Nested Relation Extraction with Iterative Neural Network

Yixuan Cao, Dian Chen, Hongwei Li, Ping Luo
Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS),
Institute of Computing Technology, CAS, Beijing 100190, China.
University of Chinese Academy of Sciences, Beijing 100049, China.
{caoyixuan, chendian, lihongwei, luop}@ict.ac.cn

ABSTRACT

Natural language is used to describe objective facts, including simple relations like "Jobs was the CEO of Apple", and complex relations like "the GDP of the United States in 2018 grew 2.9% compared with 2017". For the latter example, the growth rate relation is between two other relations. Due to the complex nature of language, this kind of nested relations is expressed frequently, especially in professional documents in fields like economics, finance, and biomedicine. But extracting nested relations is challenging, and research on this problem is almost vacant. In this paper, we formally formulate the nested relation extraction problem, and come up with a solution using Iterative Neural Network. Specifically, we observe that the nested relation structures can be expressed as a Directed Acyclic Graph (DAG), and propose the model to simultaneously consider the word sequence of natural language in the horizontal direction and the DAG structure in the vertical direction. Based on two nested relation extraction tasks, namely semantic causality relation extraction and formula extraction, we show that the proposed model works well on them. Moreover, we speed up the DAG-LSTM training significantly by a simple parallelization solution.

ACM Reference Format:
Yixuan Cao, Dian Chen, Hongwei Li, Ping Luo. 2019. Nested Relation Extraction with Iterative Neural Network. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3358003

1 INTRODUCTION

Relation Extraction (RE) is an important task in natural language processing. It is the foundation and data source for many downstream tasks such as knowledge base construction, question answering, etc. The task of RE is that given a sentence and entities in this sentence (such as persons and locations), determine whether there are relations among some of the entities, and what the relation types are. The traditional relation extraction problem extracts relations between two entities, such as PERSON-ORG relation between `Steve Jobs` and `Apple`.

There are two major research directions in this field: supervised and distant supervised (DS) RE. The main difference is that the training data of DS RE comes from some heuristic alignment between sentences and knowledge bases like FreeBase, whereas the training data of supervised RE comes from manual annotation. The research interest of DS approach focuses on how to handle the noisy data [10, 32]. Supervised RE assumes the training data is clean, and focuses more on model structure and external information incorporation [22, 30, 35].

The `GDP` in `China` was `$13.41 trillion` in `2018`, `$7.08 trillion` less compared with `U.S.`, `$1.17 trillion` more compared with `2017`.

Expressed facts:

1. (China GDP in 2018) = $13.41 trillion
2. (U.S. GDP in 2018) - (China GDP in 2018) = $7.08 trillion
3. (China GDP in 2018) - (China GDP in 2017) = $1.17 trillion

Relations:

R_{C8} : Economic-Index(China, GDP, 2018)
R_{C8e} : Equal(R_{C8}, $12.24 trillion)
R_{U8} : Economic-Index(U.S., GDP, 2018)
R_{U-C} : Subtract(R_{U8}, R_{C8})
R_{U-Ce}: Equal(R_{U-C}, $7.08 trillion)
R_{C7} : Economic-Index(China, GDP, 2017)
R_{8-7} : Subtract(R_{C8} - R_{C7})
R_{8-7e}: Equal(R_{8-7}, $1.17 trillion)

Figure 1: Example of nested relations

While our research adopts a supervised approach, we want to categorize RE from a different perspective: the complexity of relations to extract. Traditional relation extraction researches focus on extraction of simple relations: relations between two entities (binary relations). But real world relations can be complex.

Complex relations are expressed in natural language frequently, especially in economics, finance, biomedicine and other fields where people need to express in a precise way. A complex relation may have multiple participants and the participant may be another relation. For example, in the field of health, sentence "`2.5 mg Albuterol` may be used to treat `acute exacerbations`, particularly in `children`." expresses a relation with 4 participants (`Albuterol`, `acute exacerbations`, `2.5 mg`, `children`) to accurately describe the fact [9]. This kind of information appears in a vast amount of medical journals. Extracting the complete information from the corpus is the fundament of large scale knowledge base construction.

Another example shown in Figure 1 expresses relations about the economic statistics which could come from politicians who want to convey his/her opinion on certain issues. Since we are in a society congested with false-hoods, hyperboles and half-truths, a lot of efforts have been done on fact-checking of political discourses and scientific papers to ferret out misinformation [12, 23]. Accurate

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3358003

and complete structured relation extraction from those claims is a vital component to this end. To automatically vet the claim in Figure 1, we need a general extractor for world economic information. Specifically, we have to extract fact that involves time, economic index, country and value entities like `(China GDP in 2018)=$13.41 trillion`. Moreover, the claim about the difference between `(U.S. GDP in 2018)` and `(China GDP in 2018)` described by "$7.08 trillion less" also has to be extracted.

We summarize the complexity of relations into two aspects: high-arity and nested. Traditional RE extracts relations between two entities, such as `PERSON-ORG` relation between `Steve Jobs` and `Apple`. *High-arity* extends the concept of relation to be among multiple entities. *Nested* extends the concept of relation to be upon other relations. To describe the first fact of Figure 1, we first need a high-arity relation among three entities R_{C8}: `Economic-Index(China, GDP, 2018)`, where `Economic-Index` is the relation type. We say this relation R_{C8} has arity 3 (3 operands). In other words, this is a 3-ary relation. Secondly, we need a nested relation R_{C8e}: `Equal(`R_{C8}`, $13.41 trillion)`. Note that although R_{C8e} is a 2-ary (binary) relation, its first operand is another relation R_{C8}. This is different from traditional RE which only extract relations among entities. R_{U-C}, R_{U-Ce}, R_{8-7}, R_{8-7e} are all nested relations.

Although high arity and nested are different, they are related to each other. The R_{C8} relation in Figure 1 may be decomposed further into 2 binary relations: `((China, GDP), 2018)`. The relations are represented nested and relation types are omitted for brevity. More generally, for an ordered n-ary relation (the entities in relation are ordered), we can always decompose it into $n-1$ nested binary relations. However, using high-arity relation to represent nested relations may add complexity to the problem definition. For example, if we define a "+" relation to be between two entities or other "+" relations, nested relations like $((a+b)+c)$ can represent addition among 3 elements. Without nested relation, we have to define "2+" for 2-ary addition, and "3+" for 3-ary addition, which is inflexible and complex. From this aspect, nested relations are more flexible and expressive than high arity relations. So, in this paper, we focus on extraction of nested relations.

Researches on formulating and developing general models for nested RE are almost vacant as far as we know. In this paper, we give a formal formulation about nested relation extraction. Then, an Iterative Neural Network is proposed to extract nested relations layer by layer. It extracts one layer of relations at a time by generating all possible candidates and classifying them. The positive candidates become new relations to generate next layer candidates. This process iterates until no new candidates can be generated. The neural network has two major parts, the horizontal part includes Bidirectional-LSTM and attention mechanism to represent entities, and the vertical part is a relation representation neural network composed of DAG-LSTM which will grow as we extract more nested relations.

Speed of neural network is crucial for its prevalence in research and in practice. Parallelized computing promoted the widespread usage of CNN and RNN. One extreme is that although RNN has been optimized to process multiple sentences in parallel, since it has to process tokens serially, more parallelized models like attention-based Transformer [8] have emerged to replace RNN models on some tasks. With the development of deep learning, more and more complex structure neural networks are developed for different tasks. The tree- or DAG-LSTM used in this paper has been applied in many tasks [22, 26, 36]. In their task, tree-LSTM is applied on a given structure like dependency tree, with relatively few structure nodes. On the contrary, our model attempts to extract the structure by generating a lot of candidate nodes. That means tree-LSTM will be time consuming. Therefore, the naive serial implementation of DAG-LSTM is not suitable for GPU computation due to the lack of parallelization. In this paper, we introduce a simple algorithm to parallelize the computation of DAG-LSTM. The key idea is that all relations in the same layer in a batch can be computed in parallel. And the result shows that it can be hundreds of times faster than serial implementation when the number of candidates is huge.

Two example tasks that contain a large amount of nested relations were introduced in experiment. The semantic causality relation extraction task extracts semantic cause-and-effect relations between clauses or other cause-and-effect relations. People express cause-and-effect relations when they want to explain or support their opinions. Extracting this information may help collect cause-and-effect relations between real world events for common sense knowledge construction. And the result can also be used for fact-checking. The formula extraction task extracts financial indexes, and the arithmetic relations such as proportions and subtractions. This task involves highly nested relations (up to 4 or 5 layers) with diverse structures. The results on these tasks show the necessity of nested relation extraction and the effectiveness of our model.

2 NESTED RELATION EXTRACTION

Nested Relation Extraction is the problem that given a text and typed entities, extracts semantic relations among multiple elements (entities or other relations) expressed by text.

A relation extraction task is usually carried out following a predefined task-specific schema. The ACE 2004 relation extraction task [1] defined five general types of relations, and further subdivided into a total of 24 types/subtypes of relations. Only semantic relations fall into these categories will be annotated for training and extracted during inference. For the example shown in Figure 1, one task may want to extract all relations shown in figure, while another task only needs to extract the first fact and neglect the other two. Thus defining the schema of a task is important in real world problem, since this is the guide for annotation and model construction.

The schema of traditional binary relation between two entities is relatively simple. Here, we give a general framework to define schema for nested relation extraction task by introducing notions of *entity*, *relation*, and *configuration* of relations. For a specific nested RE task, we need to define the task schema at the beginning to formally define what kind of relations under which constraints to extract (but semantically what is a relation is not defined here).

Entity. An entity consists of a word sequence and an entity label (type), denoted as $l(w_i, ..., w_j)$. It is a mention of certain real world entity. The example shown in Figure 1 contains entities like

- `Index(GDP)`,
- `Country(China)`, and
- `Value($, 13.41, trillion)`

where Index, Country, Value are labels and GDP, China, $13.41 trillion are words in entities. The notation of relations in Figure 1 omitted the entity types. We denote the set of all kinds of entity labels in this task as E.

Note that this definition where entities are typed is in line with tasks like ACE 2004 and ACE 2005 [3]. But it is different from SemEval-2007 Task 4 and SemEval-2010 Task 8 in which all entities are "nominals" [11, 13]. Attaching labels to entities has advantages and disadvantages. The advantage is that we can shrink the search space and reduce the number of classification by constraining what kind of combination of entities may form a relation. The disadvantage is that we shift some of the burden to the preceding named entity recognition module, and errors from that module may propagate. In this paper, we attach labels to entities in order to automate the extraction process (introduced in section 3).

Relation. We use *element* to refer to a relation or an entity. One relation consists of a tuple of elements and a relation label (the type of the relation). The behavior of one type of relations is defined as follows. Suppose l is a label, L_i is a set of labels, R^l the set of elements with label l, and $R^{L_i} = \bigcup_{l \in L_i} R^l$. R^l satisfies

$$R^l \subseteq R^{L_1} \times \cdots \times R^{L_k}$$

which means R^l is a subset of Cartesian product of R^{L_i}. We call Cartesian product $L_1 \times \cdots \times L_k$ as the **configuration** of R^l, denoted as C^l. The behavior of relations with label l is defined by C^l. We denote L as the set of all relation labels in this task.

A tuple of elements $(e_1, ..., e_k) \in R^l$, or has l relation, if and only if

(1) this tuple satisfies C^l, that is $e_i \in R^{L_i}$ for all $i = 1, ..., k$, and
(2) the l relation among these elements is semantically described in the sentence.

We denote this relation instance as

$$l(e_1, ..., e_k),$$

and we call e_i the i-th operand of the relation.

For example, the configuration of Economic-Index(EI) relation is {Country,Region} × {Index} × {Year,Quarter }. That means the first operand of an EI relation must be an entity with label Country or Region, the second operand must be an entity with label Index, and the third operand must be an entity with label Year or Quarter. The formal notation of R_{C8} should be

EI(Country(China), Index(GDP), Year(2018))

where each of its operand is a typed entity instead of words. According to condition (1), tuples not satisfying the configuration, like (China, U.S., 2018), will never have EI relation. Suppose we add "The president of Canada said" at the beginning of the sentence in Figure 1. (Canada, GDP, 2018) satisfying the configuration of EI. However, according to condition (2), since it is not semantically described in sentence, this tuple does not have EI relation. This is why R^l is a subset of $R^{L_1} \times \cdots \times R^{L_k}$.

Schema. Based on the definitions above, a schema of a nested relation extraction task is defined by specifying the entity label set E, the relation label set l and the configuration of each relation $C^l, l \in L$.

Take the economic performance extraction task shown in Figure 1 as example. The entity label set E, relation label set L, and configurations of relations are as follows:

$$\begin{aligned} E =& \{\texttt{Country,Region,Index,Year,Quarter,Value}\} \\ L =& \{\texttt{Economic-Index(EI)}, +, -, \div, =\} \\ C^{\texttt{EI}} =& \{\texttt{Country,Region}\} \times \{\texttt{Index}\} \times \{\texttt{Year,Quarter}\} \\ C^{-} =& \{\texttt{EI,Value}, +, -, \div\} \times \{\texttt{EI,Value}, +, -, \div\} \\ C^{+} =& C^{\div} = C^{=} = C^{-} \end{aligned}$$

When the configurations of some relations contain other relations, the relation structure can become nested. − and ÷ relations are mutually included, so the relations can be arbitrarily nested in this task.

3 SOLUTION WITH ITERATIVE NEURAL NETWORK

The major challenge of nested relation extraction is that we cannot determine the total number of candidate tuples in a sentence, even though we know the number of entities. This is because relations can be arbitrarily nested. The traditional binary relation extraction or high-arity relation extraction problem extracts flatten relations. Therefore, they can enumerate the tuples of entities (candidates) to classify. However, when the relations become nested, only the lowest layer of candidates (among entities) may be generated and classified using the traditional methods. To solve this problem, we propose to generate candidate tuples for classification on the fly when new relations are extracted layer by layer. And the generating process follows the task schema.

To classify a candidate, some RE methods generate features based on the left and right context of two entities, including hand-crafted features and CNN-based features [32]. These methods are not applicable to nested situation where the operand of a relation may be another relation. A nested relation could involve arbitrary number of entities, and defining context of multiple entities is complicated. Fortunately, deep learning is good at representation. Existing methods generate a vector of relation for classification. But that vector may also represent the semantic meaning of a relation. We represent relations and entities by unified distributed representations, and classify based on these representations. Thus, we are able to extract nested relations layer by layer iteratively.

3.1 Iterative Neural Network

In this subsection, we will first give an overview about the process to iteratively generate and classify candidates. Then we introduce the model structure and details.

We define a *candidate* with label l as a tuple of elements that satisfies $(e_1, ..., e_k) \in R^{L_1} \times ... \times R^{L_k}$, where $C^l = (L_1, ..., L_k)$, but do not know whether $(e_1, ..., e_k) \in R^l$. So a candidate is a tuple that satisfies the configuration of its relation type, but whether its semantic meaning is correct or not is to be determined. We denote it as $l \leftarrow (e_1, ..., e_k)$.A left arrow is used to distinguish it with the notation of relation.

Process Overview. The overall process of iterative neural network is to extract relations layer by layer iteratively. There are three steps to extract one layer of relations:

(1) generate candidates according to configurations
(2) classify them

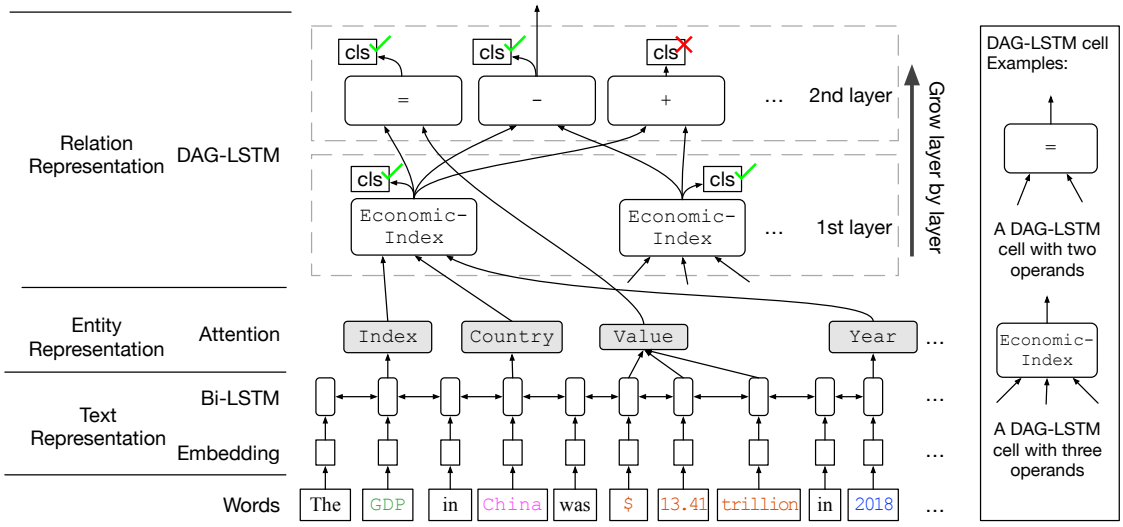

Figure 2: Model structure with an input example

(3) set the positive candidates as relations.

Notice that, in step 1, all candidates that can be generated (not generated in previous layers) from current state are generated. At the beginning of the i-th layer, we put all entities and relations extracted from previous layers into a set R, all candidates generated before into a set Q. The candidates we need to generate is the set:

$$\{ l \leftarrow (e_1, ..., e_{k_l}) \mid l \in L \wedge e_i \in R \wedge (e_1, ..., e_{k_l}) \text{ satisfies } C^l \wedge l \leftarrow (e_1, ..., e_{k_l}) \notin Q \}$$

To classify candidates, we propose an iterative neural network. It is composed of three major modules. In horizontal orientation, there are text representation module and entity representation module. And in vertical orientation, there is relation representation module. The relation representation module is suitable for the iterative generating and classifying process, since its structure is flexible.

The whole iterative neural network model is illustrated in Figure 2. We use the example in Figure 1 to show the model structure. The words are presented at bottom, followed by text representation model composed of embedding layer and Bidirectional-LSTM. Then each entity is represented using attention mechanism over its tokens. Finally, in relation representation model, hidden vectors of relations and candidates are computed layer by layer using DAG-LSTM cells. Hidden vectors of candidates are fed into fully connected networks for classification (shown as "cls" boxes). The positive ones (with check mark) might connect to the next layer. For example the "-" cell is fed into next layer because it is operand of other candidates. However, "=" cell is positive but not fed into next layer because it is not operand of any other candidates according to the task schema discussed in Section 2. And the negative ones (with cross mark) are discarded.

The text representation module consists of a word embedding layer and a bi-directional LSTM [15] layer. This module converts each word into a hidden vector h_i that encodes information about this word and its context.

The embedding layer has an embedding look up table, which is a large matrix. The i-th vector in the matrix is the embedding of the i-th word in vocabulary.

The uni-directional LSTM network is a variant of recurrent neural network. It takes as input a token sequence. At each step (each token), a LSTM cell combines previous step cell and hidden vectors and the current step input vector, to produce a new hidden vector:

$$h_t = f(h_{t-1}, c_{t-1}, x_t).$$

The hidden vector encodes information of the input sequence from the beginning to current step. Bi-directional LSTM contains two uni-directional LSTMs (one takes as input the forward sequence and the other the backward sequence). And hidden vector of each word is the concatenation of two LSTM outputs.

The entity representation module is used to represent entities. One entity might consist of multiple words. Thus, multiple hidden vectors of words are associated with an entity. However, a fixed length representation is wanted for an entity. So we apply a dot product attention [18] on words in this entity. Suppose the word vectors in this entity are $(h_1, h_2, ..., h_n)$, the hidden state h of entity is computed as:

$$s_i = w^T h_i, \text{ for } i = 1, ..., n$$
$$a_i = \frac{\exp(s_i)}{\sum_{j=1}^{n} \exp(s_j)}$$
$$h = \sum_{i=1}^{n} a_i h_i$$

where w is a trainable parameter.

The relation representation module applies DAG-LSTM to represent relations and candidates. The DAG-LSTM is an extension of Tree-LSTM [7, 26] that has been applied in many tasks [22, 26]. One DAG-LSTM cell combines hidden vectors of multiple operands into a vector to represent its corresponding relation or candidate. And multiple layers of DAG-LSTM cells will let the information

Table 1: Notions and notations used in speed up method.

Notation	Description
b	Number of sentences in a batch
n_a^e, n_a^r	Number of elements (entities and relations) in the a-th sentence in batch
n	$\max_{a=1,\ldots,b}\{n_a^e + n_a^r\}$
d	Dimension of hidden vector
H	$H \in R^{b\times n\times d}$. The tensor of hidden vectors of all relations
N	Number of relations in one layer in a batch
A, T, S	Each has N rows. $A_j = a, T_j = i_r, S_j = [i_{o_1}, \ldots, i_{o_k}]$ means that the j-th row of them record a relation r, which comes from the a-th sentence with index i_r, and its operand l has index i_{o_l}.

flow among the nested relations and candidates. One DAG-LSTM cell takes as input the hidden and cell states h_j, c_j from each of its operands, and the embedding e of the relation label. For simplicity, we concatenate hidden vectors into vector $v = [h_1, \ldots, h_k]$. The computation process is as follows:

$$\begin{aligned} i &= \sigma\left(W^i e + U^i v + b^i\right) \\ f_j &= \sigma\left(W_j^f e + U_j^f v + b_j^f\right), \quad j = 1, \ldots, k \\ o &= \sigma\left(W^o e + U^o v + b^o\right) \\ \hat{c} &= \tanh\left(W^c e + U^c v + b^c\right) \\ c &= i \odot \hat{c} + \textstyle\sum_{j=1}^{k} f_j \odot c_j \\ h &= \tanh(c) \odot o \end{aligned} \tag{1}$$

where W^*, U^*, b^* are network parameters, $\odot$ is element-wise product, h and c are hidden and cell states of this relation or candidate. To classify a candidate, we feed its hidden vector h into a multi-layer fully connected networks followed by Softmax function to compute its probability of correctness.

Training. The annotated data only contain correct relations. To train our model, we generate all possible candidates. Candidates appear in annotation results are positive samples, and others are negative samples. One sentence contains many samples (candidates), we compute though text and entity representation module for one time, which output the hidden vectors of all entities. Then all candidates in this sentence are computed based on these vectors. This will reduce redundant computations comparing to re-compute entity hidden vectors for each candidate.

The training objective is to minimize the negative log-likelihood over all candidates:

$$L = \sum_{s\in D} \sum_{c\in C(s)} y_c \log(p(c)) + (1 - y_c) \log(1 - p(c))$$

where D is dataset, s is one sentence, $C(s)$ is the set of all candidates in s, y_c is the label of candidate c, and $p(c)$ is the probability our model predict that c is a relation.

3.2 Speedup

Most speedup methods in neural network perform computation in parallel for all samples in a batch. This requires all samples in one batch have the same "size". For example, in CNN, all input pictures must be resized to the same hight-width-channel size; in RNN, all input sentences are truncated or padded into the same length.

The problem of nested relation extraction is that different sentences have different relation structures. However, as shown in Figure 2, the relation representation part can be decomposed by layer. Moreover, the computations of relations in the same layer are independent, because they only require hidden vectors of elements from lower layers. If we computed all hidden vectors of lower layers in advance, relations in the same layer can be computed in parallel in a vectorized way on GPU. The implementation overview is to make a large tensor to record hidden vectors of all relations, and fetch vectors as inputs or set resulting vectors by fancy indexing. We assume there is only one type of relation with k operands ($|L| = 1$), the extension to $|L| > 1$ is straightforward. Notions and notations used in this subsection are summarized in Table 1. Details are introduced below with a concrete example.

Suppose we are training the model. In one update step, b sentences are given in a batch, and the a-th sentence has n_a^e entities and n_a^r relations. For each sentence, we sort all relations and candidates according to their topological orders and put entities and relations into an element list. If an element has operands, its operands must positioned before it in the list. Then each element has an *index* in list from 0 to $n_a^e + n_a^e - 1$.

Take Figure 1 as example, during training, we know the number of relations, and how to layer them. $\mathsf{R_{C8}}$, $\mathsf{R_{U8}}$, and $\mathsf{R_{C7}}$ are in the first layer since they are based on entities. $\mathsf{R_{C8e}}$, $\mathsf{R_{U\text{-}C}}$, $\mathsf{R_{8\text{-}7}}$ are in the second layer, since one of their operands is in the first layer. $\mathsf{R_{U\text{-}Ce}}$ and $\mathsf{R_{8\text{-}7e}}$ are in the third layer, since one of their operands is in the second layer. So the list is (0.GDP, 1.China, ..., 7.2017, 8.$\mathsf{R_{C8}}$, 9.$\mathsf{R_{U8}}$, 10.$\mathsf{R_{C7}}$, ..., 15.$\mathsf{R_{8\text{-}7e}}$). Suppose this sentence is the 3rd sentence in the batch hereafter.

We allocate a large tensor $H \in R^{b\times n\times d}$ in advance, where $n = \max_{a=1,\ldots,b}\{n_a^e + n_a^r\}$, d is the hidden size, and $H_{a,i}$ is used to record the hidden vector of the element with index i in the a-th sentence. For example, $H_{3,8}$ records the hidden vector of $\mathsf{R_{C8}}$. We do not record hidden vectors of candidates in H since they will be sent to classification once computed, and not involved in the following computation. At the beginning, H is randomly initialized. Then vectors of entities are put in, followed by the vectors of the first layer relations, and so on.

For *one layer* of relations of b sentences in batch (N relations in total), we construct three matrices, sentence index matrix A, target element index matrix T and source element index matrix S, which record information about where to get the operand vectors and where to put the result vectors from and to H. Suppose one relation

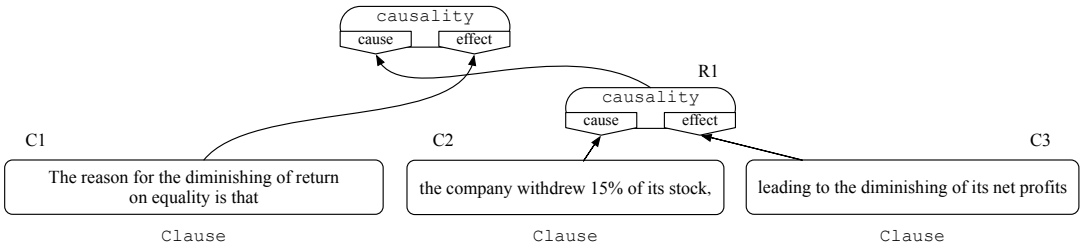

Figure 3: Examples for semantic causality.

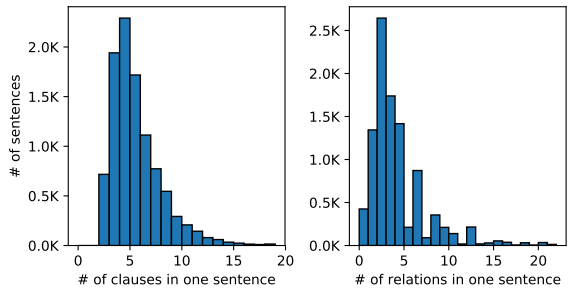

Figure 4: Distributions of semantic causality dataset.

in this layer has index i_r in the a-th sentence, and its k-th operand has index i_{o_k}, and we record it in the j-th row in A, T and S. Then, $A_j = a$, $T_j = i_r$, and $S_j = [i_{o_1}, ..., i_{o_k}]$. For example, if R_{C8} is the j-th row, $A_j = 3$, $T_j = 8$, and $S_j = [1, 0, 3]$.

By fancy indexing like $H[A, S]$, we can get the hidden vectors of operands of all relations in this layer, $H^S \in R^{N\times k\times d}$. Equation 1 can be applied to the input of one sample, which is one row of H^S. By similar batch computation method of ordinary LSTM implementation, Equation 1 can be applied to H^S as a batch. Finally, the result tenors are set into $H[A, T]$.

Notice that each layer will require its own A, T, S matrices, since only relations in the same layer can be computed in parallel. And this parallelization method will become more efficient as the batch size and the number of relations or candidates increase.

4 EXPERIMENT

4.1 Semantic Causality Relation Extraction

Task Description. Semantic causality relation is the relation that describes cause and effect between clauses or relations. For example, Figure 3 (a) shows a sentence with three clauses C1, C2, and C3. C2 is the cause of C3, which forms a relation R1. And R1 (or the combination of C2 and C3) is the cause of C1. The task schema is:

$$\begin{aligned} E =& \{\texttt{Clause}\}, \\ L =& \{\texttt{Causality}\}, \\ C^{\texttt{Causality}} =& \{\texttt{Clause,Causality}\}\times \\ & \{\texttt{Clause,Causality}\} \end{aligned}$$

Statistics. Ten thousand of sentences are selected and manually annotated from bond prospectus in Chinese. The statistics about the distribution of the number of relations and clauses in a sentence are shown in Figure 4. The percentage of sentences that have 0, 1, 2, and ≥ 3 layers of relations are 4.25% , 81.13%, 13.72%, and 0.90% respectively. So, 14.62% of sentences have nested relations.

Baseline. At first glance, this problem seems to be simple due to the existence of keywords like "reason", "leading to", and "because". So we attempt to solve this problem by rule. By digging into the annotated data, 293 distinct keywords are summarized into 4 categories. Then, each sentence is converted into a pattern sequence consists of clauses and keywords, and 1344 patterns are found. We summarize over 130 most frequent patterns based on these patterns and their relation results, and come up with an expression parsing-like algorithm. This algorithm can cover 518 patterns, more than 88% of sentences.

Settings. The hyper-parameter settings of our model is as follows. The vocabulary size is 6000 because the vocabulary in financial dataset is relatively concentrated. The embedding size and hidden vector size of Bi-LSTM are set to 256. We add position encodings [27] to word embeddings. Since Bi-LSTM will concatenate hidden vectors from forward and backward, we set hidden size of DAG-LSTM to 512. The final fully connected layer is $512 \times 1024 \times 2$. We use Adadelta [31] as the optimizer with learning rate = 1.0. The batch size is 8.

Result Analysis. The evaluation metric is precision, recall and F1. They are defined as

$$\begin{aligned} \text{Precision} =& \frac{|\hat{R} \cap R|}{|\hat{R}|} \\ \text{Recall} =& \frac{|\hat{R} \cap R|}{|R|} \\ \text{F1} =& \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned}$$

where $\hat{R}$ is the set of predicted relations, and R is the set of annotated relations. The comparison of relation instances between $\hat{R}$ and R will compare both the operand tuple and the label. The results on test set are shown in Table 2. Besides representing the overall result, we also report the result by layer. And the number of relations in each layer are shown in table. Since the relations in layer 3 or higher are rare as discussed above, we report the result of relations in layer 1, and relations in layers ≥ 2.

We first compare the results between rule based algorithm and our model. "**From scratch**" means that the model or the rule predicts relations layer by layer, and the error in one layer will propagate to its subsequent layers. This is the situation when the model

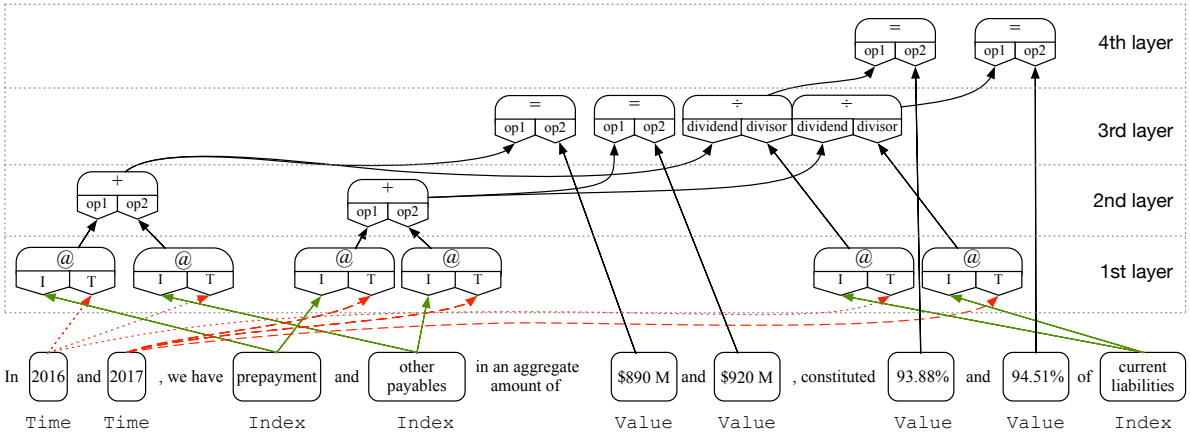

Figure 5: Examples for formula extraction.

Table 2: Results on semantic causality extraction.

	Layer:	All	1	≥ 2
Rule (From scratch)	Precision	70.80	73.07	48.11
	Recall	17.48	19.53	6.74
	F1	28.04	30.82	11.82
Our Model (From scratch)	Precision	78.51	82.27	63.12
	Recall	79.41	80.62	73.48
	F1	78.96	81.43	67.91
Our Model (Guided)	Precision	81.63	82.27	78.75
	Recall	80.99	80.62	82.81
	F1	81.31	81.43	80.73

is applied in real world. The first part of table shows that the performance of rule based algorithm is poor on this dataset. The precision of rule algorithm is lower than our model (about 10% lower in layer 1), and it degrades dramatically in layers ≥ 2. Although we have put efforts on rule designing and many rules are collected, the recall is still very low (< 18%). Designing more rules may increase its recall. But since there is a balance between precision and recall, more rules may hurt the precision which is already lower than model, and may bring conflict between rules. The second part shows the result of our model. The overall F1 score is 81.08, which is much higher than rule algorithm. Layer evaluation shows a drop of performance on the second layer but significantly better than hand-crafted rules. We think there are two reasons for this drop: 1) extraction the high layer relation is intrinsically harder, and 2) according to the statistics introduced above, the number of high layer relations is much smaller than the first layer. The comparison between rule and our model indicates that the semantic causality extraction is a non-trivial problem, despite the existence of strong causal conjunction features like "since" and "because".

In the last part of Table 2, "**Guided**" means that the prediction results of current layer are replaced by correct results when predicting the following layers. This is the situation when we are training the model. We report this result to show the phenomenon of error propagation. The overall performance of "Guided" is better than "From scratch" for our model. The performances at layer one are the same since their entities are the same. The performance of "Guided" is 12.82% better than "From scratch" in layers ≥ 2. That means the error in the first layer will propagate through layers and hurt the performance in following layers. This brings in two types of errors. If a relation is not recalled in the first layer, then relations in subsequent layers which take it as operand can not be extracted. This will affect the recall of subsequent layers. If a relation is falsely extracted, the process might generate strange candidates based on it which are never seen during training. This will affect the precision of subsequent layers. 21.04% of relations extracted from the first layer are wrong (precision=78.96%). We think this might be the third reason for the performance degradation in high layers following the intrinsic hardness and lack of data.

4.2 Formula Extraction

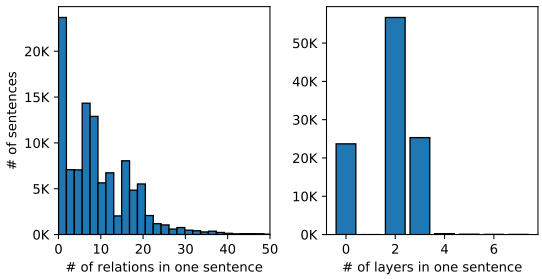

Figure 6: Distributions of formula extraction dataset.

Task Description. Verbal descriptions over the numerical relationships among some objective measures widely exist in financial documents. We collect a large manually annotated dataset from bond prospectus.

Figure 3 shows an example from our dataset. Three types of entities are extracted in advance. A time entity describes a certain time like year 2016. An index entity describes a financial index,

Table 3: Results on formula extraction.

Layer:		All	1	2	3	4	≥ 5
#relations		238,290	98,163	103,110	34,517	1,308	1,192
From scratch	P	96.05	97.93	95.83	93.05	70.92	69.25
	R	95.85	97.42	96.07	93.29	71.87	54.60
	F1	95.95	97.67	95.95	93.17	71.39	61.06
Guided	P	97.97	97.93	98.07	98.12	92.58	93.79
	R	97.99	97.42	98.45	98.53	93.43	93.79
	F1	97.98	97.67	98.26	98.32	93.00	93.79

like "other payables". And a value entity describes a value like $ 80 million or 98%. The first kind of relation is the relation between time and index. They are shown as relations with "@" mark. This kind of relations refers to financial indexes at certain time, which should be equal to some numbers. Then, the higher layers of relations are summations, divisions and equations. The task schema is:

$$
\begin{aligned}
E &= \{\texttt{Time, Index, Value}\} \\
L &= \{\texttt{@, +, -, ÷, =, <, CGR, Prop}\} \\
C^{@} &= \{\texttt{Time}\} \times \{\texttt{Index}\} \\
C^{+} &= \{\texttt{@, +, -, ÷, Value}\} \times \\
&\quad \{\texttt{@, +, -, ÷, Value}\} \\
C^{=} &= C^{<} = C^{-} = C^{\div} = C^{\texttt{CGR}} = C^{+} \\
C^{\texttt{Prop}} &= \{@, +, -, \div\}
\end{aligned}
$$

The CGR (compound growth rate) relation is the relation described in sentence like "the compound growth rate of income between 2005 to 2018". The Prop relation is a division where the divisor is omitted in sentence, so it is a relation with only one operand. And we convert all ">" relations into "<" relations by reversing their operands since we think they are syntactically similar in natural language.

Statistics. More than a hundred thousand of sentences are annotated. The statistics about the distribution of the number of relations and layers in a sentence are shown in Figure 6. Note that there are sparks at number of relation = 0. Since the sentences are not strictly filtered, there are many sentences that contain no relations. Also note that no sentences have one layer of relations. The first layer of relations must be @ relations according to the definition above. Our annotation guide requires to annotate @ relations that are operands (or descendants) of equation or comparison relations. If it is not operand of any other relations, it will not be annotated. So, if there are relations in a sentence, it must have at least two layers. Second layer relations are relations like = relations between @ relations and numbers, or the arithmetic relation between @ relations. Third layer relations are relations like = relations between arithmetic relations and numbers, or more complex arithmetic relations. The distribution shows that the relations are highly nested in this dataset: more than one fourth of sentences have 3 or 4 layers of relations.

Settings. This task is too complex to write rule based algorithm. To our best knowledge, there are no existing methods on extracting such highly nested relations. So we only test on our algorithm.

The hyper-parameter setting is exactly the same with the semantic causality problem.

Result Analysis. We use the same evaluation metric like semantic causality extraction: precision, recall and F1 on typed relations. The results on test set are shown in Table 3. The first intuition is that this result is better than semantic causality. We think this largely attribute to the large amount of data: there are 10 times more data in this task, and deep learning models are data hungry.

The result of Guided on different layers are relatively stable, F1 score drops slightly from 97.98% on layer 1 to 93.79% on layers ≥ 5. That shows our model's capability of extracting nested relations. Notice that, F1 score in layer 1 is lower than layer 2 and 3. The reason is as follows. Highly nested relations need lower layer relations as operands. As mentioned above, our annotation guide asks to annotate relations that are descendants of equation or comparison relations only. Some of lower layer relations are very subtle because the model has to discover the high layer relations first before extracting them. Thus extracting these lower layer relations are not simpler than high layer ones. On the other hand, when the lower relations are extracted, extracting higher layer relations becomes easier. The results on layer 4 and 5 are lower than the first three layers because the number of relations in layer 4 and 5 is one order of magnitude smaller, as shown in "#relation" row.

The result of From scratch is the same as Guided one layer 1 since they have same candidates. The result of From scratch has a sharp drop on layers 4. F1 score of From scratch drops from 93.17% to 71.39%. The gap between Guided and From scratch is over 20% on this layer and layers ≥ 5. Thus, the sharp drop should attribute to the error propagation. Comparing to the result of semantic causality extraction, the error propagate in this task is worse, as the number of layers is larger. There are researches on alleviating error propagation problem from NER to RE by joint training[22] and novel tagging [33]. How to alleviate error propagation in nested RE could be an interesting future work.

Suppose there is a RE method capable of extracting ternary relation with configuration {Time}×{Index}×{Value}. The upper-bound of this method is extracting 63.85% of all relations. If we get each root of nested structure and analyze its structure, we find there are 161 kinds of structures in test data annotation. If we want to use flat relation to represent all of these structures, we need to define 161 types of relations. Moreover, 43.75% of root nodes involve four or more entities. That shows RE methods for "flatten" relation extraction are not suitable for this task.

4.3 Speedup by parallelization

We conduct experiments to test the speedup ratio of the parallelization method we introduce. The model is implemented in Pytorch [24]. The parallelization implementation follows the method in Section3.2. In comparison, the serial implementation iterates through each row of A, T and S to compute one relation at a time, which is the situation when no parallelization is conducted.

The times spent on updating on 3200 sentences for serial and parallelized implementations are shown in Figure 7. Times on semantic causality extraction are shown on the left side, and times on formula extraction are shown on the right side. Serial implementation may have batch size > 1 because other parts of the model

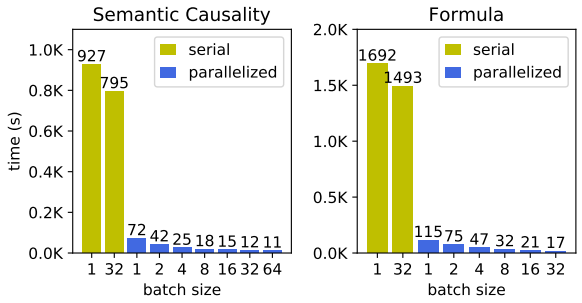

Figure 7: Average time on updating 3200 sentences.

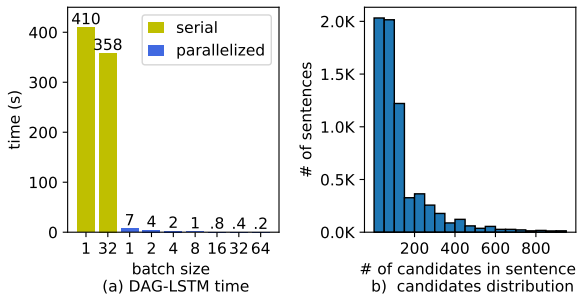

Figure 8: Average time of DAG-LSTM (forward computation) on updating 3200 sentences on semantic causality.

except relation representation module can be computed in batch. The following discussion focuses on semantic causality extraction, but applies to formula extraction too. The first observation is that parallelized implementations are dozens of times faster than serial implementations. For batch size = 1, the parallelized method is 12 times faster; and for batch size = 32, the parallelized method is 72 times faster. The second observation is that the training time drops as the batch size increase for both serial and parallelized implementation. The relative time reduction of parallelized one is more significant: 83.33% time is saved when batch size increased from 1 to 32.

Because the updating time consists of many parts, such as data transmission, token representation module, back-propagation, etc., the overall time reduction might be a result of other components rather than relation module. To get a more precise measurement, we record the exact time used in the forward computation (not include the back-propagation computation) in relation representation module. The result for semantic causality is shown in Figure 8 (a). The huge speedup ratio is more clear. For serial implementation, the relation representation module computation time takes half of total update time. And for parallelized implementation, that proportion is much smaller, which means DAG-LSTM is not the time bottleneck of this model.

The speedup ratio of parallelized implementation is huge because the number of candidates in one sentence is huge. Figure 8(b) shows the distribution of sentences for the number of candidates in them. The average number of candidates is 150.74 per sentence, which means when batch size = 1, the parallelized DAG-LSTM computes 150 relations in parallel, while the serial one computes 1 relation at a time for 150 times. And when batch size = 32, this number goes to 4800. This explains the huge time gap between the serial and parallelized implementations.

5 RELATED WORK

Relation Extraction. Most of RE works focus on extracting relations between two entities [2, 21, 22]. Madaan et al. [19] worked on extracting relations that contain numbers (like the formula extraction problem in this paper) but stop at only one layer. Distant supervision is an important research direction in relation extraction [10, 32]. But in this paper, we focus on supervised relation extraction. Recently, people are extending the scope of relation extraction. Ernst et al. [9] proposed a system to harvest high-arity relationships. In bioinformatics area, the demand for high-arity, nested relation has been lasting for years [34]. McDonald et al. proposed algorithm for high-arity relation extraction [20] by discovering binary relation cliques.

People also noticed the incompleteness and information loss for binary relation extraction during the construction of knowledge graphs and ontologies like YAGO [25]. So they attached temporal, geospatial and other prepositional information to relations. Bhutani et al. [5] studied to extract nested prepositions in open information extraction problem. Based on result of dependency parsing, they designed hand-crafted templates, and applied bootstrapping and pattern learning. They assumed the inter-proposition relations can be inferred from dependency tree, and the proposition nesting is carried out by linking simple propositions with rules. These semi-supervised methods may suffer from low precision problem, but this phenomenon indicates the demand and necessity of complex relation for more precise information acquisition. But no systematic and generic solutions about nested relations were proposed to our best knowledge.

Tree-LSTM has been used in sentiment analysis [36], semantic relatedness classification [26], and traditional binary flatten relation extraction [22]. These works used Tree-LSTM on parsed dependency trees as input to integrate external knowledges and get a better representation of sentence or relation pair. However, in our task, instead of applying Tree-LSTM on a given static structure, we use it to extract the task specific nested structures.

Structure Extraction. Many researches work on extracting machine-understandable structures from natural language utterance. Most of them are categorized into semantic parsing. Semantic parsing is a wide concept [6] which covers works that map a natural language utterance into a formal representation like Lisp [16] or logical expressions [28, 29], database queries [4, 6, 16], and UCCA structures [14]. Most of them utilized Seq2Seq models, or transition-based models, which had to serialize the output structure into a sequence. But that may suffer from the situation that the generated sequences can be invalid [29]. Our proposed solution which directly generates a structure might provide a new approach in this field.

Speedup The widespread usage of complex neural network in natural language processing has put forward an urgent demand for faster training speed. To speed up models with complex structure,

a framework "Fold" [17] is proposed. It treats the neural network computation graph as a tree, and groups computation nodes in the same layer together to compute in parallel. However, learning Fold changes the way of how to program with TensorFlow, as it introduces many new and complicated concepts. It stopped updating since Nov. 2017. The method we propose is lightweight which may be easier to implement and extend.

6 CONCLUSION

In this paper, we extend the concept of relation extraction from flatten relations to nested relations, because the expressions of nested relation are common in natural language. We propose a formal formulation of nested relation extraction problem, which covers the simple binary flatten relation extraction problems, high-arity relation extraction and nested relation extraction. With the development of advanced deep learning models, extracting nested relations becomes possible. We introduce an Iterative Neural Network that consists of text, entity, and relation representation modules. The relation representation module utilize DAG-LSTM to extract complex structured relations. It is able to extract relations layer by layer iteratively, generating candidates on the fly.

Because the number of candidates is huge in each sentence, the naive serial implementation of DAG-LSTM is slow. Therefore, we propose to put relations and candidates in the same layer together, and compute them in parallel. This method is simple yet very effective, speeds up the training process by dozens of times.

Two experiments are conducted on two different nested relation extraction tasks. This model performs well on both tasks. The result shows an error propagation problem for high layer relations. We think this might be an interesting future work.

7 ACKNOWLEDGEMENTS

This work was supported by the National Key Research and Development Program of China under Grant No. 2017YFB1002104, the National Natural Science Foundation of China under Grant No. U1811461, and the Innovation Program of Institute of Computing Technology, CAS.

REFERENCES

[1] Rodrigo Agerri and German Rigau. 2016. Robust multilingual named entity recognition with shallow semi-supervised features. *Artificial Intelligence* (2016).
[2] Charu C Aggarwal and ChengXiang Zhai. 2012. *Mining text data*.
[3] Jacqueline Aguilar, Charley Beller, Paul McNamee, Benjamin Van Durme, Stephanie Strassel, Zhiyi Song, and Joe Ellis. 2014. A comparison of the events and relations across ace, ere, tac-kbp, and framenet annotation standards. In *Proceedings of the Second Workshop on EVENTS: Definition, Detection, Coreference, and Representation*.
[4] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *EMNLP*.
[5] Nikita Bhutani, H V Jagadish, and Dragomir Radev. 2016. Nested Propositions in Open Information Extraction. In *EMNLP*.
[6] Phil Blunsom, Nando de Freitas, Edward Grefenstette, and Karl Moritz Hermann. 2014. A deep architecture for semantic parsing. In *ACL Workshop on Semantic Parsing*.
[7] Yixuan Cao, Hongwei Li, Ping Luo, and Jiaquan Yao. 2018. Towards Automatic Numerical Cross-Checking: Extracting Formulas from Text. In *WWW*.
[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.
[9] Patrick Ernst, Amy Siu, and Gerhard Weikum. 2018. HighLife: Higher-arity Fact Harvesting. In *WWW*.
[10] Jun Feng, Minlie Huang, Li Zhao, Yang Yang, and Xiaoyan Zhu. 2018. Reinforcement Learning for Relation Classification from Noisy Data. In *AAAI*.
[11] Roxana Girju, Preslav Nakov, Vivi Nastase, Stan Szpakowicz, Peter Turney, and Deniz Yuret. 2007. Semeval-2007 task 04: Classification of semantic relations between nominals. In *Proceedings of the 4th International Workshop on Semantic Evaluations*.
[12] Naeemul Hassan, Fatma Arslan, Chengkai Li, and Mark Tremayne. 2017. Toward Automated Fact-Checking: Detecting Check-worthy Factual Claims by ClaimBuster. In *KDD*.
[13] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2009. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*.
[14] Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A Transition-Based Directed Acyclic Graph Parser for UCCA. In *ACL*.
[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* (1997).
[16] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *ACL*.
[17] Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig. 2017. Deep Learning with Dynamic Computation Graphs. In *ICLR*.
[18] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.
[19] Aman Madaan, Ashish Mittal, G Ramakrishnan Mausam, Ganesh Ramakrishnan, and Sunita Sarawagi. 2016. Numerical Relation Extraction with Minimal Supervision.. In *AAAI*.
[20] Ryan McDonald, Fernando Pereira, Seth Kulick, Scott Winters, Yang Jin, and Pete White. 2005. Simple algorithms for complex relation extraction with applications to biomedical IE. In *ACL*.
[21] Mike Mintz, Steven Bills, Rion Snow, and Jurafsky Dan. 2009. Distant supervision for relation extraction without labeled data. In *ACL*.
[22] Makoto Miwa and Mohit Bansal. 2016. End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. In *ACL*.
[23] Michèle B. Nuijten, Chris H. J. Hartgerink, Marcel A. L. M. van Assen, Sacha Epskamp, and Jelte M. Wicherts. 2016. The prevalence of statistical reporting errors in psychology (1985–2013). *Behavior Research Methods* (2016).
[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-Workshop*.
[25] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2008. YAGO: A Large Ontology from Wikipedia and WordNet. In *WWW*.
[26] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *ACL*.
[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.
[28] Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a Semantic Parser Overnight. In *ACL*.
[29] Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based Structured Prediction for Semantic Parsing. In *ACL*.
[30] Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. 2015. Classifying relations via long short term memory networks along shortest dependency paths. In *EMNLP*.
[31] Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *arXiv* (2012).
[32] Daojian Zeng, Kang Liu, Yubo Chen, and Jun Zhao. 2015. Distant Supervision for Relation Extraction via Piecewise Convolutional Neural Networks. In *EMNLP*.
[33] Suncong Zheng, Feng Wang, Hongyun Bao, Yuexing Hao, Peng Zhou, and Bo Xu. 2017. Joint Extraction of Entities and Relations Based on a Novel Tagging Scheme. In *ACL*.
[34] Deyu Zhou, Dayou Zhong, and Yulan He. 2014. Biomedical relation extraction: from binary to complex. *Computational and mathematical methods in medicine* (2014).
[35] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *ACL*.
[36] Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. 2015. Long Short-Term Memory over Recursive Structures. In *ICML*.