Bert & TextCNN on SemEval-2017 Task4 subTask A: Sentiment Analysis in twitter

Huiqiang Jiang
{1801210840}@pku.edu.cn

Abstract

In this paper, we present two deep-learning models that competed at SemEval-2017 Task 4 "Sentiment Analysis in Twitter subTask A". We used Text Convolutional Neural Network (TextCNN), on encoder end using a pre-trained word embeddings on a big collection of Twitter from RepLab 2013 Dataset. Also, we using a text processing tool build by DataStories which performs tokenization, word normalization, segmentation, and spell correction. And we also evaluation in Bert and some traditional machine learning like logistic regression(LR), Support Vector Machines(SVM). The evaluation shows that embedding, text processor is most important, and neural network model better than traditional machine learning model. We achieve a 69.39% Recall score on Task A, Macro-F1 is 69.91%, accuracy is 70.03%. Every score is better than the rank 1 in the competition. In the future, we want to combine the TextCNN with Bert. All code is available to the research community.

1 Introduction

Sentiment Analysis is a typical task in Natural Language Processing (NLP) domain. It is easy to start with some very simple method (e.g. positive - negative words counting). But If you want to get a good result, It is difficult. Because Twitter dataSet is informal and creative writing style. It means that you will encounter lots of out-of-vocabulary (OOV) situation. So maybe embedding is a very important point in the task. The practical applications of this task are wide, from personalized recommendation, chatbot, to public opinion monitoring(e.g. Presidential debates, World Cup, etc.).

In the last few years, deep learning techniques have significantly out-performed traditional methods in several NLP tasks, and sentiment analysis is no exception to this trend. Especially Bert published on last year. Two of the most popular deep learning techniques for sentiment analysis are TextCNNs and Transform model like Bert. In this paper, we evaluation both two model.

In this paper, we present two deep-learning systems that competed at SemEval-2017 Task 4. Our first model base on a single-layer CNN model, equipped with 3 filters. The other model base on Bert which is a pre-training and fine-tuning model by the multi-layer transform.

- We evaluation this task in some traditional model, like SVM, LR. In this model, we get some decent result. We also used a text processor to change text format, like change 'HTTP://' to '<url>'. It reduces the bag of word size, and reduce the out-of-vocabulary (OOV) situation.
- A pre-train embedding basing on RepLab 2013 Dataset. And we train this embedding in two way, word2vec and fastText. We find that the model pre-train by fastText work is better than pre-training by word2vec in LogisticRegression. So we choose the embedding by fastText as encoder tools.
- And we train model in textCNN & Bert. we also evaluate the effect of pad position. All code can be found in github.[1]

2 System Description

In this section, we describe our text processor and illustrate how we leverage pre-training word embedding and how to encode in our model.

2.1 Text Processor

The text processor reference by DataStroies (Baziotis et al., 2017)[2]. In this processor, we change

[1] github.com/iofu728/SemEval2017-Task4-SentimentAnalysis.

[2] github.com/cbaziotis/ekphrasis

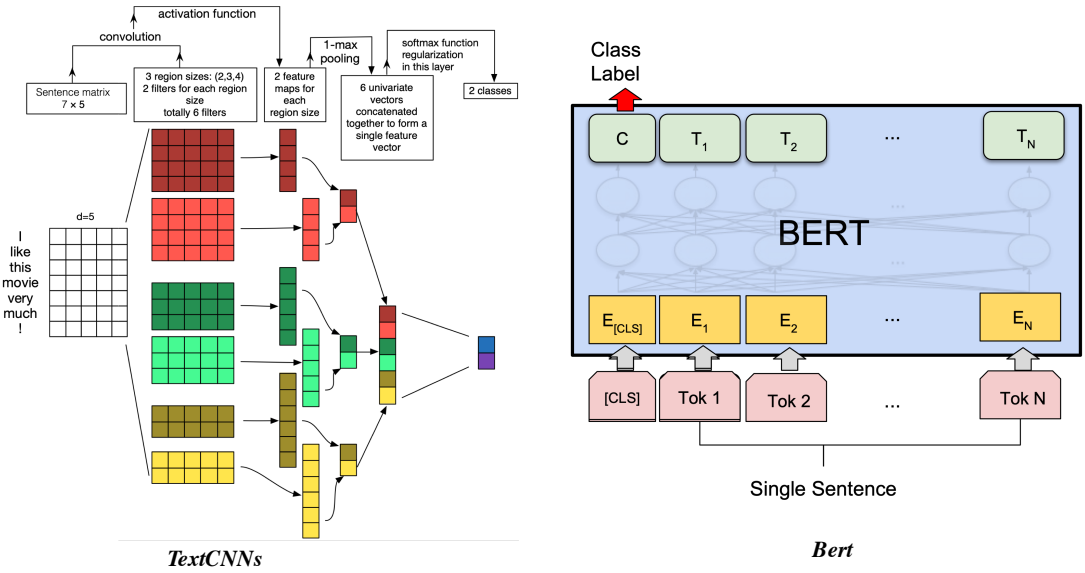

Figure 1: Two model in this paper, TextCNN & Bert

emoji to a semantic word, change URL to label '<url>'. And we also lowercase all words and do some simple spell correction. This work can reduce the OOV situation. Before using the text processor, the ratio of OOV is 76.75%(135619/177692). The ration of OOV using the text processor is 9.60%(4491/45578). From this data, the text processing also reduces the word num.

After that, we alignment the sentences to max sentences line. In this processing, we add the pad to before first, after first, before the last one, after the lastest one respectively.

2.2 Pre-training Embedding

we pre-training the word embedding obtained from RepLab 2013 Dataset (Amigó et al., 2013)[3] about 9GB data. We train for word2vec & fastText, having text processor, no text processor and respectively evaluation in LogisticRegression model.

2.3 TextCNN

We first test in TextCNN model. Its architecture is almost identical to CNN. The input of the network are the tweets, which are tokenized into words which disposed of by text processor. And each word is encoded by a word vector representation, i.e. a word embedding. We also follow add zero-padding strategy such that all tweets have the same matrix dimension $X \in \mathbb{R}^{s' \times d}$, where we chose $s' = MaxSen.$=64(using TextProcessor)/35(no TextProcessor). We then apply several convolution operations of various sizes to this matrix. A single convolution involves a filtering matrix $w \in \mathbb{R}^{h \times d}$ where h is the size of the convolution, meaning the number of words it spans. The convolution operation is defined as

$$c_i = f\left(\sum_{j,k} w_{j,k} \left(X_{[i:i+h-1]}\right)_{j,k} + b\right) \quad (1)$$

where $b \in \mathbb{R}$ is a bias term and f(x) is a non-linear function, which we chose to be the 'relu' function. The output $c \in \mathbb{R}^{s'-h+1}$ is, therefore, a concatenation of the convolution operator over all possible window of words in the tweet. We can use multiple filtering matrices to learn different features, and additionally, we can use multiple convolution sizes to focus on smaller or larger regions of the tweets. In practice, we used the filter size $[6, 7, 8]$ and we used a total of 256 filtering matrices for each filter size.

[3] nlp.uned.es/replab2013/

Embedding Model	Text Processor	Recall	Precision	Macro-F1	Accuracy	Pos.	Neu.	Neg.
no	no	45.70	43.12	44.38	48.18	43.03	**74.68**	11.66
word2Vec	no	44.87	42.48	43.65	51.37	60.81	57.27	9.37
fastText	no	43.87	42.04	42.93	46.60	44.97	72.93	8.21
no	ekphrasis	61.07	62.15	61.61	62.34	**64.00**	64.83	57.63
word2vec	ekphrasis	61.49	62.35	61.92	**64.37**	62.52	68.72	55.80
fastText	ekphrasis	**62.67**	**62.07**	**63.36**	63.81	63.03	61.58	**67.60**

Table 1: Comparison between different Embedding mode & Text Processor using LibLinear logistic regression model on subtask A data (in %)

Sentences1:	**Indian Mi Fans**. Are Are Are **you** ok?
Sentences2:	**Indian Mi Fans**. Are **you** ok?
Before entity1:	[PAD] [PAD] **Indian Mi Fans**. Are Are Are **you** ok?
	[PAD] [PAD] [PAD] [PAD] **Indian Mi Fans**. Are **you** ok?
After entity1:	**Indian Mi Fans**. [PAD] [PAD] Are Are Are **you** ok?
	Indian Mi Fans. [PAD] [PAD] [PAD] [PAD] Are **you** ok?
Before entity2:	**Indian Mi Fans**. Are Are Are [PAD] [PAD] **you** ok?
	Indian Mi Fans. Are [PAD] [PAD] [PAD] [PAD] **you** ok?
After entity2:	**Indian Mi Fans**. Are Are Are **you** ok? [PAD] [PAD]
	Indian Mi Fans. Are **you** ok? [PAD] [PAD] [PAD] [PAD]

Table 2: Pad example (MAX sentences size = 12)

2.4 Bert

BERT is one of the key innovations in the recent progress of contextualized representation learning (Peters et al., 2018; Howard and Ruder, 2018; Radford et al., 2018; Devlin et al., 2018). The idea behind the progress is that even though the word embedding layer (in a typical neural network for NLP) is trained from large-scale corpora, training a wide variety of neural architectures that encode contextual representations only from the limited supervised data on end tasks is insufficient. Unlike ELMo (Peters et al., 2018) and ULMFiT (Howard and Ruder, 2018) that are intended to provide additional features for a particular architecture that bears human's understanding of the end task, BERT adopts a fine-tuning approach that requires almost no specific architecture for each end task. This is desired as an intelligent agent should minimize the use of prior human knowledge in the model design. Instead, it should learn such knowledge from data. BERT has two parameter intensive settings:

- **BERT$_{\mathbf{BASE}}$**: L=12, H=768, A=12, Total Parameters=110M
- **BERT$_{\mathbf{LARGE}}$**: L=24, H=1024, A=16, Total Parameters=340M

3 Experiments

In this section, we evaluate in embedding, text processor, model.

3.1 Embedding

Word embedding almost the important thing in NLP task. So we take some word embedding method and train dataSet to tune the effect of word embedding.

First of all, the train dataSet of competition is so small that the effect of training word embedding is bad. So we import some outer dataSet to optimization the effect of word embedding. The domain of our task is about scientific papers. So, we load dataset on RepLab 2013 Dataset.

We also do some work on different word embedding methods, like word2vec, fastText. FatsText do the best job in our experiment. Bert doesn't have a good effect on our task. We think it may be caused by the difficult word style between pre-train model and twitter dataSet.

Text Position	Pad position	Recall	Precision	Macro-F1	Accuracy	Pos.	Neu.	Neg.
no	before 0	61.39	49.50	54.81	57.57	41.76	**85.10**	21.64
no	after 1	64.77	51.93	57.64	61.10	53.97	82.90	18.91
no	before -1	63.08	52.18	57.11	59.20	45.72	83.17	27.64
no	after end	62.05	57.54	59.71	63.00	57.58	76.85	38.19
ekphrasis	before 0	61.61	**65.62**	63.55	63.70	69.12	56.54	**71.21**
ekphrasis	after 1	**66.70**	59.73	63.02	66.27	60.88	80.92	37.37
ekphrasis	before -1	64.71	61.61	63.12	65.23	**77.62**	61.70	45.51
ekphrasis	after end	65.36	62.37	**63.83**	**66.93**	70.02	71.12	45.98

Table 3: Comparison between different Text Processor & Pad position using TextCNN model on subtask A data (in %)

Text Processor	Recall	Precision	Macro-F1	Accuracy	Pos.	Neu.	Neg.
no	68.52	**70.54**	69.52	69.48	**73.39**	66.21	**72.03**
ekphrasis	**69.32**	70.51	**69.91**	**70.03**	71.75	**68.52**	71.27

Table 4: Comparison between different Text Processor using Bert on subtask A data (in %)

3.2 TextCNN

Word embedding almost the important thing in NLP task. So we take some word embedding method and train dataSet to tune the effect of word embedding.

First of all, the train dataSet of competition is so small that the effect of training word embedding is bad. So we import some outer dataSet to optimization the effect of word embedding. The domain of our task is about scientific papers. So, we load dataset on RepLab 2013 Dataset.

We also do some work on different word embedding methods, like word2vec, fastText. FatsText do the best job in our experiment. Bert doesn't have a good effect on our task. We think it may be caused by the difficult word style between pre-train model and twitter dataSet.

The lens between two entities is different in dataSet. TextCNN needs every sentence to have the same lens, so a naive idea is to change the pad position. We can put '[PAD]' before the first word, after the first word, before the latest word, after the latest word. The result of 4 methods should be different. So we take some experimentation to explore this problem. We use TextCNN with Train dataSet, use 256 filters, filter size=[6,7,8], embedding size=300, learning rate = 0.0003, batch size = 64, decay step=1000..

The evaluation shows that padding position is a vital parameter in our model. In all subTask, we found that adding pad before entity1 is a good way to improve the performance of our model.

3.3 Bert

Compare with pre-training Bert with fine-tune Bert, the score significantly Improve. The position embedding & Transformer construction work well to obtain information in context. In the future, we want to combine TextCNN & Bert, using TextCNN instead of Transform to evaluate the effect of Transform.

4 Conclusion

In this paper, we introduce two deep-learning sentiment analysis model. FastText is the best way in our jobs on the embedding layer. Both TextCNN and work well in this task than traditional mechanical learning. And Bert can get more information on context. The pad position also has some effect on this model. We achieve a 69.39% Recall score on Task A, Macro-F1 is 69.91%, accuracy is 70.03%. Every score is better than the rank 1 in the competition. In the future, we want to combine the TextCNN with Bert. It may have a more fantastic effect.

References

E. Amigó, J. Carrillo de Albornoz, I. Chugur, A. Corujo, J. Gonzalo, T. Martín, E. Meij, M. de Rijke, and D. Spina. 2013. Overview of RepLab 2013: Evaluating Online Reputation Monitoring Systems.

In *Proceedings of the Fourth International Conference of the CLEF initiative*, pages 333–352.

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.

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*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

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*.