ZQM at SemEval-2019 Task9: A Single Layer CNN Based on Pre-trained Model for Suggestion Mining

Qimin Zhou, Zhengxin Zhang, Hao Wu*, Linmao Wang
School of Information Science and Engineering, Yunnan University
Chenggong Campus, Kunming, P.R. China
{zqmynu,zzxynu}@gmail.com, haowu@ynu.edu.cn, wlmdyx@gmail.com

Abstract

This paper describes our system that competed at SemEval 2019 Task 9 - SubTask A: "Suggestion Mining from Online Reviews and Forums". Our system fuses the convolutional neural network and the latest BERT model to conduct suggestion mining. In our system, the input of convolutional neural network is the embedding vectors which are drawn from the pre-trained BERT model. And to enhance the effectiveness of the whole system, the pre-trained BERT model is fine-tuned by provided datasets before the procedure of embedding vectors extraction. Empirical results show the effectiveness of our model which obtained $9th$ position out of 34 teams with F1 score equals to 0.715.

1 Introduction

Suggestion mining is defined as the extraction of suggestions from unstructured text (Negi et al., 2018). Suggestion mining is still a relatively young research area as compared to other natural language processing issues like sentiment analysis (Negi and Buitelaar, 2015). While suggestion mining is of great commercial value for organisations to improve the quality of their entities by considering the positive and negative opinions collected from platforms. The target of this task is to automatically classify the sentences collected from online reviews and forums into two classes which are suggestion and non-suggestion respectively (Negi et al., 2019).

BERT which stands for **B**idirectional **E**ncoder **R**epresentation from **T**ransformers is the latest breakthrough in the field of NLP provided by Google Research (Devlin et al., 2018). It has substantially advanced the state-of-the-art in many NLP tasks, especially in question answering (Alberti et al., 2019). More importantly, it provides a widely applicable tool for representation learning which can be generalized to many NLP tasks.

For this subtask, we firstly learn the word or sentence embeddings utilizing the pre-trained BERT model. Then the embedding vectors are extracted from BERT as the input of the subsequent model. It is worth noting that we have fine-tuned the pre-trained BERT model with provided dataset before the embedding vectors are extracted. In other words, this part is equivalent to the conventional embedding layer. This strategy is a little bit like ELMO (Peters et al., 2018). As for the upper layer of this system, convolutional neural network (CNN) is adopted herein to process the features. Although CNN is originally invented for tackling computer vision issues, while it has subsequently been shown to be effective for many NLP tasks (Kim, 2014; Zhang and Wallace, 2015; Dong et al., 2015).

The remainder of the paper is organized as follows. Section 2 describes the detailed architecture of our system. Section 3 reports the experimental results on the given datasets. Finally, conclusions are drawn in Section 4.

2 System Description

Figure 1 gives a high-level overview of our approach. And we elaborate the details of implementation which mainly consists of following steps: (1) the preprocessing of raw data, (2) the word embedding learning via BERT model, (3) feature processing via CNN and sentences classification.

2.1 Data Preprocessing

The provided dataset is collected from feedback posts on Universal Windows Platform and annotated by (Negi et al., 2018). But the text is not standard enough as there are some spelling mistakes and few duplicate samples. To boost the performance of our system, we conduct some pre-

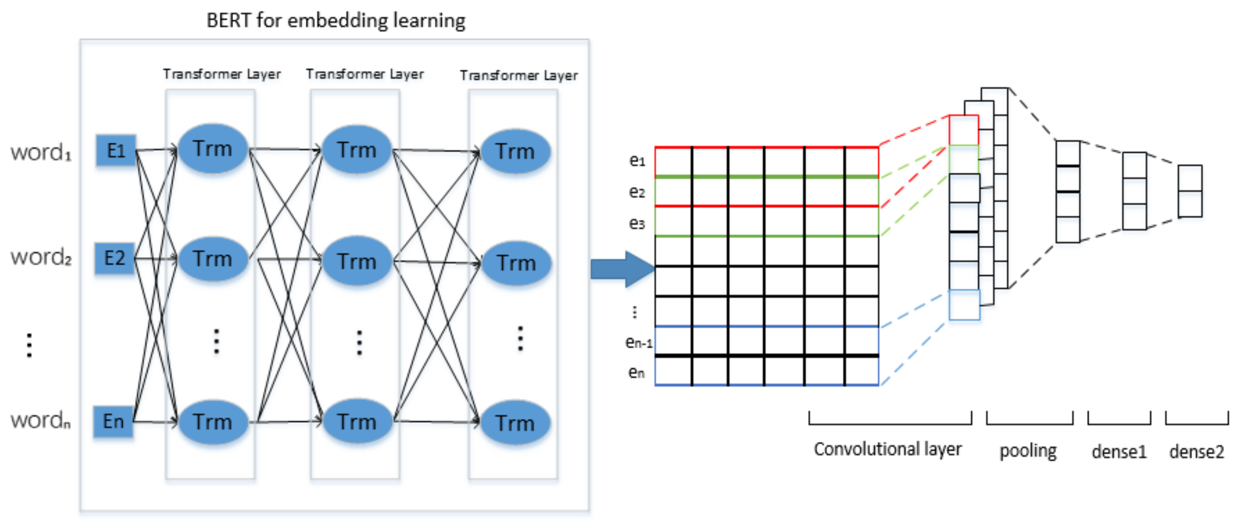

Figure 1: The architecture of our proposed model.

processing steps on the raw data. At first, web links are removed through regular expression as it does not contribute to the accuracy of classification. After that, we can take more meaningful words into consideration under the condition of a finite sentence length. And *ekphrasis* [1], a text processing tool, is utilized to conduct spelling correction (Baziotis et al., 2017). Then, all characters are converted to lowercase. Finally, duplicated samples would be excluded from the dataset.

2.2 Embedding Learning via BERT

Embedding layer usually encodes each word into a fixed-length vector for subsequent study. *Word2vec*, *Glove* and *FastText* are the most simple and popular word embedding algorithms (Mikolov et al., 2013; Pennington et al., 2014; Bojanowski et al., 2017). While there continue to be some drawbacks, such as they cannot encode the contextual information well. Recently, a few effective algorithms have been put forward such as *ELMO* and *openAI GPT* (Peters et al., 2018; Radford et al., 2018). These two pre-trained language models can encode rich syntactic and semantic information and distinguish the different meanings of a polysemy in diverse contexts, which traditional word embeddings methods cannot handle well. *ELMO* leverages the concatenation of independently trained left-to-right and right-to-left LSTM to generate features. Though LSTM can capture contextual information, the performance can be limited by the long distance of sequences to some extent. In order to deal with this problem, *openAI GPT* substitute *Transformer* for LSTM. *Transformer* rely entirely on attention mechanism to capture global dependencies (Vaswani et al., 2017).

BERT is the latest language representation model which also takes advantage of *Transformer*. Besides that, it uses the masked language model (MLM) and the bidirectional *Transformers* to capture the contextual information which has been proved to be effective (Devlin et al., 2018). In our system, we employ *BERT* as the embedding layer, in other words, we use the output of the last transformer layer from BERT as word embedding vectors. The version of pre-trained BERT model we used is BERT-BASE which has 12 layers of transformer blocks and the hidden size is equal to 768. It implies that the dimension of the output embedding vectors is equal to 768. We choose not to cover too much details of BERT as it has been elaborated on its website [2].

To boost the performance of our system by making this model better fit our data, a fine-tuning step is conducted before extracting the word embedding vectors from pre-trained model. And the fine-tuning parameters are given in Section 3.2.

2.3 Feature processing via CNN

CNN is originally invented for tackling the issues in the field of computer vision, while various C-

[1]github.com/cbaziotis/ekphrasis

[2]https://github.com/google-research/bert

NN models have subsequently been proven to be effective for many NLP tasks (Kalchbrenner et al., 2014; dos Santos and Gatti, 2014). In our work, we train a single layer CNN on the word embedding vectors drawn from BERT model. Let $e_i \in \mathbb{R}^k$ be the k-dimensional word embedding vector corresponding to the i-th word in the sentence. Then the vector representation of a sentence is denoted as Eq.1:

$$e_{1:n} = [e_1 \parallel e_2 \parallel e_3 \parallel ... \parallel e_n], \tag{1}$$

where n represents the length of sentences, and $\parallel$ denotes concat operation which can maintain the order of words in text. After that a filter involved in one-dimensional convolution operation is defined as $\mathbf{w} \in \mathbb{R}^{m \times k}$, then the convolution process can be defined as a function as Eq.2 (Kim, 2014):

$$p_i = f(\mathbf{w} \cdot e_{i:i+m-1} + b), \tag{2}$$

where p_i is a scalar which stands for the new local feature generated by a filter from a window of words $e_{i:i+m-1}$, in other words, only i-th to $i + m - 1$-th words have been taken into consideration when generate i-th local feature. m is filter size which denotes m words is taken into calculation when generating a local feature. b is a bias and f is an activation function, it is $tanh$ exactly in our system. Finally, there are $n - m + 1$ local features generated by a filter totally. Those local features can be concatenated as a global feature P:

$$P = [p_1 \parallel p_2 \parallel p_3 \parallel ... \parallel p_{n-m+1}], \tag{3}$$

where $P \in \mathbb{R}^{n-m+1}$ represents a feature map generated by a filter. Then a *max-over-time maxpooling* operation (Collobert et al., 2011) is applied to the feature maps which means that only the maximum value of P is reserved. If there are N_f filters, then N_f maximum values is generated through *maxpooling* operations. Those values can be organized as a new vector $Q \in \mathbb{R}^{N_f}$ as Eq.4:

$$Q = [max(P_1) \parallel max(P_2) \parallel ... \parallel max(P_{N_f})], \tag{4}$$

2.4 Dense layers

The pooling layer is followed by two dense layers with different number of neurons. *Dropout* (Hinton et al., 2012) is utilized to alleviate overfitting problem before the first dense layer. And we have tried different dropout rates to search the best configuration. Firstly, the output of pooling layer Q is fed into the first dense layer with 200 hidden neurons. The activation function of this dense layer is *relu* (Xu et al., 2015). Next is the second dense layer with two neurons and the corresponding activation function is *softmax*. Final output is the probability of which class the sample belongs to.

3 Experiments

3.1 Dataset

Classes	Train set	Trial Test set	Test set
0 (non-suggestions)	6415	296	746
1 (suggestions)	2085	296	87

Table 1: Data distribution

The available dataset released by organizer is split into three parts: train set, trial test set and test set. The positive and negative sample distribution of each part is described as Table 1. Apparently, there is class imbalance that the number of negative samples overwhelms the number of positive samples both in training set and test set. So for experiments, we fuse the train set and trail test set into a larger training set, and then split 10% samples as validation set randomly (8183 samples for training and 909 samples for validation). We train our model on the train set, tune the model parameters on the validation set, evaluate the model performance on the test set.

Model	Dropout	Macro average F1
Baseline	-	0.2676
Word2vec+CNN	0.4	0.3789
our model	0.1	0.7459
	0.2	0.7236
	0.3	0.7407
	0.4	0.7309
	0.5	0.7179
	0.6	0.7368
	0.7	0.7029

Table 2: The performance comparison.

Classes	Precision	Recall	F1 score
0 (non-suggestions)	0.98	0.96	0.97
1 (suggestions)	0.70	0.79	0.75

Table 3: The classification accuracy of different classes.

3.2 Experiment Results

As mentioned in Section 2.2, we conduct fine-tuning operation before extracting the word embedding vector from *BERT*. For fine-tuning, most hyperparameters are the same as the parameters

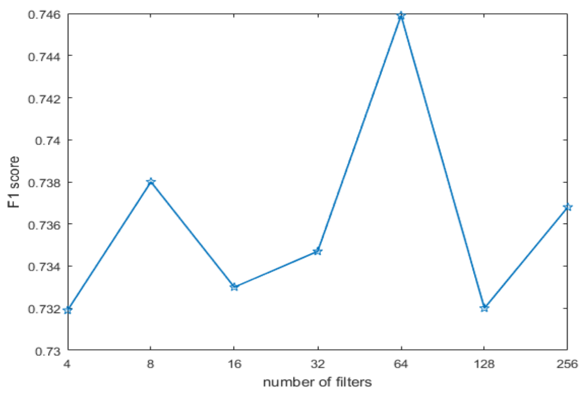

Figure 2: The impact of the number of filters.

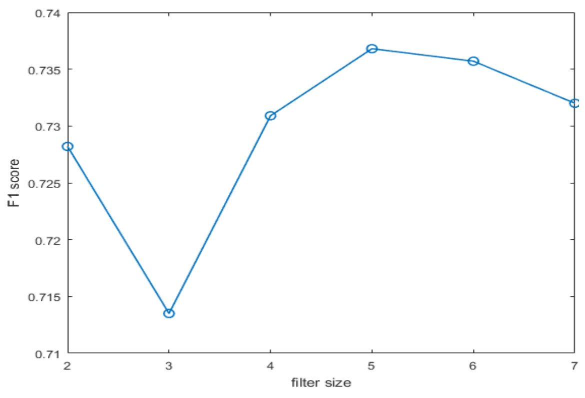

Figure 3: The impact of filter size.

of pre-trained model, with the exception of batch size, learning rate and number of training epochs. The mini-batch size is set at 32 and learning rate is set at 5e-5, the number of training epochs is configured as 5. The maximal length of sentences is configured as 50 and if the length of a sentence is less than 50, it will be padded with zero; otherwise, it will be truncated from the tail. For the CNN component, the filter size m is configured as 5 and the number of filters N_f is 64. And the dropout rates we have tried are ranged from 0.1 to 0.7 with a step of 0.1. The experimental results are shown in Table 2. In order to prove the effectiveness of word embeddings derived from BERT, we also employ Word2vec as comparison and the corresponding best dropout rate is 0.4. Obviously, no matter what the dropout rate is, our model consistently outperforms other models. And the best dropout rate of our model is around 0.1 for abovementioned configuration. While the best dropout rate may vary with other parameter configuration like filter size and the number of filters.

Table 3 shows the Precision, Recall and F1 score in term of different classes. Obviously, the system performance on negative samples is better than the performance on positive samples which is consistent with our intuition for the reason that the number of negative samples overwhelms the number of positive samples. Therefore, our system can learn more features of negative samples, which can help it recognize those negative samples accurately.

We also investigate the impact of the number of filters N_f on the classification performance with fixing the filter size m at 5 and the dropout rate at 0.1. The experimental result is shown as Figure 2. This model yields the best performance when the number of filters is equal to 64. Less filters cannot capture enough information while too many filters can result in information redundancy to some extent.

The impact of filter size on classification accuracy is shown as Figure 3. The most suitable filter size which means the window size of convolutional operation is $m = 5$. It is difficult for this system to catch the global semantic information if the window size is too small. While some local semantic information would be ignored if the window size of filter is too large. Hence, choosing a filter with moderate size is helpful for the performance improvement.

4 Conclusions

In this paper, we have proposed a neural model based on the pre-trained BERT model to deal with suggestion mining task. Our system can learn the representation of sentences or words effectively by leveraging BERT. Then the representations extracted from BERT is fed into a simple CNN layer. Experimental results show that our system is efficient on the given dataset.

As for future work, it is of necessity to tackle the imbalance between positive samples and negative samples through oversampling or undersampling. And we intend to study some innovative ways to incorporate BERT model like extracting not only the output of the last transformer layer but also the output of different transformer layers and integrating them with different weights.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (61562090).

References

Chris Alberti, Kenton Lee, and Michael Collins. 2019. A bert baseline for the natural questions. *CoRR*, abs/1901.08634.

Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. 2017. Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 260–269.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *ACL*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

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

Sapna Negi and Paul Buitelaar. 2015. Towards the extraction of customer-to-customer suggestions from reviews. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2159–2167.

Sapna Negi, Tobias Daudert, and Paul Buitelaar. 2019. Semeval-2019 task 9: Suggestion mining from online reviews and forums. In *Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval-2019)*.

Sapna Negi, Maarten de Rijke, and Paul Buitelaar. 2018. Open domain suggestion mining: Problem definition and datasets. *arXiv preprint arXiv:1806.02179*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*.

Cicero dos Santos and Maira Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Ye Zhang and Byron Wallace. 2015. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.