Introduction

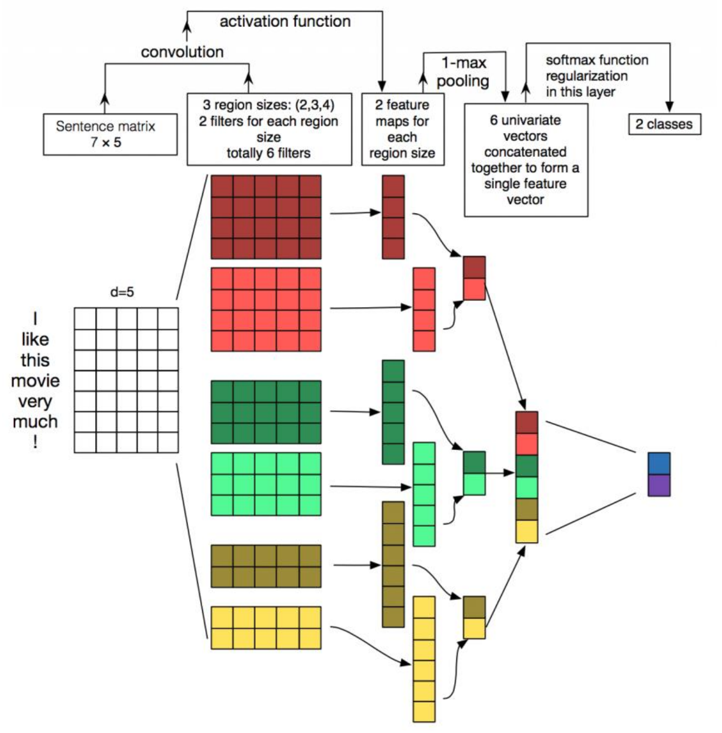
Sentiment analysis has long been a problem for business, marketing and management areas for more value earned in the decision process. Previous work about sentiment analysis has been focusing on document-level, sentence-level and word-level sentiment extraction, with both supervised and unsupervised approaches. In this project, we are introducing a classification model for sentiment analysis with context information participated in the feature space, and we'll compare two popular models based on their ability to learn from labeled data and perform classification tasks.

Description of individual work

We will build a Convolutional Neural Network (CNN) classifier and Long Short-Term Memory (LSTM) to predict whether a sentiment is positive or negative, and I will discuss the CNN.

Though the convolutional neural network is typically used for tasks such as image classification, its principle may also be applied for text classification purposes. A CNN can be used to convert text into stacked vector representations which may then be treated as an 'image'. The CNN may then use filters that are of the same length as word embeddings which allows it to make shapes corresponding to ngrams. Ngrams allow the model to understand spatial representations of text which is important for any text classification task. The preprocessing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, we can use cnn to learn these filters/characteristics.

The picture is an example of CNN architecture for sentence classification.


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Here we depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. So a univariate feature vector is generated from all six maps, and the final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence show two possible output states.

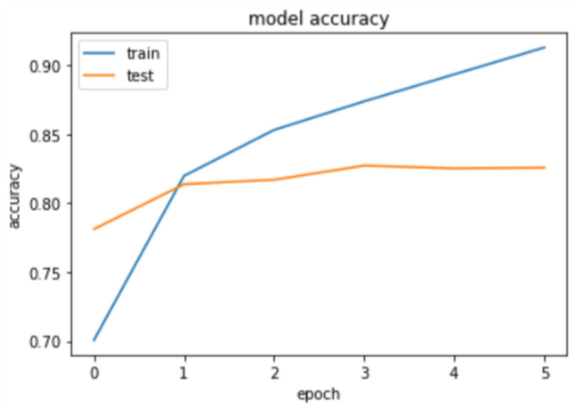
Then I would describe the portion of the work that I did on the project in detail. I did data preprocessing, first we would do text preprocessing steps using the way of natural processing language. For example, take a text string as a parameter, perform preprocessing on the string to remove special character from the string like html tags, punctuations, numbers, single character, and multiple spaces, then we preprocess the reviews and store them in a list, after this we divide our dataset into 80% for training set and 20% for testing.

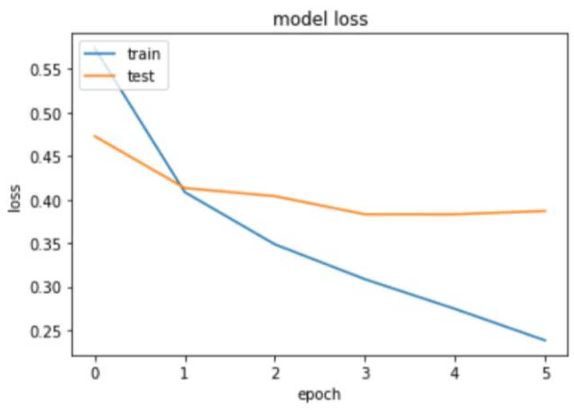
In the step of preparing the embedding layer, we found the vocabulary size and perform padding on training and testing set, then the clove word embeddings need to be loaded, we could create a dictionary to contain words as keys and their corresponding embedding list as values. Finally, we could create a matrix where each number will correspond to the index of the work in the corpus.

In text classification, though in some cases our model fit the training data after 6 epochs, there were instances where we were required to change the epoch size to 15 to avoid overfitting. Because we wanted to find the optimal relation between batch size and learning rate, so we did four cases arose from the different model fits that we tested. In order to keep the test set separate so as to not overfit the data and cause a generalizing gap, we also used 33% of the training data as validation data and evaluated our model’s performance using the test set.

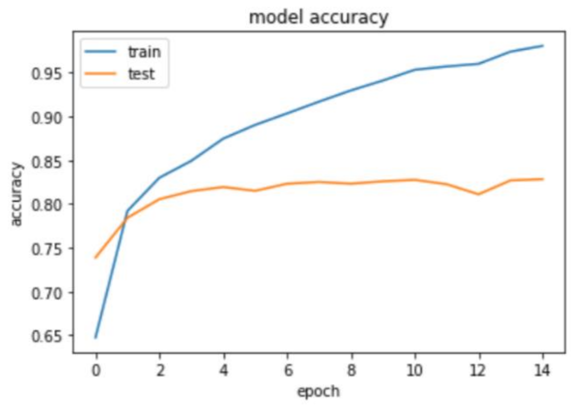
Results

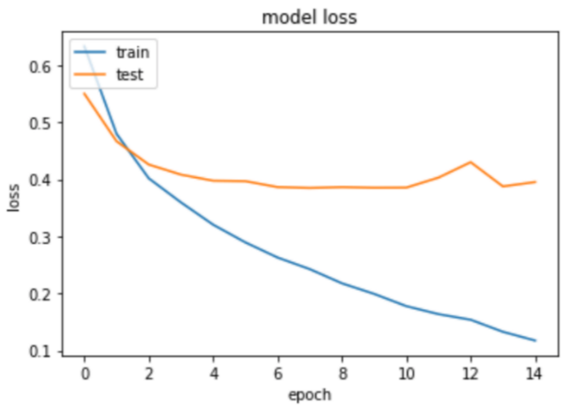
Firstly, we are choosing epochs as 6, batch size as 128, and the test accuracy gives us around 0.84, and model accuracy and model loss look like the following plots.



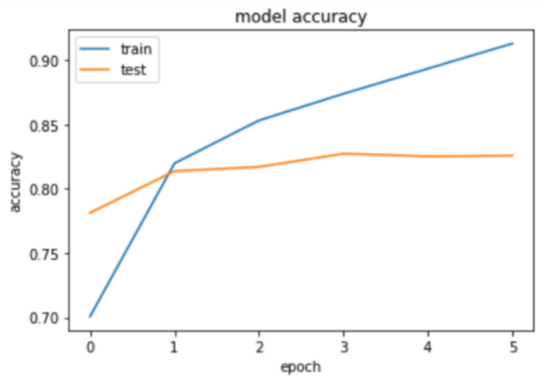


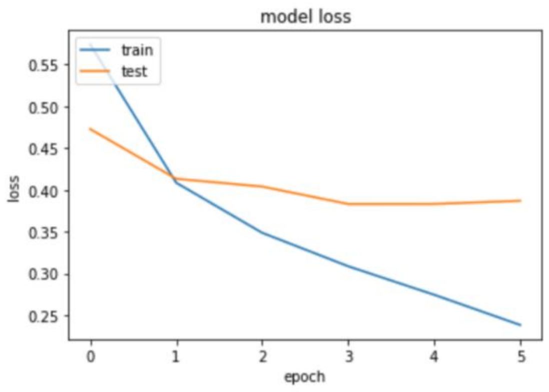
Next we changed the batch size as 32, and we get the test accuracy that is almost 0.84, we can see the plots:



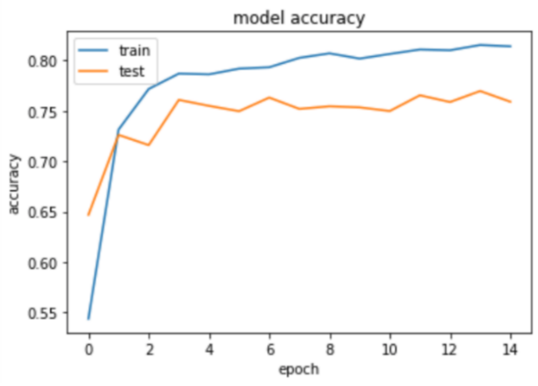


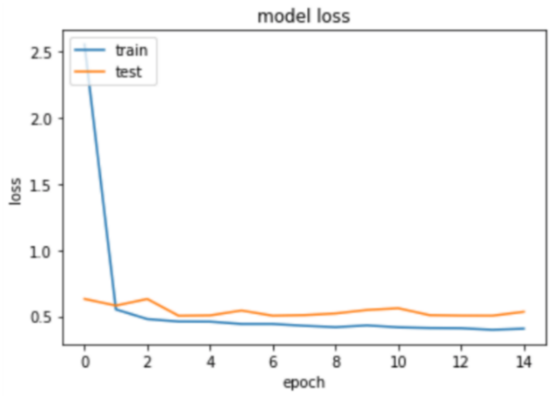
Then we are considering configuring the learning rate and batch size. As mentioned above, we read that lower batch sizes are able to capture gradients better and the test accuracy lost by higher batch sizes can be made up by increasing the learning rate. As we trained the model as batch size is 128, so we set 32 and 256 as the low batch size and high batch size separately, then with 0.01 and 0.1 of learning rate relevantly. So we get the related accuracy scores for each of them. One is about 0.78 with lower parameters, and the plots are shown below.





Another result we get with high parameters is around 0.76, and we can see the plots.





Summary and conclusions

Overall, a CNN is able to use filters that act as word embeddings, which allows ngram analysis. By comparing two scores and viewing the trend of two lines in plots, we can know that the accuracy is higher when learning rate and batch size are lower. Therefore, we could set those settings as our training parameters in this way.

The percentage of the code that you found will be $\frac{70-18}{70+14} \times 100 = 61.9\%$.

Reference

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/