# Event Detection with Trigger-Aware Lattice Neural Network

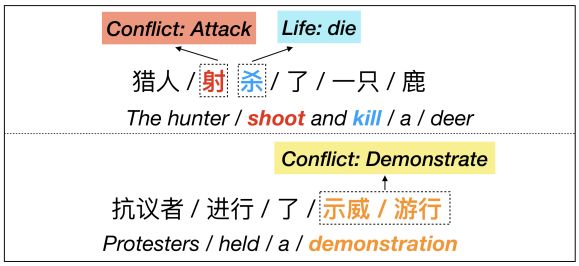
**Ning Ding**[1,2*], **Ziran Li**[1,2*], **Zhiyuan Liu**[2,3,4], **Hai-Tao Zheng**[1,2†], **Zibo Lin**[1,2]
[1]Tsinghua Shenzhen International Graduate School, Tsinghua University
[2]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[3]Institute for Artificial Intelligence, Tsinghua University, Beijing, China
[4]State Key Lab on Intelligent Technology and Systems, Tsinghua University, Beijing, China
{dingn18}@mails.tsinghua.edu.cn

## Abstract

Event detection (ED) aims to locate trigger words in raw text and then classify them into correct event types. In this task, neural network based models became mainstream in recent years. However, two problems arise when it comes to languages without natural delimiters, such as Chinese. First, word-based models severely suffer from the problem of word-trigger mismatch, limiting the performance of the methods. In addition, even if trigger words could be accurately located, the ambiguity of polysemy of triggers could still affect the trigger classification stage. To address the two issues simultaneously, we propose the Trigger-aware Lattice Neural Network (TLNN). (1) The framework dynamically incorporates word and character information so that the trigger-word mismatch issue can be avoided. (2) Moreover, for polysemous characters and words, we model all senses of them with the help of an external linguistic knowledge base, so as to alleviate the problem of ambiguous triggers. Experiments on two benchmark datasets show that our model could effectively tackle the two issues and outperforms previous state-of-the-art methods significantly, giving the best results. The source code of this paper can be obtained from https://github.com/thunlp/TLNN.

## 1 Introduction

Event Detection (ED) is a pivotal part of Event Extraction, which aims to detect the position of event triggers in raw text and classify them into corresponding event types. Conventionally, the stage of locating trigger words is known as **Trigger Identification (TI)**, and the stage of classifying trigger words into particular event types




(a) An example of trigger-word mismatch.


(b) An example of polysemous trigger.

Figure 1: Examples about trigger-word mismatch and polysemous trigger in Event Detection.

is called **Trigger Classification (TC)**. Although neural network methods have achieved significant progress in event detection (Nguyen and Grishman, 2015; Chen et al., 2015; Zeng et al., 2016), both steps are still exposed to the following two issues.

In the TI stage, the problem of **trigger-word mismatch** could severely impact the performance of event detection systems. Because in languages without natural delimiters such as Chinese, mainstream approaches are mostly word-based models, in which the segmentation should be firstly performed as a necessary preprocessing step. Unfortunately, these word-wise methods neglect an important problem that a trigger could be a specific part of one word or contain multiple words. As shown in Figure 1(a), "射" (shoot) and "杀" (kill)

* indicates equal contribution
† Corresponding author: Hai-Tao Zheng. ( E-mail: zheng.haitao@sz.tsinghua.edu.cn )



*Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 347–356, Hong Kong, China, November 3–7, 2019. 

| Datasets | Trigger-word Match | | Trigger Polysemy | |
|---|---|---|---|---|
| | Match | Mismatch | Polysemy | Univocal |
| ACE 2005 | 85.39% | 14.61% | 41.64% | 58.36% |
| KBP 2017 | 76.28% | 23.72% | 52.14% | 47.86% |

Table 1: Proportion of Trigger-word mismatch and Polysemous Triggers on ACE 2005 and KBP 2017.

are two triggers that both are parts of the word "射杀" (shoot and kill) . In the other case, "示威游行" (demonstration) is a trigger that crosses two words. Under this circumstance, triggers could not be located correctly with word-based methods, thereby becoming a serious limitation of the task. Some feature-based methods are proposed (Chen and Ji, 2009; Qin et al., 2010; Li and Zhou, 2012) to alleviate the issue, but they heavily rely on the hand-crafted features. Lin et al. (2018) proposes the nugget proposal networks (NPN) in terms of this issue, which uses a neural network to model character compositional structure of trigger words in a fix-sized window. However, the mechanism of the NPN limits the scope of trigger candidates within a fix-sized window, which is inflexible and suffering from the problem of trigger overlaps.

Even if the locations of triggers can be correctly detected in the TI step, the TC step could still be severely affected by the inherent problem of **ambiguity of polysemy**. Because a trigger word with multiple word senses could be classified into different event types. Take 1(b) as an example, a polysemous trigger word "释放" (release) could represent two distinctly different event types. In the first case, the word 'release' triggers an *Attack* event (release tear gas). But in the second case, the event triggered by 'release' becomes *Release-Parole* (release a man in court).

To further illustrate that the two problems mentioned above do exist, we make manual statistics on the proportion of mismatch triggers and polysemous triggers on two widely used datasets. The statistics are illustrated in Table 1, and we can observe that data with trigger-word mismatch and trigger polysemy do account for a considerable proportion and then affect the task.

In this paper, we propose the Trigger-aware Lattice Network (TLNN), a comprehensive model that can simultaneously tackle both issues. To avoid error propagation by NLP tools like segmentor, we take characters as the basic units of the input sequence. Moreover, we utilize HowNet (Dong and Dong, 2003), an external knowledge base that manually annotates polysemous Chinese and English words, to obtain the sense-level information. Further, we develop the trigger-aware lattice LSTM as the feature extractor of our model, which could leverage character-level, word-level and sense-level information at the same time. More specifically, in order to address the trigger-word mismatch issue, we construct short cut paths to link the cell state between the start and the end characters for each word. It is worth mentioning that the paths are sense-level, which means all the sense information of words that end in one specific character will flow into the memory cell of the character. Hence, with the utilization of multiple granularity of information (character, word and word sense), the problem of polysemous triggers could be effectively alleviated.

We conduct sets of experiments on two real-world datasets in the task of event detection. Empirical results of the main experiments show that our model can efficiently address both mentioned issues. With comprehensive comparisons with other proposed methods, our model achieves the state-of-the-art results on both datasets. Further, sets of subsidiary experiments are conducted to further analyze how TLNN addresses the two issues.

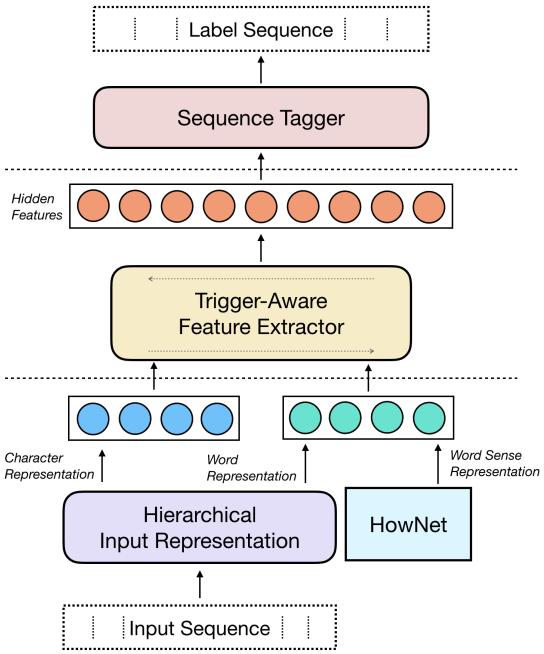



Figure 2: The architecture of TLNN.

## 2 Methodology

In the paper, event detection is regarded as a sequence labelling task. For each character, the model should identify if it is a part of one trigger and correctly classify the trigger into one specific event type.

The architecture of our model is shown in Figure 2, which primarily includes the following three parts: (1) **Hierarchical Representation Learning**, which reveals the character-level, word-level and sense-level embedding vectors in an unsupervised way. (2) **Trigger-aware Feature Extractor**, which automatically extracts different levels of semantic features by a tree structure LSTM model. (3) **Sequence Tagger**, which calculates the probability of being a trigger for each character candidate.

### 2.1 Hierarchical Representation Learning

Given an input sequence $S = \{c_1, c_2, ..., c_N\}$, where $c_i$ represents the $i$th character in the sequence. In character level, each character will be represented as an embedding vector $\boldsymbol{x^c}$ by Skip-Gram method (Mikolov et al., 2013).

$$\boldsymbol{x}_i^c = \boldsymbol{e}(c_i) \tag{1}$$

In the word level, the input sequence $S$ could also be $S = \{w_1, w_2, ..., w_M\}$, where the basic unit is a single word $w_i$. In this paper, we will use two indexes $b$ and $e$ to represent the start and the end of a word. In this case, the word embeddings are:

$$\boldsymbol{x}_{b,e}^w = \boldsymbol{e}(w_{b,e}) \tag{2}$$

However, the Skip-Gram method maps each word to only one single embedding, ignoring the fact that many words have multiple senses. Hence representation of finer granularity is still necessary to represent deep semantics. With the help of HowNet (Dong and Dong, 2003), we can obtain the representation of each sense of a character or a word. For each character $c$, there are possible multiple senses $sen^{(c_i)} \in S^{(c)}$ annotated in HowNet. Similarly, for each word $w$, the senses could be $sen^{(w_i)} \in S^{(w)}$. Consequently, we can obtain the embeddings of senses by jointly learning word and sense embeddings via Skip-gram manner. This mechanism is also applied to (Niu et al., 2017).

$$\boldsymbol{s}_j^{c_i} = \boldsymbol{e}(sen_j^{(c_i)}) \tag{3}$$

$$\boldsymbol{s}_j^{w_{b,e}} = \boldsymbol{e}(sen_j^{(w_{b,e})}) \tag{4}$$

where $sen_j^{(c_i)}$ and $sen_j^{(w_{b,e})}$ represents the $j$th sense for the character $c_i$ and word $w_{b,e}$ in the sequence. And then $\boldsymbol{s_j^{c_i}}$ and $\boldsymbol{s_j^{w_{b,e}}}$ are the embeddings of $c_i$ and $w_{b,e}$.

### 2.2 Trigger-Aware Feature Extractor

The trigger-aware feature extractor is the core component of our model. After training, the outputs of the extractor are the hidden state vectors **h** of an input sentence.

**Conventional LSTM.** LSTM (Hochreiter and Schmidhuber, 1997) is an extension of the recurrent neural network (RNN) with additional gates to control the information. Traditionally, there are following basic gates in LSTM: input gate $\boldsymbol{i}$, output gate $\boldsymbol{o}$ and forget gate $\boldsymbol{f}$. They collectively controls which information will be reserved, forgotten and output. All three gates are accompanied by corresponding weight matrix $\boldsymbol{W}$. Current cell state $\boldsymbol{c}$ records all historical information flow up to the current time. Therefore, the character-based LSTM functions are:

$$\begin{bmatrix} \boldsymbol{i}_i^c \\ \boldsymbol{o}_i^c \\ \boldsymbol{f}_i^c \\ \tilde{\boldsymbol{c}}_i^c \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} (\boldsymbol{W}^{c\top} \begin{bmatrix} \boldsymbol{x}_i^c \\ \boldsymbol{h}_{i-1}^c \end{bmatrix} + \boldsymbol{b}^c) \tag{5}$$

$$\boldsymbol{c}_i^c = \boldsymbol{f}_i^c \odot \boldsymbol{c}_{i-1}^c + \boldsymbol{i}_i^c \odot \tilde{\boldsymbol{c}}_i^c \tag{6}$$

$$\boldsymbol{h}_i^c = \boldsymbol{o}_i^c \odot tanh(\boldsymbol{c}_i^c) \tag{7}$$

where $h_i^c$ is the hidden state vector.

**Trigger-Aware Lattice LSTM.** Trigger-Aware Lattice LSTM is the core feature extractor of our framework, which is an extension of LSTM and lattice LSTM. In this subsection, We will derive and theoretically analyze the model in detail.

In this section, characters and words are assumed to have $K$ senses. As mentioned in 2.1, for the $j$th sense of the $i$th character $c_i$, the embedding would be $\boldsymbol{s_j^{c_i}}$ . Then an additional LSTMCell is utilized to integrate all senses of the character, hence the calculation of the cell gate of the multi-sense character $\boldsymbol{c_i}$ would be:

$$\begin{bmatrix} \boldsymbol{i}_j^{c_i} \\ \boldsymbol{f}_j^{c_i} \\ \tilde{\boldsymbol{c}}_j^{c_i} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ tanh \end{bmatrix} (\boldsymbol{W}^{c\top} \begin{bmatrix} \boldsymbol{s}_j^{c_i} \\ \boldsymbol{h}^{c_{i-1}} \end{bmatrix} + \boldsymbol{b}^c) \tag{8}$$

$$\boldsymbol{c}_j^{c_i} = \boldsymbol{f}_j^{c_i} \odot \boldsymbol{c}^{c_{i-1}} + \boldsymbol{i}_j^{c_i} \odot \tilde{\boldsymbol{c}}_j^{c_i} \tag{9}$$

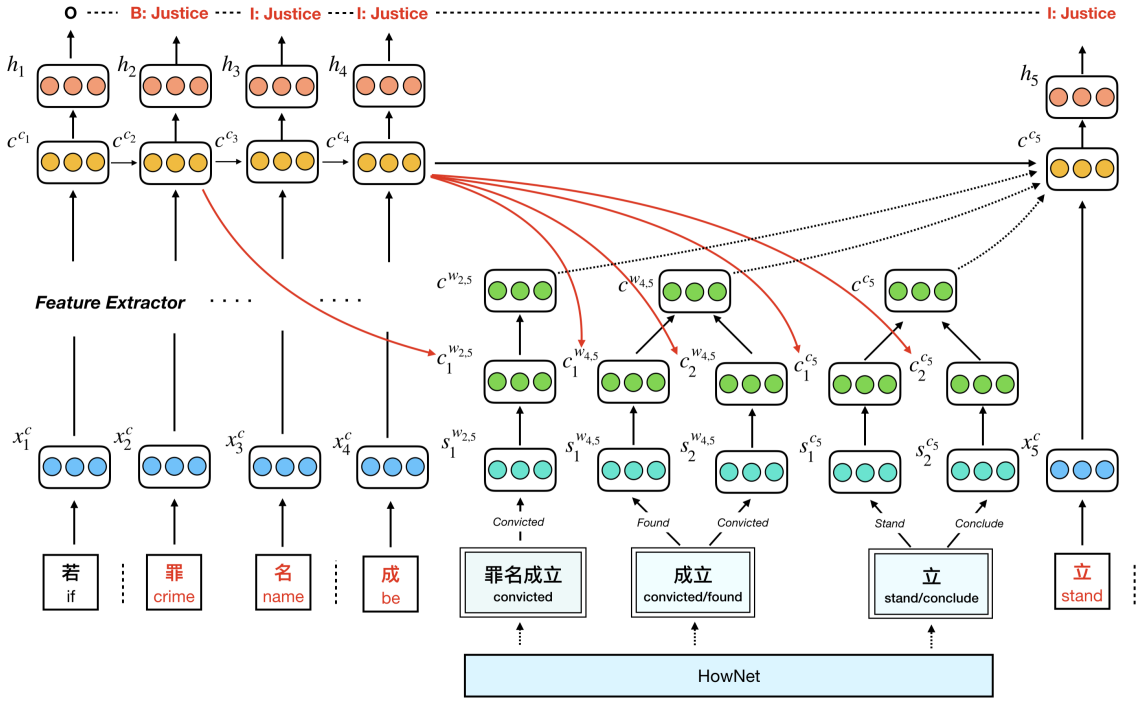



Figure 3: The structure of **Trigger-Aware Feature Extractor**, the input of the example is a part of the sentence "若罪名成立，他将被逮捕" (If convicted, he will be arrested) . In this case, "罪名成立" (convicted) is a trigger with event type *Justice: Sentence*. "成立" (convicted/found) and "立" (stand/conclude) are polysemous words. To keep the figure concise, we (1) only show two senses for each polysemous word; (2) only show the forward direction.

where $\boldsymbol{c}_j^{c_i}$ is the cell state of the $j$th sense of the $i$th character, $\boldsymbol{c}^{c_{i-1}}$ is the final cell state of the $i-1$th character. In order to obtain the cell state of the character, an additional gate is used:

$$\boldsymbol{g}_j^{c_i} = \sigma(\boldsymbol{W}^\top \begin{bmatrix} \boldsymbol{x}_i^c \\ \boldsymbol{c}_j^{c_i} \end{bmatrix} + \boldsymbol{b}) \tag{10}$$

Then all the senses should be dynamically integrated into the temporary cell state:

$$\boldsymbol{c}^{*c_i} = \sum_j^K \boldsymbol{\alpha}_j^{c_i} \odot \boldsymbol{c}_j^{c_i} \tag{11}$$

where $\boldsymbol{\alpha}_j^{c_i}$ is the character sense gate after normalization:

$$\boldsymbol{\alpha}_j^{c_i} = \frac{exp(\boldsymbol{g}_j^{c_i})}{\sum_k^K exp(\boldsymbol{g}_k^{c_i})} \tag{12}$$

Eq.11 obtains the temporary cell state of the character $c^{*c_i}$ by incorporating all the senses information of the character. However, word-level information needs to be considered as well. As mentioned in 2.1, $\boldsymbol{s}_j^{w_{b,e}}$ is the embedding for the $j$th sense of the word out $w_{b,e}$. Similar to characters, extra LSTMCell is used to calculate the cell state of each word that matches the lexicon $\mathbb{D}$.

$$\begin{bmatrix} \boldsymbol{i}_j^{w_{b,e}} \\ \boldsymbol{f}_j^{w_{b,e}} \\ \boldsymbol{c}_j^{w_{b,e}} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ tanh \end{bmatrix} (\boldsymbol{W}^{c\top} \begin{bmatrix} \boldsymbol{s}_j^{w_{b,e}} \\ \boldsymbol{h}^{c_{b-1}} \end{bmatrix} + \boldsymbol{b}^c) \tag{13}$$

$$\boldsymbol{c}_j^{w_{b,e}} = \boldsymbol{f}_j^{w_{b,e}} \odot \boldsymbol{c}^{c_{b-1}} + \boldsymbol{i}_j^{w_{b,e}} \odot \tilde{\boldsymbol{c}}_j^{w_{b,e}} \tag{14}$$

Similar to Eq.11, the cell state of the word could be computed by incorporating all the cells of senses.

$$\boldsymbol{g}_j^{w_{b,e}} = \sigma(\boldsymbol{W}^\top \begin{bmatrix} \boldsymbol{x}_b^c \\ \boldsymbol{c}_j^{w_{b,e}} \end{bmatrix} + \boldsymbol{b}) \tag{15}$$

$$\boldsymbol{c}^{w_{b,e}} = \sum_j^K \boldsymbol{\alpha}_j^{w_{b,e}} \odot \boldsymbol{c}_j^{w_{b,e}} \tag{16}$$

where $\boldsymbol{\alpha}_j^{w_{b,e}}$ is the word sense gate after normalization:

$$\boldsymbol{\alpha}_j^{w_{b,e}} = \frac{exp(\boldsymbol{g}_j^{w_{b,e}})}{\sum_k^K exp(\boldsymbol{g}_k^{w_{b,e}})} \tag{17}$$

For a character $c_i$, the temporary cell state $\boldsymbol{c}^{*c_i}$ that contains sense information is calculated by Eq.11. Moreover, we could calculate all the cell states of words that end in the index $i$ by Eq.16, which are represented as $\{\boldsymbol{c}^{w_{b,i}}|b \in [1,i], w_{b,i} \in \mathbb{D}\}$. In order to ensure the corresponding information could flow into the final cell state of $c_i$, an extra gate $\boldsymbol{g}^m_{b,i}$ is used to merge character and word cells:

$$\boldsymbol{g}^m_{b,i} = \sigma(\boldsymbol{W}^{l\top} \begin{bmatrix} \boldsymbol{x}^c_i \\ \boldsymbol{c}^w_{b,i} \end{bmatrix} + \boldsymbol{b}^l) \tag{18}$$

and the computation of the final cell state of the character $\boldsymbol{c}^{c_i}$ is:

$$\boldsymbol{c}^{c_i} = \sum_{b \in \{b'|w^d_{b',i} \in \mathbb{D}\}} \boldsymbol{\alpha}^{w_{b,i}} \odot \boldsymbol{c}^{w_{b,i}} + \boldsymbol{\alpha}^{c_i} \odot \boldsymbol{c}^{*c_i} \tag{19}$$

where $\boldsymbol{\alpha}^{w_{b,i}}$ and $\boldsymbol{\alpha}^{c_i}$ are word gate and character gate after normalization. The computation is similar to Eq. 12 and Eq. 17.

Therefore, the final cell state $\boldsymbol{c}^{c_i}$ could represent the ambiguous characters and words in a dynamic manner. Similar to Eq. 7, hidden state vectors could be calculated to transmit to the sequence tagger layer.

### 2.3 Sequence Tagger

In this paper, the event detection task is regarded as a sequence tagging problem. For an input sequence $S = \{c_1, c_2, ..., c_N\}$, there is a corresponding label sequence $L = \{y_1, y_2, ..., y_N\}$. Hidden vectors $\boldsymbol{h}$ for each character obtained in 2.2 are used as the input. We use a classic CRF layer to perform the sequence tagging, thus the probability distribution is:

$$P(L|S) = \frac{exp(\sum_{i=1}^{N}(S(y_i) + T(y_{i-1}, y_i)))}{\sum_{L' \in \mathbb{C}} exp(\sum_{i=1}^{N}(S(y'_i) + T(y'_{i-1}, y'_i)))}, \tag{20}$$

where $S$ is the score function to compute the emission score from hidden vector $\boldsymbol{h_i}$ to the label $y_i$:

$$S(y_i) = \boldsymbol{W}^{y_i}_{CRF} \boldsymbol{h}_i + b^{y_i}_{CRF}. \tag{21}$$

$\boldsymbol{W}^{y_i}_{CRF}$ and $b^{y_i}_{CRF}$ are learned parameters specific to $y_i$. And in Eq. 20, $T$ is the transition function to compute the transition score from $y_{i-1}$ to $y_i$. $\mathbb{C}$ contains all the possible label sequences on sequence $S$ and $L'$ is a random label sequence in $\mathbb{C}$.

We use standard Viterbi (Viterbi, 1967) algorithm as a decoder to decode the highest scored label sequence. The loss function of our model is log-likelihood in sentence-level.

$$Loss = \sum_{i=1}^{M} log(P(L_i|S_i)) \tag{22}$$

where $M$ is the number of sentences, $L_i$ is the correct label for the sentence $S_i$.

## 3 Experiments

### 3.1 Datasets and Experimental Settings

**Datasets.** In this paper, we conduct a series of experiments on two real-world datasets: ACE 2005 Chinese dataset (ACE2005) and TAC KBP 2017 Event Nugget Detection Evaluation dataset (KBP2017). For better comparison, we use the same data split as previous works (Chen and Ji, 2009; Zeng et al., 2016; Feng et al., 2018; Lin et al., 2018). Specifically, the ACE2005 (LDC2006T06) contains 697 articles, with 569 articles for training, 64 for validating and the rest 64 for testing. For KBP2017 Chinese dataset (LDC2017E55), we follow the same setup as Lin et al. (2018), using 506/20/167 documents as training/development/test set respectively.

**Evaluation Metrics.** Standard micro-averaged Precision, Recall and F1 are used as evaluation metrics. For ACE2005 the computation is the same as Chen and Ji (2009). To remain rigorous, we use the official evaluation toolkit [1] to perform the metrics for KBP2017.

**Hyper-Parameter Settings.** We tune the parameters of our models by grid searching on the validation dataset. Adam (Kingma and Ba, 2014) with a learning rate decay is utilized as the optimizer. The embedding sizes of characters and senses are all 50. To avoid overfitting, Dropout mechanism (Srivastava et al., 2014) is used in the system, and the dropout rate is set to 0.5. We select the best models by early stopping using the F1 results on the validation dataset. Because of the limited influence, we follow empirical settings for other hyper-parameters.

### 3.2 Overall Results

In this section, we compare our model with previous state-of-the-art methods. The proposed models are as follows:

[1] github.com/hunterhector/EvmEval

| Model | | ACE2005 | | | | | | KBP2017 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Trigger Identification | | | Trigger Classification | | | Trigger Identification | | | Trigger Classification | | |
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| Char | DMCNN | 60.10 | 61.60 | 60.90 | 57.10 | 58.50 | 57.80 | 53.67 | 49.92 | 51.73 | 50.03 | 46.53 | 48.22 |
| | C-BiLSTM* | 65.60 | 66.70 | 66.10 | 60.00 | 60.90 | 60.40 | - | - | - | - | - | - |
| | HBTNGMA | 41.67 | 59.29 | 48.94 | 38.74 | 55.13 | 45.50 | 40.52 | 46.76 | 43.41 | 35.93 | 41.47 | 38.50 |
| Word | DMCNN | 66.60 | 63.60 | 65.10 | 61.60 | 58.80 | 60.20 | 60.43 | 51.64 | 55.69 | 54.81 | 46.84 | 50.51 |
| | HNN* | **74.20** | 63.10 | 68.20 | **77.10** | 53.10 | 63.00 | - | - | - | - | - | - |
| | HBTNGMA | 54.29 | 62.82 | 58.25 | 49.86 | 57.69 | 53.49 | 46.92 | 53.57 | 50.02 | 37.54 | 42.86 | 40.03 |
| Feature | Rich-C* | 62.20 | 71.90 | 66.70 | 58.90 | 68.10 | 63.20 | - | - | - | - | - | - |
| | KBP2017 Best* | - | - | - | - | - | - | **67.76** | 45.92 | 54.74 | **62.69** | 42.48 | 50.64 |
| Hibird | NPN | 70.63 | 64.74 | 67.56 | 67.13 | 61.54 | 64.21 | 58.03 | **59.91** | 58.96 | 52.04 | 53.73 | 52.87 |
| | **TLNN (Ours)** | 67.34 | **74.68** | **70.82** | 64.45 | **71.47** | **67.78** | 65.93 | 59.07 | **62.31** | 60.72 | **54.41** | **57.39** |

Table 2: Overall results of proposed methods and TLNN on ACE2005 and KBP2017. * indicates the results adapted from the original paper. For KBP2017, "Trigger Identification" and "Trigger Classification" correspond to the "Span" and "Type" metrics in the official evaluation.

**DMCNN** (Chen et al., 2015) put forward a dynamic Multi-pooling CNN as a sentence-level feature extractor. Moreover, we add a classifier to DMCNN using IOB encoding.

**C-BiLSTM** (Zeng et al., 2016) put forward the Convolutional Bi-LSTM model for the event detection task.

**HNN** (Feng et al., 2018) designed a Hybrid Neural Network model which combines CNN with Bi-LSTM.

**HBTNGMA** (Chen et al., 2018) put forward a Hierarchical and Bias Tagging Networks with Gated Multi-level Attention Mechanisms to integrate sentence-level and document-level information collectively.

**NPN** (Lin et al., 2018) proposed a comprehensive model by automatically learning the inner compositional structures of triggers to solve the trigger mismatch problem.

The results of all the models are shown in Table 2. From the results, we can observe that:

(1) Both for ACE2005 and KBP2017, TLNN outperform other proposed models significantly, achieving the best results on two datasets. This demonstrates that the trigger-aware lattice structure could enhance the accuracy of locating triggers. Further, thanks to the usage of sense-level information, triggers could be more precisely classified into correct event types.

(2) On the TI stage, TLNN gives the best performance. By linking shortcut paths of all word candidates with the current character, the model could effectively exploit both character and word information, and then alleviates the issue of trigger-word mismatch.

| Model | ACE2005 | | KBP2017 | |
|---|---|---|---|---|
| | TI | TC | TI | TC |
| Word baseline | 61.13 | 57.81 | 56.68 | 50.98 |
| +char CNN | 61.11 | 57.52 | 57.14 | 51.66 |
| +char LSTM | 61.51 | 58.15 | 56.14 | 50.46 |
| Char baseline | 64.97 | 61.78 | 53.95 | 49.63 |
| +bigram | 66.89 | 62.95 | 57.01 | 52.71 |
| +softword | 67.79 | 64.11 | 60.68 | 54.57 |
| +softword+bigram | 68.22 | 64.23 | 60.42 | 54.85 |
| **TLNN** | **70.82** | **67.78** | **62.31** | **57.39** |

Table 3: F1-score of Word-based and Character-based baselines and TLNN on ACE2005 and KBP2017.

(3) On the TC stage, TLNN still maintain its advantages. The results indicate that the linguistic knowledge of HowNet and the unique structure to dynamically utilize sense-level information could enhance the performance on the TC stage. More located triggers could be classified into correct event types by considering the ambiguity of triggers.

### 3.3 Effect of Trigger-aware Feature Extractor

In this section, we design a set of experiments to explore the effect of the trigger-aware feature extractor. We implement strong character-based and word-based baselines by replacing the trigger-aware lattice LSTM with the standard Bi-LSTM.

For word-based baselines, the input is segmented into word sequences firstly. Furthermore, we implement extra CNN and LSTM to learn character-level features as additional modules. For character-based baselines, the basic units of the

| Model | ACE2005 | | KBP2017 | |
|---|---|---|---|---|
| | Match | Mismatch | Match | Mismatch |
| Char Baseline | 65.58 | 63.89 | 55.65 | 42.76 |
| Word Baseline | 66.30 | 2.78 | 64.43 | 17.63 |
| NPN | 65.94 | 55.56 | 56.47 | 41.53 |
| **TLNN** | **69.93** | **72.22** | **65.52** | **48.56** |

Table 4: Recall rates of two trigger-word match splits of two datasets on Trigger Identification task.

input sequence are characters. Then we enhance the character representation by adding external word-level features including bigram and softword (word in which the current character is located). Hence, both baselines could collectively utilize character and word information.

As shown in Table 3, experiments of two types of baselines and our model are conducted on ACE2005 and KBP2017. For the word baseline, although adding character-level features can improve the performance, the effects are relative limited. For the char baseline, it gains considerable improvements when word-level features are taken into account. The results of baselines indicate that integrating different level of information is an effective strategy to improve the performance of models. Compared with baselines, the TLNN achieves the best F1-score compared to all the baselines on both datasets, showing remarkable superiority and robustness. The results show that by dynamically combining multi-grained information, the trigger-aware feature extractor could effectively explore deeper semantic features than the feature-based strategies used in baselines.

### 3.4 Influence of Trigger Mismatch

In order to explore the influence of trigger mismatch problem, we split the test data of ACE2005 and KBP2017 into two types: match and mismatch. Table 1 shows the proportion of word-trigger match and mismatch on two datasets.

The recall of different methods of each split on Trigger Identification task is shown in Table 4. We can observe that:

(1) The result indicates that the word-trigger mismatch problem could severely impact the performance of the task. All approaches except ours give lower recall rates in the trigger-mismatch part than in the trigger-match part. In contrast, our model could robustly address the word-trigger mismatch problem, reaching the best results on both parts of the two datasets.

| Model | ACE2005 | | KBP2017 | |
|---|---|---|---|---|
| | **TI** | **TC** | **TI** | **TC** |
| NPN | 67.56 | 64.21 | 58.96 | 52.87 |
| TLNN | **70.82** | **67.78** | **62.31** | **57.39** |
| - w/o Sense info | 68.65 | 65.83 | 61.17 | 56.55 |

Table 5: F1-score of NPN, TLNN and TLNN - w/o Sense info on ACE2005 and KBP2017.

| Model | ACE2005 | | KBP2017 | |
|---|---|---|---|---|
| | Poly | Mono | Poly | Mono |
| NPN | 66.27 | 61.08 | 51.15 | 48.63 |
| TLNN | **69.11** | 62.56 | **58.61** | **56.56** |
| - w/o Sense info | 67.53 | **62.89** | 56.01 | 55.96 |

Table 6: F1-score of two splits of two datasets on Trigger Classification task. The splits are based on the polysemy of triggers. "Poly" and "Mono" correspond to polysemous and monosemous trigger splits.

(2) To a certain extent, the NPN model could alleviate the problem by utilizing hybrid representation learning and nugget generator in a fix-sized window. However, the mechanism is still not flexible and robust to integrate character and word information.

(3) The word-based baseline is most severely affected by the trigger-word mismatch problem. This phenomenon is explainable because if one trigger could not be segmented as a specific word in the preprocessing stage, it is impossible to be located correctly.

### 3.5 Influence of Trigger Polysemy

In this section, we mainly focus on the influence of polysemous triggers. We select NPN model for comparison. And we implement a version of TLNN without sense information, which is denoted as **TLNN - w/o Sense info** in Table 5 and Table 6.

Empirical results in Table 5 show the overall performance on ACE2005 and KBP2017. We can observe that the TLNN is weakened by removing sense information, which indicates the effectiveness of the usage of sense-level information. Even without sense information, our model could still outperform the NPN model on both two datasets.

To further explore and analyze the effect of word sense information, we split the KBP2017 dataset into two parts based on the polysemy of triggers and their contexts. The F1-score of each split is shown in Table 6, in which the TLNN yields the best results on both "Poly" parts. With-

| Sentence 1 | Word baseline | NPN | TLNN | Answer |
|---|---|---|---|---|
| 立刻/**抗**敌援友<br>**Resist** the enemies and aid the allies at once | (抗敌援友,Attack) | (刻抗,Attack) | (抗,Attack) | (抗,Attack) |
| **Sentence 2** | **NPN** | **TLNN - Sense info** | **TLNN** | **Answer** |
| **送**/他/一笔/赴/欧洲/的/旅费<br>**Send** him money for European tourism | (送,TransferPerson) | (送,TransferPerson) | (送,TransferMoney) | (送,TransferMoney) |

Table 7: Two model prediction examples. The first sentence is an example of trigger-word mismatch, while the second one is about polysemous triggers. For each prediction result, (A,B) indicates that A is a trigger with event type B.

out sense information, TLNN - w/o sense info could give comparable F1-scores with TLNN on the "Mono" parts. The results indicate that the trigger-aware feature extractor could dynamically learn all the senses of characters and words, gaining significant improvements under the condition of polysemy.

### 3.6 Case Study

Table 7 shows two examples comparing the TLNN model with other ED methods. The former example is about trigger-word mismatch, in which the correct trigger "抗"(resist) is part of the idiom word "抗敌援友" (resist the enemies and aid the allies). In this case, the word baseline gives the whole word "抗敌援友" as prediction because it is impossible for word-based methods to detect part-of-word triggers. Additionally, the NPN model recognizes a non-existent word "刻抗". The reason is that the NPN enumerates the combinations of all characters within a window as trigger candidates, which is likely to generate invalid words. In contrast, our model detects the event trigger "抗" accurately.

In the latter example, the trigger "送"(send) is a polysemous word with two different meanings: "送行"(see him off) and "送钱"(give him money). Without considering multiple word senses of polysemes, the NPN and TLNN (w/o Sense info) classify trigger "送" into wrong event type *TransferPerson*. On the contrary, the TLNN can dynamically select word sense for polysemous triggers by utilizing context information. Thus the correct event type *TransferMoney* is predicted.

## 4 Related Work

Event Detection (ED) is a crucial subtask in Event Extraction task. Feature-based methods (Ahn, 2006; Ji and Grishman, 2008; Liao and Grishman, 2010; Huang and Riloff, 2012; Patwardhan and Riloff, 2009; McClosky et al., 2011) were widely used in the ED task, but these traditional methods are heavily rely on the manual features, limiting the scalability and robustness.

Recent developments in deep learning have led to a renewed interest in neural event detection. Neural networks can automatically learn features of the input sequence and conduct token-level classification. CNN-based models are the seminal neural network models in ED (Nguyen and Grishman, 2015; Chen et al., 2015; Nguyen and Grishman, 2016). However, these models can only capture the local context features in a fixed size window. Some approaches design comprehensive models to explore the interdependency among trigger words (Chen et al., 2018; Feng et al., 2018). To further improve the ED task, some joint models are designed (Nguyen et al., 2016; Lu and Nguyen, 2018; Yang and Mitchell, 2016). These methods have achieved great success in English datasets.

However, in languages without delimiters, such as Chinese, the mismatch of word-trigger become significantly severe. Some feature-based methods are proposed to solve the problem (Chen and Ji, 2009; Qin et al., 2010; Li and Zhou, 2012), but they heavily rely on the hand-crafted features. Lin et al. (2018) proposes NPN, a neural network based method to address the issue. However, the mechanism of NPNs limits the scope of trigger candidates within a fix-sized window, which will cause two problems in the progress. First, the NPNs still cannot take all the possible trigger candidates into account, leading to meaningless computation. Furthermore, the overlap of triggers is serious in NPNs. Lattice-based models were used in other fields to combine character and word information (Li et al., 2019; Zhang and Yang, 2018; Yang et al., 2018). Mainstream methods also suffer from the problem of trigger polysemy. Lu and Nguyen (2018) proposes a multi-task learning model which uses word sense disambiguation to alleviate the effect of the trigger polysemy

problem. But in this work, word disambiguation datasets are necessary. In contrast, our model can solve both word-trigger mismatch and trigger polysemy problems at the same time.

## 5 Conclusion and Future Work

We propose a novel framework TLNN for event detection, which can simultaneously address the problems of trigger-word mismatch and polysemous triggers. With the hierarchical representation learning and the trigger-aware feature extractor, TLNN efficaciously exploits multi-grained information and learn deep semantic features. Sets of experiments on two real-world datasets show that TLNN could efficiently address the two issues and yield better empirical results than a variety of neural network models.

In future work, we will conduct experiments on more languages with and without explicit word delimiters. In addition, we will try developing a dynamic mechanism to selectively consider the sense-level information rather than take all the senses of characters and words into account.

## 6 Acknowledgement

This work is supported by National Natural Science Foundation of China (Grant No. 61773229, 61572273, 61661146007), National Key Research and Development Program of China (No. 2018YFB1004503), Basic Scientific Research Program of Shenzhen City (Grant No. JCYJ20160331184440545), and Overseas Cooperation Research Fund of Graduate School at Shenzhen, Tsinghua Univeristy (Grant No. HW2018002), Shenzhen Giiso Information Technology Co. Ltd. Finally, we would like to thank the anonymous reviewers for their helpful feedback and suggestions.

## References

David Ahn. 2006. The stages of event extraction. In *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*, pages 1–8.

Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 167–176.

Yubo Chen, Hang Yang, Kang Liu, Jun Zhao, and Yantao Jia. 2018. Collective event detection via a hierarchical and bias tagging networks with gated multi-level attention mechanisms. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1267–1276.

Zheng Chen and Heng Ji. 2009. Language specific issue and feature exploration in chinese event extraction. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 209–212.

Zhendong Dong and Qiang Dong. 2003. Hownet-a hybrid language and knowledge resource. In *Proceedings of NLP-KE*.

Xiaocheng Feng, Bing Qin, and Ting Liu. 2018. A language-independent neural network for event detection. *Science China Information Sciences*, 61(9):092106.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Ruihong Huang and Ellen Riloff. 2012. Modeling textual cohesion for event extraction. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Heng Ji and Ralph Grishman. 2008. Refining event extraction through cross-document inference. *Proceedings of ACL-08: HLT*, pages 254–262.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Peifeng Li and Guodong Zhou. 2012. Employing morphological structures and sememes for chinese event extraction. *Proceedings of COLING 2012*, pages 1619–1634.

Ziran Li, Ning Ding, Zhiyuan Liu, Haitao Zheng, and Ying Shen. 2019. Chinese relation extraction with multi-grained information and external linguistic knowledge. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 4377–4386.

Shasha Liao and Ralph Grishman. 2010. Using document level cross-event inference to improve event extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 789–797. Association for Computational Linguistics.

Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. 2018. Nugget proposal networks for chinese event detection. *arXiv preprint arXiv:1805.00249*.

Weiyi Lu and Thien Huu Nguyen. 2018. Similar but not the same: Word sense disambiguation improves event detection via neural representation matching.

In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4822–4828.

David McClosky, Mihai Surdeanu, and Christopher D Manning. 2011. Event extraction as dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1626–1635. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016. Joint event extraction via recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 300–309.

Thien Huu Nguyen and Ralph Grishman. 2015. Event detection and domain adaptation with convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 365–371.

Thien Huu Nguyen and Ralph Grishman. 2016. Modeling skip-grams for event detection with convolutional neural networks. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 886–891, Austin, Texas. Association for Computational Linguistics.

Yilin Niu, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2017. Improved word representation learning with sememes. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2049–2058, Vancouver, Canada. Association for Computational Linguistics.

Siddharth Patwardhan and Ellen Riloff. 2009. A unified model of phrasal and sentential evidence for information extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 151–160. Association for Computational Linguistics.

Bing Qin, Yanyan Zhao, Xiao Ding, Ting Liu, and Guofu Zhai. 2010. Event type recognition based on trigger expansion. *Tsinghua Science and Technology*, 15(3):251–258.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269.

Bishan Yang and Tom Mitchell. 2016. Joint extraction of events and entities within a document context. *arXiv preprint arXiv:1609.03632*.

Jie Yang, Yue Zhang, and Shuailong Liang. 2018. Subword encoding in lattice lstm for chinese word segmentation. *arXiv preprint arXiv:1810.12594*.

Ying Zeng, Honghui Yang, Yansong Feng, Zheng Wang, and Dongyan Zhao. 2016. A convolution bilstm neural network model for chinese event extraction. In *Natural Language Understanding and Intelligent Applications*, pages 275–287. Springer.

Yue Zhang and Jie Yang. 2018. Chinese ner using lattice lstm. *arXiv preprint arXiv:1805.02023*.