Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A network security entity recognition method based on feature template and CNN-BiLSTM-CRF*

Ya QIN[†1,2], Guo-wei SHEN[†‡1,2], Wen-bo ZHAO[1,2], Yan-ping CHEN[1,2], Miao YU[3], Xin JIN[4]

[1]*College of Computer Science and Technology, Guizhou University, Guiyang 550025, China*

[2]*Guizhou Provincial Key Laboratory of Public Big Data, Guiyang 550025, China*

[3]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China*

[4]*National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029, China*

[†]E-mail: qyamail@163.com; gwshen@gzu.edu.cn

Received Aug. 31, 2018; Revision accepted Mar. 11, 2019; Crosschecked June 11, 2019

Abstract: By network security threat intelligence analysis based on a security knowledge graph (SKG), multi-source threat intelligence data can be analyzed in a fine-grained manner. This has received extensive attention. It is difficult for traditional named entity recognition methods to identify mixed security entities in Chinese and English in the field of network security, and there are difficulties in accurately identifying network security entities because of insufficient features extracted. In this paper, we propose a novel FT-CNN-BiLSTM-CRF security entity recognition method based on a neural network CNN-BiLSTM-CRF model combined with a feature template (FT). The feature template is used to extract local context features, and a neural network model is used to automatically extract character features and text global features. Experimental results showed that our method can achieve an F-score of 86% on a large-scale network security dataset and outperforms other methods.

Key words: Network security entity; Security knowledge graph (SKG); Entity recognition; Feature template; Neural network
https://doi.org/10.1631/FITEE.1800520 **CLC number:** TP393.08

1 Introduction

With the development of information technology and the complex network environment, network information security has become the focus of attention around the world. More and more information leaks, infiltration attacks, and other information security threats seriously threaten citizens, businesses, and countries. To ensure the security of cyberspace, countries and companies deploy security services such as firewall, intrusion detection, and intrusion protection at key locations to monitor network security threats. These services produce much network security data, such as alarm information and monitoring logs. Due to the fragmentation and magnanimity of cyber security data, there is a lack of certain links among these data, such that network security personnel cannot obtain or integrate them. Therefore, we build a network security knowledge graph (SKG) (Liu et al., 2016) to effectively correlate, analyze, and mine a massive amount of security data. Through SKG, network security personnel can more intuitively understand network security threat intelligence (Li, 2016) and security posture to discover complex network attack patterns. The main techniques for constructing a network SKG include security entity

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61802081), the Guizhou Provincial Natural Science Foundation, China (No. 20161052), the Guizhou Provincial Public Big Data Key Laboratory Open Project, China (No. 2017BDKFJJ024), the Guizhou University Doctoral Fund, China (No. 201526), and the Major Scientific and Technological Special Project of Guizhou Province, China (No. 20183001)

ORCID: Ya QIN, https://orcid.org/0000-0002-2685-3445

recognition, relation extraction, and attribute extraction, among which security entity recognition is the foundation of a network SKG.

Security entity recognition is a specific domain entity recognition in named entity recognition (NER) (Zhang et al., 2005; Finkel and Manning, 2009). The main task is to identify different types of security entities in network security text data, such as users, malicious programs, hacking organizations, and vulnerabilities. Its purpose is to recognize and classify professional vocabularies in the network security domain, and NER is mainly to recognize person, location, and organization. NER is originally a rule-based recognition method. Domain experts and linguists make effective rules to identify entities. The formulated rules cannot be adapted to other domains because of poor transplantability. In view of the problem, researchers use machine learning methods with artificial features to identify named entities. The commonly used machine learning methods include hidden Markov models (HMMs) (Rabiner et al., 1989; Yu et al., 2006), maximum entropy models (MEMs) (Koeling, 2000), and conditional random fields (CRFs) (Lafferty et al., 2001; Luo et al., 2015). These methods require various NER tools and knowledge resources to design effective features and to set a feature template. Therefore, the selection of feature templates directly affects the performance of NER (Joshi et al., 2013; Qiu et al., 2013). Selection is still a labor-intensive and skill-dependent task. To reduce the complexity of artificial features, researchers used word embedding in deep learning algorithms to replace the artificial features (Tang et al., 2014). This solves the problem of high cost of artificial features and improves the performance of NER.

In recent years, the deep learning method (Collobert and Weston, 2008) has achieved good results in the NER common domain. Compared with a machine learning method or rule-based method, the deep learning method has stronger generalization ability and less reliance on artificial features. Therefore, several neural network architectures have been proposed for the NER task in the common domain. Collobert et al. (2011) used a neural network for the first time and it performed quite well in the NER common domain. After that, using a neural network to extract features has received extensive attention. At present, a long-short term memory (LSTM) (Hochreiter and Schmidhuber, 1997; Gers et al., 2000; Hammerton, 2003; Peng and Dredze, 2015) unit with the forget gate allows highly non-trivial long-distance dependencies to be easily learned, and this has shown great success in NER. Because LSTM cannot consider future context information, researchers propose the bi-directional LSTM (BiLSTM) model with word embedding (Feng et al., 2018) for NER, which can effectively take into account an amount of context information on the future. The BiLSTM with a CRF model (Huang et al., 2015; Dong et al., 2016; Lample et al., 2016) has exhibited better results. A convolutional neural network (CNN) (Dos Santos and Guimarães, 2015) has also been used for NER to encode character information of each word into its character representation, and Chiu and Nichols (2015) and Ma and Hovy (2016) have successfully employed CNN to extract character features on CoNLL 2003 data sets. They have obtained a good *F*-score for NER. Although neural networks have performed well in the NER common domain, there are still many problems in network security entity recognition in the network security domain.

At present, compared with common named entity recognition, network security entity recognition faces mainly the following challenges:

1. The types of security entities are diverse, numerous, and there are constantly unregistered words that appear as new security entities, such as new malware, vulnerabilities, and patches.

2. In different scenarios, network security entities have a problem of fuzzy classification. The boundaries between different types of security entities are not clear, and software names often appear in organization names. For example, Oracle represents both software and an organization.

3. Network security entities have different structures. A lot of software and vulnerabilities are named both in Chinese and English. There are also many problems between security entities such as a large number of nesting and abnormal abbreviations, so it is difficult to identify them.

To solve the above problems, we propose a novel CNN-BiLSTM-CRF model with a feature template. To evaluate the proposed model, we construct a large-scale dataset. Experimental results on our dataset show that our model can achieve better performance than previous models.

The main contributions of our work can be summarized as follows:

1. We present a novel neural network CNN-BiLSTM-CRF model with feature template (FT-CNN-BiLSTM-CRF) for network security entity recognition.

2. For mixed network security entities in Chinese and English, we manually create a few features into the feature template to extract local context features of words based on the neural network model. Results show that the combination of the feature template and neural network is effective.

3. We evaluate different network security entity recognition models over the network security dataset. The experimental results indicate that our model can improve the performance of recognition and outperforms other models.

2 Model architecture

In this section, we describe the overall architecture of our model. We introduce the components of our security entity recognition model one by one.

2.1 Feature template

The feature template (FT) is designed based on the selected features, and is a comprehensive consideration of the contextual information of the data and the location of the information. This study is aimed at security entity recognition in the network security domain, and the composition of the security entity has no regularity. If we depend only on the structure of the security entity itself and the composition of the sentence, it is difficult to achieve satisfactory results. We add the local context information of words to our model through the FT, where the context information refers to the "observation window" of the current value and a few words before and after it. The larger the observation window, the more the context information included in the template. However, too large an observation window reduces the efficiency of the model, resulting in the over-fitting phenomenon. If the observation window is too small, as less information is used, the recognition efficiency of the model is also reduced. Therefore, it is of vital importance to select an appropriate size of the observation window of the FT.

FT includes atomic templates and composite templates. The atomic templates are selected in this study, and the features developed in the template are atomic features. The template developed for the context features is as follows: $w[-2, 0]$, $w[-1, 0]$, $w[0, 0]$, $w[1, 0]$, $w[2, 0]$. The first number in square brackets indicates its relative position to the current character, and the second number indicates the column of the selected feature, including words, parts of speech, and other features. Table 1 shows an example of atomic characteristics.

Table 1 An example of atomic features

	w	y	
−2	找	*O*	
−1	到	*O*	
0	雅	B-ORG	Current token
1	虎	I-ORG	
2	的	O	

w and y represent the current word and the label of the word, respectively

Given that there are characteristics of mixture of Chinese and English in network security data, we extract the context feature of the current word to make FT. FT is implemented by imitating the method of the CRF++ tool. We define a set of feature functions $f_k(y_{i-1}, y_i, w, i)$ which are binary functions, where y_i represents the current mark, y_{i-1} represents the next mark, i is the current position, and w is the current word. In general, the value of the feature function is 1 or 0. When the feature condition is satisfied, the value is 1; otherwise, it is 0. The feature functions are summed at various positions i:

$$f_k(y,w)=\sum_{i=1}^{n} f_k(y_{i-1},y_i,w,i), \tag{1}$$

where $f_k(y, w)$ represents the sum of the feature functions at each position i. We assign a weight λ_k to the feature functions, and use $\boldsymbol{\lambda}$ to denote the weight vector of the feature function:

$$\boldsymbol{\lambda}=(\lambda_1,\lambda_2,\cdots,\lambda_k)^{\mathrm{T}}. \tag{2}$$

Then we convert the feature extracted by the feature function into a feature vector:

$$\boldsymbol{F}(y,w)=\left(f_1(y,w),f_2(y,w),\cdots,f_k(y,w)\right)^{\mathrm{T}}, \tag{3}$$

where $\boldsymbol{F}(y, w)$ represents the global feature vector. If a feature function is activated, then its weight $\boldsymbol{\lambda}$ will be added to an accumulated value, i.e., the score value of features, score′. The weight summation process can be viewed as multiplying a weight vector by a feature vector:

$$\text{score}' = \boldsymbol{\lambda} \cdot \boldsymbol{F}(y, w). \tag{4}$$

2.2 Character embedding

Character features in an entity name contain rich structural information on the entity. These features (such as character n-grams, prefixes and suffixes) are commonly employed in current security entity recognition methods. In recent years, character vector (Passos et al., 2014) in NER based on a neural network model usually has a higher dimension, resulting in a large number of hyperparameters in the model. However, in practice, the supervised corpora with annotations are usually insufficient, so it is difficult to learn the accurate hyperparameters. We are inspired by Dong et al. (2016) to use the pre-training character embedding method. Unlike previous traditional methods in which character features are based on hand-engineering, we use a large-scale unmarked network security corpus to learn character embedding for the training.

In recent years, several tools, such as word2vec (Mikolov et al., 2013a, 2013b) and GloVe (Pennington et al., 2014), have been widely used in NER. In our approach, we train the word2vec character embedding file from the network security corpus. The character embedding file has 16 691 characters, including Chinese characters, English characters, and some special characters (numbers, punctuation marks, etc.). Each character corresponds to a 100 dimensional embedding vector.

2.3 CNN for character-level representation

CNN (LéCun et al., 1998) is one of the most representative neural network structures in deep learning technology, and has become one of the research hotspots in many disciplines. The network security entities are mixed Chinese and English, such as “SQL 注入漏洞” and “Microsoft.” For the English entity in network security data, we use CNN to extract character-level features of the security entity. CNN is used mainly to deal with English in which words are composed of more fine-grained letters. These letters have hidden features such as prefix/ suffix features. Therefore, CNN is used to extract the character-level features of the security entity such as malicious software and vulnerabilities named in English. We set different random character vectors for different types of characters to distinguish the cases of characters, character types (letters, numbers, punctuation, special characters). For example, uppercase A and lowercase a correspond to two different sets of character vectors. Fig. 1 shows the process whereby CNN extracts the character-level feature of a word.

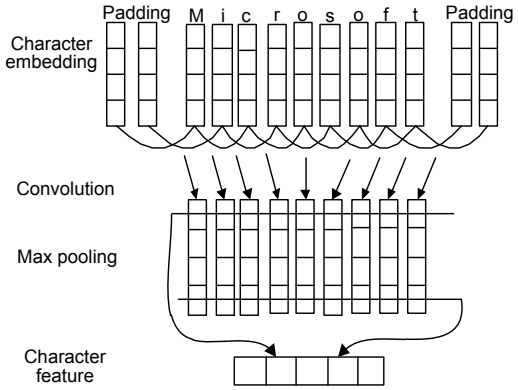

Fig. 1 The process of CNN extracting the character-level feature of the words

For an input sequence $\boldsymbol{X}=(x_1, x_2, \ldots, x_n)$ consisting of n words, each word is expressed as $\boldsymbol{x}=(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_l)$, consisting of l characters. Each word $\boldsymbol{x}$ is transformed into the corresponding character vector by querying the character lookup table C, and then character vector matrix $\boldsymbol{c}_i$ ($\boldsymbol{c}_i \in \mathbb{R}^{d \times l}$) of the word is formed, where $\boldsymbol{c}_i$ is the vector for the i^{th} character in the word. To solve the problem of different sizes of the character vector matrix due to different word lengths, we use the longest word as a baseline by padding placeholders at the left and right ends of the word. In the convolutional operation to extract character-level feature vectors, one convolutional layer is implemented between kernels $\boldsymbol{k}$ ($\boldsymbol{k} \in \mathbb{R}^{d \times m}$) of width m, and an output feature map can be obtained through an activation function $\boldsymbol{F}_j \in \mathbb{R}^{l-m+1}$. Each output map may be a combination of convolutional values of multiple input maps. Next, we apply a max pooling operation over the feature map, and take the average value or maximum value as the feature corresponding to the

filter:

$$w'_j = \max_i F_j[i]. \tag{5}$$

Finally, we obtain a representation of word w by connecting all the feature maps:

$$w' = w_1 \otimes w_2 \otimes \cdots \otimes w_l. \tag{6}$$

2.4 Bidirectional LSTM

The long short-term memory (LSTM) network (Hochreiter and Schmidhuber, 1997) as a kind of deep learning method is a specific form of recurrent neural network (RNN) (Goller and Kuchler, 1996) compared with general RNN. It can preserve redundant context information while solving long-term dependency problems. Formally, at time t, cell state $\boldsymbol{c}_t$ and hidden state $\boldsymbol{h}_t$ are updated with the following equations:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{\mathrm{xi}}\boldsymbol{x}_t + \boldsymbol{W}_{\mathrm{hi}}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{\mathrm{ci}}\boldsymbol{c}_{t-1} + \boldsymbol{b}_{\mathrm{i}}), \tag{7}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{\mathrm{xf}}\boldsymbol{x}_t + \boldsymbol{W}_{\mathrm{hf}}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{\mathrm{cf}}\boldsymbol{c}_{t-1} + \boldsymbol{b}_{\mathrm{f}}), \tag{8}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{\mathrm{xo}}\boldsymbol{x}_t + \boldsymbol{W}_{\mathrm{ho}}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{\mathrm{co}}\boldsymbol{c}_t + \boldsymbol{b}_{\mathrm{o}}), \tag{9}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \cdot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \cdot \tanh(\boldsymbol{W}_{\mathrm{xc}}\boldsymbol{x}_t + \boldsymbol{W}_{\mathrm{hc}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\mathrm{c}}), \tag{10}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \cdot \tanh(\boldsymbol{c}_t), \tag{11}$$

where $\boldsymbol{i}_t$, $\boldsymbol{f}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{c}_t$ denote the input gate, forget gate, output gate, and cell state at time t, respectively. $\boldsymbol{x}_t$ is the input vector at time t and $\boldsymbol{h}_t$ is the hidden layer vector at time t. $\sigma(\cdot)$ and $\tanh(\cdot)$ represent two different neuronal activation functions. $\boldsymbol{W}_{\mathrm{xi}}$, $\boldsymbol{W}_{\mathrm{xf}}$, $\boldsymbol{W}_{\mathrm{xo}}$, and $\boldsymbol{W}_{\mathrm{xc}}$ are the weight matrices. $\boldsymbol{b}_{\mathrm{i}}$, $\boldsymbol{b}_{\mathrm{f}}$, $\boldsymbol{b}_{\mathrm{o}}$, and $\boldsymbol{b}_{\mathrm{c}}$ are the offset vectors.

Note that LSTM takes only the past information. However, context information from the future could also be crucial. To use context information effectively, we use a bidirectional LSTM (BiLSTM) to extract features. BiLSTM computes two different hidden layer representations using the sequence (from the first word, recursively from left to right) and reverse order (recursive from the last word, recursively from right to left) for the input sequence. Then the final hidden layer representation is obtained by vector splicing. Fig. 2 shows the structure of BiLSTM.

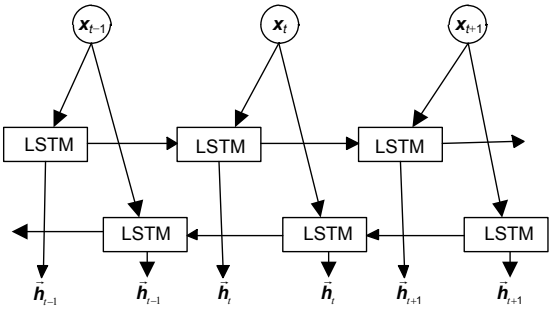

Fig. 2 BiLSTM structure

BiLSTM receives the character-level feature vector sequence (x_1, x_2, …, x_n) extracted by CNN as an input at each time state t, and then splices the eigenvector sequence $\vec{\boldsymbol{h}} = (\vec{\boldsymbol{h}}_1, \vec{\boldsymbol{h}}_2, ..., \vec{\boldsymbol{h}}_t)$ output by the forward LSTM and the eigenvector sequence $\overleftarrow{\boldsymbol{h}} = (\overleftarrow{\boldsymbol{h}}_1, \overleftarrow{\boldsymbol{h}}_2, ..., \overleftarrow{\boldsymbol{h}}_t)$ of the backward LSTM at time t to obtain a complete feature vector sequence $\overleftrightarrow{\boldsymbol{h}}_t$.

$$\overleftrightarrow{\boldsymbol{h}}_t = [\vec{\boldsymbol{h}}_t; \overleftarrow{\boldsymbol{h}}_t] \in \mathbb{R}^m. \tag{12}$$

2.5 BiLSTM-CRF model

Security entity recognition is a typical sequence labeling problem. BiLSTM is very powerful in sequence modeling and can obtain long-term contextual information. However, the tags of the security entity recognition task are not independent and have strong dependency, especially for character-based security entity recognition. For example, the B-ORG tag is followed by an I-ORG, but it cannot be an I-PER. Therefore, we access CRF after BiLSTM. We will no longer use the softmax function to calculate the probability of each tag directly as in other neural network models. Instead, we use CRF to consider the transition probabilities between tags and to calculate the probability of the entire tag sequence.

Formally, we can see from a single moment that the corresponding tag of the current character has a corresponding score. The greater the score is, the more likely the corresponding tag is. From the overall point of view of the tag sequence, there are transfer scores before and after the tag. The higher the score is, the more likely the tag transfer occurs. By adding these two scores, the tag with the highest score is the prediction result. Therefore, the preprocessing of the hidden layer is given by:

$$\boldsymbol{O}_t = \tanh(\boldsymbol{W}_{\mathrm{h}}\vec{\boldsymbol{h}}_t + \boldsymbol{b}_{\mathrm{o}}), \tag{13}$$

where $\vec{\boldsymbol{h}}_t$ is the splicing vector of the hidden layer. Its corresponding weight is $\boldsymbol{W}_{\mathrm{h}}\in\mathbb{R}^{|n|\times|m|}$ and the offset vector is $\boldsymbol{b}_{\mathrm{o}}\in\mathbb{R}^{|n|}$. $\boldsymbol{O}_t$ is the output vector of the hidden layer. $|n|$ denotes the dimension of the hidden layer. Next, the scores of all tags corresponding to character at time t are as follows:

$$\mathrm{score}'' = \boldsymbol{W}_{\mathrm{c}}\boldsymbol{O}_t + \boldsymbol{b}_{\mathrm{c}}, \tag{14}$$

where $\boldsymbol{W}_{\mathrm{c}}\in\mathbb{R}^{n\times m}$ is the weight matrix, $\boldsymbol{b}_{\mathrm{c}}\in\mathbb{R}^{n}$ is the offset vector, m denotes the number of tags, and score″ represents the score of each character corresponding to the tag. The larger the score, the more likely it is.

Transfer matrix $\boldsymbol{A}$ is set to represent the transfer score between the tags, and tags "start" and "end" are added to m tags. The total score of a tag sequence is determined by the transfer of each tag as follows:

$$S(X,y) = \sum_{i=0}^{l}\left(A_{y_i,y_{i+1}} + \mathrm{score}''(y_{i+1})\right), \tag{15}$$

where X is the input sequence, y is the prediction tag sequence, and $A_{y_i,y_{i+1}}$ is the probability of transition from tags y_i to y_{i+1}, i.e., the transfer fraction. Score″(y_{i+1}) denotes the score of the current label. y_0 and y_l are the "start" and "end" tabs, respectively, and their corresponding scores are both zero.

In the condition of input sequence X, we calculate the probability of the tag sequence y as follows:

$$p(y|X) = \mathrm{e}^{s(x,y)} \Big/ \sum_{y\in Y_X} s(X,y). \tag{16}$$

In the training process, the main goal is to maximize the probability of the correct sequence, so our model uses logarithm maximum likelihood estimation to obtain the loss function:

$$\begin{aligned}\log\left(p(y|X)\right) &= S(X,y) - \log\left(\sum_{y'\in Y_x}\mathrm{e}^{S(X,y')}\right)\\ &= S(X,y) - \log_{y'\in Y_x}\text{add}\, S(X,y').\end{aligned} \tag{17}$$

Then, we use the stochastic gradient descent learning algorithm to train parameter σ. After obtaining parameter σ, the Viterbi algorithm is used to obtain the sequence with the highest score on all sequences, and this is used as the final annotation result:

$$y^* = \arg_{y'\in Y_x}\max S(X,y'). \tag{18}$$

2.6 FT-CNN-BiLSTM-CRF model

Our security entity recognition model was inspired by Ma et al. (2016), where the feature template is combined with the model. Fig. 3 shows the architecture of our model in detail.

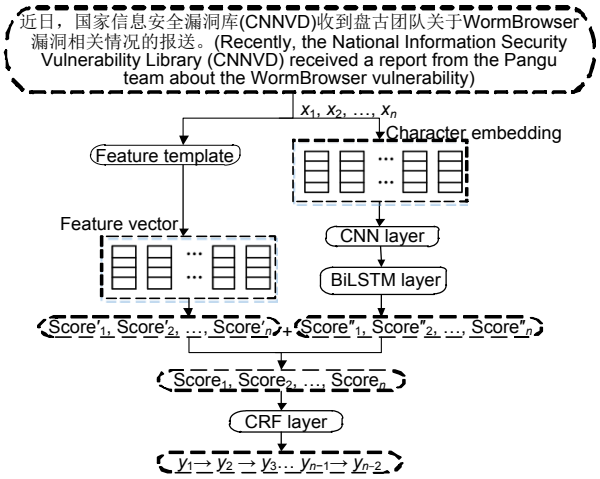

Fig. 3 Architecture of our model

CNN extracts a fixed length feature vector from character-level features in Fig. 1. These feature vectors are fed into the BiLSTM layer here

Given the Chinese and English mixed network security corpus data, we manually make an FT to extract context-based features from the input sequence, and then use pre-trained character embedding to convert the input sequence into a corresponding character vector by querying the character lookup table. For the input sequence, the character feature is computed by the CNN layer with character embedding as inputs. Then the character feature vector is fed into the BiLSTM layer to extract the information features. Finally, the information features are concatenated with the context-based feature, which is fed to the CRF layer to mark each character to obtain the best label sequence. When we calculate the tag score corresponding to the character at time t, we add the score value of the feature of the character in FT to the tag score. This can increase the score of the feature.

The higher the score, the higher the prediction accuracy:

$$\text{score} = \text{score}' + \text{score}''. \tag{19}$$

Then, as in the BiLSTM-CRF model described in Section 2.5, CRF is used to calculate the transition probability between labels and the probability of the entire tag sequence. During the training phase, the objective of the model is to maximize the log-probability of the correct tag sequence. The Viterbi algorithm is used to compute optimal tag sequences for inference.

3 Model training

3.1 Training process

In this study, our security entity recognition model based on CNN-BiLSTM-CRF with the feature template is implemented using TensorFlow. The calculation of a single model is run on the Intel® Xeon® CPU 24 cores. We select the more complex CNN-BiLSTM-CRF training algorithm with the feature template to build the security entity recognition model. The model algorithm training process is as given in Algorithm 1.

Algorithm 1 Model training

Input: $\boldsymbol{X}=(x_1, x_2, ..., x_n)$

Output: $\boldsymbol{Y}=(y_1, y_2, ..., y_n)$

```
for each epoch do
  for each batch do
    hyperparameter initialization;
    CNN model forward pass to extract the character-level feature;
    BiLSTM-CRF model forward pass and automatic learning to extract the feature:
      forward pass for forward state;
      forward pass for backward state;
    CRF forward and backward pass and calculate the global likelihood probability of the sequence;
    BiLSTM-CRF model backward pass:
      backward pass for forward state;
      backward pass for backward state;
    update hyperparameters;
  end for
end for
```

As shown in Algorithm 1, for each epoch, the model trains the entire training data into batches during the training process, and one epoch processes batches. The size of each batch of training data is determined by the hyperparameter batch_size. During model training, first the hyperparameters of the model are initialized, and then the character-level feature is extracted through the CNN model. After that, we run the BiLSTM model forward and backward pass to extract features, and calculate the output state of the model by running the CRF model. Then the wrong state can be returned from the output back to the input, which includes the backward pass for both forward and backward states of LSTM. Finally, the model hyperparameters are updated again.

3.2 Hyperparameter initialization

Our model has a large number of hyperparameters (Bergstra and Bengio, 2012) and the selection of these hyperparameters is very important. The hyperparameters of the model settings are shown in Table 2.

Table 2 Hyperparameter settings

Hyperparameter	Value
CNN window size	5
CNN filter size	30
BiLSTM hidden size	100
Learning rate	0.002
Batch size	100
Number of epochs	40
Fine tuning	True
Dropout rate	0.5
Feature template window size	5

As shown in Table 2, we explore different parameter settings, and mainly set the window size and filter size for the CNN layer. We set 100 neural units (hidden size) in the hidden layer for the BiLSTM layer, and set the size of the context feature window in the feature template. For character embedding, we fine-tune pre-trained embedding in context. In the training process, we select the Adam optimizer to optimize the training. The initial learning rate is set to be 0.002, the batch size is set to be 100, and the number of epochs is 40. To mitigate overfitting, we apply the dropout method to regularize our model, and the value of the dropout is 0.5.

4 Experimental settings

4.1 Corpus and annotation patterns

The dataset used in our experiment comes mainly from the Freebuf website and the WooYun Vulnerability Database, which includes network text data such as technology sharing, network security, and vulnerability information, totaling 15 460 pieces. We select 10 000 pieces of the original unmarked data for pre-training character embedding, and 5460 for the security entity recognition labeled corpus. There is no complete corpus in the network security domain up until now. Therefore, we need to build our own corpus, which consists of an automatically labeled part and a manually labeled part. In this study, we recognize mainly six types of security entities: person (PER), location (LOC), organization (ORG), software (SW), network relevant term (RT), and vulnerability ID (VUL_ID). For these six types of security entities, we use the BIO annotation model. B represents the start of the entity, I represents the middle of the entity, and O indicates that it is not an entity.

First, Chinese word segmentation is performed, and then the data after word segmentation is automatically labeled by matching the person database, location database, vulnerability ID database, and network related term database, labeled as B-PER, B-LOC, B-VUL_ID, and B-RT, respectively. Then we mark the labeled word segmentation data according to the character by the code. For example, B-PER and I-PER represent the first and non-first words of the person, respectively. Finally, the organization and software are annotated manually.

In this study, we use the labeled 70% cyber security corpus data as training data, 10% as validation data, 20% as testing data. The network security corpus data statistics is shown in Table 3.

4.2 Evaluation standard

In this study, we will evaluate the recognition effect of our security entity recognition model by selecting a common evaluation index system (Yang, 1999) for multi-classification problems.

We divide the test samples into real entity categories and model prediction entity categories. After model prediction, the number of security entities with a positive test category, whose real category is positive, is denoted as true positive (TP). The number of security entities with a positive prediction category, whose real category is negative, is denoted as false positive (FP). The number of security entities with a negative prediction category, whose real category is negative, is denoted as true negative (TN). The number of security entities with a negative prediction category, whose real category is positive, is denoted as false negative (FN).

Table 3 Corpus statistics

Dataset	Number		
	Training	Validation	Testing
Sentences	15 090	1989	2697
Labels	70 324	9181	16 669
Person	1139	230	249
Location	975	441	112
Organization	4059	374	1216
Software	4236	666	1794
Relevant term	59 849	7428	13 102
Vulnerability ID	66	42	196

Sentences and labels represent the total number of sentences and total number of labels in each dataset, respectively

The main evaluation indicators adopted in this study are precision (P), recall (R), F-value, and accuracy (Acc):

$$P = \mathrm{TP}/(\mathrm{TP}+\mathrm{FP})\times 100\%, \tag{20}$$

$$R = \mathrm{TP}/(\mathrm{TP}+\mathrm{FN})\times 100\%, \tag{21}$$

$$F = 2PR/(P+R)\times 100\%, \tag{22}$$

$$\mathrm{Acc} = (\mathrm{TP}+\mathrm{TN})/(\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN})\times 100\%. \tag{23}$$

4.3 Model comparison analysis

To verify the performance of our security entity recognition model, we will compare different models. Our experiments concern mainly two groups of models: previous models and extended models. The models are listed as follows:

Previous models: To illustrate how well our model can handle security entity recognition, we compare our model with the following models:

1. CRF, proposed by Lafferty et al. (2001). We use CRF++ tools for security entity recognition, which can be considered context features.

2. LSTM, proposed by Hochreiter and Schmiduber (1997). We use LSTM to automatically extract

features and softmax to predict entity types.

3. LSTM-CRF, which combines LSTM with CRF. We extract features by the LSTM model and predict entity types by CRF.

4. BiLSTM-CRF, proposed by Huang et al. (2015). We use BiLSTM to capture long-term contextual feature information. Unlike the original model, we use character embedding.

5. CNN-BiLSTM-CRF, proposed by Ma and Hovy (2016), is a truly end-to-end system. We use CNN to extract character-level features and use BiLSTM to capture long-term contextual features. Then CRF is applied for learning and inference.

Extended models: We extend the model based on the previous model to improve the performance of the model for security entity recognition. We extend three models as follows:

1. FT-LSTM-CRF, which adds the feature template based on the LSTM-CRF model. We extract contextual features of the security entity through feature templates.

2. FT-BiLSTM-CRF, which adds the feature template based on the BiLSTM-CRF model.

3. FT-CNN-BiLSTM-CRF, which adds the feature template based on the CNN-BiLSTM-CRF model. This model is the one proposed in this study. We train this model with character embedding and use the feature template to extract contextual features.

We compare these models with our security entity recognition model. All models are trained on our training data and validation data, and evaluated on our testing data, and all neural network models are run using word2vec's 100-dimensional character embedding and the same hyperparameters as shown in Table 3. The model performance comparison results are shown in Table 4 and the performance of six types of security entities on different models in Fig. 4.

Table 4 Comparison of experimental results of different models, evaluated by accuracy, precision, recall, and *F*-value (%)

Model	Acc	*P*	*R*	*F*
CRF	91.50	84.26	73.34	78.42
LSTM	92.36	83.75	80.62	82.16
LSTM-CRF	92.95	86.17	82.07	84.07
BiLSTM-CRF	92.83	84.70	**85.18**	84.94
CNN-BiLSTM-CRF	93.10	86.47	84.07	85.25
FT-LSTM-CRF	93.03	87.09	82.78	84.88
FT-BiLSTM-CRF	90.87	88.19	82.19	85.08
FT-CNN-BiLSTM-CRF	**93.31**	**88.45**	83.68	**86.00**

First, we analyze the results of the previous models. Comparing CRF with LSTM from Table 4, the *F*-value of LSTM is 3.47% higher than that of CRF. CRF has poor performance in identifying SW and ORG (Fig. 5). Most of the two types of security entities are composed of Chinese and English, and the case is mixed. This reveals that the ability of CRF to recognize such entities is not as strong as that of LSTM. LSTM-CRF combines the advantages of CRF and LSTM, using LSTM to identify complex security entities, while CRF can make full use of the relationships between adjacent labels. BiLSTM-CRF performs slightly better than LSTM-CRF. The performance of CNN-BiLSTM-CRF is better than those of other models. The *F*-value is 85.25%, which demonstrates the effectiveness of the character features extracted by CNN.

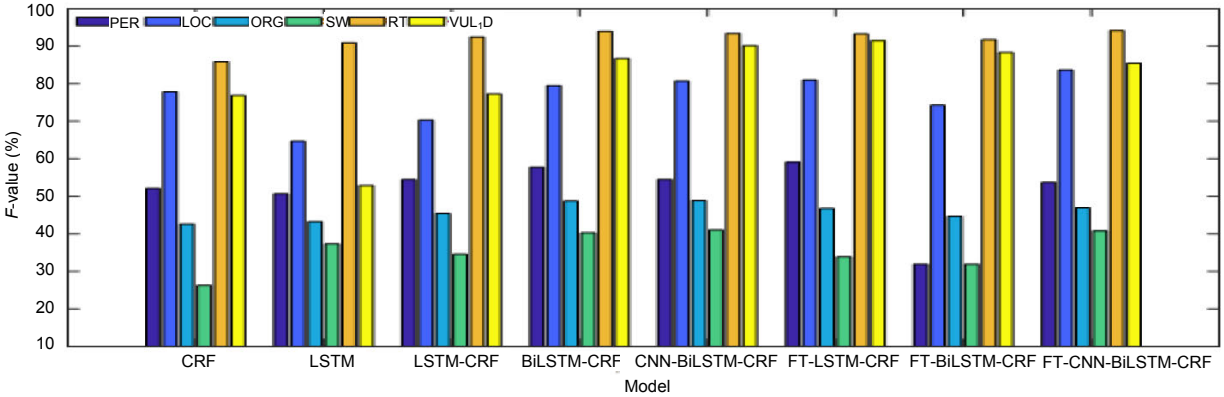

Fig. 4 Experimental results of six types of security entities in different models, evaluated by the *F*-value

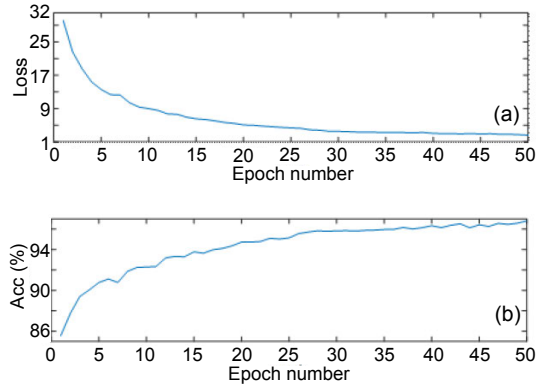

Fig. 5 The loss rate (a) and accuracy (b) of the validation data under different epoch numbers

Second, we discuss the results of our extended models. The extended model is based on the previous model by adding a feature template capable of extracting local contextual features. By comparing FT-LSTM-CRF with LSTM-CRF, the *F*-value of FT-LSTM-CRF is 0.81% higher in Table 4. FT-BiLSTM-CRF performs slightly better than BiLSTM-CRF, and we can see from Fig. 4 that BiLSTM-CRF has better performance in identifying ORG, with the *F*-value higher than that of FT-BiLSTM-CRF. Some organizations have very long names, such as "CERT 国家互联网应急中心." For such entities, the window of our FT is too small to fully extract the context features, and sometimes even reduce the recognition of the entire model. The last extended model is FT-CNN-BiLSTM-CRF proposed in this study. The performance of our model is better than those of other models. It achieves an accuracy of 93.31%, precision of 88.45%, and *F*-value of 86.00%, which is the highest in all models. As shown in Fig. 4, our model has better performance in identifying the LOC, SW, and RT, and the *F*-values of these three types of entities in our model are higher than those of other models. Most of the RT are composed of Chinese and English, such as "DDOS 攻击" and "SQL 注入漏洞." For such entities, our model uses CNN to extract features of the English part, uses FT to extract local context features, and uses BiLSTM to extract global features. Therefore, the performance of our model is improved greatly while introducing the feature template for security entity recognition.

4.4 Hyperparameter adjustment analysis

Our model is based on a neural network model. In the process of training the neural network, the hyperparameters of model training are generally initialized randomly, but the training efficiency and model performance are not high. Therefore, most of the model hyperparameters need to be adjusted to improve the model performance. In this subsection, we evaluate our model over different settings of the hyperparameters. Specifically, we are concerned about the impact of dropout (Pham et al., 2014), the number of epochs, and the pre-trained embedding of fine-tuning.

An epoch refers to a process in which complete training data is trained once and returned once by a neural network model. In the model, once is not enough, and training data needs to be transferred many times in the model. Fig. 5 shows the loss rate and the accuracy of the validation data under different epoch numbers. As shown in Fig. 5, as the number of epochs increases, the loss rate decreases quickly and then very slowly. The accuracy increases with the increase of the epoch number but does not rise after reaching a certain level. Therefore, after the training data passes multiple epochs, the curve becomes over-fitting from under-fitting. It is not always true that the larger the epoch number is, the higher the performance of the model is. It is important to select an appropriate epoch number.

With less training data, the training model can easily cause over-fitting, and the accuracy on the test set will be lower. Therefore, we set the dropout to prevent over-fitting during the training process. For each of the embeddings, we fine-tune the pre-trained embeddings in the context of security entity recognition. We can use the weights in the pre-training model to train the model without having to start from scratch. The weight of pre-training is better than that of random initialization. Fine-tuning can improve the computational efficiency and accuracy of the model.

The experimental results of adjusting the above three hyperparameters are shown in Table 5. We set the epoch number among {10, 20, 30, 40, 50}, dropout {0, 0.5}, and fine-tuning {true, false}. Through cross-validation of the training model, it can be seen that when the epoch number is 40, dropout is 0.5, and fine-tuning is true. In this case, the performance of the model is the highest for security entity recognition, in which the recall and *F*-value are 83.68% and 86%, respectively.

Table 5 Experimental results of epoch number, dropout, and fine-tuning parameter adjustment, evaluated by accuracy, precision, recall, and *F*-value (%)

Number of epochs	Dropout	Fine-tuning=false				Fine-tuning=true			
		Acc	*P*	*R*	*F*	Acc	*P*	*R*	*F*
10	0	91.89	84.25	78.26	81.14	92.46	85.16	79.77	82.38
	0.5	92.50	85.73	78.49	81.95	92.75	87.60	79.68	83.45
20	0	92.58	88.31	78.75	83.25	92.82	87.54	81.65	84.49
	0.5	92.57	86.84	80.02	83.29	92.94	88.45	80.95	84.53
30	0	92.69	86.81	79.50	82.99	92.89	88.21	81.20	84.56
	0.5	92.64	86.92	81.33	84.03	92.88	86.44	83.43	84.91
40	0	92.56	87.87	78.76	83.07	93.13	88.45	83.61	85.96
	0.5	92.20	85.90	**81.90**	83.86	**93.31**	88.45	**83.68**	**86.00**
50	0	92.59	88.09	78.78	83.17	93.14	88.99	81.12	84.87
	0.5	**92.93**	**88.79**	81.69	**85.09**	93.31	**89.45**	81.95	85.58

4.5 Network security entity extraction instance

In this study, we visualize six types of extracted network security entities in Fig. 6.

Fig. 6 Security entity word cloud

From Fig. 6 we observe that the network relevant term has the highest amount of network security text data, such as XSS cross-site scripting attacks, DDOS, and 0day. Similarly, a large number of vulnerability IDs appear, such as WooYun-2015-144542 and WooYun-2015-97677, which can be easily extracted due to certain construction rules. At the same time, our model can accurately identify and extract location in the text, such as 北京 (Beijing) and 伦敦 (London). Because our model has higher recognition performance for Chinese organizations than Chinese and English mixed organizations, we can clearly observe the organizations such as 百度 (Baidu) and 谷歌 (Google) in Fig. 6. Most software in the network text consists of Chinese and English, and a large amount of new software appears. Therefore, it is difficult to correctly extract the software. In Fig. 6, only a little software, such as Acrobat Reader, is extracted. There are fewer persons in network texts, so fewer persons are extracted.

5 Conclusions and future work

In this paper, we present a novel neural network approach for security entity recognition in the network security domain by introducing a feature template, which allows the model to extract local context features. In addition, we extract character-level feature sequences through CNN, and obtain the global feature vector sequence of network security text by BiLSTM. Finally, we obtain the optimal sequence of tags by means of labeling the security entity. The experimental results show that our proposed model has higher precision and *F*-value than other models in our network security corpus.

In future work, we will explore the issue of extracting the local features of entity. In this paper, we consider only the local context features of the entity rather than more local features. Therefore, we intend to introduce the attention mechanism of deep learning into our model. We will use the attention mechanism to extract local features and focus on the features that have important influence on security entity recognition. This is a further problem to be solved.

Compliance with ethics guidelines

Ya QIN, Guo-wei SHEN, Wen-bo ZHAO, Yan-ping CHEN, Miao YU, and Xin JIN declare that they have no conflict of interest.

References

Bergstra J, Bengio Y, 2012. Random search for hyperparameter optimization. *J Mach Learn Res*, 13(1):281-305.

Chiu JPC, Nichols E, 2015. Named entity recognition with bidirectional LSTM-CNNs. https://arxiv.org/abs/1511.08308

Collobert R, Weston J, 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. Proc ACM 25th Int Conf on Machine Learning, p.160-167. https://doi.org/10.1145/1390156.1390177

Collobert R, Weston J, Bottou L, et al., 2011. Natural language processing (almost) from scratch. *J Mach Learn Res*, 12(1):2493-2537.

Dong CH, Zhang JJ, Zong CQ, et al., 2016. Character-based LSTM-CRF with radical-level features for Chinese named entity recognition. In: Lin CY, Xue N, Zhao D, et al. (Eds.), Natural Language Understanding and Intelligent Applications. Springer, Cham, p.239-250. https://doi.org/10.1007/978-3-319-50496-4_20

Dos Santos C, Guimarães V, 2015. Boosting named entity recognition with neural character embeddings. Proc 5th Named Entity Workshop, joint with 53rd ACL and the 7th IJCNLP, p.25-33. https://doi.org/10.18653/v1/w15-3904

Feng YH, Yu H, Sun G, et al., 2018. Named entity recognition method based on BLSTM. *Comput Sci*, 45(2):261-268 (in Chinese). https://doi.org/10.11896/j.issn.1002-137X.2018.02.045

Finkel JR, Manning CD, 2009. Joint parsing and named entity recognition. Human Language Technologies: the Annual Conf of the North American Chapter of the Association of Computational Linguistics, p.326-334. https://doi.org/10.3115/1620754.1620802

Gers FA, Schmidhuber A, Cummins F, 2000. Learning to forget: continual prediction with LSTM. *Neur Comput*, 12(10):2451-2471. https://doi.org/10.1162/089976600300015015

Goller C, Kuchler A, 1996. Learning task-dependent distributed representations by backpropagation through structure. Proc Int Conf on Neural Networks, p.347-352. https://doi.org/10.1109/icnn.1996.548916

Hammerton J, 2003. Named entity recognition with long short-term memory. Proc 7th Conf on Natural Language Learning at HLT-NAACL, p.172-175. https://doi.org/10.3115/1119176.1119202

Hochreiter S, Schmidhuber J, 1997. Long short-term memory. *Neur Comput*, 9(8):1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Huang ZH, Wei X, Kai Y, 2015. Bidirectional LSTM-CRF models for sequence tagging. https://arxiv.org/abs/1508.01991

Joshi A, Lal R, Finin T, et al., 2013. Extracting cybersecurity related linked data from text. IEEE 7th Int Conf on Semantic Computing, p.252-259. https://doi.org/10.1109/icsc.2013.50

Koeling R, 2000. Chunking with maximum entropy models. Proc 2nd Workshop on Learning Language in Logic and the 4th Conf on Computational Natural Language Learning, p.139-141. https://doi.org/10.3115/1117601.1117634

Lafferty JD, McCallum A, Pereira FCN, 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. 18th Int Conf on Machine Learning, p.282-289.

Lample G, Ballesteros M, Subramanian S, et al., 2016. Neural architectures for named entity recognition. Proc NAACL-HLT, p.260-270. https://doi.org/10.18653/v1/N16-1030

LéCun Y, Bottou L, Bengio Y, et al., 1998. Gradient-based learning applied to document recognition. *Proc IEEE*, 86(11):2278-2324. https://doi.org/10.1109/5.726791

Li JH, 2016. Overview of the technologies of threat intelligence sensing, sharing and analysis in cyber space. *Chin J Network Inform Secur*, 2(2):16-29 (in Chinese). https://doi.org/10.11959/j.issn.2096-109x.2016.00028

Liu W, Li Y, Duan H, et al., 2016. Knowledge graph construction techniques. *J Comput Res Dev*, 53(3):582-600 (in Chinese). https://doi.org/10.7544/issn1000-1239.2016.20148228

Luo G, Huang XJ, Li CY, et al., 2015. Joint named entity recognition and disambiguation. Proc Conf on Empirical Methods in Natural Language Processing, p.879-888. https://doi.org/10.18653/v1/d15-1104

Ma XZ, Hovy E, 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. https://doi.org/10.18653/v1/p16-1101

Mikolov T, Chen K, Corrado G, et al., 2013a. Efficient estimation of word representations in vector space. https://arxiv.org/abs/1301.3781

Mikolov T, Sutskever I, Chen K, et al., 2013b. Distributed representations of words and phrases and their compositionality. https://arxiv.org/abs/1310.4546

Passos A, Kumar V, McCallum A, 2014. Lexicon infused phrase embeddings for named entity resolution. Proc 18th Conf on Computational Language Learning, p.78-86. https://doi.org/10.3115/v1/w14-1609

Peng NY, Dredze M, 2015. Named entity recognition for Chinese social media with jointly trained embeddings. Proc Conf on Empirical Methods in Natural Language Processing, p.548-554. https://doi.org/10.18653/v1/d15-1064

Pennington J, Socher R, Manning C, 2014. Glove: global vectors for word representation. Proc Conf on Empirical Methods in Natural Language Processing, p.1532-1543. https://doi.org/10.3115/v1/d14-1162

Pham V, Bluche T, Kermorvant C, et al., 2014. Dropout improves recurrent neural networks for handwriting recognition. 14th Int Conf on Frontiers in Handwriting Recognition, p.285-290. https://doi.org/10.1109/icfhr.2014.55

Qiu QQ, Miao DQ, Zhang ZF, 2013. Named entity recognition on Chinese microblog. *Comput Sci*, 40(6):196-198 (in Chinese). https://doi.org/10.3969/j.issn.1002-137X.2013.06.042

Rabiner LR, 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc IEEE*, 77(2):257-286. https://doi.org/10.1109/5.18626

Tang BZ, Cao HX, Wang XL, et al., 2014. Evaluating word representation features in biomedical named entity recognition tasks. *Biomed Res Int*, 2014:240403. https://doi.org/10.1155/2014/240403

Yang YM, 1999. An evaluation of statistical approaches to text categorization. *Inform Retriev*, 1(1-2):69-90. https://doi.org/10.1023/A:1009982220290

Yu HK, Zhang HP, Liu Q, et al., 2006. Chinese named entity identification using cascaded hidden Markov model. *J Commun*, 27(2):87-94 (in Chinese). https://doi.org/10.3321/j.issn:1000-436X.2006.02.013

Zhang XY, Wang T, Chen HW, 2005. Research on named entity recognition. *Comput Sci*, 32(4):44-48 (in Chinese). https://doi.org/10.3969/j.issn.1002-137X.2005.04.014