EmotionGIF 2020 using deep learning model

Anonymous ACL submission

Abstract

In EmotionGIF 2020 competition, given the dataset contain tweets and their GIF responses, labeled with the categories of the animated GIFs, the challenge is to recommend GIF categories for the unlabeled dataset. In this paper, we regard the task of recommending GIF categories as a text classification problem. We propose a LSTM-based model for dealing with this problem without any pre-trained data. By only dealing in the text field with the LSTM model, we can get a certain extent of accuracy.

1 Introduction

Nowadays, more and more people choose to use animated GIFs to assist even replace the text to express their feeling on social media. GIFs offer a more nuanced and precise way to convey emotions. Emotion detection has received a great amount of attention in recent years. The EmotionGIF 2020 competition is also related to the field of Emotion detection. The given training data contains tweets and their GIF responses, each data is labeled with the category(ies) of the animated GIFs, and the challenge is to recommend 6 GIF categories from 43 classes for all the tweets in the unlabeled evaluation dataset. In other words, our task will be classifying the text of the tweets and the recommendation of us will be the 6 most possible classes.

To deal with this task, we used the LSTM model for training, which has good performance in processing the sequential input data. When a person understands a sentence, it is usually not understood word by word, but from the previous knowledge and context to understand the meaning of the text. RNN introduced this idea to let the machine simulate human reading. The LSTM improves the problem of gradient vanishing problem during general RNN training, and also allows the model to remember longer information.

In the part of data pre-processing, we didn't do too much effort on it. Since we think that every word in the tweet may affect the emotion that the writer wants to convey. For example, the word 'yeeees' is not the official vocabulary in English, it represents the exact same meaning as 'yes'. However, 'yeeees' convey a stronger feeling to us. Most of the words used in the tweet have influences on the emotion detection problem. Moreover, the tokenizer we used is the 'TweetTokenizer' in 'nltk' library, it can nicely tokenize the words in the tweet including common abbreviation, tags and the emojis.

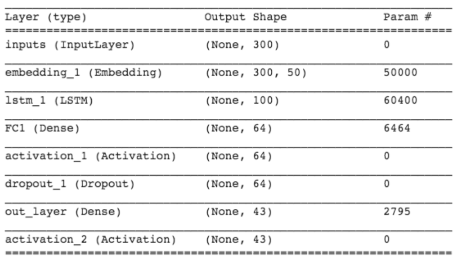
The mean average recall at 6 we got in the evaluation dataset is 0.4863.

2 Method

2.1 Data pre-processing

Tokenization and Build vocabulary. First, we lowercase all the texts and replies from the test data before tokenizing. In this paper, we choose an off-the-shelf tokenizer, NLTK.TweetTokenizer, which is capable of recognizing special elements in Tweets, e.g. hashtags and emojis. After Tokenizing, we build a vocabulary list according to the all the tokens generated from the test data. Words which appear less than 500 times in the test data, was eliminated from the vocabulary list.

Sequence matrix. Since we do the word embedding in the first part of our model, we have to convert our tweet data into a sequence matrix, which we will explain it later, as an input to the embedding layer in our model. To construct the sequence matrix from the test data, we replace each token in the test data into a number, which is the index of the word(token) in the vocabulary list we build before. We ignore the words(tokens) which is not in the vocabulary list. For instance, assume the vocabulary list contain three words: ['hello', 'you', 'how'], and the tweet is ['hello','!','how','are','you'], then the result integer sequence is [0,2,1]. Also, for every single tweet, we concatenate its text and reply into a single sequence. Since each tweet may have different length, we used 'pad_sequence' tool in 'Keras', which put some 0s in the back of the integer sequence to ensure all sequences have the same length. Now, we have constructed a sequence matrix as an input to the embedding layer.



```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
inputs (InputLayer)          (None, 300)               0
________________________________________________________________
embedding_1 (Embedding)      (None, 300, 50)           50000
________________________________________________________________
lstm_1 (LSTM)                (None, 100)               60400
________________________________________________________________
FC1 (Dense)                  (None, 64)                6464
________________________________________________________________
activation_1 (Activation)    (None, 64)                0
________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
________________________________________________________________
out_layer (Dense)            (None, 43)                2795
________________________________________________________________
activation_2 (Activation)    (None, 43)                0
================================================================
```

Figure 1: The architecture of model

2.2 The RNN Architecture

Figure 1. shows the architecture of our model. The first layer is the embedding layer, which learns to transform words into dense vectors where a vector represents the projection of the word into a continuous vector space. The input of the embedding layer is the sequence matrix we constructed in Sec. 2.1. Then, it will be feed into the LSTM layer, and then a fully connected layer. The activation function is softmax function, because we regard this task as a muti-class classification problem. The output of the networks is a vector with the length of 43, each element represents the probability of each GIF category.

3 Experiment

In the experiment, the loss function we set was 'categorical_crossentropy'. Batch size equals to 128. The output size of the first fully connected layer is 64. We do the experiment in different number of output size of FC layer, see Figure 2. As we can see in Figure 2., the model with its FC layer size of 64 reaches higher accuracy in our experiment. Thus, we choose 64 as our final FC layer size. The model will train until the validation accuracy converge, the maximum epoch number is 100. Figure 3. shows the model accuracy. As we can see in Figure 3., the validation accuray growth slows down after about 8 epoch.

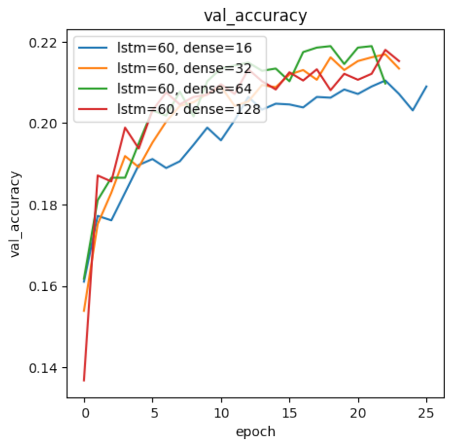


Figure 2: The accuracy of different size of first fully connected layer output

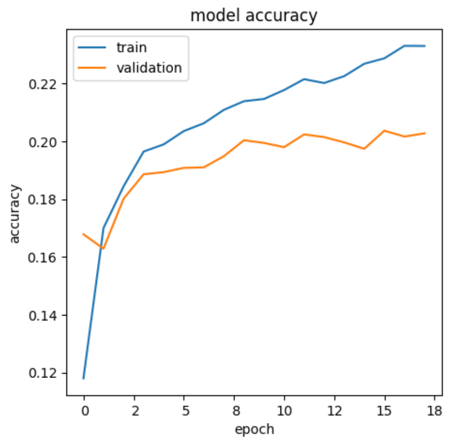


Figure 3: The accuracy of each epoch

4 Conclusion

During the whole training processing, we only used the training data given from the EmotionNLP 2020, there is no any pre-trained data. Using LSTM model to do the tweets classification, based on simple data-pre-processing, the mean recall at 6 we can get is around 0.48.

Future works. Although we thought that every word is important for the emotion detection task, there still may have some words that are redundant, the way to find out the redundant words may need to check through the dataset by ourselves, however, it's probably worth to make some effort on it. In the part of the deep learning model, there are lots of new and nice models coming out in these years, e.g. BERT. We might try other model or combine the advantages of them, to get the better performance.