Computational Linguistics and Chinese Language Processing
Vol. 23, No. 1, June 2018, pp. 1-18

Sentiment Analysis on Social Network: Using Emoticon Characteristics for Twitter Polarity Classification

Chia-Ping Chen*, Tzu-Hsuan Tseng* and Tzu-Hsuan Yang*

Abstract

In this paper, we describe a sentiment analysis system implemented for the semantic-evaluation task of message polarity classification for English on Twitter. Our system contains modules of data pre-processing, word embedding, and sentiment classification. In order to decrease the data complexity and increase the coverage of the word vector model for better learning, we perform a series of data pre-processing tasks, including emoticon normalization, specific suffix splitting, and hashtag segmentation. In word embedding, we utilize the pre-trained word vector provided by GloVe. We believe that emojis in tweets are important characteristics for Twitter sentiment classification, but most pre-trained sets of word vectors contain few or no emoji representations. Thus, we propose embedding emojis into the vector space by neural network models. We train the emoji vector with relevant words that contain descriptions and contexts of emojis. The models of long short-term memory (LSTM) and convolutional neural network (CNN) are used as our sentiment classifiers. The proposed emoji embedding is evaluated on the SemEval 2017 tasks. Using emoji embedding, we achieved recall rates of 0.652 with the LSTM classifier and 0.640 with the CNN classifier.

Keywords: Sentiment Analysis, Polarity Classification, Machine Learning, Neural Network, Word Embedding.

1. Introduction

There has been huge growth in the use of social networks, such as Twitter, in recent years. Many messages are created every day, including various topics, users' comments and views, or current emotions. Sentiment analysis, which predicts the polarity of a message, is one of the research directions on Twitter. A message on Twitter is called a tweet and is allowed to be 140 characters or less. Tweets are highly colloquial. Due to the length constraint, a tweet often

* Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan
E-mail: cpchen@cse.nsysu.edu.tw; fb74123698@gmail.com; kr60903@gmail.com

contains unofficial abbreviations, as well as emoticons and emojis. Figure 1 shows examples of tweets.

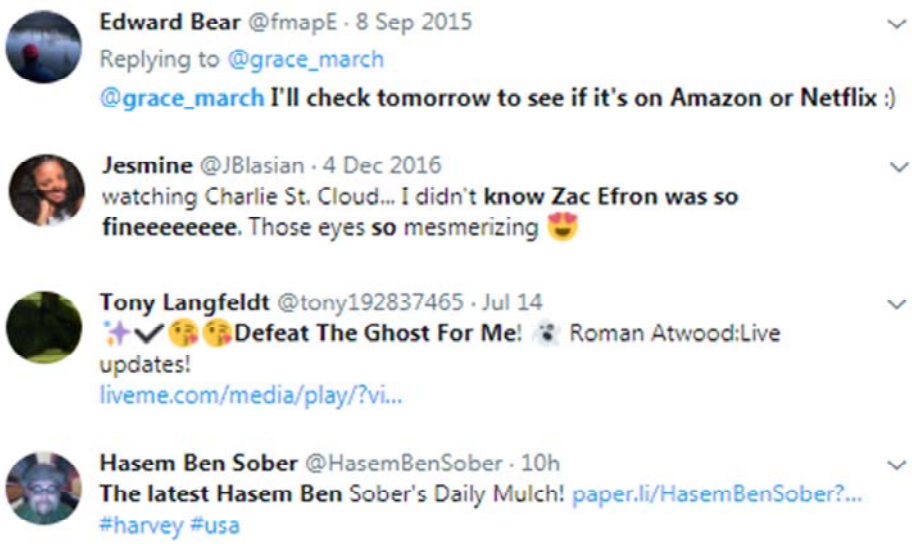

Figure 1. Examples of tweets. Tweets tend to have informal words and syntax.

In the above examples, we can see that emojis are used frequently in Tweets. Some emojis (like 😀) can be considered the natural evolution of emoticons, such as :-) and :D. In addition to facial expressions, emojis can be used for food, flags, animals, *etc*.

Unofficial abbreviations and emojis without corresponding word vectors in tweets can make the sentiment classification task difficult. In this work, we find sentiment features for these unorthodox tokens to get better results in sentiment classification.

Artificial neural networks for machine learning are mathematical models inspired by biological neural systems. Deep learning, which is neural network models based on deep neural networks, has been a very successful method and achieves state-of-the-art performance in many tasks, such as NIST handwritten digit recognition (LeCun, Bottou, Bengio, & Haffner, 1998) and ImageNet image classification (Krizhevsky, Sutskever, & Hinton, 2012). It performs well in natural language processing tasks, such as machine translation (Sutskever, Vinyals, & Le, 2014) and handwriting recognition (Graves *et al*., 2009).

For sentiment analysis, deep learning-based approaches have performed well in recent years. For example, convolution neural networks (CNN) with word embedding have been implemented for text classification (Kim, 2014), and they have achieved state-of-the-art results in SemEval 2015 (Severyn & Moschitti, 2015). SemEval 2017 Task 4 is sentiment analysis in Twitter, which is further divided into five subtasks: message polarity classification (Subtask A), topic-based message polarity classification (Subtasks B-C), and tweet

quantification (Subtasks D-E).

Most of the participants in SemEval who adopt deep learning collect millions of tweets to train word-embedding models. The top system of SemEval 2017, which achieved 0.681 of average recall, used 100 million unlabeled tweets to pre-train word-embedding models (Cliché, 2017). In contrast, our goal in this work is to achieve sound performance without a large amount of external data.

In this paper, we describe our system for SemEval 2017 Task 4 (Subtask A) for message polarity classification (Rosenthal, Farra, & Nakov, 2017). Given a message, the system decides whether the message is of positive, negative, or neutral sentiment. We extend our previous work on SemEval 2017 (Yang, Tseng, & Chen, 2017). Our system consists of data pre-processing, word embedding, and classifiers. Data pre-processing includes normalization and hashtag segmentation. We consider the importance of emojis for sentiment analysis. In addition to using pre-trained word vectors, we train the emoji vector by the neural network. For the classifiers, we choose RNN-based and CNN models. We have achieved average recall rates of 0.652 with the LSTM-based classifier, and 0.640 with the CNN-based classifier.

Our contributions are described as follows.

- We propose neural network models for emoji embedding and investigate the effects of using emoji vectors in the classifiers. Through experiments, we find that emoji vectors can improve the accuracy of prediction for the positive and negative classes.
- Besides adding emoji vectors in the system, data pre-processing is critical to the improvement of the average recall rate from 0.610 to 0.652 with the LSTM classifier. Data pre-processing is important for Twitter sentiment analysis because textual data on Twitter is informal. In particular, the effect of hashtag segmentation is the most significant.

This paper is organized as follows. In Section 2, we describe our system, consisting of data pre-processing, word embedding, emoji embedding, and classifiers. In Section 3, we introduce data in experiments, network settings, and tools. In Section 4, we present the evaluation results, along with our comments. In Section 5, we conclude and discuss future works.

2. Related Work

There has been considerable research in the field of sentiment analysis. Past research mostly has focused on long text. Pang, Lee and Vaithyanathan (2002) analyzed the performance on movie reviews using machine learning algorithms and used star ratings as polarity signals in their training data. In recent years, there have been many research projects of sentiment analysis on social networks like Twitter. Go, Bhayani and Huang (2009) used distant learning to acquire more sentiment data. Their training data consisted of tweets with emoticons, which

can be used as noisy labels. They constructed models with Naïve Bayes, Maximum Entropy, and Support Vector Machines (SVM), and they concluded that SVM outperforms the other models.

Deep learning has gained much attention in classification of Twitter text data, due to its huge success in speech recognition and computer vision. Among the top systems of SemEval, Severyn and Moschitti (2015) proposed a parameter initialization method for CNN. They used an unsupervised neural language model to initialize word embeddings that were fine-tuned by a distant supervised corpus. The pre-trained parameters were used to initialize the CNN model. Deriu, Gonzenbach, Uzdilli, Lucchi, De Luca and Jaggi (2016) utilized large amounts of data with distant supervision to train an ensemble of two-layer convolutional neural networks whose predictions were combined using a random forest classifier. Cliché (2017) used CNN models and bi-directional LSTM models. They pre-trained word embedding and fine-tuned it using distant supervision. They trained their models on Twitter data where embedding was fine-tuned again and finally combined several CNNs and LSTMs to get better performance.

Emojis are an important feature in tweets. Many studies have analyzed and trained emojis, such as Zhao, Dong, Wu and Xu (2012) and Barbieri, Ronzano and Saggion (2016). Zhao *et al*. (2012) built a system, which was the first system for sentiment analysis of Chinese tweets in Weibo. They mapped 95 emojis into four categories of sentiment. Their system employs the emojis for the generation of sentiment labels for tweets and builds an incremental learning Naïve Bayes classifier for the categorization of four types of sentiments. Barbieri *et al*. (2016) studied Twitter emojis with embedding models. They retrieved ten million tweets posted by USA users, and they made vector models of both words and emojis using several skip-gram word-embedding models.

3. Method

The system we implement for sentiment classification is shown in Figure 2. In data pre-processing, we normalize data sets to decrease the data complexity and increase the coverage of the word vector inventory. In word embedding, we utilize pre-trained word vectors provided by GloVe (Pennington, Socher, & Manning, 2014) and we train emoji embedding by neural networks. The models of LSTM and CNN are used as our sentiment classifiers.

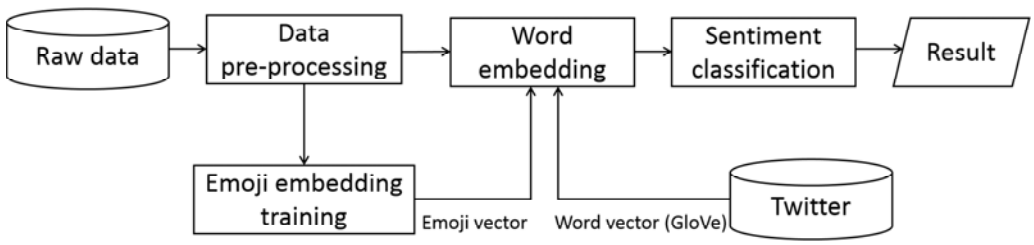

Figure 2. Our Sentiment System. We propose to add embedded emoji vectors for better sentiment classification.

3.1 Data Pre-processing

All the data used for training the emoji embedding and for training the sentiment classification models undergo a series of data pre-processing. First, we use a tokenizer to split a tweet into words, emoticons, and punctuation marks. Happytokenizer[1] is the tokenizer we use for text processing. Then, we replace URLs and USERs with normalization patterns <URL> and <USER>, respectively. All uppercase letters are converted into lowercase letters. The above pre-processing is called basic pre-processing. Next, we perform further data pre-processing based on basic pre-processing. The further data pre-processing is described as follows.

3.1.1 Emoticon Normalization

Tweets often contain a variety of emoticons, and some emoticons do not correspond to any pre-trained word vector. To reduce complexity, we normalize similar emoticons to the same token, as described in Table 1.

Table 1. Examples of emoticon normalization. We normalize similar emoticons into four categories, which are <smile>, <sadface>, <neutralface>, and <heart>, respectively.

Emoticon	Normalization
:), (:, :-), (-:, :D, :-D, ;), =), (=, =D	<smile>
:(,):, :'(,)':, =(,)=, :-(,)-:	<sadface>
:|, |:, =|, |:	<neutralface>
<3	<heart>

For example, in the case of <neutralface> category, we find :|, |:, =|, |:, and then replace them by the token <neutralface>. Thus, the emoticons are replaced by four normalized categories.

[1] http://sentiment.christopherpotts.net/tokenizing.html

3.1.2 Specific Suffix Splitting

There are many specific suffixes in English words, such as children's and Amazon's. These words may not have any corresponding word vectors and their presence increases the vocabulary size. So, we split the specific suffixes, including 's, n't, 'll, 're, 've, 'd and 'm, to decrease vocabulary. Moreover, the words resulted from splittingy do have corresponding word vectors (Nabil, Atyia, & Aly, 2016).

3.1.3 Hashtag Segmentation

Hashtag often is composed of multiple words and includes emotional words. We try two ways to split hashtags. Table 2 shows the examples of hashtag segmentation.

♦ Maximum Matching Segmentation

We use the vocabulary of GloVe as our dictionary containing approximately 570,000 words, and define a regular expression for numbers and punctuation. From the beginning of a hashtag, we split it according to the dictionary as much as possible until segmentation is finished.

♦ Unigram-based Segmentation

We train a unigram model with 0.6M tweets obtained from Twitter API. We do statistics of different words in these tweets and remove Except for the letter “a” and the letter “i”, single letters letters are removed from the unigram dictionary to avoid over-segmentation. All results of hashtag segmentation will be split according to the dictionary, so a hashtag may have multiple results. Then, we calculate sum of log probability of each word from each result as segmentation score. Finally, we take the highest score as the final segmentation result.

Table 2. Examples of hashtag segmentation. A hashtag is converted to a word sequence.

Hashtag	Maximum matching	Unigram-based
#windows10fail	windows 10 fail	windows 10 fail
#sportshalloweencostume	sports Halloween costume	sports Halloween costume
#thisisnotajoketweet	thisis notajoke tweet	this is not a joke tweet

3.2 Embedding

Since training word embedding requires a lot of time, we use the pre-trained word vector provided by GloVe. Nevertheless, many pre-trained sets of word vectors contain few or no emoji representations. Therefore, Barbieri *et al*. built skip-gram word embedding models by mapping both words and emojis in the same vector space (Barbieri *et al*., 2016). Note that we only consider emojis and do not include emoticons because emoticons already are normalized to the normalization tokens during the data pre-processing phase. We train emoji vectors by

neural networks. Figure 3 shows the model architecture. In Figure 3, ***U*** is the weight matrix from the input layer to the hidden layer and ***V*** is the weight matrix from the hidden layer to the output layer. When the embedding training is finished, the weight matrix from the input layer to the hidden layer ***U*** consists of the emoji vectors. We use pairs of an emoji and its relevant words, including the descriptive words and the contextual words, as training examples. The steps are described as follows.

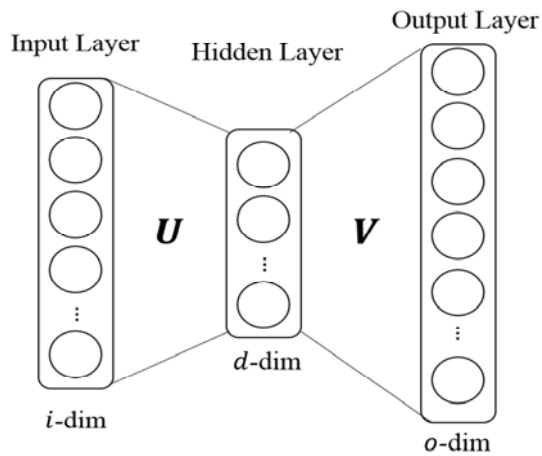

Figure 3. The model of emoji embedding. Similar to word embedding, an emoji vector is trained to predict neighboring word vectors.

3.2.1 Description Words

We crawled emojis and their descriptions from Unicode emoji standard,[2] resulting in 9,244 description words for 2,623 emojis. Every training example consists of an emoji and a sequence of words $w_1, w_2, \ldots, w_n$ describing that emoji. We tried two methods of producing the training target, described as follows. Table 3 shows examples of emojis and their descriptions.

Table 3. Examples of emojis and their description.

Emoji	Description
😀	Grinning face
😁	Beaming face with smiling eyes
🏃🏽‍♀️	Woman running: medium skin tone
💦	Sweat droplets
😾	Cat face

[2] http://www.unicode.org/emoji/charts/full-emoji-list.html

♦ Sum of the vectors of the description words

We take the sum of the individual vectors of description words as a training target, where the word vector can be found in GloVe. The description words $w_{1:n}$ correspond to pre-trained word vectors $\boldsymbol{v}_{1:n}$ where $\boldsymbol{v}_i$ is a d-dimensional word vector. The training target is $= \sum_{i=1}^{n} \boldsymbol{v}_i$. In this way, the number of training samples is 2,623.

♦ Description word splitting

We divide the description words into n training examples. For example, the description of 😀 is grinning face, so the training examples are (😀, grinning) and (😀, face). These description words have corresponding pre-trained word vectors. In this way, the number of training examples is 9,244.

The size of the input layer is equal to the number of different emojis, as emojis are represented by one-hot vectors. The size of the output layer is equal to the size of word vector, which is 100.

3.2.2 Contextual Words

We selected tweets with emojis from the aforementioned 0.6M tweets, resulting in a set of approximately 50K tweets. A tweet is $w_1, w_2, \ldots, e, \ldots, w_n$, where e is an emoji. The network input is emoji e as a one-hot vector, and the network output targets are the contextual words of e, which is $w_1, w_2, \ldots, w_n$. Training examples are $(e, w_1), (e, w_2), \ldots, (e, w_n)$. We collected about 1.7M training samples.

For the output target, we tried two kinds of representations. A sparse target representation uses a one-hot vector, while a distributed target representation uses a word vector.

3.3 Sentiment Classification

3.3.1 LSTM Classifier

Figure 4 shows the architecture of our RNN-based classifier, which contains an input layer, embedding layer, hidden layer, and soft-max layer.

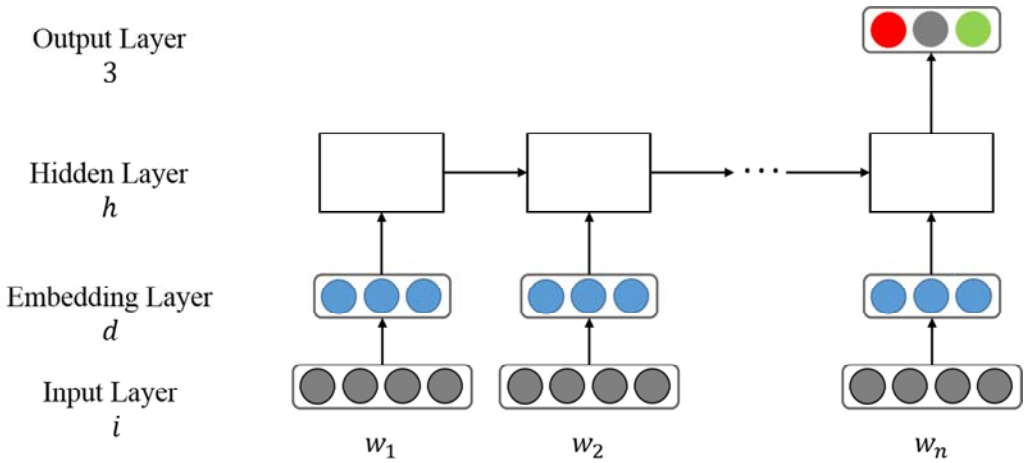

Figure 4. The architecture of the implemented classifier based on recurrent neural network with long short-term memory (LSTM) cells in the hidden layer.

In the input layer, each tweet is treated as a sequence of words and each word is input into the model at every time step. In the embedding layer, each word is converted to a word vector, where word vectors are stored in an embedding matrix provided by GloVe. In the hidden layer, we use LSTM memory cells (Hochreiter & Schmidhuber, 1997) for the long-range dependency. Different from the original recurrent unit, the LSTM cell contains gates to control states. The hidden states of the first word to the second to last word in a tweet connect to the hidden state of the next word. Only the hidden state of the last word connects to the next (output) layer. In the soft-max layer, output values are processed by soft-max function to get probabilities for classification.

3.3.2 CNN Classifier

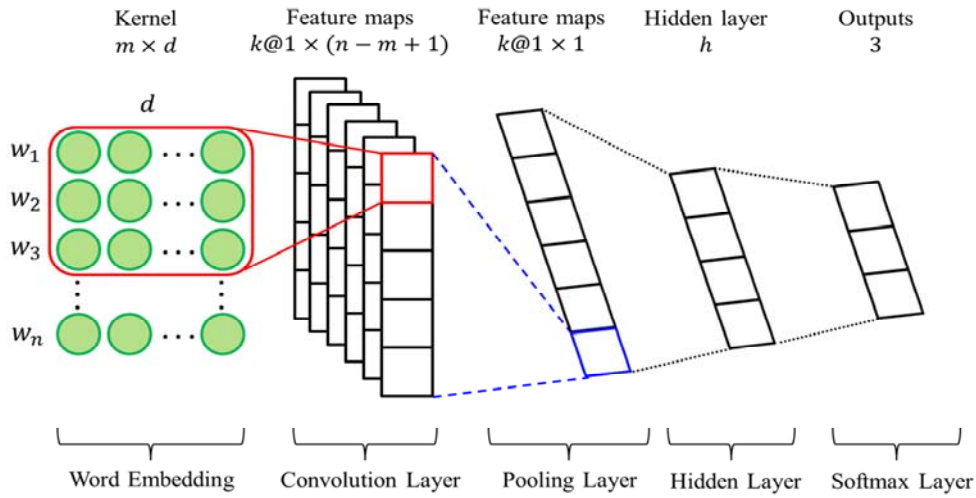

Figure 5. The architecture of the implemented classifier based on convolutional neural network (CNN).

The CNN model we use for classification is the architecture used by Kim (2014). Figure 5 shows the CNN architecture, which consists of a convolutional layer, max-pooling layer, hidden layer, and soft-max layer.

The model input is a tweet, consisting of sequence of words $w_{1:n} \in \mathcal{R}^V$, where V is the vocabulary size. In order to fix the length of the tweet, we pad input text with zeros into length n. Each tweet $w_{1:n}$ is represented by the corresponding word vector $x_{1:n}$, where x_i is the d-dimension word vector of the i-th word. Input words are embedded into dense representation through word embedding and fed into the convolutional layer. A word without an embedding vector is represented by a zero vector. After word embedding, an input tweet is mapped to an input matrix $S \in \mathcal{R}^{n \times d}$.

In the convolutional layer, kernel $K \in \mathcal{R}^{d \times m}$ slides over the input matrix with stride $s = 1$ and creates features c_i.

$$c_i = f(K * S_{i:i+m-1} + b_{conv}) \tag{1}$$

where b_{conv} is the bias at the convolutional layer, * denotes the convolution operation, and f is a nonlinear function. The feature map $y_{conv} \in \mathcal{R}^{n-m+1}$ is created by

$$y_{conv} = [c_1, c_2, \dots, c_{n-m+1}] \tag{2}$$

We use k kernels to create k feature maps, which are denoted by $Y_{conv} \in \mathcal{R}^{k \times (n-m+1)}$. Then, we apply the max-pooling operation over each feature map in order to capture important information, *i.e.*

$$y_{pool,i} = \max_j Y_{conv,i,j} \quad (3)$$

where $y_{pool,i} \in \mathcal{R}^k$ is the output after the max-pooling operation.

After the max-pooling layer, we use dropout to drop some activations for regularization randomly in order to prevent the model from over-fitting. Finally, we use a fully connected layer of size h followed by a dense layer with soft-max function for classification.

4. Results

4.1 Data

We used the SemEval 2017 data provided by the task organizers. These data are tweets in Twitter, which are labelled with three types of sentiment: positive, neutral, and negative. The training data were the tweets from SemEval 2013 to SemEval 2016, excluding SemEval 2016 testing data. The development data were the tweets from SemEval 2016 testing data. The test data were the tweets provided by the organizer of SemEval 2017. Table 4 summarizes the statistics of the data.

Table 4. Statistics of SemEval 2017: the number of tweets in datasets.

Data	Pos.	Neu.	Neg.	Total
Train	12,844	12,249	4,609	29,702
Dev	7,059	10,342	3,231	20,632
Test	2,375	5,937	3,972	12,284

4.2 Settings

We used the pre-trained word vectors provided by GloVe, which are trained with Twitter data. The dimension of word vectors can be 50, 100, or 200. We evaluated these dimensions with the SemEval 2016 dataset. The 100 and 200-dimension word vectors achieved better results, so we used 100-dimension word vectors for the SemEval 2017 tasks. We noticed that the performance is not very sensitive to the hyper-parameter of word vector size and the number of hidden layer units. For the CNN model, the number of filters k was 50. The kernel size was 3×100 with stride $s = 1$ over the input matrix. Max-pooling was applied over each feature map. After pooling, we dropped activations randomly with the probability of $p = 0.2$ and fed to the hidden layer with size $h = 20$. The hyperbolic tangent function was used as the activation function after convolution and pooling. For the LSTM model, input size i was equal to the size of word list and the size of hidden h was 50. We dropped input units for input gates and recurrent connections with the same probability of $p = 0.2$.

Next, we explain the settings of emoji embedding training. The size of input layer was equal to the number of emojis. For training with descriptive words, the size of the input layer

was 2,623, the size of the hidden layer was 100, and the size of the output layer was 100. Both hidden-layer outputs and output-layer outputs went through hyperbolic tangent function, and the loss is the mean square error.

For training with contextual words, the size of the input layer was 1,023 and the size of the hidden layer was 100. The size of the output layer was 42,670 if the target was represented by one-hot vector and was 100 if the target was represented by word vector. Output values went through the soft-max function. Both hidden-layer outputs and output-layer outputs went through the hyperbolic tangent function, and the loss is the mean square error.

All the models we used in our experiments were implemented using Keras[3] with Tensorflow[4] backend.

4.3 Baseline

We participated in SemEval 2017 task 4, which is sentiment classification in Twitter (Yang *et al.*, 2017). There are three evaluation measures in the task, which are average recall of three classes, average F-measure of positive and negative classes, and accuracy. The organizer chose average recall as the primary measure because it is more robust to class imbalance (Rosenthal *et al.*, 2017), so we focus on this performance measure. The results of each setting are obtained from an average of five runs of experiments. In each run, we trained our models with the same usage of data sets, instead of the cross-validation or leave-one-out scheme.

Table 5 shows our results of SemEval 2017. We interpolated the LSTM and CNN models to get the interpolated model for the final submission, which achieved 0.618 for average recall. Also, we list the evaluation of LSTM and CNN model. We will take this performance as our baseline.

Table 5. Results of baseline. Interpolation-baseline is the result of participating in SemEval 2017. It is an interpolation of an LSTM model and a CNN model. The result of LSTM-baseline and the result of CNN-baseline are obtained from an average of five runs of experiments.

Model	Avg. Recall	Avg. F1	Accuracy
LSTM-baseline	0.610	0.575	0.615
CNN-baseline	0.584	0.548	0.583
Interpolation-baseline	0.618	0.587	0.616

[3] https://keras.io

[4] https://www.tensorflow.org

4.4 Comparison of Data Pre-processing

In this part, we show the classification results using different data pre-processing. We summarize different statistics of pre-processed data, including vocabulary size, the number of tokens in data, and coverage on word vector. Here, coverage on word vector is the proportion of tokens with found word vectors. The word list we used was extracted from the words in the training data, and it is equal to vocabulary size. If the word vector of a word in the word list could not be found in GloVe, we used the zero vector for the word. Words in the testing data but not in the word list were removed. Each LSTM or CNN model was trained with 50 epochs. Table 6 shows the results of data pre-processing.

Table 6. Statistics of pre-processed data and results. In each pre-processing stage, we base on basic pre-processing. "emoticon" means emoticon normalization. "suffix" means specific suffix splitting. "hashtag*" means hashtag segmentation, where hashtag1 is maximum matching segmentation and hashstag2 is unigram-based segmentation.

Pre-processing	# Vocab.	# Tokens		Coverage		Avg. Recall	
		Train	Test	Train	Test	LSTM	CNN
Basic	38,353	691,261	218,821	0.914	0.886	0.610	0.584
Basic + emoticon	38,316	686,565	217,321	0.916	0.886	0.615	0.584
Basic + suffix	37,506	700,736	222,435	0.928	0.904	0.614	0.584
Basic + hashtag1	36,429	695,654	224,562	0.924	0.932	0.622	0.590
Basic + hashtag2	34,327	700,954	232,322	0.924	0.933	0.616	0.593
Basic + emoticon + suffix + hashtag1	35,542	700,410	226,621	0.940	0.948	0.625	0.594

It can be seen that the vocabulary size decreased and the coverage on word vector increased after data pre-processing. For the LSTM model, average recall of all pre-processed data was higher than the basic pre-processed data, especially data with hashtag segmentation. For the CNN model, average recall also was improved by data with hashtag segmentation. Although hashtag with unigram-based segmentation can attain better results, its average recall was lower than hashtag with maximum matching segmentation in the LSTM model. We think that the segmentation dictionary we used in maximum matching was the vocabulary of GloVe, so most words after hashtag segmentation had corresponding vectors. We combined all data pre-processing steps finally and achieved 0.625 for the LSTM model and 0.594 for the CNN model.

4.5 Evaluation of Emoji Vector

In this part, we explore the effect of adding the emoji vector. In order to have more corresponding vectors in testing data, the word list we used in this experiment was the vocabulary of GloVe. Words in the training and testing data but not in the word list were removed. If emoji vectors were added, we added the emoji to the word list.

In order to prevent the model from over-training, we used an early stopping mechanism. We added the development dataset as validation data during model training. If there was no improvement in accuracy of validation data, the model stopped training. With early stopping, each model was trained about 5-10 epochs.

The results of emoji embedding are shown in Table 7. 'No emoji' means that the emoji vector was not added. 'desc_sum' means that the sum of the description word vectors of emoji is the emoji vector. For training the emoji vector with description words, the sum of the individual word vectors of description words as a training target and dividing description words into multiple training targets are denoted as 'desc_sum_nn' and 'desc_sp_nn,' respectively. For training the emoji vector with contextual words, the training target using one-hot vector and word vector are denoted by 'skip gram_1H' and 'skip gram_vec,' respectively.

Table 7. Results of emoji embedding. Note that the refined system with 0.652 would have ranked 7th in SemEval 2017.

Pre-processing	Basic						All					
Model	LSTM			CNN			LSTM			CNN		
Early stopping	$AvgR$	F_1^{PN}	Acc.	$AvgR$	F_1^{PN}	Acc.	$AvgR$	F_1^{PN}	Acc.	$AvgR$	F_1^{PN}	Acc.
No emoji	0.634	0.601	0.627	0.624	0.594	0.618	0.651	0.630	0.639	0.629	0.599	0.612
desc_sum	0.635	0.606	0.632	0.628	0.601	0.620	0.651	0.626	0.638	0.630	0.605	0.622
desc_sum_nn	0.639	0.611	0.633	0.626	0.592	0.608	0.645	0.615	0.628	0.631	0.601	0.617
desc_sp_nn	0.638	0.607	0.623	0.626	0.596	0.615	0.652	0.628	0.632	0.638	0.618	0.623
skip gram_1H	0.639	0.613	0.633	0.622	0.593	0.617	0.642	0.614	0.633	0.640	0.618	0.620
skip gram_vec	0.639	0.616	0.635	0.620	0.586	0.610	0.646	0.619	0.626	0.633	0.604	0.614

By adding the emoji vector in systems with basic pre-processing, the average recall of the two models was mostly better than with no emoji. With all of the pre-processing, there was no significant improvement in the LSTM model. In CNN models with all of the pre-processing, the performance of adding emoji vectors was still better than without the

emoji vector.

We know that only tweets in test data have emoji, and there were 731 tweets with emoji, which made about 5.9% of the test data. Furthermore, models did not learn emoji characteristics directly during training because there were no tweets with emoji in the training data. These are possible reasons there was no significant improvement in some cases.

In order to more clearly observe the effect of adding emoji vectors for model classification, we only evaluated the test data with emojis in previous LSTM and CNN models. Table 8 shows the statistics of tweets with emoji. Table 9 shows the evaluation of tweets with emoji.

Table 8. Statistics of tweets with emoji. In our data set, only tweets in test data had emoji.

Tweet with emoji	Pos.	Neu.	Neg.	Total
Test	310	248	173	731

Table 9. Evaluation of tweets with emoji.

Pre-processing	Basic						All					
Model	LSTM			CNN			LSTM			CNN		
Early stopping	Avg$R$	F_1^{PN}	Acc.	Avg$R$	F_1^{PN}	Acc.	Avg$R$	F_1^{PN}	Acc.	Avg$R$	F_1^{PN}	Acc.
No emoji	0.621	0.665	0.638	0.621	0.661	0.630	0.644	0.696	0.656	0.624	0.679	0.637
desc_sum	0.636	0.682	0.639	0.611	0.651	0.605	0.648	0.701	0.651	0.615	0.661	0.617
desc_sum_nn	0.629	0.665	0.637	0.601	0.640	0.616	0.633	0.679	0.650	0.617	0.669	0.635
desc_sp_nn	0.599	0.639	0.614	0.606	0.647	0.617	0.630	0.682	0.645	0.624	0.683	0.635
skip gram_1H	0.632	0.666	0.636	0.604	0.641	0.612	0.638	0.685	0.648	0.634	0.692	0.643
skip gram_vec	0.611	0.637	0.624	0.591	0.622	0.608	0.627	0.670	0.649	0.619	0.665	0.636

The results show that the effect of adding emoji is not obvious. From our observation on the performance of the three classes, the addition of emoji vectors decreases the prediction of neutral class dramatically, but increases the prediction of positive and negative classes. Besides, we found that emoji vectors are more similar to each other than to the word vectors in the embedding space. Thus, they can contribute supplementary information for sentiment analysis.

5. Conclusion

We implemented our sentiment analysis system for sentiment analysis of Twitter data organized in SemEval 2017. This system consists of data pre-processing, word and emoji embedding, and classifier. From our observation, the data complexity decreases after data pre-processing with improved performance on classification, especially with hashtag segmentation. We found that adding emoji vectors can improve the performance on classification, especially for CNN models, and model training with an early stopping mechanism can prevent the model from over-training.

In data pre-processing, we process data with basic pre-processing and all pre-processing, including emoticon normalization, specific suffix splitting, and hashtag segmentation. In word embedding, we train emoji embedding with the descriptive words or the contextual words of emojis. For our models, we set the vocabulary of GloVe as the word list of models instead of the vocabulary in training data and validation data. In addition, we used an early stopping mechanism to train our models. Our system achieved 0.652 for LSTM model and 0.640 for CNN model, which would have ranked 7th in SemEval 2017.

Regarding future works, we hope to get closer in performance to the leaders on the task leader-board. As mentioned in Section 4.5, the models did not learn emoji characteristics directly during training. Thus, we want to collect tweets with emojis for training and do further evaluation on our models. We also will try the fine-tuned word vector and make it suitable for sentiment classification.

Reference

Barbieri, F., Ronzano, F., & Saggion, H. (2016). What does this Emoji Mean? A Vector Space Skip-Gram Model for Twitter Emojis. In *Proceedings of the LREC* 2016.

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., ... Strzalkowski, T. (1991). A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of the Workshop on Speech and Natural language (HLT '91)*, 306-311. doi: 10.3115/112405.112467

Cliché, M. (2017). BB_twtr at SemEval-2017 Task 4: Twitter Sentiment Analysis with CNNs and LSTMs. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval 2017)*. doi: 10.18653/v1/S17-2094

Deriu, J., Gonzenbach, M., Uzdilli, F., Lucchi, A., De Luca, V., & Jaggi, M. (2016). SwissCheese at SemEval-2016 Task 4: Sentiment Classification Using an Ensemble of Convolutional Neural Networks with Distant Supervision. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*, 1124-1128. doi: 10.18653/v1/S16-1173

Go, A., Bhayani, R., & Huang, L. (2009). *Twitter Sentiment Classification using Distant Supervision*. CS224N Project Report, Stanford, 1(12).

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., & Schmidhuber, J. (2009). A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE transactions on pattern analysis and machine intelligence*, *31*(5), 855-868. doi: 10.1109/TPAMI.2008.137

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735-1780. doi: 10.1162/neco.1997.9.8.1735

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1746-1751.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems (NIPS'12)*, 1097-1105.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, *86*(11), 2278-2324. doi: 10.1109/5.726791

Nabil, M., Atyia, A., & Aly, M. (2016). CUFE at SemEval-2016 Task 4: A Gated Recurrent Model for Sentiment Classification. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*, 52-57. doi: 10.18653/v1/S16-1005

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: Sentiment Classification using Machine Learning Techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing (EMNLP '02)*, 10, 79-86. doi: 10.3115/1118693.1118704

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532-1543. doi: 10.3115/v1/D14-1162

Rosenthal, S., Farra, N., & Nakov, P. (2017). SemEval-2017 Task 4: Sentiment Analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval 2017)*, 502-518.

Severyn, A., & Moschitti, A. (2015). UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, 464-469. doi: 10.18653/v1/S15-2079

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *Advances in neural information processing systems (NIPS'14)*, 3104-3112.

Yang, T. H., Tseng, T. H., & Chen, C. P. (2017). deepSA at SemEval-2017 Task 4: Interpolated Deep Neural Networks for Sentiment Analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval 2017)*, 616-620. doi: 10.18653/v1/S17-2101

Zhao, J., Dong, L., Wu, J., & Xu, K. (2012). Moodlens: an Emoticon-based Sentiment Analysis System for Chinese Tweets. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '12)*, 1528-1531. doi: 10.1145/2339530.2339772