Towards Automatic Numerical Cross-Checking: Extracting Formulas from Text

Yixuan Cao$^{1,2}$, Hongwei Li$^{1,2}$, Ping Luo$^1$, Jiaquan Yao$^3$

$^1$Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing 100190, China
$^2$University of Chinese Academy of Sciences, Beijing 100049, China
$^3$School of Management, Jinan University, Guangzhou 510632, China
{caoyixuan, lihongwei, luop}@ict.ac.cn, {jiaquanyao}@gmail.com,

ABSTRACT

Verbal descriptions over the numerical relationships among some objective measures widely exist in the published documents on Web, especially in the financial fields. However, due to large volumes of documents and limited time for manual cross-check, these claims might be inconsistent with the original structured data of the related indicators even after official publishing. Such errors can seriously affect investors’ assessment of the company and may cause them to undervalue the firm even if the mistakes are made unintentionally instead of deliberately. It creates an opportunity for automated Numerical Cross-Checking (NCC) systems. This paper introduces the key component of such a system, formula extractor, which extracts formulas from verbal descriptions of numerical claims. Specifically, we formulate this task as a DAG-structure prediction problem, and propose an iterative relation extraction model to address it. In our model, we apply a bi-directional LSTM followed by a DAG-structured LSTM to extract formulas layer by layer iteratively. Then, the model is built using a human-labeled dataset of tens of thousands of sentences. The evaluation shows that this model is effective in formula extraction. At the relation level, the model achieves a 97.78% precision and 98.33% recall. At the sentence level, the predictions over 92.02% of sentences are perfect. Overall, the project for NCC has received wide recognition in the Chinese financial community.

1 INTRODUCTION

Claims over the numerical relationships among some objective measures widely exist in the published documents on Web. For example, various financial documents (e.g., IPO prospectus, bond prospectus, corporate annual report etc.) contain large fraction of verbal descriptions over the finance indicators of the corporates. Example 1 in Fig. 1 is such a typical sentence. It describes the amount of sum of prepayment and other payables for a company at two time point (the end of 2015 and 2016), and also the share of this sum in current liability. Also, Fig. 1 includes another two examples of such verbal descriptions from the fields of natural sciences (computer science and meteorology, respectively). All these sentences aim to verbally describe the “accurate” numerical relationships of some objective indicators in a quantitative way.

Even though these claims of numerical relationships are published officially, they might be inconsistent with the structured data of the indicators, which generate the verbal description. See a real-world example for this inconsistency in Fig. 2, which is detected automatically by our proposed system. From the given sentence, we can get the formula related to the indicators. Note that it is only one of the formulas which exactly match the semantics of the sentence. The document, which includes this sentence, also contains a table of these finance indicators. Putting these original data from table into the left side of the formula, we find that prepayment and other payables aggregates constituted 91.13% of current liability in 2015. This conflicts with the number 93.88% given in the text.

Since the essential requirements for disclosure documents, especially in the financial areas, are “authenticity, accuracy and completeness”, these numerical errors might bring about huge reputation risk, and even economic losses. In 2011, Goldman Sachs made one critical typo in several-hundred pages of filings – wrote “×” when they wanted “/” . That caused the trading price spiking and the trading was eventually suspended. This typo resulted in $45 million loss for Goldman$^1$. Recently, in 2016 Postal Savings Bank of China (PSBC) announced its annual report, in which the total liability was written to “¥ 80,000 billion” whereas the actual value should be “¥ 8 billion”. The news spread online rapidly and severely harmed the reputation of PSBC among investors$^2$. Since these disclosure documents usually have the force of law, these errors and typos should be thoroughly removed before publishing.

---

This paper is published under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution. WWW 2018, April 23–27, 2018, Lyon, France © 2018 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC BY 4.0 License.

1http://www.economist.com/node/18744559?story_id=18744559
2http://www.cs.f.cn/p20161223000079.html
For the years ended December 31, 2015 and December 31, 2016, we had prepayment and other payables in an aggregate amount of approximately $1,890 million, $1,743 million, constituted 93.88% and 93.14% of current liabilities, respectively.

Output formulas:

\[ \frac{\text{Dec 31, 2015 prepayment} + \text{Dec 31, 2015 other payables}}{\text{Dec 31, 2016 prepayment} + \text{Dec 31, 2016 other payables}} = \frac{\$1,890\text{ million}}{\$1,743\text{ million}} = 1.1 \]

\[ \frac{\text{Dec 31, 2015 prepayment} + \text{Dec 31, 2015 other payables}}{\text{Dec 31, 2015 current liabilities} + \text{Dec 31, 2015 other payables}} = \frac{\$1,890\text{ million}}{\$1,743\text{ million} + \text{Dec 31, 2015 current liabilities}} = 1.2 \]

\[ \frac{\text{Dec 31, 2015 prepayment} + \text{Dec 31, 2015 other payables}}{\text{Dec 31, 2016 current liabilities} + \text{Dec 31, 2016 other payables}} = \frac{\$1,890\text{ million}}{\text{Dec 31, 2016 current liabilities} + \$1,743\text{ million}} = 1.3 \]

\[ \frac{\text{Dec 31, 2015 prepayment} + \text{Dec 31, 2015 other payables}}{\text{Dec 31, 2016 current liabilities}} = \frac{\$1,890\text{ million}}{\text{Dec 31, 2016 current liabilities}} = 1.4 \]

**Example 1**

**Example 2**

The second part takes 50% of running time in whole algorithm.

Output formulas:

\[ \text{The second part} \div \text{whole algorithm} = \frac{0.5}{2} = 0.25 \]

**Example 3**

In the period from 1880 to 2012, the global average (land and ocean) surface temperature has increased by 0.85 [0.65 to 1.06] °C.

Output formulas:

\[ \text{global average surface temperature} \div 1880 \text{ global average} \text{ surface temperature} = \frac{0.85}{1880} = 0.45 \]

Figure 1: Examples of sentences and their formulas.

Figure 2: Numerical Cross-Checking detects inconsistency between verbal description and original data.

It well documented in finance literature that information in the disclosure documents is always one of the greatest concerns to the investors. Lawrence [14] shows that individual investors invest more in firms with clear and concise financial disclosure, since such disclosures reduce individuals’ relative information disadvantage. Choudhary et al. [7] point out that even immaterial errors in the documents contain information about firm’s financial reporting reliability. Hence, accounting errors may cause the investors to lose trust in the firms reporting quality, and decrease investments in these firms. Besides, investors’ attention to the accounting errors can lead to damage to firms’ reputation.

To this aim, there is a special job called “authorized reading” to conduct Numerical Cross-Checking manually. Since the original data tables are updated frequently during writing, the first draft of disclosure often contains many data-inconsistency errors. Based on user interview of 10 Chinese investment bankers, on average it takes one employee with 3-year work experiences one week for the task of cross-check over a 500-page document. Additionally, there is usually a hard deadline to publish the disclose documents, thus the time left for cross-check is limited. More importantly, conducting cross-check for a long period of time definitely induces fatigue, tiredness, and carelessness. Thus, even after manual cross-check, these data-inconsistency errors are still inevitable due to the large volumes of documents, frequent updates of the original table, limited time for cross-check, and largely the fatigue resulted from this intellectually demanding, laborious, time-consuming process.

Therefore, this challenge creates an opportunity for automated Numerical Cross-Checking (NCC) systems. As there is simply no existing system that truly does this automatically, NCC technology is clearly falling behind. There are some related systems developed. ClaimBuster [10] is a fact-checking system that aims to automatically check important factual claims, especially claims in political discourses. Technically, it focused on detecting check-worthy factual claims while the other two components of matching claims and checking claims are still ongoing. Currently, it cannot support precise cross-check over numerical indicators. StatCheck [20] uses rule-based program to check inconsistency errors in the null-hypothesis significance testing, presented in the academic papers in major psychology journals. It finds that one in eight papers contains a grossly inconsistent p-value that may have affected the statistical conclusion. While the relevant tools and techniques can assist this task in various steps, a full-fledged, end-to-end solution to NCC does not exist.

To fill this gap, we are building an end-to-end system for computer-assisted Numerical Cross-Checking which uses machine learning, natural language processing, and database query techniques to automate this task. It expects a financial document as input and detects all potential conflicts in numerical relationships, where Fig. 2 is a typical example output by this system. In an evaluation over 1,000 officially published Chinese prospectus, our system found that 68.92% of them contain data-inconsistency errors, and on average each document with inconsistency has 4.26 errors (confirmed by professionals’ re-check).

The system has three major components: formula extraction (extract numerical relationships from text), table extraction (extract data from tables), and consistency checking. While the improvement of the full-fledged system is still ongoing, in this paper we focus on the key component of the system, formula extractor, which extracts formulas from verbal descriptions of numerical claims.
Given the plethora of discourses and narratives exposed on Web, this component extracts the formulas, which are semantically expressed by the claim sentences. Fig. 1 shows some typical examples of input and output of this component. This is essentially the key step to automate the cross-check process.

We propose an iterative relation extraction (IRE) model to extract formulas from text. Fig. 3 illustrates how Formula 1.3 in Fig. 1 is converted into a directed acyclic graph (DAG) structure. In this graph each node refers to a binary relation, whose type can be any computation relation (say +, −, ×, ÷ etc.) or any compare relation (say =, <, > etc.). The two operands for a relation can be entities (see the leaves in the bottom), or the relations at the lower layers. See the node in the red solid rectangle as an example. It is a relation of ÷ with two operands at the lower layers. Its left operand refers to the node with a + relation, representing the sum of prepayment and other payables (at the end of 2015), while its right operand refers to a " loosen" node 3, representing the value of current liability at 2015 year end. In this case the two operands are both the relations generated at the lower layers. Also, see the root node with a "=" relation, with the left operand of "= " and the right one of the leaf entity 93.88%. This root node actually represents the whole formula. Thus, any formula can be represented as DAG, in which each internal node has two children and the nodes at the lower layers can be the children of the nodes at the upper layers. It means that the relations generated at the lower layers can be recursively used as the operands of some new relations at the upper layers. This is why we call this process iterative relation extraction. Previous studies of relationship extraction mostly focus on the relations among entities, thus they only perform relationship extraction for a single layer. In the proposed IRE model, it extracts relations from bottom to top iteratively until a formula is generated finally. In this process the resultant relations extracted at the lower layers can be used as the input of the relationships at the upper layers. In this way, the layer-by-layer iterative computing process generates the ultimate formula.

In this paper we develop the neural-based method for IRE. In this model we consider the sequential information in two directions, namely horizontally and vertically. In horizontal direction we consider the word sequence of the original input sentence. In vertical direction we model the sequences from leaves to root in the DAG for formula generation. Specially, we use two different LSTM modules to model the sequential information in these two directions so that the building of a relation depends on not only the relations generated earlier, but also the semantics embedded in the original sentence. Finally, the model is trained on a labelled dataset of tens of thousands of sentences, which are collected from Chinese public financial documents. The evaluation result shows that our model archives at 97.78% precision and 98.33% recall at relation level. And on 92.02% of sentences, it predicts the formulas without any mistakes. Additionally, to accelerate the training process, two techniques are developed: parallel by layer and parallel by batch. With these two techniques the average training time for one epoch speeds up more than 5 times. These techniques might shed some light on speeding up models with tree or DAG structure inputs.

Last but not least, our study makes contribution to the measurement error problem in the proxies of accounting errors and reporting bias by introducing more precise proxy of accounting errors. This problem is currently a big concern and remains unsolved in the literature. Researchers tend to rely on restatement data to construct proxies for accounting errors, and focus on examining the possible reasons and consequences of these errors. However, there exists measurement error problem in the existing proxies, which is caused by the limited sample size of restatement data [8]. The errors undetected or detected but untracked by the restatement data can lead to underestimating the incidence of accounting errors. With NCC system, we are able to find out errors in a much larger scope of disclosure documents, which makes us to estimate accounting errors more precisely.

The rest of the paper is structured as follows: Section 2 gives a definition of formula and its structure. Section 3 describes our iterative relation extraction model. Then we evaluate our model in Section 5, summarize related work in Section 6 and conclude in Section 7.

### 2 STRUCTURE OF FORMULAS IN TEXT

In this section we will first define the structure of one formula, then extend that to the structure of all formulas in one sentence, and finally discuss the characteristic of the structure. The discussion in this section is based on Fig. 4. Notice that Fig. 3 is a subgraph of Fig. 4 (nodes with solid border). And we use the operator to represent each relation instead of long expression for brevity.

#### 2.1 Structure of One Formula

Intuitively a formula is a formatted expression about something equal, or greater / less than another thing. For example, revenue at 2017 equals some amount of money; the growth rate of revenue from 2016 to 2017 equals a number. A formula is constructed by components including entities and relations. We will introduce these components first and then define the structure of a formula.

---

3The meaning of @ will be detailed in Section 2.
**Relation.** From Layer 1 to Layer 4 in Fig. 4, we see patterns that two nodes in lower layer point to one node in upper layer. That structure is called relation.

**Definition 2.1. Relation.** A relation consists of three parts: left operand \( c_l \), right operand \( c_r \), and operator \( o \):

\[
\text{r : } (c_l, o, c_r)
\]

where each operand \( c_x \) could be an entity or another relation, and \( o \) is an operator like “+” chosen from a predefined operator set.

Notice on two things. First, the definition of relation is recursive: the two operands of a relation can be other relations. Second, the left operand and the right operand are not commutative \((a - b) \) is clearly different with \((b - a)\). Although some operators such as “+” and “=” are commutative mathematically, we make rules to force all operators to be non-commutative. For example, in Fig 4, \( r_{41} \) has two operands \( r_{33} \) and 93.88%. The first word in \( r_{33} \) “Dec 31, 2015” appears earlier than 93.88% in sentence. We set the rule that \( r_{33} \) states 93.88%. Since the comparison relations will not be used as the operands of other relations, we call them terminal nodes.

Using the definitions above, a formula is a structure consisting of relations and entities. With one comparison at the top, all the descendants, including arithmetic relations and entities, construct a formula.

### 2.2 Formulas in a Sentence

But one sentence might express multiple formulas whose structures are entangled together. This makes the structure of all formulas in a sentence more complex in several ways.

First, The structure of formulas in one sentence might have multiple roots, such as Fig. 4 illustrated. Second, one entity or relation could point to multiple relations, that is, the same occurrence of entities and relations can be reused multiple times by other relations. For example, “Dec 31, 2015” is the right operand of \( r_{11}, r_{12}, \) and \( r_{13} \). \( r_{21} \) is the left operand of both \( r_{33} \) and \( r_{33} \).

**Layer.** The structure is also layered. The dotted lines groups relations by layer. A layer of relations is all relations that have the same layer number. The layer number of a relation \( r : (c_l, o, c_r) \) is defined as \( l(r) = \max(l(c_l), l(c_r)) + 1 \). We set \( l(c_x) = 0 \) if \( c_x \) is an entity.

From the definition and analysis above, the structure of formulas in one sentence has the following characteristics:

1. a formula is defined in a recursive way from the bottom components: time, indicator and values, through several layers of nested relations, to a comparison;
2. it may have multiple roots;
3. relations and entities could be reused; and
4. every relation has two ordered operands and one operator.

The first three characteristics match exactly with the DAG (Directed Acyclic Graph) structure. The fourth characteristic constrains the DAG to an ordered, binary DAG, where each node is associated with an operator. Here the ordered binary DAG means that every node has exactly two non-commutative children.

Therefore, the problem of extracting formulas in a sentence is defined as extracting the ordered binary DAG structure of formulas.

### 3 FORMULA EXTRACTION

Motivated by the recursive property of formula, we come up with an iterative relation extraction (IRE) model to extract formulas layer by layer iteratively, from bottom to top. In this section, we first give an overview of our model. Then, we dive into all the modules, including the embedding, Bi-LSTM, and DAG-LSTM modules, to introduce the details.
3.1 Framework of IRE Model

The key idea of the IRE model is extracting relations layer by layer iteratively. It extracts one layer of relations by first generating relation candidates and then doing binary classifications. When it terminates, the nodes with positive labels from all layers constitute the formulas. The process is as follows.

Store the extracted time, indicator and value entities in sets $T, I, V$ respectively. Let operator set $O = \{\@, +, -, \times, \div, >, <, =\}$. Use set $N$ to record the nodes in the DAG structure. At the beginning, $N = T \cup I \cup V$.

First layer. First, generate all the possible candidates:

$$C = \{(n_1, o, n_2) | n_1, n_2 \in N, o \in O\} - N.$$  \hfill (2)

Then, predict all candidates in $C$. Suppose the nodes with positive labels are $N_i^+$, update $N = N \cup N_i^+$.

Repeat. Afterwards, we repeat the process in the first layer, where at layer $i$ the candidate set is generated using Eq. 2, but with updated $N$. Note that for a candidate of layer $i$, the operands come from $N_i$, which includes relations and entities of all lower layers $k, k < i$. The process terminates when there is a layer $j$, $N_j = \emptyset$, and $N$ contains the extracted structure of formulas.

In our work, we constrain that the first layer only contain the relations of “@”, and all the other layers do not include “@” relations.

Algorithm 1 Model framework

Extract time entities $T$, indicator entities $I$, and value entities $V$. Use subscript $a$ for arithmetic operator or relation, $c$ for comparison operator or relation.

Operators $O_a = \{\@, +, -, \times, \div\}$, $O_c = \{>, <, =\}$

Entities and arithmetic relations $R_a = T \cup I \cup V$

Comparison relations $R_c = \emptyset$

repeat

\begin{align*}
C_a &= \{(r_1, o, r_2) | r_1, r_2 \in N, r_1 \neq r_2, o \in O_a\} - N \\
C_c &= \{(r_1, o, r_2) | r_1, r_2 \in N, r_1 \neq r_2, o \in O_c\} - R_c \\
R_a^+ &= \text{predict}(C_a); \ R_c^+ = \text{predict}(C_c) \\
R_a &= R_a \cup R_a^+; \ R_c = R_c \cup R_c^+ \\
\end{align*}

until $R_a^+ = \emptyset$

return $R_a \cup R_c$

Algorithm 1 describes this process in detail. And we demonstrate this process by an example in Fig. 5. The input sentence is shown at the bottom. At each layer, the bottom row includes current entities and relations, and boxes above them are candidates. Each column (in the dotted rectangle) contains all operators between two operands (we only draw arrows on the operator “=” for brevity). Black arrow connects relation with its left operand, while red with the right operand. Positive candidates are highlighted in green. And the dashed arrows from the green boxes show that these nodes will be used as the operands in the higher layers. The process terminates after Layer 3 since no arithmetic candidates are positive, and thus no more candidates could be generated further.

In our work, the predict function is a neural network consists of three modules (Fig. 6): embedding module, Bi-LSTM module and DAG module. In the horizontal direction, the embedding module embeds words into high dimensional dense vectors, and the Bi-LSTM module informs each word its context. In the vertical direction, the DAG-LSTM unit extract relations iteratively from leaves to roots. All these three module are trained together and their parameters are updated simultaneously. The details are described below.

3.2 Embedding module and Bi-LSTM module

The embedding module contains a look-up table that maps each token in a sentence to a distributional representation: $(e_1, e_2, ..., e_n) = E(w_1, w_2, ..., w_n)$. Since each $e_i$ is isolated from each other, we feed them into a Bi-LSTM network to connect them together. For each $w_i$, Bi-LSTM concatenates the hidden states of forward and backward uni-directional LSTMs as its hidden state $h_i$. We can assume that $h_i$ contains long-term and short-term context information of $w_i$ [9, 22], which make the next step, namely growing the DAG structure from each token’s hidden state, possible.
3.3 DAG-LSTM module

In this paper, we consider only one type of DAG structure – ordered binary DAG. It can be to extend to other types of DAG [25].

We introduce this module by the feed-forward process, the process to predict formulas in a sentence. The key idea is to predict one layer at a time iteratively which has been discussed in Algorithm 1. Now we focus on the DAG computation part.

Operator Embedding. The operator is a very important component of a relation. For example, a division relation between two revenues might be compared with a fraction or percentage later, but should not be compared with some amount of money. On the contrary, if you only know two revenues have a relation, the information is ambiguous. Since entities are representing by hidden states, we make a special embedding matrix \( E_o \) to embed each operator \( o \) into a distributional representation \( o_t = E_o(o) \). \( E_o \) will be updated simultaneously with other parameters during training.

DAG-LSTM Unit. During the process of iteratively generating and classifying, for each candidate, we want to embed all information of that candidate into a hidden state. Then the hidden state can be used for classification, or as the representation of that candidate to be operand of other candidates. So we use the hidden states of operands and the embedding of operator to compute a candidate’s hidden state. LSTM is well suited to this work.

The input for a relation candidate \( r_i = (l, o, r) \) includes the hidden state \( h_l \) of the left operand, \( h_r \) of the right operand, and an embedding \( o_t \) of the operator. The following equations compute the hidden state of \( r_i \):

\[
\begin{align*}
i_t &= \sigma(W_i \cdot o_t + U_i \cdot [h_l, h_r] + b_i) \\
f_t &= \sigma(W_f \cdot o_t + U_f \cdot [h_l, h_r] + b_f) \\
f_t' &= \sigma(W_f' \cdot o_t + U_f' \cdot [h_l, h_r] + b'_f) \\
o_t &= \sigma(W_o \cdot o_t + U_o \cdot [h_l, h_r] + b_o) \\
\hat{c}_t &= \text{tanh}(W_c \cdot o_t + U_c \cdot [h_l, h_r] + b_c) \\
c_t &= f_t \odot c_{t-1} + f_t' \odot \hat{c}_t + i_t \odot \hat{c}_t \\
h_t &= \text{tanh}(c_t) \odot o_t
\end{align*}
\]

where \( \sigma \) denotes the logistic function, \( \odot \) denotes element-wise multiplication, \( W, U \) and \( b \) are weight matrices and bias vectors, and subscripts \( i, f, o, c \) indicate different parameters. Conventionally we call \( i_t, f_t, o_t \) as input, forget, and output gate.

This computation inherits the spirit of sequential LSTM which uses memory cells and gates to transfer long-term information. The difference is that we use two separate forget gates for the left child and right child respectively [25, 29].

To do classification, we apply a linear transformation followed by softmax function on \( h_t: s_t = \text{softmax}(W_t \cdot h_t) \), to predict whether the node is positive or not.

Our method fuse the information of operator into the output hidden vector, and use an universal binary classification for all operators. This approach should alleviate the subsequent relation extractions in the following layers. There is another approach that only use operands to compute the hidden state, and feed that hidden state to a binary classification specified by \( o \). The idea behind our approach is that by fusing the information of the children with the information of the operator explicitly, we allow the operator information to be transferred throughout the DAG structure by hidden states.

Tracking hidden states. To put all the DAG-LSTM units in one sentence together, we keep a matrix \( \mathbf{H} \) to stores all nodes’ hidden state. First, \( \mathbf{H} \) records all hidden state of tokens returned from Bi-LSTM. Then, iteratively, after predicting all candidates in one layer, the hidden states of positives are appended to \( \mathbf{H} \) and become available for the subsequent relations. Saving only the hidden states of the previous layer is not enough because a relation can have operand from any lower layers.

Go through the process by an example in Fig. 5. At the beginning of DAG-LSTM, we have \( \mathbf{H} = \{h_1(\text{ln}), h_2(2012), h_3(\text{tax payables}), h_4(\text{constituents}), h_5(20\%), h_6(\text{total liabilities}), h_{10}(\text{of})\} \).

At layer 1, there are two candidates \( c_1(\text{tax payables}, @, 2012) \), \( c_2(\text{total liabilities}, @) \). For the first one, using \( h_1, h_2, E_o(\text{of}) \) as input, compute the hidden state and classify. The same process for the second candidate. Assume we predict them as positives. Then we append \( h_{11}(\text{for } c_1), h_{12}(\text{for } c_2) \) to \( \mathbf{H} \).

At layer 2, many candidates are generated but their operands are \( r_1, r_2, r_3 \) in layer 1 and 20%. We get their hidden vectors \( h_{11}, h_{12}, h_y \) from \( \mathbf{H} \) respectively and compute the hidden vectors of candidates and classify. Assume only \( c_4 : (r_1, @, r_2) \) is positive. We append its hidden vector \( h_{13} \) to \( \mathbf{H} \).

At layer 3, all the hidden vectors required are \( h_{11}, h_7, h_{12}, h_{13} \). Only \( c_4 : (20\%, @, r_3) \) is positive. We append its hidden vector \( h_{14} \) to \( \mathbf{H} \). Since no arithmetic relations are positive, the process finishes.

4 TRAINING AND SPEEDUP

4.1 Data Preparation and Loss Function

During training, we have the ground truth data: the DAG structure of formulas. These are positive nodes in DAG. Other possible relation candidates are negative nodes. If we train a model to predict all candidate nodes correctly, we could ensure that the model will extract all formulas from sentence correctly. Therefore, these positive and negative candidates are enough to train a reliable model for prediction.

The loss function is defined as the overall cross entropy. Suppose all sentences we have are \( S \). Each sentence \( s^i \) have a candidate set \( C^i \) that includes both positive and negative relation candidates in the DAG. \( C^i \) contains \( n^i \) candidates: \( c^i_1, ..., c^i_{n^i} \). Their labels are \( y^i_1, y^i_2, ..., y^i_{n^i} \), and the probabilities of being positive from model are \( p(c^i_1), p(c^i_2), ..., p(c^i_{n^i}) \). Then the loss function is:

\[
L = - \sum_{x \in S} \sum_{c^i_j \in C^i} y^i_j \log(p(c^i_j)) + (1 - y^i_j) \log(1 - p(c^i_j))
\]

4.2 Speedup

The slow training speed is a bottleneck for the development of neural networks with complex input structure like our model. Especially, current implementations compute one example at a time which is poorly suited for GPU computation [5]. We propose methods that parallelize computation inside one sentence by layer, and parallelize among sentences. They are shown in Fig. 7.

One sentence might have thousands of relation candidates. The computed hidden state of one relation or entity might be the input
of another relation. So during training, we regard all candidates in one sentence as one example and compute them in one pass.

Parallel by layer in one sentence. As Fig. 5 shows, all candidates in the same layer share the same hidden states from lower layers. Therefore, one layer of candidates can be computed in parallel.

Parallel by layer and batch. Most current implementations of tree-LSTM in Theano or Torch have to compute one example at a time. Besides computing all candidates in the same layer in one sentence at a time, we compute all candidates in the same layer for all sentences in a batch in parallel.

5 EXPERIMENT

5.1 Data and settings

We collected a dataset consisted of tens of thousands of labelled sentences. Since understanding financial text requires domain knowledge, we trained several people to construct this dataset. Each sentence is labelled by three people to minimize the labelling errors. Fig. 8 gives some statistic about our dataset. Distributions of sentences with regard to number of words, relations, formulas, candidates and layers are shown in a) to e). Most sentences have 20-40 words, less than 50 relations, less than 10 formulas, up to hundreds of candidates, and 2 to 5 layers. But some sentences have more than 200 words, hundreds of relations, and thousands of candidates. Sub-figure f) depicts the number of relations of each operator. "@", "+", and "−" are the most common operators where "\times" and "÷" are about ten times rarer than them. The operators are very imbalanced. And the difference between the number of candidates and relations also indicates the imbalance between positive and negative samples.

The model is implemented in Theano [26]. We only keep 3500 most frequent tokens and the rest are represent by a special UNKNOWN token. We also assign each type of entities (time, value, and indicator) a special token. The word embedding size is 128. Bi-LSTM doubles the size of hidden state of each token. And the hidden size of relation is also 256. We use Adadelta [27] as our optimizer. The batch size is 16.

From the collected data we find descriptions about "growth rate" are popular. If we only use \{+,-,\times,\div\}, they require two relations. To reduce the layer number, we add special operators "\times", "\div". Similarly, we add "\div\" for "decline rate".

5.2 Effective evaluation

We evaluate the result in several ways to the performance at different levels. At relation level, we calculate the precision (P), recall (R), and F1-score (F1). At formula level, we also report these metrics where precision is defined as (number of formulas we predict correctly) / (number of formulas in ground truth). At sentence level, we calculate accuracy, the percentage of sentences whose formulas are all correctly extracted. The relation and sentence level metrics are more directly related to user experience for NCC.

To evaluate the effectiveness of our model, we compare it with a model that does not fuse the operator embedding into the hidden vector. That means when computing hidden vectors of one relation in Equation 3, we omit the \(W \cdot a_t\) term. The evaluation result in Table 1 shows that using operator embedding will improve the model performance, especially at formula and sentence level. Model with operator embedding predicts correctly on 92.02% of sentences, which is a very promising result.

Moreover, we analyze the model performance with regard to the number of relations and words. Formulas, relations, etc. which showed similar patterns. Fig. 9 illustrates the performance with regard to the number of relations in a sentence. Sentences are grouped by relation number. Blue solid line is the sentence level accuracy on different groups. Red dashed line is the percentage of sentences in that group in test data. There is a trend that the model is prone to make more errors on sentences with more relations (long sentences). There

---

Table 1: Results (%) on test set, with or without operator embedding.

<table>
<thead>
<tr>
<th></th>
<th>Relation</th>
<th></th>
<th>Formula</th>
<th></th>
<th></th>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
<td>F1</td>
<td>P</td>
<td>R</td>
<td>F1</td>
</tr>
<tr>
<td>With</td>
<td>97.78</td>
<td>98.33</td>
<td>98.06</td>
<td>96.53</td>
<td>96.45</td>
<td>96.49</td>
</tr>
<tr>
<td>Without</td>
<td>97.28</td>
<td>97.17</td>
<td>97.23</td>
<td>97.17</td>
<td>94.58</td>
<td>95.84</td>
</tr>
</tbody>
</table>

https://github.com/dasguptar/tree_lstm
https://github.com/afarnen/cumulonimbus/tree_mn
are two possible reasons: long sentences are hard intrinsically, and long sentences are relatively rare in training data. Only 1.24% of sentences have more than 40 relations, but the average accuracy in these groups is around 60%, which is still a non-trivial result (accuracy drops sharply for sentences with more than 50 relations, but there are only 11 sentences in that group).

5.3 Case Study

Fig. 11 gives some cases that the model prediction contains mistakes, which illustrates the typical problems we are facing. The detailed analysis on each case is shown in figure and we summarize as follows: a) the verbal expression might be ambiguous and confuse the model, but this should be avoided in formal documents; b) performance on some operators is worse than others, that should attribute to the lack of training data (in training data, “+,” “>” are about ten times rarer than “,” “<”); c) model might make mistakes when the expression is complicated; d) there are some complicate linguistic phenomenons hard to handle.

5.4 Speedup Result

Fig. 10 shows the training time of one epoch on ten thousand sentences. If no parallelization is applied, the average running time is 389.38s. Parallelizing by layer (parallel by layer and batch, where batch size=1) reduce the time to 193.00s. If we parallelize by layer and batch, and set batch size to 2, 4, ..., the training time keeps falling, and reaches 72.50s when batch size is 32. Comparing to 389.38s, it saves 81.36% of the training time, which means more than 5 times speedup. The result shows that methods we proposed reduce the training time significantly.

6 RELATED WORK

We introduce the related work in three aspects: some related NLP tasks, neural network models, and speeding up methods.

Related NLP tasks. The most related tasks include relation extraction and semantic parsing. Relation extraction has a long history. Most works focus on relations between two entities [1, 18, 19]. A recent work “numerical relation extraction” [17] try to extract relations that contain numbers such like “inflation rate(India, 10.9%)”. They stop at only one layer of relation, while our problem asks to take a step forward to extracting “relation over relation”, which is more difficult considering the complexity of structure.

Semantic Parsing is a wide concept [4] which maps a natural language sentence into a formal representation of its meaning. Our work is a type of semantic parsing that map natural language to the formula structure. Other semantic parsings include mapping natural language to database queries for question answering [3, 4, 15]. And Hershcochov et al. [11] mapped natural language to a predefined UCCA structure using transition-based parsing. By adding more operations to transition set, it solved the reentrancy, discontinuous, non-terminal problems. But in that work, every token in a sentence is a part of the final structure which is different with our problem and transition-based method is not suited for our problem due to the long distance relations.

A closely related problem is so-called “equation parsing” which extract the equation structure in a sentence. There are some preliminary works in this field. They assume one sentence contains one equation (our model can handle multiple formulas), and the equation is a tree structure while our problem need to extract DAG structure. Specifically, Roy et al. [21] extracted all “triggers” followed by predicting the structure based on Cockey–Younger–Kasami algorithm combined with SVM. It assume the structure to be projective. Koncel-Kedzior et al. [13] applied integer linear programming to generate candidate tree structures then chosen one of them with SVM.

Model structure. From the perspective of model structure, our work combines sequential LSTM and structured LSTM, and shares some basic idea with recursive neural network. Bidirectional LSTM is effective to represent tokens in its sentential context [9, 12]. It has been widely used in many works such as semantic parsing [11], relation extraction [19], neural machine translation [2, 6] etc.

Structured LSTM started from tree structure, and developed rapidly to DAG structure. Tree-LSTM extend LSTM from sequential to structural. There are many variations of Tree-LSTM which share the same idea but are slightly different in the calculation detail depending on the task. Leveraging the tree structure information of sentence could outperformed state of the art models in sentiment classification [25, 29]. And it is also applied in relation extraction [19] recently. DAG structured LSTM has been proposed recently and applied to many fields, like using DAG to represent the different semantic composition [28], using DAG RNN to connect all pixels in one image for scene labeling [23]. But most of tree and DAG structured LSTM concentrated on incorporating structure information as input to model to get a better representation, while in our work we use DAG-LSTM both to get a good representation and to do structure prediction.
Recursive neural network [24] is proposed for image segment and natural language parsing in a recursive way. The overall framework shares insights with ours, but there are some differences. First, our model want to understand the structure of formulas and may end up with multiple roots, while recursive NN want to get a global idea of one sentence or image and merged into one root finally. Second, recursive NN “merge” two nodes into one parent and think the parent contains the information of its children, so the children are removed from the subsequent process which is clearly not compatible with the reuse characteristic of formulas.

Speeding up neural network models. The wild usage of complex neural network in natural language processing has proposed an urgent demand for faster training speed. Moshe et al. [16] treats neural network as a computation graph, and computes all nodes with same “depth” in graph in parallel. That work implemented in tensorflow shows that the parallelization method has a big speedup power on tree input. Our work are built on Theano and shows significant speedup on DAG structure. Our method is build on pure Theano without additional wrappers, which shows it is not too complex to extend an implementation from sequential computing to parallel computing.

7 CONCLUSION

When investors find out inconsistent errors in disclosure documents, they may doubt the authenticity of the whole document, which can be quite a great cost to the firms at some time. In order to detect these accounting errors more effectively, this paper proposes the iterative relation extraction model to extract formula from text. This is a key component of our automatic Numerical Cross-Checking (NCC) platform. By defining formula as a DAG structure, our model is able to extract the DAG with high accuracy. Based on this model, out system is able to detect numerical inconsistencies in real applications and has become widely accepted in the Chinese financial community. The development of this system will be ongoing.

ACKNOWLEDGMENTS

The research work was supported by the National Natural Science Foundation of China under Grant No. 61473274, the National Key Research and Development Program of China under Grant No. 2017YFB1002104. Jiaquan Yao was partially supported by the National Natural Science Foundation of China under Grant No. 71502152.
REFERENCES

Science & Business Media.
translation by jointly learning to align and translate. ICLR (2015).
Parsing on Freebase from Question-Answer Pairs. In EMNLP.
2014. A deep architecture for semantic parsing. In ACL Workshop on Semantic
Parsing.
[5] Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D.
Manning, and Christopher Potts. 2016. A Fast Unified Model for Parsing and
Sentence Understanding. In ACL.
Neural Machine Translation with a Syntax-Aware Encoder and Decoder. In ACL.
characteristics of financial reporting, errors deemed immaterial by managers.
(2016).
recognition with deep bidirectional LSTM. In Automatic Speech Recognition and
Understanding (ASRU).
Automated Fact-Checking: Detecting Check-worthy Factual Claims by
ClaimBuster. In KDD.
Directed Acyclic Graph Parser for UCCA. ACL (2017).
parsing using bidirectional LSTM feature representations. TACL (2016).
[13] Rik Koncel-Kedziorski, Hamaneh Haji-shirz, Ashish Sabharwal, Oren Etzioni,
and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations.
ACL (2015).
Accounting and Economics (2013).
Neural symbolic machines: Learning semantic parsers on freebase with weak
supervision. ACL (2017).
[17] Aman Madaan, Ashish Mittal, G Ramakrishnan Mausam, Ganesh Ramakrish-
Supervision. In AAAI.
[18] Mike Mintz, Steven Bills, Rion Snow, and Jurafsky Dan. 2009. Distant supervision
for relation extraction without labeled data. In ACL.
on sequences and tree structures. ACL (2016).
Epskamp, and Jetie M. Wicherts. 2016. The prevalence of statistical reporting
Sentences to Grounded Equations. In EMNLP.
networks for scene labeling. In CVPR.
natural scenes and natural language with recursive neural networks. In ICML.
semantic representations from tree-structured long short-term memory networks.
ACL (2015).
[26] Theano Development Team. 2016. Theano: A Python framework for fast compu-
(2012).
Short-Term Memory for Semantic Compositionality. In NAACL HLT.
over recursive structures. In ICML.