IKM at SemEval-2017 Task 8: Convolutional Neural Networks for Stance Detection and Rumor Verification

Yi-Chin Chen, Zhao-Yang Liu, Hung-Yu Kao
Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan, ROC
{kimberycc,kjes89011}@gmail.com
hykao@mail.ncku.edu.tw

Abstract

This paper describes our approach for SemEval-2017 Task 8. We aim at detecting the stance of tweets and determining the veracity of the given rumor. We utilize a convolutional neural network for short text categorization using multiple filter sizes. Our approach beats the baseline classifiers on different event data with good F_1 scores. The best of our submitted runs achieves rank 1st among all scores on subtask B.

1 Introduction

Rumors in social networks are widely noticed due to the broad success of online social media. Unconfirmed rumors usually spark discussion before being verified. These have created cost for society and panic among people. Rather than relying on human observers to identify trending rumors, it would be helpful to detect them automatically and limit the damage immediately. However, identifying false rumors early is a hard task without sufficient evidence such as responses, retweet and fact checking sites. Instead of propagation structure, context-level patterns are more obvious and useful for the identification of rumors at this stage – in particular, observing the different patterns of stances amongst participants (Qazvinian et al., 2011).

Recent research has proposed a 4-way classification task to encompass all the different kinds of reactions to rumors (Arkaitz et al., 2016). A schema of classifications including supporting, denying, querying and commenting (SDQC) is applied in SemEval2017 Task 8.

In this paper, we describe a system for stance classification and rumor verification in tweets. For the first task, we are given tree-structured conversations, where replies are triggered by a source tweet. We need to categorize the replies into one of the SDQC categories by reply-source pairs. The second task is about rumor verification. Our system is for the closed variant – which means the veracity of a rumor will have to be predicted solely without external data.

It is a challenging NLP task. Statements containing sarcasm, irony and metaphor often need personal experience to be able to infer their broader context (Kreuz and Caucci, 2007). Furthermore, lots of background knowledge is required to do the fact checking (Reichel and Lendvai, 2016).

In this paper, we develop convolutional neural network models for both tasks. Our system relies on a supervised classifier, using text features of different word representation methods such as learning word embedding through training and pre-trained word embedding model like GloVe (Pennington et al., 2014). The experiment section presents our results and discusses the performance of our work.

2 Related Work

Rumor verification from online social media has developed into a popular subject in recent years. The most common features were proposed by Castillo (2011) who classified useful features into four categories: message-based features, user-based features, topic-based features, and propagation-based features. However, this approach is limited because of the data skew problem when false rumors are less common. Thus, most existing approaches attempt to classify truthfulness by utilizing information beyond the content of the posts – propagation structure, for example. Ke Wu (2015) et al., proposed a novel message propagation pattern based on the users who transmit this message. But most of these features are available only when the rumors have been responded to by many users. Our task, on the other hand, is to do the initial classification on content features which are available much earlier.

3 System Overview

Our system employs a convolutional neural network mainly inspired by Kim (2014). We chose models by testing on LOO (Leave One Out) validation performance. LOO can be simply explained as that we test on each conversation thread by retraining models on the other threads. In the following section, our CNN Tweet Model is briefly explained.

3.1 Data Preprocessing

Before applying the models, we need to do some transforms of the irregular input text. At first, we remove URLs and username with '@' tags that do not contribute to sentiment analysis. In this case, URLs and usernames are considered as noise without external data. Furthermore, we convert all letters to lower case. Besides removal, it is worth mentioning that we leave important clues such as hashtags and some special characters. Question marks and exclamation marks, for example, have proven to be helpful (Zhao, 2015).

3.2 Convolutional Model

There are two steps for the process of encoding tweets into matrices that are then passed to the input layer. This model is illustrated in Figure 1. First, we use word embedding to convert each word in the tweet into a vector. We randomly initialize the word embedding matrix. Each row of this matrix is a vector that represents a word in the vocabulary. Then we learn the embedding weights during the training process. Second, we concatenate these word vectors to produce a matrix representing the sentence. In the matrix, each row

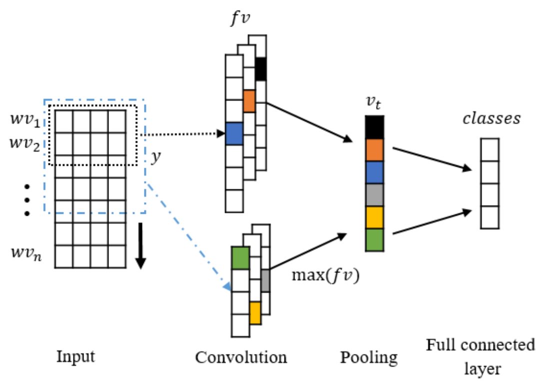

Figure 1: Architecture of Word-Embedding Convolutional Model

represents one word in the tweet as follows:

$$tm = \begin{bmatrix} wv_1 \\ wv_2 \\ \vdots \\ wv_n \end{bmatrix}_{n\times d} \tag{1}$$

Where tm is a word matrix formed by the concatenation of each word vector.

In the convolutional layer, we use tm as input and select a window size y to slide over the matrix. To extract local features in the region of the window, a filter matrix $fm \in R^{y\times d}$ is used to produce element-wise multiplication and non-linear operations on the matrix values in the window at every position. The following is an example of this operation:

$$el_i = g\left(fm \cdot \begin{bmatrix} wv_i \\ \vdots \\ wv_{i+y-1} \end{bmatrix} + b\right) \tag{2}$$

Where fm is the filter matrix. The values of the filter matrix will be learned by the CNN from the training process. b is the bias term, g is the non-linear function, and el_i is an element of a local feature vector. After we slide the window through the whole matrix, we get a local feature vector of the input tweet as:

$$fv = [el_1, el_2, \cdots, el_{n-y+1}] \tag{3}$$

Where $fv \in R^{n-y+1}$ is a local feature vector with *n-y+1* elements.

For the purpose of dealing with continuous n words which may represent special meaning in NLP (e.g. "Boston Globe"), we use multiple window sizes to produce different feature vectors. Thus, the idea of a different window size applied to capturing features is similar to n-grams. Meanwhile, we use different filter matrices to extract different local features of the tweet in each window.

A pooling layer is used for simplifying the information of the output from the convolutional layer. We extract the maximum value from each local feature vector to form a condensed representation vector. For every local feature vector, only the most important feature is extracted and noise is ignored. After the max-pooling operation, we can concatenate all maximum values of each column as follows:

$$v_t = \begin{bmatrix} max(fv_1) \\ \vdots \\ max(fv_m) \end{bmatrix} \tag{4}$$

Where v_t is the global feature vector representing the tweet.

Through the pooling layer, if we use the same window size and filter matrix on different tweets, we can make sure the global feature size is fixed.

For classification, we feed the global feature vectors of the tweet into a fully connected layer to calculate the probability distribution. A softmax activation function is applied as follows:

$$P(y = i|v_t, b) = \frac{e^{w^T{}_i v_t + b_i}}{\sum_{i'=1} e^{w^T{}_{i'} v_t + b_{i'}}} \tag{5}$$

Where v_t is the input vector, $w^T{}_{i'}$ is the i'-th column of weight matrix W. With the probabilities over the four classes, we take the class with the maximum value as the label for the given input tweet.

4 Tasks and Model Training

During the training phase, our CNN model automatically learns the values of its filters based on the task.

In task A, the tweets are classified into four categories: supporting, denying, querying and commenting. We defined the ground truth vector *p* as a one-hot vector. The parameter *d* used in the word embedding is 128. The number of filters in the convolutional layers is 128. The probability of dropout is set to 0.5. Adam Optimization algorithm is used to optimize our network's loss function. Moreover, there are three filter region sizes in our system: 2, 3 and 4, each of which has 2 filters.

In order to deal with the imbalance of classes in the data, balanced mini-batching was applied. In the statistics, more than 64% of the instances belong to the commenting class. We chose 16 instances with each class from training set randomly, which means that there are 64 instances in a batch.

A voting scheme is applied to decrease the uncertainty of training on randomly selected samples. We trained 5 models to predict the same testing data and took a vote for the final prediction. By performing training multiple times independently we achieved more robust results.

In subtask B, most of the parameter settings were the same as in Task A. Because the output classes are rumor and non-rumor, we discard the label "unverified". In addition, we use the probability in section 3.2 to define the credibility of our answer c. The credibility in the interval [0, 1] is normalized as:

$$c = \frac{\max(P(y=0,1|v_t,b))}{\sum_{i=0,1} P(y=i|v_t,b)} \tag{6}$$

5 Evaluation

We conduct experiments using the rumor datasets annotated for stance (Zubiaga et al., 2016). The statistics of the datasets are shown in Table 1. For subtask B, conversation threads are not available for the participants and the use of external data is forbidden on the closed variant.

Subtask A				
Stance	Support	Deny	Query	Comment
Training	841(20%)	333(8%)	330(8%)	2734(65%)
Testing	94(9%)	71(7%)	106(10%)	778(74%)
Subtask B				
Veracity	True		False	unverified
Training	127(47%)		50(18%)	95(35%)
Testing	8(40%)		12(60%)	0

Table 1: Statistics of datasets for subtask A and B.

5.1 Baselines

We compare our result with Lukasik's (2016) in Table 2. We follow their LOO settings and test on the same dataset. The report includes accuracy (Acc) and macro average of F_1 scores across all labels (F_1) from Lukasik's baseline.

The results show our deep learning model is the best method in terms of F1 score. Especially, the CNN model beats all the other methods. While the RNN method is not performing well on this task. Another issue is the GloVe embedding – the pretraining model sometimes lacks some of the vocabulary from new events. Nevertheless, GloVe is still competitive with the CNN method for the Ferguson event.

Event	Ottawa		Ferguson	
	Acc	F_1	Acc	F_1
GP	62.28	42.41	64.31	32.9
Lang. model	53.2	42.66	49.56	34.35
NB	61.76	40.64	62.05	31.29
HP Approx.	67.77	32.29	68.44	25.99
HP Grad.	63.43	42.4	63.23	33.14
CNN	61.74	**44.9**	62.31	36.49
CNN(GloVe)	59.61	38.87	63.03	**39.48**
RNN(GloVe)	52.49	38.66	51.49	32.52

Table 2: Accuracy and F_1 scores for different methods across datasets. The upper lines of the results are our baseline.

Window Sizes	Precision	Recall	F_1
3	0.39	0.42	0.40
3,4	0.43	0.42	0.43
2,3,4	0.43	0.40	0.42
3,4,5	**0.45**	**0.45**	**0.45**
2,3,4,5	0.44	0.45	0.44

Table 3: results of using different window sizes.

5.2 Window Sizes for Filters

Table 3 lists the results of using different window sizes for the filters in the tweet encoding process. We set different window sizes to observe the impact. The experiment was performed with the same settings as in section 5.1 for the Ottawa event. We obtain the best performance when the window size combination is (3, 4, 5). Different window sizes 2, 3 and 4 correspond to the encoding for the bigrams, trigrams and four-grams of the tweets respectively. We can see that the performance decrease slightly with the window size increases. That is, insufficient grams can lose some features while too many grams can bring noise.

5.3 Official Results[1]

Our submission results to the subtask A achieve an accuracy of 0.701. The statistical details of each class are given in Table 4. We notice that the comment stance is the easiest to detect, since they take a large part of the data. The number of query stances are similar to support and deny, while it has much better precision and recall because the features of queries are more obvious. Likewise, there are some negative words in the deny stance as features. However, it is challenging to extract features of supporting which results in a poorer performance.

Stance	Precision	Recall	Accuracy
Support	0.19	0.20	0.20
Deny	0.31	0.07	0.07
Query	0.58	0.45	0.45
Comment	0.78	0.85	0.85

Table 4: Result on test data for subtask A.

Team	Score	RMSE
DFKI DKT	0.393	0.845
ECNU	0.464	0.736
IITP	0.286	0.807
IKM	**0.536**	**0.763**
NileTMRG	0.536	0.672

Table 5: Rank on test data for subtask B.

The rank of subtask B is summarized in Table 5. As we can see our model performs best among the official scores. Our code is available on github for anyone who has interest in further exploration[2].

6 Conclusion

We develop a convolutional neural network system for detecting twitter stance and rumor veracity determination in this paper. Compared with the baseline approach, our system obtains good results on stance detection. In addition, on the test set of SemEval2017 Task8B, we ranked 2nd in the official evaluation run.

Reference

Castillo, Carlos, Marcelo Mendoza, and Barbara Poblete. "Information credibility on twitter. " *Proceedings of the 20th international conference on World wide web*. ACM, 2011.

Wu Ke, Song Yang, and Kenny Q. Zhu. "False rumors detection on sina weibo by propagation structures." *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 2015.

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

Lukasik, Michal, Srijith, P.K, Vu, Duy, Bontcheva, Kalina, Zubiaga, Arkaitz and Cohn. "Hawkes processes for continuous time sequence classification: an application to rumor stance classification in

[1] Results and task detail can be found on http://alt.qcri.org/semeval2017/task8/

[2] https://github.com/kimber-chen/Twitter-stance-classification-by-TensorFlow

twitter." Proceedings of 54th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2016.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Vahed Qazvinian, Emily Rosengren, Dragomir R. Radev, and Qiaozhu Mei. 2011. Rumor has it: Identifying misinformation in microblogs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '11,* pages 1589–1599.

Kreuz, Roger J., and Gina M. Caucci. "Lexical influences on the perception of sarcasm." *Proceedings of the Workshop on computational approaches to Figurative Language*. Association for Computational Linguistics, 2007.

Reichel, Uwe D., and Piroska Lendvai. "Veracity computing from lexical cues and perceived certainty trends." *arXiv preprint arXiv:1611.02590* (2016).

Zhao, Zhe, Paul Resnick, and Qiaozhu Mei. "Enquiring minds: Early detection of rumors in social media from enquiry posts." *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015.

Arkaitz Zubiaga ,Maria Liakata,Rob Procter, Geraldine Wong Sak Hoi, and Peter Tolmie. "Analysing how people orient to and spread rumors in social media by looking at conversational threads." *PloS one* 11.3 (2016): e0150989.

Arkaitz Zubiaga, Elena Kochkina, Maria Liakata, Rob Procter, and Michal Lukasik. "Stance classification in rumors as a sequential task exploiting the tree structure of social media conversations." *arXiv preprint arXiv:1609.09028* (2016)