CS 584 Final Project: Tweet Sentiment Extraction

Linsen Li, Wei Yang
Stevens Institute of Technology
lli69@stevens.edu, wyang17@stevens.edu

Abstract

With all of the tweets circulating every second it is hard to tell whether the sentiment behind a specific tweet will impact a company, or a person's, brand for being viral (positive), or devastate profit because it strikes a negative tone. Capturing sentiment in a language is important in these times. With the dataset from Kaggle, we implemented Multi-channel CNNs (convolutional neural network), Multi-layer RNNs(recurrent neural network) for classification as our first step. With grid search and three-fold cross-validation (on optimizers, filter size, number of filters, embedding size, pre-trained embedding), the best hyper-parameters combination on CNNs achieved 82% accuracy, and the Multi-layer RNNs(Stacked Bidirectional LSTM) achieved 84% accuracy based on the "summarized/selected text" on the dataset. However, on the "original text", these two structures can only achieve 72% and 74% separately. Therefore, a more complex RNNs structures are introduced to make the comparison. We combined the summarization and classification tasks together in the final combined RNN model. With adding the categorical loss of the summarization part on the final classification loss, the model with both "original text" and "selected text" can achieve 86.14% accuracy on the test set. Finally, to further improve the combined model structure in the future, we extracted the summarization part of combined RNNs to compare with the Attention Sequence to Sequence model, evaluating on the BLEU score.

1 Introduction

Our project is a task of text classification and keyword extraction, more specific, a problem of sentiment analysis. Sentiment analysis becomes more and more useful in our data overload world. For instance, it is extremely useful in social media monitoring as it allows us to gain an overview of the wider public opinion behind certain topics. Tweet is one of the most popular social media platform in the world. In this task we use the data set of tweets text with label. We will build neural network classifier to do the text classification and keyword extraction. We will make improvement for our model, for instance, using different word embedding methods, tuning the hyperparameters for the deep learning model, trying more advanced model such as Transformer, BERT, etc. Last but not the least, we will compare each of the model we have talked about. Then evaluate the performance of each method.

2 Related Work

Deep learning models have achieved remarkable results in nature language processing. Much of the work with deep learning methods has involved text classification through neural language models. Some work used Convolutional neural networks (CNN) for Sentence Classification (Yoon Kim, et al, 2014, Ye Zhang, Byron Wallace., et al, 2016). Wen et al. (2016) and Adel and Schutze (2017) support CNN over GRU/LSTM for classification of long sentences. In addition, Yin et al.(2016) achieve better performance of attention-based CNN than attention-based LSTM for answer selection. However, Arkhipenko et al. (2016) compare word2vec (Mikolov et al., 2013), CNN, GRU and LSTM in sentiment analysis of Russian tweets, and find GRU performs better.

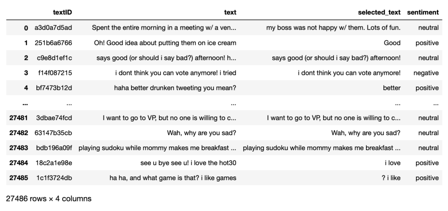
So much work have been done for text classification previously. There is no absolute answer for which deep learning structure is the best in this task. It depends on how much the task is dependent upon long semantics or feature detection. Firstly, we will use these previous results to complete the classifier. In addition to this, our project will do the feature extraction for the origin text. We will use this feature detection to make our model more robust. This is inspired by some neural machine translation model like Seq2seq. We will make improvement basing on the previous work.

3 Dataset and Features

3.1 Dataset description

The data set is offered by Kaggle. We have a training set with 27486 tweets example and a test set with 3535 instance. The training set has 4 features, including:

1. TextID: Unique ID for each piece of text
2. Text: The text of the tweet, which we are to classify
3. Selected_text[train only]: The sentiment key words from the Text that supports the tweet's sentiment.
4. Sentiment: The general sentiment of the tweet, including 3 classes: positive, negative and neutral.



	textID	text	selected_text	sentiment
0	a3d0a7d5ad	Spent the entire morning in a meeting w/ a ven...	my boss was not happy w/ them. Lots of fun.	neutral
1	251b6a6766	Oh! Good idea about putting them on ice cream	Good	positive
2	c9e8d1ef1c	says good (or should i say bad?) afternoon! h...	says good (or should i say bad?) afternoon!	neutral
3	f14f087215	i dont think you can vote anymore! i tried	i dont think you can vote anymore!	negative
4	bf7473b12d	haha better drunken tweeting you mean?	better	positive
...	...	...	...	...
27481	3dbae74fcd	I want to go to VP, but no one is willing to c...	I want to go to VP, but no one is willing to c...	neutral
27482	63147b35cb	Wah, why are you sad?	Wah, why are you sad?	neutral
27483	bdb196a09f	playing sudoku while mommy makes me breakfast ...	playing sudoku while mommy makes me breakfast ...	neutral
27484	18c2a1e98e	see u bye see u! i love the hot30	i love	positive
27485	1c1f3724db	ha ha, and what game is that? i like games	? i like	positive

27486 rows × 4 columns

Figure 1: Training set visualization

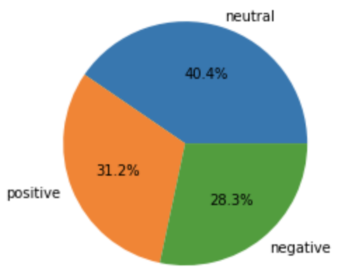
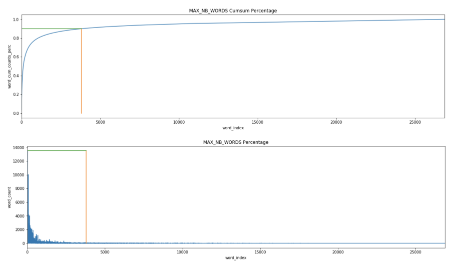


Figure 2: Percentage for each sentiment in training set - balance dataset

For the classification task, the input of the model is the Text column or Selected_text column. The label is the sentiment which has three categories. For the key words extraction task, the input of the input is the Text column and the label is the Selected_text column.

3.2 Exploratory Data Analysis (EDA)



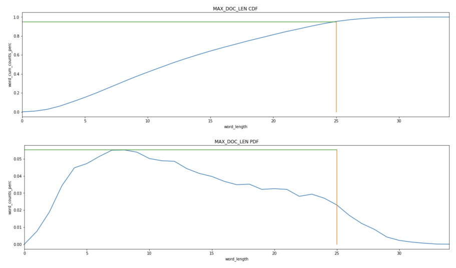


Figure 3: Max number of word percentage and cumsum percentage

Figure 4: Max document length CDF and PDF

In order to reduce noise,we deleted website with regular expression and treat exclamatory and question mark as single word.Because these two symbols are the important features for sentiment classification.For RNN, we also keep " dot " as single word.

To improve efficiency, we plot these graph in the above. We count words in each sentences and sorted by the count. We make a cut with 90% to get higher freq word in vocabuary. As you can see, if we set with 95%,the big gap between those two settings is nearly double, which is really computational expensive. With the same logic, we make a cut with 95% the length of all sentences, which is 27.

4 Methods

4.1 Recurrent Neural Network(RNNs)

Recurrent Neural Network(RNNs) is widely used in Nature language processing problem. Simple RNN is one of the most basic model. The core idea is to apply the same weight W repeatedly. The input can be sequence of data, in this case is the sequence of word embedding. The output can be a feature vector, in this case is the 'sentiment' label. However, one of the biggest problem for SimpleRNN is Vanishing gradient problem. For a RNNs model and chain rules, we have the equations:

$$\mathbf{h}^{(t)} = \sigma\left(W_h \mathbf{h}^{(t-1)} + W_x \mathbf{x}^{(t)} + \mathbf{b}_1\right) \tag{1}$$

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(W_h \mathbf{h}^{(t-1)} + W_x \mathbf{x}^{(t)} + \mathbf{b}_1\right)\right) W_h \tag{2}$$

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \tag{3}$$

If W_h is small, then $\boldsymbol{W}_h^{(i-j)}$ gets vanishing small as i and j get further apart. And this will cause $\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}}$ close to zero. This is the Vanishing gradient problem. As we all know the method for update the weight W in deep neural network is backpropagation. If the gradient is nearly zero, then the update for this parameter is nearly zero. This problem will make the model forget the early input of the sequence. Besides, RNNs may also have the problem of exploding gradient, which is the opposite of Vanishing gradient. That's why researchers build an another RNNs structure Long Short Term Memory(LSTM) to deal with these problem. The main difference between LSTM and SimpleRNN is that LSTM has a Conveyor belt, which make the past information directly flow to the future. LSTM also has forget gate and input gate, which are use to control the information flow in Conveyor belt. The update formula for conveyor belt is:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{4}$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{5}$$
$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{6}$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{7}$$

where f_t is the output of forget gate, i_t is the output of input gate, C_t is the cell state at time t. Then, LSTM has a output gate o_t to update the hidden state h_t:

$$\begin{aligned} o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\ h_t &= o_t * \tanh\left(C_t\right) \end{aligned} \tag{8}$$

note that LSTM has four times more parameters than SimpleRNN, they are W_f, W_i, W_C, W_o.
Since LSTM has better performance than SimpleRNN, we will use LSTM as our RNNs cell. Besides, we will use following techniques to improve our RNNs model:

- GloVe pre-train embedding: Using pre-train word embedding is to avoid the over-fitting in embedding layer. If we don't use this technique, the number of parameter in the embedding layer will be too huge, which will definitely cause over-fitting.
- Stacked RNN: This is a technique to make the RNNs into a deep neural network. In deep learning, a deeper neural network is always better when the size of training data is big enough.
- Bidirectional RNNs: Since RNNs is a sequence structure, when the input sequence is long, it will lose the information from the beginning. Bidirectional RNNs will fully use the text data.It won't lose information from the head of the text.
- Dropout: Dropout is a technique widely use in deep neural network. It forces the neural network to make prediction basing on part of data. We use it to avoid over-fitting problem. Note that different from the fully connected neural network, RNNs also have a Dropout in recurrent structure, which is called recurrent Dropout.

4.2 Multi-channel Convoluted Neural Network(CNNs)

Originally, Convoluted Neural Networks(CNN) are specialized networks for application in computer vision.It accept images as raw input (preserving spatial information) and build up (learn) a hierarchy of features (no hand-crafted features necessary). Besides, the CNN structure can also apply in a Nature language processing problem. Yoon Kim has post a paper: YoonKim(2014):Convolutional Neural Networks for Sentence Classification. EMNLP 2014. In our project, our CNN model is inspired by this paper. The following is a illustration of this model from that paper:

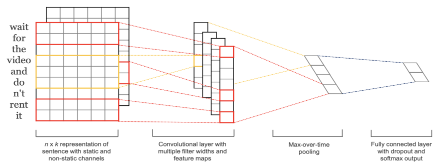

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 5: Model architecture with two channels for an example sentence

This is called multi-channel CNN. We know that the Convoluted layer acts like the feature extraction. For Computer Vision, it extracts detail from the picture, like the edge, shape of the item. For Nature Language Processing,similarly, it extracts detail from the text, like Bigram, Trigram. Recall that in the feature engineering, there is a n-gram level in TF-IDF. In this CNN part, the idea is similar. For example, if we choose a filter with the size of 3, then this filter is to extract the trigram feature of the origin text. The multi-channel CNN means we use different size of filter to extract information. This is different from the CNN in Computer Vision.

We use pre-train embedding and Dropout in our CNN model.Besides, we will also apply some techniques in CNN to make our CNN robust:

- Batch Normalization: This technique is often used in CNNs.It the standardization of hidden layers. The aim of Batch Normalization is to make the model converge fast.Batch Normalization layer is always after convoluted and before activation. The following is the specific steps for Batch Normalization:
  · Let $\mathbf{x}^{(k)} \in \mathbb{R}^d$ be the output of the k -th hidden layer.
  ·$\widehat{\mu} \in \mathbb{R}^d$: Non-trainable. Just record them in the forward pass;
  ·$\widehat{\sigma} \in \mathbb{R}^d$: use them in the backpropagation.
  ·$\gamma \in \mathbb{R}^d$: scaling parameter (trainable).
  ·$\beta \in \mathbb{R}^d$: shifting parameter (trainable).
  · Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\widehat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$
  · Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \cdot \gamma_j + \beta_j$, for $j = 1, \cdots, d$

4.3 Seq2seq model

Seq2seq model is a combination of two RNNs models. One is called encoder and another one is called decoder. In this project,the input of the encoder is the Text column. The output of the encoder would be the last hidden state, which is also the input of decoder. For the training part, the decoder acts like a text generator: the input is a word embedding in the sequence and the corresponding label is the embedding of next word.

Seq2seq model usually is used for machine translation. However, in this project we use it to do the key words summarizing. As a result, the encoder and decoder share the same vocabulary. Note that we will use Bidirectional RNNs only in the encoder part. Besides, we will use attention to improve our Seq2seq model.

- Attention: For traditional Seq2seq, although we can use LSTM as our RNN cell, the final state is still incapable of remembering a long sequence.Attention tremendously improves Seq2Seq model.With attention, Seq2Seq model does not forget source input and the decoder knows where to focus. The core idea of attention is to compute the alignment between the state in decoder and all hidden state in encoder using an align function:

$$\alpha_{ij} = \operatorname{align}(\mathrm{h}_i, \mathrm{s}_j) \tag{9}$$

  For each s_j in decoder, we can compute α_{1j},α_{2j},α_{3j} ... α_{mj}(assume we have m hidden state in encoder). Then for state j we can compute Context vector by:

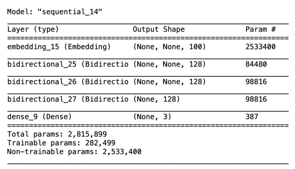
$$c_j = \alpha_{1j} h_1 + \cdots + \alpha_{mj} h_m \tag{10}$$

c_j is the Context vector. For Seq2seq decoder we just use s_j. With attention we will use c_j+ s_j instead of s_j.We can see that c_j is actually the linear combination of all hidden state in encoder. That is why we call it attention. c_j will pay attention to the hidden state with a large corresponding coefficient α_{ij}.

5 Experiments, Results and Discussion

5.1 Experiments for text classification using RNNs

As we have discussed before, we use stacked Bidirectional LSTM with pre-train embedding as our classification model. We use two similar models to do the classification on Text column and Select_text column separately. The model structure and result on test data is following:



```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_15 (Embedding)     (None, None, 100)         2533400
_________________________________________________________________
bidirectional_25 (Bidirectio (None, None, 128)         84480
_________________________________________________________________
bidirectional_26 (Bidirectio (None, None, 128)         98816
_________________________________________________________________
bidirectional_27 (Bidirectio (None, 128)               98816
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 387
=================================================================
Total params: 2,815,899
Trainable params: 282,499
Non-trainable params: 2,533,400
_________________________________________________________________
```

Figure 6: The summary of RNN model

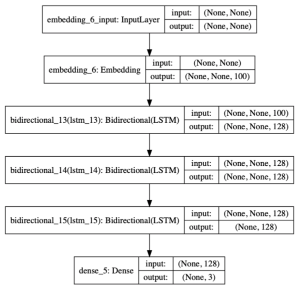

embedding_6_input: InputLayer
input: (None, None)
output: (None, None)
embedding_6: Embedding
input: (None, None)
output: (None, None, 100)
bidirectional_13(lstm_13): Bidirectional(LSTM)
input: (None, None, 100)
output: (None, None, 128)
bidirectional_14(lstm_14): Bidirectional(LSTM)
input: (None, None, 128)
output: (None, None, 128)
bidirectional_15(lstm_15): Bidirectional(LSTM)
input: (None, None, 128)
output: (None, 128)
dense_5: Dense
input: (None, 128)
output: (None, 3)

Figure 7: The structure of RNN model

Table 1: RNNs result on Text data

	Precision	Recall	F1-score
Neutral(0)	0.78	0.82	0.80
Positive(1)	0.82	0.74	0.78
Negative(2)	0.70	0.72	0.71
Accuracy			0.77
Macro avg	0.77	0.76	0.76
Weighted avg	0.77	0.77	0.77

Table 2: RNNs result on Selected_Text data

	Precision	Recall	F1-score
Neutral (0)	0.85	0.85	0.85
Positive(1)	0.86	0.84	0.85
Negative(2)	0.80	0.82	0.81
Accuracy			0.84
Macro avg	0.84	0.84	0.84
Weighted avg	0.84	0.84	0.84

The result shows that with Text data we can achieve an accuracy as 0.77 while with Select_text data we can achieve an accuracy as 0.84. This means with key words extraction, the accuracy of classification for sentiment will be improved significant. We also find that for both cases, the performance for Negative is the worst among the three. This may because the proportion of Negative sample is the smallest.

So far this experiment tells us that the key word extraction will definitely improve the classification accuracy.

5.2 Experiments for text classification using Multi-channel CNN

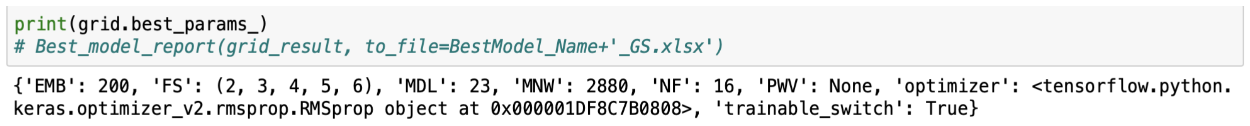
In this experiment, for finding the better hyper parameters in the multi-channel CNNs, we implemented grid search with 3-fold Cross validation. Considering that the CNN is quite easy to over fit in this classification tasks, the early stopping techniques introduce in this part, which is monitor the validation loss on each fold.

With experiment, the part of grid search result conclusion:

- RMSprop(lr=0.001) VS adam(lr=0.0001): We find that RMSprop converge faster than other optimizers, but it will be easily overfitting. Adam with smaller learning rate is a better choice in this experiment.
- Filter size = [2,3,4,5], [2,3,4,5,6]: Each channel has different filter size to capture specific local features (bag of words). Unique word features do not take into account since it will introduce lots of noise in the features map for the classification.
- Number of filters=16,24: This is the number of convolution kernels for each channel. We find that if this number is too large, the model will face a problem of Over-fitting.

- Embedding = [150, 200]: The pre-train embedding is not necessarily fixed in the CNNs. The pretrained Word Vector will help the CNNs converge, but this will not improve final accuracy on the classification.

With further experiment, using grid search with more than 1000 combination of parameters, we get our best parameter:



```
print(grid.best_params_)
# Best_model_report(grid_result, to_file=BestModel_Name+'_GS.xlsx')
```

```
{'EMB': 200, 'FS': (2, 3, 4, 5, 6), 'MDL': 23, 'MNW': 2880, 'NF': 16, 'PWV': None, 'optimizer': <tensorflow.python.
keras.optimizer_v2.rmsprop.RMSprop object at 0x000001DF8C7B0808>, 'trainable_switch': True}
```

Figure 8: The grid search result

We use the parameter from the grid search result to feed out model.Since we have find the different performance using Text data or Select_text data in RNNs part, now we only do the experiment on Select_text data. The following are the structure of multi-channel CNN and model prediction result on test set:

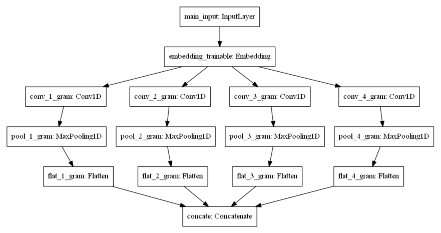


Figure 9: The structure of multi-channel CNN

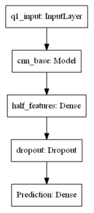


Figure 10: The structure of Dense for classification

Table 3: Multi-channel CNN result on Select_Text data

	Precision	Recall	F1-score	Support
Neutral (0)	0.82	0.85	0.83	1112
Positive(1)	0.90	0.78	0.82	857
Negative(2)	0.82	0.76	0.79	780
Accuracy			0.82	2749
Macro avg	0.84	0.80	0.82	2749
Weighted avg	0.80	0.80	0.80	2749

The result shows that Multi-channel model on Select_Text data can achieve an accuracy of 82%, which is not better than RNNs. Note that the Positive sample has the best performance on CNN model although the proportion of it is not the biggest. This may because CNN is better in feature extraction and the positive feature are more likely to be extracted. For example, the positive words are more representative than others words.

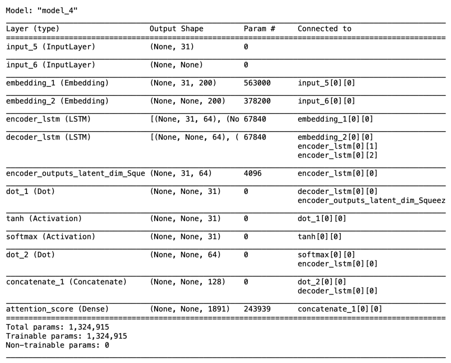
5.3 Experiments for Key words extraction using Seq2seq with attention

In this experiment, we use Seq2seq model to do the Key words extraction. Because We already have the column: ”selected text” in the data set. So, we can use this data set to train a Sequence to sequence model, for summarizing those selected text. The hyper-parameters is a little bit different. We keep the ”dot” and reduce the input length for the decoder because most of summarized text is comparatively short.The embedding and latent dimension for LSTM is the same. And we implement the additive attention as comparison. For getting specific word as context, in the attention output layer, we set the L1 normalization in the activation function. In other words, we want some specific words as output.

After we trained the model, we need a inference model to make the prediction. We build a Inference model with Additive attention, there are three way to calculate the score in attention:

$$\operatorname{score}\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right)= \begin{cases}\boldsymbol{h}_t^{\top} \overline{\boldsymbol{h}}_s & \text{dot} \\ \boldsymbol{h}_t^{\top} \boldsymbol{W}_{\boldsymbol{a}} \overline{\boldsymbol{h}}_s & \text{general} \\ \boldsymbol{v}_a^{\top} \tanh \left(\boldsymbol{W}_{\boldsymbol{a}}\left[\boldsymbol{h}_t ; \overline{\boldsymbol{h}}_s\right]\right) & \text{concat}\end{cases} \tag{11}$$

The following is the structure of our Seq2seq model with attention:



```
Model: "model_4"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            (None, 31)           0
__________________________________________________________________________________________________
input_6 (InputLayer)            (None, None)         0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 31, 200)      563000      input_5[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, None, 200)    378200      input_6[0][0]
__________________________________________________________________________________________________
encoder_lstm (LSTM)             [(None, 31, 64), (No 67840       embedding_1[0][0]
__________________________________________________________________________________________________
decoder_lstm (LSTM)             [(None, None, 64), ( 67840       embedding_2[0][0]
                                                                 encoder_lstm[0][1]
                                                                 encoder_lstm[0][2]
__________________________________________________________________________________________________
encoder_outputs_latent_dim_Sque (None, 31, 64)       4096        encoder_lstm[0][0]
__________________________________________________________________________________________________
dot_1 (Dot)                     (None, None, 31)     0           decoder_lstm[0][0]
                                                                 encoder_outputs_latent_dim_Squeez
__________________________________________________________________________________________________
tanh (Activation)               (None, None, 31)     0           dot_1[0][0]
__________________________________________________________________________________________________
softmax (Activation)            (None, None, 31)     0           tanh[0][0]
__________________________________________________________________________________________________
dot_2 (Dot)                     (None, None, 64)     0           softmax[0][0]
                                                                 encoder_lstm[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, None, 128)    0           dot_2[0][0]
                                                                 decoder_lstm[0][0]
__________________________________________________________________________________________________
attention_score (Dense)         (None, None, 1891)   243939      concatenate_1[0][0]
==================================================================================================
Total params: 1,324,915
Trainable params: 1,324,915
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 11: The summary of Seq2seq model with attention

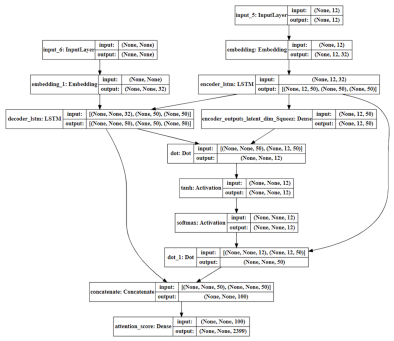

input_5: InputLayer | input: (None, 12) | output: (None, 12)
input_6: InputLayer | input: (None, None) | output: (None, None)
embedding: Embedding | input: (None, 12) | output: (None, 12, 32)
embedding_1: Embedding | input: (None, None) | output: (None, None, 32)
encoder_lstm: LSTM | input: (None, 12, 32) | output: [(None, 12, 50), (None, 50), (None, 50)]
decoder_lstm: LSTM | input: [(None, None, 32), (None, 50), (None, 50)] | output: [(None, None, 50), (None, 50), (None, 50)]
encoder_outputs_latent_dim_Squeez: Dense | input: (None, 12, 50) | output: (None, 12, 50)
dot: Dot | input: [(None, None, 50), (None, 12, 50)] | output: (None, None, 12)
tanh: Activation | input: (None, None, 12) | output: (None, None, 12)
softmax: Activation | input: (None, None, 12) | output: (None, None, 12)
dot_1: Dot | input: [(None, None, 12), (None, 12, 50)] | output: (None, None, 50)
concatenate: Concatenate | input: [(None, None, 50), (None, None, 50)] | output: (None, None, 100)
attention_score: Dense | input: (None, None, 100) | output: (None, None, 2399)

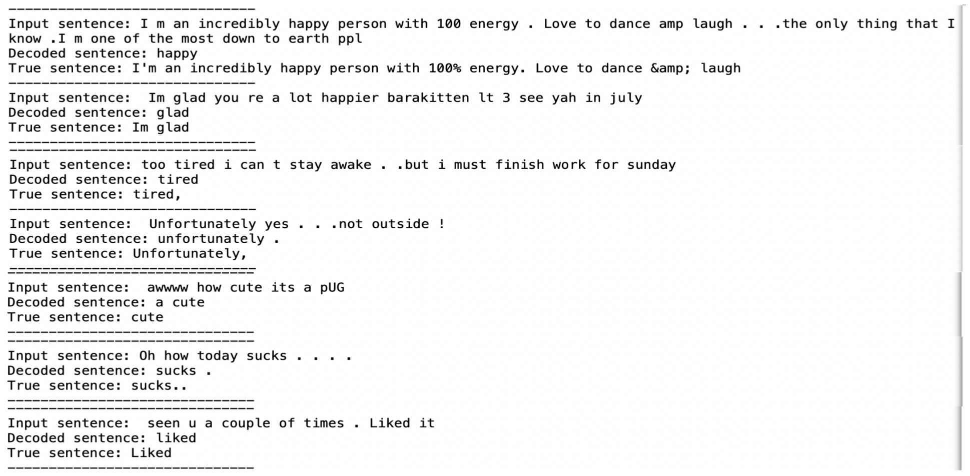
Figure 12: The structure of Seq2seq model with attention

We use the BLEU (bilingual evaluation understudy) to evaluate our model and the result is below:

Table 4: The BLEU result for Seq2seq model

	Seq2seq	Seq2seq with attention
BLEU	0.04534	0.10367

The BLEU result shows that with attention our Seq2seq model will improve significantly. We also want to show some inference result BLEU Score on validation set for this key words extraction model:



```
------------------------------
Input sentence: I m an incredibly happy person with 100 energy . Love to dance amp laugh . . .the only thing that I
know .I m one of the most down to earth ppl
Decoded sentence: happy
True sentence: I'm an incredibly happy person with 100% energy. Love to dance &amp; laugh
------------------------------
Input sentence:  Im glad you re a lot happier barakitten lt 3 see yah in july
Decoded sentence: glad
True sentence: Im glad
------------------------------
------------------------------
Input sentence: too tired i can t stay awake . .but i must finish work for sunday
Decoded sentence: tired
True sentence: tired,
------------------------------
Input sentence:  Unfortunately yes . . .not outside !
Decoded sentence: unfortunately .
True sentence: Unfortunately,
------------------------------
------------------------------
Input sentence:  awwww how cute its a pUG
Decoded sentence: a cute
True sentence: cute
------------------------------
------------------------------
Input sentence: Oh how today sucks . . . .
Decoded sentence: sucks .
True sentence: sucks..
------------------------------
------------------------------
Input sentence:  seen u a couple of times . Liked it
Decoded sentence: liked
True sentence: Liked
------------------------------
```

Figure 13: Some inference results on validation set

Recall that the reason why we want build this Seq2seq model is to do the key words extraction. We want these key words to help us in prediction of sentiment. The previous RNNs experiment result indeed prove that Key words can improve the classification accuracy. Since we have build this inference model, we use it on the Text column on the test set and get a new column(Inference_Text), which is the key word from the Text column. We use this new column as the input of our RNNs classification model and make the prediction again, the classification result is bellow:

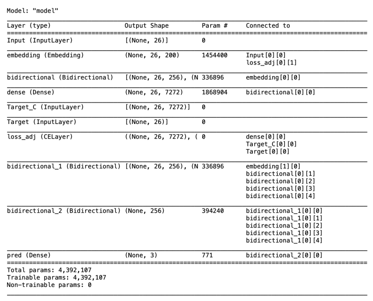
Table 5: RNNs result on Inference_Text data

	Precision	Recall	F1-score	Support
Neutral (0)	0.82	0.81	0.81	1112
Positive(1)	0.81	0.78	0.82	857
Negative(2)	0.79	0.76	0.79	780
Accuracy			0.81	2749
Macro avg	0.81	0.80	0.80	2749
Weighted avg	0.80	0.80	0.80	2749

Recall that in the previous RNNs experiment. The accuracy using Text data is 0.77 and the accuracy using Select_text data is 0.84. Now using our key word extraction model, the accuracy is 0.81, which is better than the previous Text data performance. This means our Seq2seq model really works for extracting key words. Since for the test data we don't really have the Select_text data like the training data, this means this Seq2seq model is a big improvement for sentiment classification.

5.4 Experiments for Combined RNNs for both summarization and classification together

In this experiment, we are considering to build a model to combine the Test data and Select_text data to make the classification more robust. The idea is to connect a summarizing model and classification model together. We will define a new loss function to combine two loss functions together. While we are training, we are doing summarizing and classification together. The following is the summary and structure of the model:



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Input (InputLayer)              [(None, 26)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 26, 200)      1454400     Input[0][0]
                                                                 loss_adj[0][1]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   [(None, 26, 256), (N 336896      embedding[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 26, 7272)     1868904     bidirectional[0][0]
__________________________________________________________________________________________________
Target_C (InputLayer)           [(None, 26, 7272)]   0
__________________________________________________________________________________________________
Target (InputLayer)             [(None, 26)]         0
__________________________________________________________________________________________________
loss_adj (CELayer)              ((None, 26, 7272), ( 0           dense[0][0]
                                                                 Target_C[0][0]
                                                                 Target[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) [(None, 26, 256), (N 336896      embedding[1][0]
                                                                 bidirectional[0][1]
                                                                 bidirectional[0][2]
                                                                 bidirectional[0][3]
                                                                 bidirectional[0][4]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 256)          394240      bidirectional_1[0][0]
                                                                 bidirectional_1[0][1]
                                                                 bidirectional_1[0][2]
                                                                 bidirectional_1[0][3]
                                                                 bidirectional_1[0][4]
__________________________________________________________________________________________________
pred (Dense)                    (None, 3)            771         bidirectional_2[0][0]
==================================================================================================
Total params: 4,392,107
Trainable params: 4,392,107
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 14: The summary of Combined model

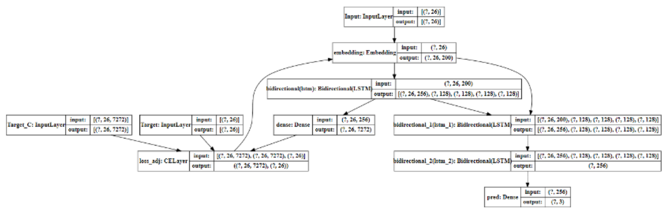

Input: InputLayer
input: [(?, 26)]
output: [(?, 26)]
embedding: Embedding
input: (?, 26)
output: (?, 26, 200)
bidirectional(lstm): Bidirectional(LSTM)
input: (?, 26, 200)
output: [(?, 26, 256), (?, 128), (?, 128), (?, 128), (?, 128)]
Target_C: InputLayer
input: [(?, 26, 7272)]
output: [(?, 26, 7272)]
Target: InputLayer
input: [(?, 26)]
output: [(?, 26)]
dense: Dense
input: (?, 26, 256)
output: (?, 26, 7272)
bidirectional_1(lstm_1): Bidirectional(LSTM)
input: [(?, 26, 200), (?, 128), (?, 128), (?, 128), (?, 128)]
output: [(?, 26, 256), (?, 128), (?, 128), (?, 128), (?, 128)]
loss_adj: CELayer
input: [(?, 26, 7272), (?, 26, 7272), (?, 26)]
output: ((?, 26, 7272), (?, 26))
bidirectional_2(lstm_2): Bidirectional(LSTM)
input: [(?, 26, 256), (?, 128), (?, 128), (?, 128), (?, 128)]
output: (?, 256)
pred: Dense
input: (?, 256)
output: (?, 3)

Figure 15: The structure of Combined model

This model has three input:

- (1) Original text input
- (2) Categorical selected text Input(used for calculating Categorical crossentroypy)
- (3) Selected text input

While training, the Original text input goes through embedding layer and the first LSTM + Dense layer, then gets the summarizing probability distribution. So, this part is the summarizing model. While training, the loss of this part will be counted to the whole loss. After training, we can pick this model up and use it to do the summarizing sentiment key words independently.

The idea of this part is actually inspired by the Supervised Autoencoder model. In Autoencoder, the input is a high dimension data. The Autoencoder will reduce the high dimension vector into low dimension vector in the middle of the neural network. If we connect another model to accept the low dimension vector and have a label to supervise this model, we can make the low dimension vector to be what we want it to be. In this case, the low dimension vector is the summarizing words. The supervised model is the classification model. We want to force the low dimension vector to have a better performance in sentiment classification task.

The Add loss layer is to compare the summarizing words and Categorical selected text Input. Then, add this loss to the whole loss. We want the the first LSTM layer try it best to learning the key words feature. We also want to use the Select_text data to correct the parameter of this model.

At the same time, for the classification part, we have:

- (1) The output of the first LSTM layer will be the initial input state for the second LSTM layer.
- (2) After the Selected text Input goes through the embedding layer, it will also be the input for the second LSTM layer. This part is the combination of Text data and Select_text data,which makes the classifier more robust.
- (3) At last, we connect the previous model the second LSTM layer. This part is to do the classification task, just like the previous RNNs classification model we have defined before.

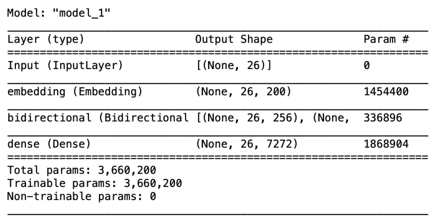
After we have trained the model, we test it on the test set and get the following result:

Table 6: Combined RNN model result on test data

	Precision	Recall	F1-score	Support
Neutral (0)	0.87	0.92	0.89	1112
Positive(1)	0.91	0.80	0.85	857
Negative(2)	0.81	0.84	0.82	780
Accuracy			0.86	2749
Macro avg	0.86	0.85	0.86	2749
Weighted avg	0.87	0.86	0.86	2749

At last, we get the accuracy of 0.86, which is the highest among all previous model we have built.

Now we can pick out the summarizing part of the Combined model and use it to do the summarizing on the validation data. At the same time, we can pick out the classification part.



```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           [(None, 26)]              0
_________________________________________________________________
embedding (Embedding)        (None, 26, 200)           1454400
_________________________________________________________________
bidirectional (Bidirectional [(None, 26, 256), (None,  336896
_________________________________________________________________
dense (Dense)                (None, 26, 7272)          1868904
=================================================================
Total params: 3,660,200
Trainable params: 3,660,200
Non-trainable params: 0
_________________________________________________________________
```

Figure 16: The summary of summarizing part of the Combined RNNs

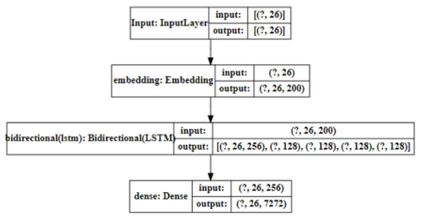
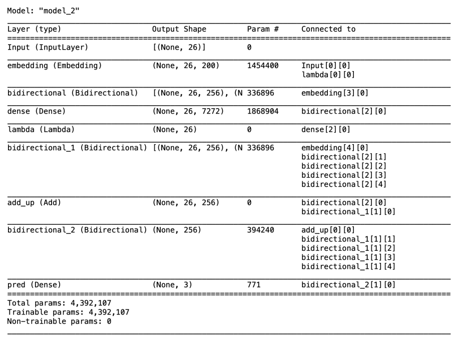


Figure 17: The structure of summarizing part of the Combined RNNs



```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Input (InputLayer)              [(None, 26)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 26, 200)      1454400     Input[0][0]
                                                                 lambda[0][0]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   [(None, 26, 256), (N 336896      embedding[3][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 26, 7272)     1868904     bidirectional[2][0]
__________________________________________________________________________________________________
lambda (Lambda)                 (None, 26)           0           dense[2][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) [(None, 26, 256), (N 336896      embedding[4][0]
                                                                 bidirectional[2][1]
                                                                 bidirectional[2][2]
                                                                 bidirectional[2][3]
                                                                 bidirectional[2][4]
__________________________________________________________________________________________________
add_up (Add)                    (None, 26, 256)      0           bidirectional[2][0]
                                                                 bidirectional_1[1][0]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 256)          394240      add_up[0][0]
                                                                 bidirectional_1[1][1]
                                                                 bidirectional_1[1][2]
                                                                 bidirectional_1[1][3]
                                                                 bidirectional_1[1][4]
__________________________________________________________________________________________________
pred (Dense)                    (None, 3)            771         bidirectional_2[1][0]
==================================================================================================
Total params: 4,392,107
Trainable params: 4,392,107
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 18: The summary of classification part of the Combined RNNs

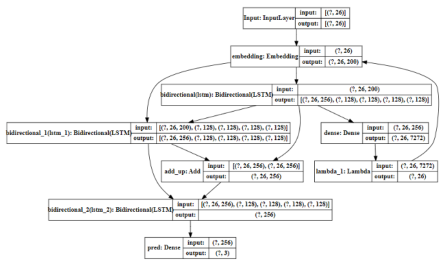

Input: InputLayer
input: [(?, 26)]
output: [(?, 26)]
embedding: Embedding
input: (?, 26)
output: (?, 26, 200)
bidirectional(lstm): Bidirectional(LSTM)
input: (?, 26, 200)
output: [(?, 26, 256), (?, 128), (?, 128), (?, 128), (?, 128)]
bidirectional_1(lstm_1): Bidirectional(LSTM)
input: [(?, 26, 200), (?, 128), (?, 128), (?, 128), (?, 128)]
output: [(?, 26, 256), (?, 128), (?, 128), (?, 128), (?, 128)]
dense: Dense
input: (?, 26, 256)
output: (?, 26, 7272)
add_up: Add
input: [(?, 26, 256), (?, 26, 256)]
output: (?, 26, 256)
lambda_1: Lambda
input: (?, 26, 7272)
output: (?, 26)
bidirectional_2(lstm_2): Bidirectional(LSTM)
input: [(?, 26, 256), (?, 128), (?, 128), (?, 128), (?, 128)]
output: (?, 256)
pred: Dense
input: (?, 256)
output: (?, 3)

Figure 19: The structure of classification part of the Combined RNNs

The BLEU score for this summarizing model is 0.446890, which is much higher than the previous Seq2seq model. This may because the summarizing text is basically the subset of original text. It will definitely include the Select_text part.

6 Conclusion and Future Work

The aim of this project is the text classification for the sentiment. We include the key words extraction is to improve the accuracy of the classification. We have applied four models: RNNs, Multi-channel CNN, Seq2seq+attention+RNN, Combined RNNs. The result is following:

Table 7: Classification result for different models

	Precision	Recall	F1-score	Accuracy
RNNs(LSTM)	0.84	0.84	0.84	0.84
Multi-channel CNN	0.84	0.80	0.82	0.82
Seq2seq+attention+RNN	0.81	0.80	0.80	0.81
Combined RNNs	0.87	0.86	0.86	0.86

We can conclude that the Combined RNNs is the best model in this classification task because it combine the key words extraction and classification together.

In the future, we would improve our models in several ways.Firstly, recall our Combined RNNs model, we define a new loss function for the whole model, including summarizing loss and classification loss. We should make a trade off, that is, a reasonable proportion for different part of loss. Besides, we should increase the model complexity of the summarizing part to increase the accuracy. For instance, add attention to this LSTM structure. Moreover, self-attention may also been considered to replace the RNNs structure. We may replace all RNNs structure in Combined RNNs model with multi-head self-attention layers.

References

[1] Bojanowski, Piotr, et al. ”Enriching word vectors with subword information.” Transactions of the Association for Computational Linguistics 5 (2017): 135-146.

[2] Dos Santos, Cicero, and Maira Gatti. ”Deep convolutional neural networks for sentiment analysis of short texts.” Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers. 2014.

[3] YoonKim(2014):Convolutional Neural Networks for Sentence Classification. EMNLP 2014

[4] Zhang and Wallace (2015) A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification

[5] ”Densely Connected Convolutional Networks”, Huang et al, 2017.