SCIENCE CHINA
Information Sciences

• LETTER •

September 2021, Vol. 64 199102:1–199102:3
https://doi.org/10.1007/s11432-019-1520-6

DeepDir: a deep learning approach for API directive detection

Jingxuan ZHANG[1,2,3*], He JIANG[4], Shuai LU[5], Ge LI[5] & Xin CHEN[3,6]

[1]*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016*, China;*
[2]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210023*, China;*
[3]*Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, Hangzhou* 310018*, China;*
[4]*School of Software, Dalian University of Technology, Dalian* 116024*, China;*
[5]*School of Electronic Engineering and Computer Science, Peking University, Beijing* 100871*, China;*
[6]*School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou* 310018*, China*

Received 30 April 2019/Accepted 20 July 2019/Published online 24 November 2020

Citation Zhang J X, Jiang H, Lu S, et al. DeepDir: a deep learning approach for API directive detection. Sci China Inf Sci, 2021, 64(9): 199102, https://doi.org/10.1007/s11432-019-1520-6

Dear editor,

Software developers tend to reuse existing libraries to facilitate their development process and implement certain functionalities by invoking application programming interfaces (APIs) [1]. However, it remains a challenging task for developers to correctly use APIs [2], so they often consult API learning resources [3, 4]. As one of the most important API learning resources, API specifications (also known as API references) detail the instructions on legal API usages with different types of knowledge [1], e.g., functionalities, concepts, and code samples. Out of these knowledge types, developers should particularly pay attention to API directives, i.e., the natural language statements to describe clear constraints or guidelines that developers should be aware of when programming with APIs [5]. Once API directives are neglected, fatal development and performance bugs may be easily produced in programming. Hence, it could be ideal if API directives can be automatically detected.

Monperrus et al. [5] proposed an approach to resolve the task of automatically detecting API directives in API specifications by leveraging a set of syntactic patterns. However, this approach simply treats API specifications as bags-of-words and lacks a deep semantic understanding of API specifications. The drawbacks of the bags-of-words assumption in the existing approach motivate us to shift our attention to deep learning models. Deep learning models can obtain a deep understanding of high-level semantics for natural language in two aspects. On the one hand, they can learn the distributional representations of words. Two words with similar semantics are close to each other in the semantic space, so they have a similar effect on deep learning models [6]. On the other hand, deep learning models can distinguish semantic differences between sentences with different word sequences. Hence, sentences with similar word sequences usually achieve similar results in deep learning models.

We propose DeepDir, a deep learning approach to automatically detect API directives in API specifications. Figure 1 shows the workflow of DeepDir. It consists of two phases, i.e., the training phase and the prediction phase [7]. The training phase trains a bi-directional long short-term memory (Bi-LSTM) network to learn the semantic differences between directives and non-directives. The prediction phase utilizes the trained Bi-LSTM network to predict whether a new sentence is a directive or not. We detail the two phases as follows.

(1) The training phase. First, given a training set and a validation set, we over sample API directives to make their proportion to be 50%. According to our statistics, API directives are extremely scarce in API specifications. For example, API directives only take up 4.87%, 6.62%, and 11.89% in the Java, JFace, and commons.collections API specifications respectively. By employing the over sampling strategy, API directives are fully exposed and DeepDir will not be overwhelmed by non-directives.

Then, each word in a sentence in the training set is embedded as a vector illustrating its distributional representation. The word embedding layer maps each word into an equally sized vector in the continuous vector space and the words sharing similar context are close to each other [8]. In this study, the words in each sentence are mapped into vectors with 150 dimensions by default. The value of each dimension is randomly initialized in the range of 0 to 1. The word vectors are trained together with the Bi-LSTM network. This procedure brings two benefits. First. there is no need to manually search for specific text corpora to train the vectors for the words, so it can save a lot of human efforts. Second, the word vectors are trained based on the API specification corpus, so they can better reflect the meaning and semantic of the words in API specifications. Therefore, along with the training of the Bi-LSTM network, the word vectors can be also learned simultaneously. Fur-

* Corresponding author (email: jxzhang@nuaa.edu.cn)

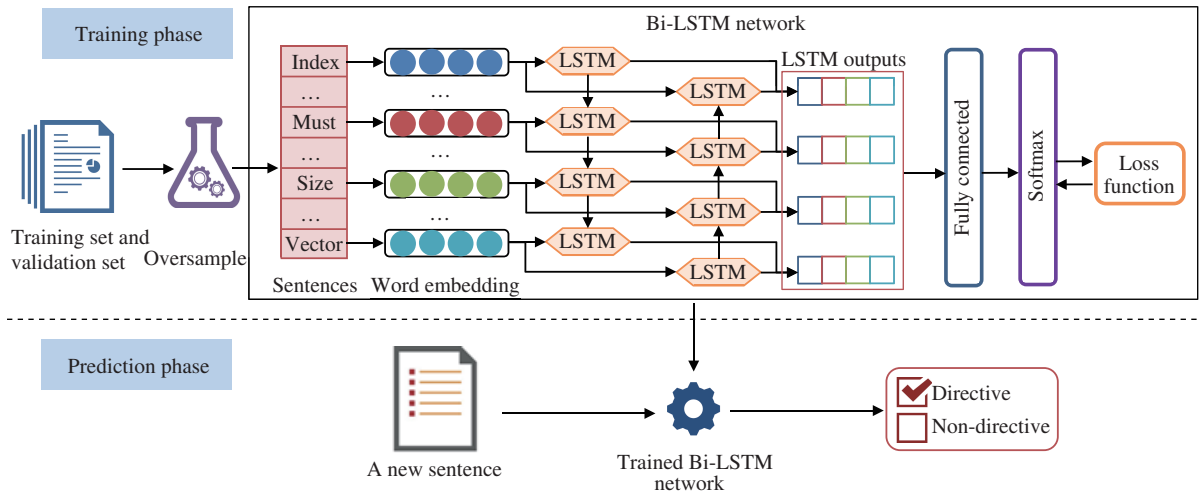

Figure 1 (Color online) The workflow of DeepDir.

thermore, the sentences in the training set are accordingly embedded into vector sequences, which are regarded as the inputs of the Bi-LSTM network.

Next, after word vectors are initialized, the vector sequences are input into two LSTM layers for learning in both the forward and backward directions. The two LSTM layers exploit the previous and future context regarding the current position, and learn the semantic differences between directives and non-directives from two directions. The two LSTM layers consist of a set of LSTM units. Each LSTM unit contains four main components, i.e., an input gate, a forget gate, an output gate, and a recurrent connection storing the state of the LSTM unit. Specifically, the input gate controls the new information to be stored in the LSTM unit, and the output gate decides the information the LSTM unit needs to output. Meanwhile, the forget gate chooses what information will be discarded from the state of the LSTM unit. The recurrent connection passes the information from the previous to the future.

Specifically, for a given sentence s (with length L) in an API specification, Let $s = (w_{I_1}, w_{I_2}, \ldots, w_{I_L})$, where w_{I_t} denotes the I_tth word in the vocabulary. The forward LSTM unit takes the sequence $\langle x_{I_1}, x_{I_2}, \ldots, x_{I_L}\rangle$ as input, where $x_{I_t} \in \mathbb{R}^d$ is the embedded vector of the I_tth word of the sentence. The equations for the forward pass of a LSTM unit are calculated as follows:

$$f_t = \sigma(W_f x_{I_t} + U_f h_{t-1} + b_f), \tag{1}$$

$$i_t = \sigma(W_i x_{I_t} + U_i h_{t-1} + b_i), \tag{2}$$

$$o_t = \sigma(W_o x_{I_t} + U_o h_{t-1} + b_o), \tag{3}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma(W_c x_t + U_c h_{t-1} + b_c), \tag{4}$$

$$h_t = o_t \circ \sigma(c_t), \tag{5}$$

where f_t, i_t, o_t, c_t, and $h_t \in \mathbb{R}^h$ $(1 \leqslant t \leqslant L)$ are forget gate vectors, input gate vectors, output gate vectors, cell state vectors, and output vectors, respectively. The weight matrices and the bias vector parameters $\{W_f, W_i, W_o\} \subseteq \mathbb{R}^{h\times d}$, $\{U_f, U_i, U_o\} \subseteq \mathbb{R}^{h\times h}$, and $\{b_f, b_i, b_o\} \subseteq \mathbb{R}^h$ need to be learned during the training phase. In addition, the embedded vectors of the total words in the vocabulary $\{x_1, x_2, \ldots, x_V\}$ are also determined during training. Similarly, the backward LSTM unit takes the sequence $\langle x_{I_L}, x_{I_{L-1}}, \ldots, x_{I_1}\rangle$ as input and outputs $\langle h'_1, h'_2, \ldots, h'_L\rangle$.

Finally, The fully connected layer receives the last outputs of the forward and backward LSTM units. Formally, the outputs of the fully connected layer are calculated as follows:

$$z = \sigma(W_{\text{full}}[h_L; h'_L] + b_{\text{full}}), \tag{6}$$

where $z \in \mathbb{R}^K$, $W_{\text{full}} \in \mathbb{R}^{K\times 2h}$ and $b_{\text{full}} \in \mathbb{R}^K$ are the weight matrices and the bias vector parameters to be learned. The softmax layer is stacked to turn the outputs into probabilities of classes. For class labels $\{\text{class}_1, \text{class}_2, \ldots, \text{class}_K\}$, the probability of s belongs to the jth class is calculated by the following softmax function:

$$p(\text{label}_s = \text{class}_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}. \tag{7}$$

By minimizing the loss function in the training set, the Bi-LSTM network can be fully trained. We optimize the loss function by gradient descendant. We employ the cross entropy as the loss function, which can be calculated as follows:

$$L = \frac{1}{M}\sum_{i=1}^{M} -(y'_i \log(y_i) + (1 - y'_i)\log(1 - y_i)), \tag{8}$$

where M is the number of sentences in the training set. In DeepDir, the class labels are {Directive, Non-Directive}. For a sentence s_i, if label_{s_i} = Directive, $y'_i = 1$. Otherwise, $y'_i = 0$. In addition, $y_i = p(\text{label}_{s_i} = \text{Directive})$.

DeepDir is validated on the validation set by F-measure in every 100 minibatches to evaluate its performance and avoid over-fitting. We select the model that achieves the best results on the validation set as the final model.

(2) The prediction phase. After the Bi-LSTM network is fully trained, it can be used to predict whether a new sentence in an API specification is an API directive or not. In the same way as the training phase, the sentence is first embedded into a word vector sequence. Then, the trained Bi-LSTM network receives the word vector sequence and outputs whether this new sentence is a directive or non-directive.

Results. We conduct some experiments to validate the effectiveness of DeepDir over an annotated API directive corpus with more than 85 thousand sentences from three API specifications, i.e., the Java, JFace, and commons.collections API specifications. We employ the approach proposed by

Monperrus et al. [5] as the baseline to compare. The experimental results reveal that DeepDir is superior to the baseline approach in terms of all the evaluation metrics. For example, the baseline approach only achieves an average precision of 31.10%. In contrast, DeepDir can achieve an average precision of 51.62%. The average improvement of DeepDir against the baseline approach is 20.52%. In terms of the average recall, the baseline approach achieves 73.21%. Meanwhile, DeepDir achieves 80.90%. From the perspective of the average F-measure, the baseline approach only obtains 41.42%. In contrast, DeepDir can reach to 62.20%. On average, DeepDir improves the baseline approach by 20.78% in terms of F-measure. Hence, DeepDir significantly improves the performance of API directive detection compared against the baseline approach.

Conclusion. API directive is one of the most important knowledge in API specifications. Existing approach only relies on syntactic patterns to detect API directives and lacks a deep semantic understanding. In this study, we propose a deep learning approach DeepDir to automatically detect API directives. Experimental results show that DeepDir significantly improves the state-of-the-art approach by 20.78% on average in terms of F-measure.

Acknowledgements This work was partially supported by National Key Research and Development Plan of China (Grant No. 2018YFB1003900).

References

1 Maalej W, Robillard M P. Patterns of knowledge in API reference documentation. IIEEE Trans Softw Eng, 2013, 39: 1264–1282
2 Jiang H, Zhang J X, Ren Z L, et al. An unsupervised approach for discovering relevant tutorial fragments for APIs. In: Proceedings of the 39th International Conference on Software Engineering (ICSE 17), 2017. 38–48
3 Huang Q, Xia X, Xing Z C, et al. API method recommendation without worrying about the task-API knowledge gap. In: Proceedings of International Conference on Automated Software Engineering (ASE 18), 2018. 293–304
4 Robillard M P, Chhetri Y B. Recommending reference API documentation. Empir Softw Eng, 2015, 20: 1558–1586
5 Monperrus M, Eichberg M, Tekes E, et al. What should developers be aware of? An empirical study on the directives of API documentation. Empir Softw Eng, 2012, 17: 703–737
6 Hu X, Li G, Xia X, et al. Deep code comment generation. In: Proceedings of IEEE International Conference on Program Comprehension (ICPC 18), 2018. 200–210
7 Chen X, Jiang H, Chen Z Y, et al. Automatic test report augmentation to assist crowdsourced testing. Front Comput Sci, 2019, 13: 943–959
8 Li X C, Jiang H, Kamei Y, et al. Bridging semantic gaps between natural languages and APIs with word embedding. IEEE Trans Softw Eng, 2020, 46: 1081–1097