Multi-class Sentiment Analysis Using a One Dimensional Convolution Neural Network

1st Shouxi Li
dept. Electrical and Computer Engineering
Lakhead University
Thunder Bay, Canada
0883145

***Abstract*—This report is conducted after the experiment of building a one-dimensional convolutional neural network performing a multi-class sentiment analysis. The data used in the experiment is the 'rotten tomatoes movie reviews', which contains 5 classes in total. There are several models tested in the experiment, which includes a base model and other two models with pre-trained embeddings in two configurations. The experiment is conducted in the Google Colab using the Keras library and the models are build using the sequential method. The performance metrics in evaluation are 'accuracy', 'precision', 'recall', and the 'f1 score'. Figures and other graphical representations will be used to give a more straightforward impression.**

***Index Terms*—natural language processing, sentiment analysis, convolutional neural network, GLoVe embedding, Keras**

I. Introduction

The growing and developing industry of graphics hardware promises the industry of machine learning and deep learning [1]. Now the powerful chip inside the graphic processing unit (GPU) have significantly boost the speed of training and testing of the artificial neural networks, which makes it possible, in this experiment, to choose GPUs as acceleration hardware to make save the training time. The network built in this experiment is a one-dimensional convolutional neural network that includes several layers, which will be explained in detail in the coming section.

The data set used in the experiment is the 'rotten tomatoes movie reviews' that can be obtained from GitHub, and raw data set is a file with the 'tsv' extension, which makes it really easy to be imported into Colab. The data set originally contains four columns, which are the 'phraseID', 'sentenceID', 'phrase', and 'sentiment'. The 'sentenceID' is the index of the sentences that are included in the data set, and since they split the whole sentence into several phrases, the 'phraseID' is the index of the phrase that is split from the original sentence. The 'phrase' is the content of the split sentences and the 'sentiment' is the corresponding sentiment. The sentiment in the data set is in numerical representation starting from 0 to 4, which means that the neural network will be doing a multi-class single-label classification.

As mentioned above, there are several models tested in the experiment, which includes some models using a pre-trained embedding layer. Specifically, the embedding method used here is the Global Vectors for Word Representation (GLoVe). The GLoVe embedding is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space [2]. The use of a pre-trained embedding layer here aims at looking for the performance gap between the base line model and the model that has a pre-trained embedding.

All the models tested in the experiment will be stored in 'h5' extension files and the tested GLoVe data will be included for reproduction. The number of epochs is fixed to be 20 in order to see the performance gap and techniques such as early stopping, randomized search, etc. will not be applied in this experiment in order to control the embedding method to be the only variable.

II. Literature Review

The global vectors for word representations, as known as GLoVe, is a word vector technique that rode the wave of word vectors after a brief silence. The advantage of GloVe is that, it does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors [2]. The idea of this GLoVe embedding is to, firstly, capture the global corpus statistics. Let the matrix of word-word co-occurrence counts be denoted by *X*, whose entries $X_{i,j}$ tabulate the number of word *i*. Let $X_i = \sum_k X_{i,k}$ be the number of times any word appears in the context of word *i*. Finally, let $P_{i,j} = P(j|i) = X_{i,j}/X_i$ be the probability that word *j* appear in the context of word *i*.

According to the authors, the GLoVe model performs significantly better than the other baselines, often with smaller vector seizes and smaller corpora. However, what is interesting is that their results show that the 'word2vec' tool is somewhat better than most of those previously published results, which could be resulted from the factors like the choice to use negative sampling (which typically works better than the hierarchical softmax), the number of negative samples, and the choice of the corpus.

III. Methodology

A. Data Preprocess

The experiment begins with importing the 'tsv' file as the raw data set, and there is a visualization (Fig.1) after the importing. Following is to read the 'Phrase' as the input of

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2

Fig. 1. Visualize the first three rows in the data set.

the model and the 'Sentiment' as the label of the input. As required in the manual, the data will be divided into two parts, with 70% goes into training and the remaining 30% goes into testing. At this stage, the 'train_test_split' function will be called and the argument 'random_state' will be set to 2003 in order to guarantee that every time the data will be split identically for reproduction.

Since the data set comes in English form, it is necessary to clean the text such that punctuation like comma, question mark, exclamation mark, etc. would be removed because these symbols usually do not contain a specific semantic or syntactic meaning. In order to do this, a helper function named 'tokenize_review' is customized, this function will first collect all the characters in the corpus and drop those punctuation, then those collected characters will be lemmatized to drop the prefix or suffix such as 'ing', 'ed', etc. since those elements do not contribute a lot to the sentiment of a sentence. However, before applying this helper function to the raw data set, since the 'Phrase' is split in a random order, it is necessary to index them in numerical order, otherwise it will produce an error in the latter part.

As mentioned above, the sentiment distribution ranks from 0 to 4 in the raw data set, then it is possible that there exists an unbalanced distribution. By visualizing such distribution (Fig.2), it is clear that the sentiment appears in a 'bell' shape. Under this circumstance, to balance the distribution of 'sentiment' may help to stabilize the performance of the neural network. To accomplish it, the up-sampling technique will be conducted here, which aims at sampling every 'sentiment' up to have 75000 samples, apart from that, the original sentiment from 0 to 4 will be shifted to 1 to 5 to make more sense to the readers. What should be acknowledged here is that the execution of up-sampling will change the number of samples in a data set and thus should only be carried out on the training data set while the testing data set remains untouched. The reason is that, in reality, testing data is unknown before the model receives them. For example, it is possible that most of the reviews regarding to a specific objective are negative, however, until the model receives those data, there is no clue showing how will the sentiment of the reviews appear. After cleaning and up-sampling the training data, since all the models here in the experiment are built using the Keras library, it would be pretty convenient to use the built-in preprocess functions to handle the data. Prior to pass the data to the function, it is necessary to calculate the length of the sentences in the data set and the count the number of unique

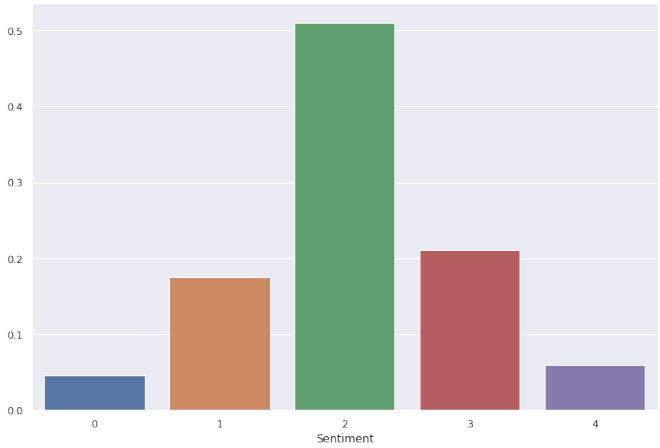

Fig. 2. Visualize the distribution of the sentiment.

words, because Keras requires such arguments in turning the original lexical representations into numerical representations. For example, in this experiment, the total number of unique words is 13723, and the longest sentence in the training set contains 48 words, which means that, after the transforming process, the original 'Phrase' will turn into vectors that all have 48 elements ranging within 13723. The reason why to use 48 as the length of those vector is that all the sentences will be shorter than that 48 words one, however, it is possible that, in the future, the received testing sentence contains more than 48 words, in that case, that whole sentence will be cut to fit the 48 words limit, which makes it possible to lose semantic or syntactic meaning, however, by choosing a large element limit, it will require more memory in the GPU, which could cut the efficiency of the system. After passing the original representations into the Keras built in functions, the data now could be fed into the network. However, since the manual requires evaluation metrics in 'accuracy', 'recall', 'precision', 'f1 score' and by default Keras only provide the 'accuracy' metric, then it is necessary to define those metrics.

The 'recall', 'precision', 'f1 score' depend on the relation between the ground truth and the prediction, which is shown below (TABLE. 1). The calculation of those metrics is as follow:

$$Recall = \frac{TP}{TP + FP} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

TABLE I
COMBINATION OF GROUND TRUTH AND PREDICTION

Total Population		Ground Truth	
		Positive Truth	***Negative Truth***
Predicted Condition	***Predicted Positive***	True Positive (TP)	False Positive (FP)
	Predicted Negative	False Negative (FN)	True Negative (TN)

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3}$$

Once the evaluation metrics are defined, it is ready for the training.

B. Base Model

The base model built here is a quit simple one-dimensional convolutional neural network, with only eight layers, of which the first layer being the embedding layer, whose target is to transform the raw input to another vector in a different dimension. The architecture (Fig. 3) is shown as below. The

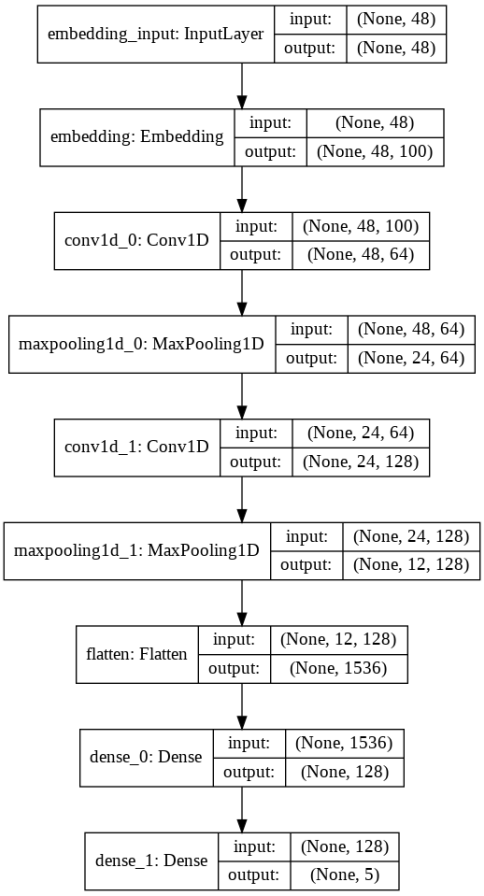

Fig. 3. Architecture of the base model.

embedding dimension here is set to be 100, the reason is that in the next part, the model using the pre-trained GLoVe embedding is a 100-dimensional embedding model. The number of training epoch here is set to be 50, with batch size equals to 128, since the model will not be trained over a long time, a callback function 'checkpoint' here is used, which can save the architecture and the parameters of the model after each epoch once the monitored validation 'f1 score' increases. However, during the training time, from the feedback of the 'checkpoint' function, the best performance occurs at the seventh epoch with a 0.62 validation 'f1 score', 0.63 validation 'precision', 0.62 validation 'recall', 0.62 validation 'accuracy', which means that in the coming training period, the model will start overfitting. The variation of those performance metrics is as shown below, which also indicates that the model starts overfitting within even 50 epochs.

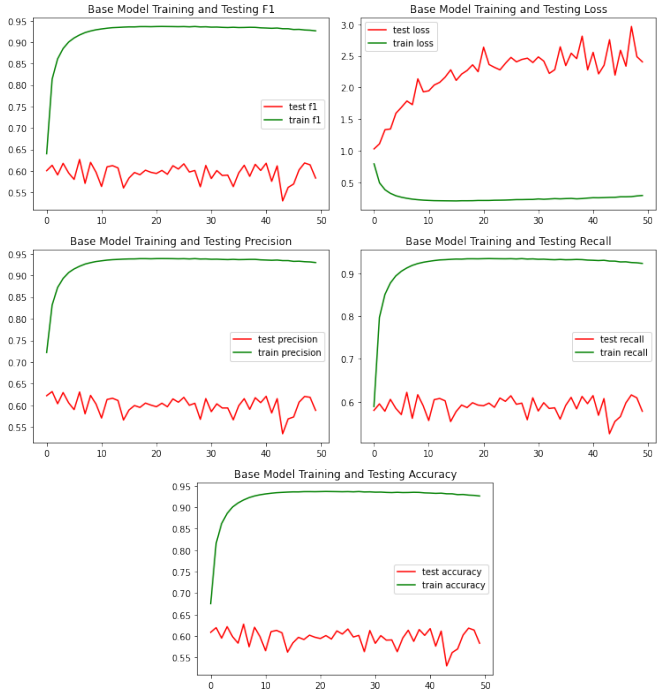

Fig. 4. Performance of the base model.

C. GLoVe Embedding

As mentioned earlier, this experiment also conducted using the GLoVe embedding, moreover, the model is trained with a non-trainable embedding layer and a trainable embedding layer respectively. The training parameter settings are identical as the base model, so that the trainability of the embedding layer is the only variable in the experiment. After 50 epochs of training, the variation of those performance metrics is as follows (Fig. 5 and Fig. 6). What is interesting is that, no matter whether this pre-trained embedding layer is set to be trainable or non-trainable, the performance of this model is not

as good as the base model, the reason why this phenomena occur is that the GLoVe embedding is trained under a specific number of tokens, namely 60 billion, which may not cover the characters occur in this experiment. However, the trainable model performs better than the non-trainable model with 0.61 validation 'f1 score' rather than 0.51 validation 'f1 score'.

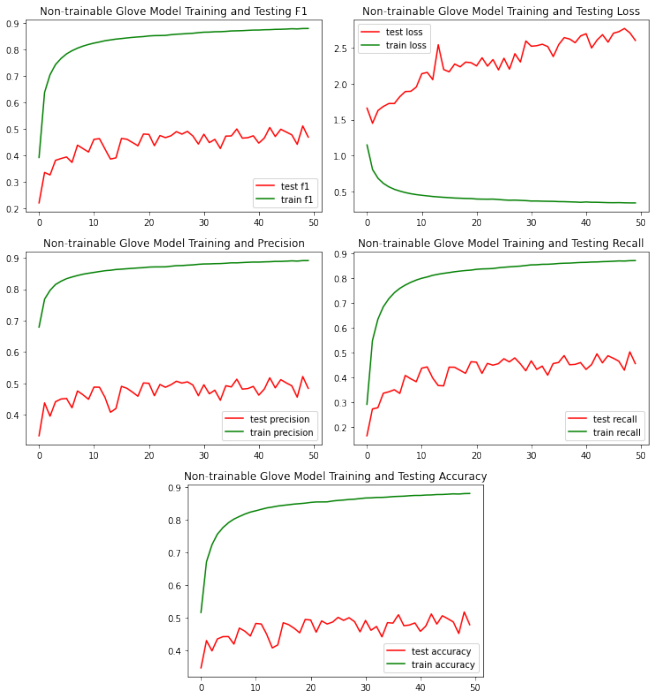

Fig. 5. Performance of the non-trainable GloVe model.

D. Deep Model

The last model conducted in the experiment here is a deeper model, with no pre-trained embedding layer. Similar to the base model, the only difference here is the depth of the model, with more convolution layers than the original base model. Considering the space limitation in the report, the architecture will not be shown here. The training parameter settings are as the same as those in the base model, however, with the increase depth of this model and the training time per epoch from around 16 seconds to around 31 seconds, the performance (Fig. 7) does not increase as much, which remains at 0.62 validation 'f1 score' if only considering the first two decimals.

IV. Model Comparison

As discussed above, the best model in this experiment is, surprisingly, the base model. To make a straightforward impression, this image (Fig. 8) compares the change of validation 'f1 score' among those four models.

In total, there are four models conducted in the experiment, namely, the base model, the model with a non-trainable GLoVe embedding layer, the model with a trainable GLoVe embedding layer, and a model with more depth compared to the original base one. The training and testing result show that among the four models, the base model is the best, however,

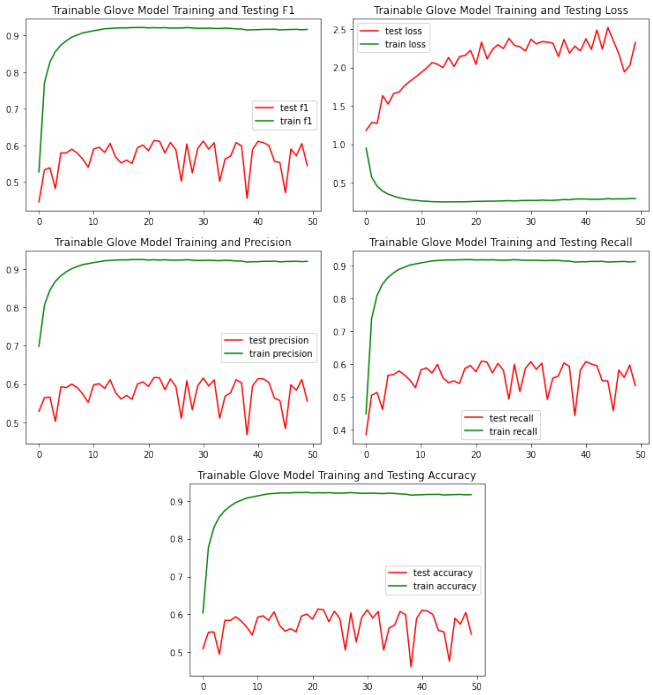

Fig. 6. Performance of the trainable GLoVe model.

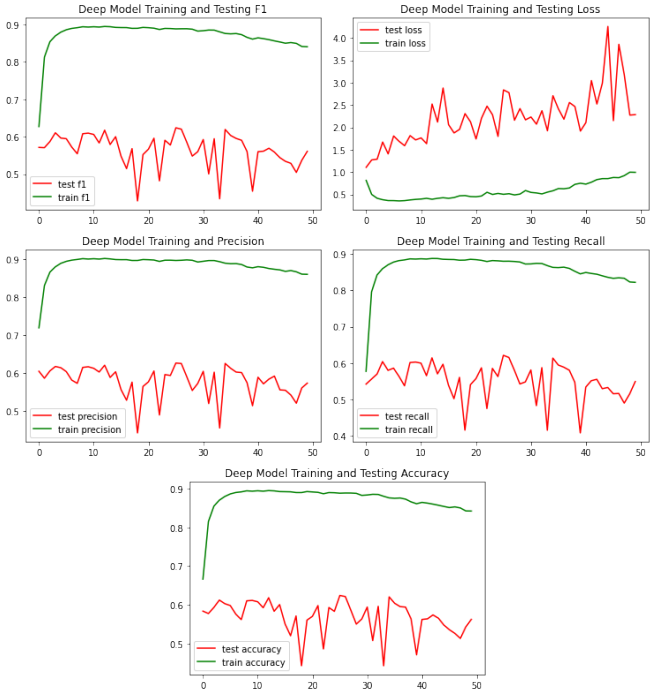

Fig. 7. Performance of the deep model.

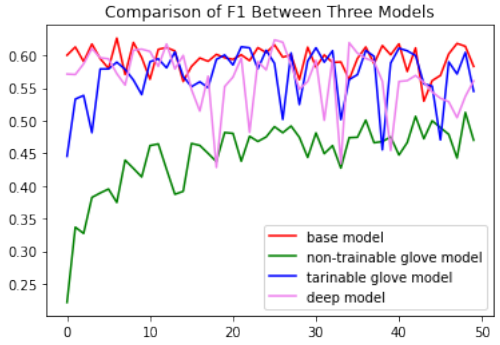

Fig. 8. Comparison of validation f1 score among four models.

it is not robust enough since it reaches the peak performance at the seventh epoch and then starts over fitting, moreover, the figure (Fig.4) shows that the loss keeps increasing and other performance metrics oscillate during the running time validation.

As for the GLoVe embedding, since the source here is the 100 dimensional embedding pre-trained on the Wikipedia2014 and Gigaword5 (newswire text data) data set [2], which may not cover the tokens that appear in this 'rotten tomatoes movie reviews' data set, could result in the model performs not as expected. However, the non-trainable GLoVe embedding model does show less oscillation and a steady increase in all performance metrics than the trainable one.

The deep model built here is the one that oscillates the most compared with other models, which means that the depth of a model is not the key to perform well on this data set. Instead, the key is to use a different model rather than a simple one-dimensional convolutional neural network or to deal with the raw text before feeding them into the network.

REFERENCES

[1] Razzak, M. I., Naz, S., & Zaib, A. (2018). Deep learning for medical image processing: Overview, challenges and the future. In Classification in BioApps (pp. 323-350). Springer, Cham.

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