Event Extraction via Bidirectional Long Short-Term Memory Tensor Neural Networks

Yubo Chen(✉), Shulin Liu, Shizhu He, Kang Liu, and Jun Zhao

National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
{yubo.chen,shulin.liu,shizhu.he,kliu,jzhao}@nlpr.ia.ac.cn

Abstract. Traditional approaches to the task of ACE event extraction usually rely on complicated natural language processing (NLP) tools and elaborately designed features. Which suffer from error propagation of the existing tools and take a large amount of human effort. And nearly all of approaches extract each argument of an event separately without considering the interaction between candidate arguments. By contrast, we propose a novel event-extraction method, which aims to automatically extract valuable clues without using complicated NLP tools and predict all arguments of an event simultaneously. In our model, we exploit a context-aware word representation model based on Long Short-Term Memory Networks (LSTM) to capture the semantics of words from plain texts. In addition, we propose a tensor layer to explore the interaction between candidate arguments and predict all arguments simultaneously. The experimental results show that our approach significantly outperforms other state-of-the-art methods.

1 Introduction

Event extraction is an important and challenging task in Information Extraction (IE), which aims to discover event triggers with specific types and their arguments. The task is quite difficult because a larger field of view is often needed to understand how facts tie together. To capture valuable clues for event extraction, current state-of-the-art methods [11,13,16–18] often use a set of elaborately designed features that are extracted by textual analysis. For example, consider the following sentences:

S1: *He has* ***fired*** *his air defense chief.*
S2: *An American tank* ***fired*** *on the Hotel.*

In S1, *fired* is a trigger of type *End-Position*. While in S2 *fired* is a trigger of type *Attack*, which is more common than type *End-Position*. Because of the ambiguity, a traditional approach may mislabel *fired* in S1 as a trigger of *Attack*. However, if we know that the context words "*air defense chief*" is a job title, we have ample evidence to predict that *fired* in S1 is a trigger of type *End-position*. To capture these semantics, traditional methods often use the part-of-speech

M. Sun et al. (Eds.): CCL and NLP-NABD 2016, LNAI 10035, pp. 190–203, 2016.
DOI: 10.1007/978-3-319-47674-2_17

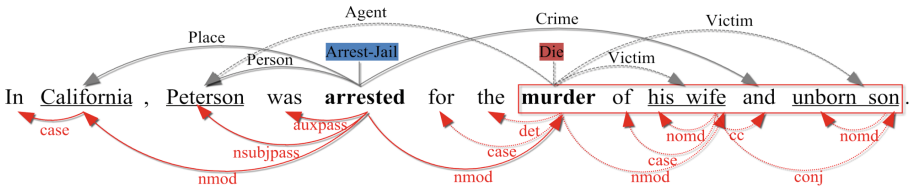

Fig. 1. Event mentions and syntactic parser results of S3. The upper side shows two event mentions that share one arguments: the *Arrest-Jail* event mention, triggered by "arrested", and the *Die* event mention, triggered by "murder". The lower side shows the dependency results.

tags (POS), entity information, and morphology features (e.g., token, lemma, etc.). However, these features for words are a kind of one-hot representation, which may suffer from the data sparsity problem and overlook the impact of the context words. Furthermore, these methods often rely on human ingenuity for designing such elaborate features, which is a time-consuming process and lacks generalization. [22].

Besides the semantics of words, to extract events more precisely, we also need to understand semantic relations between words. Previous methods often use the syntactic features to capture these semantics. For example, in S3, there are two events that share one argument as shown in Fig. 1, from the dependency relation of *nsubjpass* between the argument *Peterson* and trigger *arrested*, we can induce a *Person* role to *Peterson* in the *Arrest-Jail* event. However, for the *Die* event, the argument word *Peterson* and its trigger word *murder* are in different clauses, and there is no direct dependency path between them. Thus it is difficult to find the *Agent* role between them using traditional dependency features. In addition, extracting such features depends heavily on the performance of NLP systems, which could suffer from error propagation.

S3: *In California, Peterson was arrested for the murder of his wife and unborn son.*

To correctly attach *Peterson* to *murder* as a *Agent* argument, we not merely need to understand the semantics of each word but also need to exploit internal semantic relation over the entire sentence such that the *Die* event results in *Arrest-Jail* event. Recent improvements of Recurrent Neural Networks (RNNs) have been proven to be efficient for learning semantics of words, which take the impact of the context into consideration [12,14]. Unfortunately, the recurrent structure of RNNs also make it endure the "vanishing gradients" problem, which makes it difficult for the RNN model to learn long-distance correlations in a sequence [8,9]. So we use a RNN with Bidirectional Long Short-Term Memory (BLSTM) unit [6,10] to addresses this problem.

Most of LSTM-based RNNs are for sequence modeling task, which could use the representation of RNNs for each word directly. While in the task of event extraction, to capture the most useful information within a sentence,

a representation of the entire sentence is needed, which can be acquired by applying a max-pooling layer over the RNNs [15]. However, in event extraction, one sentence may contain two or more events, and these events may share the argument with different roles, as shown in Fig. 1. We apply a dynamic multi-pooling layer in our LSTM-based framework, which can capture the valuable semantics of a whole sentence automatically and reserve information more comprehensively to extract events [4].

Besides the problem stated above, both of traditional approaches [1,13,18] and deep neural network based approaches [4,20] did not model the interaction between the candidate arguments and predict them separately. However, these interactions are important to predict arguments. For example, in Fig. 1, if we know "his wife" and "unborn son" are paralleled in the sentence, we are easy to predict they play same role in the corresponding event. Thus we propose a tensor layer to explore the interaction between candidate arguments automatically.

In this paper, we present a novel framework dubbed Bidirectional Dynamic Multi-Pooling Long Short-Term Memory Tensor Neural Networks (BDLSTM-TNNs) for event extraction, which can automatically induce valuable clues for event extraction without complicated NLP preprocessing and predict candidate arguments simultaneously. We propose a Bidirectional Long Short-Term Memory Network with Dynamic Multi-Pooling (BDLSTM) to extract event triggers and arguments separately, which can capture meaningful semantics of words with taking the context words into consideration and capture more valuable information for event extraction within a sentence automatically. And we devise a tensor layer, which aims to explore interaction between candidate arguments and predict them jointly. We conduct experiments on a widely used ACE2005 event extraction dataset, and the experimental results show that our approach outperforms other state-of-the-art methods.

2 Event Extraction Task

In this paper, we focus on the event extraction task defined in Automatic Content Extraction[1] (ACE) evaluation, where an event is defined as a specific occurrence involving participants. First, we introduce some ACE terminology to understand this task more easily:

- **Event mention:** a phrase or sentence within which an event is described, including a trigger and arguments.
- **Event trigger:** the word that most clearly expresses the occurrence of an event.
- **Event argument**: an entity mention, temporal expression or value (e.g. JobTitle) that is involved in an event (viz., participants).
- **Argument role:** the relationship between an argument to the event in which it participates.

[1] http://projects.ldc.upenn.edu/ace/.

Given a text document, an event extraction system should predict event triggers with specific subtypes and their arguments. The upper side of Fig. 1 depicts the event triggers and their arguments for S3 in Sect. 1. ACE defines 8 event types and 33 subtypes, such as *Arrest-Jail* or *Die*.

3 Methodology

In this paper, event extraction is formulated as a two-stage, multi-class classification via Bidirectional Dynamic Multi-pooling Long Short-Term Memory Networks (BDLSTM) with the automatically learned valuable clues. The first stage is called *trigger classification*, in which we use a BDLSTM to classify each word in a sentence to identify trigger words. If one sentence has triggers, the second stage is conducted, which applies a similar BDLSTM to assign arguments to triggers and align the roles of the arguments. We call this *argument classification*. To explore interaction between candidate arguments and predict all candidate arguments simultaneously, we design a tensor layer in the stage of argument classification. Because the argument classification is more complicated, we first describe the methodology of how predict each candidate argument separately by BDLSTM in Sects. 3.1 and 3.4 and then illustrate the difference between the BDLSTMs that are used for trigger classification and those used for argument classification in Sect. 3.5. Finally, we introduce the Bidirectional Dynamic Multi-Pooling Long Short-Term Memory Tensor Neural Networks (BDLSTM-TNN) and illustrate how it predict all candidate arguments jointly in Sects. 3.6 and 3.7.

Figure 2 describes the BDLSTM architecture of argument classification, which primarily involves the following three components: (i) context-aware word representation; (ii) dynamic multi-pooling layer; (iii) argument classifier output layer.

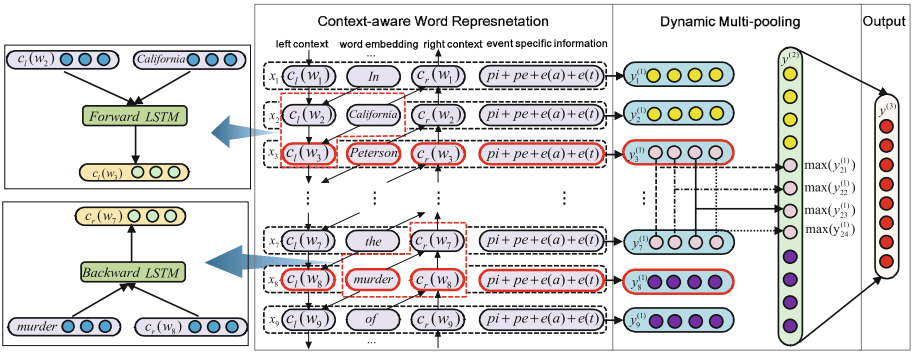

Fig. 2. The architecture for argument classification in the event extraction. It illustrates the processing of one instance with the predict trigger *murder* and the candidate argument *Peterson*.

3.1 Context-Aware Word Representation Using LSTM Networks

The semantics of words and relations between words serve as important clues for event extraction [11,17]. RNNs with LSTM unit is a good choice to learn the context-aware semantics [12,15]. In event extraction, both forward and backward words are important for understanding a word. Thus, we propose to use a Bidirectional Long Short-Term Memory (BLSTM) architecture for representing words.

Context-Aware Word Representation. This subsection illustrates the components of the context-aware word representation. As shown in Fig. 2, it primarily involves the following four parts: (i) Word embedding, which aims to capture the background of a word; (ii) left context and right context, which help to disambiguate the semantics of the current word and reveal the relations between the words; (iii) position information, which aims to specify which words are the predicted trigger or candidate argument; (iv) predicted event type, which aims to embed the predicted event type from the stage of trigger classification into the argument classification.

(i) word embedding: We use vectors ($e(w_i)$, $e(t)$ and $e(a)$) to represent a word w_i, the predicted trigger word t and the candidate argument a, which are transformed by looking up word embeddings. The word embeddings are trained by a Skip-gram model [2] from a significant amount of unlabeled data.

(ii) Left context and right context: We take all the words on the left side of the word w_i as the left-side context $c_l(w_i)$, while all the words on the right side as the right-side context $c_r(w_i)$. As shown in Fig. 2, the context of each word is different. The left-side context $c_l(w_i)$ of word w_i is calculated using Eq. (1), where $e(w_{i-1})$ is the word embedding of word w_{i-1}, $c_l(w_{i-1})$ is the left-side context of the previous word w_{i-1}. Function f is the operation of LSTM, which we will illustrate in the next section. The right-side context $c_r(w_i)$ is calculated in a similar manner, as shown in Eq. (2).

$$c_l(w_i) = f(c_l(w_{i-1}), e(w_{i-1})) \tag{1}$$

$$c_r(w_i) = f(c_r(w_{i+1}), e(w_{i+1})) \tag{2}$$

(iii) Position information: Position information (pi) is defined as the relative distance of the current word to the predicted trigger or candidate argument as [4] did.

(iv) Predicted event type: We encode the predicted event type (pe) of the trigger as in the position information.

Finally, we define the representation of word w_i in Eq. 3, where $\oplus$ is a concatenation operator.

$$x_i = c_l(w_i) \oplus e_{w_i} \oplus c_r(w_i) \oplus pi \oplus pe \oplus e(a) \oplus e(t) \tag{3}$$

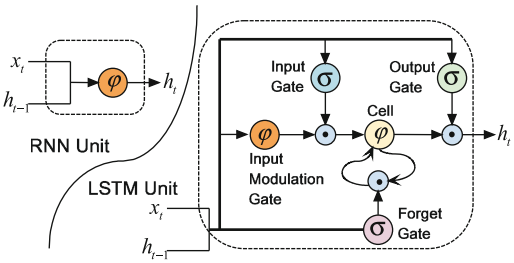

Fig. 3. A basic RNN unit and an LSTM unit

In this manner, using this contextual information and event specific informations, our model may be better to disambiguate the meaning of the word w_i and the semantic relations to others words, especially the interactions to the predicted trigger and the candidate argument. We apply a linear transformation together with the tanh activation function to x_i and send the result to the next layer.

$$y_i^{(1)} = \tanh(W_1 x_i + b_1) \tag{4}$$

Where i ranges from 1 to n and n is the length of sentence. The next layer $y^{(1)}$ is a matrix $y^{(1)} \in \mathbb{R}^{n \times m}$ and m is the dimension of $y_i^{(1)}$.

Long Short-Term Memory Networks. Traditional RNNs (Fig. 3) are able to process input sequences of arbitrary length via the recurrent application of a transition function on a hidden state vector h_t. Commonly, the RNN unit calculates the hidden states via the recurrence Eq. 5. where φ is an element-wise non-linearity, such as a sigmoid or hyperbolic tangent, x_t is the input, and h_t is the hidden state at time t.

$$h_t = \varphi(W_h x_t + U_h h_{t-1} + b_h) \tag{5}$$

Unfortunately, a problem with traditional RNNs is that during training, it is difficult to learn long-distance correlations in a sequence, because components of gradient vector can decay exponentially over long sequences [3,8]. The LSTM architecture [10] provide a solution by incorporating a memory unit that allows the network to learn when to forget previous hidden states and when to update hidden states given new information. In this paper, we use the LSTM unit as described in [23]. As shown in Fig. 3, the LSTM unit at each time step t is a collection of vectors: an input gate i_t, an input modulation gate u_t, a forget gate f_t, an output gate o_t, a memory cell c_t and a hidden state h_t. Where x_t is the input at the current time step, see [23] for details. As shown in Fig. 2, we use a BLSTM, which consists of two LSTMs that are run in parallel. One on the input sequence and the other on the reverse of the input sequence. At each time step, the forward hidden state aligns to $c_l(w_t)$, and the backward hidden state aligns to $c_r(w_t)$. This setup allows the model to capture both past and future information.

3.2 Dynamic Multi-pooling

The size of the layer $y^{(1)} \in \mathbb{R}^{n \times m}$ depends on the number of tokens in the input sentence. In order to apply subsequent layers, traditional RNNs [15] apply a max-pooling operation, which take each dimension of $y^{(1)}$ as a pool and get one max value for each dimension. However, single max-pooling is not sufficient for event extraction. Because one sentence may contain two or more events, and one argument candidate may play a different role with a different trigger [4]. To solve this problem, [4] devise a dynamic multi-pooling layer for Convolutional Neural Networks (CNNs). We apply a similar layer to our BLSTM. We split each dimension of $y^{(1)}$ into three parts according to the candidate argument and predicted trigger in the argument classification stage.

As shown in Fig. 2, the j-th dimension of the $y^{(1)}$ is divided into three sections $y_{1j}^{(1)}, y_{2j}^{(1)}, y_{3j}^{(1)}$ by "Peterson"and "murder". The dynamic multi-pooling can be expressed as Eq. 6, where $1 \leq i \leq 3$ and $1 \leq j \leq m$.

$$y_{ij}^{(2)} = \max(y_{ij}^{(1)}) \tag{6}$$

Through the dynamic multi-pooling layer, we obtain the $y_{ij}^{(2)}$ for each dimension of $y^{(1)}$. Then, we concatenate all $y_{ij}^{(2)}$ to form a vector $y^{(2)} \in \mathbb{R}^{3m}$, which can be considered as valuable clues and contains the key semantics of the whole sentence to classify argument precisely.

3.3 Output

To compute the confidence of each argument role, the vector $y^{(2)}$ is fed into a classifier.

$$O = W_2 y^{(2)} + b_2 \tag{7}$$

where, $W_2 \in \mathbb{R}^{n_1 \times (3m)}$ is the transformation matrix and $O \in \mathbb{R}^{n_1}$ is the final output of the network, where n_1 is equal to the number of the argument role including the "None role" label for the candidate argument which don't play any role in the event. For regularization, we also employ dropout [7] on the penultimate layer.

3.4 Training

We define all of the parameters for the stage of argument classification to be trained as $\theta = (E, c_l(w_1), c_r(w_n), pi, pe, W_1, b_1, W_2, b_2, lf, lb)$. Specifically, the parameters are word embeddings E, the initial contexts $c_l(w_1)$ and $c_r(w_n)$, position embeddings pi, predicted event type embeddings pe, transformation matrices parameters W_1, b_1, W_2, b_2, and the parameters of forward-LSTM lf and backward-LSTM lb.

Given an input example s, the network with parameter θ outputs the vector O, where the i-th component O_i contains the score for argument role i.

To obtain the conditional probability $p(i|x,\theta)$, we apply a softmax operation over all argument role types:

$$p(i|x,\theta) = \frac{e^{o_i}}{\sum_{k=1}^{n_1} e^{o_k}} \tag{8}$$

Given all of our (suppose T) training examples (x_i, y_i), we can then define the objective function as follows:

$$J(\theta) = \sum_{i=1}^{T} \log p(y_i|x_i,\theta) \tag{9}$$

To compute the network parameter θ, we maximize the log likelihood $J(\theta)$ through stochastic gradient descent over shuffled mini-batches with the Adadelta [24] rule.

3.5 Model for Trigger Classification

The method proposed above is also suitable for trigger classification, but trigger classification only need to find triggers in the sentence, which is less complicated than argument classification. Thus we can used a simplified version of BDLSTM. In context-aware word representation of trigger classification, we do not use the position of the candidate argument. Furthermore, instead of splitting the sentence into three parts, the sentence is split into two parts by a candidate trigger. Except for the above change, we classify a trigger as the classification of an argument.

3.6 BDLSTM-TNNs

Though BDLSTM can extract events from plain texts with automatically generate features, it cannot explore the interaction between candidate arguments, which is important to help assign a right role to a candidate argument. Thus we propose the Bidirectional Dynamic Multi-Pooling Long Short-Term Memory Tensor Neural Networks (BDLSTM-TNNs) to solve this problem in argument classification and we use a same BDLSTM in trigger classification as illustrate above. To model the interaction between candidate arguments, we devise a tensor layer, which approved to be effective for capturing multiple interactions among words [21]. As shown in Fig. 4, we predict all candidate argument simultaneously. For each candidate argument, we use $y^{(2)} \in \mathbb{R}^{n_f}$ get from BDLSTM as input. n_f is the dimension of $y^{(2)}$. If there are n_c candidates, the input feature is $A \in \mathbb{R}^{n_f \times n_c}$. We use a 3-way tensor $T^{[1:n_t]} \in \mathbb{R}^{n_t \times n_f \times n_f}$ in tensor layer, where n_t is the length of the interaction vector. The internal relation I between candidate arguments is calculated as follows:

$$I = A^T T^{[1:n_t]} A; I_i = A^T T^{[i]} A \tag{10}$$

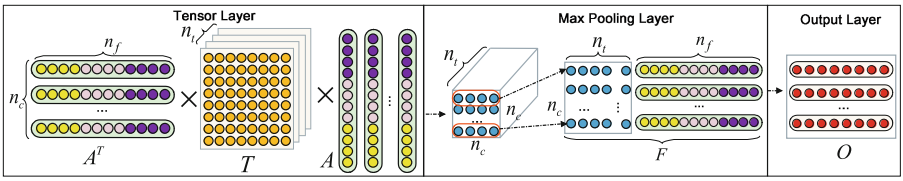

Fig. 4. The architecture (better viewed in color) for BDLSTM-TNNs. It illustrates the processing of predicting n_c candidate arguments simultaneously by using a 3-way tensor T. (Color figure online)

where, $I \in \mathbb{R}^{n_t \times n_c \times n_c}$ is the result of tensor layer, which include the interaction between candidate arguments. To make the model independent of the number of candidate arguments, we apply a max-pooling layer to capture the most valuable internal relation as follows:

$$I_{max}(i,j) = \max_{k=1}^{n_c} I_j(i,k) \tag{11}$$

where, $1 \leq i \leq n_c$, $1 \leq j \leq n_t$ and $I_{max} \in \mathbb{R}^{n_c \times n_t}$. Then we concatenate candidate arguments A and the interaction I_{max} as a whole feature $F \in \mathbb{R}^{n_c \times (n_t + n_f)}$. Each candidate argument has a corresponding feature F_i. Finally the output O_i is computed as follows:

$$O_i = W_3 F_i + b_3 \tag{12}$$

where $O_i \in \mathbb{R}^{n_1}$ is the output of the i-th candidate argument and $O_i(j)$ is the score for argument i to be assign as role j.

3.7 BDLSTM-TNNs Training

We define all of the parameters for the stage of argument classification to be trained as $\theta = (E, c_l(w_1), c_r(w_n), pi, pe, W_1, b_1, W_3, b_3, lf, lb, T)$. Specifically, the parameters are 3-way tensor T, transformation matrixes parameters W_3, b_3, and corresponding parameters of BDLSTM. Given an input instance x_i of sentence s_i with the trigger word tw_i and event type et_i, we use y_i to denote the correct role sequence for x_i, $\hat{y}_i$ to denote the predicted role sequence and $Y_{(x_i)}$ to denote the set of all possible role sequences for x_i. Thus as Eq. 12 shown, $O_{\hat{y}_{ij}}(k)$ is the score of j-th candidate argument to be assigned with role k for x_i, and we compute a sentence level score as follows:

$$s(x_i, \hat{y}_i, \theta) = \sum_{j=1}^{n_c} O_{\hat{y}_{ij}}(k) \tag{13}$$

Where k is the predicted role for j-th candidate argument in x_i. We define a margin loss $\Delta(y_i, \hat{y}_i)$ as follows:

$$\Delta(y_i, \hat{y}_i) = \sum_{j=1}^{n_c} \alpha 1\{y_{ij} \neq \hat{y}_{ij}\} \tag{14}$$

Where α is a discount parameter. Given all of our (suppose T) training examples (x_i, y_i), we can then define the objective function as follows:

$$J(\theta) = \frac{1}{T}\sum_{i=1}^{T} l_i(\theta) + \frac{\lambda}{2}\|\theta\|^2 \tag{15}$$

$$l_i(\theta) = \max_{\hat{y}\in Y_{x_i}} (s(x_i, \hat{y}_i, \theta) + \Delta(y_i, \hat{y}_i) - s(x_i, y_i, \theta)) \tag{16}$$

To increase the score of the correct role sequence y_i and decrease the highest score of incorrect sequence $\hat{y}$, we minimizing the object sated above.

4 Experiments

4.1 Dataset and Evaluation Metric

We utilized the ACE 2005 corpus as our dataset. For comparison, as the same as [4,11,17,18], we used the same test set with 40 newswire articles and the same development set with 30 other documents randomly selected from different genres and the rest 529 documents are used for training. Similar to previous work [4,11,17,18], we use the following criteria to judge the correctness of each predicted event mention:

- A trigger is correct if its event subtype and offsets match those of a reference trigger.
- An argument is correctly identified if its event subtype and offsets match those of any of the reference argument mentions.
- An argument is correctly classified if its event subtype, offsets and argument role match any of the reference argument mentions.

Finally we use *Precision (P)*, *Recall (R)* and *F measure (F)* as the evaluation metrics.

4.2 Our Method Vs. State-of-the-Art Methods

We select the following state-of-the-art methods for comparison.
(1) **Hong's baseline** is the system proposed by [11], which only employs basic human-designed features; (2) **Liao's cross-event** is the method proposed by [18], which uses document-level information to improve the performance of ACE event extraction; (3) **Hong's cross-entity** is the method proposed by [11], which extracts event by using cross-entity inference. To the best of our knowledge, it is the best-reported feature-based system; (4) **Li's structure** is the method proposed by [17], which extracts events based on structure prediction. It is the best-reported structure-based system; (5) **Chen's DMCNN** is the method proposed by [4], which extracts events based on convolutional neural networks. It is the best-reported neural-based system.

Following [4,17], we tuned the model parameters on the development through grid search. Specifically, in the trigger classification, we set the batch size as 150,

Table 1. Overall performance on blind test data

Methods	Trigger identification (%)			Trigger identification + classification (%)			Argument identification (%)			Argument role (%)		
	P	R	F	P	R	F	P	R	F	P	R	F
Hong's baseline		N/A		67.6	53.5	59.7	46.5	37.1	41.3	41.0	32.8	36.5
Liao's cross-event		N/A		68.7	68.9	68.8	50.9	49.7	50.3	45.1	44.1	44.6
Hong's cross-entity		N/A		72.9	64.3	68.3	53.4	52.9	53.1	51.6	45.5	48.3
Li's structure	76.9	65.0	70.4	73.7	62.3	67.5	69.8	47.9	56.8	64.7	44.4	52.7
Chen's DMCNN	80.4	67.7	**73.5**	75.6	63.6	**69.1**	68.8	51.9	59.1	62.2	46.9	53.5
BDLSTM-TNNs	78.9	66.5	72.2	75.3	63.4	68.9	69.8	52.7	**60.0**	62.9	47.5	**54.1**

n_f as 100, and the dimension of the pi as 5. In the argument classification, we set the batch size as 50, n_f as 150, n_t as 180 and the dimension of the pi and pe as 5. We train the word embedding using the Skip-gram algorithm[2] on the NYT corpus[3].

Table 1 shows the overall performance on the blind test dataset. From the results, we can see that the BDLSTM-TNNs model achieves the best performance among all of the compared methods in the stage of argument classification and get the best performance among all of methods expect for DMCNN method. BDLSTM-TNNs can improve the best F_1 [4] in the state-of-the-arts for argument classification by 0.6 % and competitive result for trigger classification. Moreover, we get larger gain compared with traditional methods [11,17,18]. This demonstrates the effectiveness of the proposed method. We believe the reason is that the clues we automatically learned can capture more meaningful semantic regularities of words.

4.3 RNN vs. LSTM vs. BLSTM vs. DLSTM vs. BDLSTM

This subsection studies the effectiveness of our context-aware word representation models. As shown in Table 2. BLSTM is described in Sect. 3.1 which use both forward LSTM and Backward LSTM with max pooling layer. LSTM only

Table 2. Comparison of RNN and different LSTMs

Model	Trigger	Argument
	F_1	F_1
RNN	61.5	41.5
LSTM	63.8	43.8
BLSTM	65.3	46.7
DLSTM	66.5	47.8
BDLSTM	**68.9**	**50.3**

[2] https://code.google.com/p/word2vec/.
[3] https://catalog.ldc.upenn.edu/LDC2008T19.

use one forward LSTM. RNN use a traditional RNN instead of LSTM and BDLSTM apply a dynamic multi-pooling instead max pooling in BLSTM. As shown in results, the methods based on LSTM (LSTM, BLSTM, DLSTM and BDLSTM) makes significant improvements compared with the RNN baseline in the classification of both the trigger and argument. It demonstrated that LSTM is more powerful than RNN for event extraction. Moreover, a comparison of BLSTM with LSTM illustrates that BLSTM achieves a better performance. We can make a same observation when comparing BDLSTM with BLSTM. It proves that both forward and backward of words are important to understand the semantic of a word, and the dynamic multi-pooling layer is effective for capture more valuable information to extract an event when using recurrent neural networks.

4.4 Effect of Tensor Layer

In this subsection, we prove the effectiveness of the tensor layer for exploring the interaction between candidate arguments. Because we used the same BDLSTM in trigger classification, we only evaluate its performance for argument classification. Results are shown in Table 3. It proves that tensor layer is useful to capture the internal relation between candidate arguments. Specifically, BDLSTM-TNNs gain a 5.5 % improvement for argument identification and 3.8 % improvement for argument classification.

Table 3. Comparison of BDLSTM and BDLSTM-TNNs for argument classification

Model	Identification	Classification
	F_1	F_1
BDLSTM	54.5	50.3
BDLSTM-TNNs	**60.0**	**54.1**

5 Related Work

Event extraction is one of important topics in NLP. Many approaches have been explored for event extraction. Nearly all of the ACE event extraction use supervised paradigm. We further divide supervised approaches into feature-based methods, structure-based methods and neural-based methods.

In feature-based methods, a diverse set of strategies has been exploited to convert classification clues into feature vectors. [1] uses a set of traditional features (e.g., full word, pos tag, dependency features) to extract the event. [13] combined global evidence from related documents with local decisions for the event extraction. To capture more clues from the texts, [5,11,18] proposed the cross-event and cross-entity inference for the ACE event task. Although these

approaches achieve high performance, feature-based methods suffer from the problem of selecting a suitable feature set when converting the classification clues into feature vectors.

In structure-based methods, researchers treat event extraction as the task of predicting the structure of the event in a sentence. [19] casted the problem of biomedical event extraction as a dependency parsing problem. [17] presented a joint framework for ACE event extraction based on structured perceptron. These methods yield relatively high performance. However, the performance of these methods depend strongly on the quality of the designed features and endure the errors in the existing NLP tools.

In neural-based method, researchers try to extract events from plain text by using automatically generating features without using complicated NLP tools [4,20]. But they did not consider interaction between candidate arguments and they predicted all candidate arguments separately.

6 Conclusion

This paper proposes a novel event extraction method (BDLSTM-TNNs), which can automatically extract valuable clues from plain texts without complicated NLP preprocessing and predict candidate arguments simultaneously. A context-aware word representation model based on BDLSTM is introduced to capture semantics of words. In addition, a tensor layer is devised to capture interaction between candidate arguments and predict them jointly. The experimental results prove the effectiveness of the proposed method.

Acknowledgement. This work was supported by the Natural Science Foundation of China (No. 61533018), the National Basic Research Program of China (No. 2014CB340503) and the National Natural Science Foundation of China (No. 61272332). And this work was also supported by Google through focused research awards program.

References

1. Ahn, D.: The stages of event extraction. In: Proceedings of ACL, pp. 1–8 (2006)
2. Baroni, M., Dinu, G., Kruszewski, G.: Dont count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In: Proceedings of ACL, pp. 238–247 (2014)
3. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
4. Chen, Y., Xu, L., Liu, K., Zeng, D., Zhao, J.: Event extraction via dynamic multi-pooling convolutional neural networks. In: Proceedings of ACL, pp. 167–176 (2015)
5. Gupta, P., Ji, H.: Predicting unknown time arguments based on cross-event propagation. In: Proceedings of ACL-IJCNLP, pp. 369–372 (2009)
6. Hermann, K.M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., Blunsom, P.: Teaching machines to read and comprehend. In: Advances in Neural Information Processing Systems, pp. 1684–1692 (2015)

7. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors (2012). arXiv preprint arXiv:1207.0580
8. Hochreiter, S.: The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. **6**(02), 107–116 (1998)
9. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies (2001)
10. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
11. Hong, Y., Zhang, J., Ma, B., Yao, J., Zhou, G., Zhu, Q.: Using cross-entity inference to improve event extraction. In: Proceedings of ACL, pp. 1127–1136 (2011)
12. Irsoy, O., Cardie, C.: Opinion mining with deep recurrent neural networks. In: Proceedings of EMNLP, pp. 720–728 (2014)
13. Ji, H., Grishman, R.: Refining event extraction through cross-document inference. In: Proceedings of ACL, pp. 254–262 (2008)
14. Kalchbrenner, N., Blunsom, P.: Recurrent convolutional neural networks for discourse compositionality (2013). arXiv preprint arXiv:1306.3584
15. Lai, S., Xu, L., Liu, K., Zhao, J.: Recurrent convolutional neural networks for text classification. In: Proceedings of AAAI (2015)
16. Li, Q., Ji, H., Hong, Y., Li, S.: Constructing information networks using one single model. In: Proceedings of EMNLP (2014)
17. Li, Q., Ji, H., Huang, L.: Joint event extraction via structured prediction with global features. In: Proceedings of ACL, pp. 73–82 (2013)
18. Liao, S., Grishman, R.: Using document level cross-event inference to improve event extraction. In: Proceedings of ACL, pp. 789–797 (2010)
19. McClosky, D., Surdeanu, M., Manning, C.D.: Event extraction as dependency parsing. In: Proceedings of ACL, pp. 1626–1635 (2011)
20. Nguyen, T.H., Grishman, R.: Event detection and domain adaptation with convolutional neural networks. In: Proceedings of ACL, pp. 365–371 (2015)
21. Socher, R., Perelygin, A., Wu, J.Y., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP), vol. 1631, pp. 1642. Citeseer (2013)
22. Turian, J., Ratinov, L., Bengio, Y.: Word representations: a simple and general method for semi-supervised learning. In: Proceedings of ACL, pp. 384–394 (2010)
23. Zaremba, W., Sutskever, I.: Learning to execute (2014). arXiv preprint arXiv:1410.4615
24. Zeiler, M.D.: Adadelta: an adaptive learning rate method (2012). arXiv preprint arXiv:1212.5701