BiTCNN: A Bi-Channel Tree Convolution Based Neural Network Model for Relation Classification

Feiliang Ren(✉), Yongcheng Li, Rongsheng Zhao, Di Zhou, and Zhihui Liu

School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China
renfeiliang@cse.neu.edu.cn

Abstract. Relation classification is an important task in natural language processing (NLP) fields. State-of-the-art methods are mainly based on deep neural networks. This paper proposes a bi-channel tree convolution based neural network model, *BiTCNN*, which combines syntactic tree features and other lexical level features together in a deeper manner for relation classification. First, each input sentence is parsed into a syntactic tree. Then, this tree is decomposed into two sub-tree sequences with top-down decomposition strategy and bottom-up decomposition strategy. Each sub-tree represents a suitable semantic fragment in the input sentence and is converted into a real-valued vector. Then these vectors are fed into a bi-channel convolutional neural network model and the convolution operations re performed on them. Finally, the outputs of the bi-channel convolution operations are combined together and fed into a series of linear transformation operations to get the final relation classification result. Our method integrates syntactic tree features and convolutional neural network architecture together and elaborates their advantages fully. The proposed method is evaluated on the SemEval 2010 data set. Extensive experiments show that our method achieves better relation classification results compared with other state-of-the-art methods.

Keywords: Relation classification · Syntactic parsing tree · Tree convolution Convolutional neural networks

1 Introduction

The aim of relation classification is that given a sentence in which two entities are labeled, to select a proper relation type from a predefined set for these entities. For example, given a sentence "*The system as described above has its greatest application in an arrayed <e1> configuration </e1> of antenna <e2> elements </e2>*", a relation classification system aims to identify that there is a "*Component-Whole*" relationship from *e2* to *e1*. Obviously, accurate relation classification results would benefit lots of NLP tasks, such as sentence interpretations, Q&A, knowledge graph construction, ontology learning, and so on. Thus, lots of researchers have devoted to this research field.

M. Zhang et al. (Eds.): NLPCC 2018, LNAI 11108, pp. 158–170, 2018.
https://doi.org/10.1007/978-3-319-99495-6_14

For relation classification, early research mostly focused on features based methods. Usually, these methods firstly select some syntactic and semantic features from the given sentences. Then the selected features are fed into some classification models like support vector machines, maximum entropy, etc. Recently, deep neural network (DNN) based methods have been widely explored in relation classification and have achieved state-of-the-art experimental results. The core of these methods is to embed features into real-valued vectors, and then feed these vectors into DNN architectures. Usually, deep convolutional neural networks (CNN) and deep recurrent neural networks (RNN) are two most widely used architectures for relation classification.

In most recent years, inspired by the broad consensus that syntactic tree structures are of great help and the great success of DNN, more and more research attention is being paid to the methods that integrate syntactic tree features into DNN models. However, most of these existing methods used syntactic tree in a very shallow manner: syntactic tree structure is often taken as an intermediate supporter from which a specific kind of context can be extracted for CNN or RNN models. Obviously, such shallow manner does not make full use of the rich semantic information carried by syntactic tree structures.

It is worth noting that Socher et al. (2013a, b) introduced Compositional Vector Grammar (CVG for short), which used a syntactically untied RNN model to learn a syntactic-semantic compositional vector representation for the category nodes in a syntactic tree. Inspired by their work, we propose a new relation classification method that integrates syntactic tree structures into CNN model with a deeper manner. Specifically, in our method, each input sentence is first parsed into a syntactic tree. Then this tree is decomposed into two sub-tree sequences with bottom-up and top-down decomposition methods respectively. Thirdly, each sub-tree is encoded into a real-valued vector. Fourthly, the two sub-tree vector sequences are fed into a bi-channel CNN model to generate final classification result. Experimental results show that our method achieves better results compared with other baseline methods.

2 Related Work

Generally, there are three widely used DNN architectures for relation classification: CNN, RNN, and their combination.

Zeng et al. (2014) proposed a CNN based approach for relation classification. In their method, sentence level features are learned through a CNN model that takes word embedding features and position embedding features as input. In parallel, lexical level features are extracted from some context windows that are around the labeled entities. Then sentence level features and lexical level features are concatenated into a single vector. This vector is fed into a softmax classifier for relation prediction. Wang et al. (2016) proposed a multi-level attention CNN model for relation classification. In their method, two levels of attentions are used in order to better discern patterns in heterogeneous contexts.

Socher et al. (2012) used RNN for relation classification. In their method, they build recursive sentence representations based on syntactic parsing. Zhang and Wang (2015) investigated a temporal structured RNN with only words as input. They used a

bi-directional model with a pooling layer on top. Xu et al. (2015a, b) picked up heterogeneous information along the left and right sub-path of the Shortest Dependent Path (SDP) respectively, leveraging RNN with LSTM. In their method, the SDP retains most relevant information to relation classification, while eliminating irrelevant words in the sentence. And the multichannel LSTM networks allow effective information integration from heterogeneous sources over the dependency paths. Meanwhile, a customized dropout strategy regularizes the neural network to alleviate over-fitting. Besides, there is also other similar work. For example, Hashimoto et al. (2013) explicitly weighted phrases' importance in RNNs. Ebrahimi and Dou (2015) rebuilt an RNN on the dependency path between two labeled entities.

Some researchers combined CNN and RNN for relation classification. For example, Vu et al. (2015) investigated CNN and RNN as well as their combination for relation classification. They proposed extended middle context, a new context representation for CNN architecture. The extended middle context uses all parts of the sentence (the relation arguments, left/right and between of the relation arguments) and pays special attention to the middle part. Meanwhile, they used a connectionist bi-directional RNN model and a ranking loss function is introduced for the RNN model. Finally, CNN and RNN were combined with a simple voting scheme. Cai et al. (2016) proposed a bidirectional neural network BRCNN, which consists of two RCNNs that can learn features along SDP inversely at the same time. Specifically, information of words and dependency relations are used with a two-channel RNN model with LSTM units. The features of dependency units in SDP are extracted by a convolution layer. Liu et al. (2015) used a RNN to model the sub-trees, and a CNN to capture the most important features on the SDP.

3 Our Model

Figure 1 demonstrates the architecture of our method. The network takes sentences as input and extracts syntactic tree features and other useful features. These features are converted into real-valued vector representations and then fed into a bi-channel CNN model for relation type decision. From Fig. 1 we can see that there are six main components in our method: tree decomposition, feature extraction, convolution transformation, max-pooling operation, linear transformation and output.

3.1 Tree Decomposition

Each input sentence will firstly be parsed into a syntactic tree by the Stanford Parser. Then this tree is decomposed into two sub-tree sequences with bottom-up decomposition method and top-down decomposition method. These two kinds of decomposition methods, whose algorithms are shown in Figs. 2 and 3 respectively, complement each other and are expected to generate more meaningful and less ambiguous semantic fragments than words.

For the top-down tree decomposition method, its generated sub-trees don't contain any word information. It is expected to extract the common syntactic sub-tree structures for a specific kind of relationship type, and is also expected to alleviate the over-fitting

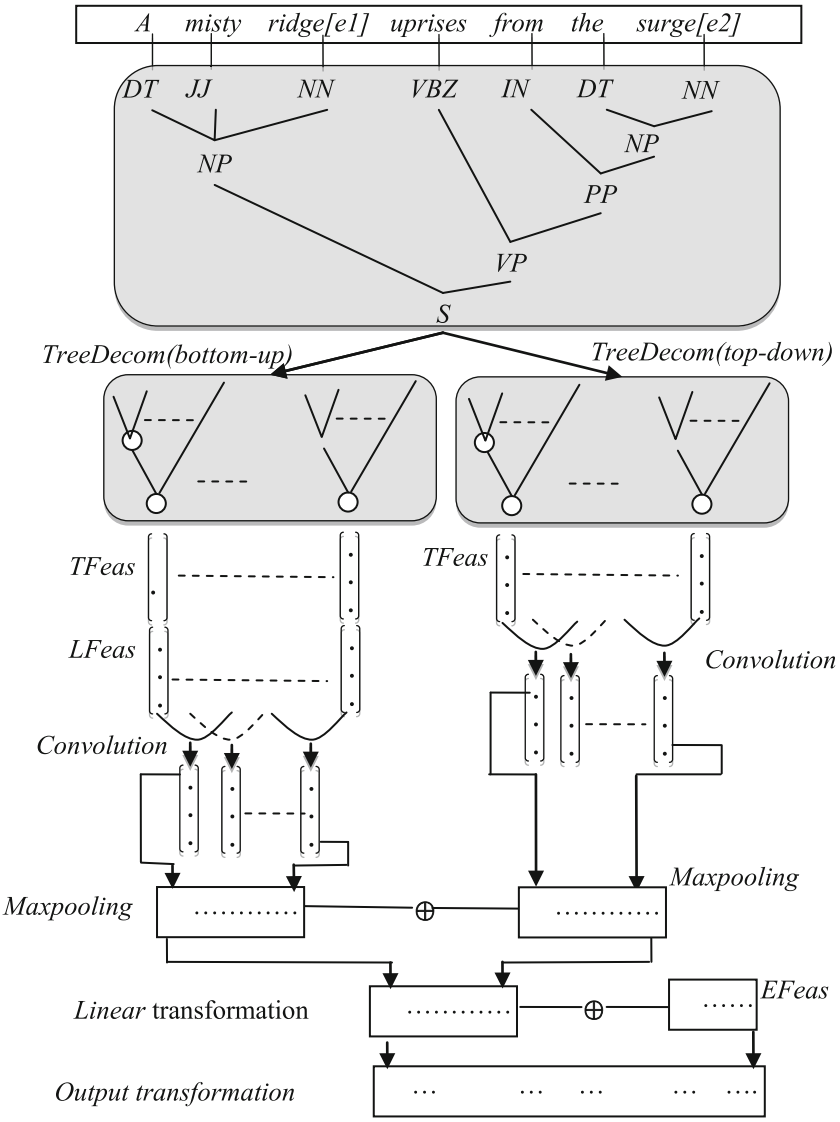

Fig. 1. Architecture of our method

issue by using these abstract sub-tree structures as features. Taking the sentence in Fig. 1 as an example, the sub-tree sequence generated by this method is: (S (NP) (VP)), (NP ((DT) (JJ) (NN))), (VP (VBZ) (PP)), (PP (IN) (NP)), (NP (DT) (NN)).

As for the bottom-up method, it complements with the top-down method. In this method, word information is taken into consideration. Taking the sentence in Fig. 1 as an example, if the hyper parameters in Fig. 3 are set as: h = 3, Δ = 3, and k = 3, the sub-tree sequence generated by this method would be: (NP ((DT A) (JJ misty) (NN ridge))), (VP (VBZ uprises) (PP (IN from) (NP (DT the) (NN surge)))).

Input: a syntactic tree T Output: a sub-tree sequence $Set_1(T)$
Procedure: 1: Exit if all of the leaf nodes in T have been processed. Otherwise, go to step 2. 2: Take the root node and all of its son nodes as the first sub-tree and add this sub-tree into $Set_1(T)$, move to the son layer of the root node. 3: Exit if the number of covered layers equals to the height of T. Otherwise, go to step 3. 4: Visit all of the non-leaf nodes in current layer by the depth-first strategy. Take an unprocessed non-leaf node and all of its son nodes as a sub-tree candidate, if all of its nodes in this sub-tree are non-leaf nodes, add it into $Set_1(T)$. 5: Move to the next layer and go to step 2.

Fig. 2. Algorithm of top-down tree decomposition

Input: a syntactic tree T Output: a sub-tree sequence $Set_2(T)$
Procedure: 1: Exit if all of the leaf nodes have been processed. Otherwise, go to step 2. 2: Take the left-most unprocessed leaf node l_i. 3: Move upward h layer from l_i to find its ancestor node a_i and extract the sub-tree t_i that takes a_i as root node. If all the out-degrees in t_i are less than Δ and the height of t_i is less than k, add t_i into $Set_2(T)$, then go to step 2. Otherwise, go to step 4. 4: Let $h = h - 1$ and go to step 3.

Fig. 3. Algorithm of bottom-up tree decomposition

3.2 Feature Extraction

There are three kinds of features used in our model: syntactic tree features, lexical level features and entities level features. In Fig. 1, they are denoted as ***TFeas***, ***LFeas*** and ***EFeas*** respectively.

- Syntactic Tree Features

Syntactic tree structures can carry more semantic and syntactic information compared with characters, words, or phrases. To make full use of such information, we embed each sub-tree into a d^{tree}-dimensional real-valued vector. Just like a word embedding vector can encode different meaning of the word into a vector, we hope that a tree embedding vectors can also encode as much as possible semantic and syntactic information for a tree structure. In our method, this kind of tree embedding vectors is initialized with the method proposed by Socher et al. (2013a, b). Their method can assign a vector representation for both the nodes in a parsing tree and the whole parsing tree itself. But in our model, we don't care the representations of any inner categories in a parsing tree. Thus, our method assigns a vector representation for the whole tree structure only.

- Lexical Level Features

Lexical level features refer to the features that are related to words. Thus only the sub-trees generated by the bottom-up decomposition method will involve this kind of features. There are two kinds of lexical level features used in our method: word embedding features and position features. The final lexical level features, ***LFeas***, are the concatenations of these two kinds of features.

1. **Word Embedding Features**

Word embedding is a kind of word representation method and is widely used in DNN models. It converts a word into a real-valued vector representation to capture the rich syntactic and semantic features possessed by this word. Generally, a word embedding table is a d^w*|V| real-valued matrix where each column corresponds to a word representation. d^w is the dimension of a embedding vector, and |V| is the vocabulary size.

For a sub-tree t_i that is generated by the bottom-up decomposition method, each of its leaf nodes has a word embedding vector. If there are m leaf nodes in t_i, its word embedding features $wf(t_i)$ would be the arithmetic mean of its m words' embeddings, as shown in:

$$wf(t_i)_j = avg \sum_{k=1}^{m} w_{kj},\ 1 \le j \le d^w \tag{1}$$

2. **Position Features**

Position features are used to specify which input items are the labeled entities in a sentence or how close an input item is to the labeled entities. They have been proved to be effective for relation classification (Dos et al. (2015); Zeng et al. (2014)). This kind of features maps a distance value into a randomly initiated d^{dst}-dimensional real-valued vector.

For a sub-tree t_i that is generated by the bottom-up decomposition method, each of its leaf nodes has two kinds of position features that are related with e_1 and e_2 respectively. Accordingly, there are two kinds of position features for t_i.

If there are m leaf nodes in t_i, its position features related with e_1, denoted as $pf(e_1)$, would be the arithmetic mean of its m leaf nodes' position features related with e_1. The computation process is shown in formula 2.

$$pf(e_1)_j = avg \sum_{k=1}^{m} p_{kj},\ 1 \le j \le d^{dst} \tag{2}$$

Similarly, the computation process for t_i's position features related with e_2, denoted as $pf(e_2)$, is shown in formula 3.

$$pf(e_2)_j = avg \sum_{k=1}^{m} p_{kj},\ 1 \le j \le d^{dst} \tag{3}$$

Finally, the position features for t_i is the concatenation of $pf(e_1)$ and $pf(e_2)$.

3. **Entities Level Features**

Previous research shows that words between the two labeled entities could provide more useful cues for relation decision. So an attention scheme is used here to enhance the features extracted from the words that are between the two labeled entities. The enhanced feature is called entities level features and is denoted as ***EFeas*** in Fig. 1.

We first get the syntactic tree for the text that is from the first labeled entity to the second labeled entity. Then the syntactic tree features and the lexical level features are extracted with the same method introduced previously. Thirdly, these two kinds of features are concatenated to form a new feature vector that is denoted as ***EntityF***. The final ***EFeas*** is generated with formula 4.

$$\boldsymbol{EFeas} = tanh(M_1 * \boldsymbol{EntityF} + b_1) \tag{4}$$

Where $M_1 \in R^{h1 * def}$ is a transformation matrix, h_1 is a hyper parameter that denotes the transformed size, def is the dimension of vector $\boldsymbol{EntityF}$, b_1 is a bias term.

From Fig. 1 we can see that ***EFeas*** would be concatenated with the linear transformed max-pooling features. To maintain feature consistency, here we use the linear transformed ***EntityF*** as ***EFeas***.

3.3 Convolution Transformation

Convolution transformation is a kind of linear transformation and is expected to extract more abstract features.

For the sub-tree sequence that is generated by the bottom-up method, each of its sub-tree t_i is represented by a vector concatenation of ***TFeas(i)*** and ***LFeas(i)***. The convolution transformation process is written as formula 5.

$$\boldsymbol{CMtrB_i} = M_2 * (\boldsymbol{TFeas}(\boldsymbol{t_i}) \oplus \boldsymbol{LFeas}(\boldsymbol{t_i})) + b_2 \qquad \forall i \tag{5}$$

For the sub-tree sequence that is generated by the top-down method, each of its sub-tree t_i is represented by ***TFeas(i)***. The convolution process is written as formula 6.

$$\boldsymbol{CMtrT_i} = M_3 * \boldsymbol{TFeas}(\boldsymbol{t_i}) + b_3 \qquad \forall i \tag{6}$$

In above formulas, $M_2 \in R^{h2 * dbf}$ and $M_3 \in R^{h3 * dtf}$ are two transformation matrices, h_2 and h_3 are the sizes of transformed hidden layers, d^{bf} and d^{tf} are the dimensions of sub-tree vectors in the bottom-up and top-down tree sequences respectively. b_2 and b_3 are bias terms. $\oplus$ denotes the concatenation operation.

3.4 Max-Pooling Operation

After convolution transformation, both ***CMtrB*** and ***CMtrT*** depend on the length of input sequence. To apply subsequent standard affine operations, max-pooling operation is used to capture the most useful and fixed size local features from the output of convolution transformation. This process is written as formula 7 and 8.

$$p_{bi} = max_n CMtrB(i, n) \qquad 0 \leq i \leq h_2 \tag{7}$$

$$p_{ti} = max_n CMtrT(i, n) \qquad 0 \leq i \leq h3 \tag{8}$$

After max-pooling operation, $\boldsymbol{p_b}$ and $\boldsymbol{p_t}$ will have h_2 and h_3 elements respectively, which are no longer related to the length of input.

3.5 Linear Transformation

After the max-pooling operation, $\boldsymbol{p_b}$ and $\boldsymbol{p_t}$ are concatenated together to form a new vector $\boldsymbol{p}$. Then $\boldsymbol{p}$ is fed into a linear transformation layer to perform affine transformation. This process is written as formula (9).

$$f = tanh(M_4 * \boldsymbol{p} + b_4) \tag{9}$$

$M_4 \in R^{h4*(h2+h3)}$ is the transformation matrix and h_4 is the size of hidden units in this layer, and b_4 isa bias term.

3.6 Output

After linear transformations, vector f and vector ***EFeas*** are concatenated together to form a new vector $\boldsymbol{o}$. Then $\boldsymbol{o}$ is fed into a linear output layer to compute the confidence scores for each possible relationship type. A softmax classifier is further used to get the probability distribution y over all relation labels as formula 10.

$$y = softmax(M_5 * \boldsymbol{o} + b_5) \tag{10}$$

Here $M_5 \in R^{h5*(h1+h4)}$ and h_5 is the number of possible relation types. Softmax is computed with formula 11.

$$y_i = e^{xi} / \sum_m e^{xm} \tag{11}$$

3.7 Dropout Operation

Over-fitting is an issue that cannot be ignored in DNN models. Hinton et al. (2012) proposed dropout method that has been proved to be effective for alleviating over-fitting. This method randomly sets a proportion (called drop rate, a hyper-parameter) of features to zero during training. It is expected to obtain less interdependent

network units, thus over-fitting issue is expected to be alleviated. In our method, dropout strategy is taken at feature extraction phase and linear transformation phase. Specially, we take dropout operation on ***EntityF***, ***TFeas***, ***LFeas*** in formula 4 ~ 6 respectively, and on ***p*** in formula 9. The drop rates for them are denoted as $dp_{1\sim4}$ respectively.

3.8 Training Procedure

All the parameters in our method can be denoted as $\theta = (E^w, E^t, E^p, M_1, M_2, M_3, M_4, M_5, b_1, b_2, b_3, b_4, b_5)$, where E^w, E^t and E^p represent the embeddings of word, syntactic tree and position respectively. E^w is initialized by the pre-trained embeddings SENNA (Collobert et al. 2011).Et is initialized with the method introduced by Socher et al. (2013a, b). E^p, transformation matrices, and bias terms are randomly initialized. All the parameters are tuned using the back propagation method. Stochastic gradient descent optimization technique is used for training. Formally, we try to maximize following objective function.

$$J(\theta) = \sum_{i=1}^{N} logy_i \tag{12}$$

where N is the total number of training samples. During training, each input sentence is considered independently. And each parameter is updated by applying following update rule, in which η is the learning rate.

$$\theta = \theta + \eta * \partial logy / \partial \theta \tag{13}$$

4 Experiments and Analysis

4.1 Datasets

The SemEval-2010 Task 8 dataset is used to evaluate our method. In this dataset, there are 8000 training sentences and 2717 test sentences. For each sentence, two entities that are expected to be predicted a relation type are labeled. There are 9 relation types whose directions need to be considered and an extra artificial relation "*Other*" which does not need to consider the direction. Thus totally there are 19 relation types in this dataset.

Macro-averaged F1 score (excluding "*Other*"), the official evaluation metric, is used here and the direction is considered. During experiments, all the syntactic trees are generated by the Stanford Parser (Klein and Manning, 2003). We apply a cross-validation procedure on the training data to select suitable hyper-parameters. Finally, the best configurations obtained are: $d^{dst} = 75$, $\eta = 0.001$, $h_{1\sim4}$ are 250, 200, 200 and 300 respectively, $dp_{1\sim4}$ are all set to 0.5. In Fig. 3, Δ, h and k are set to 3, 3 and 3 respectively. Other parameters, d^w and d^{tree} are 50 and 25 respectively.

4.2 Experimental Results and Analyses

In the first part of our experiment, we evaluate the contributions of different kinds of features and different convolutional channels. To this end, we implement a CNN model that is similar to the one described in Zeng et al. (2014). This CNN model is denoted as *baseline* in which word embedding features and position features are used. Besides, we implement two other CNN models: one is with the bottom-up tree convolution channel, and the other is with the top-down tree convolution channel. Then we investigate how the performance changes when different kinds of features are added. The experimental results are reported in Table 1.

Table 1. Performance of our method with different features

Model	F1
Baseline	82.4
Bottom-up tree convolution(without LFeas)	73.0
Bottom-up tree convolution + *WordEmb feature*	74.6
Bottom-up tree convolution + *Position feature*	74.0
Bottom-up tree convolution + *LFeas*	76.3
Top-down tree convolution	65.5
Our model	84.8

We can see that our model is very effective and it outperforms the baseline system greatly. Also we can see that different convolution channels have different classification performance. Even without ***LFeas*** features, the bottom-up tree CNN model achieves better performance than the top-down tree CNN model. We think the main reason is that in the bottom-up model, word information is retained and this kind of information would play positive role for relation classification. This can be further proved by the experimental results: the performance improves when different kinds of lexical features are added in the bottom-up tree CNN model.

In the second part of our experiment, we compare our method with several other state-of-the-art DNN based methods. Because the datasets used are the same, we directly copy the experimental results reported in Zeng et al. (2014). The comparison results are shown in Table.

From Table 2 we can see that our method achieves better results compared with other methods. It is also worth noting that our method is the ONLY ONE that does not use any external language-dependent resources like WordNet. This shows the effectiveness of embedding syntactic tree features into CNN architecture for relation classification.

In the third part of our experiment, we compare the classification performance for different types of relationships. The comparison results are reported in Table 3.

From the experimental results we can see that the performance of different types of relationships is very different. Even excluding "*Other*" type, the best performance (for example, "*cause-effect*" and "*entity-destination*") is almost 10% higher than the worst performance (for example, "*product-producer*" and "*content-container*"). Further

Table 2. Comparisons with other methods

Method	Features and extra resources used	F1
CNN	WordNet	82.7
SVM	POS, prefixes, morphological, WordNet, dependency parse, Levin classed, ProBank, FrameNet, NomLex-Plus, Google n-gram, paraphrases, TextRunner	82.2
RNN	- POS, NER, WordNet	74.8 77.6
MVRNN	- POS, NER, WordNet	79.1 82.4
Our model	parsing trees	84.8

Table 3. Classification results of different relationships

Relationship	P	R	F
Cause-Effect	95.43%	88.17%	91.65%
Component-Whole	80.45%	81.76%	81.10%
Content-Container	84.38%	77.14%	80.60%
Entity-Destination	95.21%	87.97%	91.45%
Entity-Origin	89.15%	85.82%	87.45%
Instrument-Agency	83.97%	76.61%	80.12%
Member-Collection	91.85%	76.70%	83.59%
Message-Topic	88.51%	74.76%	81.05%
Product-Producer	82.68%	78.60%	80.59%
Other	40.75%	71.43%	51.89%

investigation shows that for the relationships that model can classify better, there are usually some clear indicating words in the original sentences. For example, words "*cause*", "*caused*", and "*causes*" are often in the sentences where a "*cause-effect*" relationship holds. On the contrary, there are usually few of such kind of indicating words for the relationships that model classifies worse. As a result, max-pooling operation couldn't work well in such cases.

5 Conclusions and Future Work

In this paper, we propose a new relation classification method. The main contributions of our method are listed as follows.

First, our method uses syntactic tree structures in a deeper manner: the input sentence is parsed into a syntactic tree. And this tree is further decomposed into two sub-tree sequences and the convolution operations are performed on the sub-tree embeddings directly.

Second, we design two decomposition methods to guarantee the tree decomposition process performed in a reasonable way, which means that each of the generated subtrees has a relatively complete structure and can express a complete meaning.

However, there are still some other issues needed to be further investigated. For example, experimental results show that there are big performance gaps between different types of relationships, which should be further investigated in the future.

Acknowledgements. This work is supported by the National Natural Science Foundation of China (NSFC No. 61572120, 61672138 and 61432013).

References

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)

Wang, L., Cao, Z., de Melo, G., Liu, Z.: Relation classification via multi-level attention CNNs. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 1298–1307 (2016)

Cai, R., Zhang, X., Wang, H.: Bidirectional recurrent convolutional neural network for relation classification. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 756–765 (2016)

Xu, K., Feng, Y., Huang, S., Zhao, D.: Semantic relation classification via convolutional neural networks with simple negative sampling. In: Proceedings of 2015 Conference on Empirical Methods in Natural Language Processing, pp. 536–540 (2015a)

Xu, Y., Mou, L., Li, G., Chen, Y., Peng, H., Jin, Z.: Classifying relations via long short term memory networks along shortest dependency paths. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1785–1794 (2015b)

Zeng, D., Liu, K., Lai, S., Zhou, G., Zhao, J.: Relation classification via convolutional deep neural network. In: Proceedings of the 25th International Conference on Computational Linguistics, pp. 2335–2344 (2014)

Zhang, Z., Zhao, H., Qin, L.: Probabilistic graph-based dependency parsing with convolutional neural network. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 1382–1392 (2016)

Socher, R., Bauer, J., Manning, C.D., Ng, A.Y.: Parsing with compositional vector grammars. In: Proceedings of the 51th Annual Meeting of the Association for Computational Linguistics, pp. 455–465 (2013a)

Socher, R., Perelygin, A., Wu, J.Y., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1631–1642 (2013b)

dos Santos, C.N., Xiang, B., Zhou, B.: Classifying relations by ranking with convolutional neural networks. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, pp. 626–634 (2015)

Liu, Y., Wei, F., Li, S., Ji, H., Zhou, M., Wang, H.: A dependency-based neural network for relation classification. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, pp: 285–290 (2015)

Vu, N.T., Adel, H., Gupta, P., Schutze, H.: Combining recurrent and convolutional neural networks for relation classification. In: Proceedings of NAACL-HLT 2016, pp. 534–539 (2015)

Hashimoto, K., Miwa, M., Tsuruoka, Y., Chikayama, T.: Simple customization of recursive neural networks for semantic relation classification. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1372–1376 (2013)

Socher, R., Huval, B., Manning, C.D., Ng, A.Y.: Semantic compositionality through recursive matrix-vector spaces. In: Proceedings of the 2012 Joint Conference on EMNLP and Computational Natural Language Learning, pp. 1201–1211 (2012)