Event Detection via Recurrent Neural Network and Argument Prediction

Wentao Wu(✉), Xiaoxu Zhu, Jiaming Tao, and Peifeng Li

School of Computer Science and Technology,
Soochow University, Suzhou, China
{wtwu, jmtao}@stu.suda.edu.cn, {xxzhu, pfli}@suda.edu.cn

Abstract. This paper tackles the task of event detection, which involves identifying and categorizing the events. Currently event detection remains a challenging task due to the difficulty at encoding the event semantics in complicate contexts. The core semantics of an event may derive from its trigger and arguments. However, most of previous studies failed to capture the argument semantics in event detection. To address this issue, this paper first provides a rule-based method to predict candidate arguments on the event types of possibilities, and then proposes a recurrent neural network model RNN-ARG with the attention mechanism for event detection to capture meaningful semantic regularities form these predicted candidate arguments. The experimental results on the ACE 2005 English corpus show that our approach achieves competitive results compared with previous work.

Keywords: Event detection · Argument prediction · Recurrent neural network

1 Introduction

Event extraction is divided into two subtasks, event detection (or trigger extraction) and argument extraction. The former focuses on identifying event triggers and categorizing their event types, while the latter aims to extract various arguments of a specific event type and assign them roles. Commonly, event triggers are single verbs or nominalizations that evoke some real-world events, while event arguments are composed of entity instances and play a certain role in an event. For example, in the sentence "**He** *died* in the wave of kidnappings in **Iraq**", event detection should recognize the token "died" as the trigger of the event type *died* and argument extraction should identify the entities "He" and "Iraq" as the arguments of this *died* event, and assign them the roles *Victim* and *Place*, respectively. This paper focuses on event detection because it is still the bottleneck of event extraction for its low performance.

Pipeline models are widely used in previous studies (Liao and Grishman [1]; Hong et al. [2]), where argument extraction is the subsequent stage of event detection. Therefore, argument information cannot be applied to event detection directly. However, event arguments are very effective for event detection. Take the following two sentences as examples:

M. Zhang et al. (Eds.): NLPCC 2018, LNAI 11109, pp. 235–245, 2018.
https://doi.org/10.1007/978-3-319-99501-4_20

S1: *Iraqis have* ***fired*** *sand missiles in this war.*
S2: MSNBC *has* ***fired*** *Phil Donahue.*

The sentences S1 and S2 have the same trigger word "fired". With the argument information, it is easy to identify S1 as an *Attack* event due to the entity "missiles" (*Weapon*) and S2 as an *End-Position* event due to the entity "Phil Donahue" (*Person*).

Unfortunately, during the stage of event detection, we do not know which entities act as the arguments of events. Most previous methods (e.g., Ji and Grishman [3]; Liao and Grishman [1]; Hong et al. [2]) approximatively used the either syntactically or physically nearest entities to the trigger as argument features. However, many arguments are far from their triggers in either syntactically or physically distance. Besides, neural network models (Nguyen and Grishman [4, 5], Chen et al. [6]; Sha et al. [7]) were applied to event detection, most of them focused on sequence and chunk information from specific contexts, ignoring the effect of argument information.

Arguments are capable of providing significant clues to event detection, how to provide accurate argument information to event detection is vital to the performance of event detection. To tackle this issue, we first propose a method to predict candidate arguments on the event types of possibilities and then apply them to a recurrent neural network to detect events. The experimental results on the ACE 2005[1] English dataset show that our model outperforms the state-of-the-art baselines.

2 Related Work

Various methods have been proposed for event detection. Early research has primarily focused on local-sentence representations, such as the lexical features (e.g., full word, POS), syntactic features (e.g., dependency features) and external knowledge features (WordNet) (Ahn [8]). Currently, global inference and joint model are widely used in event detection. Ji and Grishman [3] combined global evidence from related documents with local decisions. Gupta and Ji [9], Liao and Grishman [1] and Hong et al. [2] proposed cross-event and cross-entity inference for the event extraction task.

Representation-based approaches have been introduced into event detection very recently, which represent candidate event mentions by embeddings and fed them into neural networks (Chen et al. [6]; Nguyen and Grishman [4, 5]). Furthermore, Nguyen et al. [10] employed bidirectional RNN, which could jointly extract event trigger and arguments. However, joint model only makes remarkable improvements to argument extraction, but insignificant to event detection. Sha et al. [7] employed a dependency bridge recurrent neural network (dbRNN) for event extraction, which simultaneously applying tree structure and sequence structure in RNN to capture sequence and syntax information from specific context. Unfortunately, it totally ignored importance of arguments for event detection. Liu et al. [11] proposed a three-layer Artificial Neural Networks (ANNs) model on annotated arguments and the words around them to model

[1] https://catalog.ldc.upenn.edu/LDC2006T06.

the event detection task. It exploits argument information explicitly for event detection via supervised attention mechanisms, which construct gold attention for each trigger candidate based on annotated arguments in the training procedure.

3 Approach

We first propose a rule-based method to predict candidate arguments and then introduce RNN-ARG (Recurrent Neural Networks with Arguments) to detect events. In our model, we incorporate a Bi-LSTM (Bi-directional Long Short-Term Memory) to model the preceding and following information of a word and use another Bi-LSTM model with the attention mechanism to learn the representation of trigger and arguments.

3.1 Argument Prediction

When we cannot recognize event type from the semantics of trigger directly, it is very important to consider argument semantics. Most of previous studies adopted the syntactically or physically nearest entities to the trigger to represent argument semantics. Due to upstream errors from the syntax parsing and the diversity of sentence expression, this method always introduces many pseudo arguments to vent detection and then harms the precision. Another method is to consider all of the entities in the event sentence as arguments. Obviously, it will introduce more pseudo arguments to event detection. In this paper, we propose an event arguments prediction method to predict based on the event types of possibilities and the corresponding entity types, which can act as event roles following the definitions of event types.

In an event, the event type dominates its argument numbers and argument types, i.e., roles. Hence, we must detect the event type firstly and then select candidate arguments following the definition of the event type. Unfortunately, we do not know the event type before event detection. However, we can enumerate all possible types of a trigger word according to the annotation training data. In particular, 85.6% of the trigger words in the training set only refer to one event type, 11.7% of them belong to two distinct event types, and the rest (2.7%) has three or more event types.

Firstly, we enumerate all possible types of a candidate trigger *w*. If *w* does not appear in the training set, we first calculate the similarities between *w* and each annotated trigger in training set using WordNet similarity. Then we find a trigger *tri* in the training set, who has the highest similarity with *w*. For instance, in S3 the candidate trigger "discussed" does not appear in the training set and its most similar trigger is "talked".

S3: *Chretien/PER said that he/PER and Bush/PER, who/PER had not spoken since late February/TIME,* ***discussed*** *issues including Iraq/GPE and aid to Africa/GPE.*

Secondly, we select top two high frequency event types of the candidate trigger *w* (if *w* does not occur in the training set, we use *tri* to replace it) according to the statistics on the training set. Because "discussed" does not occur in the training set, we use its similar trigger "talked" to find event types. The trigger word "talked" belongs to

the event type *Phone-Write* (60%) and *Meet* (40%). Hence, we choose the top two event types *Phone-Write* and *Meet* for the trigger "discussed".

Finally, we extract the candidate arguments on entity type matching. For an event type *i*, we first extract all entity types, who can act as a role (we do not consider the role *Place* and *Time* because they do not have obvious event type discrimination) of this event type following the event definition, and store them to the list $EntType_i$. Then from the list of the annotated entities in the event mention, we extract the entities whose types belong to $EntType_i$, as candidate arguments. For example, the event type *Phone-Write* only has one role *Entity*, which can fill entities whose type are PER/ORG. Hence the list of entities [Chretien, he, Bush, who] can act as candidate arguments of *Phone-Write* event, due to their entity types PER or ORG.

3.2 RNN-ARG Model

The RNN-ARG model is showed in Fig. 1. The model contains two Bi-LSTM neural networks. Specifically, we first use a Bi-LSTM to encode semantics of each word with its preceding and following information. Then we add an attention-based Bi-LSTM neural network to capture the semantics of trigger and arguments.

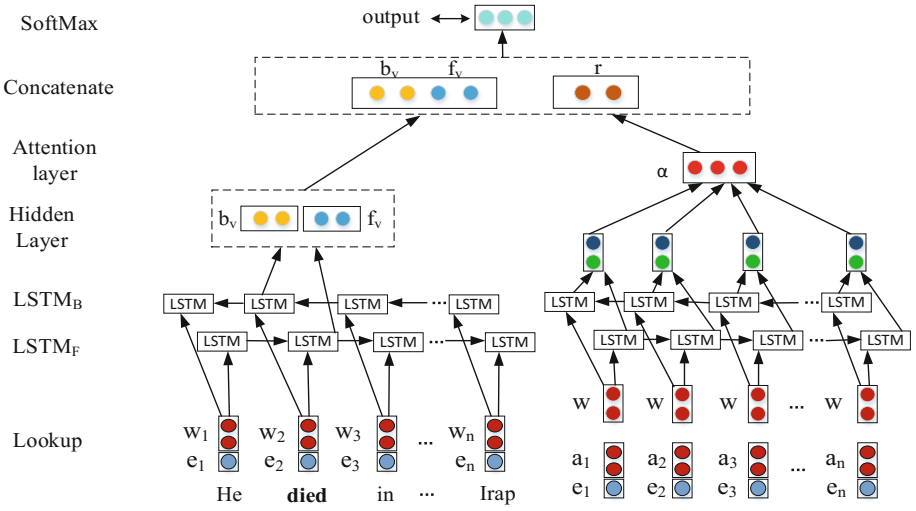

Fig. 1. The architecture of the model RNN-ARG (here the trigger candidate is "died")

- **Bi-LSTM**

In the sentence encoding phase, we take all the words of the whole sentence as the input and each token w_i into a real-valued vector x_i using the concatenation of the following two vectors:

(1) **Word Embedding Table**: Word embeddings are able to capture the meaningful semantic resularities (Bengio et al. [12]) and we train the word embeddings using Skip-Gram[2] (Mikolov et al. [13]) algorithm on the NYT corpus[3].
(2) **Entity Type Embedding Table**: Following existing work (Li et al. [14]; Chen et al. [6]; Nguyen and Grishman [4]), we exploit the annotated entity information as additional features. We randomly initialize embedding vectors for each entity type (including "None" which refers to an undefined entity type) and update it in the training procedure.

The transformation from the token w_i to the vector x_i essentially converts the input sentence W into a sequence of real-valued vectors $\mathrm{X} = (x_1, x_2, \ldots, x_n)$, to be used by recurrent neural networks to learn a more effective representation.

At each step *t,* the LSTM accepts current input $\boldsymbol{x}_t$ and previous hidden state $\boldsymbol{h}_{t\text{-}1}$ to compute hidden state. We run $LSTM_F$ from the beginning to the end of sequence, and run $LSTM_B$ from the end to the beginning of the sequence, while producing sequences of vectors for forward propagation $\vec{\boldsymbol{h}}_t = LSTM_F\left(\boldsymbol{x}_t, \vec{\boldsymbol{h}}_{t-1}\right)$ and another for the backward propagation $\overleftarrow{\boldsymbol{h}}_t = LSTM_B\left(\boldsymbol{x}_t, \overleftarrow{\boldsymbol{h}}_{t+1}\right)$ respectively. Afterwards we concatenate the vectors $\boldsymbol{h}_t = \left[\vec{\boldsymbol{h}}_t; \overleftarrow{\boldsymbol{h}}_t\right]$ for each time step, and $\boldsymbol{h}_t$ is reduced into a single vector as the current state. The LSTM holds a state representative as a continuous vector passed to the subsequent time step, and it is capable of modeling long-range dependencies due to its gated memory. Afterwards, we concatenate of the hidden states f_v of the $LSTM_F$ and b_v of the $LSTM_B$ as the final output of Bi-LSTM, instead of averaging the last hidden vectors of the $LSTM_F$ and $LSTM_B$.

- **Bi-LSTM with Attention**

Different from sentence encoding which used the entire sentence as input of Bi-LSTM. In this encoding phase, we take the candidate trigger and its predicted arguments as the input of an event. We use w to denote the current candidate trigger, $[\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n]$ to denote the candidate arguments/entities in the list, which is extracted by argument prediction method, and $[\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n]$ to denote the entity types of the candidate arguments in the list. The element at position i of the input sequence is resolved by a vector $\boldsymbol{v}_i$ as follows:

$$\boldsymbol{v}_i = \boldsymbol{w} \oplus \boldsymbol{a}_i \oplus \boldsymbol{e}_i \tag{1}$$

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

where $\oplus$ is the concatenation operator. Note that, both $\boldsymbol{w}$, $\boldsymbol{a}_i$ and $\boldsymbol{e}_i$ are originally in symbolic representation. Before entering the recurrent neural network, we transform them into real-valued vector by two look-up tables, Word Embedding Table and Entity Type Embedding Table. For the recurrent setup, we use a layer of LSTM networks in a bidirectional manner. The forward and backward LSTMs traverse the sequence v_i, producing sequences of vectors $\vec{\boldsymbol{h}}_t$ and $\overleftarrow{\boldsymbol{h}}_t$ respectively. Then the output at time t is $\boldsymbol{h}_t = \left[\vec{\boldsymbol{h}}_t; \overleftarrow{\boldsymbol{h}}_t\right]$. Finally, this model acquires weighted sum over $\boldsymbol{h}_t$ by using attention, and calculates the final vectors of the argument semantics. The attention mechanism lets the model decide the importance of each predict argument by weighing them when constructing the representation of the sequence. The attention layer contains the trainable vector $\boldsymbol{\omega}$ (of the same dimensionality as vectors $\boldsymbol{h}_t$) which is used to dynamically produce a weight vector $\boldsymbol{\alpha}$ over time steps t as follows.

$$\boldsymbol{\alpha} = \text{softmax}(\boldsymbol{\omega}^{\text{T}}\tanh(\boldsymbol{H})) \tag{2}$$

where $\boldsymbol{H}$ is a matrix consisting of vectors $\boldsymbol{h}_t$. The output layer $\boldsymbol{r}$ is the weighted sum of vectors in $\boldsymbol{H}$:

$$\boldsymbol{r} = \boldsymbol{H}\boldsymbol{\alpha}^{\text{T}} \tag{3}$$

This representation vector $\boldsymbol{r}$ obtained from the attention layer is a high-level encoding of the trigger and arguments, which is used as input to the final softmax layer for the classification.

- **Final Classification**

Finally, we concatenate the sequence feature $\boldsymbol{b}_v$ and $\boldsymbol{f}_v$ which are learned from the Bi-LSTM, and argument semantic $\boldsymbol{r}$, which is the output of attention based Bi-LSTM, as a single vector $\boldsymbol{F} = [\boldsymbol{b}_v, \boldsymbol{f}_v, \boldsymbol{r}]$. To compute the confidence of each event type, the feature vector $\boldsymbol{F} \in \mathbb{R}^{4d}$, where d is the dimension of hidden state vector, is fed into a classifier. We exploit a softmax approach to identify trigger candidates and classify each trigger candidate as a specific event type as follows.

$$\boldsymbol{O} = \boldsymbol{W}_s\boldsymbol{F} + \boldsymbol{b}_s \tag{4}$$

where $\boldsymbol{W}_s \in \mathbb{R}^{n\times(4d)}$ is the transformation matrix and $\boldsymbol{O} \in \mathbb{R}^n$ is the final output of the network, n is equal to the number of the event type including the "None" label for the candidate trigger which do not belong to any event type. For regularization, we also employ dropout (Kim [15]) on the penultimate layer.

3.3 Model Training

We define all of the parameters for the stage of trigger classification to be trained as $\theta = (\boldsymbol{E}, \boldsymbol{ET}, \boldsymbol{lf}, \boldsymbol{lb}, \boldsymbol{rf}, \boldsymbol{rb}, \boldsymbol{\omega}, \boldsymbol{W}_s, \boldsymbol{b}_s)$. Specifically, $\boldsymbol{E}$ is the word embedding, $\boldsymbol{ET}$ is the embedding of the entity type, $\boldsymbol{lf}$ and $\boldsymbol{lb}$ are parameters of forward-LSTM and backward-LSTM, $\boldsymbol{rf}$ and $\boldsymbol{rb}$ are parameters of another forward-LSTM and

backward-LSTM. $\boldsymbol{\omega}$ is parameters of attention layer, $\boldsymbol{W}_s$ and $\boldsymbol{b}_s$ are all of the parameters of the output layer.

Given an input example $\mathbf{x}$, the network with parameter θ outputs the vector $\mathbf{O}$, where the i-th value o_i of $\mathbf{O}$ is the confident score for classifying $\mathbf{x}$ to the i-th event type. To obtain the conditional probability $p(i|\mathbf{x}, \theta)$, we apply a softmax operation over all event types:

$$p(i|\mathbf{x}, \theta) = \frac{e^{o_i}}{\sum_{k=1}^{m} e^{o_k}} \tag{5}$$

The model can be trained in an end-to-end way by back propagation, where the objective function (loss function) is the cross-entropy loss. Given all of our (suppose T) training examples $(\mathbf{x}_i; y_i)$, we can then define the negative log-likelihood loss function as follows:

$$J(\theta) = -\sum_{i=1}^{T} \log p(y^{(i)}|\mathbf{x}^{(i)}, \theta) \tag{6}$$

To compute the network parameter θ, we train the model by stochastic gradient descent over shuffled mini-batches with Adam (Kingma and Ba [16]) rule.

4 Experiments

We first introduce the experimental setting and then report the experimental results and analysis.

4.1 Experimental Setting

We evaluate our model on the ACE 2005 English corpus and use the same data split and annotated entity mention as the previous work (Liao and Grishman [1]; Hong et al. [2]; Li et al. [14]; Nguyen and Grishman [4]). This data split includes 40 newswire documents for the test set, 30 other documents for the development set and remaining 529 documents as the training set. Besides, we report the micro-average Precision (P), Recall (R) and F1-score (F1), following the standards defined in (Ji and Grishman [3]). In our evaluation, we extract all annotated trigger words in the training set and use them to find the candidate triggers in the test set. Finally, 85.3% of trigger mentions are selected as candidates.

Hyper-parameters are tuned on the development set. We set 300, 50 dimensions for the word embeddings, the entity type embeddings, respectively. The hidden layer vector dimension of LSTM is 128. To prevent overfitting, we inherit the values for the other parameters from (Kim [15]), the dropout rate is set to 0.5, the mini-batch size is 50.

4.2 Experiments Results

We compare our model with the following baselines: (1) **CNN** (Chen et al. [6]), which exploits a dynamic multi-pooling convolutional neural network for event detection;

(2) **JointM** (Nguyen et al. [10]), which employs a bi-directional RNN to jointly extract event triggers and arguments; (3) **dbRNN** (Lei et al. [7]), which proposes a novel dependency bridge recurrent neural network (dbRNN) for event extraction; (4) **ANN-ATT** (Liu et al. [11]), which leverages additional arguments information for event detection.

Table 1 shows the results of the above five models and RNN-ARG achieves the comparably F1-score. Compared with two simple neural networks models CNN and JointM, our model RNN-ARG significantly improves the F1-score on event detection (trigger classification) by 2.5 and 2.3, respectively. This verifies that combined neural networks model is an appreciate model for the task of event extraction to capture more event semantics, and the attention layer is effective for capture more valuable information to extract an event when using recurrent neural networks.

Table 1. Comparison of event detection models on ACE.

Model	Trigger identification			Trigger classification		
	P	R	F	P	R	F
CNN	80.4	67.7	73.5	75.6	63.6	69.1
JointM	68.5	75.7	71.9	66.0	73.0	69.3
dbRNN	N/A	N/A	N/A	74.1	69.8	71.9
ANN-ATT	N/A	N/A	N/A	78.0	66.3	71.7
RNN-ARG	75.3	71.5	73.4	73.6	**69.8**	**71.6**

Compared with dbRNN, which carry syntactically related information when modeling each word by enhancing Bi-LSTM with dependency bridges. Our model only used two simple recurrent neural networks, but it still performs comparably with the enhanced model. In particularly, two models have achieved the same highest recall. This result justifies the effectiveness of the argument semantics and our RNN-ARG to detect events. With the additional predicted argument information, our RNN-ARG can learn more vital information from triggers, arguments and their combination.

Compared with ANN-ATT, which also introduces argument semantics to their model, RNN-ARG has achieved a very close F1-score on event detection, with the higher recall (+3.5%). ANN-ATT used annotated arguments to train attention mechanism and it only captured existed combination of trigger and arguments, ignoring other possible combination of trigger and arguments. However, our argument prediction can enumerate all possible arguments and provide more argument semantic information. Its disadvantage is it will introduce lots of pseudo arguments to our RNN-ARG to reduce the precision.

4.3 Analysis

To analyze the effectiveness of the argument semantics, we conduct the following models for comparison: (1) **ALLENT**, which regards all entities in the event mention

as candidate arguments; (2) **ARG-1**, which predicts arguments based on only top one high frequency event type on the training set; (3) **ARG-1 w/o Att**, which refers to **ARG-1** without the attention mechanism; (4) **RNN-ARG**, our model which predicts arguments based on top two high frequency event types on the training set; (5) **ALL-TYP**, which predicts arguments based on all possible event types on the training set.

Table 2 shows the results of the above five models and RNN-ARG achieves the highest F1-score. In the training set, most trigger words only refer to 1–2 distinct event types. 85.6% of the trigger words refer to one event type, 11.7% of the trigger words belong to two distinct event types. These figures also justify that ARG-1 can achieve a relatively high F1-score.

Table 2. Experimental results of the variants of RNN-ARG on ACE.

Model	Trigger identification			Trigger classification		
	P	R	F	P	R	F
ALLENT	77.9	67.4	72.3	74.8	64.8	69.4
ARG-1(w/o Att)	78.5	67.6	72.6	76.2	65.3	70.3
ARG-1	75.2	71.0	73.1	72.8	68.7	70.7
RNN-ARG	75.3	71.5	73.4	73.6	**69.8**	**71.6**
ALLTYP	77.9	68.5	72.9	75.4	66.4	70.6

The statistics on the trigger words belonging to two distinct event types shows that 63.1% of the distribution on event type is larger than 3:7. This figure ensures that RNN-ARG outperforms ARG-1 in F1-score. Besides, the F1-score of ALLTYP is smaller than that of RNN-ARG, because it will introduce many pseudo arguments to the model.

ALLENT takes all the entities in the sentence containing the event mention as predicted arguments, and it obviously introduces many pseudo arguments to the model and then lead to ambiguity. Thus, its F1-score is lower than those of the other four models. Besides, ARG-1 outperforms ARG-1 w/o Att in F1-score and this result shows that the attention mechanism can optimize the final output vector, and mine richer semantic information from triggers and arguments.

Moreover, we also analyze the error results in our model RNN-ARG. Table 1 shows that 21.6% of pseudo instances are identified as event mentions by mistake. The main reason is that a trigger word may have more than one tense (especially support verbs, such as “sent”, “go”). Annotation ambiguity is also a problem and many event mentions are not annotated in the ACE corpus. For example, the trigger word “shot” in the sentence “she was shot herself”, which actually contains an *Attack* event and a *Die* event, only be assigned one event type for the annotation rule: each trigger mention only has one event type.

16.7% of trigger mentions in the test set belong to unknown trigger words (never appear in the training set), these mentions cannot be identified due to our candidate selection mechanism mentioned in Subsect 4.1. Otherwise, those nominal triggers

(5.8%) (such as "tours" and "missions") are hard to be recognized, because they lack enough event arguments to indicate their event type. Finally, some trigger words may belong to more than one event type, and they are easy to be identified wrongly.

5 Conclusion

Arguments are capable of providing significant clues to event detection. This paper proposes a novel RNN-ARG model with the attention mechanism and predicted arguments to detect events. The experimental results show the effectiveness of our model. Our future work will focus on extracting accurate arguments and giving effective representation to detect events.

Acknowledgments. The authors would like to thank three anonymous reviewers for their comments on this paper. This research was supported by the National Natural Science Foundation of China under Grant Nos. 61772354, 61773276 and 61472265, and was also supported by the Strategic Pioneer Research Projects of Defense Science and Technology under Grant No. 17-ZLXDXX-02-06-02-04.

References

1. Liao, S., Grishman, R.: Using document level cross-event inference to improve event extraction. In: ACL 2010, pp. 789–797 (2010)
2. Hong, Y., et al.: Using cross-entity inference to improve event extraction. In: ACL 2011, pp. 1127–1136 (2011)
3. Ji, H., Grishman, R.: Refining event extraction through cross-document inference. In: ACL-HLT 2008, pp. 254–262 (2008)
4. Nguyen, H.T., Grishman, R.: Event detection and domain adaptation with convolutional neural networks. In: ACL 2015, pp. 365–371 (2015)
5. Nguyen, H.T., Grishman, R.: Modeling skip-grams for event detection with convolutional neural networks. In: EMNLP 2016, pp. 886–891 (2016)
6. Chen, Y., Xu, L., Liu, K., Zeng, D., Zhao, J.: Event extraction via dynamic multi-pooling convolutional neural networks. In: ACL 2015, pp. 167–176 (2015)
7. Sha, L., Qian, F., Chang, B., Sui, Z.: Jointly extraction event trigger and arguments by dependency-bridge RNN and tensor-based argument interaction. In: AAAI 2018 (2018)
8. Ahn, D.: The stages of event extraction. In: Proceedings of the ACL 2006, pp. 1–8 (2006)
9. Gupta, P., Ji, H.: Predicting unknown time arguments based on cross event propagation. In: ACL-IJCNLP 2009, pp. 369–372 (2009)
10. Nguyen, H.T., Cho, K., Grishman, R.: Joint event extraction via recurrent neural networks. In: ACL 2016, pp. 300–309 (2016)
11. Liu, S., Chen, Y., Liu, K., Zhao, J.: Exploiting argument Information to improve event detection via supervised attention mechanisms. In: ACL-2017, pp. 1789–1798 (2017)
12. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**, 1137–1155 (2003)
13. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS 2013, UK, pp. 3111–3119 (2013)

14. Li, Q., Ji, H., Huang L.: Joint event extraction via structured prediction with global features. In: ACL 2013, pp. 73–82 (2013)
15. Kim, Y.: Convolutional neural networks for sentence classification. In: EMNLP 2014, pp. 1746–1751 (2014)
16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv: 1412.6980 (2014)