, and NT-LSTM). We hypothesize that the main reason for this is the use of carefully hand-engineered features, many of which depend on third-party tools (e.g., dependency parsing). Table 10 illustrates four examples whose solution requires mathematical expressions with a single operator. In the first two cases our NT-LSTM model is outperformed by the current state-of-the-art ALGES which correctly predicts the commutative operators ($+$ in the first example and $\times$ in the second one). We have found that these correctly predicted commutative cases are highly correlated with the *entity match* feature (i.e., when the noun phrase connected to the number such as "pounds" in the first example is the same in two numbers). This feature has high positive correlation with addition and negative correlation with multiplication operations, which is illustrated in the first and second examples respectively. It also requires an additional dependency parsing which, in case of ALGES, is performed using Stanford Dependency Parser[7]. Other word-based features are also highly correlated with some operations. For example, the presence of the word "and" in the description of the problem is correlated with addition. However, while these features may be a strong indicators of some operators, their application is limited to problems where the underlying patterns appear. This is illustrated in the last two examples that contain two features highly correlated with the addition (i.e., *entity match* and "and" word), but that require a different (non-commutative) operation in their solutions. In both cases, biased by the most likely

[7]More concretely, the Stanford Dependency Parser in CoreNLP 3.4 is used.

Problem Text	ALGES	NT-LSTM
Diane is a beekeeper. Last year, she harvested 2,479 **pounds** of honey. This year, she bought some new hives **and** increased her honey harvest by 6,085 **pounds**. How many pounds of honey did Diane harvest this year?	$6,085 + 2,479$	$6,085 - 2,479$
Jack has a section filled with short story booklets. If each booklet has 9 **pages** and there are 49 **booklets** in the short story section, how many pages will Jack need to go through if he plans to read them all?	9×49	$9 + 49$
Benny received 67 **dollars** for his birthday. He went to a sporting goods store **and** bought a baseball glove, baseball, **and** bat. He had 33 **dollars** left over. How much did he spent on the baseball gear?	$67 + 33$	$67 - 33$
Janes mom picked cherry tomatoes from their backyard. If she gathered 56 **cherry tomatoes and** is about to place them in small jars which can contain 8 **cherry tomatoes** at a time, how many jars will she need?	$56 + 8$	$56/8$

Table 10: Examples of problems that require a single operation to be solved. The first two involve commutative operations ($+$ and $\times$ respectively) where our NT-LSTM model fails compared to the feature-based model (ALGES; Koncel-Kedziorski et al. (2015)). The rest of the examples illustrate cases where ALGES fails and NT-LSTM returns the correct answer. The words that represent features used in ALGES that are highly correlated with the predicted operation (*entity match* and the word *"and"*) are highlighted.

feature-based operation, the answer given by ALGES is incorrect. This contrasts with our feature-independent NT-LSTM model which manages to predict the correct equation. This is reflected in Tables 5 and 7, where the features-based approach falls short in capturing the more intricate nature of solutions involving non-commutative operations ($\text{Single}_{\text{asym}}$ and $\text{Multi}_{\text{asym}}$). In these cases, our tree-based NT-LSTM model exhibits superior performance.

6. Conclusion

In this work we addressed the reasoning component involved in solving arithmetic word problems. We proposed a recursive tree architecture to encode the underlying equations for solving arithmetic word problems. More concretely, we proposed to use two different Tree-LSTM architectures for the task of scoring candidate equations. We performed an extensive experimental study on the SingleEQ dataset and demonstrated

consistent effectiveness (i.e., more than 3% increase in accuracy on the Full dataset and more than 15% for a subset of complex reasoning tasks) of our models compared to current state-of-the-art.

We observed that, while very strong on simple instances involving single operations, the current feature-based state-of-the-art model exhibits a significant gap in performance for mathematical problems whose solution comprises non-commutative and/or multiple operations. This reveals the weakness of this method to capture the intricate nature of reasoning necessary to solve more complex arithmetic problems. Furthermore, our experiments show that, while a traditional sequential approach based on recurrent encoding implemented using BiLSTMs over the equation proves to be a robust baseline, it is outperformed by our recursive Tree-LSTM architecture to encode the candidate solution equation on more complicated problems that require multiple operations to be solved. This difference in performance becomes more significant as we introduce additional noise in our set of candidates by adding incorrect equations that contain non-commutative operations.

Acknowledgment

Part of the research leading to these results has received funding from (i) the European Union's Horizon 2020 research and innovation programme for the CPN project under grant agreement no. 761488, and (ii) the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

References

Amini, A., Gabriel, S., Lin, S., Koncel-Kedziorski, R., Choi, Y., & Hajishirzi, H. (2019). Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 2357–2367).

Bakman, Y. (2007). Robust understanding of word problems with extraneous information.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157–166.

Bobrow, D. G. (1964). Natural language input for a computer problem solving system.

Charniak, E. (1968). Carps: a program which solves calculus word problems.

Charniak, E. (1969). Computer solution of calculus word problems. In *Proceedings of the 1st international joint conference on Artificial intelligence* (pp. 303–316). Morgan Kaufmann Publishers Inc.

Chen, Q., Zhu, X., Ling, Z.-H., Wei, S., Jiang, H., & Inkpen, D. (2017). Enhanced LSTM for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1657–1668).

Chiang, T.-R., & Chen, Y.-N. (2019). Semantically-aligned equation generation for solving and reasoning math word problems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 2656–2668).

Efron, B., & Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

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

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hosseini, M. J., Hajishirzi, H., Etzioni, O., & Kushman, N. (2014). Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 523–533).

Huang, D., Liu, J., Lin, C.-Y., & Yin, J. (2018). Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics* (pp. 213–223).

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*. San Diego, USA.

Koncel-Kedziorski, R., Hajishirzi, H., Sabharwal, A., Etzioni, O., & Ang, S. D. (2015). Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, *3*, 585–597.

Kushman, N., Artzi, Y., Zettlemoyer, L., & Barzilay, R. (2014). Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 271–281). volume 1.

Li, J., Wang, L., Zhang, J., Wang, Y., Dai, B. T., & Zhang, D. (2019). Modeling intra-relation in math word problems with different functional multi-head attentions. In *Proceedings of the 57th Conference of the Association for Computational Linguistics* (pp. 6162–6167).

Ling, W., Yogatama, D., Dyer, C., & Blunsom, P. (2017). Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 158–167).

Mitra, A., & Baral, C. (2016). Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 2144–2153). volume 1.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543).

Roy, S., & Roth, D. (2015). Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1743–1752).

Roy, S., & Roth, D. (2017). Unit dependency graph and its application to arithmetic word problem solving. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Roy, S., & Roth, D. (2018). Mapping to declarative knowledge for word problem solving. *Transactions of the Association of Computational Linguistics*, *6*, 159–172.

Shi, S., Wang, Y., Lin, C.-Y., Liu, X., & Rui, Y. (2015). Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1132–1142).

Socher, R., Bauer, J., Manning, C. D. et al. (2013). Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 455–465). volume 1.

Socher, R., Lin, C. C., Manning, C., & Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 129–136).

Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (pp. 1556–1566). volume 1.

Wang, L., Wang, Y., Cai, D., Zhang, D., & Liu, X. (2018a). Translating a math word problem to a expression tree. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 1064–1069).

Wang, L., Zhang, D., Gao, L., Song, J., Guo, L., & Shen, H. T. (2018b). Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Wang, Y., Liu, X., & Shi, S. (2017). Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 845–854).

Zhang, D., Wang, L., Zhang, L., Dai, B. T., & Shen, H. T. (2019). The gap of semantic parsing: A survey on automatic math word problem solvers. IEEE.