System Implementation for SemEval-2017 Task 4 Subtask A Based on Interpolated Deep Neural Networks

[1]Tzu-Hsuan Yang, [2]Tzu-Hsuan Tseng, and [3]Chia-Ping Chen
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
[1]m043040003@student.nsysu.edu.tw
[2]m043040013@student.nsysu.edu.tw
[3]cpchen@cse.nsysu.edu.tw

Abstract

In this paper, we describe our system implementation for sentiment analysis in Twitter. This system combines two models based on deep neural networks, namely a convolutional neural network (CNN) and a long short-term memory (LSTM) recurrent neural network, through interpolation. Distributed representation of words as vectors are input to the system, and the output is a sentiment class. The neural network models are trained exclusively with the data sets provided by the organizers of SemEval-2017 Task 4 Subtask A. Overall, this system has achieved 0.618 for the average recall rate, 0.587 for the average F1 score, and 0.618 for accuracy.

1 Introduction

Analysis of digital content created and spread in social networks are becoming instrumental in public affairs. Twitter is one of the popular social networks, so there are more and more researches on Twitter recently, including sentiment analysis, which predicts the polarity of a message.

A message submitted to Twitter is called a tweet. Millions of tweets are created every hour, expressing users' views or emotions towards all sorts of topics. Different from a document or an article, a tweet is limited in length to 140 characters. In addition, tweets are often colloquial and may contain emotional symbols called emoticons.

For sentiment analysis, deep learning-based approaches have performed well in recent years. For example, convolution neural networks (CNN) with word embeddings have been implemented for text classification (Kim, 2014), and have achieved state-of-the-art results in SemEval 2015 (Severyn and Moschitti, 2015).

In this paper, we describe our system for SemEval-2017 Task 4 Subtask A for message polarity classification (Rosenthal et al., 2017). It classifies the sentiment of a tweet as positive, neutral, or negative. Our system combines a CNN and a recurrent neural network (RNN) based on long short-term memory (LSTM) cells. We use word embeddings in both models and interpolate them. Our submission achieved 0.618 for average recall, which ranked 19th out of 39 participating teams for subtask A.

This paper is organized as follows. In Section 2, we review previous studies on sentiment analysis in Twitter. In Section 3, we describe data, preprocessing steps, model architectures, and tools used in developing our system. In Section 4, we present the evaluation results along with our comments. In Section 5, we draw conclusion and discuss future works.

2 Related Works

In this section, we briefly review the research works of sentiment analysis in Twitter based on deep neural networks. A one-layer convolution neural network with embeddings can achieve high performance on sentiment analysis (Kim, 2014). In SemEval 2016, quite a few submissions were based on neural networks. A CNN model with word embedding is implemented for all subtasks (Ruder et al., 2016). The model performs well on three-point scale sentiment classification, while performing poorly on five-point scale sentiment classification. A GRU-based model with two kinds of embedding used for general and task-specific purpose can be more efficient than CNN models (Nabil et al., 2016).

Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017), pages 616–620,
Vancouver, Canada, August 3 - 4, 2017.

	Vocab.	Pos.	Neu.	Neg.	Total
train	29039	2607	1712	713	5032
test		5939	8672	2596	17207

Table 1: Statistics of SemEval-2016.

	Vocab.	Pos.	Neu.	Neg.	Total
train	38532	12844	12249	4609	29702
dev		7059	10341	3231	20632

Table 2: Statistics of SemEval-2017.

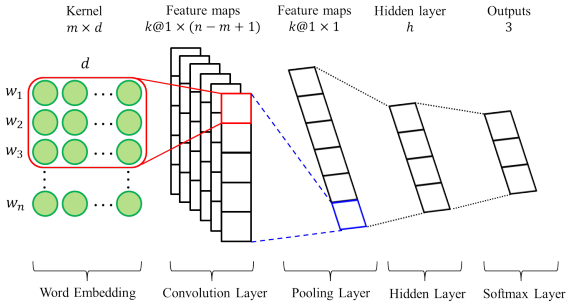

Figure 1: CNN architecture.

3 Experiment

3.1 Data

We use two datasets called SemEval-2016 and SemEval-2017. Tables 1 and 2 summarize the statistics of these datasets.

For the set of SemEval-2016, we obtain 5032 tweets for train data and 17207 tweets for test data from twitter API, respectively. Although some of the original tweets were not available in the beginning, we still use this SemEval-2016 data set for evaluating different models and tuning hyper-parameters.

The SemEval-2017 is provided by task organizers. It contains SemEval data used in the years from 2013 to 2016. We use 2013-train, 2013-dev, 2013-test, 2014-sarcasm, 2014-test, 2015-train, 2015-test, 2016-train, 2016-dev, and 2016-devtest as train data. The 2017-dev data is used for test data, which is almost the same as the 2016-test. The models trained with SemEval-2017 data is used for final submission.

A tweet is pre-processed before it is used in the neural networks. First, we use a tokenizer to split a tweet into words, emoticons and punctuation marks. Then, we replace URLs and USERs with normalization patterns <URL> and <USER>, respectively. All uppercase letters are converted to lowercase letters. Word list contains different words in the training data, and vocabulary size is the size of word list. During test, words not in the word list are removed. After pre-processing, words are converted to vectors by GloVe (Pennington et al., 2014). Then the sequence of embedding word vectors are input to neural networks.

3.2 System

3.2.1 CNN

The CNN model we use is the architecture used by Kim (Kim, 2014), which consists of a non-linear convolution layer, max-pooling layer, one hidden layer, and softmax layer. Figure 1 depicts our CNN model.

The input of this model is a pre-processed tweet, which is treated as a sequence of words. We pad input texts with zeros to the length n.

A pre-processed tweet $w_{1:n}$ is represented by the corresponding word embedding $x_{1:n}$, where x_i is the d-dimensional word vector of i-th word. The word embedding is a parameterized function mapping words to vectors as a lookup table parameterized by a matrix. Through word-embedding, input words are embedded into dense representation, and then feed to the convolution layer. Words out-of-embeddings will be represented by zero vector. And each input texts will be mapped to a $n \times d$ input matrix.

At the convolution layer, filters of size $m \times d$ slide over the input matrix and creates $(n-m+1)$ features each filter. We use k filters to create k feature maps. Thus, the size of the convolutional layer is $k \times 1 \times (n-m+1)$.

We apply the max pooling operation over each feature map (Kim, 2014). After max pooling, we use dropout by randomly drop out some activation values while training for regularization in order to prevent the model from overfitting (Srivastava et al., 2014). Then we add a hidden layer to get the appropriate representation and a dense layer with softmax function to get probabilities for classification.

3.2.2 RNN

Figure 2 shows our architecture of RNN-based model, which contains input layer, embedding layer, hidden layer and softmax layer.

At the input layer, each tweet is treated as a sequence of words $w_1, w_2, ..., w_n$, where n is the maximum tweet length. In order to fix the length

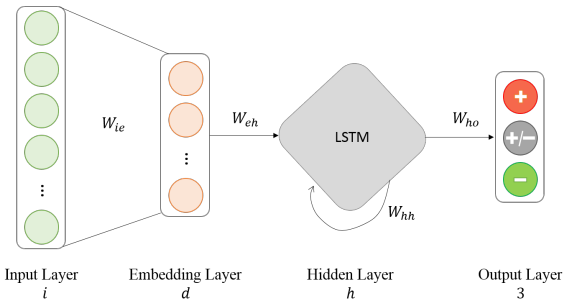

Figure 2: LSTM-based RNN architecture.

of tweet, we pad zero at beginning of tweets whose length is less than n. The size of input layer is equal to the size of word list, and each word is represented by a one-hot vector.

At the embedding layer, each word is converted to a word vector. We use pre-trained word vectors, GloVe, where word vectors are stored in a matrix. Specifically, a word in the word list is represented by the corresponding row vector (or a leading sub-vector), while a word not in the word list is represented by a zero vector.

At the hidden layer, we choose LSTM memory cell (Hochreiter and Schmidhuber, 1997) for its long-range dependency. It is argued that LSTM can get better results than simple RNN. The model contains one hidden layer, which size is h. The hidden states of first word to $(n-1)$-th word in a tweet connect to the hidden state of the next word. Only the hidden state of n-th word connect to the next (output) layer. Also, we add dropout to the hidden layer for regularization.

At the softmax layer, output values through a softmax function model the probabilities of three classes. During test phase, the sentiment class with the greatest probability is the output sentiment.

3.2.3 Interpolation

On SemEval-2016 data, performances of SA systems with respect to different sentiment classes have shown significant difference. Thus, we interpolate them to achieve better generalisation. After models are trained respectively, we interpolate them with weight λ

$$p_{\text{interp}} = \lambda \times p_{\text{lstm}} + (1-\lambda) \times p_{\text{cnn}} \quad (1)$$

where p_{lstm} and p_{cnn} are the probability of the LSTM and CNN model, respectively, and p_{interp} is the interpolated probability.

3.2.4 Settings

The maximum length for the tweets in SemEval-2017 data set is $n = 99$. The dimension of word vector is set to $d = 100$ at first, and then varied to a few values.

For CNN model, we choose $k = 50$ filters with size 3×100 with stride $s = 1$ over the input matrix. Max pooling is applied over each feature map. Then, we drop activations randomly with the probability $p = 0.2$ and feed to the hidden layer with size $h = 20$.

For RNN-based model, input size i is the size of word list and hidden size h is 50. We drop input units for input gates and recurrent connections with same probability $p = 0.2$.

We have tried rectified linear units (ReLU) and hyperbolic tangent (tanh) function for the activation function, and it seems that tanh performs better than ReLU in our experiments. We use cross entropy for the objective function and Adam algorithm for optimization. Finally, the CNN and LSTM models are interpolated with weight $\lambda = 0.6$ through a grid search.

3.3 Tool

The tokenizer for text pre-processing is the Happytokenizer[1]. All models we use in our experiments are implemented using Keras[2] with Tensorflow[3] backend.

4 Result

4.1 Comparison of Representations

First, we compare one-hot representation (sparse) and word vector representation (distributed). We train simple RNN and LSTM-based model and evaluate them on SemEval-2016 data. Each model contains one hidden layer with 50 hidden units. For models using word embeddings, the dimension of a word vector is $d = 100$.

The results are shown in Table 3. We can see that word vectors work better than one-hot vectors, except for the F1 score of RNN. We also observe that RNN model with embedding is prone to predict negative class as positive, and LSTM model predicts more accurately over all classes.

[1] http://sentiment.christopherpotts.net/tokenizing.html
[2] https://keras.io/
[3] https://www.tensorflow.org/

	RNN		LSTM	
	sparse	dist.	sparse	dist.
R_{pos}	0.634	0.867	0.726	0.807
R_{neu}	0.339	0.401	0.377	0.444
R_{neg}	0.227	0.014	0.271	0.344
Avg R	0.400	0.427	0.458	0.532
Avg F1	0.365	0.310	0.427	0.515
Acc.	0.424	0.503	0.482	0.554

Table 3: One-hot (sparse) vs. word vector (dist.).

system ID	Avg R	Avg F1	Acc.
RNN-50-20	0.417	0.319	0.485
RNN-100-50	0.427	0.310	0.503
RNN-200-50	0.436	0.410	0.453
LSTM-50-20	0.504	0.496	0.516
LSTM-100-50	0.532	0.515	0.554
LSTM-200-50	0.537	0.522	0.549
LSTM-200-100	0.512	0.500	0.523

Table 4: RNN vs. LSTM. The numbers in a system ID indicate the dimension of word vector and the number of neurons in the hidden layer.

4.2 Comparison of RNN and LSTM

Table 4 list the results of the comparison of RNN and LSTM using SemEval-2016 data. The results of LSTM model are better than RNN model, showing that long-range dependency within text message is useful in sentiment analysis.

4.3 Comparison of Data Amounts

Table 5 shows the results of LSTM and CNN on SemEval-2016 and SemEval-2017 data. As expected, various measures of performance are improved with an increase in the amount of train data.

4.4 Model Interpolation

From Table 5, we can see that CNN performs better than LSTM on negative class, and LSTM performs better than CNN on positive and neutral classes. Thus, by combining their strengths, better generalization can often be achieved than an individual system.

We tune hyper-parameter λ of interpolation via a grid search. We choose word vector size $d = 100$ for both models, one hidden layer with 50 hidden neurons for LSTM model, and number of filters $k = 50$ and fully connected size $h = 20$ for CNN model.

Model	LSTM-100-50		CNN-100-50-20	
	2016	2017	2016	2017
R_{pos}	0.807	0.729	0.824	0.697
R_{neu}	0.444	0.633	0.335	0.502
R_{neg}	0.344	0.451	0.344	0.606
Avg R	0.532	0.604	0.501	0.602
Avg F1	0.515	0.581	0.487	0.564
Acc.	0.554	0.637	0.505	0.585

Table 5: Comparison of LSTM and CNN using different amounts of data. Here the numbers in a CNN system ID indicate the dimension of word vector, the number of filters, and the size of the hidden layer.

		Avg R	Avg F1	Acc.
2017 dev	baseline	0.333	0.255	0.342
	LSTM	0.604	0.581	0.637
	CNN	0.602	0.564	0.585
	interpolation	0.631	0.604	0.640
2017 test	baseline	0.333	0.162	0.193
	interpolation	0.618	0.587	0.616

Table 6: Results on SemEval-2017 data with interpolation weight $\lambda = 0.6$.

Eventually, the interpolated system gets 0.618 for average recall rate on subtask A on SemEval 2017 test data, as shown in Table 6.

5 Conclusion

We implemented CNN and LSTM models with word embedding for sentiment analysis in Twitter data organized in SemEval 2017. Our experiments reveled an interesting point that LSTM model performs well on positive and neutral classes, while CNN model performs average on all classes. The final submission is based on model interpolation, with the weight decided by development set. It achieved 0.618 for 3-class average recall rate, 0.587 for 2-class average F1-score, and 0.618 for accuracy.

For near-future works, we hope to get closer in performance to the leaders on the board, respectively 0.681, 0.685, and 0.657. We will start by looking at methods that deal with data imbalance, as well as adversarial training approaches.

Acknowledgments

We thank the Ministry of Science and Technology of Taiwan, ROC for funding this research.

References

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

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

Mahmoud Nabil, Amir Atyia, and Mohamed Aly. 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)*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Sara Rosenthal, Noura Farra, and Preslav Nakov. 2017. SemEval-2017 task 4: Sentiment analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation*. Association for Computational Linguistics, Vancouver, Canada, SemEval '17.

Sebastian Ruder, Parsa Ghaffari, and John G Breslin. 2016. Insight-1 at semeval-2016 task 4: Convolutional neural networks for sentiment classification and quantification. *arXiv preprint arXiv:1609.02746* .

Aliaksei Severyn and Alessandro Moschitti. 2015. Unitn: Training deep convolutional neural network for twitter sentiment classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), Association for Computational Linguistics, Denver, Colorado*. pages 464–469.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.