Report: Exploration on Sentiment Analysis in Twitter

Weiheng Li

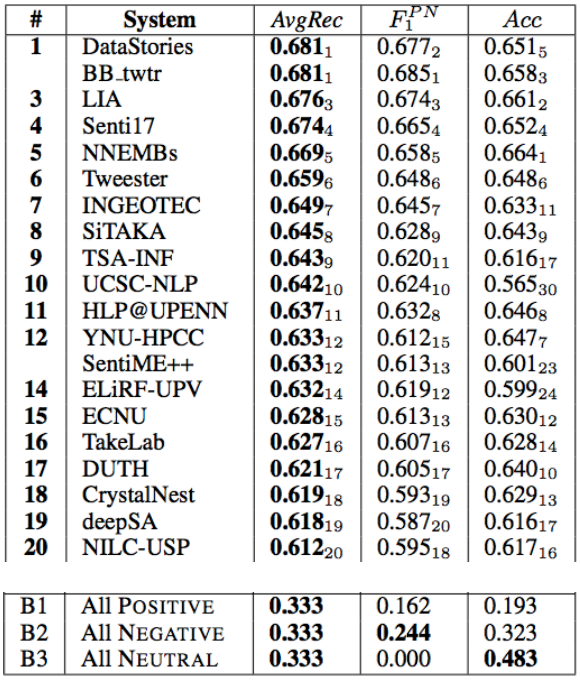
1) Introduction

Social media has been part of our life for many years. APPs like Facebook, Instagram, Twitter generate huge amount of information everyday. The popular of social media app provide us some new content to be applied in deep learning. SemEval (Semantic Evaluation) is an ongoing series of evaluations of computational semantic analysis systems, organized under the umbrella of SIGLEX, the Special Interest Group on the Lexicon of the Association for Computational Linguistics. This paper is mainly to explore some new neural network structure used in the task 4 of the competition held by SemEval, Sentiment Analysis in Twitter.[1]

This is a task also concerned with natural language processing. It is different from normal NLP task because content in social media is combined with creativity of app users and social media culture. For example the word *good* may be written as *goooood* and you don't know how many letter 'o' there would be. And many unkown emojis and typos also effect the prediction accurace.

2) Background (or Literature Review)

Because of the fact that I don't have a GPU and my laptop is not fast enough for too much calculation task and my time is limited. I decide to focus on subtask A of this task. This task is to classify the sentiment of twitter messages. All messages are classify into positive, negative, or neutral.



#	System	*AvgRec*	F_1^{PN}	*Acc*
1	DataStories	$\mathbf{0.681}_{1}$	0.677_{2}	0.651_{5}
	BB_twtr	$\mathbf{0.681}_{1}$	0.685_{1}	0.658_{3}
3	LIA	$\mathbf{0.676}_{3}$	0.674_{3}	0.661_{2}
4	Senti17	$\mathbf{0.674}_{4}$	0.665_{4}	0.652_{4}
5	NNEMBs	$\mathbf{0.669}_{5}$	0.658_{5}	0.664_{1}
6	Tweester	$\mathbf{0.659}_{6}$	0.648_{6}	0.648_{6}
7	INGEOTEC	$\mathbf{0.649}_{7}$	0.645_{7}	0.633_{11}
8	SiTAKA	$\mathbf{0.645}_{8}$	0.628_{9}	0.643_{9}
9	TSA-INF	$\mathbf{0.643}_{9}$	0.620_{11}	0.616_{17}
10	UCSC-NLP	$\mathbf{0.642}_{10}$	0.624_{10}	0.565_{30}
11	HLP@UPENN	$\mathbf{0.637}_{11}$	0.632_{8}	0.646_{8}
12	YNU-HPCC	$\mathbf{0.633}_{12}$	0.612_{15}	0.647_{7}
	SentiME++	$\mathbf{0.633}_{12}$	0.613_{13}	0.601_{23}
14	ELiRF-UPV	$\mathbf{0.632}_{14}$	0.619_{12}	0.599_{24}
15	ECNU	$\mathbf{0.628}_{15}$	0.613_{13}	0.630_{12}
16	TakeLab	$\mathbf{0.627}_{16}$	0.607_{16}	0.628_{14}
17	DUTH	$\mathbf{0.621}_{17}$	0.605_{17}	0.640_{10}
18	CrystalNest	$\mathbf{0.619}_{18}$	0.593_{19}	0.629_{13}
19	deepSA	$\mathbf{0.618}_{19}$	0.587_{20}	0.616_{17}
20	NILC-USP	$\mathbf{0.612}_{20}$	0.595_{18}	0.617_{16}
B1	All POSITIVE	**0.333**	0.162	0.193
B2	All NEGATIVE	**0.333**	**0.244**	0.323
B3	All NEUTRAL	**0.333**	0.000	**0.483**

This subtask appeared in task4 in both 2016 and 2017 SemEval competition. The accuracy of every team is not as high in homework we did, around 60%, because of the limitation of preprocess and classifier. [2]

In this task, we should first try to preprocess all twitter sentences and turn them into machine language (Integer vector). It's a big problem to deal with informal and "creative" writing style, with improper use of grammar, figurative language, misspellings and slang. Also, we need to build proper neural network to train the system.

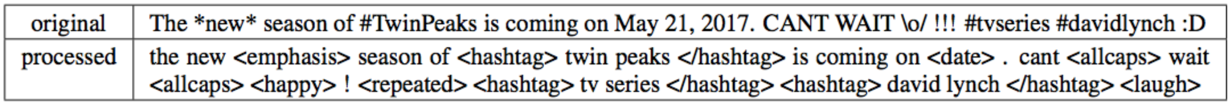
In previous runs of the Task[3][4], sentiment analysis was usually tackled using hand-crafted features and/or sentiment feeding them to classifiers such as Naive Bayes or Support Vector Machines (SVM). To solve a short sentence such as Twitter this might be a good idea. But to reach a high accuracy, we need to generete features to get connection between words both in a long distance and short distance. That results both in redundant and missing features. The system would become too large and slow to train on a normal laptop.

The evaluation of result adopted by SemEval is f1 score. We should consider about accuracy of all three class and get average of three accuracy. The f1 score of all team is not very high compared with other classification task. This image show the f1 score, total accuracy and accuracy of nag and pos classes of top 20 team

3) Methodology

Sentense preprocess: There are some twitter Tokenizers in the github and python library. I choose the python tool ekphrasis to do our sentense preprocess task. (we can use 'pip install ekphrasis' to install this package but I put all file in the root dir because I did some modification to their source code) [7]

ekphrasis can resume some words that decoded in a mistaken decoding method. (schÃ¶n→ schön) It can omit some useless word, unpack contractions, replace some thing like data, emoji, money into more recognizable expression method like the image below.

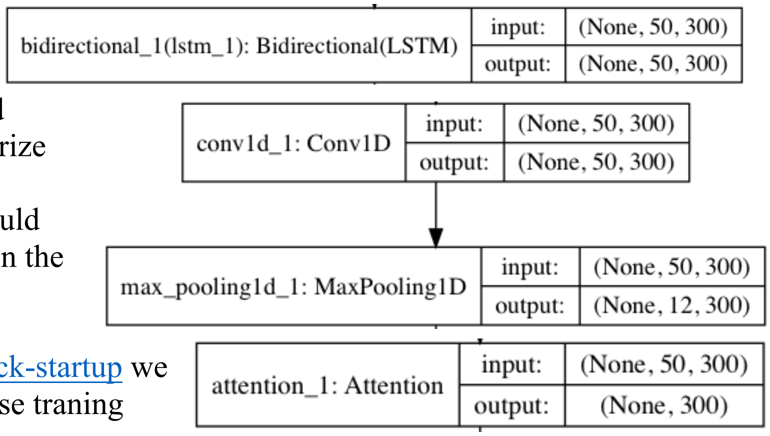


original	The *new* season of #TwinPeaks is coming on May 21, 2017. CANT WAIT \o/ !!! #tvseries #davidlynch :D
processed	the new <emphasis> season of <hashtag> twin peaks </hashtag> is coming on <date> . cant <allcaps> wait <allcaps> <happy> ! <repeated> <hashtag> tv series </hashtag> <hashtag> david lynch </hashtag> <laugh>

Word2vector: The team *Datastory* collected a big dataset of 330M English Twitter messages, gathered from 12/2012 to 07/2016. using GloVe[9] to generate a vocabulary of more than 660k words. Generate vector for each word. Each word can be switched to a vector with 300 dimmension. I download the file from their github and extract those data as my word embedding.[8]

We keep 50 words for every sentence. If the word number is more than 50, we crop it into a 50. If less than 50 we use 0 to fill it. Every word is represented as a 300 dimension vector. So the input x_traning data has a shape of [None, 50, 300]. And y_training is represented in one hot format, so it has a shape of [None, 3].

This project is to test the effect of different combination of different kind of layer. There are three kind of layer to be test:



bidirectional_1(lstm_1): Bidirectional(LSTM) — input: (None, 50, 300) — output: (None, 50, 300)

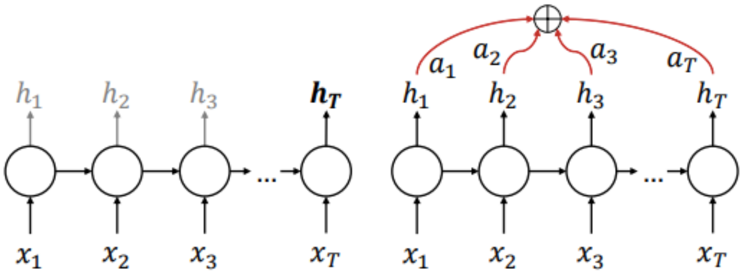
conv1d_1: Conv1D — input: (None, 50, 300) — output: (None, 50, 300)

max_pooling1d_1: MaxPooling1D — input: (None, 50, 300) — output: (None, 12, 300)

attention_1: Attention — input: (None, 50, 300) — output: (None, 300)

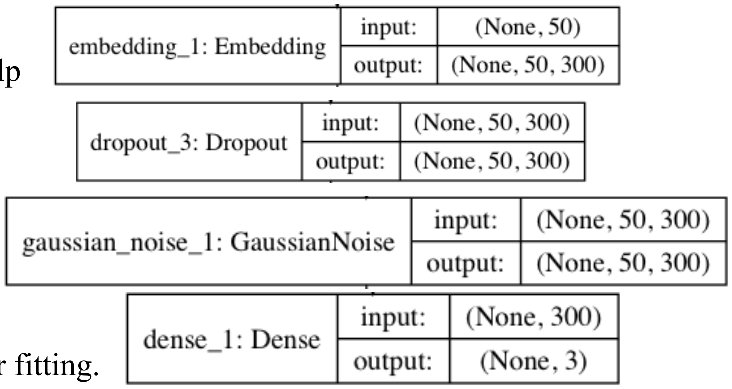
LSTM/BiLSTM: We should use LSTM to extract information hiding in order of words in both long term and short term. BiLSTM can help in memorize words order in two directions. When followed by attention layer, LSTM should output not only final state but all state in the whole process.[5]

Convolutional+MaxPooling: From https://github.com/geektown/keras-quick-startup we can see Conv+Pooling layer can increase traning speed for 4 times and get the similiar accuracy. I would try to add Conv+MaxPooling layer at the beginning to increase training speed.



(a) Regular RNN (b) Attention RNN

Attention: An RNN updates its hidden state hi as it processes a sequence and at the end, the hidden state holds a summary of all the processed information. In order to amplify the contribution of important elements in the final representation we use an attention mechanism[6] that aggregates all the hidden states using their relative importance.



There are some other layers I also adoped to help use solve problem of over-fitting or get a better result or get a correct dimension of data.
Embedding: Embedding layer is a very popular layer use in NLP to emplement word2vector.
Dropout: I use Dropout after each layer to reduce p robl em of over-fitting.
GaussianNoise: Also a method to metigate over fitting.
Dense: to get the final classification result

4) Implementation and Experimental Results

Total: 44613 {negative: 15.58%, neutral: 44.77%, positive: 39.65%}
Total: 4957 {negative: 15.57%, neutral: 44.79%, positive: 39.64%}
From my hw3 task4, I implemented LSTM sentiment analysis on the IMBD data set and get a very high accuracy (around 0.88). So I would use single LSTM layer as benchmark to evaluate other NN structure. In the experiment, the f1 point of classification may change after every epoch. Because of the limitation of time and CPU, I can only run 20 epoch for every structure. (around 1 - 10h depending on complexity of structure). I'd show the line graph of f1 point of every structure here. Some apparently outstanding strucure would be mentioned in my conclusion.

I tried many structures myself as shown below:

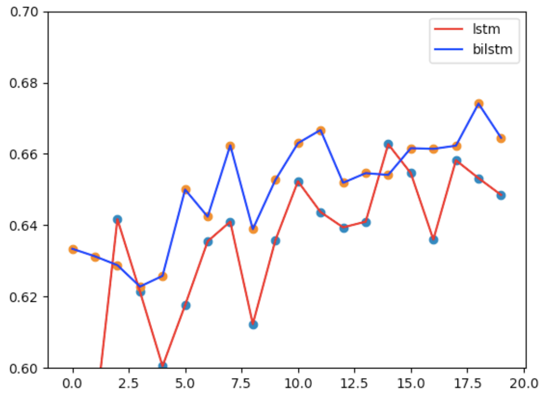


Fig.1

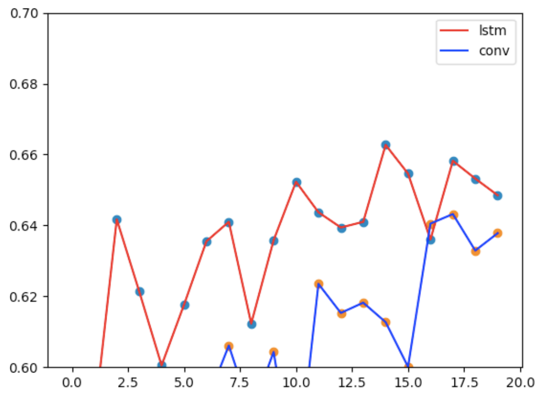
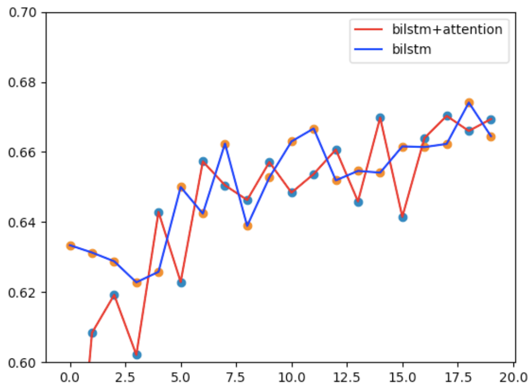


Fig.2

First image shows that BiLSTM has better performance than LSTM. Second shows LSTM is better than only using convolutional layer. So next experiment would use BiLSTM as benchmark.



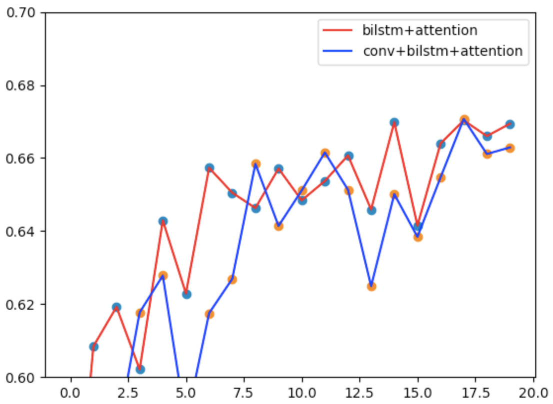
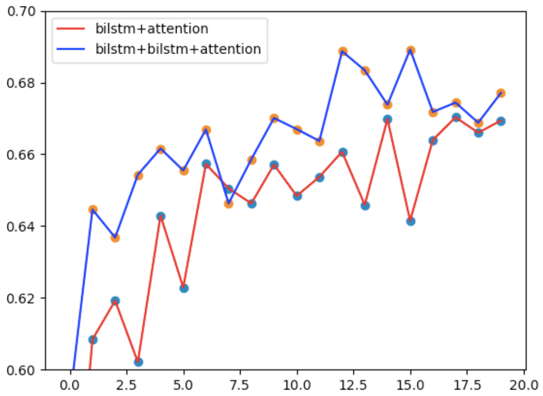


Fig.3

Fig.4

Third image shows that BiLSTM +Attention is equally well compared with bilstm. Fourth image shows that BiLSTM +Attention is a little better than Conv+ BiLSTM +attention, but the time is far more than Conv+ BiLSTM +Attention (around 4 times).



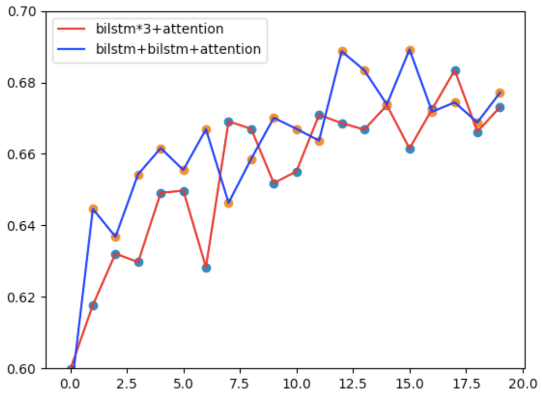


Fig.5

Fig.6

Fifth image shows that BiLSTM+BiLSTM+Attention is better than BiLSTM+Attention. Sixth image shows three BiLSTM is no better than two BiLSTM. This two figures can show us that. Two BiLSTM is best in my task.

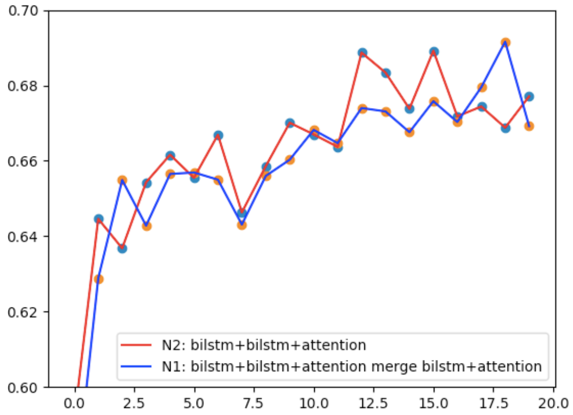


Fig.7

Last image shows that if we merge Attention layer after one BiLSTM and two BiLSTM the accuracy would be similar to BiLSTM+BiLSTM+Attention. I would mention first as N1 structure and second as N2 structure. Structure of N1 and N2 generated by Keras are attached in the end of this report.

5) Conclusion and Discussion

From the experiments, I think I figure out some better structure which may be able to apply in sentiment analysis N1 and N2. From the last image, we can see when considering the best f1 score, N1 is better than N2 but the average accuracy seems N2 is better. And, N1 is still having a trend to increase at the end of 20th epoch. We don't know what would happen if we run for 100 epoches. This need more experiments to test.

In the 2017 competition, the highest score is 0.681. With same dataset, I can achieve a highest f1 score of 0.692 with my N1 structure. And my N2 also can achieve a score of 0.689. One thing to mention, I get this score with limitation of hardware and time, I have no high ability GPU but just my laptop and I have no time to test too many epoch for each structure, only 20 epoches. I believe I can get a even high grade with better hardware.

There is still some other structure I want to try, for example, merge of [BiLSTM+Attention, BiLSTM+BiLSTM+Attention, BiLSTM+BiLSTM+BiLSTM+Attention] to get more middle process data. But this structure is so complicated that I have no time to try. In the future I want to try more structure and more epoches to find which is the best structure.

Reference:

[1] semeval2017: http://alt.qcri.org/semeval2017/index.php?id=participants

[2] 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) (pp. 502-518).

[3] https://github.com/edilsonacjr/semeval2017

[4] https://github.com/lxdv/SemEval-2017

[5] https://keras.io/layers/recurrent/#lstm

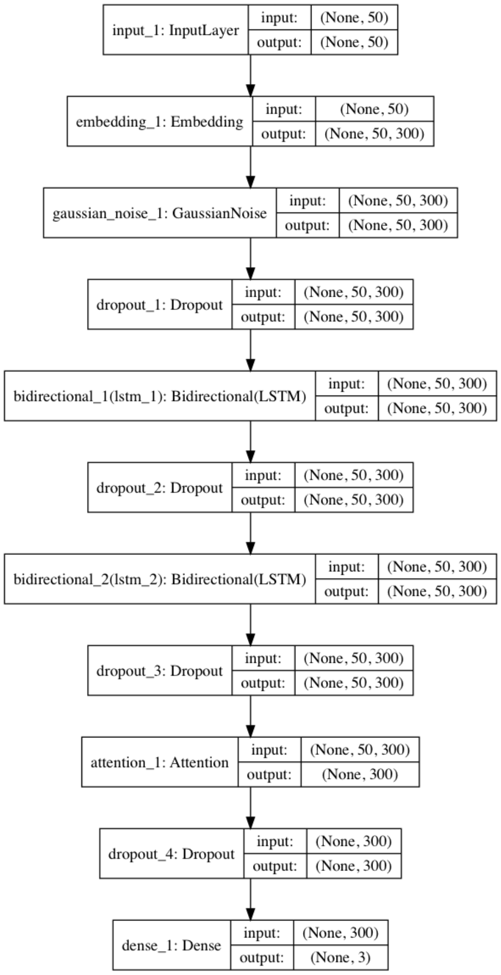
[6] Baziotis, C., Pelekis, N., & Doulkeridis, C. (2017). DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis. In Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017) (pp. 747-754).

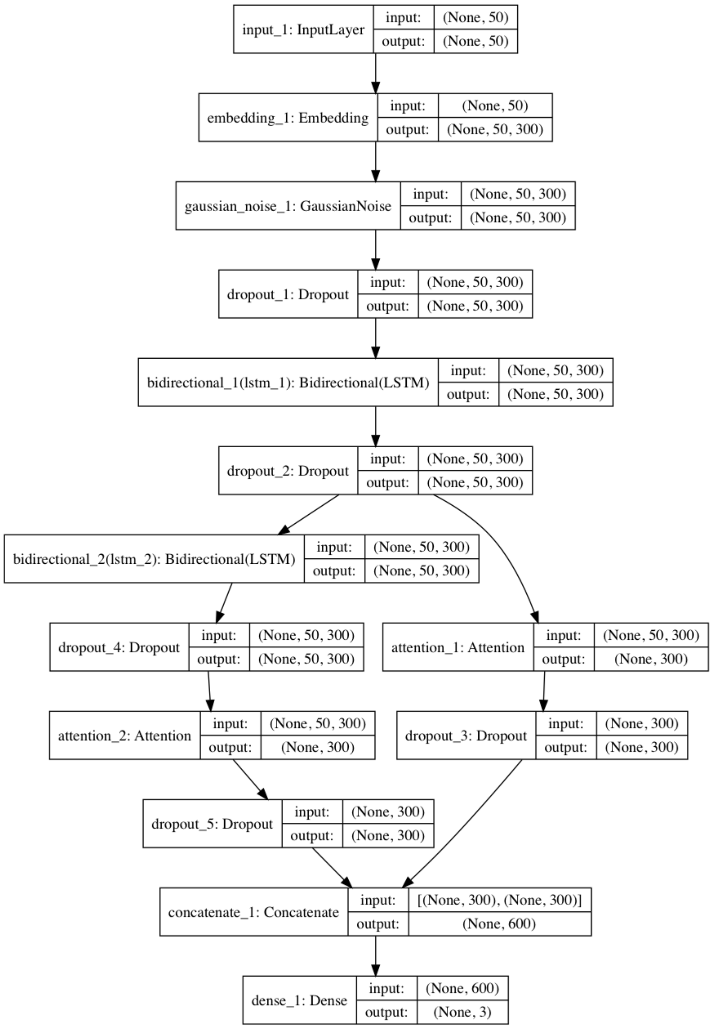
[7] https://github.com/cbaziotis/keras-utilities

[8] https://github.com/cbaziotis/datastories-semeval2017-task4

[9] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

N2: N1:


input_1: InputLayer
input: (None, 50)
output: (None, 50)
embedding_1: Embedding
input: (None, 50)
output: (None, 50, 300)
gaussian_noise_1: GaussianNoise
input: (None, 50, 300)
output: (None, 50, 300)
dropout_1: Dropout
input: (None, 50, 300)
output: (None, 50, 300)
bidirectional_1(lstm_1): Bidirectional(LSTM)
input: (None, 50, 300)
output: (None, 50, 300)
dropout_2: Dropout
input: (None, 50, 300)
output: (None, 50, 300)
bidirectional_2(lstm_2): Bidirectional(LSTM)
input: (None, 50, 300)
output: (None, 50, 300)
dropout_3: Dropout
input: (None, 50, 300)
output: (None, 50, 300)
attention_1: Attention
input: (None, 50, 300)
output: (None, 300)
dropout_4: Dropout
input: (None, 300)
output: (None, 300)
dense_1: Dense
input: (None, 300)
output: (None, 3)


input_1: InputLayer
input: (None, 50)
output: (None, 50)
embedding_1: Embedding
input: (None, 50)
output: (None, 50, 300)
gaussian_noise_1: GaussianNoise
input: (None, 50, 300)
output: (None, 50, 300)
dropout_1: Dropout
input: (None, 50, 300)
output: (None, 50, 300)
bidirectional_1(lstm_1): Bidirectional(LSTM)
input: (None, 50, 300)
output: (None, 50, 300)
dropout_2: Dropout
input: (None, 50, 300)
output: (None, 50, 300)
bidirectional_2(lstm_2): Bidirectional(LSTM)
input: (None, 50, 300)
output: (None, 50, 300)
dropout_4: Dropout
input: (None, 50, 300)
output: (None, 50, 300)
attention_1: Attention
input: (None, 50, 300)
output: (None, 300)
attention_2: Attention
input: (None, 50, 300)
output: (None, 300)
dropout_3: Dropout
input: (None, 300)
output: (None, 300)
dropout_5: Dropout
input: (None, 300)
output: (None, 300)
concatenate_1: Concatenate
input: [(None, 300), (None, 300)]
output: (None, 600)
dense_1: Dense
input: (None, 600)
output: (None, 3)